# Splunk Data Compliance Pipelines for FSI Workshop

Lab Guide

## Overview

This lab guide contains the hands-on exercises for the **Splunk Data Compliance Pipelines for FSI** workshop.

Please follow the steps documented here to complete the exercises.

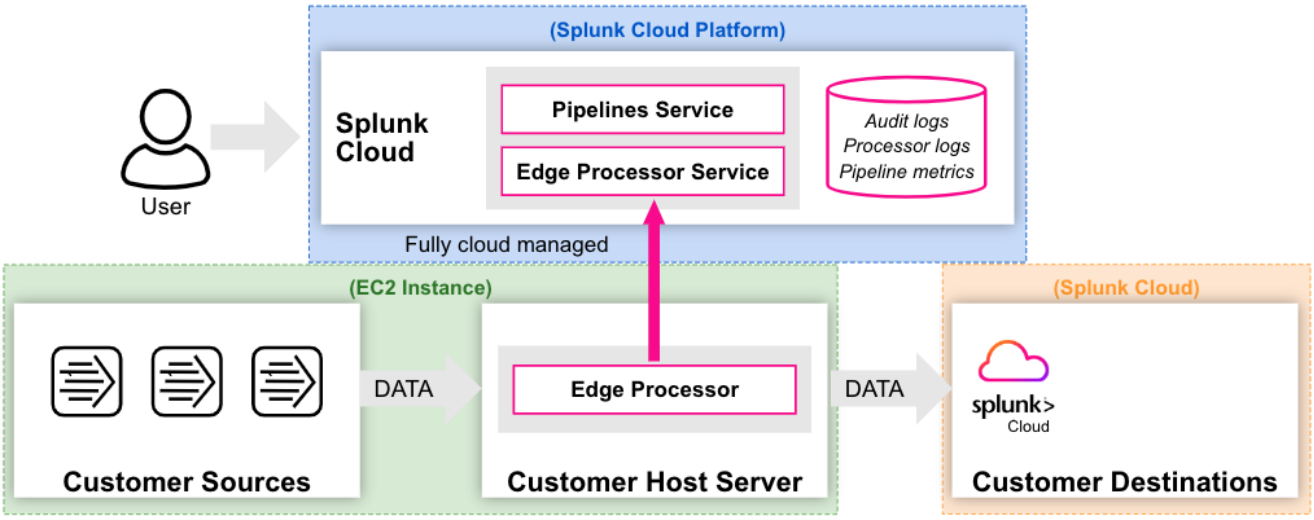
## Table of Contents

# Exercise 1 – Access Your Lab Environment

## Description

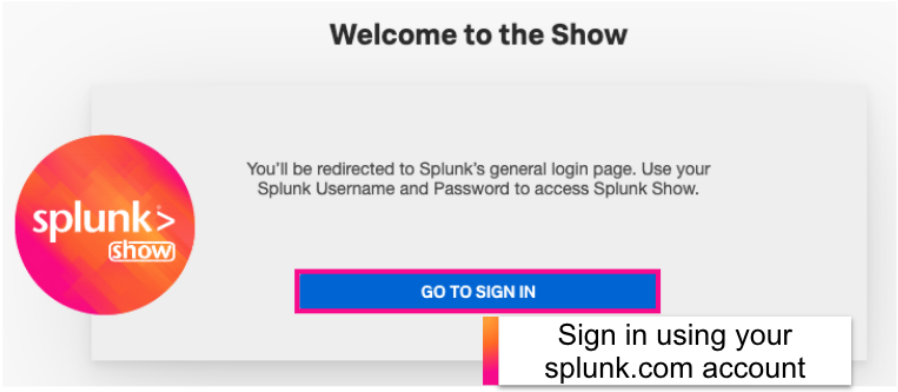
Before you can get hands-on with Edge Processor Service, first you'll need your own environment. Today's lab environment consists of the following:|

- **Splunk Cloud** - your data will be indexed here once it has been processed by Edge Processor.
- **Edge Processor service** - hosted in Splunk Cloud Platform, this is the cloud-based console through which you will setup, configure and manage Edge Processors, including creating pipelines for data processing (more on this later!)
- **EC2 instance (with demo data)** - this is where you will install a local Edge Processor node, which we will configure via the cloud-based Edge Processor service. Processed data will be forwarded to Splunk Cloud. In the lab environment this is referred to as '*EP - Edge Node*'.



## Log in to Splunk Show

1. Browse to https://show.splunk.com and log in using your **Splunk.com account**.



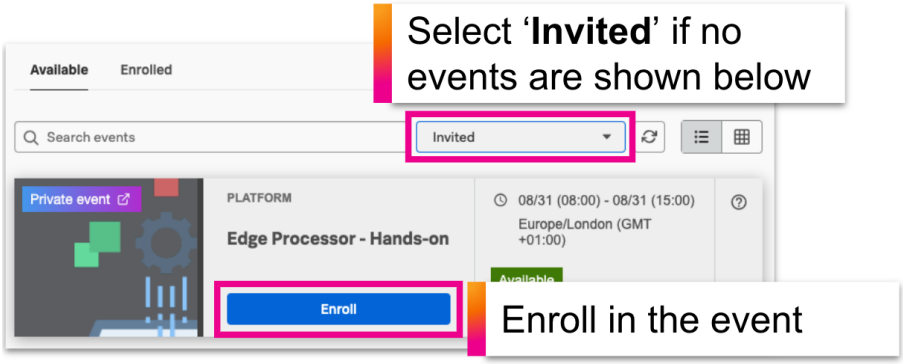

---

🔑 **Don't have a Splunk.com Account?**

To access our hands-on workshop events you will need a Splunk.com account. If you don't already have a Splunk.com account, don't worry - it only takes a few minutes to create one! Please create one here.

---

2. Once logged in to Splunk Show you will see the event page for the event that you have been invited to. If no events are listed, try selecting '**Invited**' from the dropdown list.
Click on **Enroll** to join the event.



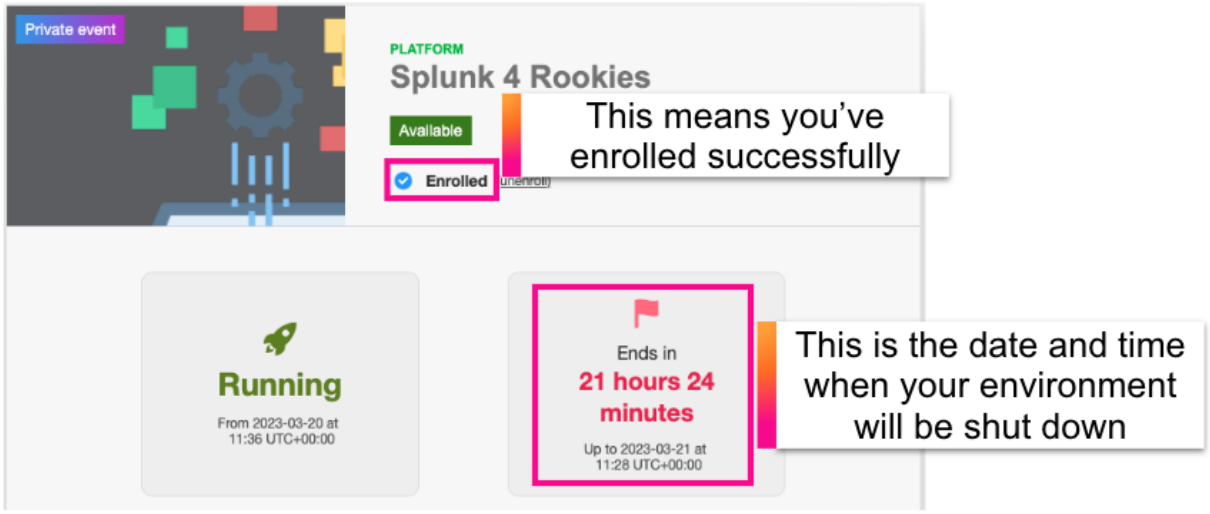

The page will refresh and the event will now display 'Enrolled'.

---

🕗 **Lab environment expiration**

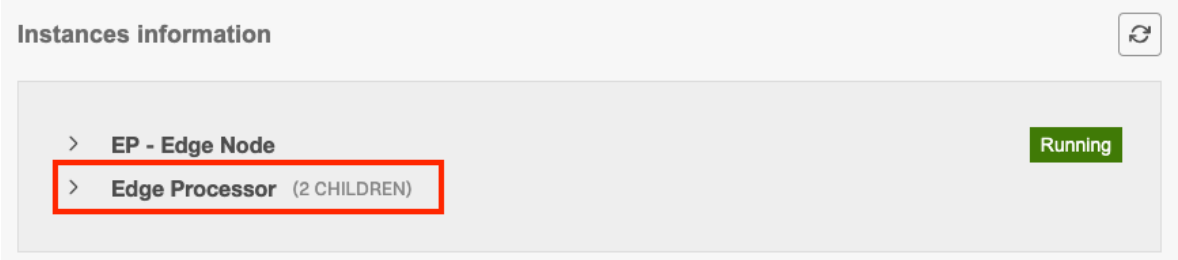
All Splunk environments that are part of this workshop event will automatically be shut down at the date and time specified on this screen so feel free to continue to play around with your lab environment until then!

---




## Access Your Splunk Cloud Instance

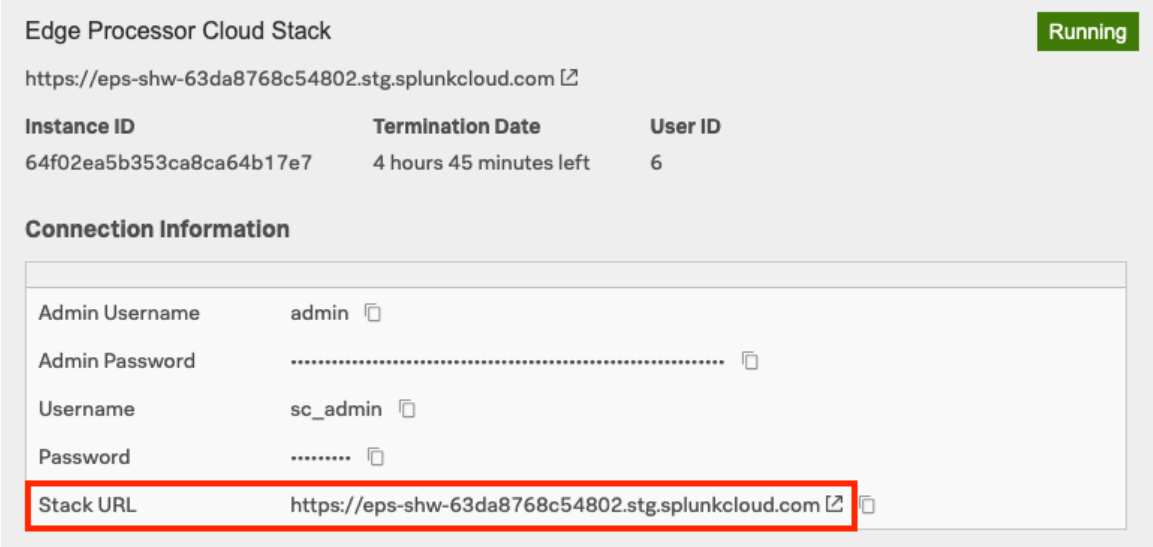
3. Browse to the Splunk Show event page and scroll down to the **Instances information** section. Expand out the '**Edge Processor**' section.



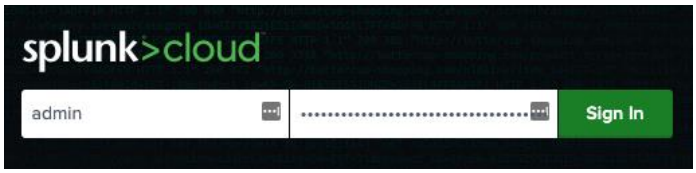

Here you will find credentials for:

- Your **Edge Processor Cloud Stack**, i.e. your Splunk Cloud stack; and
- Your **Edge Processor SCS Tenant**, i.e. the Edge Processor environment

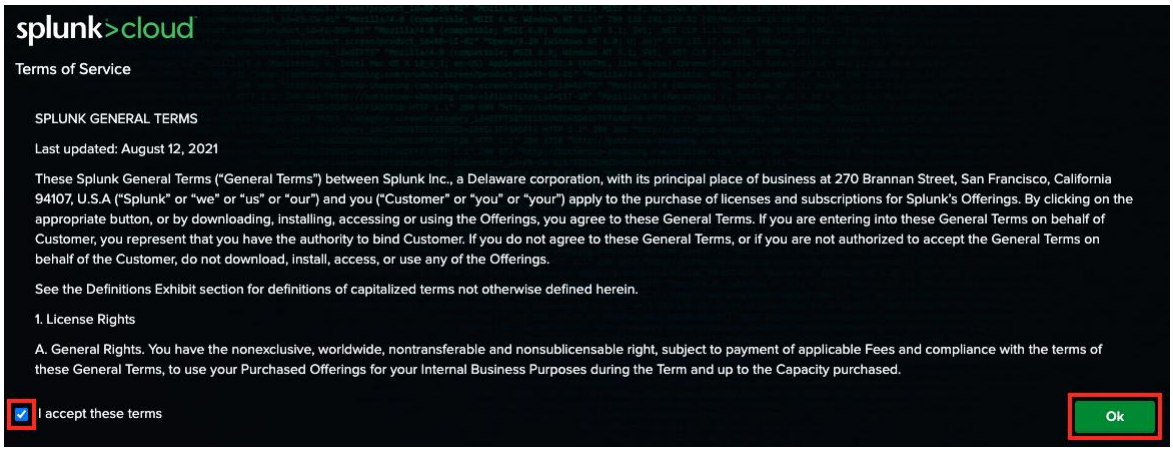
4. Locate the '**Edge Processor Cloud Stack**' section and click on the '**Stack URL**' link to open your Splunk Cloud stack.




Use the admin credentials provided here to login to your Splunk account.

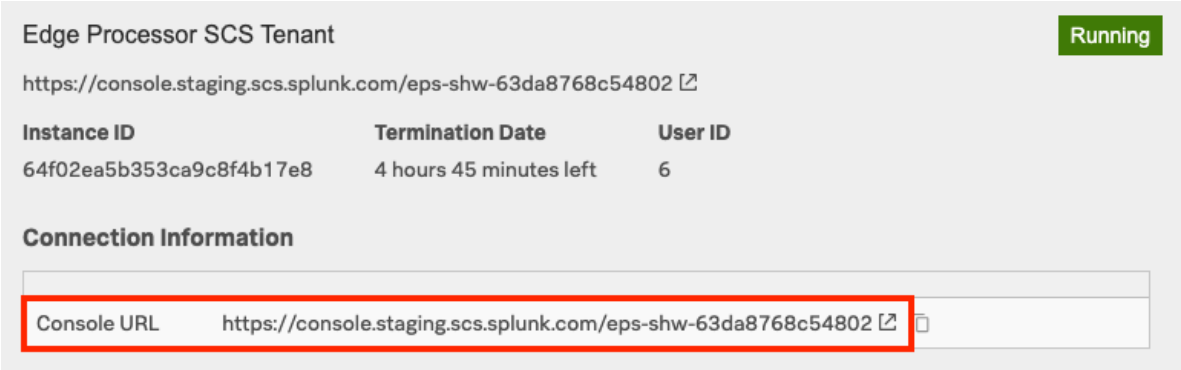



5. Accept the Terms of Service and click on **OK**.



## Access Edge Processor Service

6. On the Splunk Show event page, under the **Instances information** section, locate the '**Edge Processor SCS Tenant**' connection information and click on the link provided.



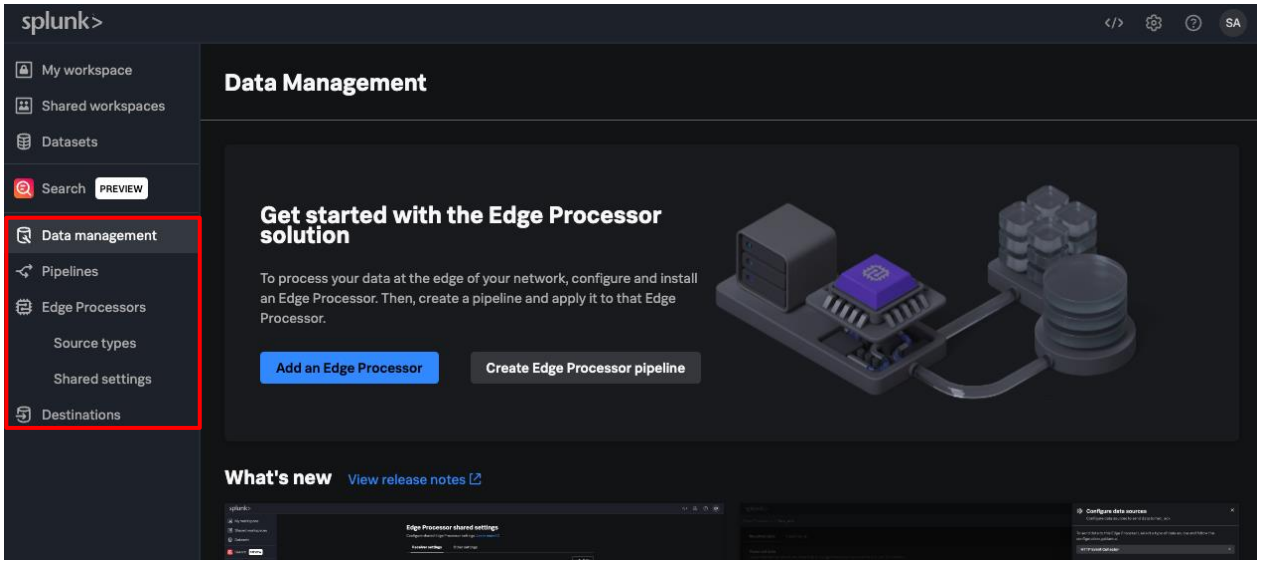

---

**Single Sign-On (SSO)**

Single Sign-on (SSO) is configured between the Edge Processor service ('SCS Tenant') and Splunk Cloud environments, so if you already logged in to your Splunk Cloud stack you should automatically be logged in to Edge Processor service. If you are prompted for credentials, use the credentials provided in the Edge Processor on Splunk Show event (listed under the 'Edge Processor Cloud Stack' section.)

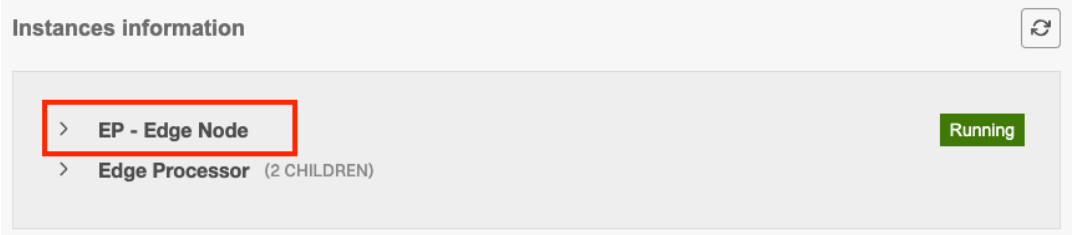
---

7. You will now be taken to the Edge Processor service environment. You can manage your Edge Processors via the menu on the left side of the page - we will create and configure these later in the workshop.

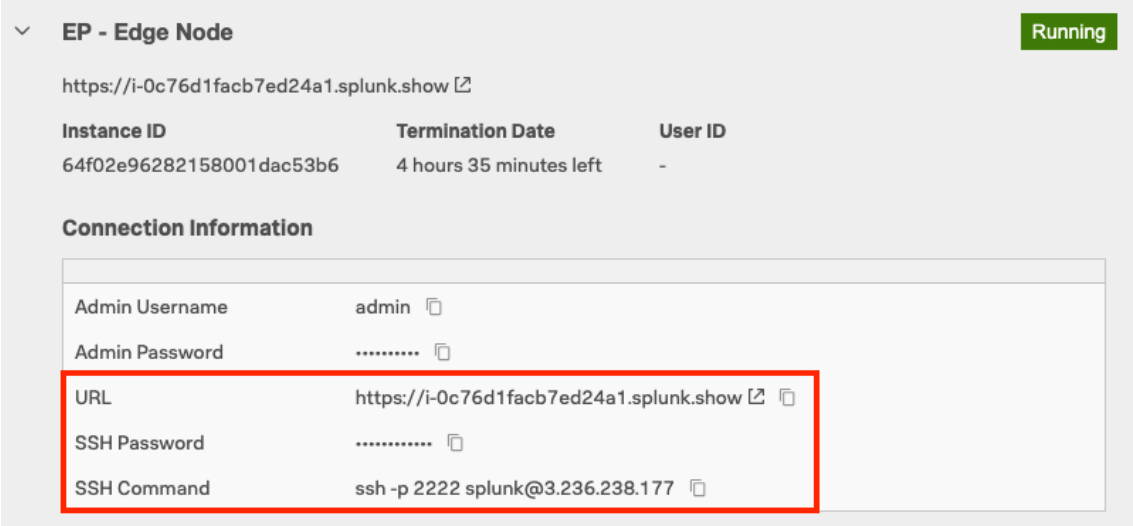


## Access Your EC2 Instance (EP - Edge Node)

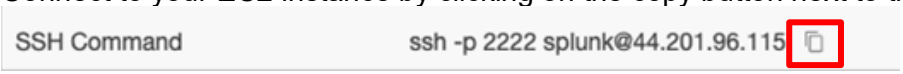
8. Browse to the Splunk Show event page and scroll down to the **Instances information** section. Expand out the '**EP - Edge Node**' section.




9. Locate the IP address and SSH details for your EC2 instance



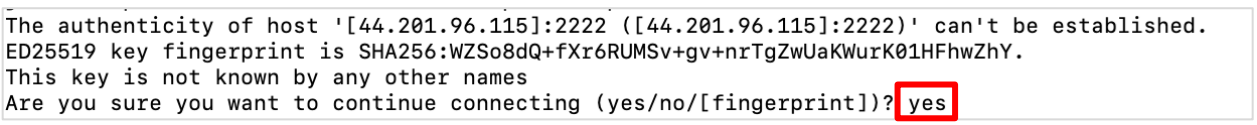

10. Connect to your EC2 instance by clicking on the copy button next to the SSH command.




11. Paste it into your terminal app of choice, such as Terminal (Mac) or PowerShell (Windows).

```
~ % ssh -p 2222 splunk@44.201.96.115
```

When prompted, type '**yes**' and hit enter to confirm the new SSH connection.




12. Copy the SSH password from the Splunk Show event by clicking on the copy button and pasting it into your terminal.

You should now have a green command prompt, meaning you are successfully connected to your EC2 instance.

```
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.15.0-1028-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue Aug 22 15:11:44 UTC 2023

  System load:  0.0                Processes:             205
  Usage of /:   3.7% of 193.81GB   Users logged in:       0
  Memory usage: 6%                 IPv4 address for ens5: 172.31.79.144
  Swap usage:   0%

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

 * Introducing Expanded Security Maintenance for Applications.
   Receive updates to over 25,000 software packages with your
   Ubuntu Pro subscription. Free for personal use.

     https://ubuntu.com/aws/pro

126 updates can be applied immediately.
84 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Wed Feb  8 19:24:19 2023 from 162.249.116.2
splunk@Domane-Demo-i-0e5ccec73c3e9c16b:~$
```

# Exercise 2 – Create an Edge Processor

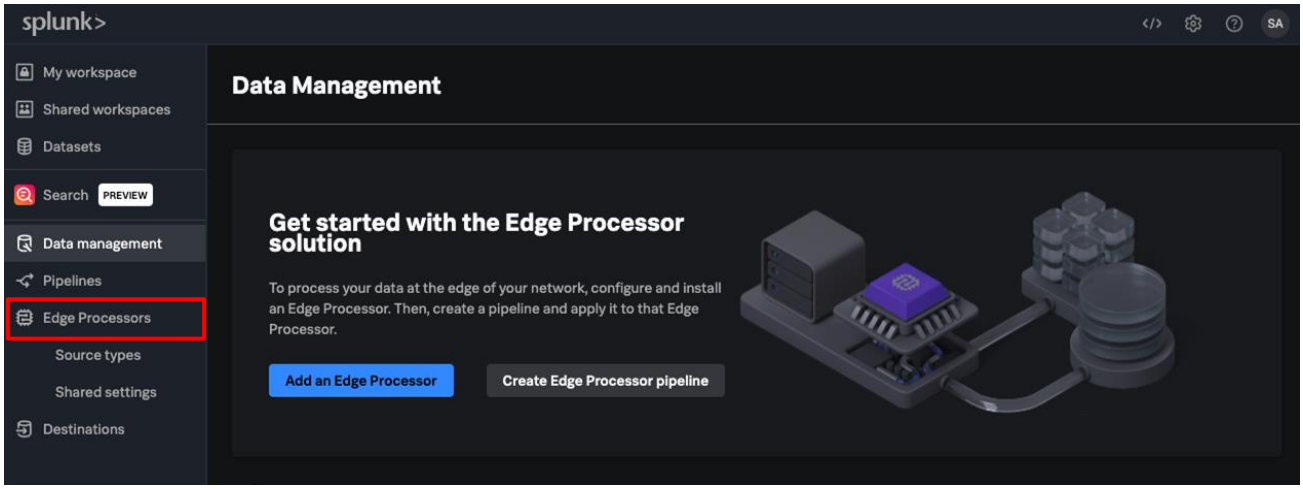
## Description

In this exercise you will create a new Edge Processor via the Edge Processor service user interface and then install a new Edge Processor instance on your EC2 server via the command line. This will allow us to create pipelines (to process data) later in the workshop.

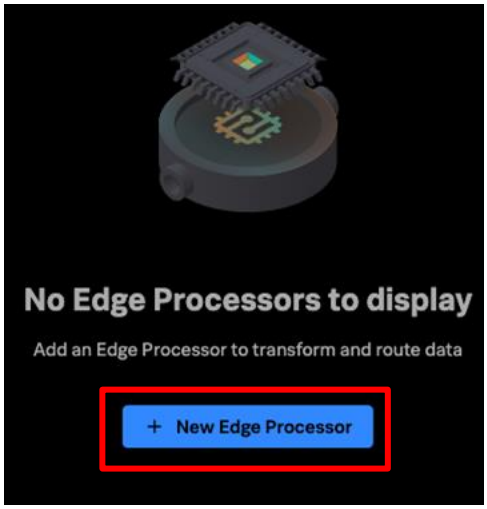
## Steps

### Add a New Edge Processor

1. Log in to the Edge Processor service (see Access Edge Processor Service) and from the menu on the left of the page click on **Edge Processors**.




