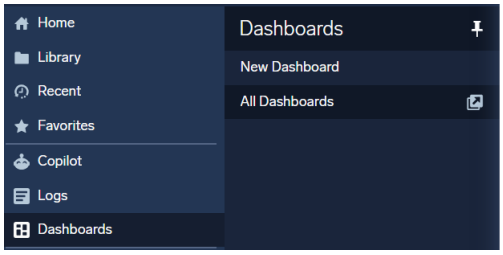
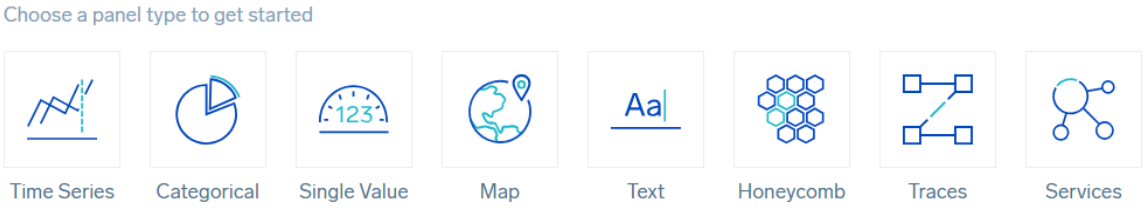
Creating a SOC Analyst's Dashboard

To create a SOC Analyst Dashboard, click on the Dashboard section in the left pane and choose the New Dashboard option.



For our first dashboard, choose the Time Series option.

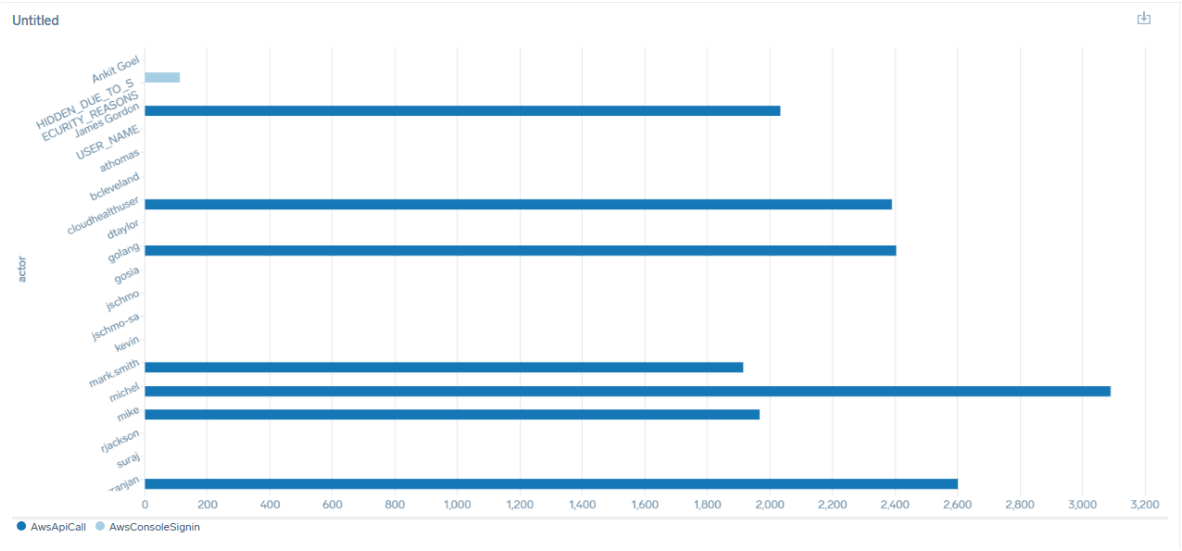


1. Top 30 User Activity Panel

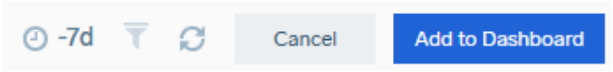
To create a user activity panel, we should use the following query.

