Google

Monthly Partner Series

Chronicle Data Acquisition

March 2023

Google Cloud

Learn...

Common log source categories

Different Ingestion methods

Forwarder management, architecture, and example configuration

How to use APIs to ingest data

Feed Management UI overview and configuration example

Common log source categories

Security

- Firewall
- Web Proxy
- CASB
- DLP
- IPS/IDS/WAF
- Auth logs
- DAM

Network

- DNS
- DHCP
- email logs

Endpoint

- Anti Virus
- EDR
- Windows OS logs
- Unix/Linux OS logs
- Endpoint DLP

Application

- Workspace
- Office 365
- Salesforce
- Other enterprise applications on-prem and cloud

Cloud

Security and endpoint logs from cloud workloads in:

- GCP
- AWS
- Azure

Common contextual information sources categories

User Management

- Active Directory
- Cloud Identity
- Azure AD

Asset Management

- ServiceNow
- Nucleus
- CMDB

Threat Intelligence

- Virus Total
- Proofpoint
- Anomali

Vulnerability

- Qualys
- Tenable
- Rapid7

How do we collect data into the platform?

Forwarder

Lightweight container on Linux or Windows.

Collection Methods:

- Syslog
- Syslog over TLS
- PCAP
- Local File-System
- Splunk API
- Kafka

Cloud

Cloud-to-cloud service via Feed Management UI

Collection Sources:

- GCS bucket
- AWS S3 / SQS
- Azure Blob
- HTTPS Files
- Common SaaS vendors
  - Azure / O365
  - Salesforce

Ingestion API

Chronicle API to send data in UDM or unstructured format.

Supported:

- UDM
- Unstructured Data
- Entity

Direct Ingestion

Chronicle directly ingests GCP log sources

Supported:

- GCP Cloud Audit
- GCP DNS
- SCC findings

How do we collect data into the platform?

Forwarder

Lightweight container on Linux or Windows.

Collection Methods:

- Syslog
- Syslog over TLS
- PCAP
- Local File-System
- Splunk API
- Kafka

Cloud

Cloud-to-cloud service via Feed Management UI

Collection Sources:

- GCS bucket
- AWS S3 / SQS
- Azure Blob
- HTTPS Files
- Common SaaS vendors
  - Azure / O365
  - Salesforce

Ingestion API

Chronicle API to send data in UDM or unstructured format.

Supported:

- UDM
- Unstructured Data
- Entity

Direct Ingestion

Chronicle directly ingests GCP log sources

Supported:

- GCP Cloud Audit
- GCP DNS
- SCC findings

Google Cloud

How do we collect data into the platform?

Forwarder

Lightweight container on Linux or Windows.

Collection Methods:

- Syslog
- Syslog over TLS
- PCAP
- Local File-System
- Splunk API
- Kafka

Cloud

Cloud-to-cloud service via Feed Management UI

Collection Sources:

- GCS bucket
- AWS S3 / SQS
- Azure Blob
- HTTPS Files
- Common SaaS vendors
  - Azure / O365
  - Salesforce

Ingestion API

Chronicle API to send data in UDM or unstructured format.

Supported:

- UDM
- Unstructured Data
- Entity

Direct Ingestion

Chronicle directly ingests GCP log sources

Supported:

- GCP Cloud Audit
- GCP DNS
- SCC findings

Google Cloud

How do we collect data into the platform?

Forwarder

Lightweight container on Linux or Windows.

Collection Methods:

- Syslog
- Syslog over TLS
- PCAP
- Local File-System
- Splunk API
- Kafka

Cloud

Cloud-to-cloud service via Feed Management UI

Collection Sources:

- GCS bucket
- AWS S3 / SQS
- Azure Blob
- HTTPS Files
- Common SaaS vendors
  - Azure / O365
  - Salesforce

Ingestion API

Chronicle API to send data in UDM or unstructured format.

Supported:

- UDM
- Unstructured Data
- Entity

Direct Ingestion

Chronicle directly ingests GCP log sources

Supported:

- GCP Cloud Audit
- GCP DNS
- SCC findings

Google Cloud

How do we collect data into the platform?

Forwarder

Lightweight container on Linux or Windows.

Collection Methods:

- Syslog
- Syslog over TLS
- PCAP
- Local File-System
- Splunk API
- Kafka

Cloud

Cloud-to-cloud service via Feed Management UI

Collection Sources:

- GCS bucket
- AWS S3 / SQS
- Azure Blob
- HTTPS Files
- Common SaaS vendors
  - Azure / O365
  - Salesforce

Ingestion API

Chronicle API to send data in UDM or unstructured format.