2. In the middle of the **Edge Processors** page click on **+ New Edge Processor**.

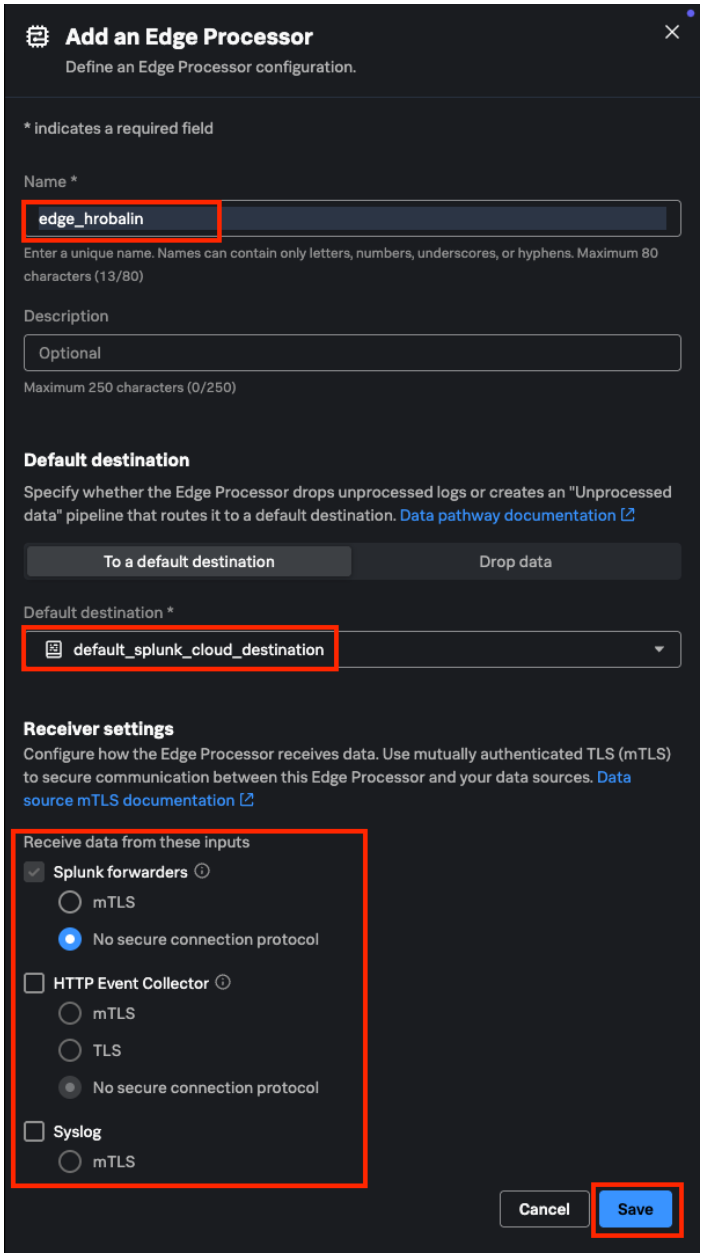


3. The **Add an Edge Processor** window will appear. Give your Edge Processor a name. Since each environment is shared among multiple people, use the name '**edge_**' followed by your own name to help identify this as your Edge Processor. For example: *edge_jsmith*

   For Default destination select '**default_splunk_cloud_destination**'.

   To help simplify our setup for this workshop, disable TLS for Splunk forwarders by checking the **No secure connection protocol option** (this will remove the checks against the options below). Uncheck the HTTP Event Collector and Syslog options.

   Click **Save** to create your Edge Processor.



4. A new window will appear. Under "**Instances**" information, you will see a clipboard containing the installation commands, which we can use to install an Edge Processor instance onto a Linux server. Copy this command by clicking on the icon on the bottom, you will see it says "**Copy to Clipboard**".


## Install a New Edge Processor Instance

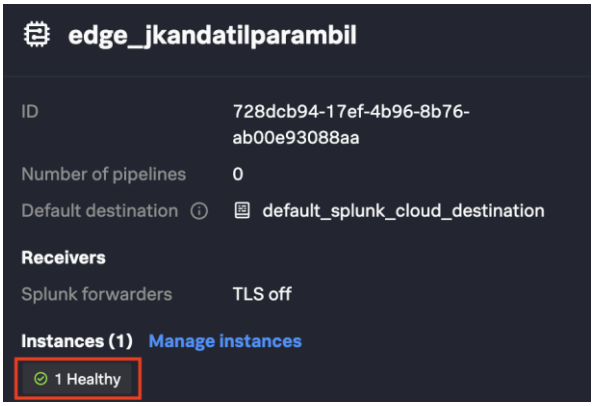
5. With your installation command already copied to your clipboard, open your SSH connection to your EC2 instance (see Access Your EC2 Instance) and paste the command. The command will run automatically.

   Due to the way the command is formatted, you will need to hit enter a second time after the initial command has run. You should now see a green prompt like before.

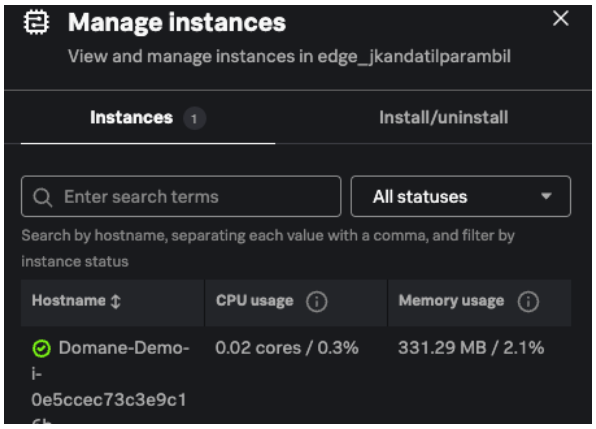
```
jkandatilparambil — splunk@Domane-Demo-i-0e5ccec73c3e9c16b: ~ — ssh -p 2222 splunk@44.201.96.115 — 111×34
>                     tar -xvzf splunk-edge.tar.gz
>
>                     # Create a configuration file that determines which Splunk Cloud Services (SCS) environme
nt, tenant, and Edge Processor cluster the instance gets associated with
>                     echo "groupId: c208c0c3-e19a-40cb-9e4a-ffdf5494b85e" > ./splunk-edge/etc/config.yaml
>                     echo "tenant: eps-shw-3c23343dca48c3" >> ./splunk-edge/etc/config.yaml
>                     echo "env: staging" >> ./splunk-edge/etc/config.yaml
>
>                     # Create a token file containing the authentication token that allows the Edge Processor
instance to connect to SCS
>                     echo "eyJhbGciOiJSUzI1NiIsImtpZCI6Inl5VW1ETlBmVTRaR1huRm4rQ3JIZitEWjlldlRDQVlwQXI2dTVGcUt
3bGsiLCJ0eXAiOiJKV1QifQ.eyJhdWQiOlsiYXBpOi8vc2NzLWRlZmF1bHQiXSwiY2lkIjoiMG9hOGcybWtxbndpNG5ka3gzNTciLCJlYyI6eyJ
zaWQiOiI2OTYwN2NlMTliNWRlM2RkYzBmNDkwMDNmMDA5YjRhMyIsInN1YiI6InNjX2FkbWluIn0sImV4cCI6MTY5Mjc2MDE4NSwiaWF0IjoxNj
kyNzE2OTg1LCJpZHAiOiJlcHNfc2h3XzNjMjMzNDNkY2E0OGMzLmVjLmlkcC5pZCIsImlzcyI6Imh0dHBzOi8vYXV0aC5zdGFnaW5nLnNjcy5zc
Gx1bmsuY29tIiwianRpIjoiY2ppY3ZlY2trbjBxODRpMWs5bDAiLCJzdWIiOiJzY19hZG1pbkBAZXBzLXNody0zYzIzMzQzZGNhNDhjMyIsInRl
bmFudCI6ImVwcy1zaHctM2MyMzM0M2RjYTQ4YzMifQ.R-DwmZqPq4efw7UaIijbJTeduCWV20hdJJxVc3d3Yxy3UCw6sq2gK6chW-sekSBOs8X8
MZrctcqWEOumAsI80q0LlbjssKNOAPWX2t9txj2xpGDtowOzt1GwpZ15K99wP70ZmYJYsSN-1S6HzDjtDs_NEjbStIDnqNUdcg9Op73IUKPv6ms
0tjzuWgQW7tUroQlwpkunck51FeQMk2T-dnLGJ1MYZILQ9833uoYDrf1_rkCfBMGFeGil7uI6Jau4_9Vn58JjlFr91DmBcfwCGfpzcB_vgYd1Cv
oNXZtAiIfvf8hsbsUaACJ8zBKPdlUaej_W0Fg7EnjJYPOciCZU8w" > splunk-edge/var/token
>
>                     # Install the Edge Processor instance
>                     mkdir -p ./splunk-edge/var/log
>                     nohup ./splunk-edge/bin/splunk-edge run >> ./splunk-edge/var/log/install-splunk-edge.out
2>&1 </dev/null &
>
>                     fi
splunk-edge/
splunk-edge/bin/
splunk-edge/bin/splunk-edge
splunk-edge/var/
splunk-edge/etc/
splunk-edge/etc/splunk-edge.service
[1] 75427
splunk@Domane-Demo-i-0e5ccec73c3e9c16b:~$
```

6. After a brief moment (it could take a couple of minutes in today's shared environment) you should see the number of instances associated with your Edge Processor turn to **(1)** with the status of 1 Pending. This will eventually change to 1 Healthy.

   You may need to refresh your browser page to see the status update.




   If you click on the blue **Manage instances** link you will see a list of instances, including the one you just installed. A green tick means the instance is healthy.



### Source types and Destinations

In a real world environment you would also need to configure source types and destinations (i.e. destinations where your data can be sent to) for your Edge Processor but to simplify today's workshop these have been pre-configured for you.

## Restart Your Splunk Heavy Forwarder (Manual Step for Today's Environment)

7. For today's lab you will need to manually restart the Splunk Heavy Forwarder that is running on your EC2 instance. Note that you would not need to do this in a production environment. Run the following command in the SSH session on your EC2 instance:

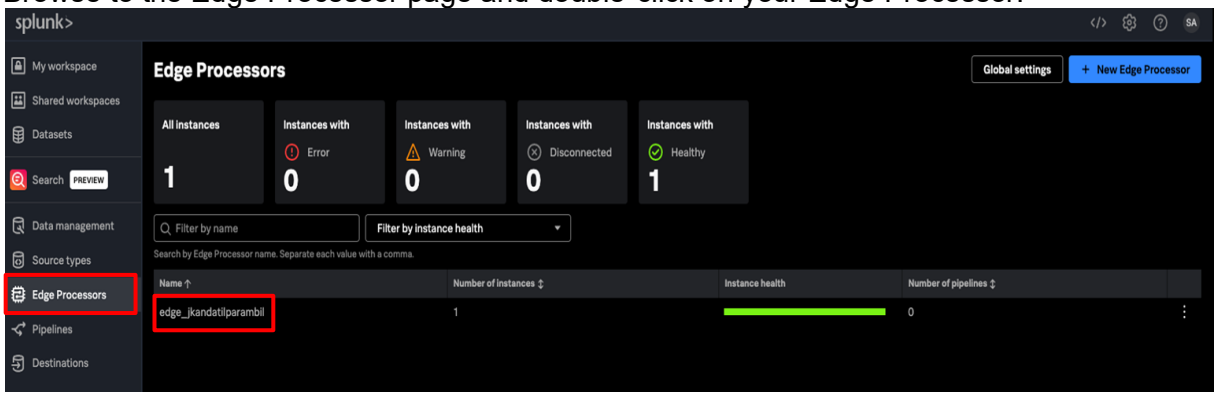
```
sudo /opt/splunk/bin/splunk restart
```

Confirm that Splunk restarts successfully:

```
If you get stuck, we're here to help.
Look for answers here: http://docs.splunk.com

The Splunk web interface is at https://Domane-Demo-i-03397b9fa7444e973

splunk@Domane-Demo-i-03397b9fa7444e973:~$
```

## Verify Edge Processor is Receiving Data

8. Browse to the Edge Processor page and double-click on your Edge Processor.




9. You have now successfully added the Edge Processor Node. We can now begin on configuring our destination locations.

# Exercise 3 – Configure an Amazon S3 Destination

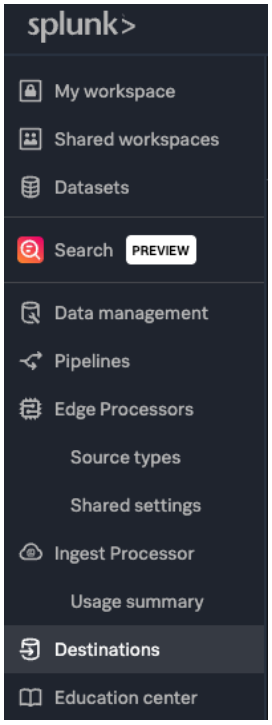
## Description

In this exercise you will add new destination to an Amazon S3 bucket. An Amazon S3 bucket can be used as an archival destination.

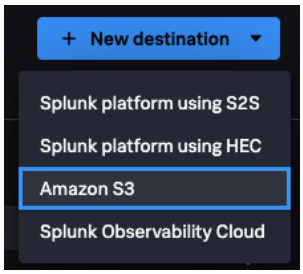
## Steps

### Add a new Amazon S3 destination

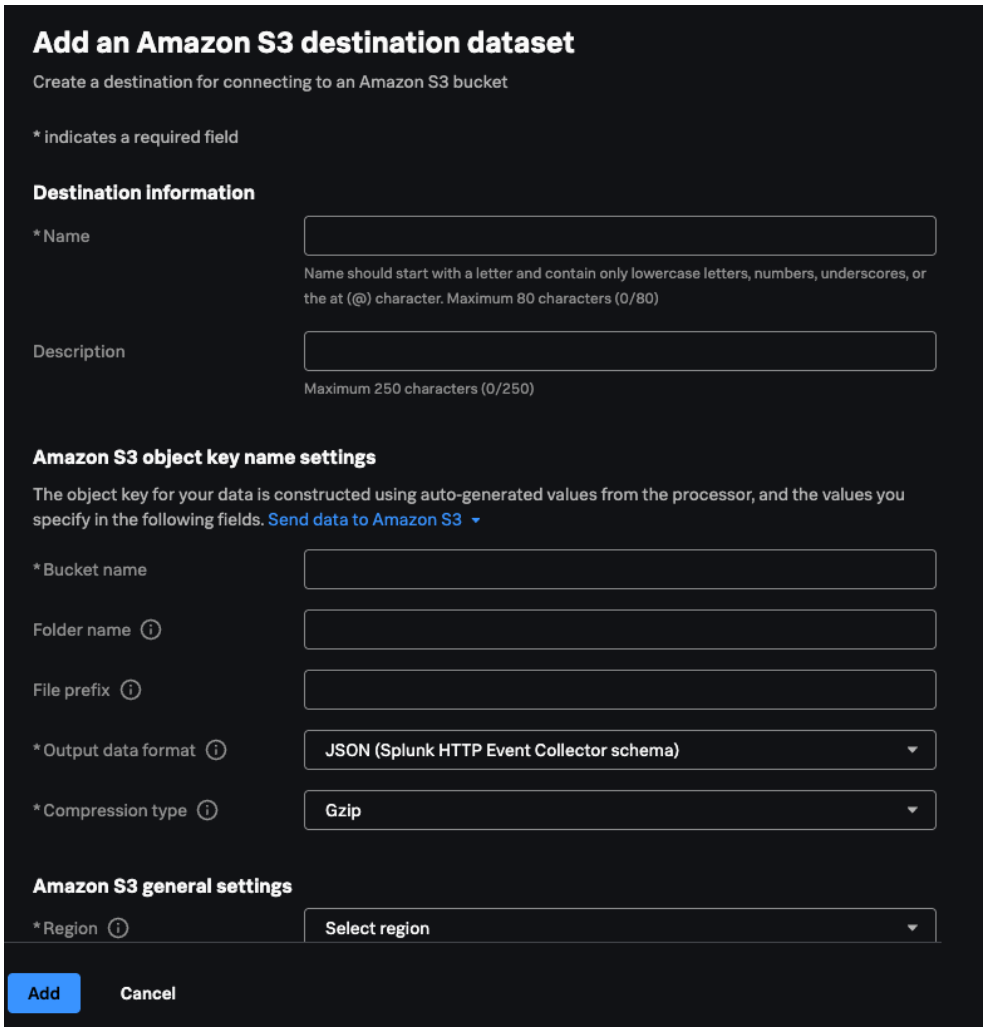
1. On the side menu on the left of the page click on **Destinations**



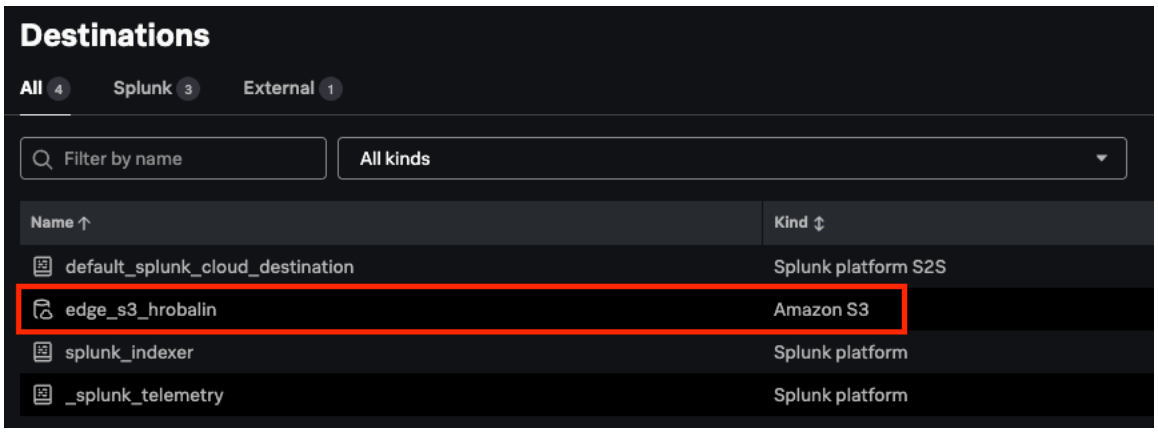

2. On the top right side, select on **+ New destination,** selecting **Amazon S3** from the dropdown.



3. A new panel will open where you will need to fill out your information. Once complete click **Add.**




4. You should now see that your new Amazon S3 destination has been added as a Destination for Edge Processor.



# Exercise 4 – Buttercup Enterprises: Global Regulations (KYC)

## Description

With your new Edge Processor installed and ready for configuration, it's time to create data pipelines to enable efficient processing at the edge.

The Buttercup Enterprises compliance team has requested that all data from Know Your Customer (KYC) applications be filtered to ensure regulatory compliance. This involves both archiving the data and making it available for analytics.

To meet these requirements, you will create a pipeline that filters incoming KYC data and routes it appropriately for secure storage and analysis.

---

⚠ **Downstream Impacts of Data Processing**

Applying masking, filtering and transforming rules to your data can impact downstream processing and analytics that rely on specific fields - such as IP addresses or usernames - to provide insights and intelligence. For example, Splunk Enterprise Security (ES) relies on IP addresses and usernames as part of its risk-based alerting (RBA) functionality, meaning that masking either of those fields would limit Splunk's ability to detect risk within your security environment.
Always check the potential impact of a pipeline before applying it!

---

## KYC Pipeline Guide: Global Regulations (Know Your Customer)

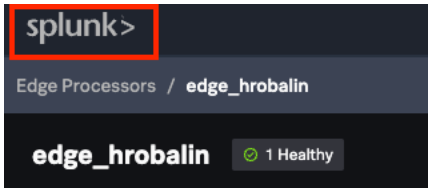
**Objective:** The Know Your Customer (KYC) pipeline is designed to process incoming KYC application data. Its primary goals are to:

1. **Archive Raw Data:** Send an unmodified copy of all raw KYC data to a secure Amazon S3 bucket for long-term storage.
2. **Sanitize PII:** Redact, mask, or hash Personally Identifiable Information (PII) such as full names, dates of birth, emails, phone numbers, street addresses, and document IDs, before the data reaches Splunk Cloud for analysis.
3. **Filter Relevant Events:** Keep only KYC events that have been "approved" (i.e., event_type == "kyc_approved" and event_status == "decision_approved").
4. **Route to Specific Index:** Send the processed and sanitized data to the kyc_events index in Splunk Cloud.
5. **Identify Your Data:** Add a custom field to easily identify your events in a shared Splunk Cloud environment.

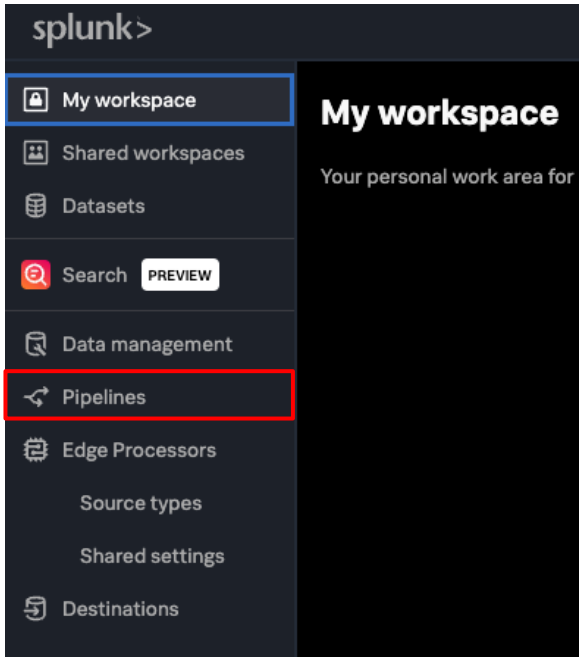
## Steps

## Create a New Pipeline

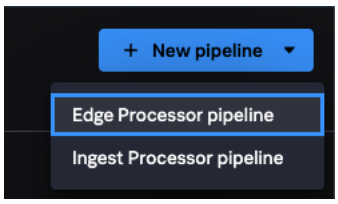
1. Log in to the Edge Processor service (see Access Edge Processor Service). If you're still on the page for your Edge Processor, you can navigate to the main page by clicking on the Splunk logo in the top left corner of the screen.



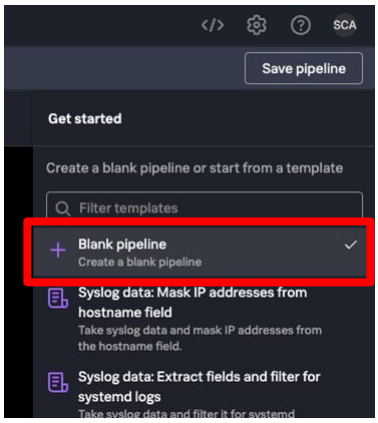

2. On the side menu on the left of the page click on **Pipelines**.



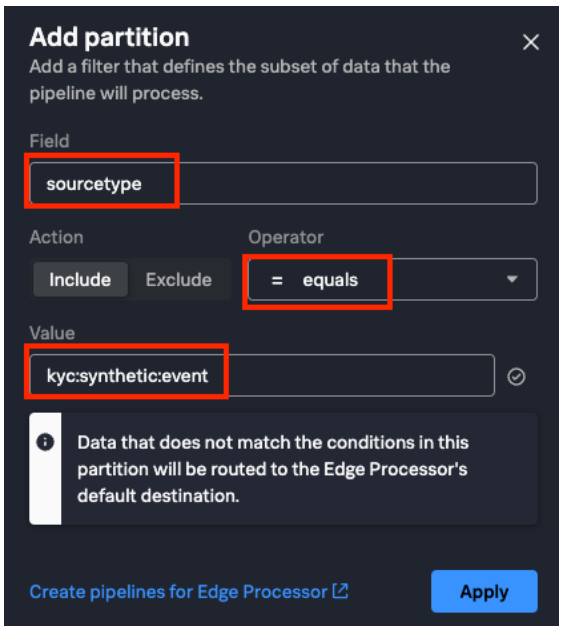

3. In the top right corner of the **Pipelines** page click on **+ New pipeline**. Select **Edge Processor pipeline**




4. On the right-hand side of the **New pipeline** page under the **Get started** pane select on **Blank pipeline** and click on **Next**.




5. On the 'Define your pipeline's partition' pane, click the add button ( ) next to the 'Partition' field. Configure the new partition rule by setting 'Field' to sourcetype, 'Action' to '**Include**', '**Operator**' to '**= equals**', and set **kyc:synthetic:event** as the '**Value**', then click '**Apply**'."



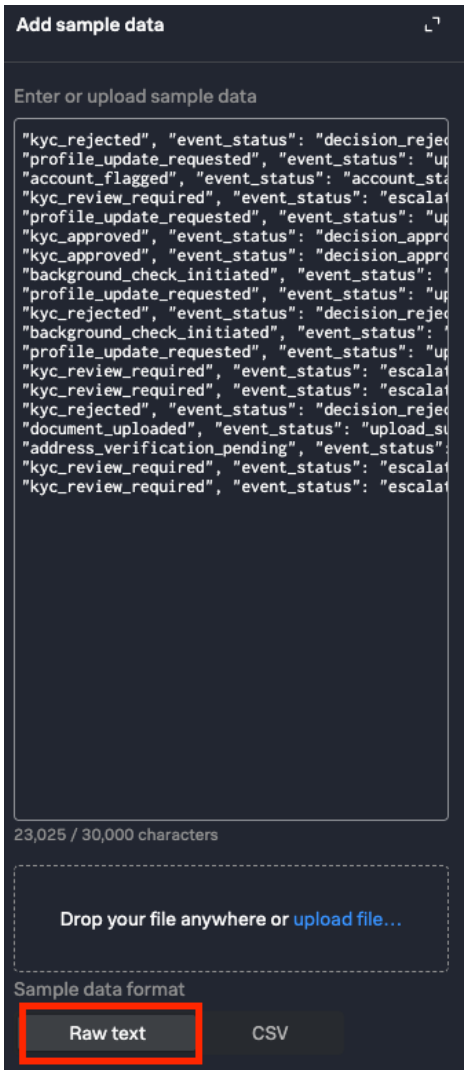

**Note:** Make sure you don't have an extra space.

And then Click on **Next** button at the bottom right.

6. An **Add sample data** menu will appear on the right side.

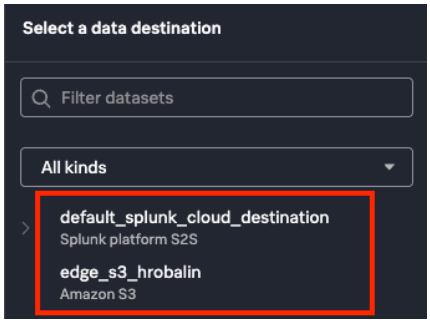
   Download the data-fsi-samples.zip file from https://splk.it/Data-FSI-Samples and extract the files to your computer. We will upload each of these samples during various exercises in this workshop.

7. Upload the **kyc_sample_events.jsonl** file that you extracted from the zip file earlier using the "**upload file…**" option at the bottom.

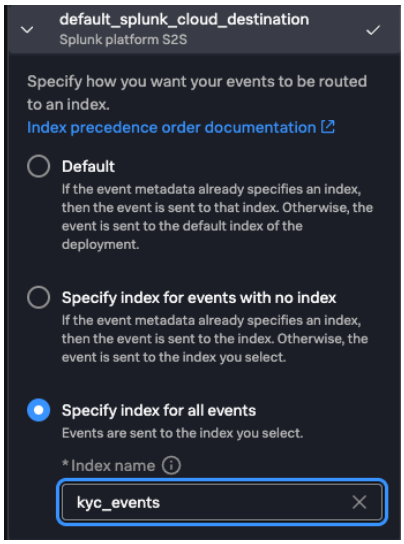



For the **Sample data format** select **Raw text** and click on **Next**.

