# Prometheus

Part 1

# Agenda

**1- Prometheus Basics**

- **What Is Prometheus?**
- **Prometheus Architecture — Bird's-Eye View**
- **Prometheus Use Cases — Strengths and Limitations**

**2- Installation and Configuration**

- **Building a Prometheus Server**
- **Configuring Prometheus**
- **Configuring an Exporter**

**3- Prometheus Data Model**

- **What Is Time-Series Data?**
- **Metrics and Labels**
- **Metric Types**

**4- Querying**

- **Query Basics**
- **Query Operators**
- **Query Functions**
- **Prometheus HTTP API**

# Prometheus Basics

# Introduction to Prometheus

**Prometheus** is an open-source **monitoring** and **alerting** tool. It collects data about applications and systems and allows you to **visualize** the data and issue **alerts** based on the data.

**Monitoring** is an important component of DevOps automation. To manage a robust and complex infrastructure, you need to be able to quickly and easily understand what is happening in your systems!

## High-Level Use Cases

- **Metric collection** - Collect important metrics about your systems and applications in one place.
- **Visualization** - Build dashboards that provide an overview of the health of your systems.
- **Alerting** - Receive an email when something is broken.

# Introduction to Prometheus

## Prometheus Background

- **Language** - Prometheus is primarily written in **Go**. Some components are written in **Java**, **Python**, and **Ruby**.
- **License** - Prometheus uses the open-source **Apache 2.0** license.
- **History** - Prometheus development was started by **Matt T. Proud** and **Julius Volz**. It was initially sponsored by **SoundCloud**. Today, it is a fully open-source project maintained by many individuals and organizations.
- **Website** - More information and full documentation can be found at **prometheus.io**.

## Prometheus Pull Model

Prometheus collects metrics using a **pull model**. This means the Prometheus server pulls metric data from exporters — agents do not push data to the Prometheus server.

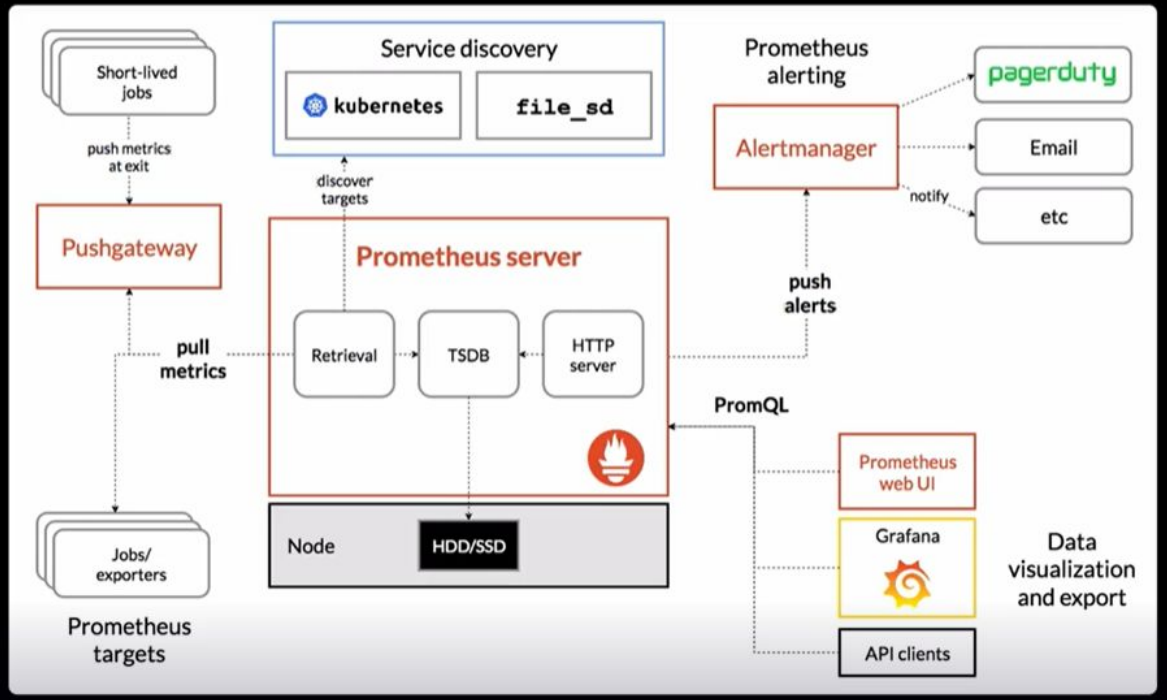
# Introduction to Prometheus

## Prometheus Architecture

## Prometheus Pull Model

Prometheus collects metrics using a **pull model**. This means the Prometheus server pulls metric data from exporters — agents do not push data to the Prometheus server.

# Introduction to Prometheus




- **Prometheus Server** - Collects metric data.
- **Exporters** - Provide metric data for Prometheus to consume.
- **Client Libraries** - Easily turn your custom application into an exporter that exposes metrics in a format Prometheus can consume.
- **Prometheus Pushgateway** - Allows pushing metrics to Prometheus for certain specific use cases.
- **Alertmanager** - Sends alerts triggered by metric data.
- **Visualization Tools** - Provide useful ways to view metric data. These are not necessarily part of Prometheus.

# Prometheus Use Cases — Strengths and Limitations

## Prometheus Use Case: Metric Collection

**Judy** is the administrator of a large infrastructure consisting of many servers, virtual machines, and components running in the cloud. She wants to be able to monitor the health of on-premises virtual machines as well as instances running in the cloud. Since all these components are interconnected, she does not want to have to search for data in multiple places when there is an outage.

Prometheus allows Judy to collect all this data in one place.

# Prometheus Use Cases — Strengths and Limitations

## Prometheus Use Case: Visualization

Mark's company has a complex IT infrastructure, and he is periodically on-call to respond to any outages that occur outside normal business hours. When an outage occurs, it often takes him a long time to discover which server or application is down. He wants to create a dashboard with graphs and charts that can show him what is happening at a glance so he can quickly diagnose problems and get back to sleep.

Prometheus allows Mark to build a dashboard that provides statistics on all his servers and applications. When something goes wrong, it is easy to spot where the problem lies.

# Prometheus Use Cases — Strengths and Limitations

## Prometheus Use Case: Alerting

Sarah's company has been having some problems with their network. She is frequently putting out fires when services become unavailable for customers. She is aware of ways to detect and fix these outages before they become serious, but it would be unrealistic to have someone constantly watch the data to determine when a problem is about to occur. She would like to build some automation to send her an email whenever the network issue is about to happen again.

With Prometheus, Sarah creates an alert that will trigger when the metric data indicates a problem is about to occur. Next time the issue is about to happen, she is able to take steps to prevent it.

# Prometheus Use Cases — Strengths and Limitations

## Prometheus Limitations

While Prometheus is a great tool for a variety of use cases, it is important to understand when it is not the best tool.

- **100% accuracy** (e.g., per-request billing) - Prometheus is designed to operate even under failure conditions. This means it will continue to provide data even if new data is not available due to failures and outages. If you need 100% up-to-the-minute accuracy, such as in the case of per-request billing, Prometheus may not be the best tool to use.

- **Non time-series data** (e.g., log aggregation) - Prometheus is built to monitor time-series metrics, especially data that is numeric. It is not the best choice for collecting more generic types of data, such as system logs.

# Installation and Configuration

# Building a Prometheus Server

## Installation Options

There are multiple ways to install and run prometheus.

 +- Use pre-compiled binaries. Download from https://prometheus.io/download/.
 +- Build from source. Source code can be found on GitHub at https://github.com/prometheus.
 +- Run with Docker using pre-built images.

# Building a Prometheus Server

```
cloud_user@wboyd1c:~$ sudo chown -R prometheus:prometheus /etc/prometheus
cloud_user@wboyd1c:~$ sudo chown prometheus:prometheus /var/lib/prometheus
cloud_user@wboyd1c:~$ prometheus --config.file=/etc/prometheus/prometheus.yml
level=info ts=2020-02-18T18:31:30.892Z caller=main.go:295 msg="no time or size retention was set so using the de
fault time retention" duration=15d
level=info ts=2020-02-18T18:31:30.892Z caller=main.go:331 msg="Starting Prometheus" version="(version=2.16.0, br
anch=HEAD, revision=b90be6f32a33c03163d700e1452b54454ddce0ec)"
level=info ts=2020-02-18T18:31:30.892Z caller=main.go:332 build_context="(go=go1.13.8, user=root@7ea0ae865f12, d
ate=20200213-23:50:02)"
level=info ts=2020-02-18T18:31:30.892Z caller=main.go:333 host_details="(Linux 4.15.0-1058-aws #60-Ubuntu SMP We
d Jan 15 22:35:20 UTC 2020 x86_64 wboyd1c.mylabserver.com (none))"
level=info ts=2020-02-18T18:31:30.892Z caller=main.go:334 fd_limits="(soft=1024, hard=1048576)"
level=info ts=2020-02-18T18:31:30.892Z caller=main.go:335 vm_limits="(soft=unlimited, hard=unlimited)"
level=info ts=2020-02-18T18:31:30.903Z caller=main.go:661 msg="Starting TSDB ..."
level=info ts=2020-02-18T18:31:30.909Z caller=web.go:508 component=web msg="Start listening for connections" add
ress=0.0.0.0:9090
level=info ts=2020-02-18T18:31:30.931Z caller=head.go:577 component=tsdb msg="replaying WAL, this may take awhil
e"
level=info ts=2020-02-18T18:31:30.931Z caller=head.go:625 component=tsdb msg="WAL segment loaded" segment=0 maxS
egment=0
level=info ts=2020-02-18T18:31:30.937Z caller=main.go:676 fs_type=EXT4_SUPER_MAGIC
level=info ts=2020-02-18T18:31:30.937Z caller=main.go:677 msg="TSDB started"
level=info ts=2020-02-18T18:31:30.937Z caller=main.go:747 msg="Loading configuration file" filename=/etc/prometh
eus/prometheus.yml
level=info ts=2020-02-18T18:31:30.942Z caller=main.go:775 msg="Completed loading of configuration file" filename
=/etc/prometheus/prometheus.yml
level=info ts=2020-02-18T18:31:30.943Z caller=main.go:630 msg="Server is ready to receive web requests."
```

# Building a Prometheus Server

The prometheus.service file

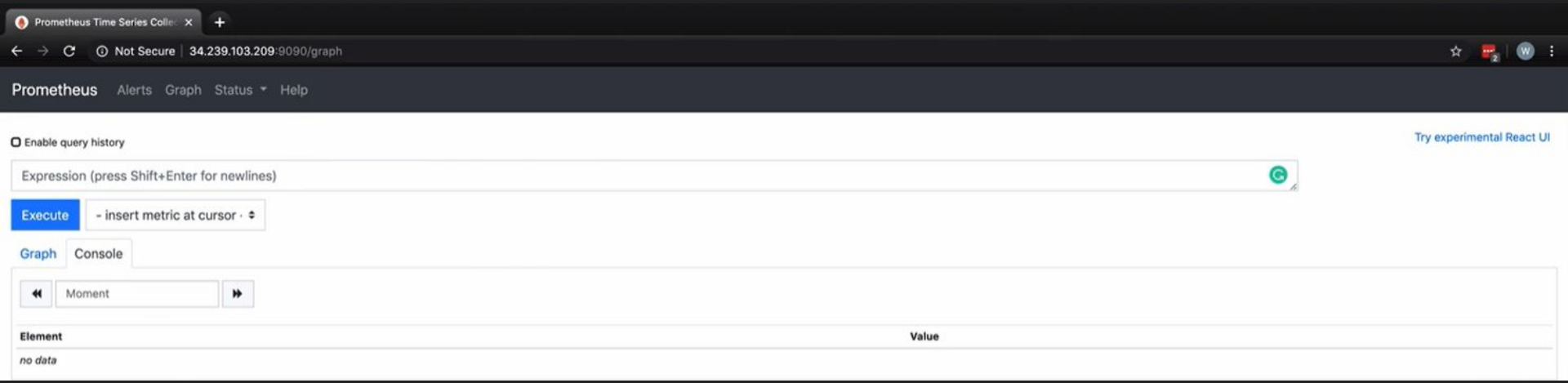
```
[Unit]
Description=Prometheus Time Series Collection and Processing Server
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
    --config.file /etc/prometheus/prometheus.yml \
    --storage.tsdb.path /var/lib/prometheus/ \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries

[Install]
WantedBy=multi-user.target
~
```

# Building a Prometheus Server

```
cloud_user@wboyd1c:~$ sudo vi  /etc/systemd/system/prometheus.service
cloud_user@wboyd1c:~$ sudo systemctl daemon-reload
cloud_user@wboyd1c:~$ sudo systemctl start prometheus
cloud_user@wboyd1c:~$ sudo systemctl enable prometheus
Created symlink /etc/systemd/system/multi-user.target.wants/prometheus.service → /etc/systemd/system/prometheus.
service.
cloud_user@wboyd1c:~$
```



# Configuring Prometheus

## The Prometheus Configuration File

Prometheus can be configured using a configuration file. Run Prometheus with the `--config.file` flag to specify the location of this file. We set up our systemd service to use the configuration file at `/etc/prometheus/prometheus.yml`.

The configuration file used the YAML format.

Refer to the Prometheus documentation for detailed information on all the available configuration options.

# Configuring Prometheus

## Reloading the Configuration

Restarting Prometheus will load the new configuration.

Prometheus can also reload its configuration at runtime without the need for a restart. One way to do this is to send a SIGHUP signal to the Prometheus process (e.g., `sudo killall -HUP prometheus`).

# Configuring Prometheus

/etc/prometheus/prometheus.yml

```
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.
```

# Configuring Prometheus

```
cloud_user@wboyd1c:~$ curl localhost:9090/api/v1/status/config
{"status":"success","data":{"yaml":"global:\n  scrape_interval: 10s\n  scrape_timeout: 10s\n  evaluation_interva
l: 15s\nalerting:\n  alertmanagers:\n  - static_configs:\n    - targets: []\n    scheme: http\n    timeout: 10s\
n    api_version: v1\nscrape_configs:\n- job_name: prometheus\n  honor_timestamps: true\n  scrape_interval: 10s\
n  scrape_timeout: 10s\n  metrics_path: /metrics\n  scheme: http\n  static_configs:\n  - targets:\n    - localho
st:9090\n"}}cloud_user@wboyd1c:~$
```

# Prometheus Exporter

## Exporters

A Prometheus exporter is any application that exposes metric data in a format that can be collected (or "scraped") by the Prometheus server.

For example, Node Exporter runs on a *NIX machine and collects a variety of system metrics. It then exposes them to Prometheus.