Supported:

- UDM
- Unstructured Data
- Entity

Direct Ingestion

Chronicle directly ingests GCP log sources

Supported:

- GCP Cloud Audit
- GCP DNS
- SCC findings

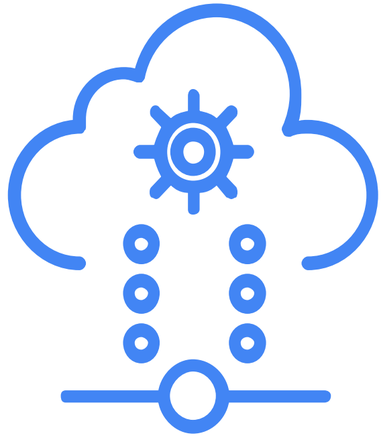
Google Cloud

Chronicle Forwarder Configuration

The Chronicle Forwarder is typically used for on-premise data collection

The Chronicle Forwarder is typically used for on-premise data collection

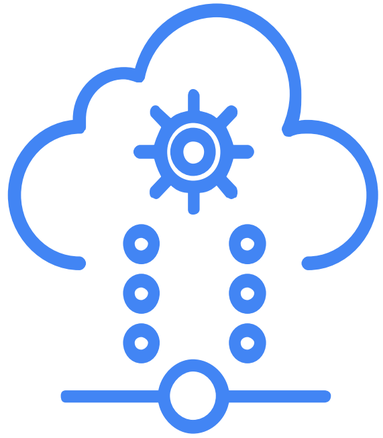
Deployed as a Docker **container** on Linux or Windows.



The Chronicle Forwarder is typically used for on-premise data collection

Deployed as a Docker **container** on Linux or Windows

One forwarder may have **multiple collectors** each corresponding to a data source.

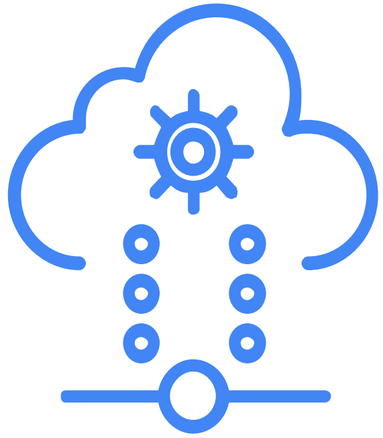


The Chronicle Forwarder is typically used for on-premise data collection

Deployed as a Docker **container** on Linux or Windows.

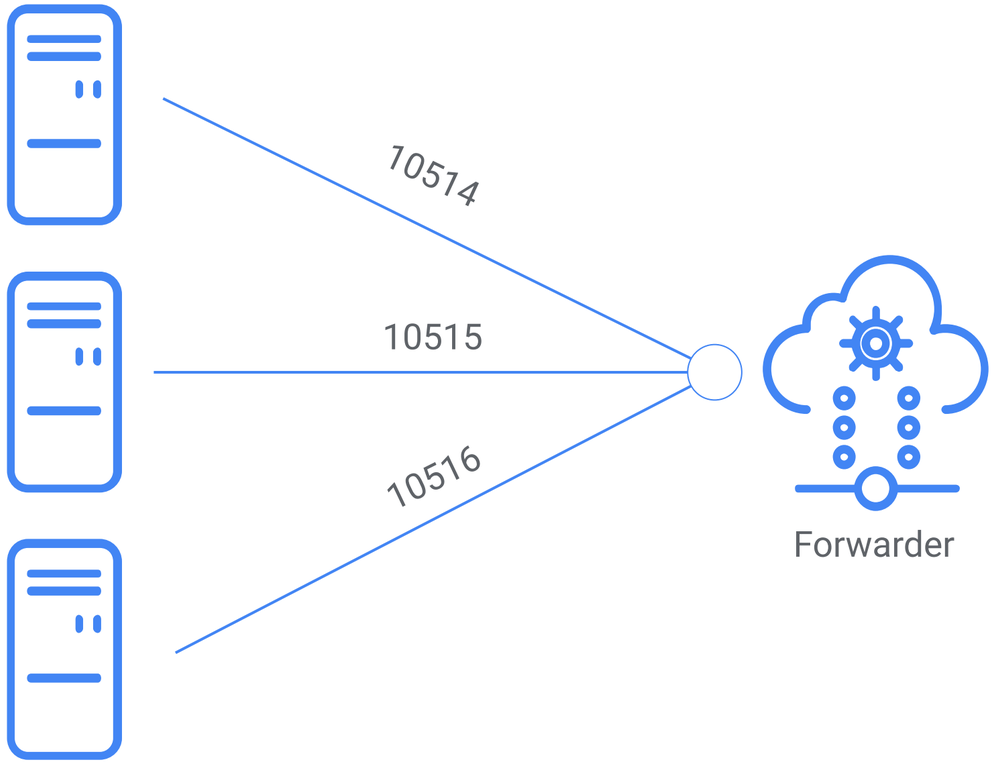
One forwarder may have **multiple collectors** each corresponding to a data source.

