IMPORTANT: No additional bug fixes or documentation updates will be released for this version. For the latest information, see the current release documentation.

Kibana Query Language

The Kibana Query Language (KQL) makes it easy to find the fields and syntax for your Elasticsearch query. If you have the Basic tier or above, simply place your cursor in the **Search** field. As you type, you'll get suggestions for fields, values, and operators.

If you prefer to use Kibana's legacy query language, based on the Lucene query syntax, click **KQL** next to the **Search** field, and then turn off KQL.

Terms query

A terms query matches documents that contain one or more **exact** terms in a field.

To match documents where the response field is `200`:

```
response:200
```

To match documents with the phrase "quick brown fox" in the `message` field.

```
message:"quick brown fox"
```

Without the quotes, the query matches documents regardless of the order in which they appear. Documents with "quick brown fox" match, and so does "quick fox brown".

NOTE Terms without fields are matched against the default field in your index settings. If a default field is not set, terms are matched against all fields. For example, a query for `response:200` searches for the value 200 in the response field, but a query for just `200` searches for 200 across all fields in your index.

Boolean queries

KQL supports `or`, `and`, and `not`. By default, `and` has a higher precedence than `or`. To override the default precedence, group operators in parentheses.

To match documents where response is `200`, extension is `php`, or both:

```
response:200 or extension:php
```

To match documents where response is `200` and extension is `php`:

```
response:200 and extension:php
```

To match documents where response is `200` or `404`.

```
response:(200 or 404)
```

To match documents where response is `200` and extension is either `php` or `css`:

```
response:200 and (extension:php or extension:css)
```

To match documents where `response` is 200 and `extension` is `php` or extension is `css`, and response is anything:

```
response:200 and extension:php or extension:css
```

To match documents where response is not `200`:

```
not response:200
```

To match documents where response is `200` but extension is not `php` or `css`.

```
response:200 and not (extension:php or extension:css)
```

To match multi-value fields that contain a list of terms:

```
tags:(success and info and security)
```

Range queries

KQL supports `>`, `>=`, `<`, and `<=` on numeric and date types.

```
account_number >= 100 and items_sold <= 200
```

Date range queries

Typically, Kibana's time filter is sufficient for setting a time range, but in some cases you might need to search on dates. Include the date range in quotes.

```
@timestamp < "2021-01-02T21:55:59"
```

```
@timestamp < "2021-01"
```

```
@timestamp < "2021"
```

Exist queries

An exist query matches documents that contain a value for a field, in this case, response:

```
response:*
```

Wildcard queries

To match documents where machine.os starts with `win`, such as "windows 7" and "windows 10":

```
machine.os:win*
```

To match multiple fields:

```
machine.os*:windows 10
```

This syntax is handy when you have text and keyword versions of a field. The query checks machine.os and machine.os.keyword for the term `windows 10`.

Nested field queries

A main consideration for querying nested fields is how to match parts of the nested query to the individual nested documents. You can:

- **Match parts of the query to a single nested document only.** This is what most users want when querying on a nested field.
- **Match parts of the query to different nested documents.** This is how a regular object field works. This query is generally less useful than matching to a single document.

In the following document, `items` is a nested field. Each document in the nested field contains a name, stock, and category.

```
{
  "grocery_name": "Elastic Eats",
  "items": [
    {
      "name": "banana",
      "stock": "12",
      "category": "fruit"
    },
    {
      "name": "peach",
      "stock": "10",
      "category": "fruit"
    },
    {
      "name": "carrot",
      "stock": "9",
      "category": "vegetable"
    },
    {
      "name": "broccoli",
      "stock": "5",
      "category": "vegetable"
    }
  ]
}
```

Match a single document

To match stores that have more than 10 bananas in stock:

```
items:{ name:banana and stock > 10 }
```

`items` is the nested path. Everything inside the curly braces (the nested group) must match a single nested document.

The following query does not return any matches because no single nested document has bananas with a stock of 9.

```
items:{ name:banana and stock:9 }
```

Match different documents

The following subqueries are in separate nested groups and can match different nested documents:

```
items:{ name:banana } and items:{ stock:9 }
```

`name:banana` matches the first document in the array and `stock:9` matches the third document in the array.

Match single and different documents

To find a store with more than 10 bananas that **also** stocks vegetables:

```
items:{ name:banana and stock > 10 } and items:{ category:vegetable }
```

The first nested group (`name:banana and stock > 10`) must match a single document, but the `category:vegetables` subquery can match a different nested document because it is in a separate group.

Nested fields inside other nested fields

KQL supports nested fields inside other nested fields—you have to specify the full path. In this document, `level1` and `level2` are nested fields:

```
{
  "level1": [
    {
      "level2": [
        {
          "prop1": "foo",
          "prop2": "bar"
        },
        {
          "prop1": "baz",
          "prop2": "qux"
        }
      ]
    }
  ]
}
```

To match on a single nested document:

```
level1.level2:{ prop1:foo and prop2:bar }
```


On this page

Terms query
Boolean queries
Range queries
Date range queries
Exist queries
Wildcard queries
Nested field queries

Kibana Guide: 7.9
What is Kibana?
What's new in 7.9
Get started
Set up Kibana
Discover
Create an index pattern
Set the time filter
Search data
Kibana Query Language
Lucene query syntax
Save a search
Save a query
Filter by field
View document data
View a document in context
View field data statistics
Dashboard
Canvas
Maps
Machine learning
Graph
Visualize
Observability
Logs
Metrics
APM
Uptime
Elastic Security
Dev Tools
Stack Monitoring
Stack Management
Ingest Manager
Reporting
Alerting and Actions
REST API
Kibana plugins
Accessibility
Breaking Changes
Release Notes
Developer guide