# Prometheus Exporter

Scrape Config

The scrape_config section of the Prometheus config file provides a list of targets the Prometheus server will scrape, such as a Node Exporter running on a Linux machine.

Prometheus server will scrape these targets periodically to collect metric data.

# Prometheus Data Model

# What Is Time-Series Data?

**Prometheus is built around storing time-series data.**

**Time-series data consists of a series of values associated with different points in time.**

**Single Data Point vs. Time Series**

**You could track a single data point, such as the current outdoor temperature.**

- Outdoor temperature: 0C/31F

**However, if you write down the temperature once every hour, that's a time series!**

- 08:00AM___ -6C/21F
- 09:00AM___ -3C/26F
- 10:00AM___ -2C/28F
- 11:00AM___ 0C/31F

# Time-Series Metric Example

Every metric in Prometheus tracks a particular value over time.

For example, Prometheus might track the available memory for a server, with an entry in the time series for every minute.

```
02-21-2020 09:55AM - node_memory_MemAvailable_bytes=3734269952
02-21-2020 09:56AM - node_memory_MemAvailable_bytes=3734276545
02-21-2020 09:57AM - node_memory_MemAvailable_bytes=3734295563
02-21-2020 09:58AM - node_memory_MemAvailable_bytes=3734263327
```

# What Is Time-Series Data?

## Time-Series Data in Prometheus

All Prometheus data is fundamentally stored as **time-series data**.

This means Prometheus not only tracks the current **value** of each metric but also **changes** to each metric over time.

# Metrics and Labels

## Metric Names

Every metric in Prometheus has a metric name. The metric name refers to the general feature of a system or application that is being measured.

An example of a metric name:

```
node_cpu_seconds_total
```

`node_cpu_seconds_total` measures the total amount of CPU time being used in CPU seconds.

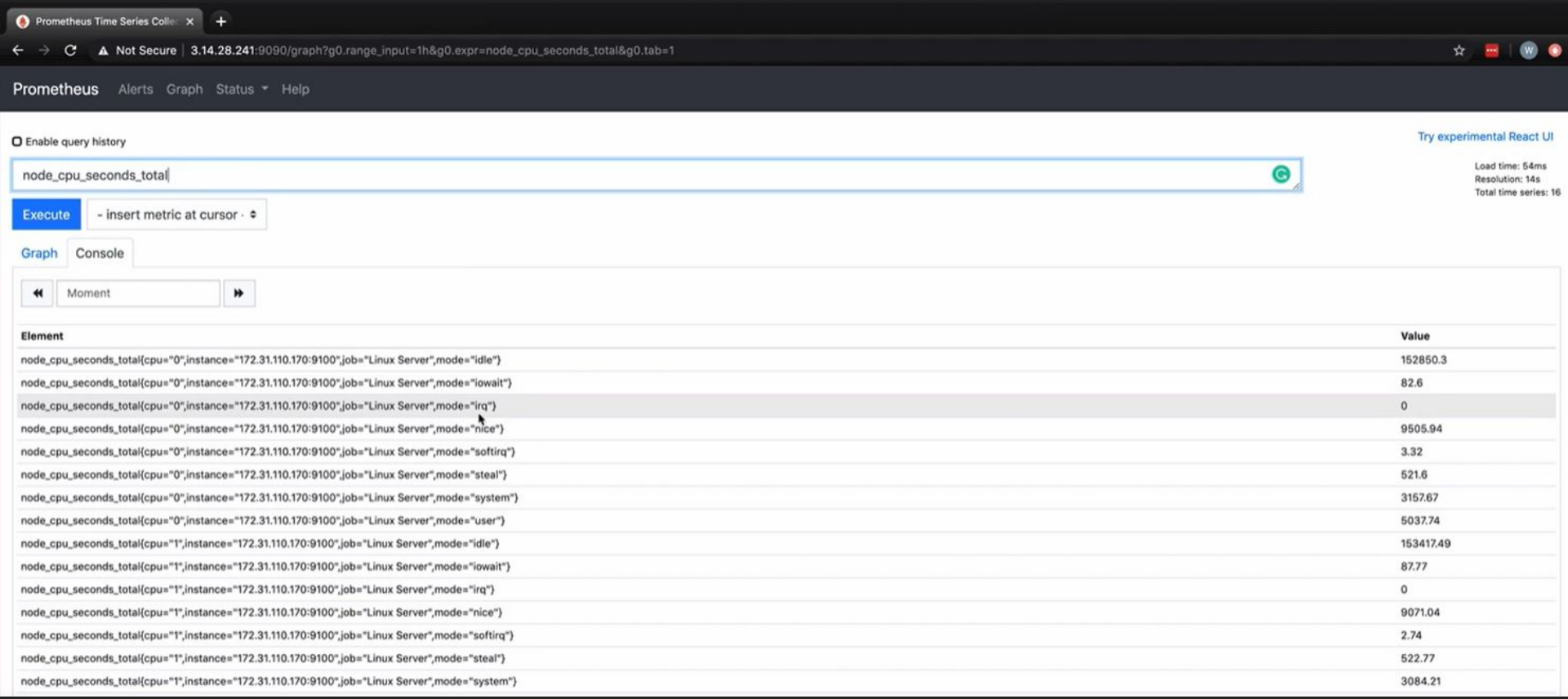
# Metrics and Labels

## Metric Names

Note that the metric name merely refers to the feature being measured. Metric names do not point to a specific data value but potentially a collection of many values.

Simply querying `node_cpu_seconds_total` would likely return a list of multiple data points, such as CPU usage for multiple CPUs on a server, or even multiple servers.

# Metrics and Labels



# Metrics and Labels

## Metric Labels

Prometheus uses labels to provide a dimensional data model. This means we can use labels to specify additional things, such as which node's CPU usage is being represented.

A unique combination of a metric name and a set of labels identifies a particular set of time-series data. This example uses a label called `cpu` to refer to usage of a specific CPU.

```
node_cpu_seconds_total{cpu="0"}
```

# Metrics and Labels

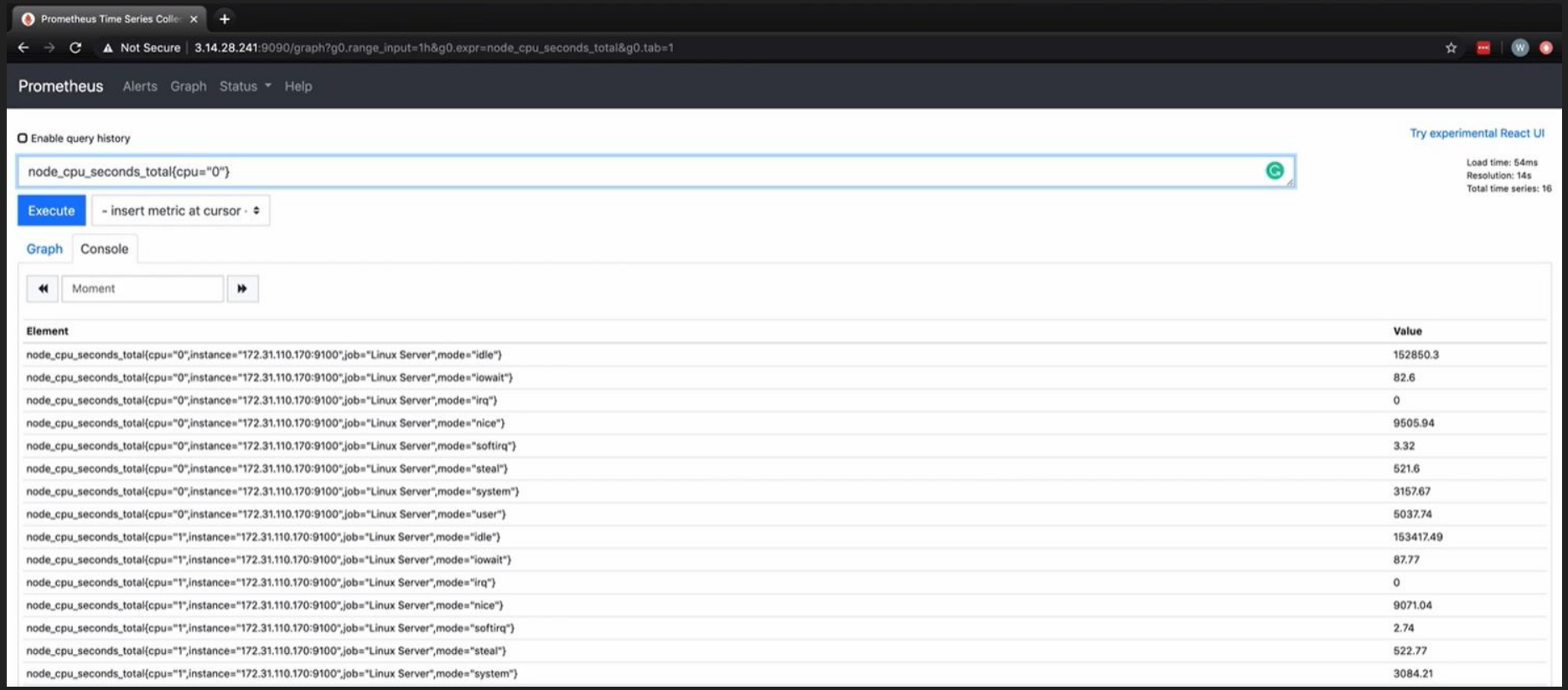
## Metric Labels

Most Prometheus metrics have multiple labels.

```
node_cpu_seconds_total{cpu="0",instance="10.0.1.102:9100",mode="idle"} 287.75
node_cpu_seconds_total{cpu="0",instance="10.0.1.102:9100",mode="user"}  51.45
```

Using metric names and labels, you can write queries that do things like average CPU usage across a whole data center as well as drill down into the CPU usage of a single CPU on a single node.

# Metrics and Labels



# Metric Types

## Metric Types

Metric types refer to different ways in which exporters represent the metric data they provide.

Metric types are not represented in any special way in a Prometheus server, but it is important to understand them in order to properly interpret your metrics.

# Metric Types

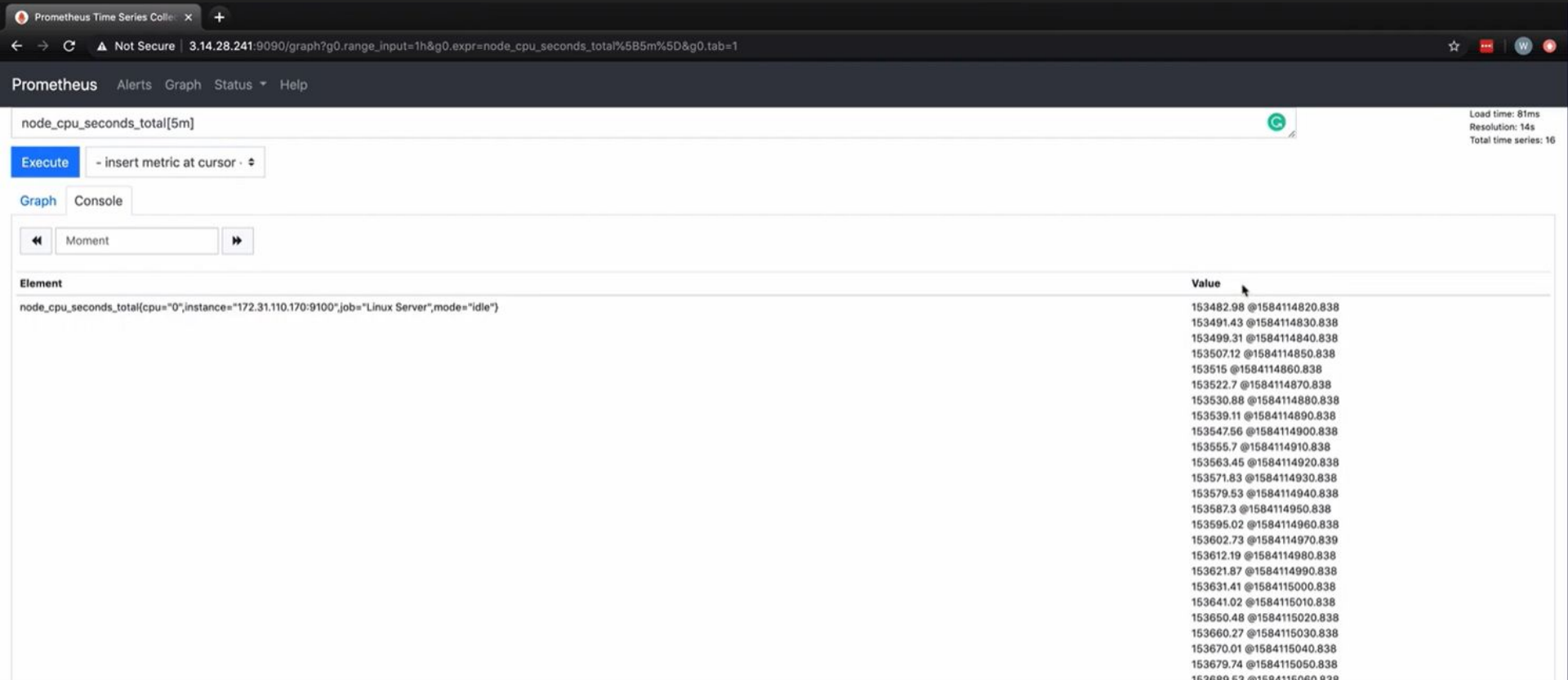
Counter

A counter is a single number that can only increase or be reset to zero. Counters represent cumulative values.

