Query Loki with Python

- Prerequisites
- Authentication
  - Multi-tenant mode
  - Grafana Cloud
  - Self-signed certificates
- Query logs within a range of time
  - Using requests
  - Using httpx
- Query logs at a single point in time
- Push logs
- Query labels and label values
- Handling errors
- Common problems

This page provides Python examples for the most common Loki HTTP API operations: querying logs, pushing log entries, and listing labels. For the full API reference including all parameters and response formats, see the Loki HTTP API documentation.

Prerequisites

Install the requests library:

```
pip install requests
```

If you need an async-capable client, httpx provides a nearly identical API:

```
pip install httpx
```

Authentication

The examples on this page connect to a local Loki instance without authentication. To use these examples with a multi-tenant or Grafana Cloud deployment, add the appropriate authentication as shown below. All functions defined on this page accept `headers`, `auth`, and `verify` parameters for this purpose.

Multi-tenant mode

If your cluster has multi-tenancy enabled, pass the tenant ID in the `X-Scope-OrgID` header:

```
headers = {"X-Scope-OrgID": "my-tenant"}
resp = requests.get(url, params=params, headers=headers)
```

Using the functions defined on this page:

```
results = query_range(
    url="http://localhost:3100",
    query='{job="varlogs"}',
    start=datetime.now() - timedelta(hours=1),
    end=datetime.now(),
    headers={"X-Scope-OrgID": "my-tenant"},
)
```

To query across multiple tenants, separate tenant names with the pipe (`|`) character:

```
headers = {"X-Scope-OrgID": "tenant1|tenant2|tenant3"}
```

Grafana Cloud

For Grafana Cloud, use basic authentication with your Grafana Cloud user and an API token:

```
resp = requests.get(url, params=params, auth=("<user>", "<API_TOKEN>"))
```

Using the functions defined on this page:

```
results = query_range(
    url="https://logs-prod-us-central1.grafana.net",
```

```
    query='{job="varlogs"}',
    start=datetime.now() - timedelta(hours=1),
    end=datetime.now(),
    auth=("<user>", "<API_TOKEN>"),
)
```

You can find the **User** and **URL** values in the Loki logging service details of your Grafana Cloud stack.

Self-signed certificates

If your Loki instance uses a self-signed TLS certificate, you can disable certificate verification:

```
resp = requests.get(url, params=params, verify=False)
```

For production use, pass the path to your CA bundle instead:

```
resp = requests.get(url, params=params, verify="/path/to/ca-bundle.crt")
```

Using the functions defined on this page:

```
results = query_range(
    url="https://loki.internal:3100",
    query='{job="varlogs"}',
    start=datetime.now() - timedelta(hours=1),
    end=datetime.now(),
    verify="/path/to/ca-bundle.crt",
)
```

Query logs within a range of time

`GET /loki/api/v1/query_range` queries logs over a time range. This is the most common query operation.

Using requests

```
import requests
from datetime import datetime, timedelta



def query_range(
    url: str,
    query: str,
    start: datetime,
    end: datetime,
    limit: int = 1000,
    headers: dict[str, str] | None = None,
    auth: tuple[str, str] | None = None,
    verify: bool | str = True,  # False to skip TLS, or path to CA bundle
) -> list:
    """Query Loki for log entries within a time range."""
    resp = requests.get(
        f"{url}/loki/api/v1/query_range",
        params={
            "query": query,
            "start": str(int(start.timestamp() * 1e9)),  # nanoseconds
            "end": str(int(end.timestamp() * 1e9)),
            "limit": limit,
            "direction": "backward",
        },
        headers=headers,
        auth=auth,
        verify=verify,
    )
    resp.raise_for_status()
    return resp.json()["data"]["result"]



results = query_range(
    url="http://localhost:3100",
    query='{job="varlogs"} |= "error"',
    start=datetime.now() - timedelta(hours=1),
    end=datetime.now(),
)
```

```
for stream in results:
    print(f"Labels: {stream['stream']}")
    for ts, line in stream["values"]:
        print(f"  {datetime.fromtimestamp(int(ts) / 1e9)}: {line}")
```

Using httpx

```
import httpx
from datetime import datetime, timedelta



def query_range(
    url: str,
    query: str,
    start: datetime,
    end: datetime,
    limit: int = 1000,
    headers: dict[str, str] | None = None,
    auth: tuple[str, str] | None = None,
    verify: bool | str = True,  # False to skip TLS, or path to CA bundle; httpx also accepts
ssl.SSLContext
) -> list:
    """Query Loki for log entries within a time range."""
    resp = httpx.get(
        f"{url}/loki/api/v1/query_range",
        params={
            "query": query,
            "start": str(int(start.timestamp() * 1e9)),  # nanoseconds
            "end": str(int(end.timestamp() * 1e9)),
            "limit": limit,
            "direction": "backward",
        },
        headers=headers,
        auth=auth,
        verify=verify,
    )
    resp.raise_for_status()
    return resp.json()["data"]["result"]
```

```
results = query_range(
    url="http://localhost:3100",
    query='{job="varlogs"} |= "error"',
    start=datetime.now() - timedelta(hours=1),
    end=datetime.now(),
)

for stream in results:
    print(f"Labels: {stream['stream']}")
    for ts, line in stream["values"]:
        print(f"  {datetime.fromtimestamp(int(ts) / 1e9)}: {line}")
```