1. Syslog and Syslog over TLS
2. Kafka
3. PCAP for DNS and DHCP
4. Reading files from the operating system
5. Queries from Splunk via the Splunk API

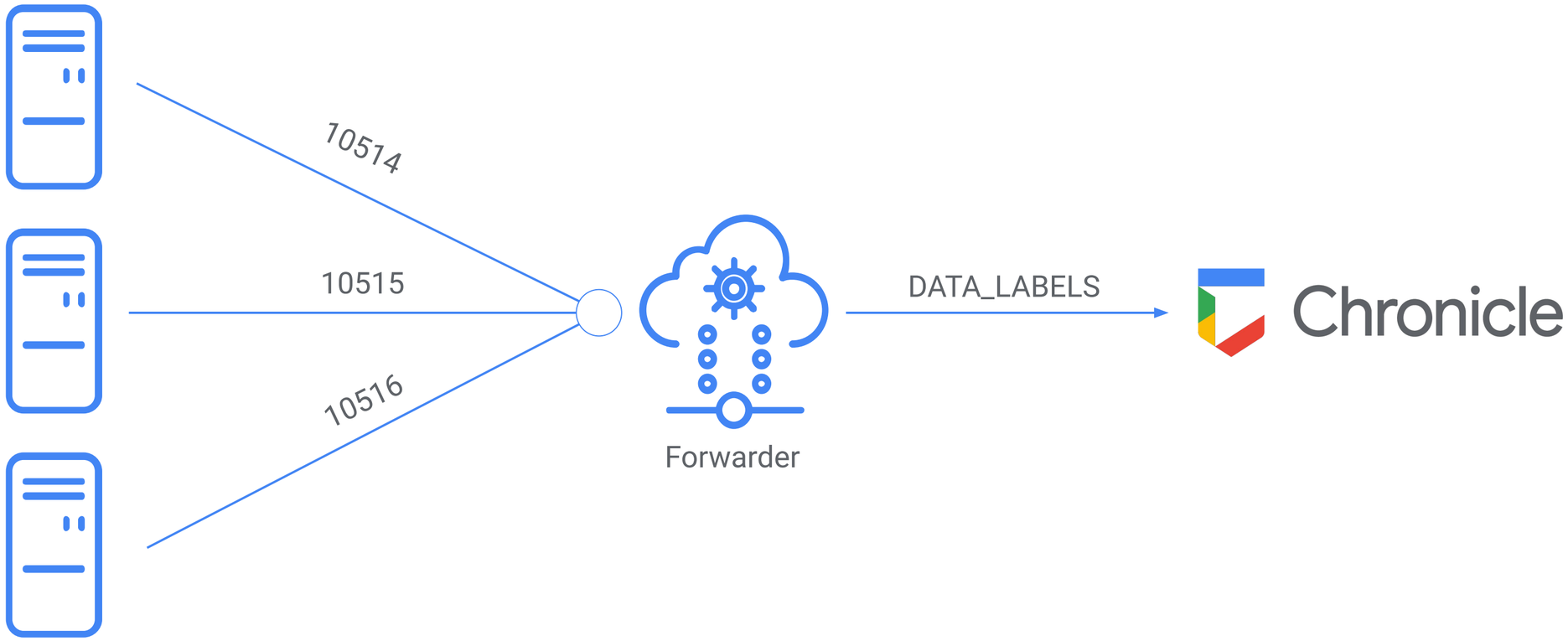


[Forwarder Installation Documentation]

The forwarder can receive multiple streams of syslog data which must be on separate ports.



The forwarder can receive multiple streams of syslog data which must be on separate ports.



Data labels are used to identify the data type being collected

Each **data source** must have an associated data label.

Determines which parser is used for **normalization**.

Full list can be retrieved via the **Partner Ingestion API**.

Required for (almost) all ingestion methods, most common in **Forwarder configurations**.

Product	Data Label
Windows DNS	WINDOWS_DNS
Cisco ASA	CISCO_ASA_FIREWALL
Carbon Black	CB_EDR

Settings and API key are defined within a single configuration file.