Total HTTP requests served:
 +- 0
 +- 12
 +- 85
 +- 276

Examples:
 +- Number of HTTP requests served by an application
 +- Number of records processed
 +- Number of application restarts
 +- Number of errors

# Metric Types



# Metric Types

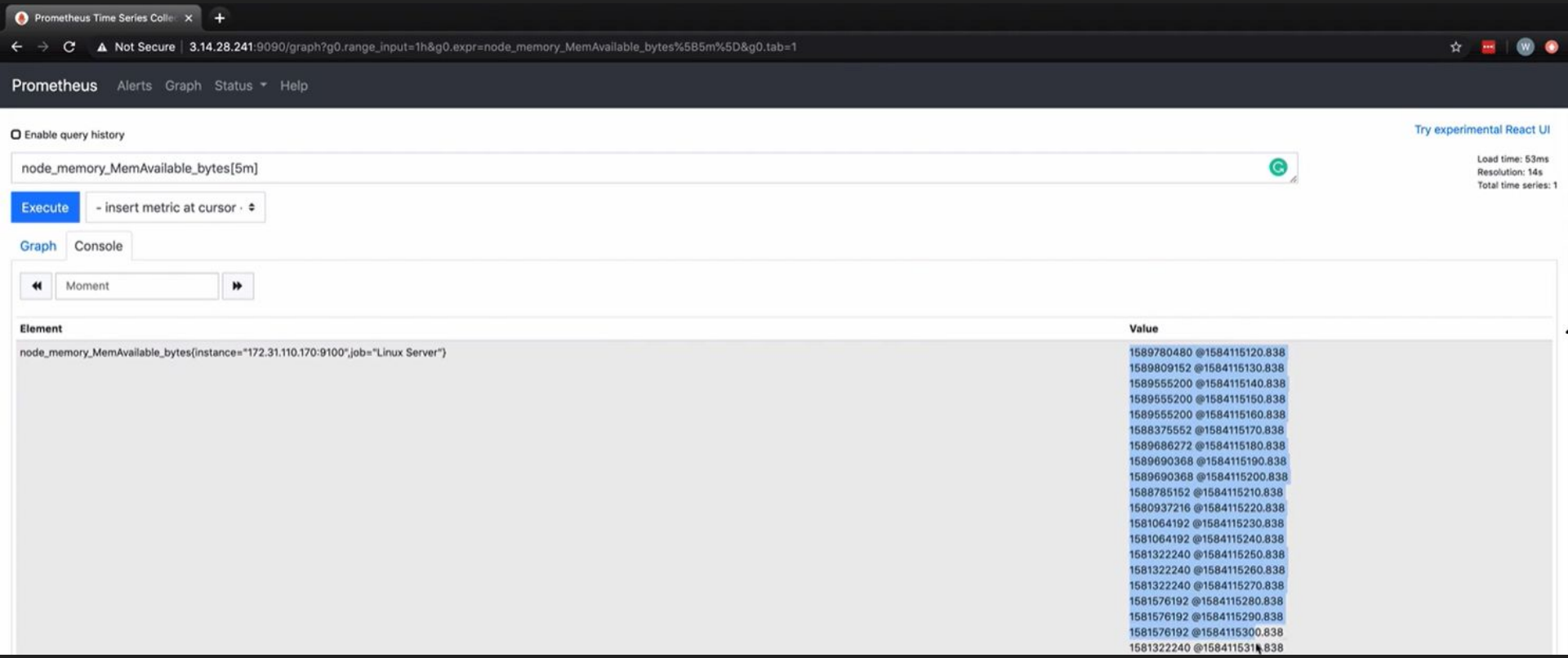
Gauge

A gauge is a single number that can increase and decrease over time.

Current HTTP requests active:
 +- 76
 +- 82
 +- 24
 +- 56

Examples:
 +- Number of concurrent HTTP requests
 +- CPU usage
 +- Memory usage
 +- Current active threads

# Metric Types



# Metric Types

Histogram

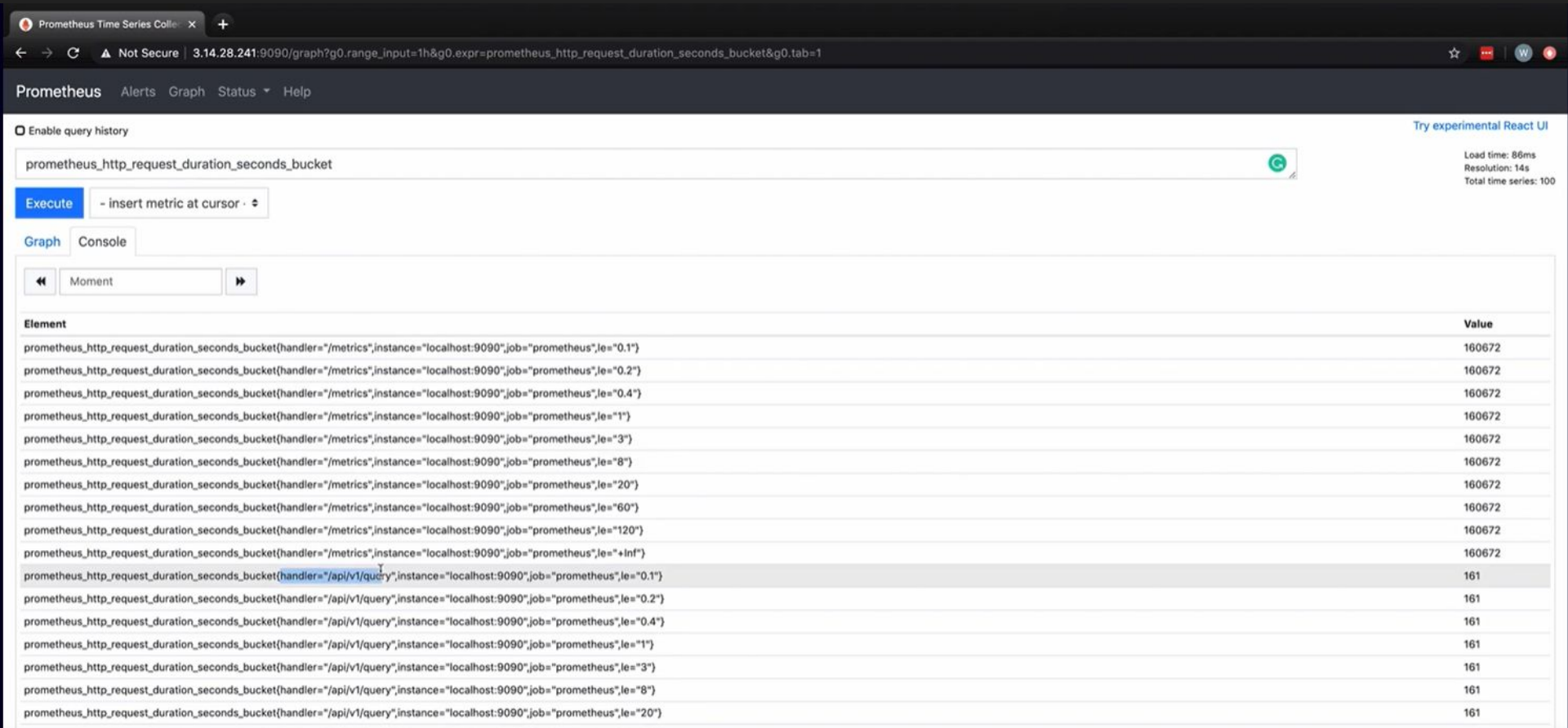
A histogram counts the number of observations/events that fall into a set of configurable buckets, each with its own separate time series. A histogram will use labels to differentiate between buckets. The below example provides the number of HTTP requests whose duration falls into each bucket.

```
http_request_duration_seconds_bucket{le="0.3"}
http_request_duration_seconds_bucket{le="0.6"}
http_request_duration_seconds_bucket{le="1.0"}
```

Histograms also include separate metric names to expose the _sum of all observed values and the total _count of events.

```
http_request_duration_seconds_sum
http_request_duration_seconds_count
```

# Metric Types



# Metric Types

Histogram

A histogram counts the number of observations/events that fall into a set of configurable buckets, each with its own separate time series. A histogram will use labels to differentiate between buckets. The below example provides the number of HTTP requests whose duration falls into each bucket.

```
http_request_duration_seconds_bucket{le="0.3"}
http_request_duration_seconds_bucket{le="0.6"}
http_request_duration_seconds_bucket{le="1.0"}
```

Histograms also include separate metric names to expose the _sum of all observed values and the total _count of events.

```
http_request_duration_seconds_sum
http_request_duration_seconds_count
```

# Metric Types

Summary

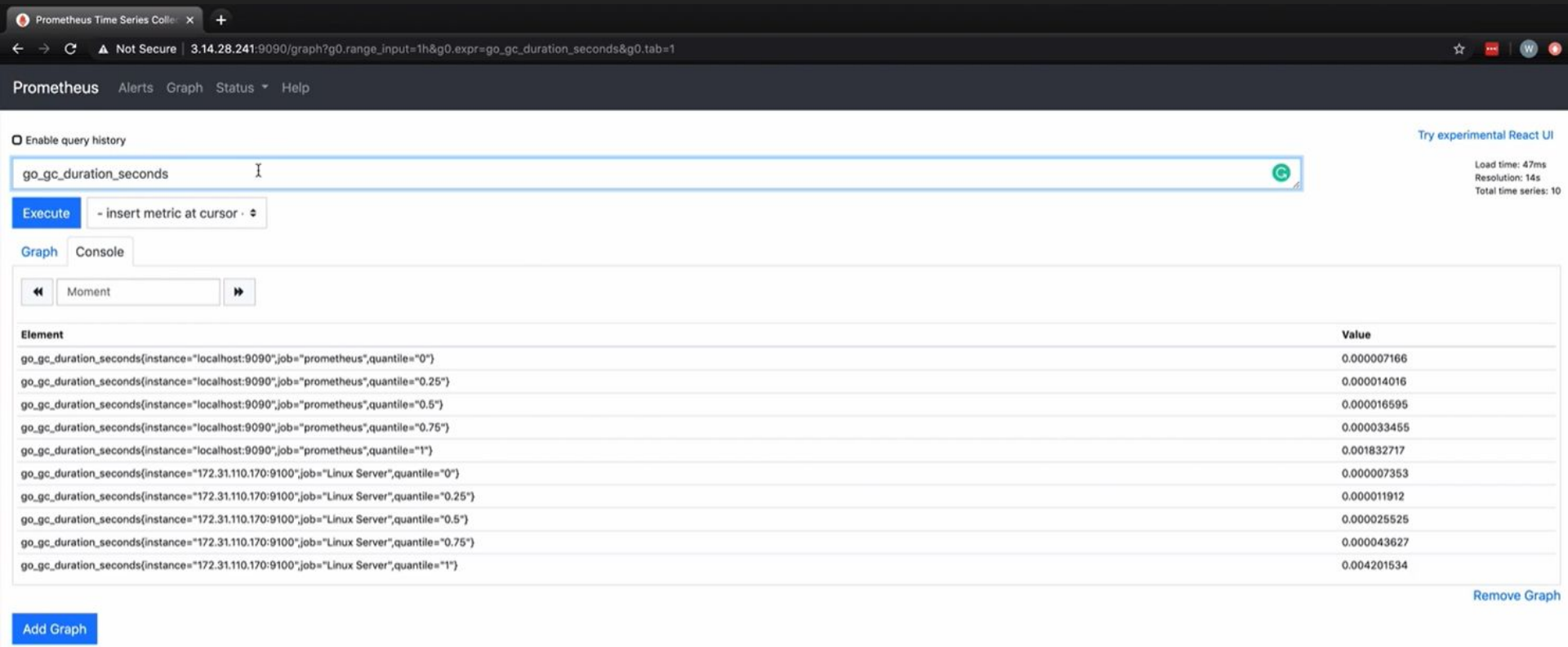
A summary is similar to a histogram, but it exposes metrics in the form of quantiles instead of buckets. While buckets divide values based on specific boundaries, quantiles divide values based on the percentiles into which they fall.

This value represents the number of HTTP requests whose duration falls within the 95th percentile of all requests or the top 5% longest requests.

```
http_request_duration_seconds{quantile="0.95"})
```

Like histograms, summaries also expose the _sum and _count metrics.

# Metric Types



# Querying

# Querying

Querying allows you to access and work with your metric data in Prometheus.

You can use **PromQL** (Prometheus Query Language) to write queries and retrieve useful information from the metric data collected by Prometheus.

You can use Prometheus queries in a variety of ways to obtain and work with data.

- Expression browser
- Prometheus HTTP API
- Visualization tools such as Grafana

# Query Basics

## Selectors

The most basic component of the PromQL query is a time-series selector. A time-series selector is simply a metric name, optionally combined with labels.

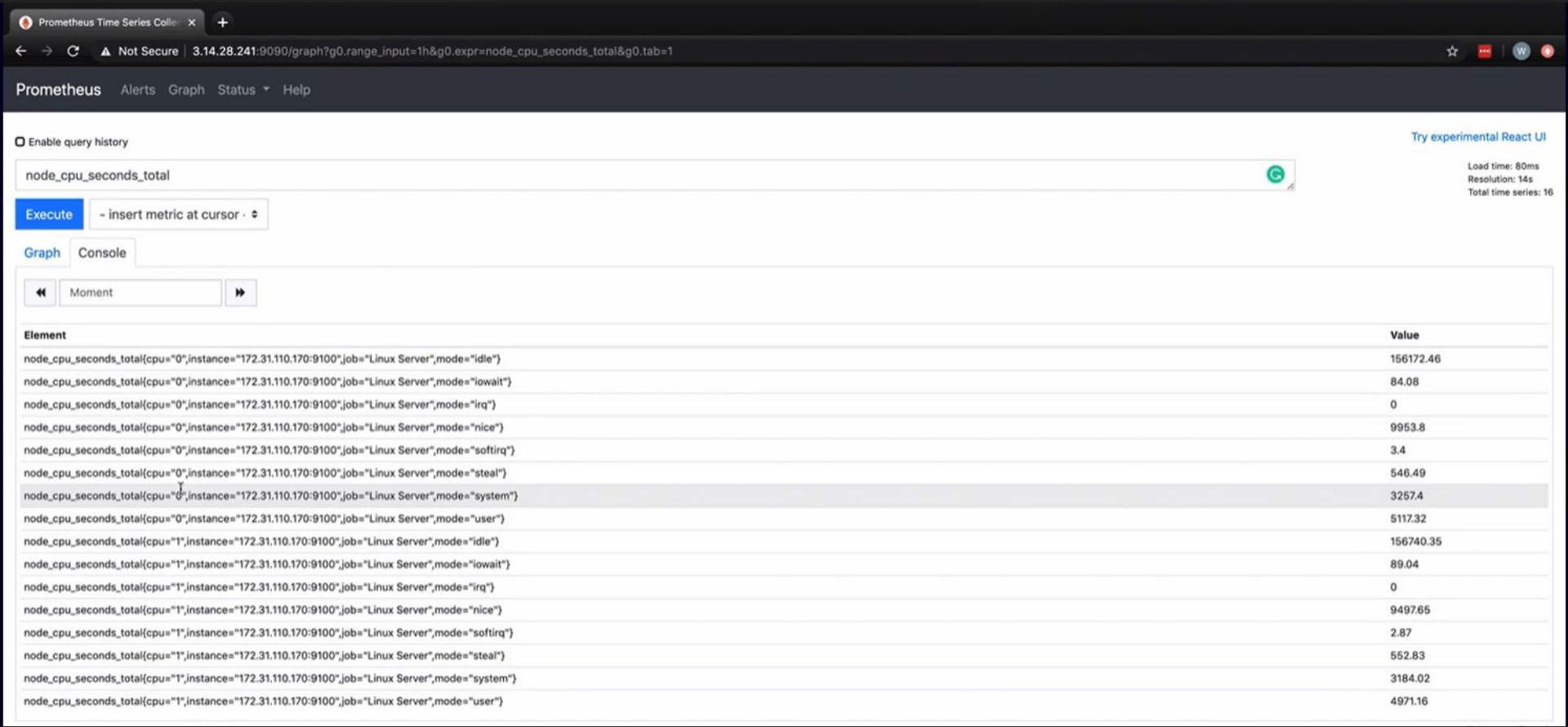
Both of the following are valid time-series selectors.

```
node_cpu_seconds_total
```