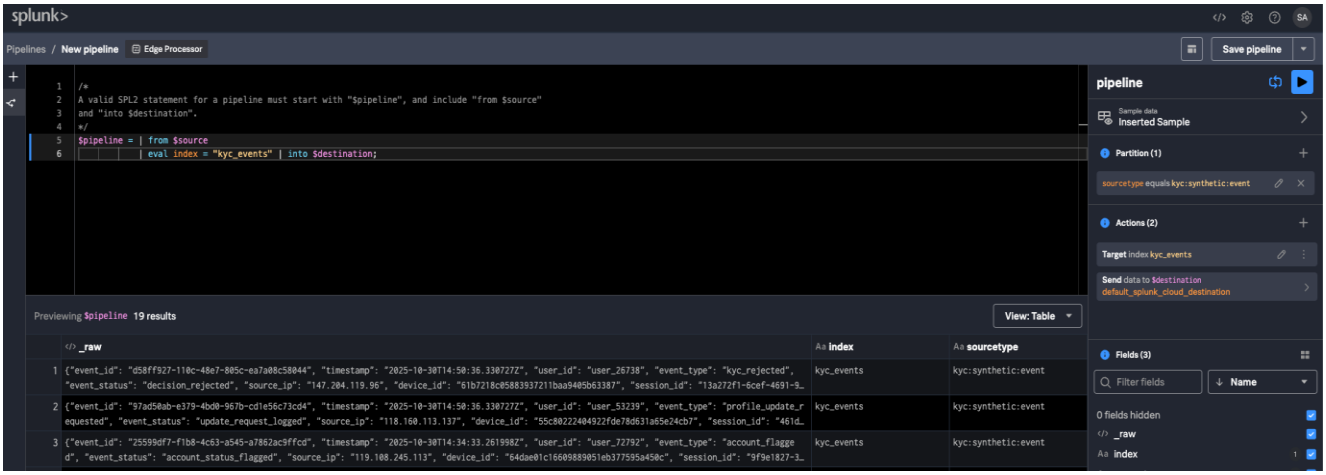
8. A "**Select a data destination"** menu will appear. Select the **default_splunk_cloud_destination** as your data destination.




9. A pane below will open, there you will select **Specify index for all events** and set the index to the **kyc_events** from the drop down, as requested, and click the "**Done**" button in the bottom.




10. A screen like below will appear:




---

As you integrate additional actions into your pipeline, please insert them between the existing pipeline stages and the final index. It is crucial to maintain the order specified in this guide. Refer to the example below:

```
$pipeline = | from $source
. . . ADD NEW ACTIONS HERE
| eval index = "kyc_events"
| into $destination;
```

---

## Define Archival Routing

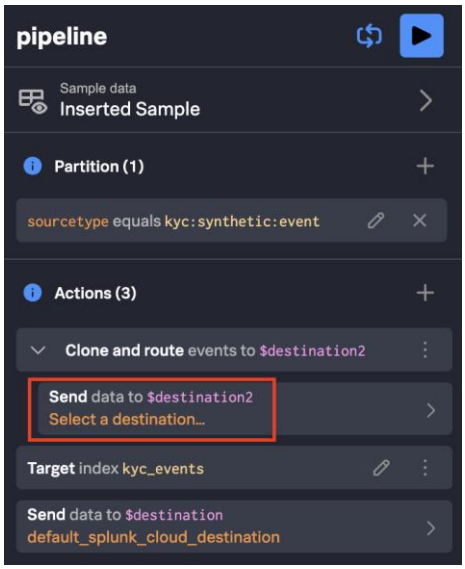
We will paste the following SPL2 statement. This will define Splunk as $destination, and Amazon S3 Bucket for Archiving as $destination2:

```
| thru [ | into $destination2 ]
```

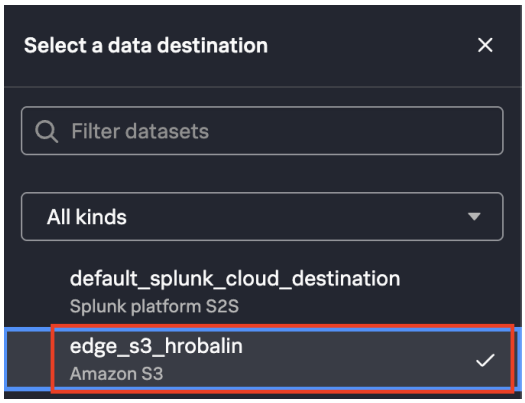
11. Your SPL2 Pipeline should now look like this:

```
$pipeline = | from $source
            | thru [ | into $destination2 ]
            | eval index = "kyc_events"
            | into $destination;
```

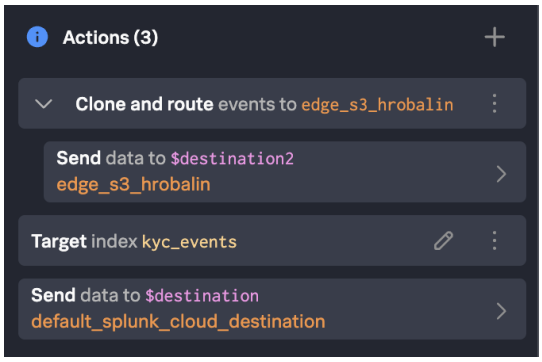
12. Now let's define in the pipeline setting which Amazon S3 Bucket will be $destination2. Click Select a destination




13. Select the Amazon S3 Endpoint for our use case: edge_s3_username.




14. Click **apply** at the bottom right.
15. **Actions** should now look like this, where $destination2 is S3 Destination:




## Define PII Sanitization (Masking, Hashing, Redaction)

16. Now we need to sanitize any of the data going into our Splunk Index. In our case since our data is a CSV (i.e. structured), we will use the replace function to replace data in our PII field:

    First, we will redact the Full Name using the eval command.

```
// Redact full_name within the 'pii' string
| eval _raw=json_set(_raw, "pii.full_name", "REDACTED_NAME")
```

17. Your SPL2 statement, which should now look like this:

```
$pipeline = | from $source

// Send raw data to secure archive (e.g., Amazon S3 Bucket)
| thru [ | into $destination2 ]

// Redact full_name within the 'pii' string
| eval _raw=json_set(_raw, "pii.full_name", "REDACTED_NAME")

// Set index for KYC events
| eval index = "kyc_events"

| into $destination;
```

**Regular Expression Syntax in Edge Processor**

Edge Processor supports Perl Compatible Regular Expressions (PCRE) syntax. When you use a command or function that uses a regular expression, such as the `rex` command or the `replace` evaluation function, you must write the regular expressions in PCRE syntax (see Supported regular expression syntax for more information). If you use a regex tester such as regex101 you will need to select '**PCRE2 (PHP >=7.3)**' from the FLAVOR menu on the left of the page in order to ensure your regular expressions will work with Edge Processor!


18. To redact the remaining PII fields we will copy, and paste the code sample below:

```
$pipeline = | from $source

            // Send raw data to secure archive (e.g., Amazon S3 Bucket)
            | thru [ | into $destination2 ]

            // Redact full_name within the 'pii' string
            | eval _raw=json_set(_raw, "pii.full_name", "REDACTED_NAME")

            // Redact date_of_birth within the 'pii' string
            | eval _raw = json_set(_raw, "pii.date_of_birth", "REDACTED_DOB")

            | eval masked_email = replace(_raw.pii.email, /^(.)(.*)(@.*)$/, "$1****$3")
            | eval _raw=json_set(_raw, "pii.email", masked_email)

            // Mask phone_number within the 'pii' string
            | eval masked_phone = replace(_raw.pii.phone_number,
/^(.{3})(.*)(.{4}[xX\d]*)$/, "$1***$3")
            | eval _raw=json_set(_raw, "pii.phone_number", masked_phone)

            // Redact street_address within the 'address' string
            | eval _raw=json_set(_raw, "address.street_address", "REDACTED_ADDRESS")

            // Extract document_id and replace with hash for all documents
            | eval document_id_0 = json_extract(_raw, "documents{0}.document_id")
            | eval hash_doc_id_0 = md5(document_id_0)
            | eval _raw = json_set(_raw, "documents{0}.document_id", hash_doc_id_0)

            | eval document_id_1 = json_extract(_raw, "documents{1}.document_id")
            | eval hash_doc_id_1 = md5(document_id_1)
            | eval _raw = if(isnotnull(document_id_1), json_set(_raw,
"documents{1}.document_id", hash_doc_id_1), _raw)
```

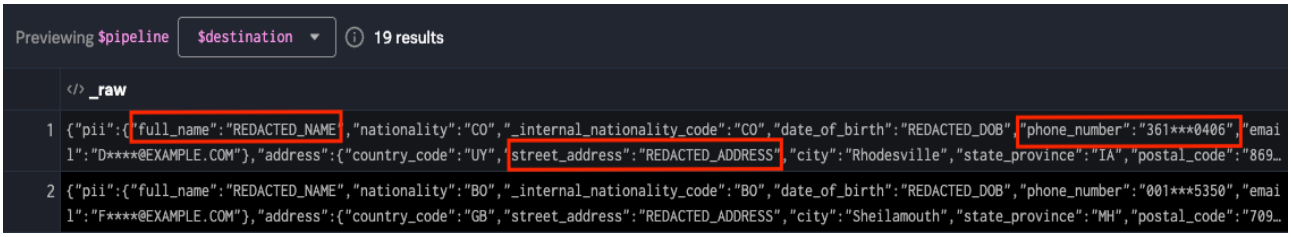
```
            // // Drop the temp field
            | fields - masked_phone, masked_email, hash_doc_id, document_id_0,
hash_doc_id_0, document_id_1, hash_doc_id_1

            // Set index for KYC events
            | eval index = "kyc_events"

            | into $destination;
```

19. Below is an example of how we can extract a value, create a hash from it, and write it back into the event:

```
// Extract document_id and replace with hash for all documents
| eval document_id = json_extract(_raw, "documents{0}.document_id")
| eval hash_doc_id = md5(document_id)
| eval _raw = json_set(_raw, "documents{0}.document_id", hash_doc_id)
```

## Preview Your Pipeline

20. Test your masking rule by clicking on the blue Preview pipeline button ( ) in the top right corner of the screen. If your pipeline filter is correct you should see some sample events containing “REDACTED” and Hash values inside of address and documents respectively.




### Sample Data

The results shown in the pipeline preview pane are from sample data that is uploaded when you configure your source types within the Edge Processor Service UI. They are not live events from your data stream.

## Filtering of Non-Actionable Data

21. We can filter out data that is not required for our analytics, helping reduce data volume. An example of how to do this here is to filter for event_status where the decision is approved. This would look like the following in SPL2:

```
// Filter to only keep successful KYC event status and event type
| where json_extract(_raw, "event_type") == "kyc_approved" AND json_extract(_raw,
"event_status") == "decision_approved"
```

22. Our final SPL2 query to filter our KYC events should look like the following:

```
$pipeline = | from $source
            // Send raw data to secure archive (e.g., Amazon S3 Bucket)
            | thru [ | into $destination2 ]

            // Redact full_name within the 'pii' string
            | eval _raw=json_set(_raw, "pii.full_name", "REDACTED_NAME")

            // Redact date_of_birth within the 'pii' string
            | eval _raw = json_set(_raw, "pii.date_of_birth", "REDACTED_DOB")

            | eval masked_email = replace(_raw.pii.email, /^(.)(.*)(@.*)$/, "$1****$3")
            | eval _raw=json_set(_raw, "pii.email", masked_email)

            // Mask phone_number within the 'pii' string
            | eval masked_phone = replace(_raw.pii.phone_number,
/^(.{3})(.*)(.{4}[xX\d]*)$/, "$1***$3")
            | eval _raw=json_set(_raw, "pii.phone_number", masked_phone)

            // Redact street_address within the 'address' string
            | eval _raw=json_set(_raw, "address.street_address", "REDACTED_ADDRESS")

            // Extract document_id and replace with hash for all documents
            | eval document_id_0 = json_extract(_raw, "documents{0}.document_id")
            | eval hash_doc_id_0 = md5(document_id_0)
            | eval _raw = json_set(_raw, "documents{0}.document_id", hash_doc_id_0)
            | eval document_id_1 = json_extract(_raw, "documents{1}.document_id")
            | eval hash_doc_id_1 = md5(document_id_1)
            | eval _raw = if(isnotnull(document_id_1), json_set(_raw,
"documents{1}.document_id", hash_doc_id_1), _raw)

            // // Drop the temp field
            | fields - masked_phone, masked_email, hash_doc_id, document_id_0,
hash_doc_id_0, document_id_1, hash_doc_id_1

            // Filter to only keep successful KYC event status and event type
            | where json_extract(_raw, "event_type") == "kyc_approved" AND
json_extract(_raw, "event_status") == "decision_approved"

            // Set index for KYC events
            | eval index = "kyc_events"
            | into $destination;
```

## Identifying our Data

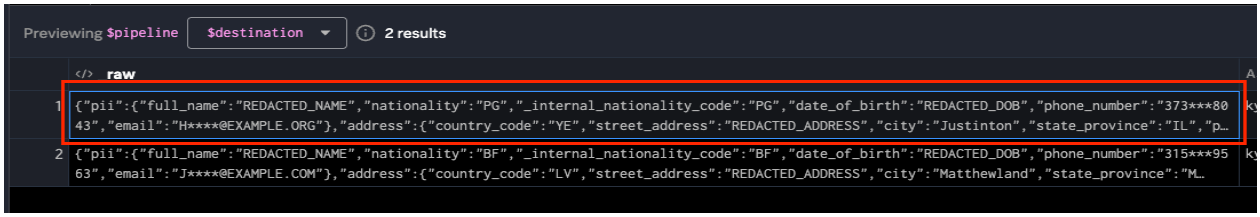
23. Finally, to help you identify data coming from your own pipeline when looking in your shared Splunk Cloud instance, add a custom indexed field to your events by adding an eval command to

your pipeline to create a field called ‘participant’ with the value set to your own name. Place this field right before we eval our index as kyc_events as seen below:

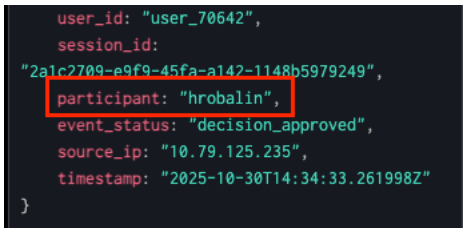
```
$pipeline =
| from $source
...
| eval _raw = json_set(_raw, "participant",  "hrobalin”) ← Update with your own name
| eval index = "kyc_events"
| into $destination;
```

To verify that the new field will be created, click on the Preview button ( ) again. Since we’re defining a new field=value pair as part of our pipeline, the new field will display in the Row preview table.

Select a field from the **_raw** column

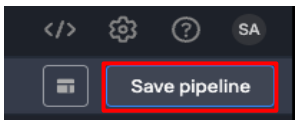



A modal will appear on the right side where the _raw event message will be expanded, if you scroll to the bottom, you will see the newly added participant field




## Save Your Pipeline

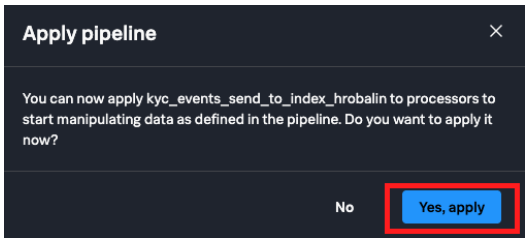
24. Save your pipeline by clicking on **Save pipeline** in the top right corner of the screen. Give your pipeline a suitable name, such as **kyc_events_send_to_index_<yourName>**. Be sure to include your name somewhere in the pipeline name to avoid confusion among those who are sharing the same environment as you!



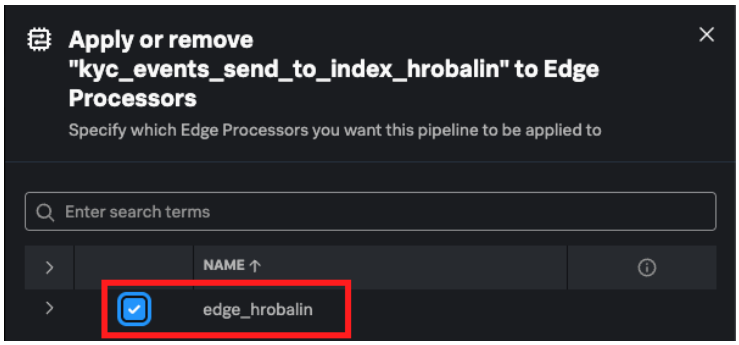

25. Click **Save** to save your pipeline.

## Apply the Pipeline to Your Edge Processor

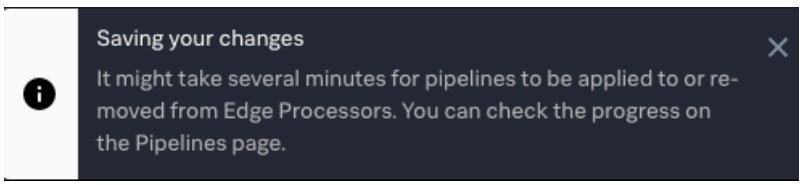
26. Now an “Apply Pipeline” dialog box will appear, click on “**Yes, apply**”.




27. A pane called “**Apply or remove <pipeline-name> to Edge Processors**” will appear, Select the edge processor you created earlier (see Add a New Edge Processor) and click on Save in the bottom.



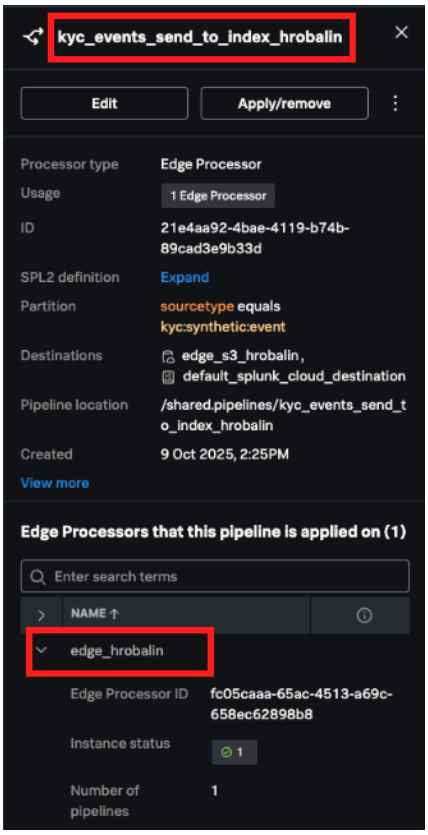

You will see a brief message stating that your changes are being saved. Note that it may take a few minutes before you see the effects of your pipeline in Splunk Cloud.



28. To check the status of your edge processor node as the pipeline is being applied, click on the click splunk> icon on the top left and then click on "**Pipelines**" on the home page and click on the pipeline created.

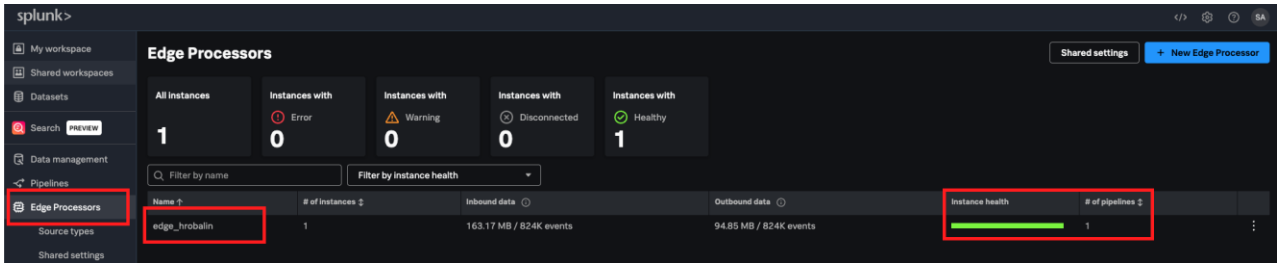
29. A panel will open up on the right of the page, showing the status of the Edge Processor instance that this pipeline has been applied to.

    Click the arrow (›) next to the instance name to view more information.




30. Once your pipeline has successfully applied the status of your instance should change to 1 Healthy.

    To verify that the instance is indeed healthy, click on **Edge Processors** in the menu on the left of the screen and refresh your web browser until your Edge Processor displays as Healthy.



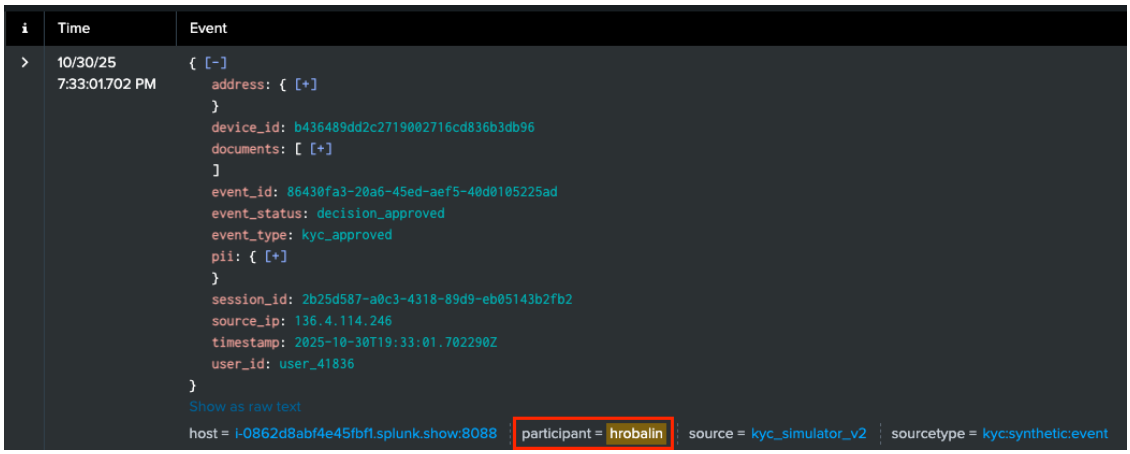

    Note that you may see the status change back to 1 Pending and even to 1 Error. However, your pipeline should still apply and eventually show as 1 Healthy.

## Check Your Data in Splunk Cloud

31. Log in to your Splunk Cloud instance (see Access Your Splunk Cloud Instance) and open the **Search & Reporting** app.

    Run the following search over the **last 15 minutes** and verify that you now see events containing '**root**' in the '**admin**' index:

    ```
    index=”kyc_events” participant=hrobalin
    ```
    ← **Remember to update the name!**




## Check Your Data in Amazon S3 Destination

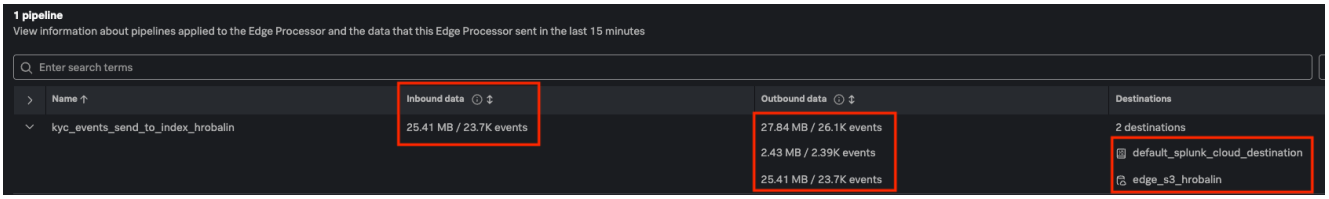
32. To access your Edge Processor, click "Splunk>" in the top-left corner. From the main screen, select "Edge Processors" from the left navigation pane.

    Double-click your desired Edge Processor to open its dashboard, which displays pipelines running on it. Locate your specific pipeline at the bottom of the screen and expand its details by clicking the '>' icon.

    Within the expanded pipeline view, you will see details regarding your Inbound Data, including its total size and the number of events. To the right, a breakdown illustrates how these events were processed and sent to various outbound destinations.

    Observe that the Amazon S3 destination's 'Size/NumofEvents' matches the inbound data. This is expected, as the S3 destination is configured to archive a full copy of all events.

    Conversely, the 'Size/NumofEvents' for the Splunk Cloud Destination is significantly lower. This reduction is intentional and results from pre-processing steps, such as filtering, designed to optimize the data sent to your Splunk Index.



# Exercise 5 - Buttercup Enterprises: AMERICAS Regulations (PCI DSS)

## Description

As Buttercup Enterprises operates internationally, it must comply with additional regulations governing data security. The next requirement is adherence to the Payment Card Industry Data Security Standard (PCI DSS), which focuses on protecting payment card information.

To ensure PCI DSS compliance, you will configure the Edge Processor to identify, filter, and securely route payment card data. This process will involve setting up a dedicated pipeline to isolate sensitive cardholder information, archive it securely, and prepare it for audit or analytics while maintaining strict security controls.

## PCI Pipeline Guide: AMERICAS Regulations (PCI DSS)

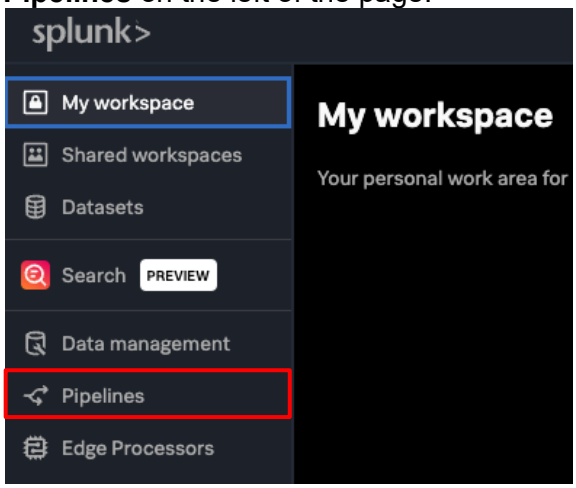
**Objective:** The Payment Card Industry Data Security Standard (PCI DSS) pipeline is designed to ensure compliance by securely handling payment card information. Its key objectives are to:

1. **Sanitize Sensitive Fields:** Redact cardholder names, and mask Primary Account Numbers (PANs) and account numbers according to PCI DSS requirements.
2. **Drop Non-Storable Fields:** Remove fields that are explicitly forbidden from being stored (e.g., CVV, track data, PIN).
3. **Route to Specific Index:** Send the processed and sanitized data to the pci_events index in Splunk Cloud.
4. **Identify Your Data:** Add a custom field to easily identify your events in a shared Splunk Cloud environment.

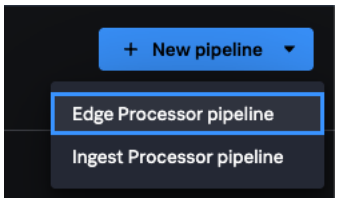
## Steps

### Create a New Pipeline

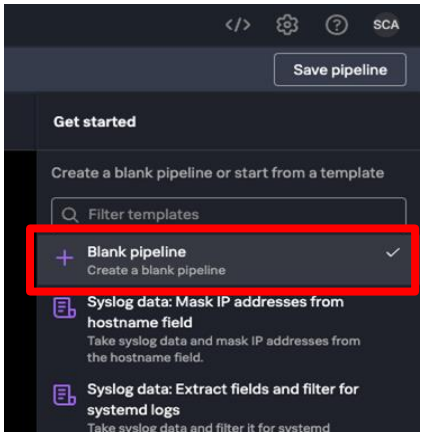
1. While logged in to the Edge Processor service (see Access Edge Processor Service) click on **Pipelines** on the left of the page.