YAML based configuration file with two sections.

Settings and API key are defined within a single configuration file.

YAML based configuration file with two sections.

The **output section** defines IDs and an API key which should not be modified.

Settings and API key are defined within a single configuration file.

YAML based configuration file with two sections.

The **output section** defines IDs and an API key which should not be modified.

The **collectors section** defines the expected data types and ingestion methods and must be modified.

Settings and API key are defined within a single configuration file.

YAML based configuration file with two sections.

The **output section** defines IDs and an API key which should not be modified.

The **collectors section** defines the expected data types and ingestion methods and must be modified.

Container or service must be **restarted** to apply changes.

Don’t modify the output section which defines IDs and the API key

```
output:
  url: malachiteingestion-pa.googleapis.com:443
  identity:
    identity:
    collector_id: <COLLECTOR_ID>
    customer_id: <CUSTOMER_ID>
    secret_key: |
      {
        "type": "service_account",
        "project_id": "malachite-<CODE>",
        "private_key_id": "<REDACTED>",
        "private_key": "-----BEGIN PRIVATE KEY-----\n<REDACTED>-----END PRIVATE
KEY-----\n",
        "client_email":
"malachite-<CODE>-1@malachite-<CODE>.iam.gserviceaccount.com",
        "client_id": "<REDACTED>",
        "auth_uri": "https://accounts.google.com/o/oauth2/auth",
        "token_uri": "https://oauth2.googleapis.com/token",
        "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
        "client_x509_cert_url": "<REDACTED>"
      }
```

Configure collectors to define the expected data types and ingestion methods

```
collectors:
  - pcap:
      common:
        enabled: true
        data_type: PCAP_DNS
        batch_n_seconds: 10
        batch_n_bytes: 1048576
      interface: any
      bpf: udp port 53
  - syslog:
      common:
        enabled: true
        data_type: WINDOWS_DHCP
        batch_n_seconds: 10
        batch_n_bytes: 1048576
      tcp_address: 0.0.0.0:10514
      udp_address: 0.0.0.0:10514
      connection_timeout_sec: 60
```

Configure collectors to define the expected data types and ingestion methods

```
collectors:
  - kafka:
      common:
        enabled: true
        data_type: NIX_SYSTEM
        batch_n_seconds: 10
        batch_n_bytes: 1048576
      username: user
      password: password
      topic: example-topic
      group_id: chronicle-forwarder
      timeout: 60s
      brokers: ["broker-1:9092", "broker-2:9093"]
      tls:
        insecureSkipVerify: true
        certificate: "/path/to/cert.pem"
        certificate_key: "/path/to/cert.key"
```

```
  - syslog:  (for TLS)
      common:
        enabled: true
        data_type: WINDOWS_DHCP
        batch_n_seconds: 10
        batch_n_bytes: 1048576
      tcp_address: 0.0.0.0:10514
      udp_address: 0.0.0.0:10514
      connection_timeout_sec: 60
      certificate: "/opt/chronicle/external/certs/edb3ae966a7bbe1f.pem"
      certificate_key: "/opt/chronicle/external/certs/forwarder.key"
```

Configure collectors to define the expected data types and ingestion methods - continued

```
collectors:
  - splunk:
      common:
        enabled: true
        data_type: BRO_DHCP
        data_hint:
        batch_n_seconds: 10
        batch_n_bytes: 1048576
      url: https://splunk1.corp.xyz.com:8089
      is_ignore_cert: true
      query_string: search index=wineventlogs sourcetype=dhcpsrvlog
  - file:
      common:
        enabled: true
        data_type: CSV_CUSTOM_IOC
        batch_n_seconds: 10
        batch_n_bytes: 1048576
      file_path: /opt/chronicle/ioc
      poll: true
```

Advanced configurations - namespace

- Use **namespace** labels to identify logs from distinct **network segments** and deconflict overlapping IP addresses.
- You can configure a namespace label for an **entire** Forwarder or within a **specific collector** of the Forwarder.
- If both are included, the specific **collector's namespace** takes precedence.

```
metadata:  <-- at Forwarder level
   namespace: east-finance
   lables:
     forwarderId: 198215d64-d86c-4fe1-902c-51271b09d3c8
output:
<output section>

collectors:
  - syslog:
      common:
        metadata:  <-- at collector level
          labels:
            forwarderId: 18005d64-d86c-4fe1-902c-51271b09d3c8
        enabled: true
        data_type: ALL_TYPES
        data_hint:
        batch_n_seconds: 10
        batch_n_bytes: 1048576
      tcp_address: 0.0.0.0:10514
      udp_address: 0.0.0.0:10514
      connection_timeout_sec: 60
```

Advanced configurations - namespace (cont'd)

Namespaces appear with the associated assets in the Chronicle user interface.