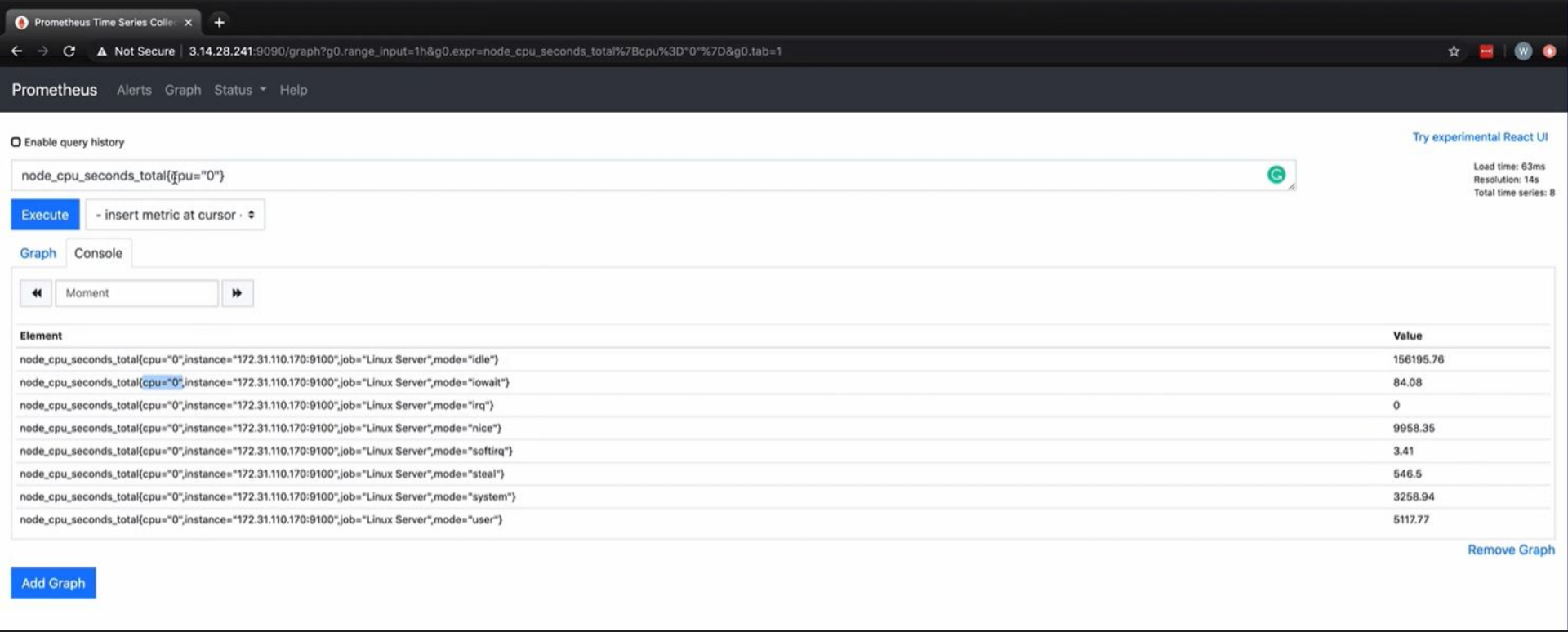
```
node_cpu_seconds_total{cpu="0"}
```

When multiple values exist for a label in the Prometheus data and you do not specify the label in your query, Prometheus will simply return data for all the values of that label.

# Query Basics



# Query Basics



# Query Basics

Label Matching

You can use a variety of operators to perform advanced matches based upon label values.

- =: Equals
- !=: Does not equal
- =~: Regex match
- !~: Does not regex match

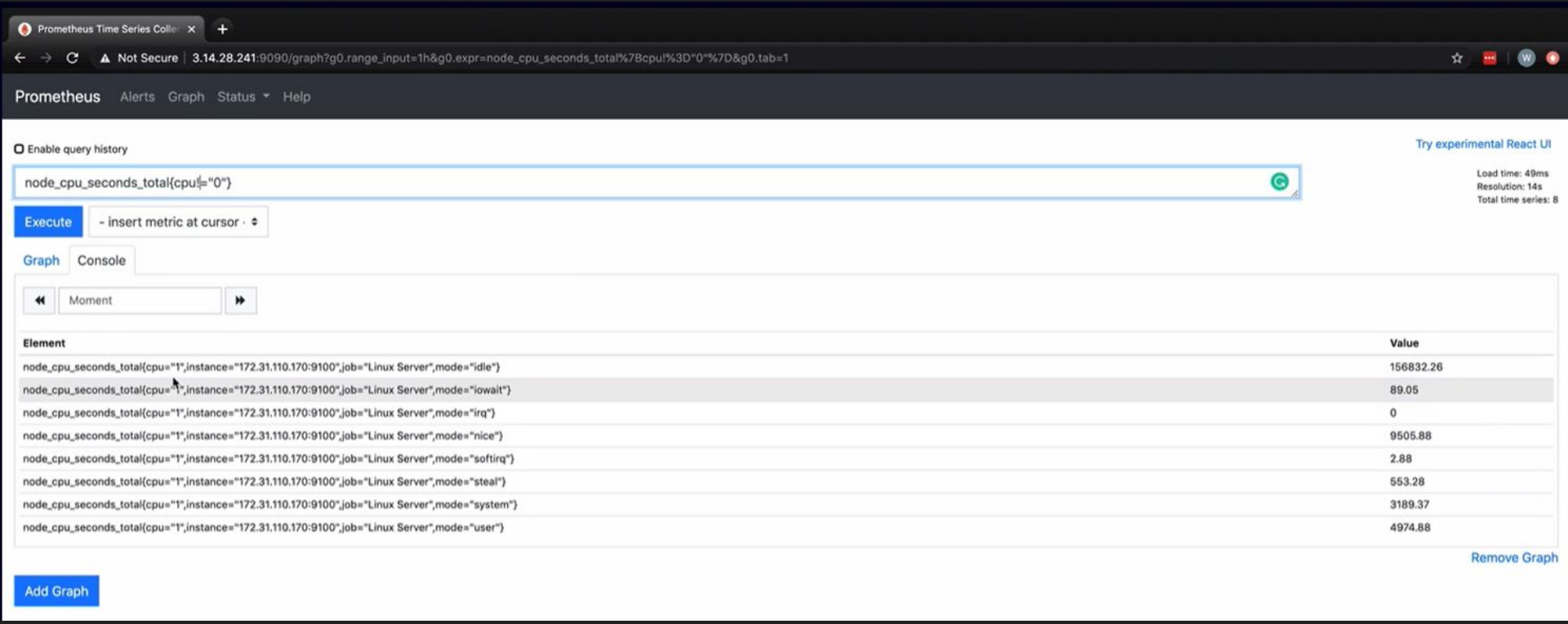
```
node_cpu_seconds_total{cpu="0"}
```

```
node_cpu_seconds_total{cpu!="0"}
```

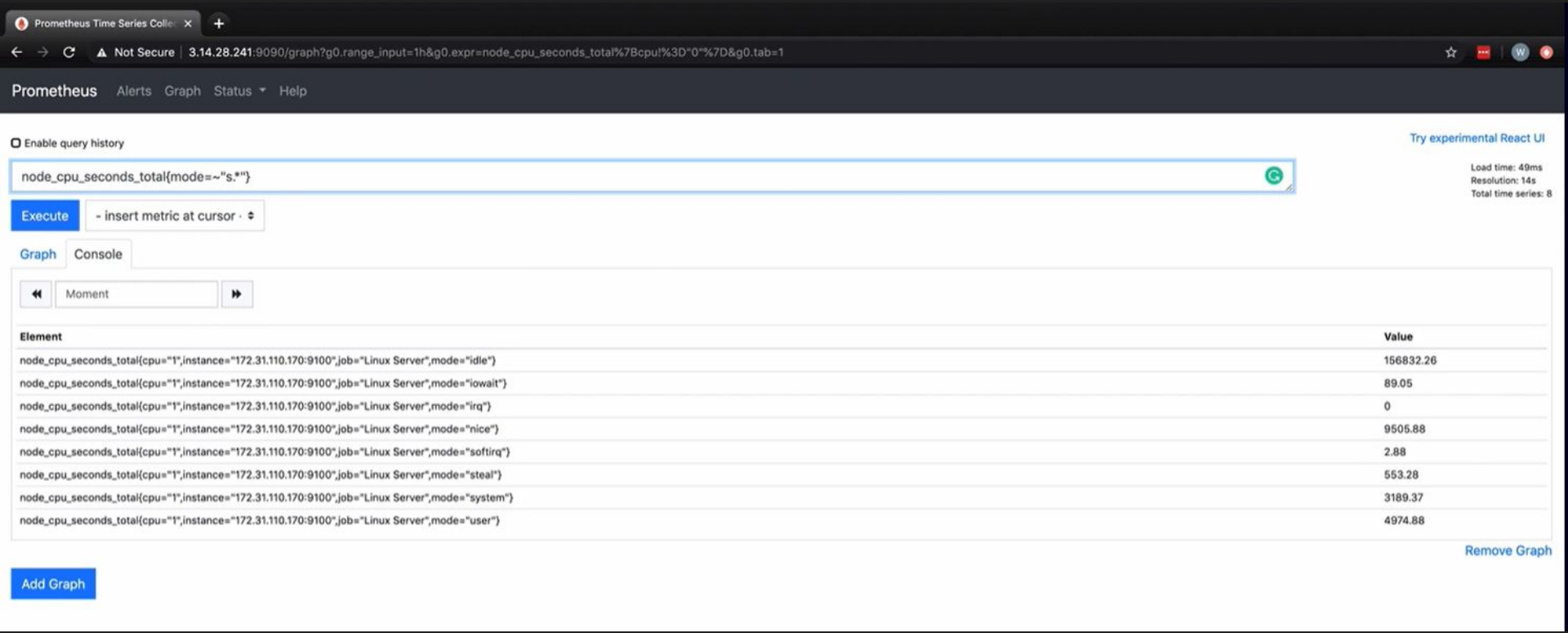
```
node_cpu_seconds_total{cpu=~"1*"}
```

```
node_cpu_seconds_total{cpu!~"1*"}
```