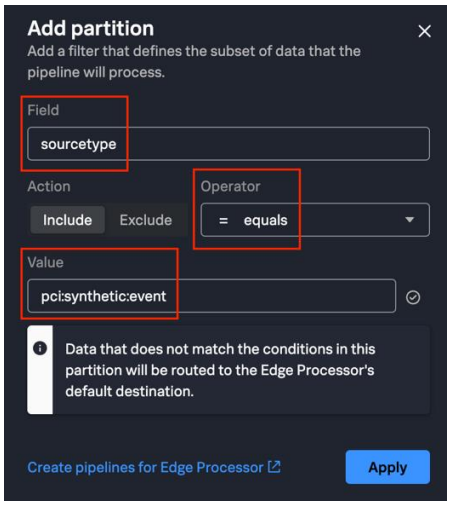
2. In the top right corner of the **Pipelines** page click on **+ New pipeline**. Select **Edge Processor pipeline**




3. On the right hand side of the **New pipeline** page under the **Get started** pane select on **Blank pipeline** and click on **Next**.



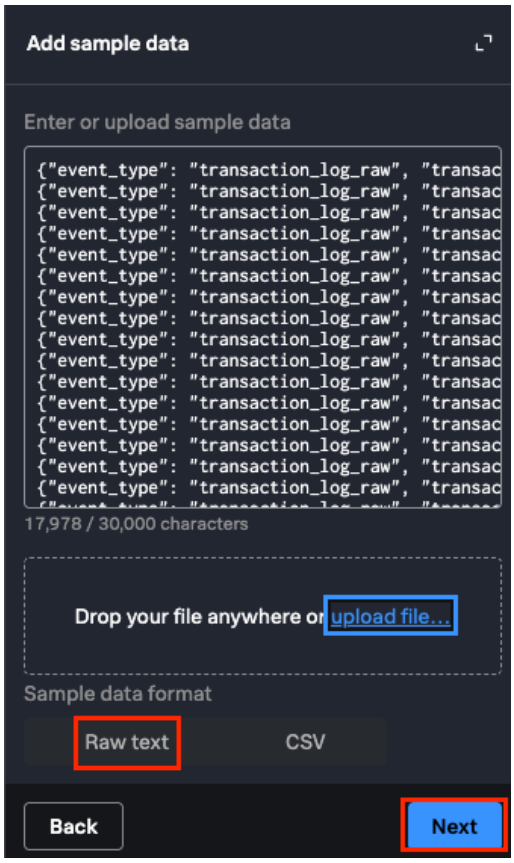

4. On the 'Define your pipeline's partition' pane, click the add button (  ) next to the 'Partition' field. Configure the new partition rule by setting 'Field' to sourcetype, 'Action' to 'Keep', 'Operator' to '= equals', and set **pci:synthetic:event** as the 'Value'."




**Note:** Make sure you don't have an extra space at the end.

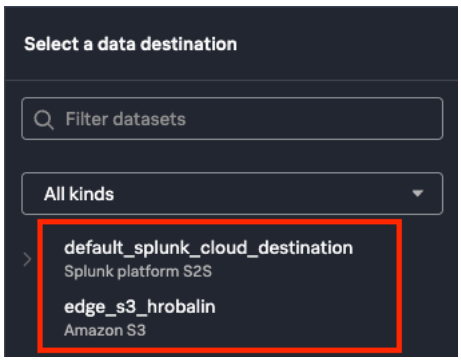
Click **Apply**, then click on **Next** at the bottom.

5. An **Add sample data** menu will appear on the right side. Upload the **pci_sample_events.jsonl** file that you extracted from the zip file earlier using the "**upload file…**" option at the bottom.



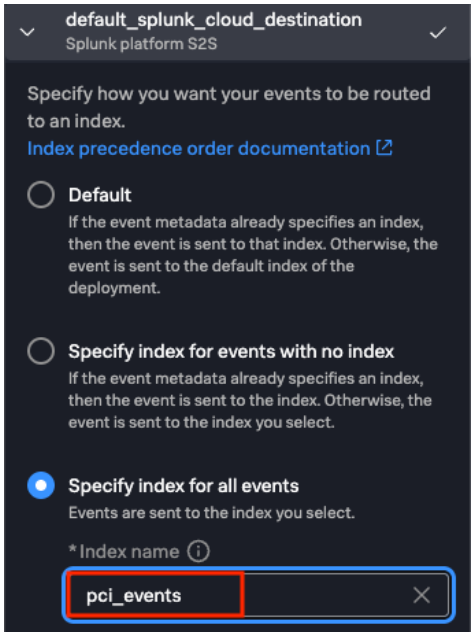

For the **Sample data format** select **Raw text** and click on **Next**.

6. A "**Select a data destination"** menu will appear. Select the **default_splunk_cloud_destination** as your data destination and click on **Next**.

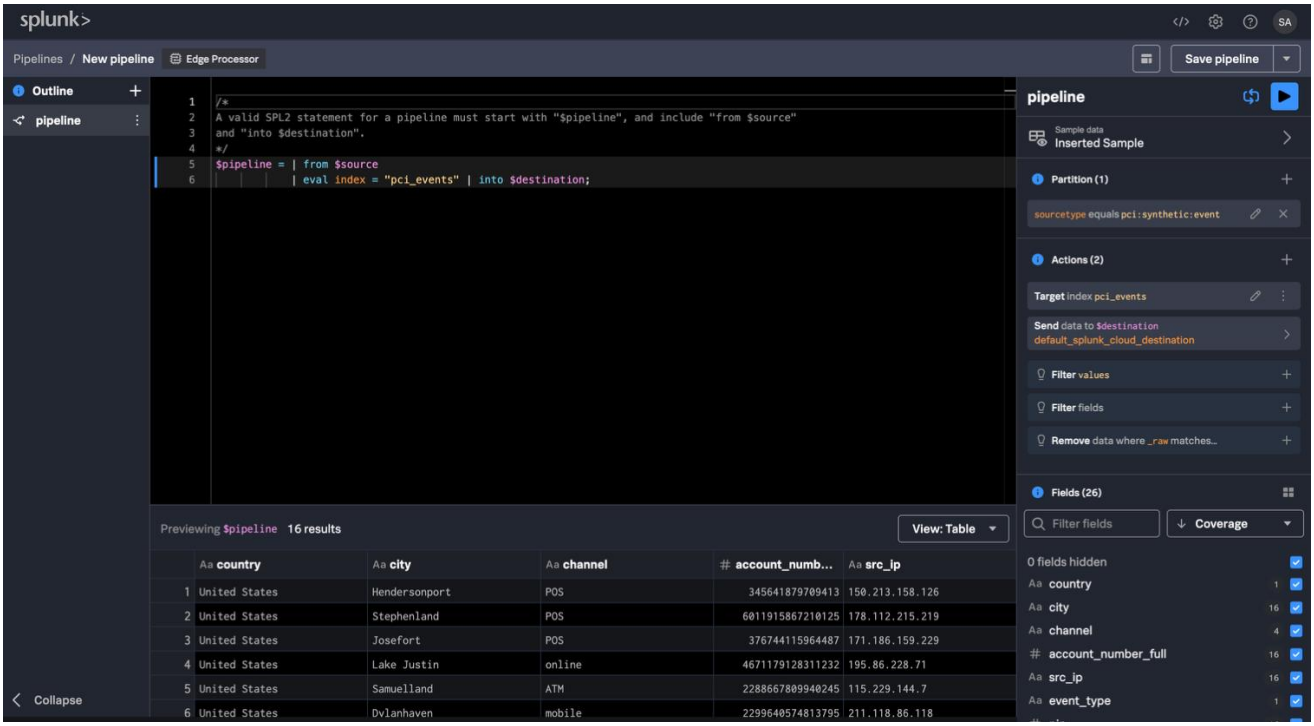


## Route Your Data to the 'pci_events Index

7. A pane below will open, there you will select **Specify index for all events** and set the index to the **pci_events** from the drop down, as requested by the Buttercup security team, and click on **Done** at the bottom.




8. A screen like below will appear:



As you integrate additional actions into your pipeline, please insert them between the existing pipeline stages and the final index. It is crucial to maintain the order specified in this guide. Refer to the example below:

```
$pipeline = | from $source
. . . ADD NEW ACTIONS HERE
| eval index = "pci_events"
| into $destination;
```

## Define PII Sanitization (Masking, Hashing, Redaction)

9. Now we need to sanitize any of the data going into our Splunk Index. In our case since our data is a CSV (i.e. structured), we will use the replace function to replace data in our PII field:

   First, we will redact the Full Name using the eval command.

```
// REDACT cardholder_name completely
| eval _raw = json_set(_raw, "cardholder_name", "REDACTED_NAME")
```

   Your SPL2 statement, should now look like this:

```
$pipeline =
| from $source

// REDACT cardholder_name completely
| eval _raw = json_set(_raw, "cardholder_name", "REDACTED_NAME")

// Set index for PCI events
| eval index = "pci_events"
| into $destination;
```

## Reviewing Fields

10. Let's review the remaining fields to see if we need to mask, redact, or hash. We will use the PCI Standard stated here to guide us:
    - **Primary Account Number (PAN)** – If storing, store only the first six and last four digits.
    - **Account Number** – If storing, mask except for last four digits (This is not the PAN).
    - **CCV** – Cannot be stored, must be dropped.
    - **Track1 Data & Track2 Data** – Cannot be stored, must be dropped.
    - **Pin, Pin Block** – Cannot be stored, must be dropped.

## Masking PAN and Account Number

11. Let's first mask the PAN:

```
// The PAN according to PCI DSS Requirement 3.4 (e.g., first 6, last 4 digits
visible)
| eval masked_pan = replace(_raw.pan, /(\d{6})(\d+)(\d{4})/, "$1******$3")
| eval _raw=json_set(_raw, "pan", masked_pan)
```

Now let's mask the Account Number:

```
// Mask account_number except last 4 digits, this is not the same as PAN
| eval masked_acct_number = replace(_raw.account_number, /(\d{8})(\d{4})/,
"XXXXXXXX$2")
| eval _raw=json_set(_raw, "account_number", masked_acct_number)
```

We have now both masked the PAN and Account Number.

12. We now need to drop our temporary fields created for the masking. We will do so below:

```
| fields - masked_pan, masked_acct_number
```

## Filtering of Non-Storable Fields

13. Part of PCI DSS Requirement 3.4 is not storing specific fields; we will do so below with the following:

```
// Drop sensitive fields that cannot be stored
| eval _raw=json_delete(_raw, "cvv", "track1_data", "track2_data", "pin",
"pin_block")
```

## PCI SPL Pipeline:

14. As you develop your SPL code, understand that the order of transformations is critical. While our current requirements specify a particular sequence, in other contexts, this order will be driven by downstream data consumption or conditional routing. Keep this in mind to ensure optimal filtering, processing, and enrichment across all your data pipelines.

15. The following is an example of how your Edge Processor Pipeline should look:

```
$pipeline =
| from $source

// REDACT cardholder_name completely
| eval _raw = json_set(_raw, "cardholder_name", "REDACTED_NAME")

// Mask the PAN according to PCI DSS Requirement 3.4 (e.g., first 6, last 4
digits visible)
```

```
| eval masked_pan = replace(_raw.pan, /(\d{6})(\d+)(\d{4})/, "$1******$3")
| eval _raw=json_set(_raw, "pan", masked_pan)

// Mask account_number except last 4 digits, this is not the same as PAN
| eval masked_acct_number = replace(_raw.account_number, /(\d{8})(\d{4})/,
"XXXXXXXX$2")
| eval _raw=json_set(_raw, "account_number", masked_acct_number)
| fields - masked_pan, masked_acct_number

// Drop sensitive fields that cannot be stored
| eval _raw=json_delete(_raw, "cvv", "track1_data", "track2_data", "pin",
"pin_block")

// Set index for PCI events
| eval index = "pci_events"
| into $destination;
```

## Preview Your Pipeline

16. Test your masking rule by clicking on the blue Preview pipeline button ( ) in the top right corner of the screen. If your pipeline filter is correct, you should see some sample events containing “REDACTED_NAME”, mask values inside of pan and account_number respectively. We will also notice the lack of the fields that were dropped.


**Sample Data**

The results shown in the pipeline preview pane are from sample data that is uploaded when you configure your source types within the Edge Processor Service UI. They are not live events from your data stream.

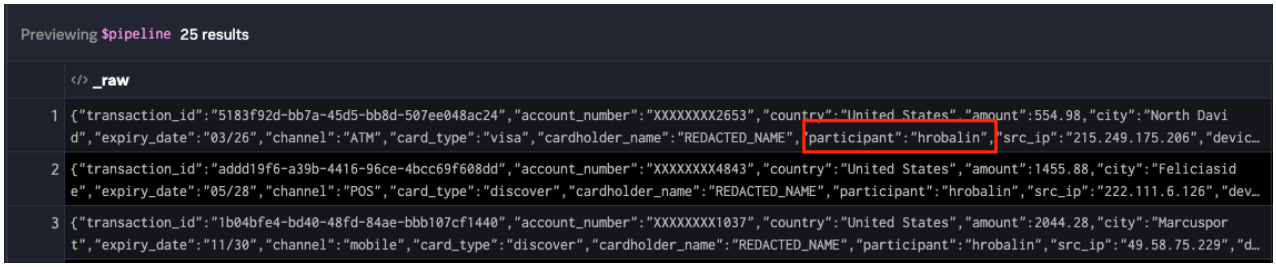
## Identifying our Data

17. Finally, to help you identify data coming from your own pipeline when looking in your shared Splunk Cloud instance, add a custom indexed field to your events by adding an eval command to your pipeline to create a field called ‘participant’ with the value set to your own name:

```
$pipeline =
| from $source
...
| eval _raw = json_set(_raw, "participant", "hrobalin") ← Update with your own
name
| eval index = "pci_events"
| into $destination;
```

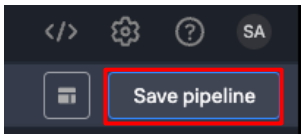
To verify that the new field will be created, click on the Preview button ( ) again. Since we're defining a new field=value pair as part of our pipeline, the new field will display in the Row preview table.

In our **_raw** field we can see our new value:




## Save Your Pipeline

18. Save your pipeline by clicking on **Save pipeline** in the top right corner of the screen. Give your pipeline a suitable name, such as pci_events_send_to_index_<your_name> Be sure to include your name somewhere in the pipeline name to avoid confusion among those who are sharing the same environment as you!

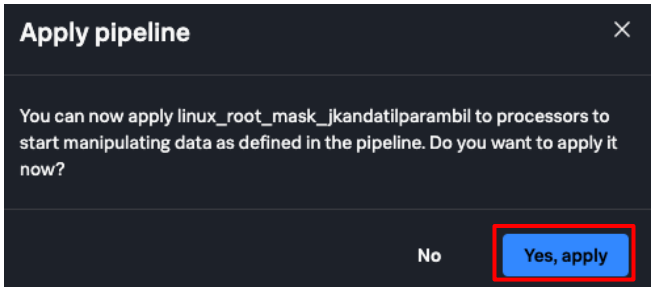




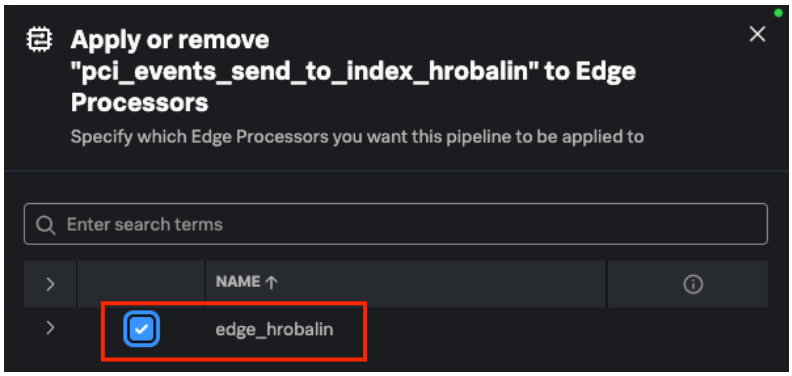
19. Click **Save** to save your pipeline.

## Apply the Pipeline to Your Edge Processor

19. Now an “Apply Pipeline” dialog box will appear, click on “**Yes, apply**”.




20. A pane called “**Apply or remove <pipeline-name> to Edge Processors**” will appear, Select the edge processor you created earlier (see Add a New Edge Processor) and click on **Save** at the bottom..



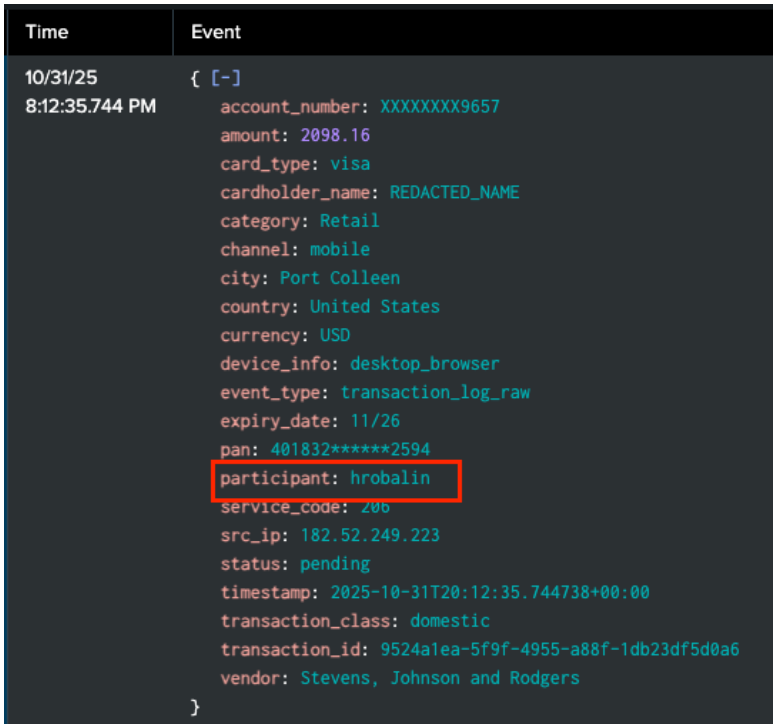

## Confirm PCI Pipeline is applied

21. Make sure the PCI Events Pipeline is applied. Navigate to your Pipelines, you should see your pipeline name, and a status if Applied or Unapplied. See image below:

## Check Your Data in Splunk Cloud

22. Log in to Splunk Cloud and open up the **Search & Reporting** app. Run the following search over the **last 4 hours** and verify that you now see the redacted events:

```
index="pci_events" participant="hrobalin"
```
**← Remember to update the name!**


Time
Event
10/31/25
8:12:35.744 PM
{ [-]
account_number: XXXXXXXX9657
amount: 2098.16
card_type: visa
cardholder_name: REDACTED_NAME
category: Retail
channel: mobile
city: Port Colleen
country: United States
currency: USD
device_info: desktop_browser
event_type: transaction_log_raw
expiry_date: 11/26
pan: 401832******2594
participant: hrobalin
service_code: 206
src_ip: 182.52.249.223
status: pending
timestamp: 2025-10-31T20:12:35.744738+00:00
transaction_class: domestic
transaction_id: 9524a1ea-5f9f-4955-a88f-1db23df5d0a6
vendor: Stevens, Johnson and Rodgers
}

# Exercise 6 – Buttercup Enterprises: EMEA Regulations (DORA)

## Description

Operating in the EMEA region, Buttercup Enterprises must also comply with the Digital Operational Resilience Act (DORA). This regulation is designed to ensure the operational resilience of financial institutions by strengthening requirements for managing and monitoring ICT (Information and Communications Technology) risks.

To support DORA compliance, you will configure the Edge Processor to monitor and manage data related to operational resilience. This involves creating a pipeline that can filter, archive, and route relevant ICT risk and incident data, enabling secure storage and timely analysis in accordance with DORA requirements.

### DORA Pipeline Guide: EMEA Regulations (Digital Operational Resilience Act)

Objective: The Digital Operational Resilience Act (DORA) pipeline aims to ensure compliance by monitoring and managing ICT risks. Its key objectives are to:

1. **Archive Raw Data Conditionally:** Send an unmodified copy of the raw data to Amazon S3 for archival.

2. **Optimize Event Size:** Drop the _raw field after archival since its contents are represented by other extracted fields.

3. **Pseudonymize/Anonymize Data:** Mask IP addresses and pseudonymize User IDs using a salted hash for enhanced privacy while allowing correlation if needed.

4. **Route Based on Severity:** Direct events with "ERROR" or "CRITICAL" severity to a specific index (dora_errors_events) and "WARNING" severity events to another (dora_warning_events).

5. **Minimize Data:** Remove unnecessary fields (incident_description, description, transaction_id, session_id) to reduce data volume for the default stream.

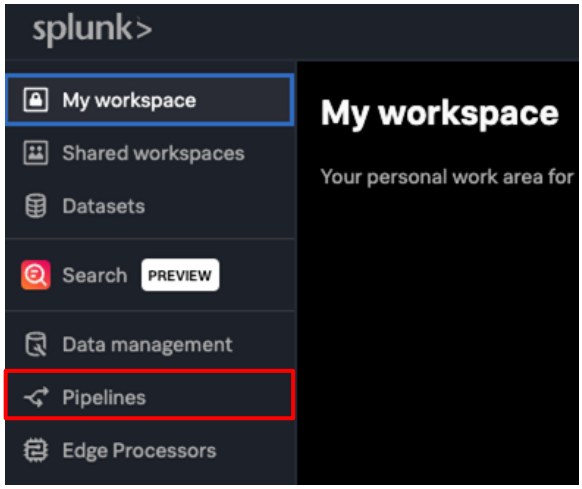
6. **Route to Default Index:** Send remaining events to the dora_events index.

7. **Identify Your Data:** Add a custom field to easily identify your events in a shared Splunk Cloud environment.

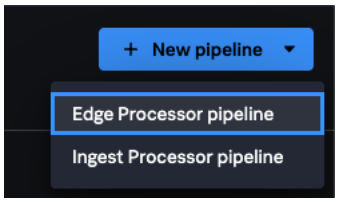
## Steps

### Create a New Pipeline

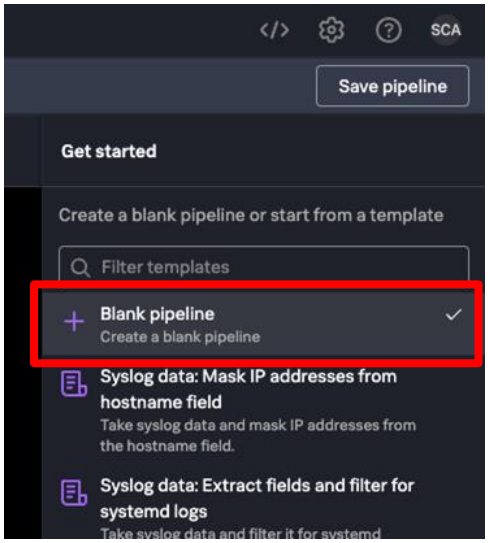
1. While logged in to the Edge Processor service (see Access Edge Processor Service) click on **Pipelines** on the left of the page.



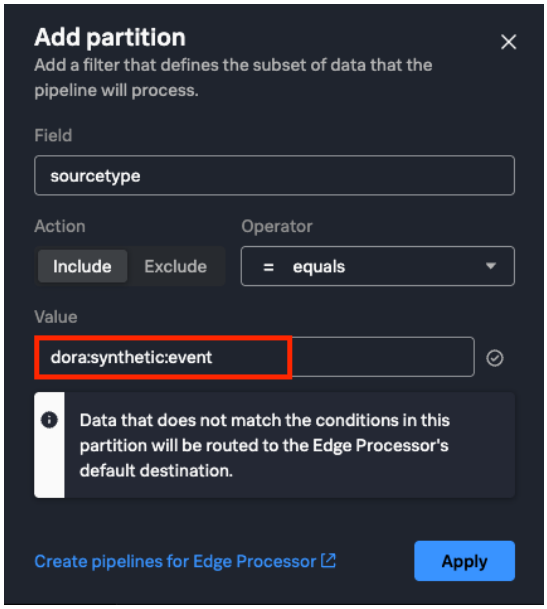

2. In the top right corner of the **Pipelines** page click on **+ New pipeline**. Select **Edge Processor pipeline**




3. On the right-hand side of the **New pipeline** page under the **Get started** pane select on **Blank pipeline** and click on **Next**.

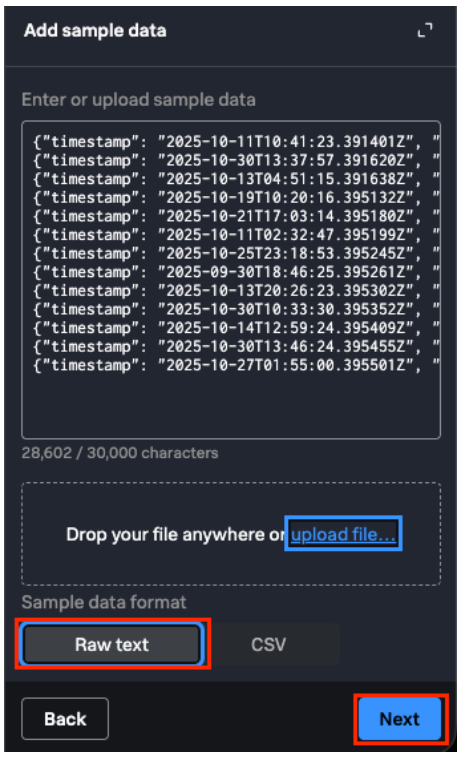


4. On the **Define your pipeline's partition** pane click on the ( + ) button next to the Partition field and select the value **sourcetype** for **Field**, select "Keep" for Action, "**= equals**" for Operator and dora:synthetic:event for **value** on the dropdown list click on and click **Apply**.



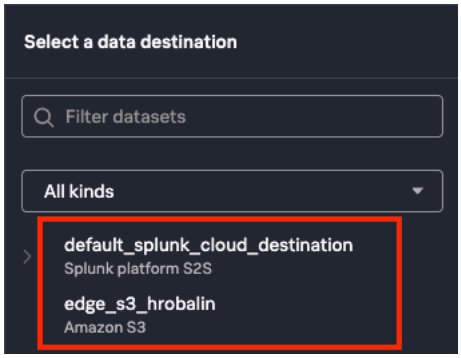

Click on **Next** at the bottom.

5. An **Add sample data** menu will appear on the right side. Upload the **dora_sample_events.jsonl** file that you extracted from the zip file earlier using the "**upload file…**" option at the bottom.



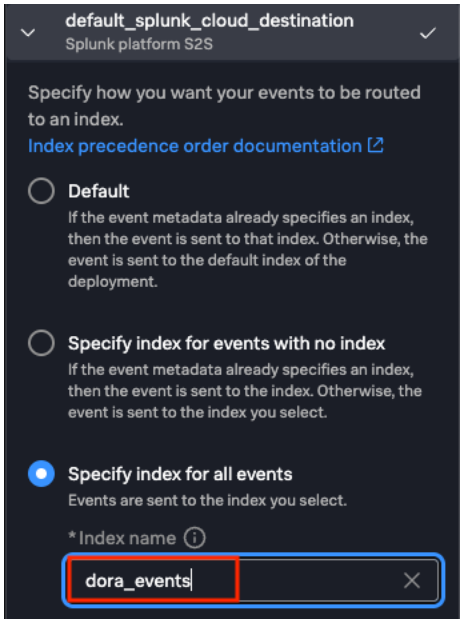
For the **Sample data format** select **Raw text** and click on **Next**.

6. A "**Select a data destination"** menu will appear. Select the **default_splunk_cloud_destination** as your data destination.