Search bar

Advanced configurations - arbitrary labels

- **Labels** are used to attach a**rbitrary metadata** to logs using key/value pairs.
- Labels can be configured for an entire Forwarder or within a specific collector of a Forwarder.
- If both are provided, the labels are merged with the collector's keys taking precedence over the Forwarder's keys if the keys overlap.

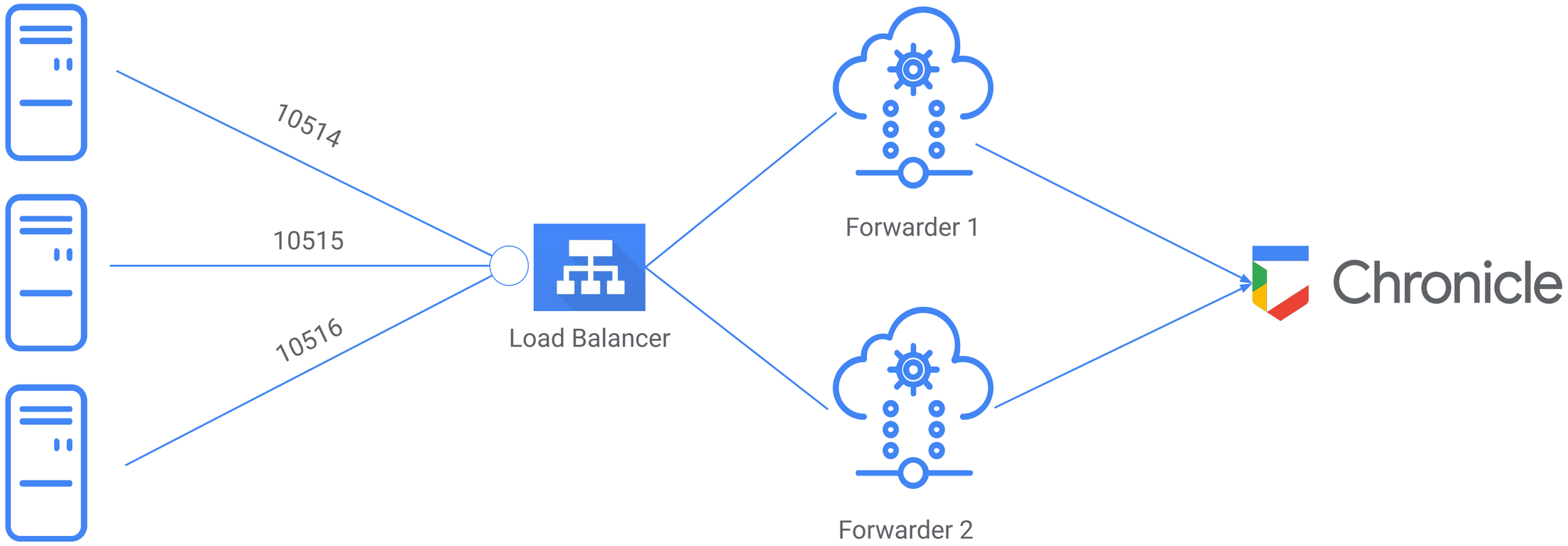
```
metadata:
   namespace: east-finance
   labels:                          at Forwarder level
     forwarderId: 198215d64-d86c-4fe1-902c-51271b09d3c8
output:
<output section>

collectors:
  - syslog:
      common:
        metadata:
          labels:                   at collector level
            forwarderId: 18005d64-d86c-4fe1-902c-51271b09d3c8
            division: finance
        enabled: true
        data_type: ALL_TYPES
        data_hint:
        batch_n_seconds: 10
        batch_n_bytes: 1048576
      tcp_address: 0.0.0.0:10514
      udp_address: 0.0.0.0:10514
      connection_timeout_sec: 60
```