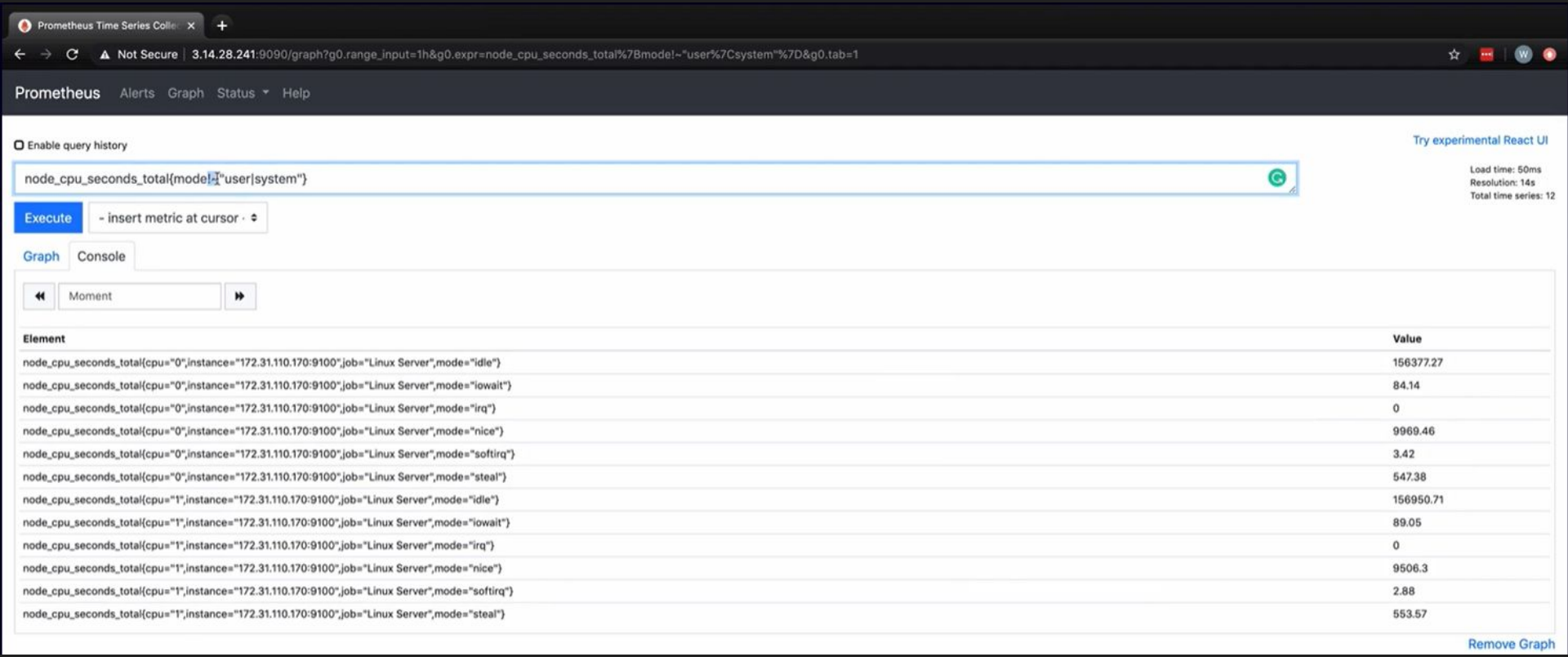
# Query Basics



# Query Basics



# Query Basics



# Query Basics

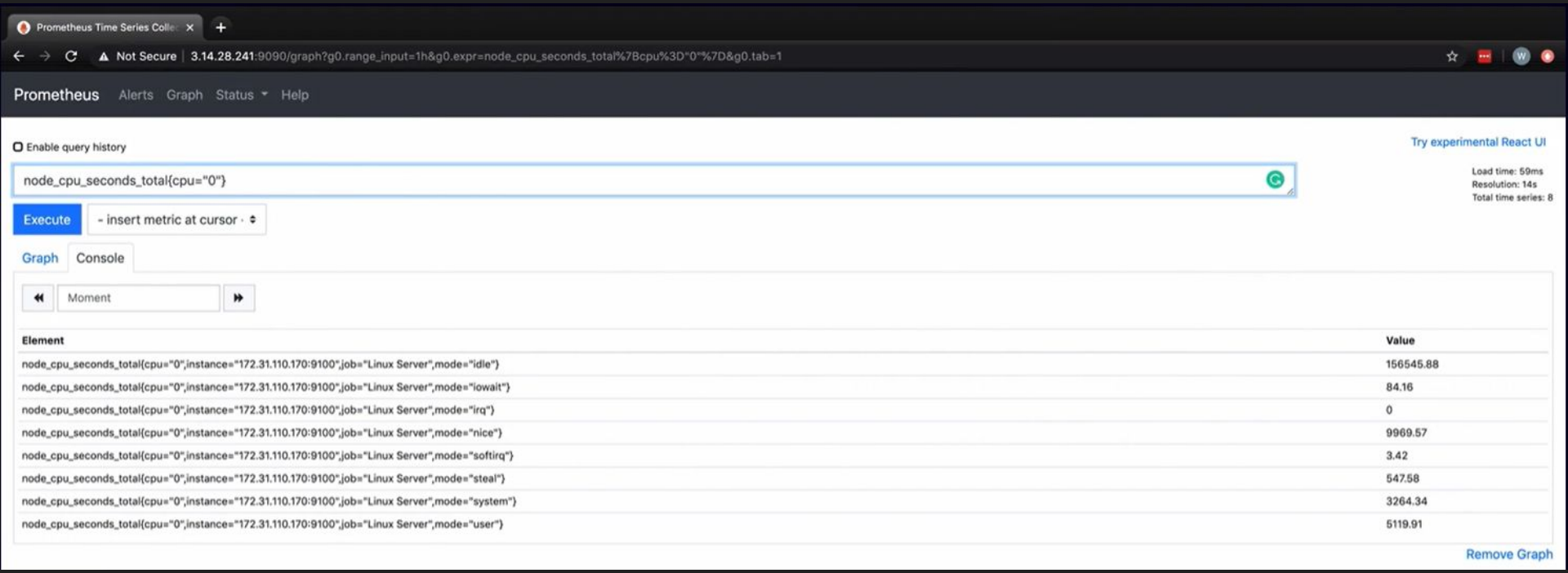
Range Vector Selectors

Since Prometheus data is fundamentally time-series data, you can select data points over a particular time range.

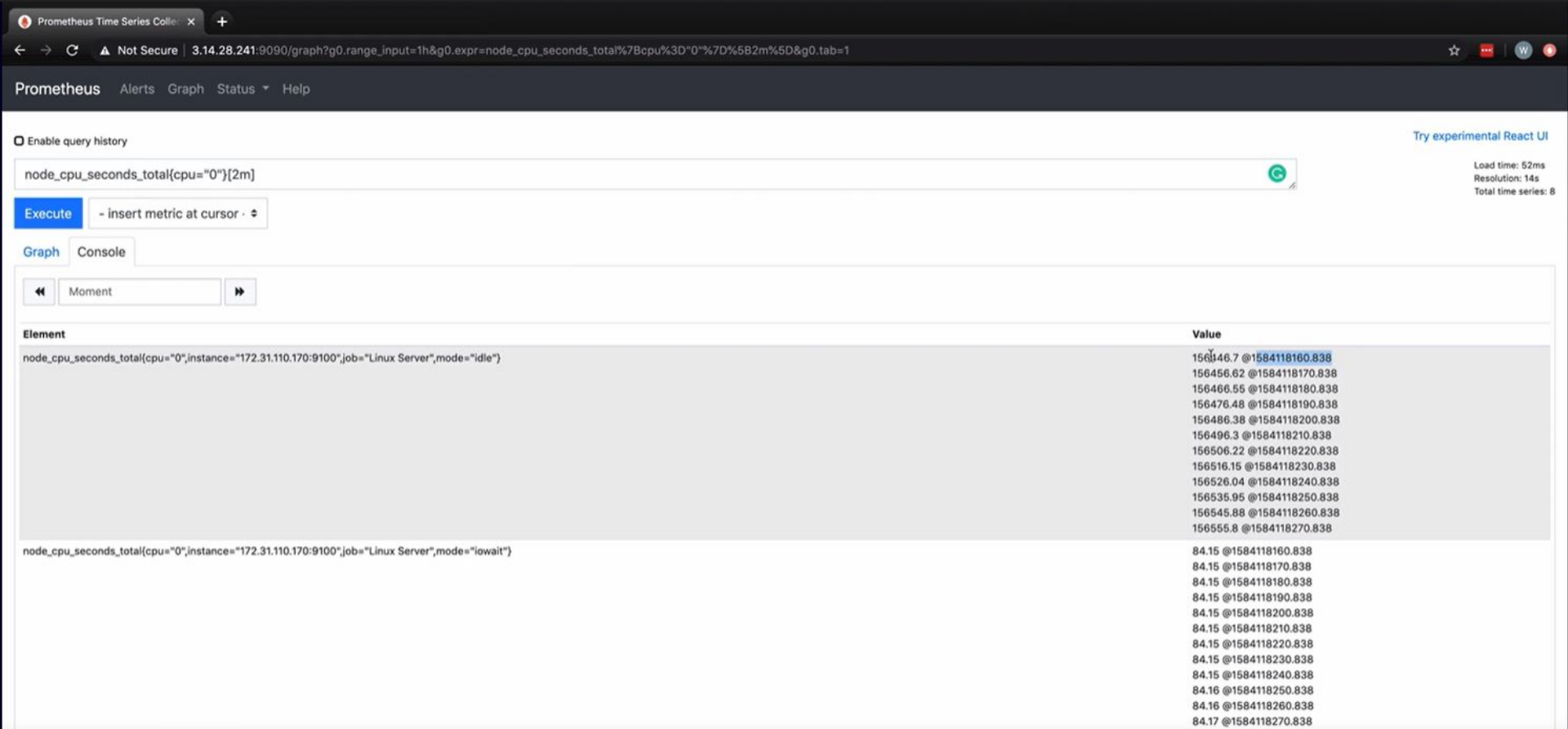
This query selects all values for the metric name and label recorded over the last two minutes.

```
node_cpu_seconds_total{cpu="0"}[2m]
```

# Query Basics



# Query Basics



# Query Basics

## Offset Modifier

You can use an offset modifier to provide a time offset to select data from the past, with or without a range selector.

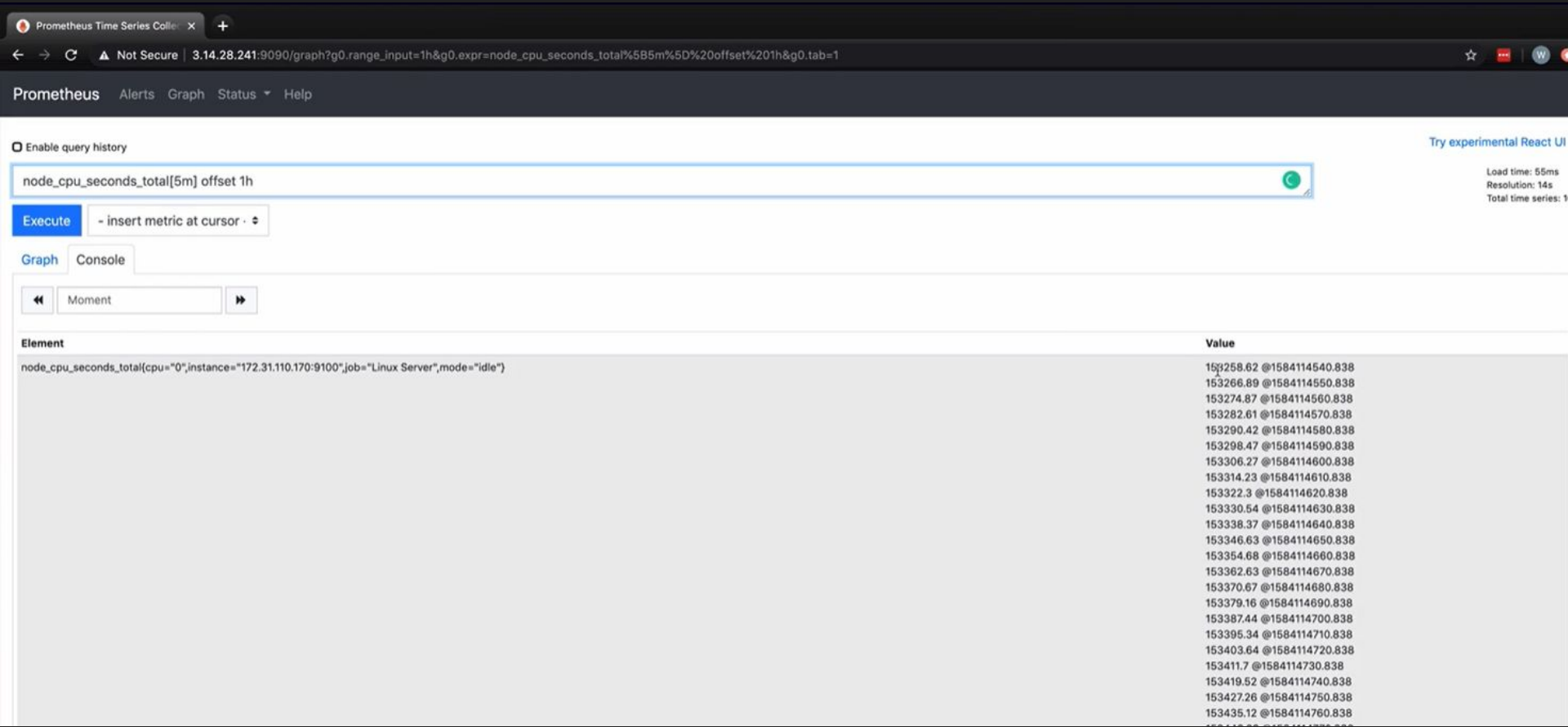
Select the CPU usage from one hour ago:

```
node_cpu_seconds_total offset 1h
```

Select CPU usage values over a five-minute period one hour ago:

```
node_cpu_seconds_total[5m] offset 1h
```

# Query Basics



# Query Operators

## Operators

Operators allow you to perform calculations based upon your metric data.

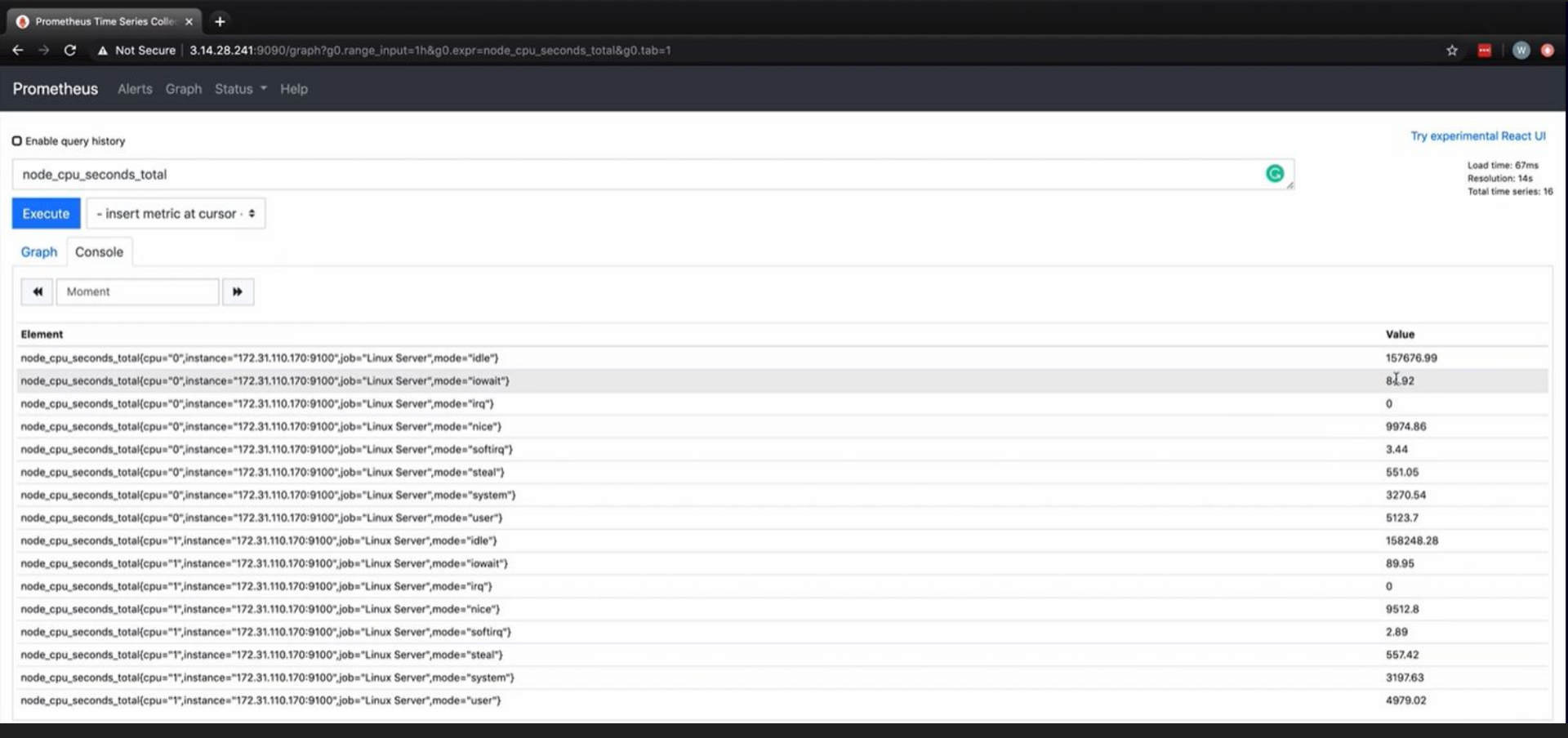
## Arithmetic Binary Operators

The following operators perform basic arithmetic on numeric values.