Query - _sourceCategory=Labs/AWS/CloudTrail
| json field=_raw "userIdentity.userName" as actor
| json field=_raw "eventType" as event_type
| json field=_raw "sourceIPAddress" as src_ip
| json field=_raw "eventName" as event_name
| count by actor,event_type,event_name
| top 30 actor by event_type,_count
| transpose row actor column event_type

As CloudTrail logs are stored in a JSON format, we have to extract the username, event type, source IP address, and event name. We do it because that way we can use the extracted values to monitor user activity. Then we use an aggregate operator to group the data according to our chosen criteria. In this example, we want the top 30 users with their counted event types shown.
The Top 30 User Activity Panel is a valuable component of a SIEM dashboard because it provides immediate visibility into user behavior, which is critical for detecting both malicious and unusual activity. It can help you detect insider threats or compromised accounts, highlight anomalous behaviour, and visualize behaviour trends over time.
Once we run the query, we can present it in a bar chart, which would look like that.

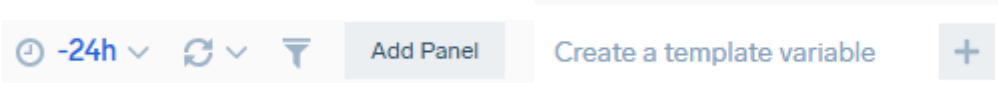


We then change the title to ‘Top 30 User activity’, and add it to the dashboard in the upper right corner.