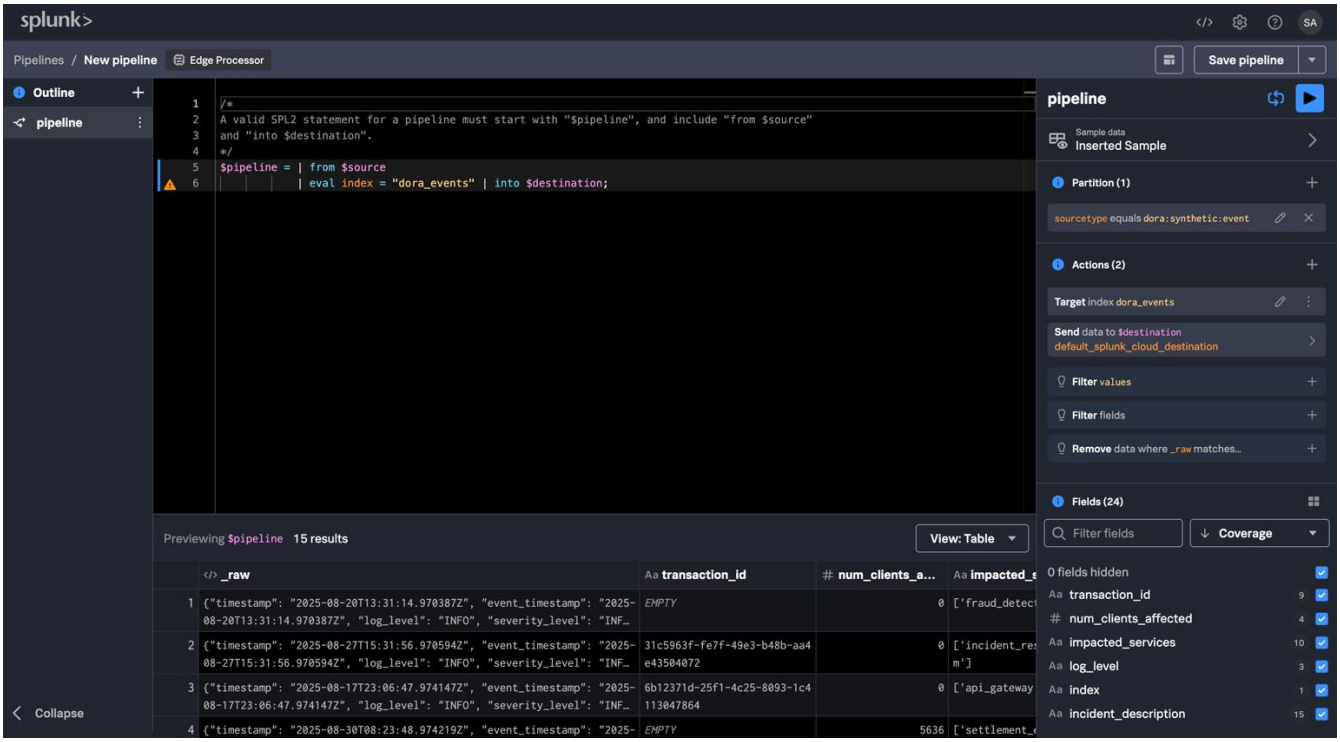



## Route Your Data to the dora_events Index

7. A pane below will open, there you will select **Specify index for all events** and set the index to the **dora_events** from the drop down, as requested by the Buttercup security team, and click on **Done** at the bottom.



8. A screen like below will appear:




As you integrate additional actions into your pipeline, please insert them between the existing pipeline stages and the final index. It is crucial to maintain the order specified in this guide. Refer to the example below:

```
$pipeline = | from $source
. . . ADD NEW ACTIONS HERE
| eval index = "dora_events"
| into $destination;
```

## Define Conditional Raw Data Archival

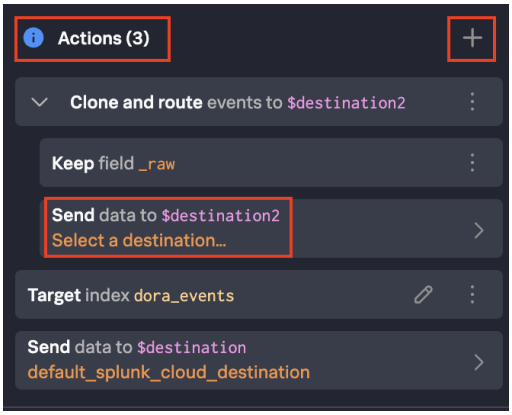
9. Certain DORA requirements require that we archive the data before any processing. This is when we can leverage Edge Processor Pipelines to route the data to an archival location such as Amazon S3, and one to be processed before landing on Splunk.

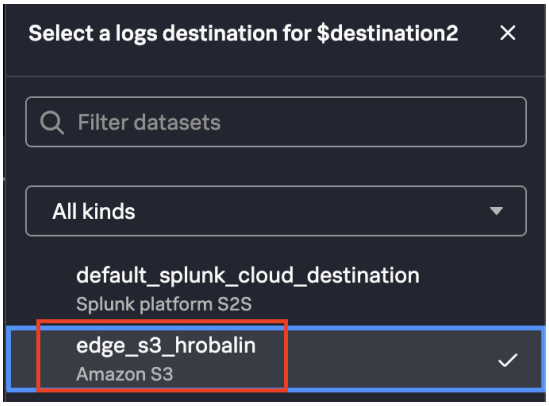
   The **_raw** field in our data provides a comprehensive, unadulterated representation of each event, incorporating all other extracted fields. This makes it a perfect candidate for efficient routing to our long-term archive.

   This routing can be configured either from the Actions menu or directly within an editor. For this demonstration, we will utilize the pipeline editor. Meaning we will write the SPL to define that we will have another route for our data. This is done with the following SPL below:

```
// Initial Archive Routing
| thru
[
  | fields _raw
  | into $destination2
]
```

Now that we have defined in SPL that we want to route data, we must use the Actions Menu on the right to configure the route to the proper destination: Amazon S3




Now the archival routing now configured, we can optimize event size by dropping the _raw field, as its contents are already represented by other extracted fields. This can be achieved using the following SPL:

```
// Drop _raw field to reduce data size after initial archive routing
| eval _raw=json_delete(_raw, "_raw")
```

Make sure to add the above SPL as well.

## Define Pseudonymization/Anonymization

10. DORA regulations necessitate the anonymization of potentially identifiable data. Our review identified the following fields as candidates for anonymization:

    - IP Address
    - User ID

    For IP Addresses, we can either mask an octet or hash the complete address. The selection between these methods will depend on specific data usage, display, and storage requirements. Both options will be presented to demonstrate the flexibility and power available for implementing diverse anonymization strategies.

11. Masking IP Address SPL:

```
// Anonymize IP address (e.g., mask last octet or hash)
| eval masked_ip_address = replace(_raw.ip_address, /\.\d+$/, ".XXX")
```

```
| eval _raw=json_set(_raw, "ip_address", masked_ip_address)
```

12. Next, we will address the User ID field. We will anonymize it by hashing, incorporating a unique and consistent salt to enable correlation across related events when necessary.

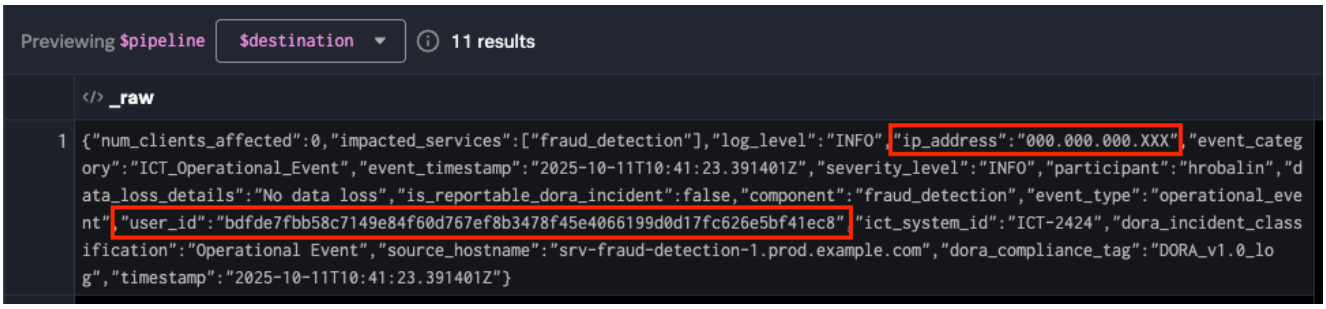
13. User ID Pseudonymization via Hashing

```
 // Pseudonymize user ID (e.g., using a salted hash if correlation is ever
needed under strict controls)
| eval hashed_user_id = if(isnotnull(_raw.user_id), sha256(_raw.user_id +
"dora_secret_salt"), null)
| eval _raw=json_set(_raw, "user_id", hashed_user_id)
```

14. Lastly, we need to drop the temporary fields created.

```
//Remove Temp Fields
| fields - masked_ip_address, hashed_user_id
```

15. Check that your fields are masked and hashed correctly by previewing the pipeline ( ). You should see the columns displayed in the preview pane:




## Define Index Destination based on Severity Level of Event

16. We know have archived a raw unmodified copy of our data, masked, and hashed sensitive data. Our team now is asking us to make sure to send specific events messages to specific index. The team would like us to filter based on Severity Levels: **Error**, **Critical,** and **Warning.**

    Index: dora_error_events -> ERROR or CRITICAL
    Index: dora_warning_events -> WARNING

17. Next, we will add the **Routing** as specified above. (Note: You can add a **Route** via the UI, in our use case it is better to go via editing the pipeline.)

    First we will start by adding our Route Import function to the top of our Pipeline, meaning this will be our new line 1:

```
import route from /splunk.ingest.commands
```

18. Now we can begin to configure our **Route** so that our pipeline match both **ERROR** and **CRITICAL** in the severity_level field.

    We will also need to make sure that we route these events to a specific index,

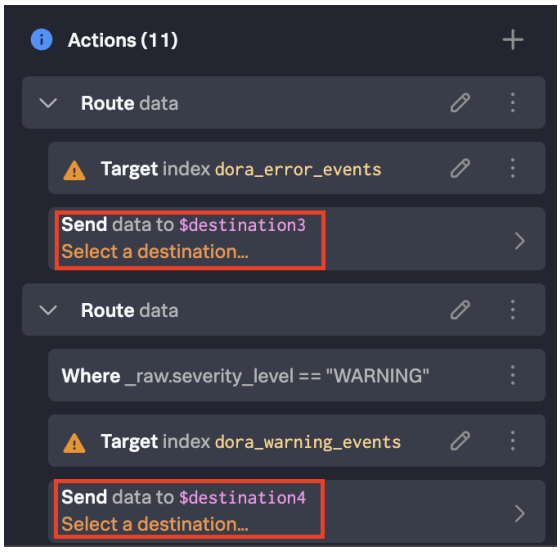
See the following for an example that creates a new route in SPL. We will use this.

```
// Filter for Error and Critical Severity Levels
  | route _raw.severity_level == "ERROR" OR _raw.severity_level =
"CRITICAL",
  [
     | eval index= "
     | into $destination3
  ]
```

19. To configure Warning Severity Level, we can use the below SPL:

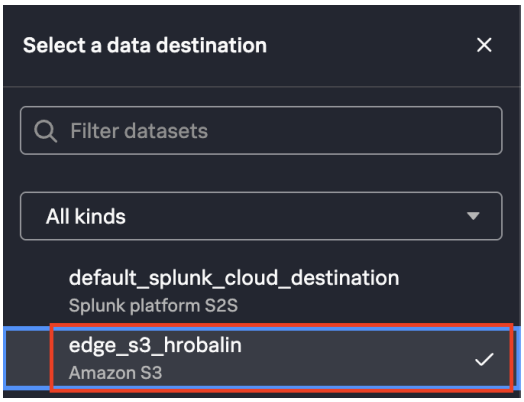
```
// Filter for Warning Severity Levels
  | route _raw.severity_level == "WARNING”,
  [
     | eval index="dora_warning_events"
     | into $destination4
  ]
```

20. Before we can preview our changes, we need to define in the pipeline setting which Amazon S3 Bucket will be $destination3 and $destination4.



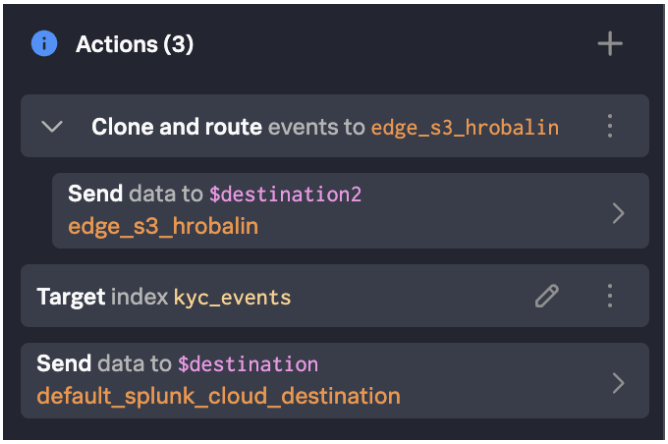

21. Click **Select a destination**

22. Select the Amazon S3 Endpoint for our use case: **edge_s3_username**




23. Click **Apply** at the bottom right.

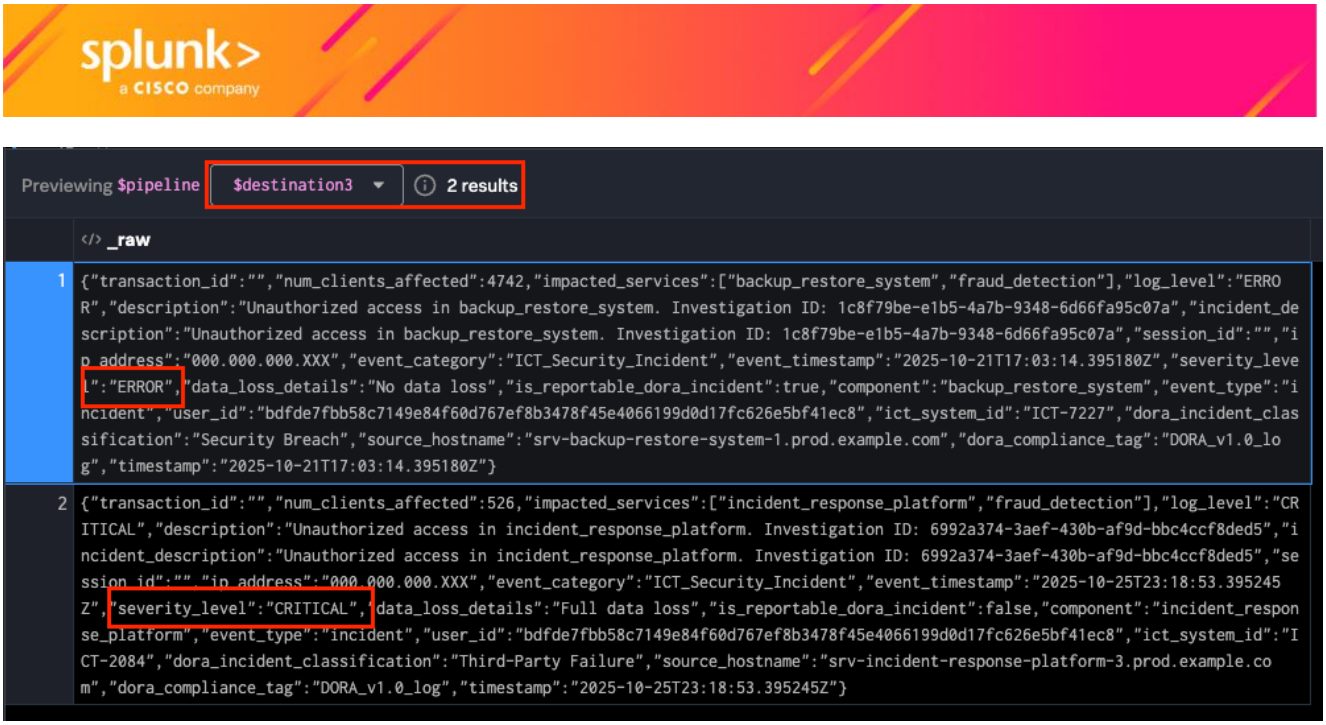
24. **Actions** should now look like this, where $destination3 is an S3 Destination:



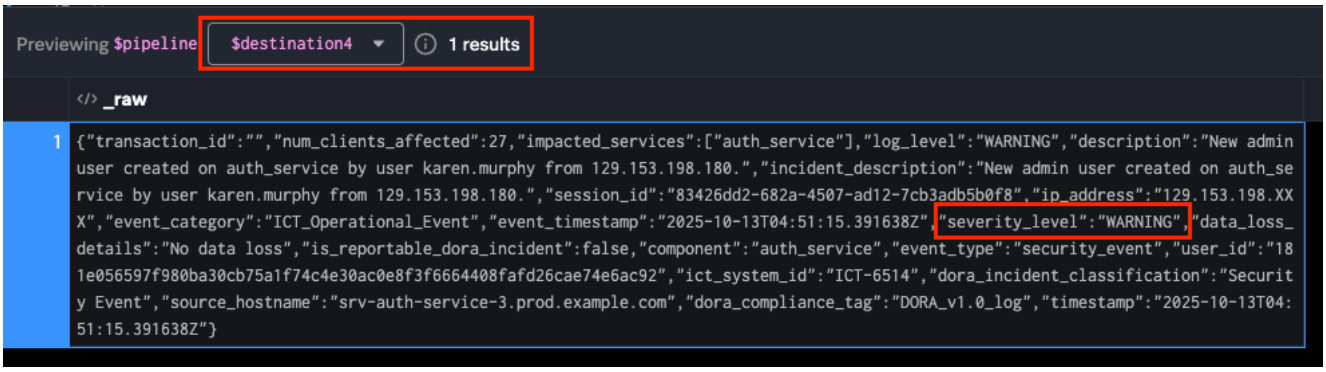

25. Repeat for $destination4.

26. Let's verify that each destination receives only its intended events. We can do this by previewing the pipeline ( ). Let's filter by destination drop down on the preview panel top left:

    If we select $destination3 which is filtering for ERROR and CRITICAL, we can see our events are indeed Error or Critical.




If we select $destination4 which is filtering for WARNING, we can see our events are indeed Warning.




Now that we have verified our pipeline is working as expected, let's proceed to the next step.

## Define Data Minimization by Field Removal

27. Now let's minimize the data, we can do this by removing fields not required.

    In our case we will remove the following:
    - incident_description, description, transaction_id, session_id

    We can do this by adding the following SPL:

```
// Remove Unnecessary Fields for Info Events
| eval _raw = json_delete(_raw, “incident_description”, “description”,
“transaction_id”, “session_id”)
```

28. Checking the Pipeline preview for $destination, you will notice that indeed we have dropped these fields.

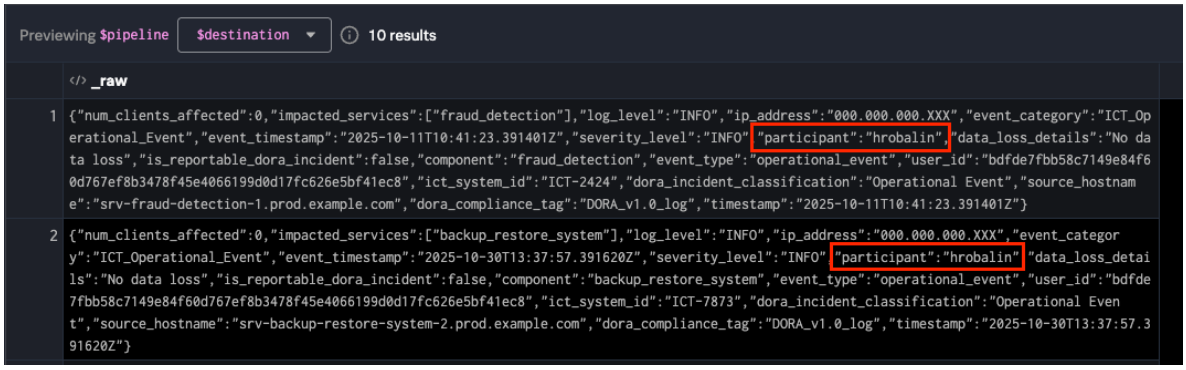
## Add a Custom Indexed Field to Your Data

29. Finally, add a '**participant**' field to your events to help you identify them in Splunk Cloud:

```
$pipeline =
| from $source
...
| eval _raw = json_set(_raw, "participant", "hrobalin") ← Update with your
   own name
| index = "dora_events"
| into $destination;
```

## Preview Your Pipeline

30. Test your update by clicking on the blue Preview pipeline button ( ) in the top right corner of the screen. You should see a preview of your events in the center of the screen showing the IP addresses redacted. You should also see your new '**participant**' field.




## DORA Pipeline

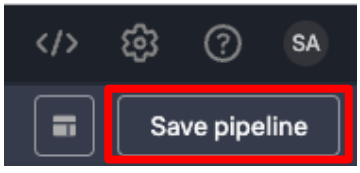
Your DORA Pipeline SPL should look like the below:

```
import route from /splunk.ingest.commands

$pipeline = | from $source

// Initial Archive Routing
| thru
[
    | eval _raw = json_extract(_raw, "_raw")
    | into $destination2
]

// Drop _raw field to reduce data size after initial archive routing
| eval _raw=json_delete(_raw, "_raw")
```

```
// Anonymize IP address (e.g., mask last octet)
| eval masked_ip_address = replace(_raw.ip_address, /\.\d+$/, ".XXX")
| eval _raw=json_set(_raw, "ip_address", masked_ip_address)

// Pseudonymize user ID (e.g., using a salted hash if correlation is ever
needed under strict controls)
| eval hashed_user_id = if(isnotnull(_raw.user_id), sha256(_raw.user_id +
"dora_secret_salt"), null)
| eval _raw=json_set(_raw, "user_id", hashed_user_id)

| fields - masked_ip_address, hashed_user_id

// Filter for Error and Critical Severity Levels
    | route _raw.severity_level == "ERROR" OR _raw.severity_level =
"CRITICAL",
    [
        | eval index="dora_errors_events"
        | into $destination3
    ]

// Filter for Warning Severity Levels
    | route _raw.severity_level == "WARNING",
    [
        | eval index="dora_warning_events"
        | into $destination4
    ]

// Remove Unnecessary Fields for Info Events
| eval _raw=json_delete(_raw, "incident_description", "description",
"transaction_id", "session_id")

| eval _raw = json_set(_raw, "participant", "hrobalin")

// Finalize and Route Info Events
| eval index = "dora_events"
| into $destination;
```

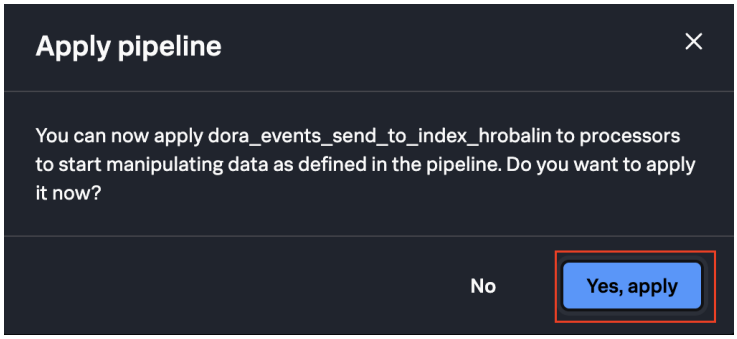
## Save Your Pipeline

31. Save your pipeline by clicking on **Save pipeline** in the top right corner of the screen. Give your pipeline a suitable name, such as **dora_events_send_to_index_<yourName>**. Remember to add your name or initials at the end to help identify it.

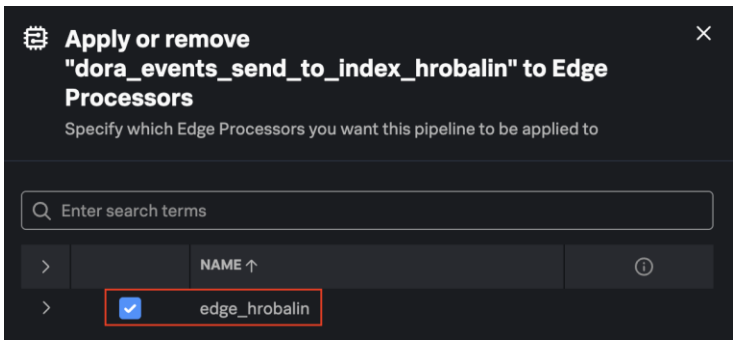



## Apply the Pipeline to Your Edge Processor

32. Now an “Apply Pipeline” dialog box will appear, click on “**Yes, apply**”.




33. A pane called “**Apply or remove <pipeline-name> to Edge Processors**” will appear, Select the edge processor you created earlier (see Add a New Edge Processor) and click on **Save** at the bottom.




## Confirm DORA Pipeline is applied

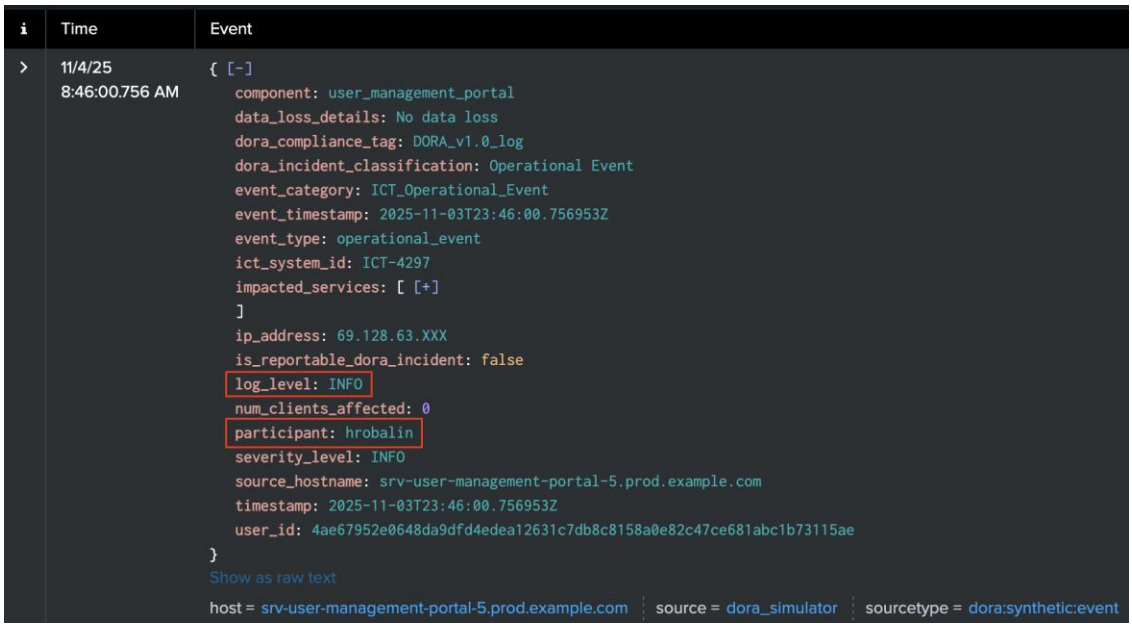
34. Make sure the DORA Events Pipeline is applied. Navigate to your Pipelines, you should see your pipeline name, and a status if Applied or Unapplied.

## Check Your Data in Splunk Cloud

35. Log in to Splunk Cloud and open up the **Search & Reporting** app. Run the following search over the **last 15 minutes** and verify that you now see the redacted events.

```
index=dora_events participant="hrobalin"
```

**← Update the name!**




Note that since our search results will only contain those events that match our IP address range you may need to wait up to a couple of minutes for events matching that range to arrive at Splunk. If you don't get any results immediately try re-running the search after a minute or two.