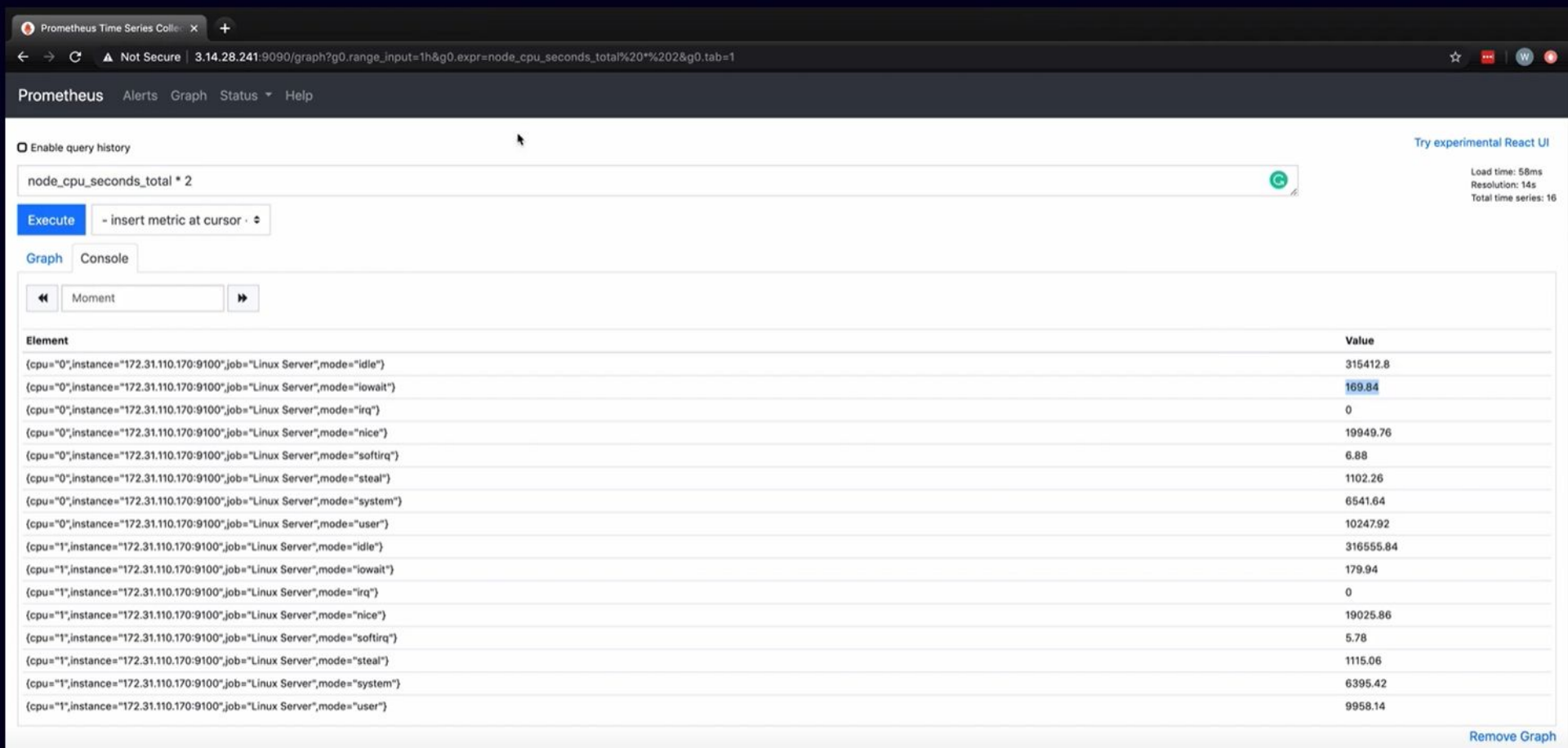
- +: Addition
- -: Subtraction
- *: Multiplication
- /: Division
- %: Modulo
- ^: Exponentiation

```
node_cpu_seconds_total * 2
```

# Query Operators



# Query Operators



# Query Operators

Matching Rules

When using operators, Prometheus uses matching rules to determine how to combine or compare records from two sets of data. By default, records only match if all of their labels match.

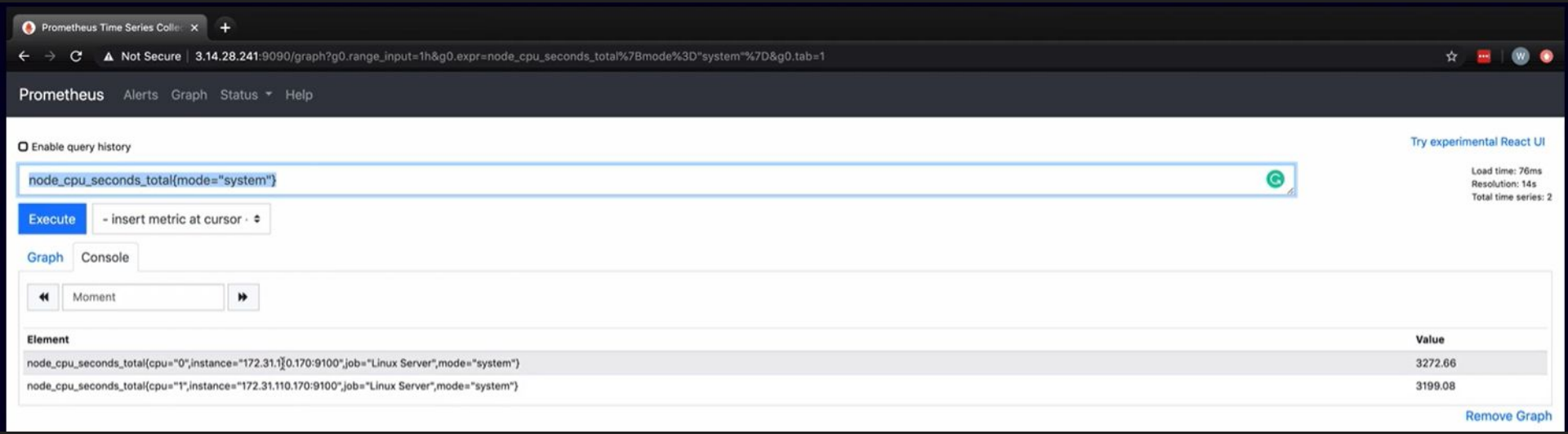
Use the following keywords to control matching behavior:

`ignoring(<label list>)` - Ignore the specified labels when matching.
`on(<label list>)` - Use only the specified labels when matching.

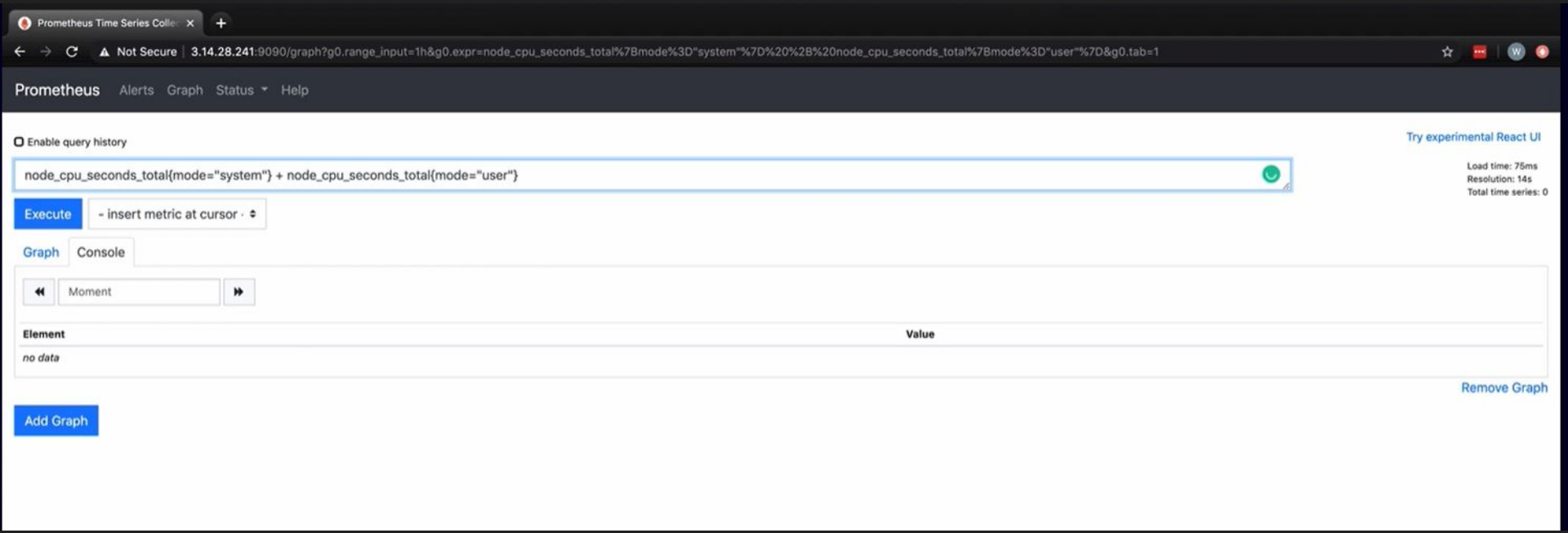
```
node_cpu_seconds_total{mode="system"} + ignoring(mode) node_cpu_seconds_total{mode="user"}
```

```
node_cpu_seconds_total{mode="system"} + on(cpu) node_cpu_seconds_total{mode="user"}
```

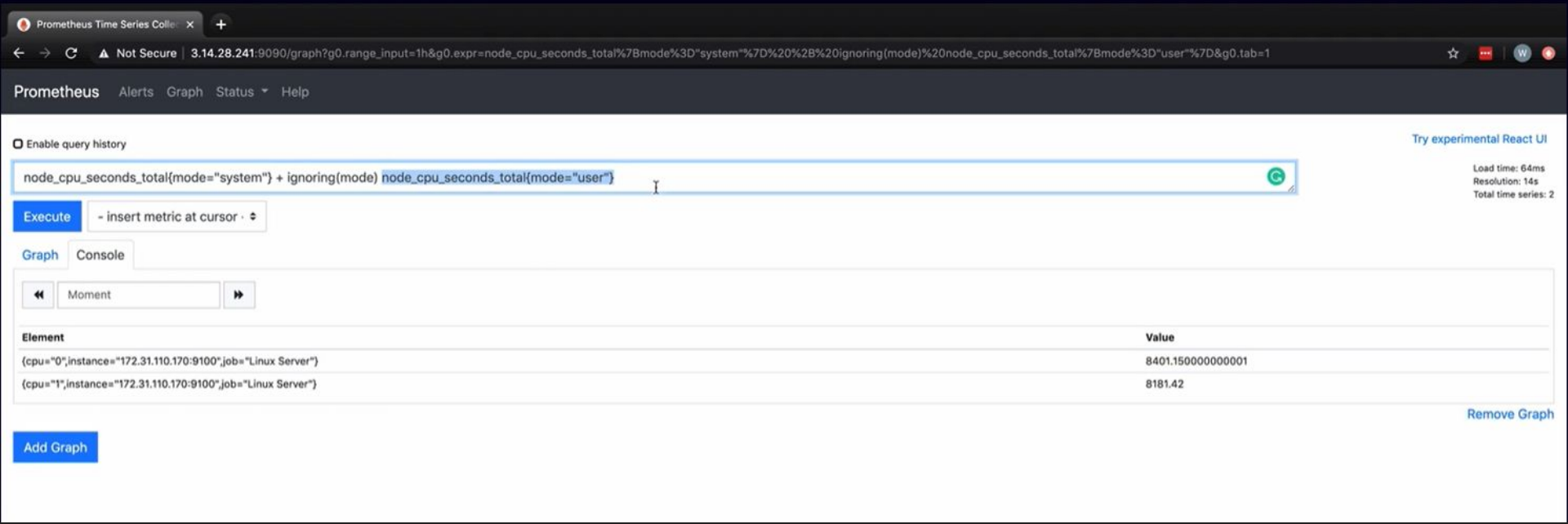
# Query Operators



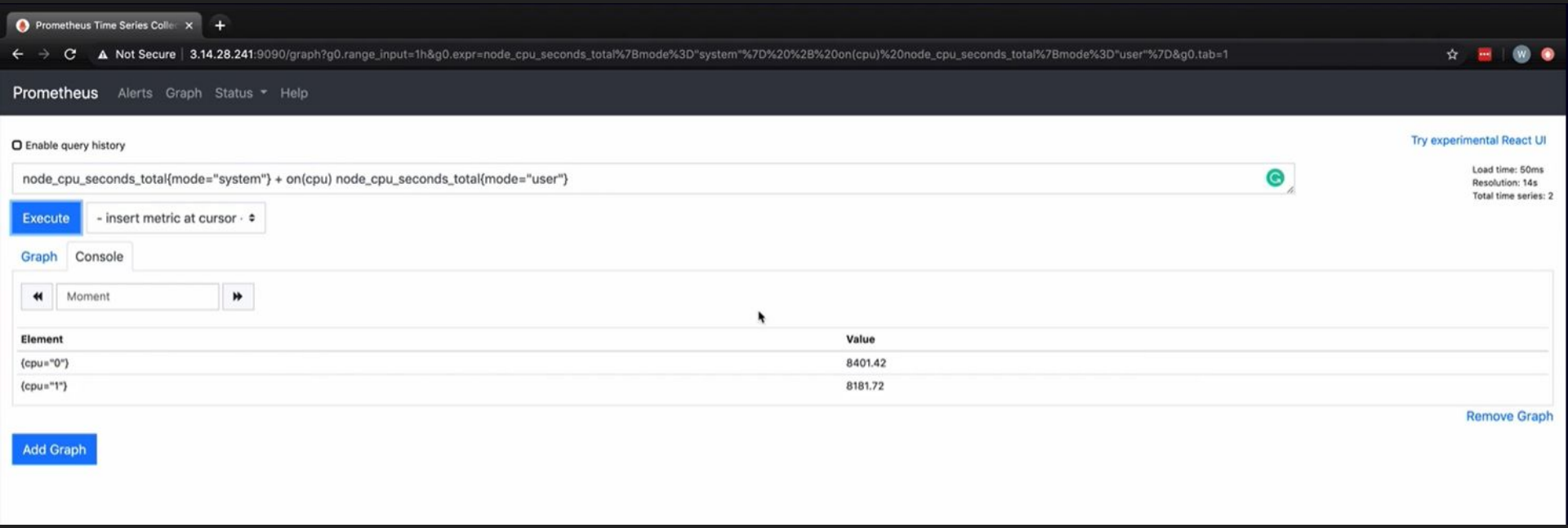
# Query Operators



# Query Operators



# Query Operators



# Query Operators

Comparison Binary Operators

By default, comparison operators filter results to only those where the comparison evaluates as true.

