GraphQL for Sui RPC (Alpha)

This content describes an alpha/beta feature or service. These early stage features and services are in active development, so details are likely to change.

This feature or service is currently available in

This section explains some of the common concepts when working with GraphQL, such as altering behavior using HTTP headers, re-using query snippets with variables and fragments, consuming paginated queries, and understanding and working within the limits enforced by the service.

Jump to the following sections for more details:

For more details on GraphQL fundamentals, see the introductory documentation from GraphQL and GitHub .

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

Based on valuable feedback from the community, the GraphQL RPC release stage has been updated from beta to alpha. Refer to the high-level timeline for beta and GA releases in the previously linked document.

The service accepts the following optional headers:

By default, each request returns the service's version in the response header: x-sui-rpc-version .

The response for the previous request looks similar to the following:

Variables offer a way to introduce dynamic inputs to a re-usable/static query. Declare variables in the parameters to a query or mutation , using the $ symbol and its type (in this example Int), which must be a scalar , enum , or input type. In the query body, refer to it by its name (prefixed with the $ symbol).

If you declare a variable but don't use it or define it (supply a value) in the query, the query fails to execute.

To learn more, read the GraphQL documentation on Variables .

In the following example, a variable supplies the ID of the epoch being queried.

Variables :

When using the online IDE, supply variables as a JSON object to the query in the Variables pane at the bottom of the main editing window. You receive a warning if you supply a variable but don't declare it.

When making a request to the GraphQL service (for example, using curl), pass the query and variables as two fields of a single JSON object:

Fragments are reusable units that you can include in queries as needed. To learn more, consult the official GraphQL documentation . The following example uses fragments to factor out a reusable snippet representing a Move value.

GraphQL supports queries that fetch multiple kinds of data, potentially nested. For example, the following query retrieves the first 20 transaction blocks (along with the digest, the sender's address, the gas object used to pay for the transaction, the gas price, and the gas budget) after a specific transaction block at epoch 97 .

If there are too many transactions to return in a single response, the service applies a limit on the maximum page size for variable size responses (like the transactionBlock query) and you must fetch further results through pagination .

Fields that return a paginated response accept at least the following optional parameters:

They also return a type that conforms to the GraphQL Cursor Connections Specification , meaning its name ends in Connection , and it contains at least the following fields:

Cursors are opaque identifiers for paginated results. The only valid source for a cursor parameter (like after and before) is a cursor field from a previous paginated response (like PageInfo.startCursor , PageInfo.endCursor , or Edge.cursor). The underlying format of the cursor is an implementation detail, and is not guaranteed to remain fixed across versions of the GraphQL service, so do not rely on it -- generating cursors out of thin air is not expected or supported.

Cursors are used to bound results from below (with after) and above (with before). In both cases, the bound is exclusive, meaning it does not include the result that the cursor points to in the bounded region.

Cursors also guarantee consistent pagination. If the first paginated query reads the state of the network at checkpoint X , then a future call to fetch the next page of results using the cursors returned by the first query continues to read from the network at checkpoint X , even if data for future checkpoints is now available.

This property requires that cursors that are used together (for example when supplying an after and before bound) are fixed on the same checkpoint, otherwise the query produces an error.

The GraphQL service does not support consistent pagination for arbitrarily old cursors. A cursor can grow stale, if the checkpoint it is from is no longer in the available range . You can query the upper- and lower-bounds of that range as follows:

The results are the first and last checkpoint for which pagination continues to work and produce a consistent result. At the time of writing the available range offers a 5- to 15-minute buffer period to finish pagination.

After results are bounded using cursors, a page size limit is applied using the first and last parameters. The service requires these parameters to be less than or equal to the max page size limit , and if you provide neither, it selects a default. In addition to setting a limit, first and last control where excess elements are discarded from. For example, if there are 10 potential results -- R0 , R1 , ..., R9 -- after cursor bounds have been applied, then

It is an error to apply both a first and a last limit.

To see these principles put into practice, consult the examples for paginating forwards and paginating backwards in the getting started guide.

The GraphQL service for Sui RPC is rate-limited on all available instances to keep network throughput optimized and to protect against excessive or abusive calls to the service.

Queries are rate-limited at the number of attempts per minute to ensure high availability of the service to all users.

In addition to rate limits, queries are also validated against a number of rules on their complexity, such as the number of nodes, the depth of the query, or their payload size. Query the serviceConfig field to retrieve these limits. An example of how to query for some of the available limits follows:

Headers

The service accepts the following optional headers:

By default, each request returns the service's version in the response header: x-sui-rpc-version .