# Exercise 7 – Buttercup Enterprises: APJC Regulations (RMiT)

## Description

In the APJC region, Buttercup Enterprises must comply with the Risk Management in Technology (RMiT) guidelines, established to enhance the technology risk management practices of financial institutions.

To address RMiT requirements, you will set up the Edge Processor to capture, filter, and securely route technology risk-related data. This includes creating a dedicated pipeline for monitoring, archiving, and analyzing data related to technology incidents and controls, ensuring compliance with RMiT standards.

### RMiT Pipeline Guide: APJC Regulations (Risk Management in Technology)

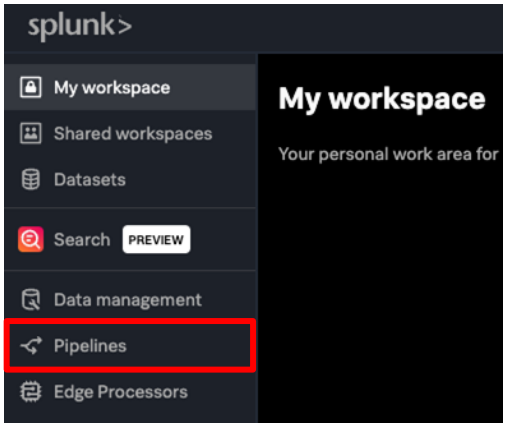
Objective: The Risk Management in Technology (RMiT) pipeline ensures compliance by capturing, filtering, and securely routing technology risk-related data. Its key objectives are to:

1. **Enrich Data with Lookups:** Enhance events with asset inventory information (criticality, location, service impact) using a lookup file.
2. **Define Risk Levels:** Assign a risk_level (High, Medium, Low) based on the criticality from the lookup.
3. **Mask Sensitive Operational Details:** Mask IP addresses, pseudonymize user_id, mask reporting_user_or_process, and redact event_description if it contains PII.
4. **Filter Non-Actionable Logs:** Retain only relevant events based on status and criticality.
5. **Route Based on Event Type:** Direct events to different Splunk indexes (rmit_cyber_events, rmit_change_events, rmit_tprm_events) based on their event_type.
6. **Minimize Data:** Remove low-value or empty fields from the default stream.
7. **Identify Your Data:** Add a custom field to easily identify your events in a shared Splunk Cloud environment.

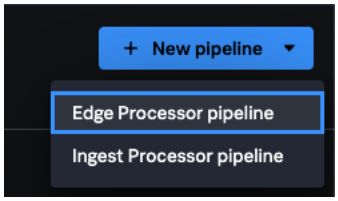
## Steps

### Create a New Pipeline

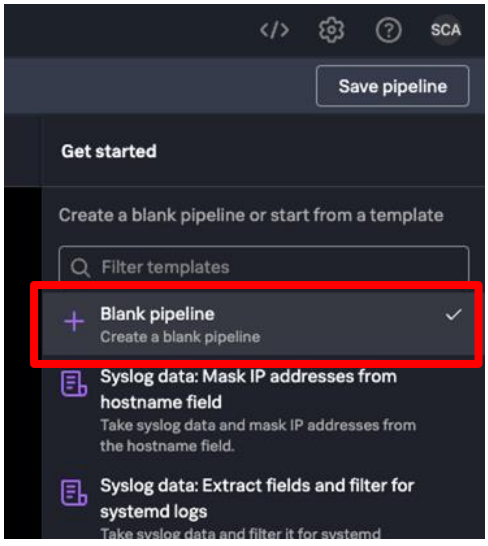
1. While logged in to the Edge Processor service (see Access Edge Processor Service) click on **Pipelines** on the left of the page.



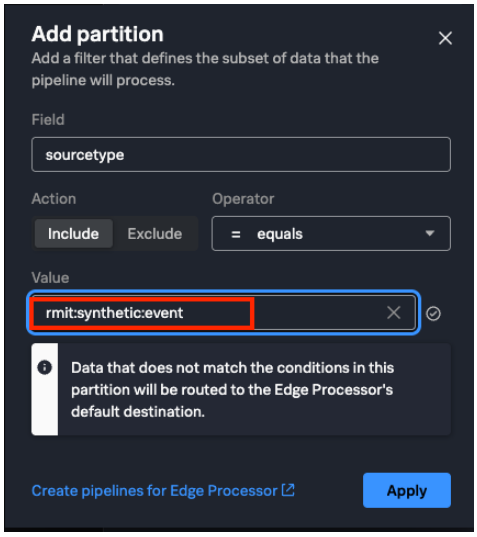

2. In the top right corner of the **Pipelines** page click on **+ New pipeline**. Select **Edge Processor pipeline**




3. On the right hand side of the **New pipeline** page under the **Get started** pane select on **Blank pipeline** and click on **Next**.



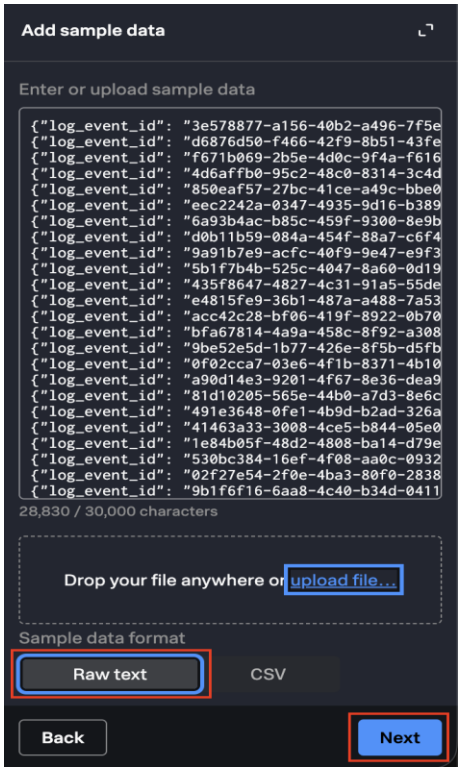
4. On the **Define your pipeline's partition** pane click on the ( + ) button next to the Partition field and select the value **sourcetype** for **Field**, select "Keep" for Action, "**= equals**" for Operator and **rmit:synthetic:event** for **value** on the dropdown list click on and click **Apply**.




**Note:** Please make sure to check that there is no extra space at the end.

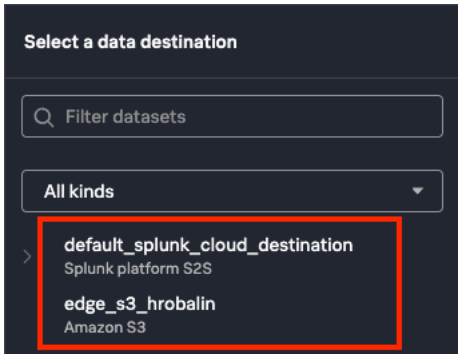
Click on **Next** at the bottom.

5. An **Add sample data** menu will appear on the right side. Upload the **rmit_sample_events.jsonl** file that you extracted from the zip file earlier using the "**upload file…**" option at the bottom.



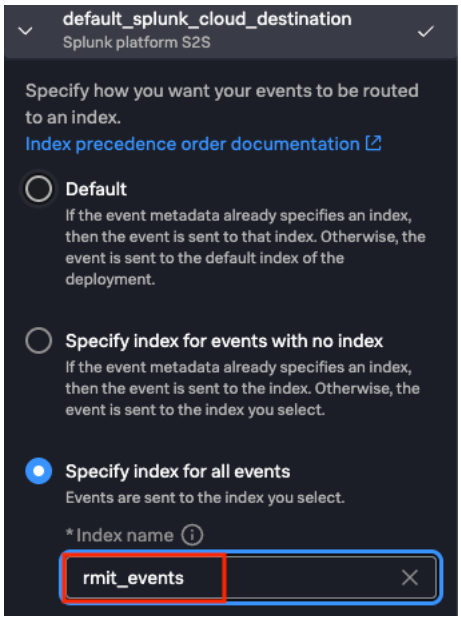
For the **Sample data format** select **Raw text** and click on **Next**.

6. A "**Select a data destination"** menu will appear. Select the **default_splunk_cloud_destination** as your data destination and click on **Next**.

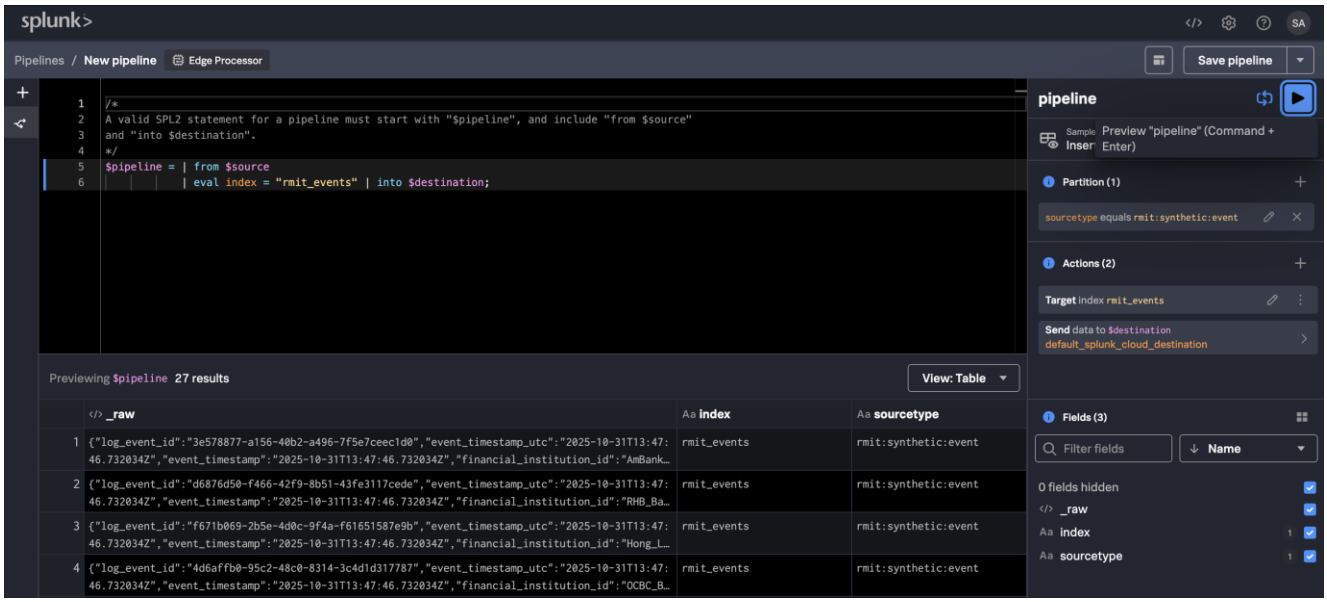



## Route Your Data to the rmit_events Index

7. A pane below will open, there you will select **Specify index for all events** and set the index to the **rmit_events** from the drop down, as requested by the Buttercup security team, and click on **Done** at the bottom.




8. A screen like below will appear:



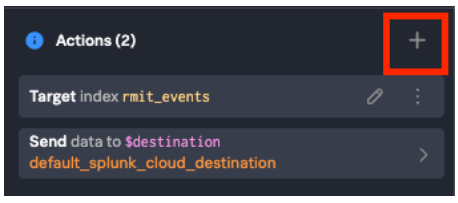

As you integrate additional actions into your pipeline, please insert them between the existing pipeline stages and the final index. It is crucial to maintain the order specified in this guide. Refer to the example below:

```
$pipeline = | from $source
. . . ADD NEW ACTIONS HERE
| eval index = "rmit_events"
| into $destination;
```

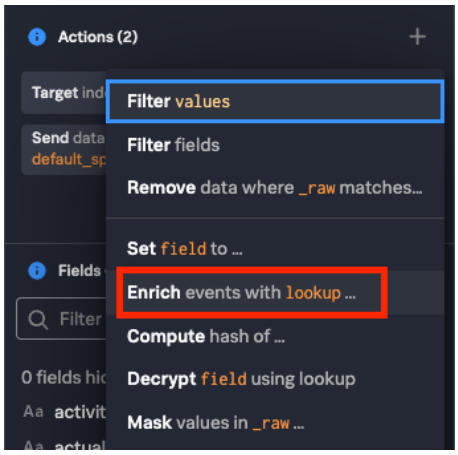
## Configure Lookup for Events

9. We want to first start by enriching this data, it will provide us more context to an event coming in as well, as allow us to do advance routing based on enriched data.

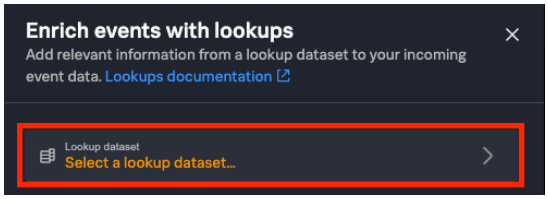
   To accomplish this, we will import a lookup using Edge Processing, through the **Actions** menu on the right side. Press the Plus button on the right side.



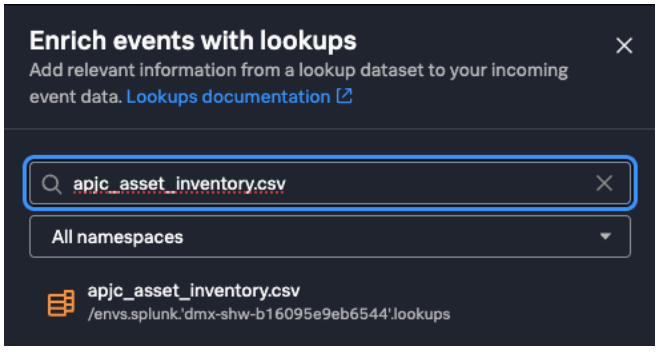
10. We will select: **Enrich** events with lookup



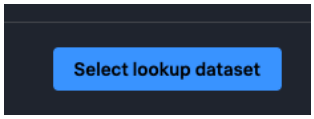

11. Now, we will click on **Select a Lookup dataset**



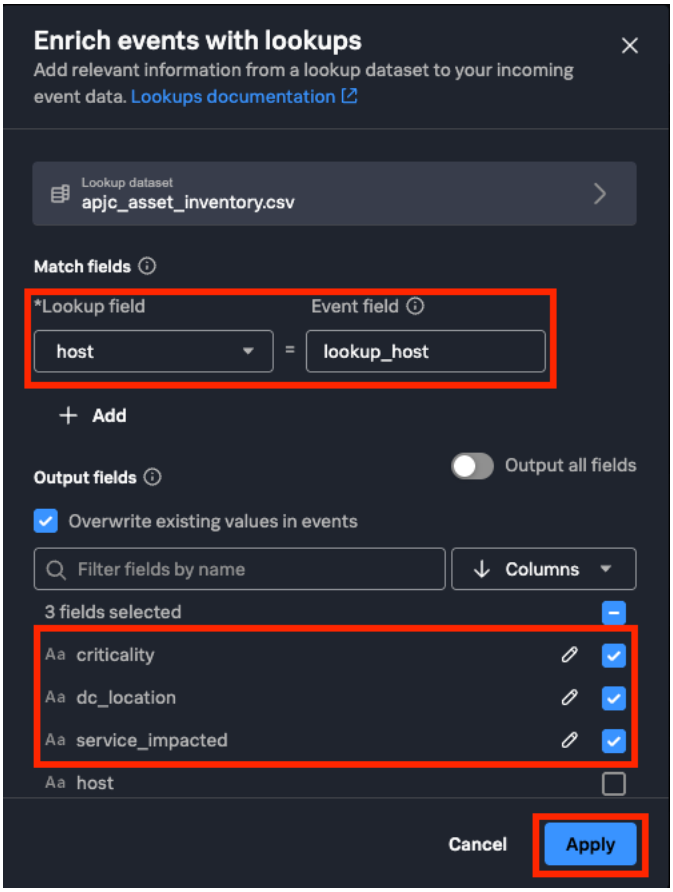

12. In our example we will search for **apjc_asset_inventory.csv**, and **select** the option that comes up:




13. Make sure to press the button on the bottom right, labeled: **Select lookup dataset**




14. Now we need to configure the Lookup fields to match, and the fields to output from the Lookup.




15. For **Match fields** configure the following:

    - Lookup field as **host**
    - Event field as **lookup_host**

16. For **Output fields** select the following:

    - **criticality**
    - **dc_location**
    - **service_impacted**

17. Confirm your configuration matches the image on the right.

18. Select **Apply.**

19. To ensure the lookup file has been correctly added and is ready for use within our pipeline, let's perform a quick check.

20. You can confirm its successful addition by looking for the following line at the top of your pipeline:

```
import 'apjc_asset_inventory.csv' from /envs.splunk.'dmx-shw-2c53feabe860f8'.lookups
```

21. You'll observe that the IDE has placed the lookup SPL line after | eval index="rmit_events". To ensure that our events are enriched before any potential routing decisions are made, the lookup must occur earlier in the pipeline.

    Therefore, we will move this lookup line to immediately follow | from $source.

    The revised pipeline should appear as follows:

```
import 'apjc_asset_inventory.csv' from /envs.splunk.dmx-show-
2c3742ae24f8.lookups

$pipeline =
| from $source
| lookup 'apjc_asset_inventory.csv' host AS lookup_host OUTPUT dc_location,
criticality, service_impacted
| eval index = "rmit_events"
| into $destination;
```

    Before we can enrich our event data, we must ensure we have a reliable key to connect it to our external asset inventory.

    Below is our lookup file; pay close attention to the header names:

```
host,criticality,service_impacted,dc_location
Maybank_MY,Tier1,Core Banking System,KUL-DC2
Maybank_MY,Tier2,RENTAS Gateway,HKG-DC3
```

    We've identified that the headers in our lookup table and the field names within our log events are inconsistent. This discrepancy necessitates the creation of a matching key to enable the integration of lookup data with our incoming events.

## Lookup Enrichment Pattern for Nested JSON Events

In this section, we demonstrate how to enrich events that arrive wrapped in a _raw JSON field by performing a lookup and injecting the results back into the event.

### Extract Fields from _raw

22. Edge Processor lookups operate on top-level fields, not nested JSON. Since our events arrive wrapped in _raw, we first extract the fields we need into temporary top-level fields using

json_extract(). We also extract the first element from the impacted_systems array using mvindex() for later processing.

### Build the Lookup Key and Perform the Lookup

23. Extract the bank name (financial_institution_id from _raw) and use it as the lookup key to match against the host column in our CSV lookup table. The lookup returns three enrichment fields: data center location, criticality tier, and service name.

### Inject Lookup Results Back into _raw

24. Using json_set(), we add each lookup result back into the _raw JSON object. Each line includes a null check to ensure we only inject valid values. This ensures the enrichment travels with the event through all subsequent transformations and routing stages.

### Why This Pattern Matters

Then we are injecting enrichment back into _raw, we preserve the complete enriched event in the archived data. When the event is routed to different destinations based on type, the enrichment (criticality tier, data center location, service name) is already part of the event payload—no need to re-look it up at query time.

25. The above can be implemented in SPL in the following way:

```
    // Extract fields from _raw for lookup key building
    | eval affected_service = json_extract(_raw, "affected_service")
    | eval change_description = json_extract(_raw, "change_description")
    | eval impacted_systems_array = json_extract(_raw, "impacted_systems")
    | eval first_impacted = if(isnotnull(impacted_systems_array),
mvindex(impacted_systems_array, 0), null)

    // Create lookup key: match on bank name only (lookup CSV has bank as the
host key)
    | eval lookup_host = json_extract(_raw, "financial_institution_id")
    | lookup 'apjc_asset_inventory.csv' host AS lookup_host OUTPUT
dc_location, criticality, service_impacted

    // Inject lookup results into _raw so enrichment travels with the event
    | eval _raw = if(isnotnull(dc_location), json_set(_raw, "dc_location",
dc_location), _raw)
    | eval _raw = if(isnotnull(criticality), json_set(_raw, "criticality",
criticality), _raw)
    | eval _raw = if(isnotnull(service_impacted), json_set(_raw,
"service_impacted", service_impacted), _raw)
```

26. Remove any temporary fields created for the lookup:

```
    // Remove temporary top-level lookup fields to avoid duplication (after
injecting into _raw)
```

```
    | fields - affected_service, change_description, impacted_systems_array,
first_impacted, lookup_host, dc_location, criticality, service_impacted
```

## Enrich with Asset and Operational Context

27. Having successfully created a composite key and mapped our lookup to the events, the next step is to enable enrichment. We aim to introduce a new field, risk_level, to assign a severity to specific services.

    We want to define the new field risk_level in this order:

    - Tier 0 -> High
    - Tier 1 -> Medium
    - All Other Tiers -> Low

    SPL Example to do this:

```
    // Extract criticality from _raw for enrichment logic
    | eval criticality_from_raw = json_extract(_raw, "criticality")
    | eval risk_level = case(criticality_from_raw == "Tier0", "High",
criticality_from_raw == "Tier1", "Medium", true, "Low")
    | eval _raw = json_set(_raw, "risk_level", risk_level)
```

28. Check that newly created field risk_level which was enriched by the lookup is showing up by previewing the pipeline ( ). You should see the columns displayed in the preview pane:

| | _raw | index | ip |
|---|---|---|---|
| 1 | 10.2.1.33 - - [29/Apr/2013 18:09:05:132] "GET /category.screen?uid=9299979e-a268-4151-a4cd-2994c6971fce&category=Misc&JSESSIONID=SD3SL1FF7ADFF8 HTTP 1.1" 503 1829 "http://www.buttercupe… | web | 10.2.1.33 |
| 2 | 10.2.1.33 - - [29/Apr/2013 18:09:05:132] "GET /category.screen?uid=f510fde4-e9aa-4e69-a713-34abc83574e5&category=Misc&JSESSIONID=SD3SL1FF7ADFF8 HTTP 1.1" 503 1829 "http://www.buttercupe… | web | 10.2.1.33 |
| 3 | 10.2.1.33 - - [29/Apr/2013 18:09:03:144] "GET /product.screen?uid=feb32a24-87be-48b6-b8bd-30efb96a6591&product_id=MCF-3&JSESSIONID=SD1SL7FF2ADFF3 HTTP 1.1" 503 3985 "http://www.buttercu… | web | 10.2.1.33 |
| 4 | 10.2.1.33 - - [29/Apr/2013 18:09:05:132] "GET /category.screen?uid=fdbcea80-0060-49fa-a0bf-db33fa1bde31&category=Misc&JSESSIONID=SD3SL1FF7ADFF8 HTTP 1.1" 503 1829 "http://www.buttercupe… | web | 10.2.1.33 |

## Mask Sensitive Operational Details

29. We have several requirements for Masking Operational Details:

    - Mask the last octet of all IP Addresses
    - Hash all User IDs using SHA256
    - Redact all Reporting User or Process
    - Redact Event Descriptions if contains PII

30. Masking IP Addresses (Src & Dest)

```
    // Mask IP addresses Source and Destination (last octet)
    | eval masked_src_ip_address = replace(_raw.source_ip, /\.\d+$/, ".XXX")
    | eval _raw = json_set(_raw, "source_ip", masked_src_ip_address)
```

```
    | eval masked_dest_ip_address = replace(_raw.destination_ip, /\.\d+$/,
".XXX")
    | eval _raw = json_set(_raw, "destination_ip", masked_dest_ip_address)
```

31. Hash all User IDs using SHA256:

```
    // Hash user_id with SHA256 and a salt
    | eval hashed_user_id = if(isnotnull(_raw.user_id), sha256(_raw.user_id +
"rmit_secret_salt"), null)
    | eval _raw = json_set(_raw, "user_id", hashed_user_id)
```

32. Redact all Reporting User or Process:

```
   // Redact reporting_user_or_process
    | eval masked_reporting = if(isnotnull(json_extract(_raw,
"reporting_user_or_process")), "REDACTED", null)
    | eval _raw = json_set(_raw, "reporting_user_or_process",
masked_reporting)
```

33. Redact Event Descriptions if contains PII:

```
    // Redact event_description if it contains PII (example: redact if
contains 'user_')
    | eval event_desc = json_extract(_raw, "event_description")
    | eval redacted_desc = if(isnotnull(event_desc) and match(event_desc,
/user_/), "REDACTED_EVENT_DESC", event_desc)
    | eval _raw = json_set(_raw, "event_description", redacted_desc)
```

34. Drop Temporary Fields:

```
    // Drop Temporary fields used for processing
    | fields - masked_src_ip_address, masked_dest_ip_address, hashed_user_id,
masked_reporting, event_desc, redacted_desc, criticality_from_raw, risk_level
```

## Filter Non-Actionable Logs

35. Next, we will drop all events where the Status is 'Closed' or 'Resolved', unless Criticality is “Tier0” or “Tier1”:

```
    // Filter out events with status = "Closed" or "Resolved" unless
criticality is "Tier0" or "Tier1"
    | eval status_from_raw = coalesce(json_extract(_raw, "status_code"),
json_extract(_raw, "status"))
    | eval criticality_check = json_extract(_raw, "criticality")
    | where (status_from_raw != "Closed" and status_from_raw != "Resolved") or
(criticality_check == "Tier0" or criticality_check == "Tier1")
```

## Route Processed Data Based on RMiT Relevance

36. Next, we must first define our routes, and we do this by creating a new field called event_route we will define this field based on what our event_type is:

    - cybersecurity -> CYBER_ROUTE
    - change_request_log -> CHANGE_ROUTE
    - third_party_risk_update -> TPRM_ROUTE
    - All Else -> DEFAULT_ROUTE

```
    // Route based on event_type - extract from _raw and store in _raw for
routing logic
    | eval event_type_from_raw = json_extract(_raw, "event_type")
    | eval event_route_value = case(event_type_from_raw ==
"cybersecurity_alert", "CYBER_ROUTE", event_type_from_raw ==
"change_request_log", "CHANGE_ROUTE", event_type_from_raw ==
"third_party_risk_update", "TPRM_ROUTE", true, "DEFAULT_ROUTE")
    | eval _raw = json_set(_raw, "event_route", event_route_value)
```

    We will also drop the temporary fields we just created:

```
   // Drop Temporary fields used for processing
    | fields - criticality_check, event_type_from_raw,
status_from_raw,event_route_value
```

37. Next, we will go ahead and start adding our routing. We can do this via the Action Menu on the right side. For our case, we will avoid this, the reason is that adding via the Action Menu will add the route to the wrong location and require extra configuration. To avoid confusion, we will manually add a route, and in a real world this is the way you would do it.

38. Since we are going the manual way for Routes, we need to manually import the route function in the pipeline, we do this by adding the following line to the top of our pipeline, above where the pipeline is importing our lookup.

```
import route from /splunk.ingest.commands
```

39. Now let's define what fields we are routing, to what destination, and their respective indexes:

    - Event_Route = CYBER_ROUTE
        - Index = rmit_cyber_events
        - $destination2
    - Event_Route = CHANGE_ROUTE
        - Index = rmit_change_events
        - $destination3
    - Event_Route = TPRM_ROUTE
        - Index = rmit_tprm_events

- $destination4

Be aware that we in our SPL below we will also drop several fields, the reason for this is that several fields are only relevant for specific routes, so we need to drop fields that are empty for these specific routes.