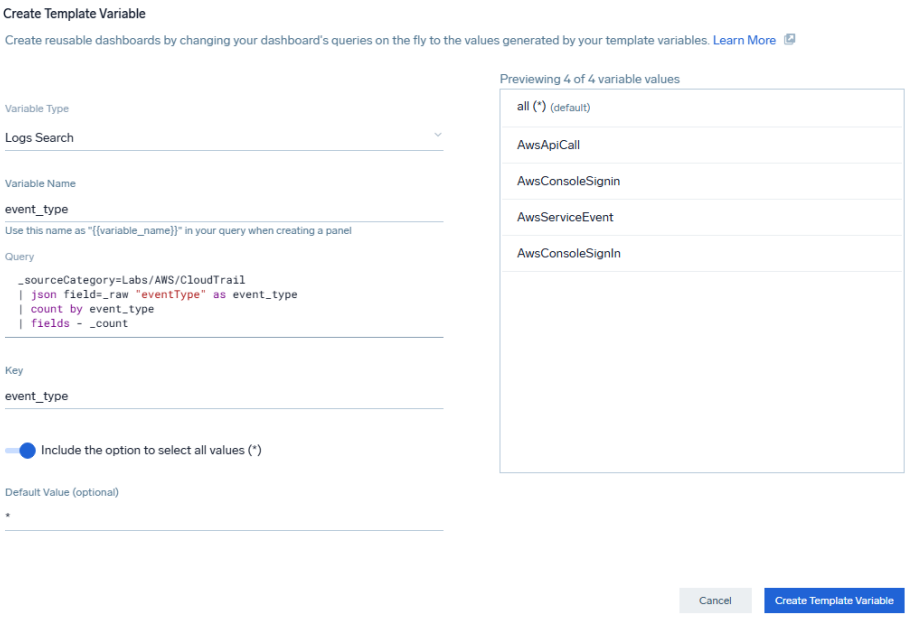


2. Dashboard Template Variables

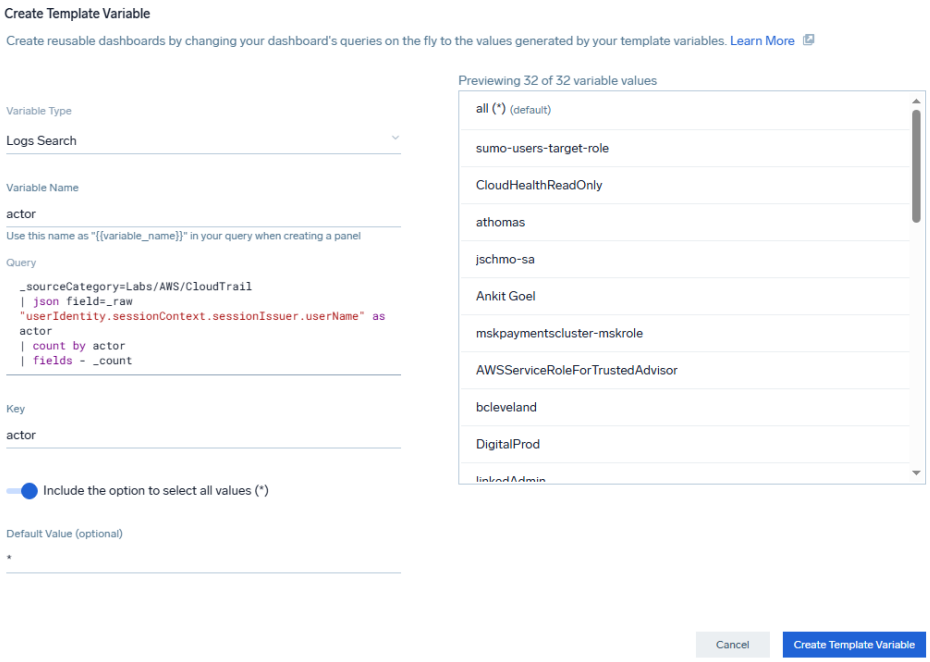
To add more flexibility to queries and dashboards, we can use template variables. To create some, in the upper right corner, change the time range to 24 hours and click on the filter icon to display the template variable bar. Then click on the plus sign to create the variable.



Then fill the form with the following details, and click ‘Create Template Variable’.



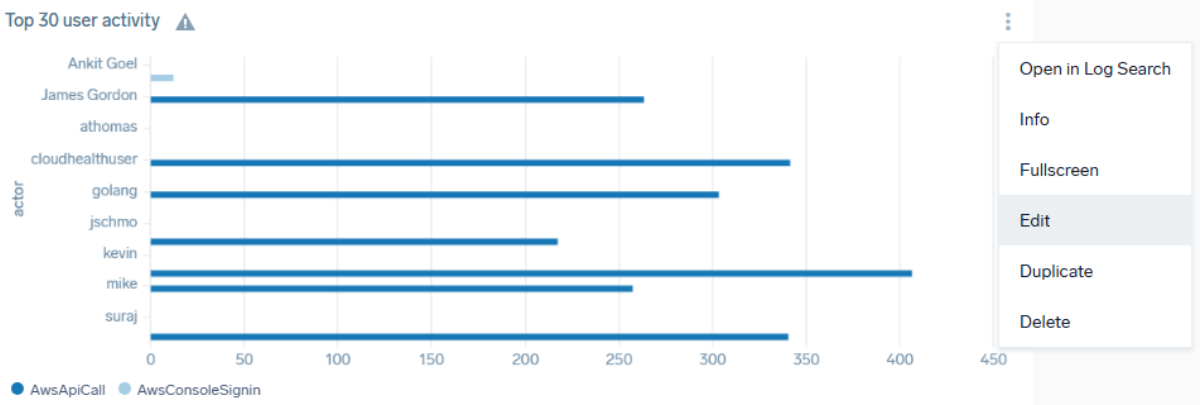
Using the same method, add another template variable to our dashboard. Use the following details.



We can now rename our dashboard to ‘Cloud Security Dashboard’.

3. Modifying queries to use the variables

We want to be able to use our variables in an already created query for the ‘Top 30 users activity’ dashboard. To do that, we need to click the 3 dots on the already created dashboard and click the ‘edit’ option.



Update your query so it looks like that.
Query - _sourceCategory=Labs/AWS/CloudTrail
| json field=_raw "userIdentity.userName" as actor
| json field=_raw "eventType" as event_type
| json field=_raw "sourceIPAddress" as src_ip
| json field=_raw "eventName" as event_name

| where event_type matches "{{event_type}}"
| where actor matches "{{actor}}"
| count by actor,event_type,event_name
| top 30 actor by event_type,_count
| transpose row actor column event_type

Now, in the right upper corner, you can click update dashboard.