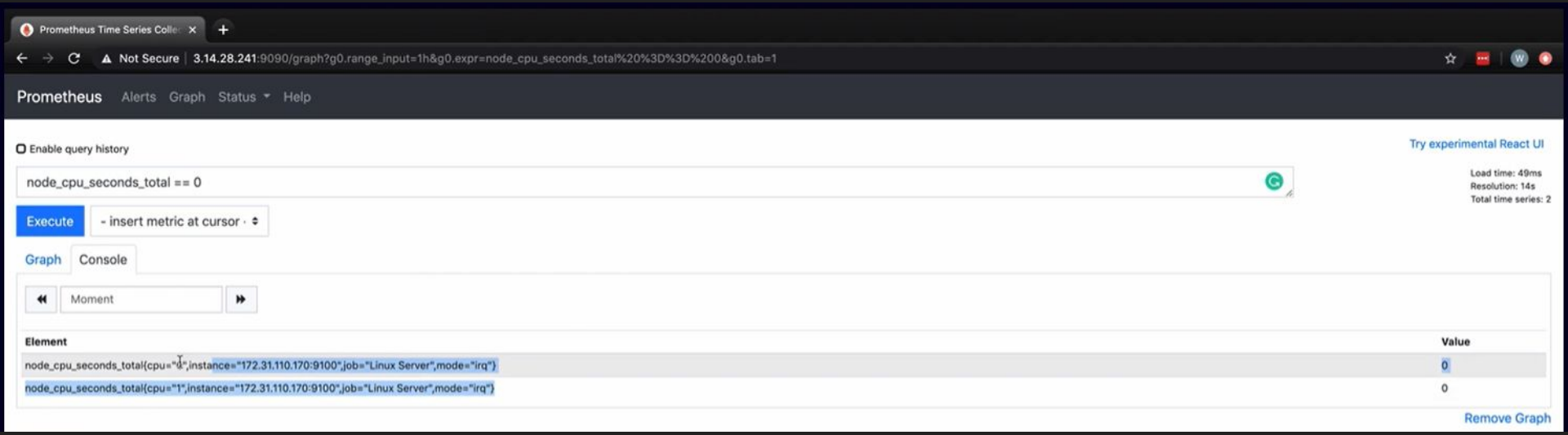
- `==`: Equal
- `!=`: Not equal
- `>`: Greater than
- `<`: Less than
- `>=`: Greater than or equal
- `<=`: Less than or equal

```
node_cpu_seconds_total == 0
```

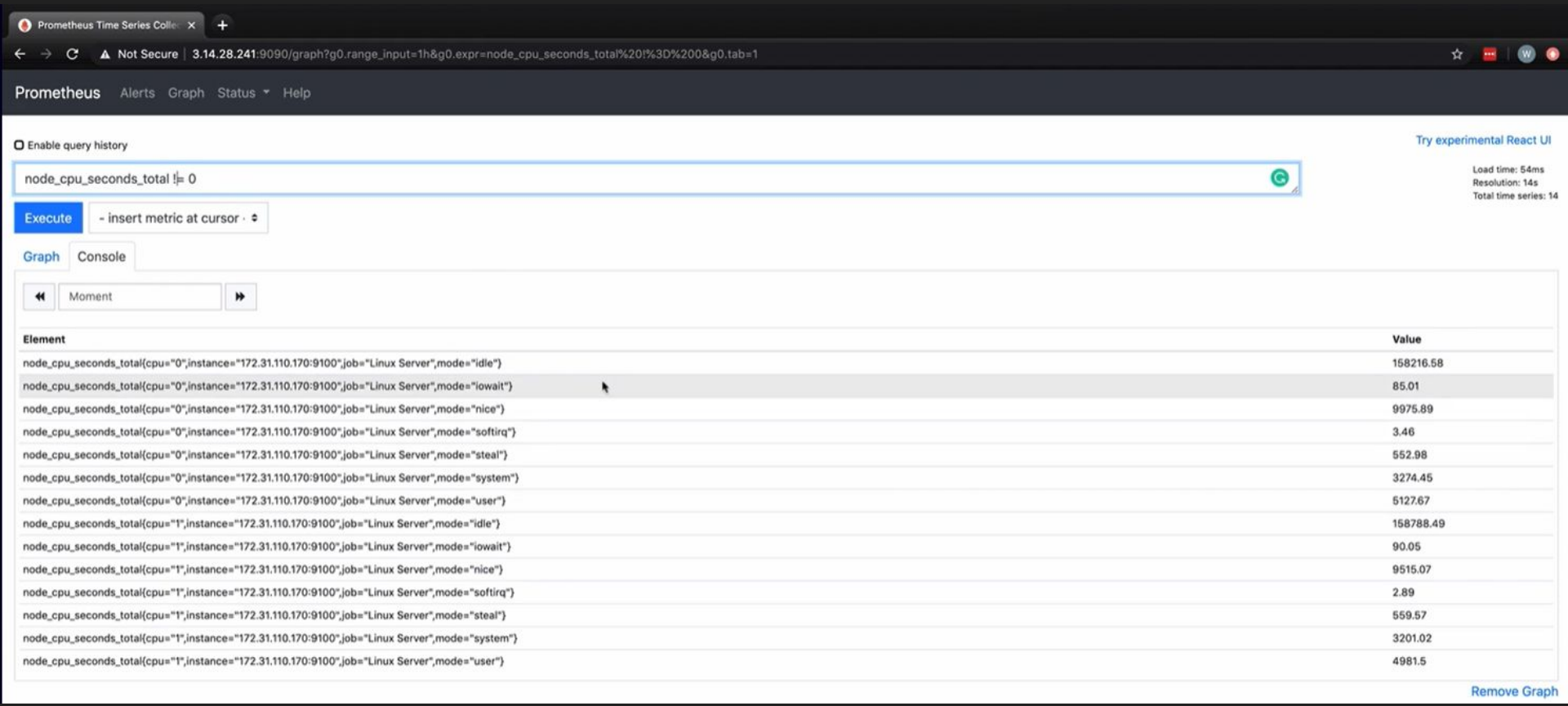
Use the `bool` modifier to instead return the boolean result of the comparison.

```
node_cpu_seconds_total == bool 0
```

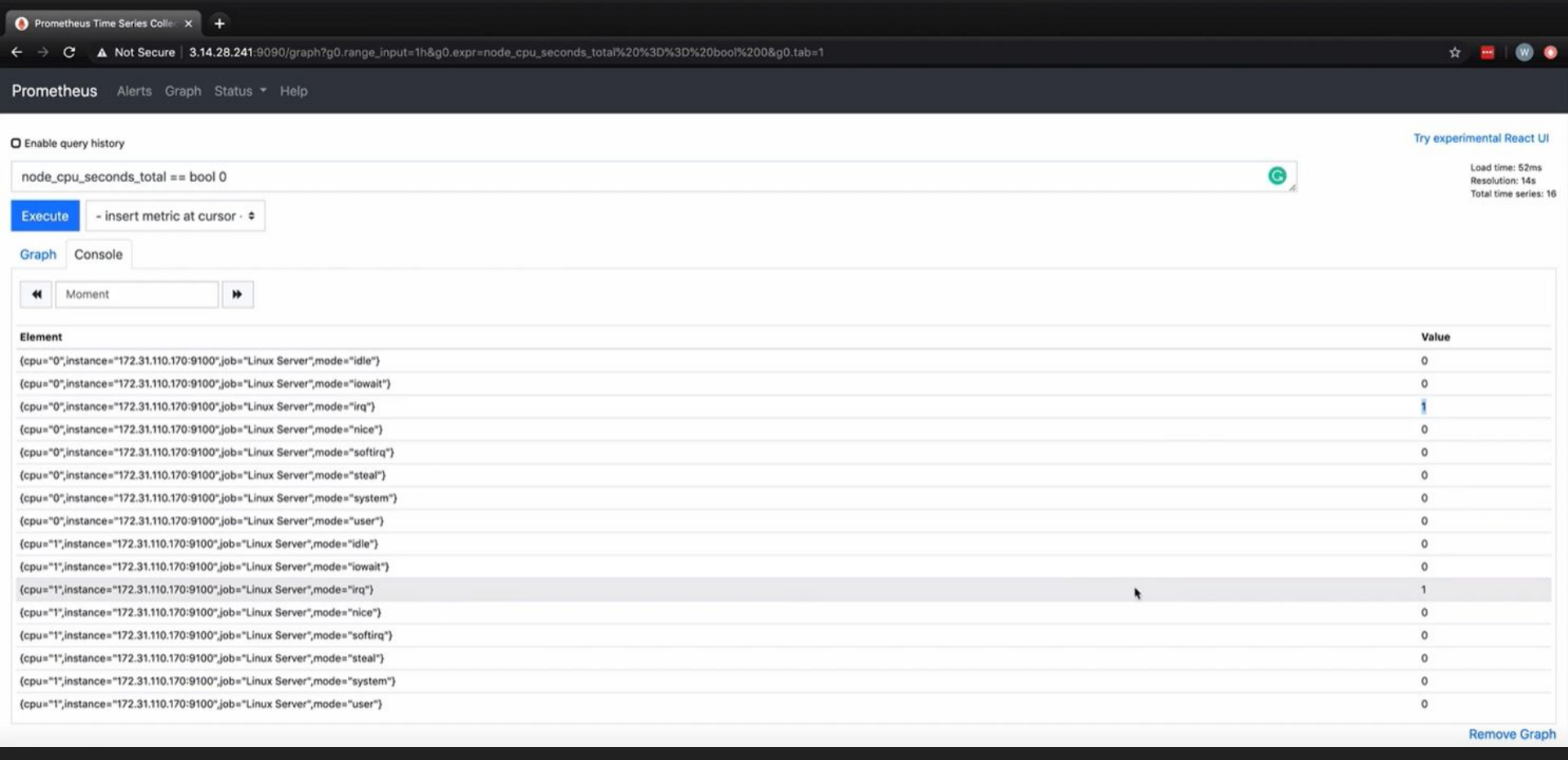
# Query Operators



# Query Operators



# Query Operators



# Query Operators

Logical/Set Binary Operators

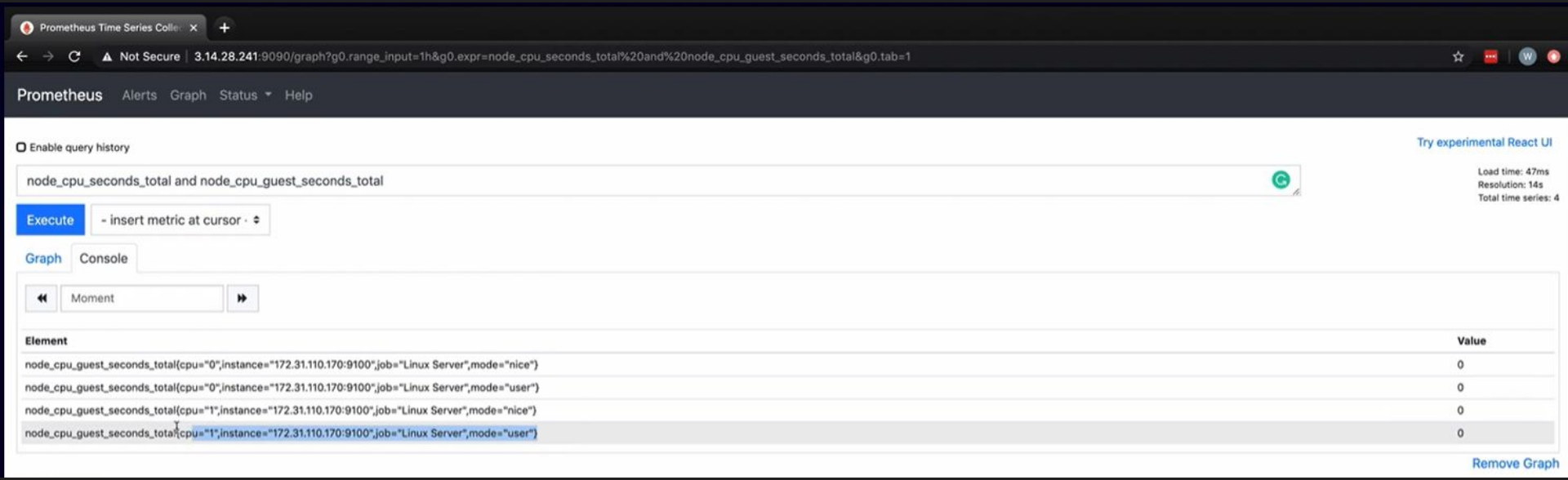
These operators combine sets of results in various ways.

 +- and: Intersection
 +- or: Union
 +- unless: Complement

These operators use labels to compare records. For example, and returns only records where the set of labels in one set of results is matched by labels in the other set.

```
node_cpu_seconds_total and node_cpu_guest_seconds_total
```

# Query Operators



# Query Operators

Aggregation Operators

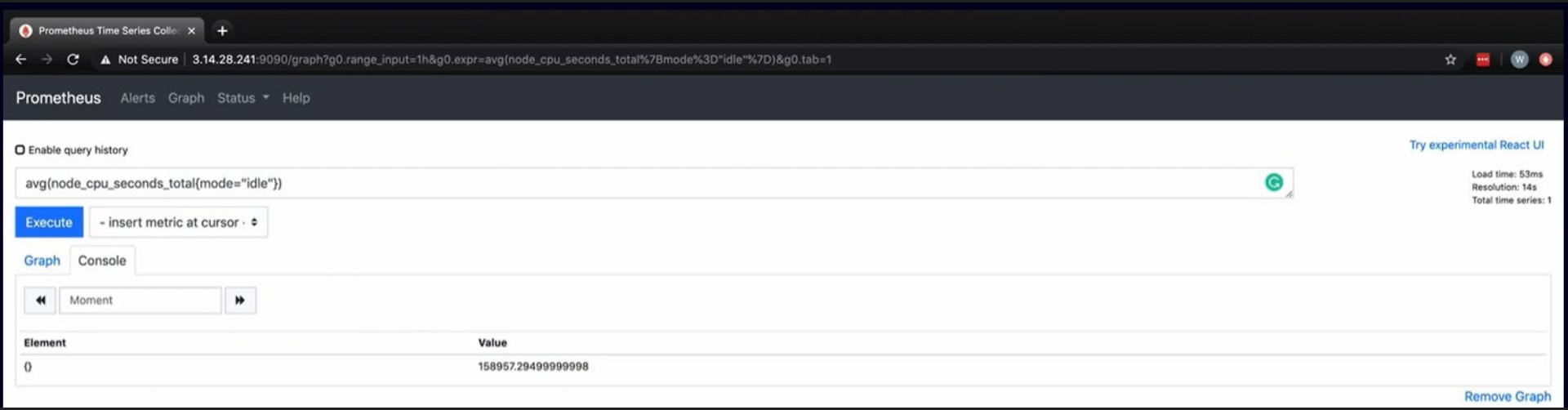
Aggregation operators combine multiple values into a single value.

- sum: Add all values together.
- min: Select the smallest value.
- max: Select the largest value.
- avg: Calculate the average of all values.
- stddev: Calculate population standard deviation over all values.
- stdvar: Calculate population standard variance over all values.
- count: Count number of values.
- count_values: Count the number of values with the same value.
- bottomk: Smallest number (k) of elements.
- topk: Largest number (k) of elements.
- quantile: Calculate the quantile for a particular dimension.