The response for the previous request looks similar to the following:

Variables offer a way to introduce dynamic inputs to a re-usable/static query. Declare variables in the parameters to a query or mutation , using the $ symbol and its type (in this example Int), which must be a scalar , enum , or input type. In the query body, refer to it by its name (prefixed with the $ symbol).

If you declare a variable but don't use it or define it (supply a value) in the query, the query fails to execute.

To learn more, read the GraphQL documentation on Variables .

In the following example, a variable supplies the ID of the epoch being queried.

Variables :

When using the online IDE, supply variables as a JSON object to the query in the Variables pane at the bottom of the main editing window. You receive a warning if you supply a variable but don't declare it.

When making a request to the GraphQL service (for example, using curl), pass the query and variables as two fields of a single JSON object:

Fragments are reusable units that you can include in queries as needed. To learn more, consult the official GraphQL documentation . The following example uses fragments to factor out a reusable snippet representing a Move value.

GraphQL supports queries that fetch multiple kinds of data, potentially nested. For example, the following query retrieves the first 20

transaction blocks (along with the digest, the sender's address, the gas object used to pay for the transaction, the gas price, and the gas budget) after a specific transaction block at epoch 97 .

If there are too many transactions to return in a single response, the service applies a limit on the maximum page size for variable size responses (like the transactionBlock query) and you must fetch further results through pagination .

Fields that return a paginated response accept at least the following optional parameters:

They also return a type that conforms to the GraphQL Cursor Connections Specification , meaning its name ends in Connection , and it contains at least the following fields:

Cursors are opaque identifiers for paginated results. The only valid source for a cursor parameter (like after and before) is a cursor field from a previous paginated response (like PageInfo.startCursor , PageInfo.endCursor , or Edge.cursor). The underlying format of the cursor is an implementation detail, and is not guaranteed to remain fixed across versions of the GraphQL service, so do not rely on it -- generating cursors out of thin air is not expected or supported.

Cursors are used to bound results from below (with after) and above (with before). In both cases, the bound is exclusive, meaning it does not include the result that the cursor points to in the bounded region.

Cursors also guarantee consistent pagination. If the first paginated query reads the state of the network at checkpoint X , then a future call to fetch the next page of results using the cursors returned by the first query continues to read from the network at checkpoint X , even if data for future checkpoints is now available.

This property requires that cursors that are used together (for example when supplying an after and before bound) are fixed on the same checkpoint, otherwise the query produces an error.

The GraphQL service does not support consistent pagination for arbitrarily old cursors. A cursor can grow stale, if the checkpoint it is from is no longer in the available range . You can query the upper- and lower-bounds of that range as follows:

The results are the first and last checkpoint for which pagination continues to work and produce a consistent result. At the time of writing the available range offers a 5- to 15-minute buffer period to finish pagination.

After results are bounded using cursors, a page size limit is applied using the first and last parameters. The service requires these parameters to be less than or equal to the max page size limit , and if you provide neither, it selects a default. In addition to setting a limit, first and last control where excess elements are discarded from. For example, if there are 10 potential results -- R0 , R1 , ..., R9 -- after cursor bounds have been applied, then

It is an error to apply both a first and a last limit.

To see these principles put into practice, consult the examples for paginating forwards and paginating backwards in the getting started guide.

The GraphQL service for Sui RPC is rate-limited on all available instances to keep network throughput optimized and to protect against excessive or abusive calls to the service.

Queries are rate-limited at the number of attempts per minute to ensure high availability of the service to all users.

In addition to rate limits, queries are also validated against a number of rules on their complexity, such as the number of nodes, the depth of the query, or their payload size. Query the serviceConfig field to retrieve these limits. An example of how to query for some of the available limits follows:

Variables

Variables offer a way to introduce dynamic inputs to a re-usable/static query. Declare variables in the parameters to a query or mutation , using the $ symbol and its type (in this example Int), which must be a scalar , enum , or input type. In the query body, refer to it by its name (prefixed with the $ symbol).

If you declare a variable but don't use it or define it (supply a value) in the query, the query fails to execute.

To learn more, read the GraphQL documentation on Variables .

In the following example, a variable supplies the ID of the epoch being queried.

Variables :

When using the online IDE, supply variables as a JSON object to the query in the Variables pane at the bottom of the main editing window. You receive a warning if you supply a variable but don't declare it.

When making a request to the GraphQL service (for example, using curl), pass the query and variables as two fields of a single JSON object:

Fragments are reusable units that you can include in queries as needed. To learn more, consult the official GraphQL documentation . The following example uses fragments to factor out a reusable snippet representing a Move value.