4. Console Logins Geo Location

The geolocation of console logins is a critical data point in SIEM for identifying potentially malicious activity, compromised accounts, and policy violations. It is important for SOC for a couple of reasons, such as identifying suspicious or unexpected locations, e.g., if your company operates only in the UK, it would be very suspicious to detect any console login from Asia or South America. It can also help to create a baseline for normal user behaviour or detect use of stolen credentials.
To create geo-location dashboard, click the 'Add Panel' and select a 'Map' option. Type in the following query and change the name of the panel to 'Geo Location of Console Logins'.
Query – _sourceCategory="Labs/AWS/CloudTrail"
| json field=_raw "userIdentity.userName" as actor
| json field=_raw "eventType" as event_type
| json field=_raw "sourceIPAddress" as src_ip
| json field=_raw "responseElements.ConsoleLogin" as result
| where !isEmpty(actor)
| where !isEmpty(event_type)
| where event_type matches "{{event_type}}"
| where actor matches "{{actor}}"
| lookup latitude, longitude, country_name, city, region from geo://location on ip=src_ip
| count by actor,src_ip,latitude,longitude,country_name,event_type

Add the map to your dashboard.

5. Top 10 number of failed login attempts.

The "Top 10 Users with Failed Login Attempts" panel is extremely useful in a SIEM dashboard because it helps identify authentication-related threats, potential account compromise, and misconfigurations. It can be an early indicator of brute-force attacks as those quite often show as many failed logins for a single user. It could also help detect a malicious user testing stolen credentials. Repeated failed logins can cause account lockout, which can affect the company's productivity. The SOC analyst can spot misconfigured systems and help to reduce the helpdesk burden.
To add the top 10 of failed logins, click 'Add Panel' and choose the 'Time Series' option. Type in the following query, select the view to 'Table', and rename the panel to 'Top 10 Number of Failed Login Attempts'.
Query – _sourceCategory=Labs/AWS/CloudTrail
| json field=_raw "userIdentity.userName" as actor
| json field=_raw "eventType" as event_type
| json field=_raw "sourceIPAddress" as src_ip
| json field=_raw "responseElements.ConsoleLogin" as result
| json field=_raw "eventName" as event_name
| where actor matches "{{actor}}"
| where !isEmpty(actor)
| where !isEmpty(event_type)
| where result = "Failure"
| where event_type = "AwsConsoleSignIn"
| timeslice 1h
| count by _timeslice,actor,event_type,event_name,result
| top 10 actor by _timeslice,event_type,result,_count

Top 10 Number of Failed Login Attempts

	actor	Time	event_type	result	_count
1	dtaylor	06/23/2025 16:00:00	AwsConsoleSignIn	Failure	2
2	jschmo-sa	06/23/2025 16:00:00	AwsConsoleSignIn	Failure	1
3	gosia	06/23/2025 15:00:00	AwsConsoleSignIn	Failure	1
4	athomas	06/23/2025 15:00:00	AwsConsoleSignIn	Failure	1
5	jschmo-sa	06/23/2025 15:00:00	AwsConsoleSignIn	Failure	1
6	athomas	06/23/2025 14:00:00	AwsConsoleSignIn	Failure	1
7	rjackson	06/23/2025 14:00:00	AwsConsoleSignIn	Failure	1
8	gosia	06/23/2025 14:00:00	AwsConsoleSignIn	Failure	1

6. Account compromise detection from a brute force attack

Brute force attacks are noted by a user falling to login for a number of times, and then logging in successfully. To add a panel with display, we should click ‘Add Panel’ and choose the ‘Time series’ option. Type in the following query, select the ‘Table’ chart type, and rename the panel to ‘Account Compromise Detection from Brute Force Attack’. Then add the panel to your dashboard.

Query – _sourceCategory=Labs/AWS/CloudTrail
| json field=_raw "userIdentity.userName" as actor
| json field=_raw "eventType" as event_type
| json field=_raw "sourceIPAddress" as src_ip
| json field=_raw "responseElements.ConsoleLogin" as result
| json field=_raw "eventName" as event_name
| where event_type matches "{{event_type}}"
| where actor matches "{{actor}}"
| timeslice 1h
| if(result = "Success",1,0) as success_count
| if(result = "Failure",1,0) as failure_count
|sum(success_count) as Success, sum(failure_count) as Failure by _timeslice,actor,event_type
| where Failure>0
*| Failure/Success*100 as Failure_percent*
| order by Failure_percent desc
| format("%.2f", Failure_percent) as Failure_percent