40. This is our SPL for CYBER_ROUTE:

```
    | route _raw.event_route == "CYBER_ROUTE",
    [
        // Remove unwanted keys from `_raw` for cyber events so the archived
payload is compact
        | eval _raw = json_delete(_raw, "change_request_id",
"change_description", "change_status", "risk_impact_assessment",
"planned_start_datetime_utc", "actual_end_datetime_utc", "third_party_id",
"vendor_name", "service_type", "risk_assessment_update_reason",
"overall_risk_rating", "key_risks_identified", "due_diligence_status")
        | eval index = "rmit_cyber_events"
        | into $destination2
    ]
```

41. This is our SPL for CHANGE_ROUTE:

```
    | route _raw.event_route == "CHANGE_ROUTE",
    [
        // Remove unwanted keys from `_raw` for change events so the archived
payload is compact
        | eval _raw = json_delete(_raw, "cyber_event_id", "threat_type",
"severity", "status", "impacted_systems", "detection_source",
"response_actions_taken", "bnm_notification_sent", "bnm_report_ref_id",
"third_party_id", "vendor_name", "service_type",
"risk_assessment_update_reason", "overall_risk_rating",
"key_risks_identified", "due_diligence_status")
        | eval index = "rmit_change_events"
        | into $destination3
    ]
```

42. This is our SPL for TPRM_ROUTE:

```
    | route _raw.event_route == "TPRM_ROUTE",
    [
        // Remove unwanted keys from `_raw` for TPRM events so the archived
payload is compact
        | eval _raw = json_delete(_raw, "cyber_event_id", "threat_type",
"severity", "status", "impacted_systems", "detection_source",
"response_actions_taken", "bnm_notification_sent", "bnm_report_ref_id",
"change_request_id", "change_description", "change_status",
```

```
"risk_impact_assessment", "planned_start_datetime_utc",
"actual_end_datetime_utc")
        | eval index = "rmit_tprm_events"
        | into $destination4
    ]
```

We will go ahead, and define the index for $destination2, as rmit_cyber_events

### Filter Low-Value or Non-Actionable Logs

43. Next, we will optimize our data by trimming and dropping low-value or non-actionable fields. We've observed that many fields remain empty unless associated with specific routes. By removing these empty fields, we can significantly reduce event size and send only relevant, populated fields to our default index.

    This can be achieved using the following SPL:

```
// Drop fields associated with event_routes filtered before as they are empty
    | fields - cyber_event_id, threat_type, severity, status,
impacted_systems, detection_source, response_actions_taken,
bnm_notification_sent, bnm_report_ref_id, change_request_id,
change_description, change_status, risk_impact_assessment,
planned_start_datetime_utc, actual_end_datetime_utc, third_party_id,
vendor_name, service_type, risk_assessment_update_reason, overall_risk_rating,
key_risks_identified, due_diligence_status
```

### Add a Custom Indexed Field to Your Data

44. Finally, add a '**participant**' field to your events to help you identify them in Splunk Cloud:

```
$pipeline =
| from $source
...
| eval _raw = json_set (_raw, " participant ", " hrobalin ") ← Update with
   your own name
| index = "rmit_events"
| into $destination;
```

### Configure Splunk Destination

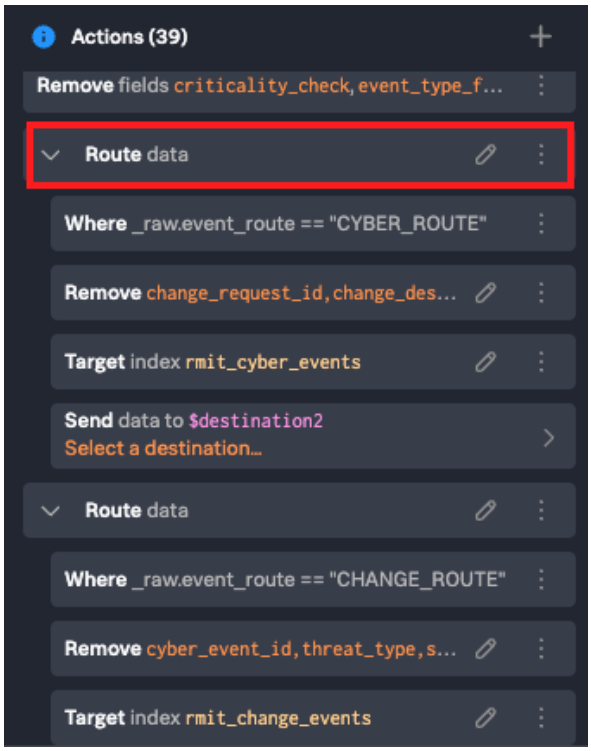
45. Before we can preview the pipeline, we will need to manually configure the destinations we are using for Log Destinations depending on your data routing requirements. We will need to do that for each of the following destinations:

    $destination2 -> default_splunk_cloud_destination

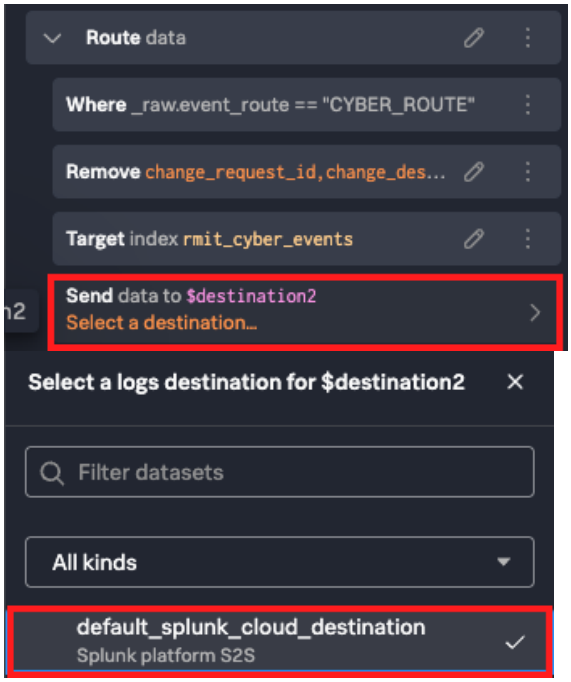
    $destination3 -> default_splunk_cloud_destination

$destination4 -> default_splunk_cloud_destination

46. On the right side, under Actions, scroll down to the Route Data sections:

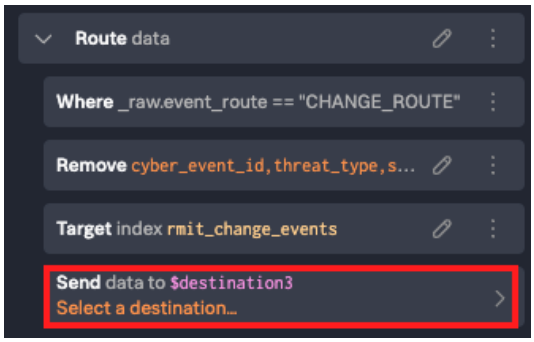


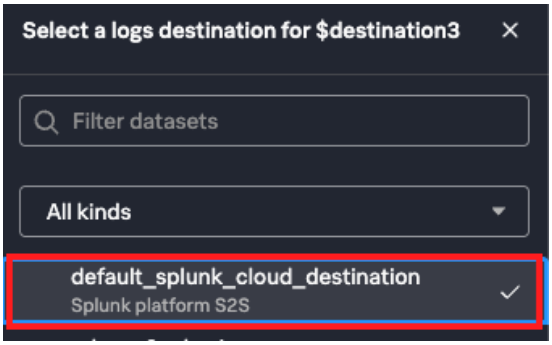

47. Select on Send data to $destination2, and select default_splunk_cloud_destination:




48. Select **Apply** on the bottom right.

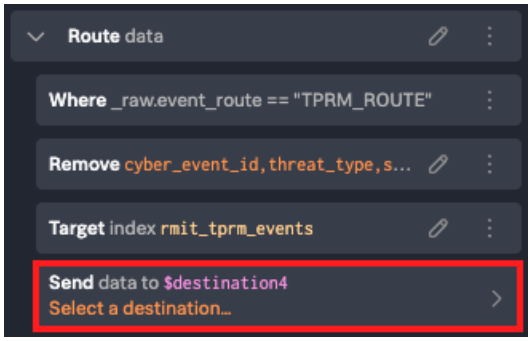
49. Select on Send data to $destination3, and select default_splunk_cloud_destination:

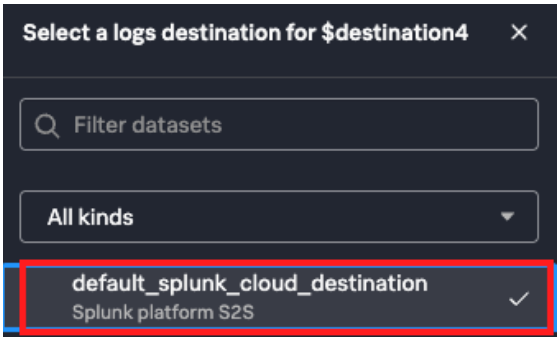



50. Select **Apply** on the bottom right.

51. Select on Send data to $destination4, and select default_splunk_cloud_destination:



52. Select **Apply** on the bottom right.

## Preview Your Pipeline

53. Test your pipeline by clicking on the blue Preview pipeline button ( ) in the top right corner of the screen. You should see a preview of your events in the center of the screen with processed data and an index field added.


## RMiT Pipeline

Your RMiT Pipeline SPL should look like the following (Be aware copying below will require updating env name on second line:

```
import route from /splunk.ingest.commands
import 'apjc_asset_inventory.csv' from /envs.splunk.'dmx-shw-9cf6b5b1083388'.lookups

$pipeline =
| from $source

    // Extract fields from _raw for lookup key building
    | eval affected_service = json_extract(_raw, "affected_service")
    | eval change_description = json_extract(_raw, "change_description")
    | eval impacted_systems_array = json_extract(_raw, "impacted_systems")
    | eval first_impacted = if(isnotnull(impacted_systems_array),
mvindex(impacted_systems_array, 0), null)

    // Create lookup key: match on bank name only (lookup CSV has bank as the host key)
    | eval lookup_host = json_extract(_raw, "financial_institution_id")
    | lookup 'apjc_asset_inventory.csv' host AS lookup_host OUTPUT dc_location,
criticality, service_impacted

    // Inject lookup results into _raw so enrichment travels with the event
    | eval _raw = if(isnotnull(dc_location), json_set(_raw, "dc_location", dc_location),
_raw)
    | eval _raw = if(isnotnull(criticality), json_set(_raw, "criticality", criticality),
_raw)
    | eval _raw = if(isnotnull(service_impacted), json_set(_raw, "service_impacted",
service_impacted), _raw)
```

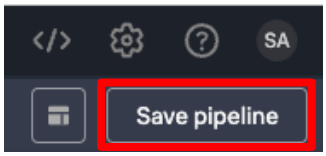
```
    // Remove temporary top-level lookup fields to avoid duplication (after injecting into
_raw)
    | fields - affected_service, change_description, impacted_systems_array,
first_impacted, lookup_host, dc_location, criticality, service_impacted

    // Extract criticality from _raw for enrichment logic
    | eval criticality_from_raw = json_extract(_raw, "criticality")
    | eval risk_level = case(criticality_from_raw == "Tier0", "High", criticality_from_raw
== "Tier1", "Medium", true, "Low")
    | eval _raw = json_set(_raw, "risk_level", risk_level)

    // Mask IP addresses Source and Destination (last octet)
    | eval masked_src_ip_address = replace(_raw.source_ip, /\.\d+$/, ".XXX")
    | eval _raw = json_set(_raw, "source_ip", masked_src_ip_address)

    | eval masked_dest_ip_address = replace(_raw.destination_ip, /\.\d+$/, ".XXX")
    | eval _raw = json_set(_raw, "destination_ip", masked_dest_ip_address)

    // Hash user_id with SHA256 and a salt
    | eval hashed_user_id = if(isnotnull(_raw.user_id), sha256(_raw.user_id +
"rmit_secret_salt"), null)
    | eval _raw = json_set(_raw, "user_id", hashed_user_id)

   // Redact reporting_user_or_process
    | eval masked_reporting = if(isnotnull(json_extract(_raw,
"reporting_user_or_process")), "REDACTED", null)
    | eval _raw = json_set(_raw, "reporting_user_or_process", masked_reporting)

    // Redact event_description if it contains PII (example: redact if contains 'user_')
    | eval event_desc = json_extract(_raw, "event_description")
    | eval redacted_desc = if(isnotnull(event_desc) and match(event_desc, /user_/),
"REDACTED_EVENT_DESC", event_desc)
    | eval _raw = json_set(_raw, "event_description", redacted_desc)

    // Drop Temporary fields used for processing
    | fields - masked_src_ip_address, masked_dest_ip_address, hashed_user_id,
masked_reporting, event_desc, redacted_desc, criticality_from_raw, risk_level

    // Filter out events with status = "Closed" or "Resolved" unless criticality is
"Tier0" or "Tier1"
    | eval status_from_raw = coalesce(json_extract(_raw, "status_code"),
json_extract(_raw, "status"))
    | eval criticality_check = json_extract(_raw, "criticality")
    | where (status_from_raw != "Closed" and status_from_raw != "Resolved") or
(criticality_check == "Tier0" or criticality_check == "Tier1")
```

```
    // Route based on event_type - extract from _raw and store in _raw for routing logic
    | eval event_type_from_raw = json_extract(_raw, "event_type")
    | eval event_route_value = case(event_type_from_raw == "cybersecurity_alert",
"CYBER_ROUTE", event_type_from_raw == "change_request_log", "CHANGE_ROUTE",
event_type_from_raw == "third_party_risk_update", "TPRM_ROUTE", true, "DEFAULT_ROUTE")
    | eval _raw = json_set(_raw, "event_route", event_route_value)

   // Drop Temporary fields used for processing
    | fields - criticality_check, event_type_from_raw, status_from_raw,event_route_value

     // Route to different indexes based on event_type
    | route _raw.event_route == "CYBER_ROUTE",
    [
        // Remove unwanted keys from `_raw` for cyber events so the archived payload is
compact
        | eval _raw = json_delete(_raw, "change_request_id", "change_description",
"change_status", "risk_impact_assessment", "planned_start_datetime_utc",
"actual_end_datetime_utc", "third_party_id", "vendor_name", "service_type",
"risk_assessment_update_reason", "overall_risk_rating", "key_risks_identified",
"due_diligence_status")
        | eval index = "rmit_cyber_events"
        | into $destination2
    ]
    | route _raw.event_route == "CHANGE_ROUTE",
    [
        // Remove unwanted keys from `_raw` for change events so the archived payload is
compact
        | eval _raw = json_delete(_raw, "cyber_event_id", "threat_type", "severity",
"status", "impacted_systems", "detection_source", "response_actions_taken",
"bnm_notification_sent", "bnm_report_ref_id", "third_party_id", "vendor_name",
"service_type", "risk_assessment_update_reason", "overall_risk_rating",
"key_risks_identified", "due_diligence_status")
        | eval index = "rmit_change_events"
        | into $destination3
    ]
    | route _raw.event_route == "TPRM_ROUTE",
    [
        // Remove unwanted keys from `_raw` for TPRM events so the archived payload is
compact
        | eval _raw = json_delete(_raw, "cyber_event_id", "threat_type", "severity",
"status", "impacted_systems", "detection_source", "response_actions_taken",
"bnm_notification_sent", "bnm_report_ref_id", "change_request_id", "change_description",
"change_status", "risk_impact_assessment", "planned_start_datetime_utc",
"actual_end_datetime_utc")
        | eval index = "rmit_tprm_events"
        | into $destination4
    ]
```

```
    // Drop fields associated with event_routes filtered before as they are empty
    | eval _raw = json_delete(_raw, "cyber_event_id", "threat_type", "severity", "status",
"impacted_systems", "detection_source", "response_actions_taken", "bnm_notification_sent",
"bnm_report_ref_id", "change_request_id", "change_description", "change_status",
"risk_impact_assessment", "planned_start_datetime_utc", "actual_end_datetime_utc",
"third_party_id", "vendor_name", "service_type", "risk_assessment_update_reason",
"overall_risk_rating", "key_risks_identified", "due_diligence_status")

    | eval index = "rmit_events"
    | into $destination;
```

## Save Your Pipeline

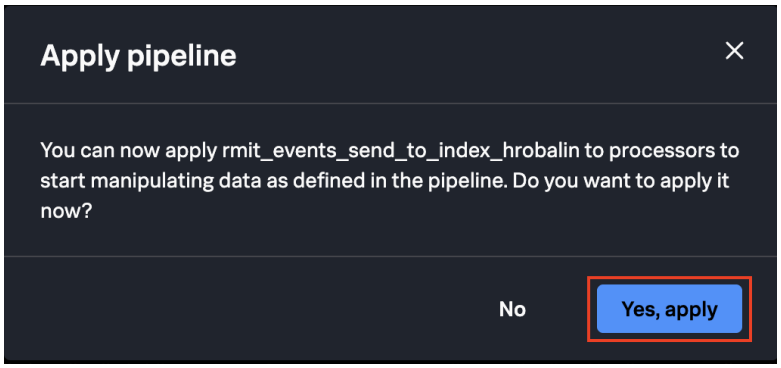
54. Save your pipeline by clicking on Save pipeline in the top right corner of the screen. Give your pipeline a suitable name, such as **rmit_events_send_to_index**_yourName> or something similar. As before, be sure to include your name or initials somewhere in the pipeline name to avoid confusion among those who are sharing the same environment as you!



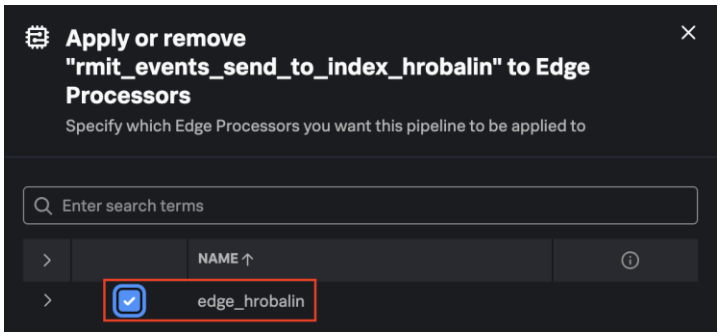


## Apply the Pipeline to Your Edge Processor

55. Now an “Apply Pipeline” dialog box will appear, click on “**Yes, apply**”.

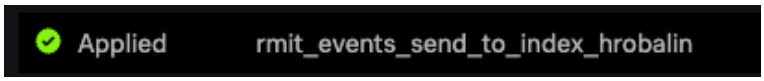



56. Check the box next to the name of the Edge Processor you created earlier and click on Save.




## Confirm RMiT Pipeline is applied

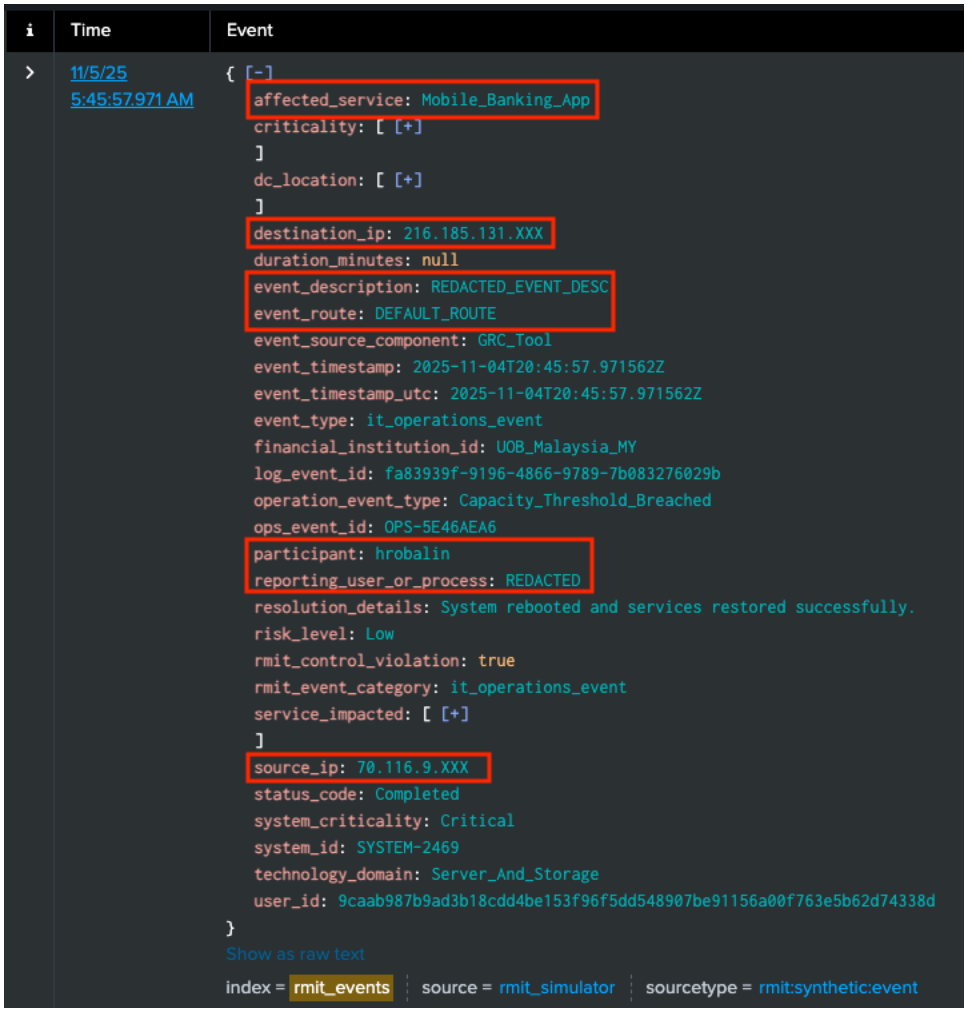
57. Make sure the RMiT Events Pipeline is applied. Navigate to your Pipelines, you should see your pipeline name, and a status if Applied or Unapplied. See image below:




## Check Your Data in Splunk Cloud

58. Log in to Splunk Cloud and open up the **Search & Reporting** app. Run the following search over the **last 15 minutes** and verify that you now see the redacted events.

```
index=rmit_events participant="hrobalin" ← Update the name!
```




Note that all the preprocessing, and enrichment that was done on this event. From redacting fields, to masking, hashing fields, and dropping fields. We can even verify that our events are routed properly.

# Exercise 8 – Buttercup Enterprises: ANZ Regulations (CPS-230)

## Description

In the ANZ region, Buttercup Enterprises must adhere to the CPS 230 standard, which focuses on operational risk management and resilience within the financial sector.

To comply with CPS 230, you will configure the Edge Processor to identify, filter, and securely route data related to operational risk events and controls. This involves creating a specialized pipeline for capturing, archiving, and analyzing relevant operational risk data, ensuring alignment with CPS 230 requirements for ongoing oversight and reporting.

## CPS 230 Pipeline Guide: A/NZ Regulations (Cross-industry Prudential Standard)

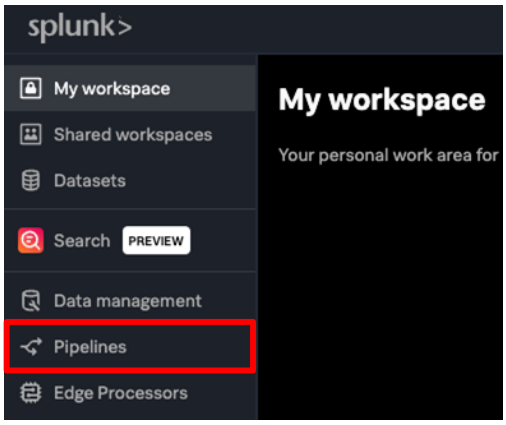
Objective: The Cross-industry Prudential Standard (CPS) 230 pipeline ensures compliance by identifying, filtering, and securely routing data related to operational risk events and controls.

1. **Enrich Data with Lookups:** Enhance events with asset inventory information (tolerance level, critical operation name, and business unit) using a lookup file.
2. **Define Risk Levels:** Assign a risk_level (High, Medium, Low) based on the criticality from the lookup.
3. **Mask Sensitive Operational Details:** Mask IP addresses, pseudonymize user IDs, mask reporting users/processes, and redact event descriptions if they contain PII.
4. **Filter Non-Actionable Logs:** Retain only relevant events based on status and criticality.
5. **Route Based on APRA Notification Candidate Status:** Direct events to different Splunk indexes (cps_apra_notify_events, cps_significant_events) based on whether they are APRA notification candidates.
6. **Minimize Data:** Remove low-value or empty fields from the default stream.
7. **Identify Your Data:** Add a custom field to easily identify your events in a shared Splunk Cloud environment.

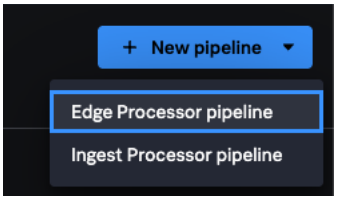
## Steps

### Create a New Pipeline

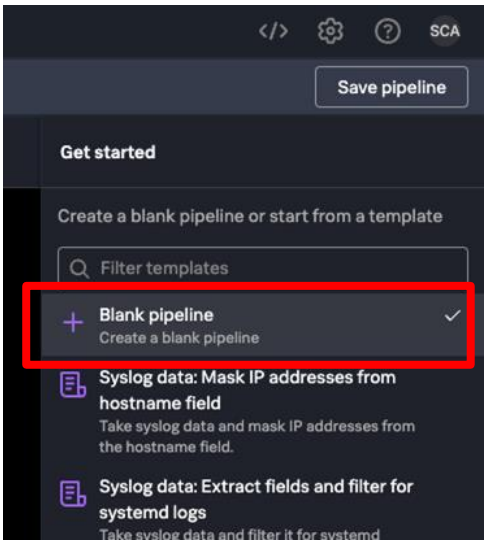
1. While logged in to the Edge Processor service (see Access Edge Processor Service) click on **Pipelines** on the left of the page.



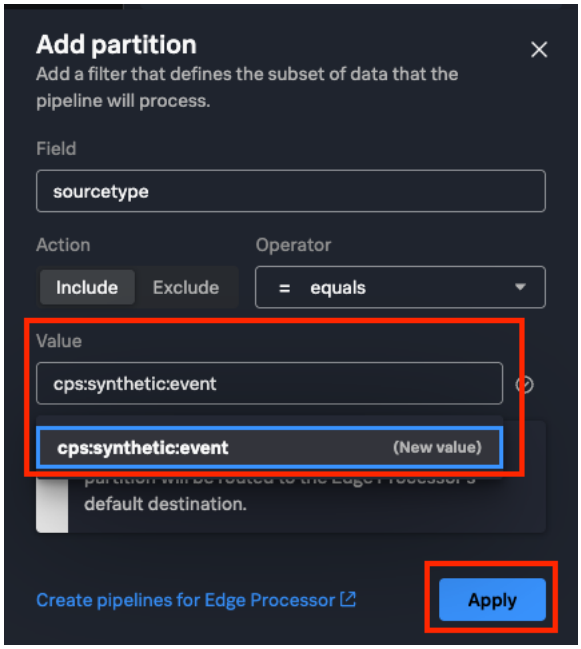

2. In the top right corner of the **Pipelines** page click on **+ New pipeline**. Select **Edge Processor pipeline**




3. On the right hand side of the **New pipeline** page under the **Get started** pane select on **Blank pipeline** and click on **Next**.