GraphQL supports queries that fetch multiple kinds of data, potentially nested. For example, the following query retrieves the first 20 transaction blocks (along with the digest, the sender's address, the gas object used to pay for the transaction, the gas price, and the gas budget) after a specific transaction block at epoch 97 .

If there are too many transactions to return in a single response, the service applies a limit on the maximum page size for variable size responses (like the transactionBlock query) and you must fetch further results through pagination .

Fields that return a paginated response accept at least the following optional parameters:

They also return a type that conforms to the GraphQL Cursor Connections Specification , meaning its name ends in Connection , and it contains at least the following fields:

Cursors are opaque identifiers for paginated results. The only valid source for a cursor parameter (like after and before) is a cursor field from a previous paginated response (like PageInfo.startCursor , PageInfo.endCursor , or Edge.cursor). The underlying format of the cursor is an implementation detail, and is not guaranteed to remain fixed across versions of the GraphQL service, so do not rely on it -- generating cursors out of thin air is not expected or supported.

Cursors are used to bound results from below (with after) and above (with before). In both cases, the bound is exclusive, meaning it does not include the result that the cursor points to in the bounded region.

Cursors also guarantee consistent pagination. If the first paginated query reads the state of the network at checkpoint X , then a future call to fetch the next page of results using the cursors returned by the first query continues to read from the network at checkpoint X , even if data for future checkpoints is now available.

This property requires that cursors that are used together (for example when supplying an after and before bound) are fixed on the same checkpoint, otherwise the query produces an error.

The GraphQL service does not support consistent pagination for arbitrarily old cursors. A cursor can grow stale, if the checkpoint it is from is no longer in the available range . You can query the upper- and lower-bounds of that range as follows:

The results are the first and last checkpoint for which pagination continues to work and produce a consistent result. At the time of writing the available range offers a 5- to 15-minute buffer period to finish pagination.

After results are bounded using cursors, a page size limit is applied using the first and last parameters. The service requires these parameters to be less than or equal to the max page size limit , and if you provide neither, it selects a default. In addition to setting a limit, first and last control where excess elements are discarded from. For example, if there are 10 potential results -- R0 , R1 , ..., R9 -- after cursor bounds have been applied, then

It is an error to apply both a first and a last limit.

To see these principles put into practice, consult the examples for paginating forwards and paginating backwards in the getting started guide.

The GraphQL service for Sui RPC is rate-limited on all available instances to keep network throughput optimized and to protect against excessive or abusive calls to the service.

Queries are rate-limited at the number of attempts per minute to ensure high availability of the service to all users.

In addition to rate limits, queries are also validated against a number of rules on their complexity, such as the number of nodes, the depth of the query, or their payload size. Query the serviceConfig field to retrieve these limits. An example of how to query for some of the available limits follows:

Fragments

Fragments are reusable units that you can include in queries as needed. To learn more, consult the official GraphQL documentation .

The following example uses fragments to factor out a reusable snippet representing a Move value.

GraphQL supports queries that fetch multiple kinds of data, potentially nested. For example, the following query retrieves the first 20 transaction blocks (along with the digest, the sender's address, the gas object used to pay for the transaction, the gas price, and the gas budget) after a specific transaction block at epoch 97 .

If there are too many transactions to return in a single response, the service applies a limit on the maximum page size for variable size responses (like the transactionBlock query) and you must fetch further results through pagination .

Fields that return a paginated response accept at least the following optional parameters:

They also return a type that conforms to the GraphQL Cursor Connections Specification , meaning its name ends in Connection , and it contains at least the following fields:

Cursors are opaque identifiers for paginated results. The only valid source for a cursor parameter (like after and before) is a cursor field from a previous paginated response (like PageInfo.startCursor , PageInfo.endCursor , or Edge.cursor). The underlying format of the cursor is an implementation detail, and is not guaranteed to remain fixed across versions of the GraphQL service, so do not rely on it -- generating cursors out of thin air is not expected or supported.

Cursors are used to bound results from below (with after) and above (with before). In both cases, the bound is exclusive, meaning it does not include the result that the cursor points to in the bounded region.

Cursors also guarantee consistent pagination. If the first paginated query reads the state of the network at checkpoint X , then a future call to fetch the next page of results using the cursors returned by the first query continues to read from the network at checkpoint X , even if data for future checkpoints is now available.

This property requires that cursors that are used together (for example when supplying an after and before bound) are fixed on the same checkpoint, otherwise the query produces an error.

