

GraphQL Queries Tutorial

LogScale collects server logs and related metrics to provide the ability to monitor the activities of servers to identify security breaches and attacks, improve server performance, and more.

Administrators and other authorized users may need to retrieve data on users, groups, and the configuration of a LogScale instance. They may also need to know the settings for alerts and other actions. To obtain any of this data, they'll have to query LogScale.

This page covers query syntax and how to execute a query. For instructional purposes, it does so with generic queries, straightforward, but fake queries. Actual queries include more components, involving complexity that is not shown here. The goal is to learn the basics of a query.

Generic Syntax for One

To begin, consider a very generic query — a fictitious one — called, `thing()`. It's meant to confirm that a given thing exists in a list of things in LogScale. Below is what its syntax might be:

```
GRAPHQL
```

```
thing( name: string!  
      ): boolean
```

The syntax shows the input within parentheses. Input is always given with a key and a value: the `name` is the key. You would give the name of the thing, the value as a string, within double-quotes.

You may have noticed the exclamation mark. That's meant to indicate that a value is required; you don't type the exclamation mark. Similarly, after the closing parenthesis there is a colon. You don't enter it in a query. It means the results of the query follow.

Here's an example using this query:

```
GRAPHQL

query {
  thing( name: "thing-one" )
}
```

You might enter that in the API Explorer, but don't since it's a made-up query. The API Explorer is covered in another tutorial. Hold off on reading it for now. Below are the would-be results from the query:

```
GRAPHQL

{
  "data": {
    "thing": true
  }
}
```

The results for this fake query uses the boolean datatype. This means that a value of `true` will be returned if the query is successful (i.e., the thing is found), and `false` if not (i.e., there is no such thing). A third possible result could be returned: if the user doesn't have permission to use the query, or if the query was entered incorrectly, or some other problem occurs, an error message will be returned.

Generic Syntax for Several

Let's consider another fictitious query, `things()`. This one supposedly provides a list of things in LogScale, and not just confirmation of existence. Below is how its syntax might look:

```
GRAPHQL

things(
  includeDisabled: boolean
): Thing!
```

For this query, there's a different input parameter. You would specify whether or not things that are disabled should be included in

specify whether things that are disabled should be included in the results. It's optional, with a default value of `false`. There's nothing in the syntax to indicate that, but it would be mentioned in the documentation page for this query.

For the results, the syntax shows it uses a `Thing` datatype — which is also made up. Were there a documentation page for this query, it would show that there are a few returned values that may be chosen. Below is a example of how the query might look in use:

GRAPHQL

```
query {  
  things( includeDisabled: True )  
    { name, id, enabled }  
}
```

This example query is asking to include disabled things in the search. It's requesting the name and identifier of each thing, and indication as to whether each is enabled or not — whatever that might mean. Notice that the boolean input parameter's value is not contained within double-quotes, and that the list of requested returned values is contained within a pair of curly-braces. This is common to real queries.

Below is how the results might look for the query above:

GRAPHQL

```
{  
  "data": {  
    "things": [  
      {  
        "name": "thing-one",  
        "id": "abc123",  
        "enabled": true  
      },  
      {  
        "name": "thing-two",  
        "id": "def456",  
        "enabled": true  
      },  
      {  
        "name": "thing-three",
```

```
      "id": "ghi789",  
      "enabled": false  
    }  
  }
```

As you can see, this is in standard json format. You could depend on this format and capture the returned values in an API script you might write in a programming language like Perl or Python.

There are more possibilities with queries: how their syntax is composed, the use of special datatypes for the input, the use of sub-datatypes for input and the results, etc. However, this was a good start for understanding queries and how they work.

There are a couple of recommended next steps for learning about using GraphQL. You can either go through the generic tutorial on mutations, which is similar to this one, or you can read and experiment with real queries with the other query tutorials (e.g., [Repository Queries](#)), which cover core queries that you'll need to learn and use often.



© 2026 CrowdStrike All other marks contained herein are the property of their respective owners.