HOW CAN WE HELP?

Search our knowledge base articles

Article Detail (?tabset-530... Attachments (?tabset-5302... Subscribe

HTTP Event Collector (HEC)

Last Updated: Jul 30, 2025

The HTTP Event Collector (HEC) lets you send data and application events to your Singularity™ Data Lake over the HTTP and Secure HTTP (HTTPS) protocols. The integration supports structured and unstructured events.

This is a direct integration. You do not need to install theSentinelOne Collector SentinelOne Collector .

The maximum payload is 10 MB of uncompressed data .

HEC ingestion URL

The URL used for this data ingestion endpoint is:

```
https://ingest (https://ingest).us1.sentinelone.net 
(http://sentinelone.net)
```

Where **us1** is the deployment region that hosts your console. To find the URL for your data center, see Services and Ports for Management (https://community.sentinelone.com/s/article/000004961).

```
https://ingest.us1.sentinelone.net (https://ingest.us1.sentinelone.net)
```

The following sections describe the authorization requirements, region availability, limits, and parameters available for the HEC endpoint

- Authorization
- Compression
- Endpoints
- Examples that use the HEC endpoint

Authorization

HEC uses a token-based authentication model. You can create the required API token by creating a *service user* or by creating an *log access keys*. Both options are available in the Management Console. Use the section below to create a service user. To learn more about *log access keys*, see SDL API keys (https://community.sentinelone.com/s/article/000006763). We recommend using *service users* over *log access keys* because SDL based API keys offer less fine-grained security protections.

In most cases you can simply use this API token as your HEC Authorization token when sending data. In rare instances some HEC clients require the Authorization token to be in UUID format. In those instances, contact Support for the UUID (Universal Unique Identifier) of the key.

Creating a service user in the console

To create a *service user*, you must have the Service Users (https://usea1-purple.sentinelone.net/docs/en/service-users-2584004.html) permissions attached. If you also need to create or edit a role for the service user, you'll need the Roles (https://usea1-purple.sentinelone.net/docs/en/roles.html) permissions too. To learn more about managing roles, see Creating a New Management User Role (/document/preview/462013#UUID-2c232022-9f5f-b42f-d184-13b70b0aad8d).

If you need to automate the creation of service user, we recommend using the `POST /service-users` API operation. To learn more, see How to automate API token generation (/document/preview/2124780#UUID-0f455bfc-3516-606c-f581-4eeca1dc3c57).

To create a service user in the console

1. In the Settings toolbar, click Users > Service Users.
2. Click **Actions** and select **Create New Service User**.
3. In **Create New Service User**, enter:
   - **Name** - The name is a string that you cannot change after you set it. This is how the user is identified in the Management console. The name cannot have an equal sign (=) or angle bracket characters (<, >).
   - **Description** - Enter an optional description to identify the user or function.
   - **Expiration Date** - Set the expiration date of the API token. The default is one year. Click **Custom** to set a different date and time.
4. Click **Next**.
5. In **Select Scope of Access** select the scope or scopes a user will have access to, and the role for each scope.
6. (Optional) You can copy the scope or scopes from an existing service users by choosing **Copy The Scope From a Different User.** Then select the service user who's scopes you want to copy.
7. Click **Create User**.

Confidential Release Notes

API token visibility

This is the only time when you can view API Token created for this service user. Make sure that you save the token string in a secure location.

When successful, these steps create your API token. Select **Copy API token**, and save the token string in a secured file.
We recommend that you use an HEC client that supports a standard bearer token. If you must use a client that only supports basic authentication, then use a SDL API key with log write access permissions.

Compression

Compression with `gzip` or `zstd` is supported, and strongly recommended. Compression will decrease your egress costs. To apply compression, add the applicable key-value pairs to the header of your request:

```
“Content-Encoding: gzip”
```

```
“Content-Encoding: zstd”
```

Endpoints

Endpoint	Description
/services/collector/event	Sends logs to your data lake using JSON format
/services/collector/raw	Sends logs to your data lake in its raw format

The `/event` endpoint is usually used to ingest structured JSON events at scale.

The `/raw` endpoint is usually used to ingest:

- Lightweight integrations with unstructured events.
- Batches of data from a stateless environment, such as Amazon Lambda.
- Infrequent, unstructured events, such as a notification or webhook.
- Whole files (txt, tar.gz).