Account Compromise Detection from Brute Force Attack

	Time	actor	event_type	Success	Failure	Failure_percent
1	06/23/2025 16:00:00	dtaylor	AwsConsoleSignIn	3	2	66.67
2	06/23/2025 07:00:00	Ankit Goel	AwsConsoleSignIn	3	2	66.67
3	06/23/2025 09:00:00	kevin	AwsConsoleSignIn	3	1	33.33
4	06/23/2025 03:00:00	kevin	AwsConsoleSignIn	3	1	33.33
5	06/23/2025 11:00:00	kevin	AwsConsoleSignIn	2	2	100.00
6	06/23/2025 02:00:00	kevin	AwsConsoleSignIn	2	2	100.00
7	06/22/2025 21:00:00	gosia	AwsConsoleSignIn	2	1	50.00
8	06/22/2025 18:00:00	kevin	AwsConsoleSignIn	2	1	50.00
9	06/23/2025 06:00:00	Ankit Goel	AwsConsoleSignIn	2	1	50.00
10	06/23/2025 10:00:00	bcleveland	AwsConsoleSignIn	2	1	50.00

7. Detecting a Landspeed Violation

A "landspeed violation" occurs when a user logs in from one IP address and then logs in again shortly after from a second IP address located an unrealistically long distance away. For example, if a user logs in from Mexico City and then logs in again just 4 hours later from Moscow, it's physically impossible to travel that distance in such a short time. This kind of activity strongly suggests compromised credentials or unauthorized access, as it’s unlikely to be legitimate behavior.
Click to ‘Add Panel’ and choose the ‘Time Series’ option. As the query is quite long, let's explain it bit by bit.

In the first section, we specify that we look for the logs from the CloudTrail, parsing out the IP addresses and usernames. We tie the username to our ‘actor’ variable and look only for the public IP addresses. Then we look for the first occurrence of each user’s login and sort them in ascending order (the oldest login will be first).

```
Query – _sourceCategory=Labs/AWS/CloudTrail
| json "userIdentity.userName","sourceIPAddress" as user, ip nodrop
| where user matches "{{actor}}"
| where isPublicIP(ip)
| min(_messagetime) AS login_time BY user, ip
| sort BY user, +login_time
```

In this section, we convert IP addresses to their decimal form and iterate through each user's login history to retrieve both the most recent and the previous login timestamps.

```
| ipv4ToNumber(ip) AS ip_decimal
| backshift ip_decimal BY user
| backshift login_time AS previous_login
| where !(isNull(_backshift))
```

This part of the query will convert the decimal IP address back to an IPv4 address

```
| toInt(floor(_backshift/pow(256,3))) AS octet1
| toInt(floor((_backshift - (octet1 * pow(256,3)))/pow(256,2))) AS octet2
| toInt(floor((_backshift - (octet1 * pow(256,3) + octet2 * pow(256,2)))/256)) AS octet3
| toInt(_backshift - (octet1 * pow(256,3) + octet2 * pow(256,2) + octet3 * 256)) AS octet4
| concat(octet1, ".", octet2, ".", octet3, ".", octet4) AS previous_ip
```

This section of the query is going to look up the country information for each IP address. We will filter out any NULL values.

```
| lookup latitude AS lat1, longitude AS long1, country_name AS country_name1 FROM
geo://location ON ip
| lookup latitude AS lat2, longitude AS long2, country_name AS country_name2 FROM
geo://location ON ip=previous_ip
| where !(isNull(lat1)) and !(isNull(long1)) and !(isNull(lat2)) and !(isNull(long2))
```

In this section of code, we will be using the Haversine function to calculate the Kilometers in distance between the two locations.

```
| haversine(lat1, long1, lat2, long2) AS distance_kms
```

Next, we calculate the time difference between the most recent and previous login events. Using the geographic distance between the two login locations, we then determine the speed at which the user would have had to travel to log in at both locations within that time frame.

```
| (login_time - previous_login)/3600000 AS login_time_delta_hrs
| distance_kms / login_time_delta_hrs AS apparent_velocity_kph
```

| where apparent_velocity_kph > 0

Add the speed threshold by which the calculated speed above would be considered "suspicious" or "unrealistic."
| 500 AS suspicious_speed
| where apparent_velocity_kph > suspicious_speed

This last section of code will clean up the results and format them for better presentation on our dashboard
| concat(ip, ", ", previous_ip) AS ip_addresses
| if(country_name1 <> country_name2, concat(country_name1, ",", country_name2), country_name1) AS countries
| fields user, ip_addresses, countries, distance_kms, login_time_delta_hrs, apparent_velocity_kph
| where !isNull(user)
| where apparent_velocity_kph != "Infinity"
| sort by apparent_velocity_kph

Under the chart type, choose 'Table', rename the panel to 'Landspeed Violation', and click 'Add to Dashboard'.