The GraphQL service does not support consistent pagination for arbitrarily old cursors. A cursor can grow stale, if the checkpoint it is from is no longer in the available range . You can query the upper- and lower-bounds of that range as follows:

The results are the first and last checkpoint for which pagination continues to work and produce a consistent result. At the time of writing the available range offers a 5- to 15-minute buffer period to finish pagination.

After results are bounded using cursors, a page size limit is applied using the first and last parameters. The service requires these parameters to be less than or equal to the max page size limit , and if you provide neither, it selects a default. In addition to setting a limit, first and last control where excess elements are discarded from. For example, if there are 10 potential results -- R0 , R1 , ..., R9 -- after cursor bounds have been applied, then

It is an error to apply both a first and a last limit.

To see these principles put into practice, consult the examples for paginating forwards and paginating backwards in the getting started guide.

The GraphQL service for Sui RPC is rate-limited on all available instances to keep network throughput optimized and to protect against excessive or abusive calls to the service.

Queries are rate-limited at the number of attempts per minute to ensure high availability of the service to all users.

In addition to rate limits, queries are also validated against a number of rules on their complexity, such as the number of nodes, the depth of the query, or their payload size. Query the serviceConfig field to retrieve these limits. An example of how to query for some of the available limits follows:

Pagination

GraphQL supports queries that fetch multiple kinds of data, potentially nested. For example, the following query retrieves the first 20 transaction blocks (along with the digest, the sender's address, the gas object used to pay for the transaction, the gas price, and the gas budget) after a specific transaction block at epoch 97 .

If there are too many transactions to return in a single response, the service applies a limit on the maximum page size for variable size responses (like the transactionBlock query) and you must fetch further results through pagination .

Fields that return a paginated response accept at least the following optional parameters:

They also return a type that conforms to the GraphQL Cursor Connections Specification , meaning its name ends in Connection , and it contains at least the following fields:

Cursors are opaque identifiers for paginated results. The only valid source for a cursor parameter (like after and before) is a cursor field from a previous paginated response (like PageInfo.startCursor , PageInfo.endCursor , or Edge.cursor). The underlying format of the cursor is an implementation detail, and is not guaranteed to remain fixed across versions of the GraphQL service, so do not rely on it -- generating cursors out of thin air is not expected or supported.

Cursors are used to bound results from below (with after) and above (with before). In both cases, the bound is exclusive, meaning it does not include the result that the cursor points to in the bounded region.

Cursors also guarantee consistent pagination. If the first paginated query reads the state of the network at checkpoint X , then a future call to fetch the next page of results using the cursors returned by the first query continues to read from the network at checkpoint X , even if data for future checkpoints is now available.

This property requires that cursors that are used together (for example when supplying an after and before bound) are fixed on the same checkpoint, otherwise the query produces an error.

The GraphQL service does not support consistent pagination for arbitrarily old cursors. A cursor can grow stale, if the checkpoint it is from is no longer in the available range . You can query the upper- and lower-bounds of that range as follows:

The results are the first and last checkpoint for which pagination continues to work and produce a consistent result. At the time of writing the available range offers a 5- to 15-minute buffer period to finish pagination.

After results are bounded using cursors, a page size limit is applied using the first and last parameters. The service requires these parameters to be less than or equal to the max page size limit , and if you provide neither, it selects a default. In addition to setting a limit, first and last control where excess elements are discarded from. For example, if there are 10 potential results -- R0 , R1 , ..., R9 -- after cursor bounds have been applied, then

It is an error to apply both a first and a last limit.

To see these principles put into practice, consult the examples for paginating forwards and paginating backwards in the getting started guide.

The GraphQL service for Sui RPC is rate-limited on all available instances to keep network throughput optimized and to protect against excessive or abusive calls to the service.

Queries are rate-limited at the number of attempts per minute to ensure high availability of the service to all users.

In addition to rate limits, queries are also validated against a number of rules on their complexity, such as the number of nodes, the depth of the query, or their payload size. Query the serviceConfig field to retrieve these limits. An example of how to query for some of the available limits follows:

Limits

The GraphQL service for Sui RPC is rate-limited on all available instances to keep network throughput optimized and to protect against excessive or abusive calls to the service.

Queries are rate-limited at the number of attempts per minute to ensure high availability of the service to all users.

In addition to rate limits, queries are also validated against a number of rules on their complexity, such as the number of nodes, the depth of the query, or their payload size. Query the serviceConfig field to retrieve these limits. An example of how to query for some of the available limits follows:

Related links