Examples that use the HEC endpoint

The following examples show how the available endpoints function, and how to structure ingestion events.

- Example 1, “Examples using `/event` endpoint”
- Example 2, “Examples using the `/raw` endpoint”
- Example 3, “Format for the `/event` endpoint”
- Example 4, “Format for the `/raw` endpoint”

Examples using `/event` endpoint

A string for the event is put in the `message` field:

```
curl -X POST https://ingest.us1.sentinelone.net/services/collector/event 
(https://ingest.us1.sentinelone.net/services/collector/event) \
        -H "Authorization: Bearer {token}" \
        -H "Content-Type: text/plain" \
        -d {
        "time" : 1537538443.0,
        "event" : "Something happened",
        "source" : "/var/log/app.log",
        "sourcetype" : "app",
        "fields" : { "env" : "prod" }
        }
```

If the event is structured JSON, fields and values are automatically parsed:

```
curl -X POST https://ingest.us1.sentinelone.net/services/collector/event 
(https://ingest.us1.sentinelone.net/services/collector/event) \
        -H "Authorization: Bearer {token}" \
        -H "Content-Type: text/plain" \
        -d {
        "time" : 1537538443.0,
        "event" : {
            "message" : "Shutdown",
            "host" : { "ip" : "127.0.0.1", "port" : 8005 }
            },
        "source" : "/var/log/app.log",
        "sourcetype" : "app",
        "fields" : { "#env" : "prod" }
        }
```

To upload a JSON file:

```
curl -X POST https://ingest.us1.sentinelone.net/services/collector/event 
(https://ingest.us1.sentinelone.net/services/collector/event) \
        -H "Authorization: Bearer {token}" \
        -H "Content-Type: application/json" \
        -d @/path/to/file.json
```

Examples using the `/raw` endpoint

Multiple events are ingested with line breaks:

```
curl -X POST https://ingest.us1.sentinelone.net/services/collector/raw 
(https://ingest.us1.sentinelone.net/services/collector/raw) \
        -H "Authorization: Bearer {token}" \
        -H "Content-Type: text/plain" \
        -d 'Log 1: Hello, World!
        Log 2: Hello Again'
```

Each line is a separate event when you upload a file:

```
curl -X https://ingest.us1.sentinelone.net/services/collector/raw 
(https://ingest.us1.sentinelone.net/services/collector/raw) \
        -H "Authorization: Bearer {token}" \
        -H "Content-Type: text/plain" \
        --data-binary @/path/to/file.txt
```

Query parameters become fields in the UI. To add custom fields to your events simply add a query parameter for each desired field. To add 2 fields labeled `dataSource.name (http://dataSource.name)` and `dataSource.vendor` , add the following query parameters to the URL

```
curl -X POST 'POST
https://ingest.us1.sentinelone.net/services/collector/raw?
(https://ingest.us1.sentinelone.net/services/collector/raw?)dataSource.name
 (http://dataSource.name)=myName&amp;dataSource.vendor=myVendor'; \
    -H "Authorization: Bearer {token}" \
    -H "Content-Type: text/plain" \
    -d "Log 1: Hello, World!
        Log 2: Hello Again"
```

To set a specific parser use the query parameter `?sourcetype=parserName`.

```
curl -X POST 'POST
https://ingest.us1.sentinelone.net/services/collector/raw?
sourcetype=parserName'
(https://ingest.us1.sentinelone.net/services/collector/raw?
sourcetype=parserName'); \
    -H "Authorization: Bearer {token}" \
    -H "Content-Type: text/plain" \
    -d "Log 1: Hello, World!
        Log 2: Hello Again"
```

Format for the `/event` endpoint

```
{
    "event" : {               // Required.
        "message": "This happened",
        "severity": "INFO"
    },
    "time" : 1537538443.0,  // Optional. UNIX epoch time, in the format
<sec>.<ms>, UTC.
    "host": "prod-us-1",    // Optional. Usually the hostname you sent
data from.
    "source" : "/var/log/app.log",  // Optional. The source or logfile of
the event.
    "sourcetype" : "app",   // Optional. Sets a parser to extract fields;
    "index": "main",        // Optional. Index name.
    "fields" : { "#datacenter" : "amazon-east1" }  // Optional. Fields to
add to the event.
}
```

- **event**

  Required. The event to ingest. The event can be a string, which is put in the `message` field:

  ```
  "event":  "Hello world!"
  ```

  An event can also be structured JSON:

  ```
  "event": {
          "message": "Something happened",
          "severity": "INFO"
      }
  ```

  Fields are automatically extracted from JSON key-value attributes. The above example extracts `message` and `severity` fields. With structured JSON, it is not necessary to set a `sourcetype` (parser).