Landspeed Violation

	user	ip_addresses	countries	distance_kms	login_time_delta_hrs	apparent_velocity_kph
1	mike	119.204.147.197, 201.82.28.45	Korea (south),Brazil	18,375.05	2.7777777777777776e-7	66,150,180,000
2	bcleveland	191.197.193.135, 223.18.219.39	Brazil,Hong Kong	18,363.14	2.7777777777777776e-7	66,107,304,000
3	gosia	13.125.38.187, 200.153.155.221	Korea (south),Brazil	18,339.75	2.7777777777777776e-7	66,023,100,000.00001
4	kevin	199.186.27.217, 89.145.138.12	Chile,Russian Federation	16,914.49	2.7777777777777776e-7	60,892,164,000.00001
5	athomas	85.145.46.4, 13.70.109.148	Netherlands,Australia	16,700.32	2.7777777777777776e-7	60,121,152,000
6	Ankit Goel	161.20.158.131, 220.239.174.110	Switzerland,Australia	16,637.14	2.7777777777777776e-7	59,893,704,000
7	gosia	86.180.4.163, 203.25.23.53	United Kingdom,Australia	16,537.31	2.7777777777777776e-7	59,534,316,000.00001
8	gosia	14.203.43.234, 80.6.233.108	Australia,United Kingdom	16,490.6	2.7777777777777776e-7	59,366,160,000
9	athomas	171.136.247.114, 58.170.201.193	United States,Australia	16,455.61	2.7777777777777776e-7	59,240,196,000.00001
10	suraj	137.219.197.240, 185.204.142.108	Australia,France	16,233.71	2.7777777777777776e-7	58,441,356,000

8. Using Crowdstrike Data

To assess whether any logged IP addresses are associated with known threats or malicious activity, we leverage Sumo Logic's integration with CrowdStrike. This integration allows us to enrich our data by querying IP addresses, email addresses, URLs, and other entities against CrowdStrike's threat intelligence database.
Click 'Add Panel' and choose the 'Time Series' option. Use the following query, select the 'Line Chart', rename the panel to 'Crowdstrike Data', and add the panel to the dashboard.
Query - _sourceCategory=Labs/AWS/CloudTrail
| parse regex "(?<ip_address>\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})" multi
| where ip_address != "0.0.0.0" and ip_address != "127.0.0.1"
| lookup type, actor, raw, threatlevel as malicious_confidence from sumo://threat/cs on threat=ip_address

```
| where type="ip_address" and !isNull(malicious_confidence)
| if(isEmpty(actor), "Unassigned", actor) as Actor
| parse field=raw "\"ip_address_types\":[\"*\"]" as ip_address_types nodrop
| parse field=raw "\"kill_chains\":[\"*\"]" as kill_chains nodrop
| timeslice 1m
| count by _timeslice, Actor, malicious_confidence
| sort by _timeslice
```

Crowdstrike Data

	Time	actor	malicious_confidence	_count
1	06/23/2025 22:19:00	Unassigned	unverified	4
2	06/23/2025 22:14:00	Unassigned	unverified	18
3	06/23/2025 22:14:00	Unassigned	high	3
4	06/23/2025 22:14:00	VELVETCHOLLIMA	unverified	2
5	06/23/2025 22:14:00	Unassigned	medium	2
6	06/23/2025 22:14:00	ETHEREALPANDA	unverified	2
7	06/23/2025 22:14:00	SPECTRALKITTEN	medium	2
8	06/23/2025 22:06:00	SPECTRALKITTEN	unverified	2
9	06/23/2025 22:06:00	Unassigned	unverified	77
10	06/23/2025 22:06:00	RECESSSPIDER	unverified	2