Advanced configurations - regex filters

- **Filters** must include a regular expression and, optionally, define a behavior - allow or block (default) - when there is a match
- May define an **arbitrary number** of filters.
- **Block filters** take precedence over allow filters.
- Filters must be assigned a **name**.
- Filters defined at the root of the configuration are **merged** with filters defined at the collector level.
- **Collector level filters** take precedence in cases of conflicting names.

```
regex_filters:                          <-- root filter
  allow_filter:                         <-- filter label
    regexp: ^<[1-9][0-9]?$>.*$
    behavior_on_match: allow
collectors:
- syslog:
    common:
      regex_filters:
        block_filter_1:
          regexp: ^.*foo.*$
          behavior_on_match: block      <--
        block_filter_2:
          regexp: ^.*bar.*$
      batch_n_bytes: 1048576
      data_type: NIX_SYSTEM
      enabled: true
    tcp_address: 0.0.0.0:30000
    connection_timeout_sec: 60
- syslog:
    common:
      batch_n_bytes: 1048576
      batch_n_seconds: 10
      data_type: WINEVTLOG
      enabled: true
    tcp_address: 0.0.0.0:30001
```

Load Balancing and HA



Load Balancing and HA

- Layer 4 load balancer may be installed between the data source and forwarder instances
- Allows a customer to distribute log collection across multiple forwarders or send logs to a different forwarder if one fails
- Supported with the **syslog** collection type **only**
- Linux forwarder includes a built-in HTTP server that responds to HTTP health checks from the load balancer
- Liveness checks for container schedulers/orchestrators
  - http://<host:port>/meta/available
- Readiness checks and traditional load balancer health checks
  - http://<host:port>/meta/ready

```
collectors:
- syslog:
    common:
      batch_n_bytes: 1048576
      batch_n_seconds: 10
      data_hint: null
      data_type: NIX_SYSTEM
      enabled: true
    tcp_address: 0.0.0.0:30000
    connection_timeout_sec: 60
- syslog:
    common:
      batch_n_bytes: 1048576
      batch_n_seconds: 10
      data_hint: null
      data_type: WINEVTLOG
      enabled: true
    tcp_address: 0.0.0.0:30001
    connection_timeout_sec: 60
server:
  graceful_timeout: 15s
  drain_timeout: 10s
  http:
    port: 8080
    host: 0.0.0.0
    read_timeout: 3s
    read_header_timeout: 3s
    write_timeout: 3s
    idle_timeout: 3s
    routes:
    - meta:
        available_status: 204
        ready_status: 204
        unready_status: 503
```

System requirements & Network needs

- **RAM**—1 GB for each collected data type
- **CPU**—2 CPUs for each 10,000 events per second (EPS)
- **Disk**—100 MB of disk space is sufficient, regardless of how much data the Chronicle forwarder handles.

Required connections between Forwarder and Chronicle

Connection Type	Destination	Port
TCP	**malachiteingestion-pa.googleapis.com**	443
TCP	**europe-malachiteingestion-pa.googleapis.com**	443
TCP	**accounts.google.com**	443
TCP	**gcr.io**	443
TCP	**oauth2.googleapis.com**	443
TCP	**storage.googleapis.com**	443
TCP	**clientservices.googleapis.com**	443

How to get the latest list of Log Types?

- API (v1):

  `https://malachiteingestion-pa.googleapis.com/v1/logtypes?key=<your key>`

  - Example:

  `curl --request GET`
  `https://malachiteingestion-pa.googleapis.com/v1/logtypes?key=<YourKey>`

- API (v2):

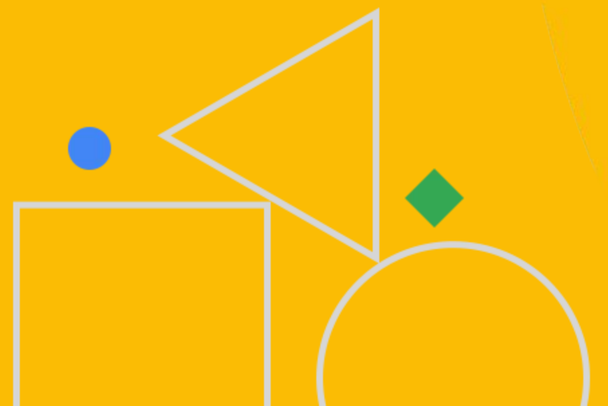
  `python3 -m ingestion.list_log_types -r {asia-southeast1,europe,us}`

- Output:

```
{ "logTypes": [
   {
         "logType": "ACALVIO",
         "description": "Acalvio"
      },
      {
      "logType": "ACCELLION",
      "description": "Accellion"
      }, ...
```

Google Cloud

Cloud-to-Chronicle



Chronicle can ingest data directly from the Cloud

- Most commonly from a **cloud storage bucket**, and many **API options available**.
- Requires **credentials** or **permissions** to access the data.
- 2 ways of ingestion:
  - Feeds in Chronicle UI
    - Native support for the pre-built APIs
  - Cloud Ingestion Scripts
    - Customize support using cloud functions
    - https://github.com/chronicle/ingestion-scripts
    - Documentation: https://cloud.google.com/chronicle/docs/ingestion/ingest-using-cloud-functions

Chronicle can ingest data directly from the Cloud

Most commonly from a **cloud storage bucket**, and many **API options available**.