  Nested fields are flattened in `.` dot format. For example, the fields `host.ip` and `host.port` are extracted from `{"host" : { "ip" : "127.0.0.1", "port" : 8005 }}`.

  You can send events in batches as concatenated JSON, with no delimiters. For example:

```
{
  "event":"Something happened",
  "time": 1447828325
}

{
  "event":"Something else happened",
  "time": 1447828326
}

{
  "time" : 1537538443.0,
  "event" : "Fri Sep 21 2018 14:00:43 GMT - the app did this",
  "source" : "/var/log/app.log",
  "sourcetype" : "app",
  "fields" : { "#env" : "prod" }
}

{
  "time" : 1537538927.0,
  "event" : {
        "message": "This happened",
        "severity": "INFO"
    },
  "fields" : { "#datacenter" : "amazon-east1" }
}
```

- **time**

  Optional. Time of the event in UNIX epoch time, in the format `<sec>.<ms>` , UTC. For example, `1433188255.500` : 1433188255 seconds and 500 milliseconds after UNIX epoch, which is Monday, June 1, 2015, at 7:50:55 PM GMT.

  If a `time` is not set, one is assigned at the time of ingestion. If a parser is set with the timestamp field (/document/preview/2069945#UUID-6825c635-fde5-0d43-3890-a9f4babc9354), it overrides all other times.

  In the UI, this property becomes the `timestamp` field.

- **host**

  Optional. Usually the host name of the client you sent data from.

  In the UI, this property becomes the `serverHost` field.

- **source**

  Optional. The source or logfile of the event. For example, if you send data from an application, set this to the name of the application.

- **sourcetype**

  Optional. The name of the parser (/document/preview/1872300#UUID-5b63677d-13ac-a6dd-d981-9766dd6ff425) to extract fields from the event. If you send structured JSON, the event it is automatically parsed, regardless of `sourcetype` .

  In the UI, this property becomes the `parser` field.

- **fields**

  Optional. A non-nested JSON object with key-value pairs. Used to attach more attributes to an event, for example server, region, or other attributes not included in the event itself.

  In the UI, each key becomes a field name.

Format for the `/raw` endpoint

- Query parameters in the URL become fields in the UI for the event.

- You can upload multiple events with line breaks (`\n` , `\r` , or `\r\n`).
- If you upload a file, each line is a separate event. Test first because some applications remove line breaks. For `curl` , you can try the --data-binary flag (https://curl.se/docs/manpage.html).
- Line grouping is supported through multi-line message parsing rules (/document/preview/2069950#UUID-f8005d9d-bbe2-82f7-0bfb-e4b66d2b8bdd). Line grouping is not supported across upload batches.

Unsupported Features

- Deduplication is not supported. The endpoints are designed for high reliability and uptime. Clients will rarely have to retry requests, the main cause of duplicate records.
- If two log lines have the same `timestamp` , we guarantee ordering in the same upload, but not when the lines are split in batches.

HTTP Event Collector (HEC) API Limits

Name	Default	Description
Rate of HTTP Event Collector (HEC) log ingestion requests	1000 requests/sec	The maximum number of HTTP Event Collector (HEC) log ingestion requests
Bytes per second (BPS) for HTTP Event Collector (HEC) log ingestion requests	2 GB	The maximum number of bytes per second per account for OpenTelemetry log ingestion requests

Was this article helpful? Yes No

Related Articles

HTTP Plugin
(/s/article/000008655)

SDL API - addEvents
(/s/article/000006773)

SDL API Keys
(/s/article/000006763)

Working with HTTP Monitors
(/s/article/000006759)

SDL API Overview
(/s/article/000006771)