Query logs at a single point in time

`GET /loki/api/v1/query` evaluates a query at a single point in time. Use this for instant metric queries such as aggregations with `rate()`, `count_over_time()`, or `bytes_over_time()`. Log stream selectors (queries that return log lines) are not supported as instant queries; use `query_range` instead.

```
import requests
from datetime import datetime



def query_instant(
    url: str,
    query: str,
    headers: dict[str, str] | None = None,
    auth: tuple[str, str] | None = None,
    verify: bool | str = True,  # False to skip TLS, or path to CA bundle
) -> list:
    """Run an instant metric query against Loki."""
    resp = requests.get(
        f"{url}/loki/api/v1/query",
        params={
            "query": query,
            "time": str(int(datetime.now().timestamp() * 1e9)),
```

```
        },
        headers=headers,
        auth=auth,
        verify=verify,
    )
    resp.raise_for_status()
    return resp.json()["data"]["result"]


results = query_instant(
    url="http://localhost:3100",
    query='sum(rate({job="varlogs"}[10m])) by (level)',
)

for entry in results:
    print(f"{entry['metric']}: {entry['value'][1]}")
```

Push logs

`POST /loki/api/v1/push` sends log entries to Loki using the JSON format.

```
import json
import time
import requests


def push_logs(
    url: str,
    labels: dict[str, str],
    entries: list[tuple[str, str]],
    headers: dict[str, str] | None = None,
    auth: tuple[str, str] | None = None,
    verify: bool | str = True,  # False to skip TLS, or path to CA bundle
) -> None:
    """Push log entries to Loki.

    Args:
```

```
        url: Loki base URL.
        labels: Stream labels, for example {"job": "myapp", "env": "dev"}.
        entries: List of (timestamp_ns, log_line) tuples. Use
                 str(int(time.time() * 1e9)) to get a nanosecond timestamp.
    """
    payload = {
        "streams": [
            {
                "stream": labels,
                "values": [list(e) for e in entries],
            }
        ]
    }
    req_headers = {**(headers or {}), "Content-Type": "application/json"}
    resp = requests.post(
        f"{url}/loki/api/v1/push",
        headers=req_headers,
        data=json.dumps(payload),
        auth=auth,
        verify=verify,
    )
    resp.raise_for_status()


now_ns = str(int(time.time() * 1e9))
push_logs(
    url="http://localhost:3100",
    labels={"job": "myapp", "env": "dev"},
    entries=[
        (now_ns, "application started"),
        (now_ns, "listening on port 8080"),
    ],
)
```

Query labels and label values

`GET /loki/api/v1/labels` returns the list of known label names. `GET /loki/api/v1/label/<name>/values` returns the values for a specific label.

```
import requests


def get_labels(
    url: str,
    headers: dict[str, str] | None = None,
    auth: tuple[str, str] | None = None,
    verify: bool | str = True,  # False to skip TLS, or path to CA bundle
) -> list[str]:
    """List all known label names."""
    resp = requests.get(
        f"{url}/loki/api/v1/labels",
        headers=headers,
        auth=auth,
        verify=verify,
    )
    resp.raise_for_status()
    return resp.json()["data"]


def get_label_values(
    url: str,
    label: str,
    headers: dict[str, str] | None = None,
    auth: tuple[str, str] | None = None,
    verify: bool | str = True,  # False to skip TLS, or path to CA bundle
) -> list[str]:
    """List values for a specific label."""
    resp = requests.get(
        f"{url}/loki/api/v1/label/{label}/values",
        headers=headers,
        auth=auth,
        verify=verify,
    )
    resp.raise_for_status()
    return resp.json()["data"]


labels = get_labels("http://localhost:3100")
```

```
print(f"Labels: {labels}")

for label in labels:
    values = get_label_values("http://localhost:3100", label)
    print(f"  {label}: {values}")
```

Handling errors

Loki returns standard HTTP status codes. Common errors include:

Status	Meaning	Typical cause
400	Bad Request	Invalid LogQL syntax
429	Too Many Requests	Rate limit exceeded
5xx	Server Error	Loki is unavailable or overloaded

Use `raise_for_status()` to catch errors, and inspect the response body for details:

```
import time
import requests


def query_with_retry(
    url: str,
    query: str,
    max_retries: int = 3,
    backoff: float = 1.0,
    headers: dict[str, str] | None = None,
    auth: tuple[str, str] | None = None,
    verify: bool | str = True,  # False to skip TLS, or path to CA bundle
) -> dict:
    """Query Loki with simple retry logic for rate limits."""
    for attempt in range(max_retries):
        resp = requests.get(
            f"{url}/loki/api/v1/query",
            params={"query": query},
            headers=headers,
            auth=auth,
```

```
            verify=verify,
        )
        if resp.status_code == 429:
            wait = backoff * (2 ** attempt)
            print(f"Rate limited, retrying in {wait}s...")
            time.sleep(wait)
            continue
        resp.raise_for_status()
        return resp.json()
    raise Exception(f"Query failed after {max_retries} retries")
```

Common problems

- **Timestamps are in nanoseconds.** Loki expects Unix timestamps in nanoseconds, not seconds or milliseconds. Multiply `time.time()` by `1e9` and convert to a string.
- **At least one label matcher is required.** You cannot query without a stream selector. `{job="myapp"}` works; an empty selector does not.
- **The `direction` parameter changes result ordering.** Use `backward` (the default) to get the most recent entries first, or `forward` to get the oldest entries first.
- **The instant query endpoint only supports metric queries.** Log stream selectors like `{job="myapp"}` return a 400 error on `/query`. Use `/query_range` for log queries, and `/query` for aggregations like `rate()` or `count_over_time()`.
- **Use the `limit` parameter to control result size.** The default is 100 entries. For large time ranges, set a higher limit or paginate by adjusting the `start` parameter based on the last received timestamp.

□□□ #2
□ □□□ □□□ 2026-10-31 13:48:22 UTC
□ □□□ □□□ 2026-10-31 13:48:23 UTC