Requires **credentials** or **permissions** to access the data.

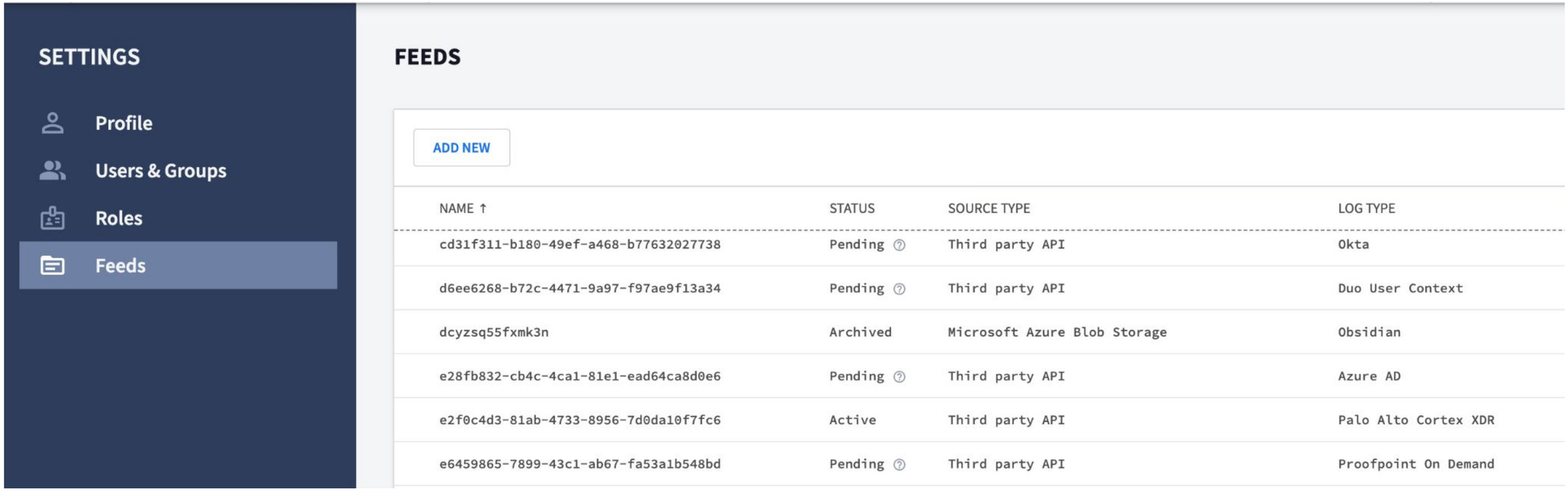
Feed Management UI

1. Google Cloud Storage (GCS)
2. Amazon S3 / SQS
3. Azure Blob Storage
4. Azure AD & Office 365 (and many others)
5. HTTPS
6. SFTP

Google Cloud

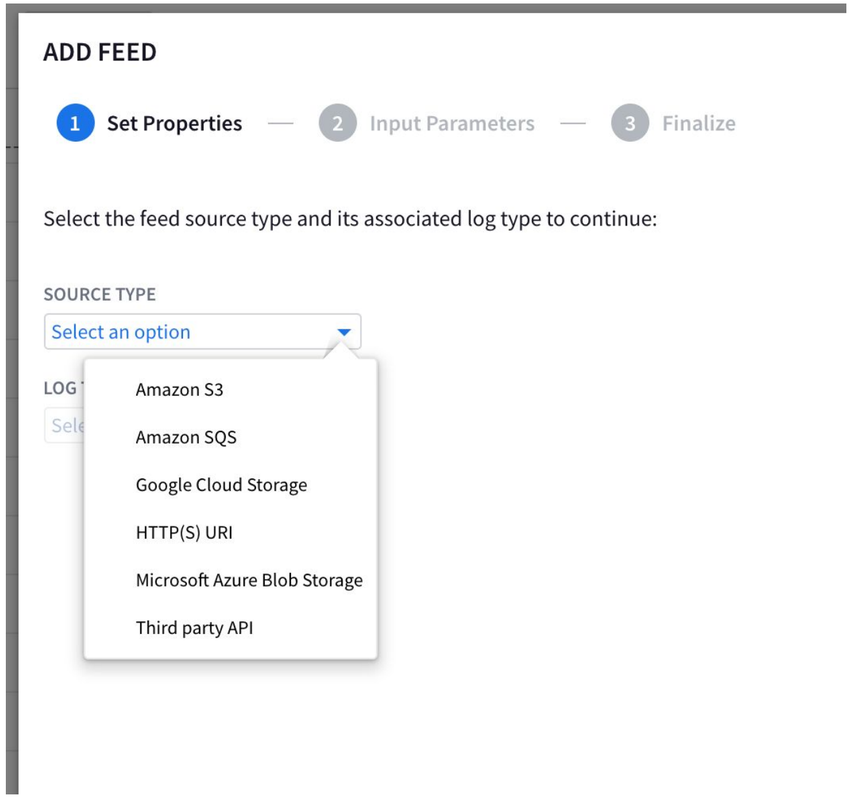
Configuring a Cloud Feed within Chronicle UI