This query uses avg to get the average idle time between all CPUs.

```
avg(node_cpu_seconds_total{mode="idle"})
```

# Query Operators



# Query Functions

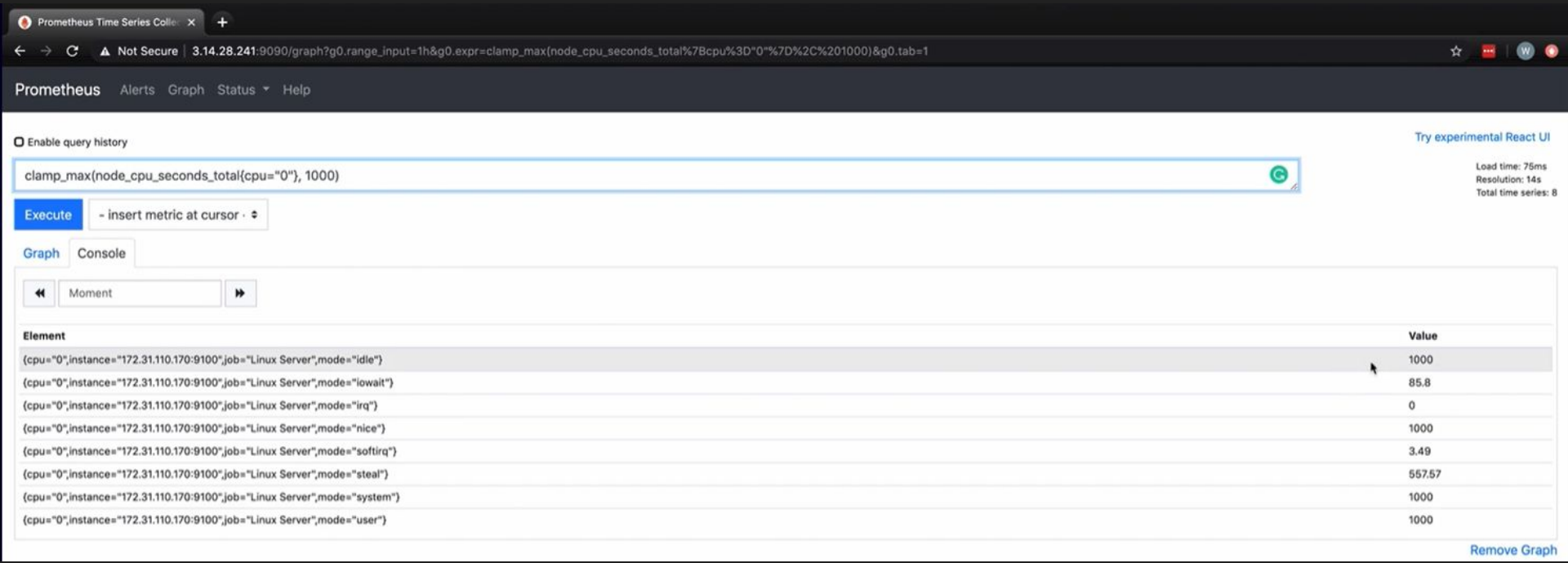
Functions

Functions provide a wide array of built-in functionality to aid in the process of writing queries.

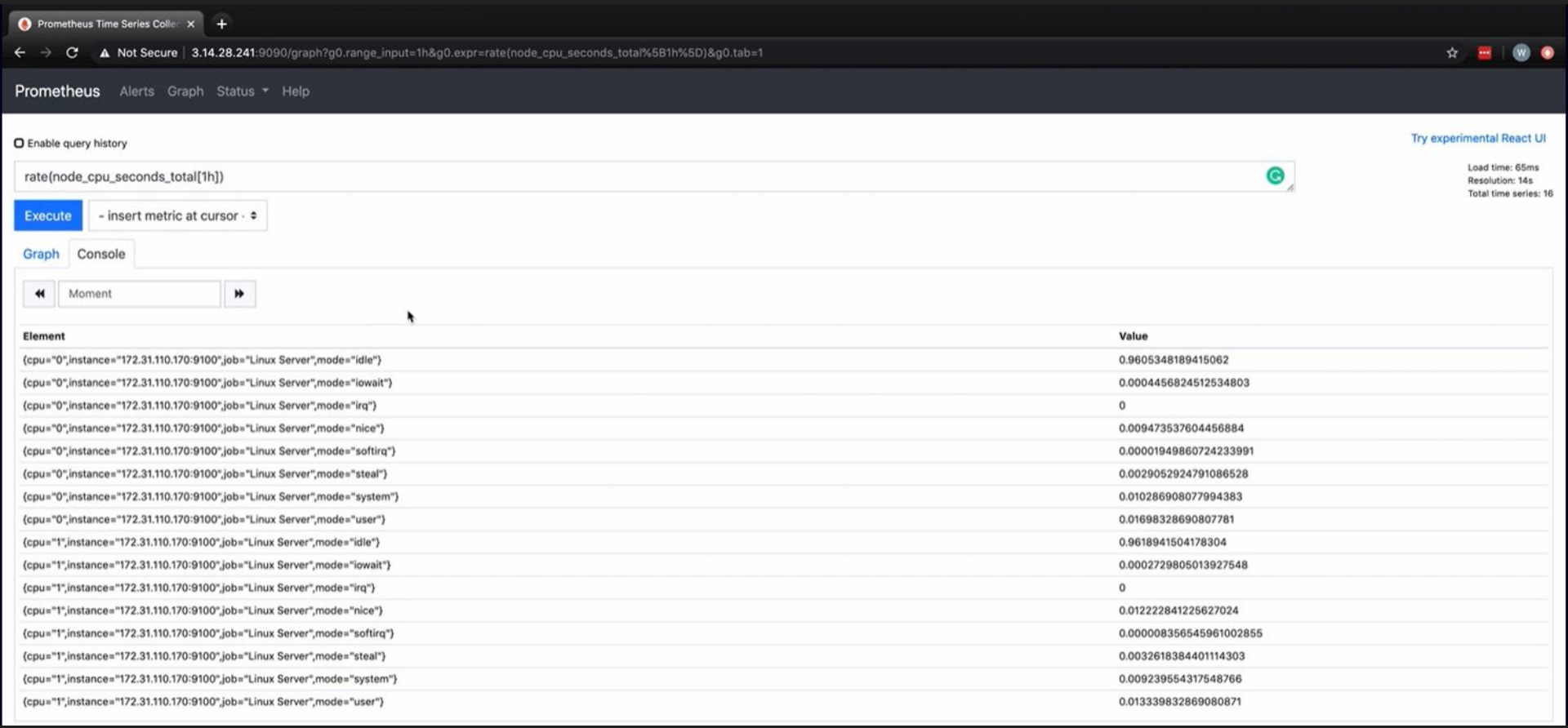
Consult the Prometheus documentation for a full list of available functions.

For example:
 +- `abs()`: Calculates absolute value
 +- `clamp_max()`: Returns values, but replaces them with a maximum value if they
 |  exceed that value
 +- `clamp_min()`: Returns values, but replaces them with a minimum value if they
    are less than that value

# Query Functions



# Query Functions



# Prometheus HTTP API

## HTTP API

Prometheus provides an HTTP API you can use to execute queries and obtain results using HTTP requests.

This API is a useful way to interact with Prometheus, especially if you are building your own custom tools that require access to Prometheus data.

# Prometheus HTTP API

```
cloud_user@wboyd1c:~$ curl localhost:9090/api/v1/query?query=node_cpu_seconds_total
{"status":"success","data":{"resultType":"vector","result":[{"metric":{"__name__":"node_cpu_seconds_total","cpu"
:"0","instance":"172.31.110.170:9100","job":"Linux Server","mode":"idle"},"value":[1584122849.099,"160952.75"]},
{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"172.31.110.170:9100","job":"Linux Server","
mode":"iowait"},"value":[1584122849.099,"85.93"]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","inst
ance":"172.31.110.170:9100","job":"Linux Server","mode":"irq"},"value":[1584122849.099,"0"]},{"metric":{"__name_
_":"node_cpu_seconds_total","cpu":"0","instance":"172.31.110.170:9100","job":"Linux Server","mode":"nice"},"valu
e":[1584122849.099,"10011.58"]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"172.31.110.
170:9100","job":"Linux Server","mode":"softirq"},"value":[1584122849.099,"3.5"]},{"metric":{"__name__":"node_cpu
_seconds_total","cpu":"0","instance":"172.31.110.170:9100","job":"Linux Server","mode":"steal"},"value":[1584122
849.099,"561.72"]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"172.31.110.170:9100","jo
b":"Linux Server","mode":"system"},"value":[1584122849.099,"3309.92"]},{"metric":{"__name__":"node_cpu_seconds_t
otal","cpu":"0","instance":"172.31.110.170:9100","job":"Linux Server","mode":"user"},"value":[1584122849.099,"51
92.94"]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"1","instance":"172.31.110.170:9100","job":"Linux
Server","mode":"idle"},"value":[1584122849.099,"161531.12"]},{"metric":{"__name__":"node_cpu_seconds_total","cpu
":"1","instance":"172.31.110.170:9100","job":"Linux Server","mode":"iowait"},"value":[1584122849.099,"91.3"]},{"
metric":{"__name__":"node_cpu_seconds_total","cpu":"1","instance":"172.31.110.170:9100","job":"Linux Server","mo
de":"irq"},"value":[1584122849.099,"0"]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"1","instance":"17
2.31.110.170:9100","job":"Linux Server","mode":"nice"},"value":[1584122849.099,"9563.83"]},{"metric":{"__name__"
:"node_cpu_seconds_total","cpu":"1","instance":"172.31.110.170:9100","job":"Linux Server","mode":"softirq"},"val
ue":[1584122849.099,"2.92"]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"1","instance":"172.31.110.170
:9100","job":"Linux Server","mode":"steal"},"value":[1584122849.099,"569.27"]},{"metric":{"__name__":"node_cpu_s
econds_total","cpu":"1","instance":"172.31.110.170:9100","job":"Linux Server","mode":"system"},"value":[15841228
49.099,"3230.92"]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"1","instance":"172.31.110.170:9100","jo
b":"Linux Server","mode":"user"},"value":[1584122849.099,"5029.66"]}]}}cloud_user@wboyd1c:~$
```

# Prometheus HTTP API

```
cloud_user@wboyd1c:~$ curl localhost:9090/api/v1/query --data-urlencode "query=node_cpu_seconds_total{cpu=\"0\"}
"
{"status":"success","data":{"resultType":"vector","result":[{"metric":{"__name__":"node_cpu_seconds_total","cpu"
:"0","instance":"172.31.110.170:9100","job":"Linux Server","mode":"idle"},"value":[1584122990.425,"161087.98"]},
{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"172.31.110.170:9100","job":"Linux Server","
mode":"iowait"},"value":[1584122990.425,"85.94"]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","inst
ance":"172.31.110.170:9100","job":"Linux Server","mode":"irq"},"value":[1584122990.425,"0"]},{"metric":{"__name_
_":"node_cpu_seconds_total","cpu":"0","instance":"172.31.110.170:9100","job":"Linux Server","mode":"nice"},"valu
e":[1584122990.425,"10013.84"]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"172.31.110.
170:9100","job":"Linux Server","mode":"softirq"},"value":[1584122990.425,"3.5"]},{"metric":{"__name__":"node_cpu
_seconds_total","cpu":"0","instance":"172.31.110.170:9100","job":"Linux Server","mode":"steal"},"value":[1584122
990.425,"562.92"]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"172.31.110.170:9100","jo
b":"Linux Server","mode":"system"},"value":[1584122990.425,"3310.24"]},{"metric":{"__name__":"node_cpu_seconds_t
otal","cpu":"0","instance":"172.31.110.170:9100","job":"Linux Server","mode":"user"},"value":[1584122990.425,"51
94.69"]}]}}cloud_user@wboyd1c:~$
```

# Prometheus HTTP API

```
cloud_user@wboyd1c:~$ start=$(date --date '-5 min' +'%Y-%m-%dT%H:%M:%SZ')
cloud_user@wboyd1c:~$ end=$(date +'%Y-%m-%dT%H:%M:%SZ')
cloud_user@wboyd1c:~$ curl "localhost:9090/api/v1/query_range?query=node_cpu_seconds_total&start=$start&end=$end
&step=1m"
{"status":"success","data":{"resultType":"matrix","result":[{"metric":{"__name__":"node_cpu_seconds_total","cpu"
:"0","instance":"172.31.110.170:9100","job":"Linux Server","mode":"idle"},"values":[[1584122783,"160894.5"],[158
4122843,"160952.75"],[1584122903,"161010.58"],[1584122963,"161068.83"],[1584123023,"161126.79"],[1584123083,"161
184.48"]]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"172.31.110.170:9100","job":"Linu
x Server","mode":"iowait"},"values":[[1584122783,"85.93"],[1584122843,"85.93"],[1584122903,"85.93"],[1584122963,
"85.94"],[1584123023,"85.95"],[1584123083,"85.95"]]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","i
nstance":"172.31.110.170:9100","job":"Linux Server","mode":"irq"},"values":[[1584122783,"0"],[1584122843,"0"],[1
584122903,"0"],[1584122963,"0"],[1584123023,"0"],[1584123083,"0"]]},{"metric":{"__name__":"node_cpu_seconds_tota
l","cpu":"0","instance":"172.31.110.170:9100","job":"Linux Server","mode":"nice"},"values":[[1584122783,"10011"]
,[1584122843,"10011.58"],[1584122903,"10012.75"],[1584122963,"10013.32"],[1584123023,"10014.25"],[1584123083,"10
015.58"]]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"172.31.110.170:9100","job":"Linu
x Server","mode":"softirq"},"values":[[1584122783,"3.5"],[1584122843,"3.5"],[1584122903,"3.5"],[1584122963,"3.5"
],[1584123023,"3.5"],[1584123083,"3.5"]]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"1
72.31.110.170:9100","job":"Linux Server","mode":"steal"},"values":[[1584122783,"561.27"],[1584122843,"561.72"],[
1584122903,"562.21"],[1584122963,"562.72"],[1584123023,"563.21"],[1584123083,"563.75"]]},{"metric":{"__name__":"
node_cpu_seconds_total","cpu":"0","instance":"172.31.110.170:9100","job":"Linux Server","mode":"system"},"values
":[[1584122783,"3309.77"],[1584122843,"3309.92"],[1584122903,"3310.06"],[1584122963,"3310.2"],[1584123023,"3310.
36"],[1584123083,"3310.52"]]},{"metric":{"__name__":"node_cpu_seconds_total","cpu":"0","instance":"172.31.110.17
0:9100","job":"Linux Server","mode":"user"},"values":[[1584122783,"5192.04"],[1584122843,"5192.94"],[1584122903,
```

# Thank You!