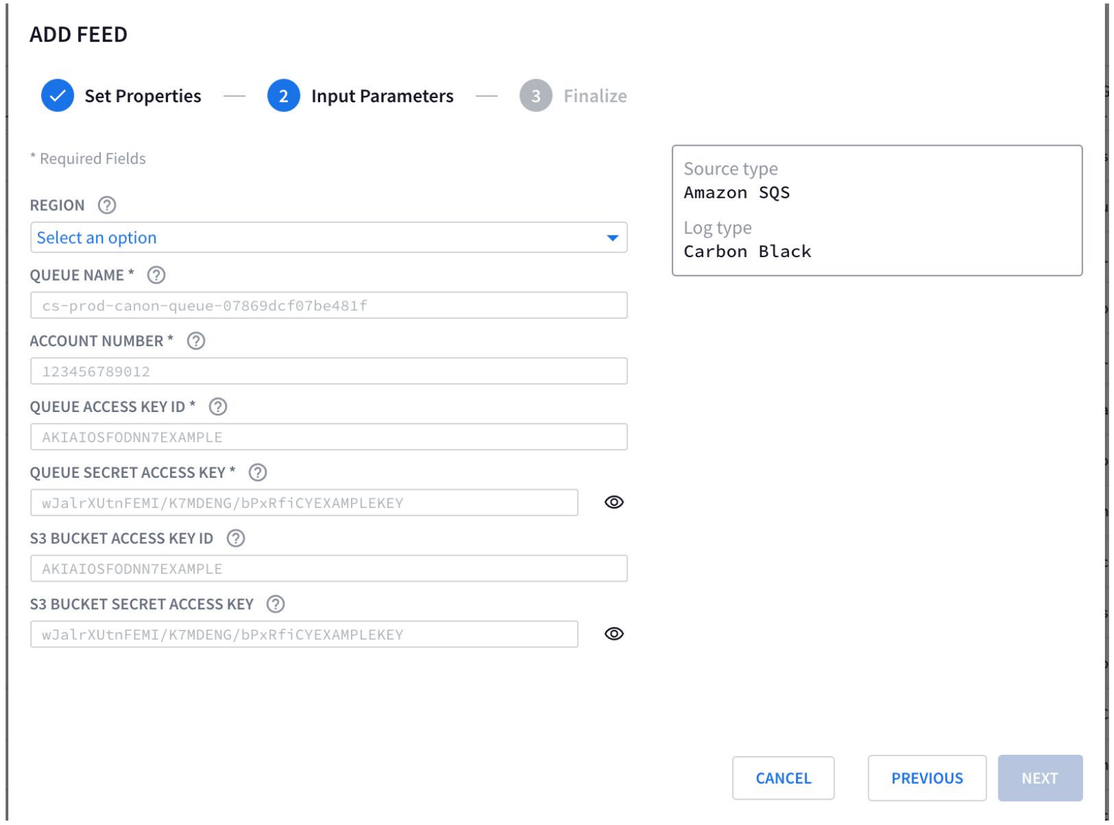
Go to **Settings** under the Application menu, click **Feeds**, then **Add New**, as shown:



Configuring a Cloud Feed within Chronicle UI (cont'd)

Then configure **properties** and other **input parameters** as needed for the specific Feed.





Google Cloud

Example ingestion script as a Cloud Function

- This example assumes that you have access to a Google VM via SSH.
- Upload the ingestion script either using the file interface or SCP.
- Requires **credentials** or **permissions** to access the data.
- All ingestion scripts other than Pub/Sub poll the data source on a periodic basis. Pub/Sub is a continuous data stream.

```
gcloud functions deploy <FUNCTION NAME> --entry-point main --trigger-http --runtime python39 --env-vars-file .env.yml
```

Thank you.