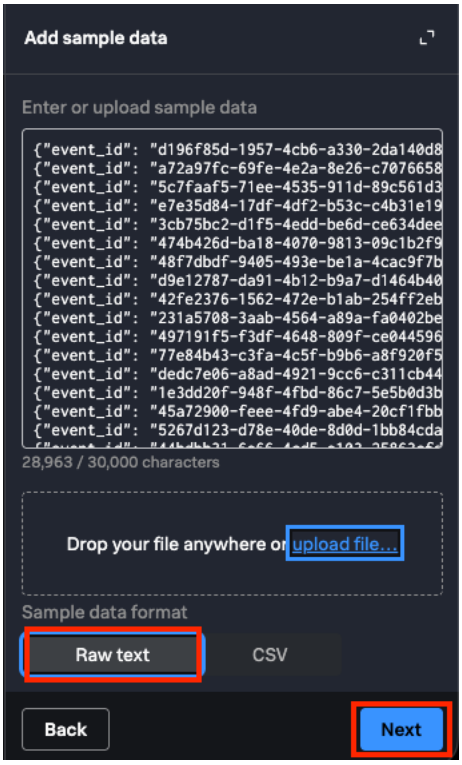


4. On the **Define your pipeline's partition** pane click on the ( + ) button next to the Partition field and select the value **sourcetype** for **Field**, select "Keep" for Action, "**= equals**" for Operator and write **cps:synthetic:event** for the **value (this will be a new value)** and click **Apply**.



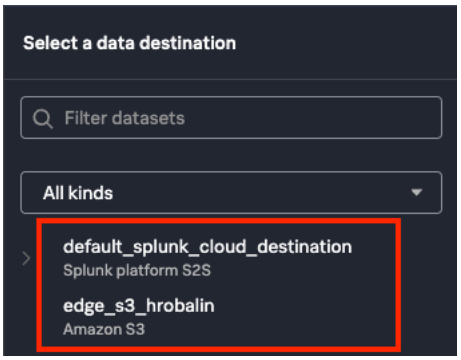

And then click on **Next** button at the bottom right.

5. An **Add sample data** menu will appear on the right side. Upload the **cps_sample_events.jsonl** file using the "**upload file…**" option at the bottom.



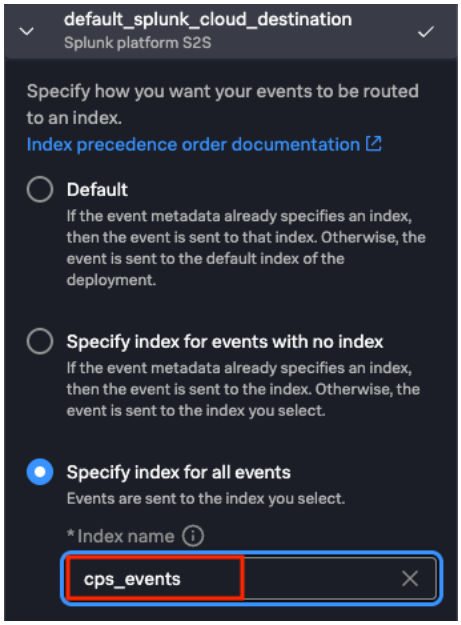

For the **Sample data format** select **Raw text** and click on **Next**.

6. A "**Select a data destination"** menu will appear. Select the **default_splunk_cloud_destination** as your data destination and click on **Next**.

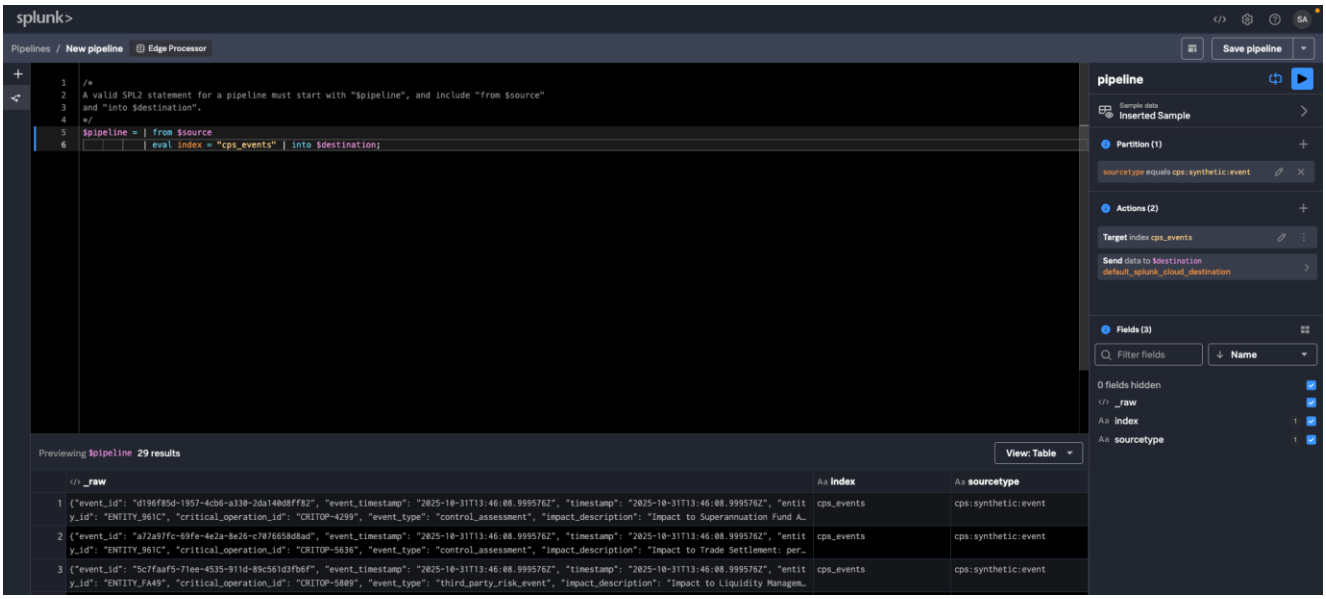


## Route Your Data to the cps_events Index

7. A pane below will open, there you will select **Specify index for all events** and set the index to the **cps_events** from the drop down, as requested by the Buttercup security team, and click on **Done** at the bottom.




8. A screen like below will appear:

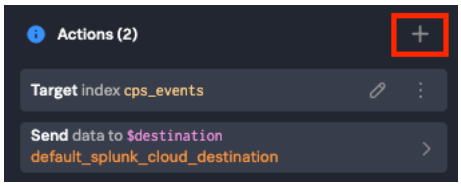


As you integrate additional actions into your pipeline, please insert them between the existing pipeline stages and the final index. It is crucial to maintain the order specified in this guide. Refer to the example below:

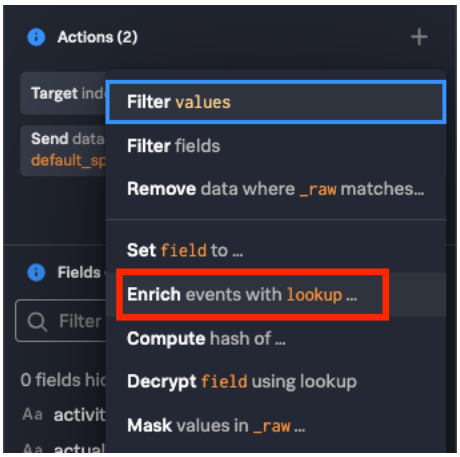
```
$pipeline = | from $source
. . . ADD NEW ACTIONS HERE
| eval index = "cps_events"
| into $destination;
```

## Configure Lookup for Events

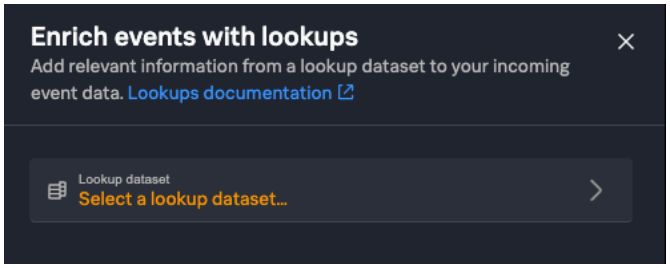
9. We want to first start by enriching this data, it will provide us more context to an event coming in as well, as allow us to do advance routing based on enriched data. To accomplish this, we will import a lookup using Edge Processing, through the **Actions** menu on the right side. Press the Plus button on the right side.



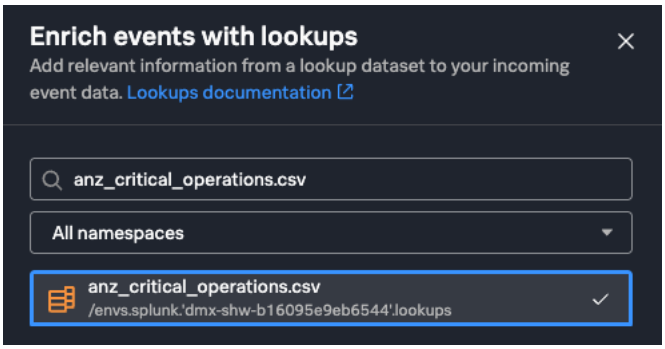

10. We will select: **Enrich** events with lookup



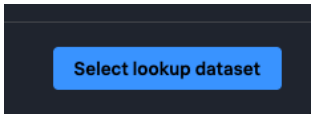

11. Now we need to select a Lookup dataset.




12. In our example we will search for **anz_critical_operations.csv**, and select the option that comes up.




13. Make sure to press the button on the bottom right, labeled: **Select lookup dataset**



14. Now we need to configure the Lookup fields to match, and the fields to output from the Lookup.

15. For **Match fields** configure the following:

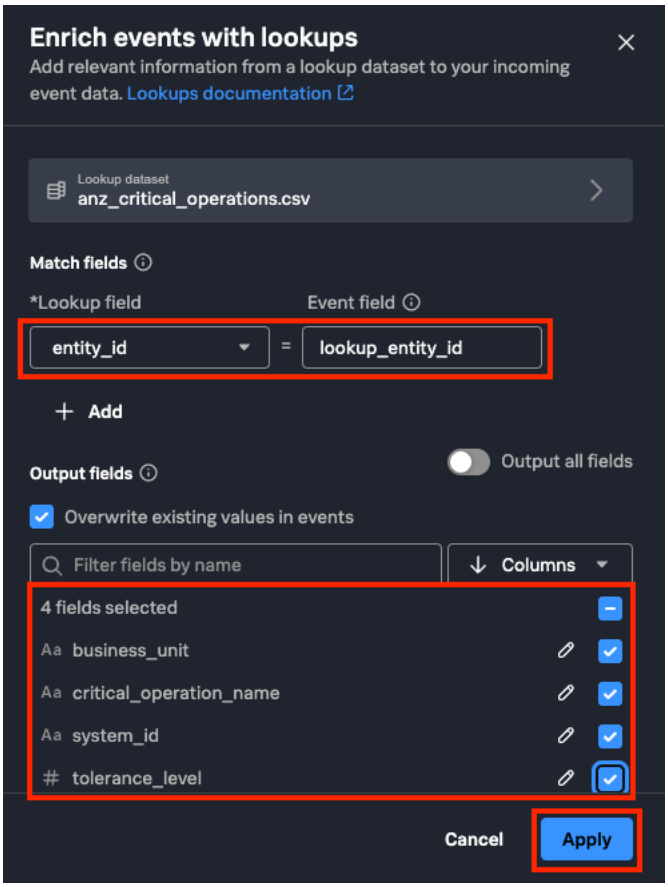
    - Lookup field as **entity_id**
    - Event field as **lookup_entity_id**

16. For **Output fields** select the following

    - **business_unit**
    - **critical_operation_name**
    - **system_id**
    - **tolerance_level**

17. Confirm your configuration matches the image on the right.

18. Select **Apply.**




19. Confirm lookup file is added to pipeline, and ready to use.

    To confirm that you have added your lookup file correctly, you should see the following line at the top of your pipeline (be aware the dmx instance id may be different for your environment) :

```
import anz_critical_operations.csv /from /envs.splunk.'dmx-shw-
2c53feabe860f8'.lookups
```

20. You'll observe that the IDE has placed the lookup SPL line after **| eval index="cps_events".** To ensure that our events are enriched before any potential routing decisions are made based on that enrichment, the lookup must occur earlier in the pipeline. Therefore, we will move this lookup line to immediately follow **| from $source**.

    The revised pipeline should appear as follows:

```
import 'anz_critical_operations.csv' from /envs.splunk.buttercup.lookups

$pipeline =
| from $source
    // Lookup enrichment
    | lookup 'anz_critical_operations.csv' entity_id OUTPUT system_id,
critical_operation_name, tolerance_level, business_unit
    | eval index = "cps_events"
    | into $destination;
```

21. Before we can enrich our event data, we must ensure we have a reliable key to connect it to our external asset inventory.

22. Below is our lookup file; pay close attention to the header names:

```
system_id,entity_id,critical_operation_name,tolerance_level,business_unit
sys_001,ENTITY_6885,Payments Processing,60,Operations
sys_002,ENTITY_961C,Customer Account Management,180,Operations
```

## Lookup Enrichment Pattern for Nested JSON Events

In this section, we demonstrate how to enrich events that arrive wrapped in a _raw JSON field by performing a lookup and injecting the results back into the event.

### Extract Lookup Key from _raw

23. Edge Processor lookups require top-level fields to match against CSV data. Since our events arrive wrapped in _raw, we first extract the entity_id (which identifies the critical operation) into a temporary top-level field.

### Perform the Lookup

24. We match the extracted **entity_id** against the CSV lookup table's **entity_id** column. The lookup returns four enrichment fields:

    - **system_id**: System identifier for the critical operation
    - critical**_operation_name**: Human-readable operation name (e.g., "Payments Processing")
    - **tolerance_level**: Maximum downtime tolerance in minutes
    - **business_unit**: Department that owns the operation

### Inject Enrichment back into _raw

25. Using json_set(), we add each lookup result back into the _raw JSON object. Each line includes a null check to ensure we only inject valid values. This ensures the enrichment travels with the event through all subsequent transformations and routing stages, so the archived record contains complete context about which critical operation was affected.

### Clean Up Temporary Fields

26. We remove the temporary top-level fields after injection. This prevents duplication—the enrichment now lives in _raw where it belongs, and the top-level fields are no longer needed.

### Why This Pattern Matters

27. By injecting enrichment into _raw, we preserve the complete enriched event in the archived data. When analysts or compliance officers query these events later, they can immediately see which critical operation was affected, its tolerance level, and business unit—without needing to perform a separate lookup at query time. This is especially important for compliance audits where event

context must be frozen and immutable.

28. The above can be implemented in the following SPL:

```
    // Extract entity_id from _raw for lookup key building
    | eval lookup_entity_id = json_extract(_raw, "entity_id")

    // Lookup enrichment: match on entity_id from events to entity_id in CSV
    | lookup 'anz_critical_operations.csv' entity_id AS lookup_entity_id
OUTPUT system_id, critical_operation_name, tolerance_level, business_unit

    // Inject lookup results into _raw so enrichment travels with the event
    | eval _raw = if(isnotnull(system_id), json_set(_raw, "system_id",
system_id), _raw)
    | eval _raw = if(isnotnull(critical_operation_name), json_set(_raw,
"critical_operation_name", critical_operation_name), _raw)
    | eval _raw = if(isnotnull(tolerance_level), json_set(_raw,
"tolerance_level", tolerance_level), _raw)
    | eval _raw = if(isnotnull(business_unit), json_set(_raw, "business_unit",
business_unit), _raw)

    // Remove temporary top-level lookup fields to avoid duplication (after
injecting into _raw)
    | fields - lookup_entity_id, system_id, critical_operation_name,
tolerance_level, business_unit
```

29. To implement the above, copy, and replace the current lookup line added by the editor.

### Define Enrichment with Critical Operation (CO) and Tolerance Context

30. To further enrich our data, we will assign a severity_numeric to each event, based on the incident's severity. We will achieve this by extracting the relevant information from the _raw field and using a case statement to convert it into a numerical value. This numerical assignment is particularly useful for optimizing computational processes.

    Insert the following SPL into your pipeline, immediately after the previous addition.

```
    // Extract severity from _raw and convert to numeric for filtering
    | eval cps230_incident_severity = json_extract(_raw,
"cps230_incident_severity")
    | eval severity_numeric = case(
        cps230_incident_severity == "Critical", 4,
        cps230_incident_severity == "High", 3,
        cps230_incident_severity == "Medium", 2,
        cps230_incident_severity == "Low", 1,
        true, 0
    )
```

## Define Filtering for Significance and Actionability

31. Next in our step we need to filter events based on the following:

    - APRA Notifications Candidate is True
    - Severity Level: Critical & High

32. To accomplish this, we can use the SPL below:

```
// Filtering: keep events that are APRA notification candidates OR have high
severity

| where json_extract(_raw, "apra_notification_candidate") == true OR
severity_numeric >= 3
```

## Define Data Masking and Hashing

33. We have filtered the events we want to keep. Before we can proceed and route these events, we will need to mask, and hash

```
    // Pseudonymize reporter_user_id
    | eval reporter_user_id = json_extract(_raw, "reporter_user_id")
    | eval hashed_reporter_id = if(isnotnull(reporter_user_id),
sha256(reporter_user_id + "cps230_salt"), null)
    | eval _raw = json_set(_raw, "reporter_user_id", hashed_reporter_id)

    // Mask vendor_name
    | eval vendor_name = json_extract(_raw, "vendor_name")
    | eval masked_vendor = if(isnotnull(vendor_name), "REDACTED", null)
    | eval _raw = json_set(_raw, "vendor_name", masked_vendor)

    // Drop temporary fields used for processing
    | fields - lookup_entity_id, system_id, critical_operation_name,
tolerance_level, business_unit, reporter_user_id, hashed_reporter_id,
vendor_name, masked_vendor, cps230_incident_severity, severity_numeric
```

## Define Archival Routing

34. Now that we have filtered, hashed, and masked our data, we must archive it before sending it to Splunk.

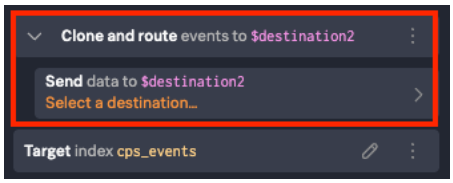
    This can be done in two ways, via the UI, or via the pipeline itself. In our case we will manually do it. The reason for this is that if we do via the UI, the UI will put the archive query in the wrong location.

35. Similar as we have done before let's paste the following SPL2 statement after the above SPL.

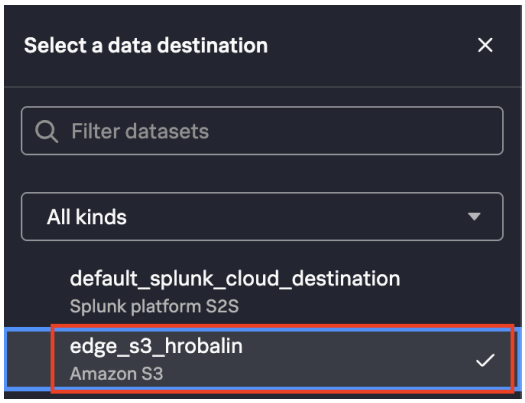
    This will define Splunk as $destination, and Amazon S3 Bucket for Archiving as $destination2.

```
// Send raw data to secure archive (e.g., Amazon S3 Bucket)
| thru [ | into $destination2 ]
```

36. Now let's define in the pipeline setting which Amazon S3 Bucket will be $destination2. Click Select a destination

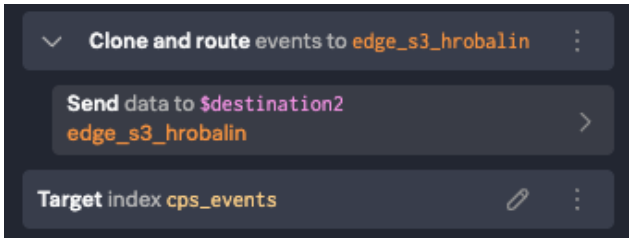



37. Select the Amazon S3 Endpoint for our use case: edge_s3_username.




38. Click **Apply** at the bottom right.

39. **Actions** should now look like this, where $destination2 is an S3 Destination:




## Route Processed Data Based on CPS 230 APRA Notification

40. Following the archival of all events, our focus shifts to filtering for 'APRA notification candidates,' whose criteria are outlined below:

- apra_notification_candidate -> true
- apra_notification_candidate -> false

41. Next, we will go ahead and start adding our routing. We can do this via the Action Menu on the right side. For our case, we will avoid this, the reason is that adding via the Action Menu will add

the route to the wrong location and require extra configuration. To avoid confusion, we will manually add a route, and in a real world this is the way you would do it.

42. Since we are going the manual way for Routes, we need to manually import the route function in the pipeline, we do this by adding the following line to the top of our pipeline, above where the pipeline is importing our lookup.

```
import route from /splunk.ingest.commands
```

43. Now let's define what fields we are routing, to what destination, and their respective indexes:

- apra_notification_candidate = true
  - Index = cps_apra_notify_events
  - $destination3
- apra_notification_candidate = false
  - Index = cps_significant_events
  - $destination4

44. This is our SPL for true:

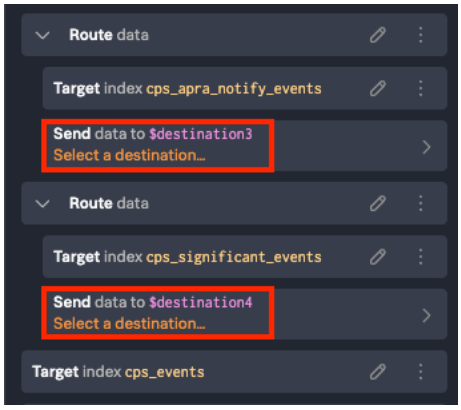
```
// Route APRA notification candidate events to APRA notifications index
| route _raw.apra_notification_candidate == true,
[
    | eval index = "cps_apra_notify_events"
    | into $destination3
]
```

45. This is our SPL for false:

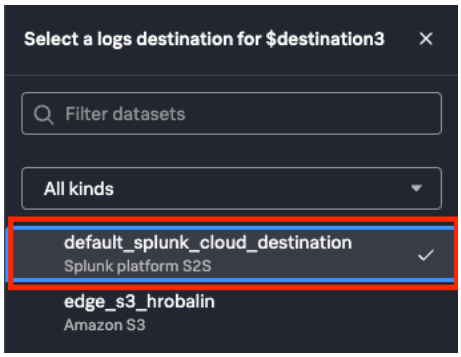
```
// Route non-APRA events to significant events index
| route _raw.apra_notification_candidate == false,
[
    | eval index = "cps_significant_events"
    | into $destination4
]
```

46. Navigate to the 'Route Section' located under 'Actions' on the right-hand side (Make sure to scroll down).

    You'll notice that we need to **Select a destination**

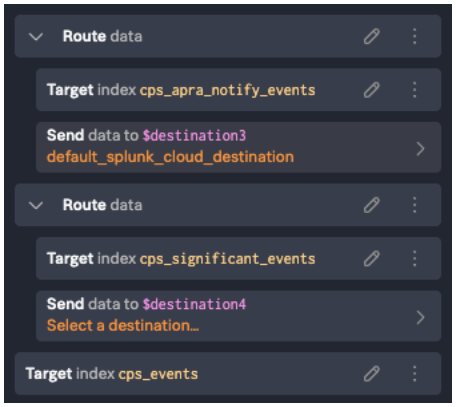



47. Click on Select a destination for $destination3, a new pane will appear, select default_splunk_cloud_destination.

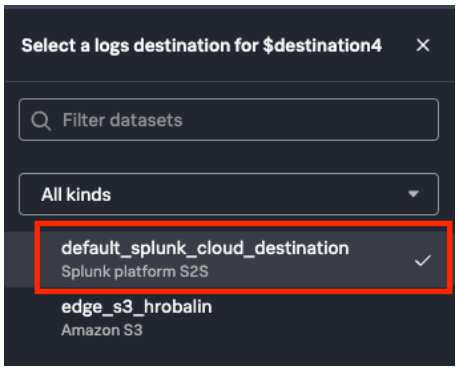



    Click on **Apply** at the bottom right.

48. Let's confirm our selection is updated:



49. Click on Select a destination for $destination4, a new pane will appear, select default_splunk_cloud_destination.

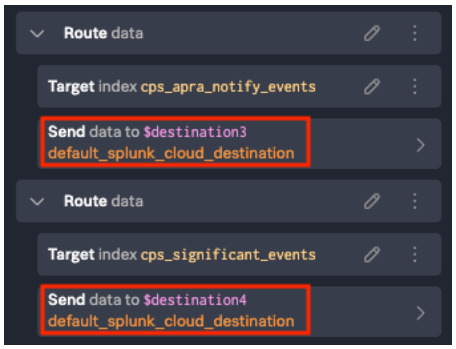



Click on **Apply** at the bottom right.

50. Let's confirm that we have configured both correctly.

Navigate to the 'Route Section' located under 'Actions' on the right-hand side (Make sure to scroll down).

We should see something like the following:




51. We have now configured two distinct destinations for specific events, enabling routing to indexes monitored by different teams and supporting use cases that demand rapid notification.

## Add a Custom Indexed Field to Your Data

52. Finally, add a '**participant**' field to your events to help you identify them in Splunk Cloud.

In this case make sure to add the line to **before** the Route to specific indexes.

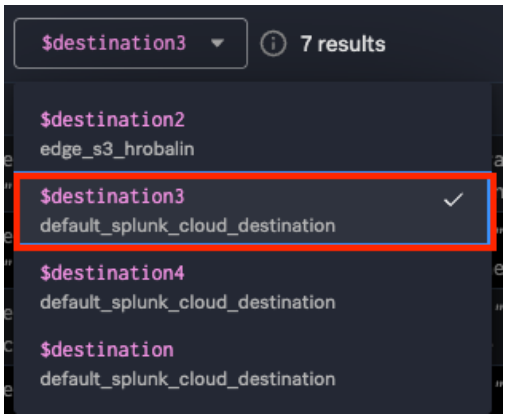
```
$pipeline =
| from $source
...
| eval _raw = json_set(_raw, "participant", "hrobalin") ← Update with your
   own name
//Route to different Indexes based on APRA Notifications
```

```
...
| index = “cps_events”
| into $destination;
```

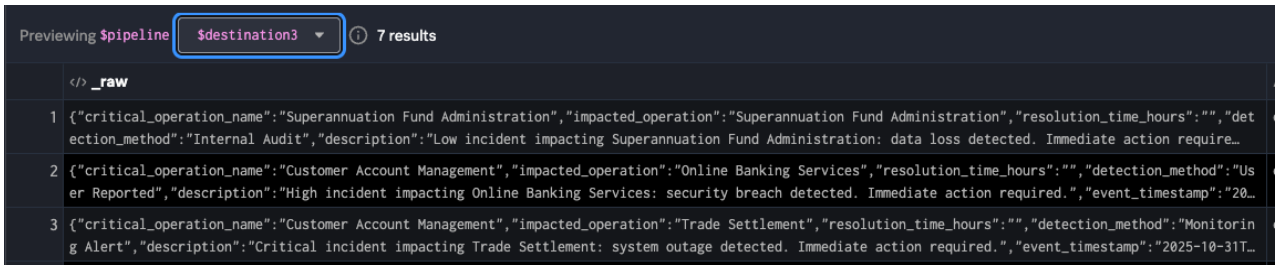
## Preview Your Pipeline

53. Test your pipeline by clicking on the blue Preview pipeline button ( ) in the top right corner of the screen. You should see that the pipeline window has a dropdown, this dropdown allows us to select the preview for all the different destinations configured.

    We will select $destination3




    Now we should see something like the below:




    If we inspect the other destination such as $destination and $destination4, you will notice that the preview is empty. This is not a mistake.

    The reason for this is that the CPS pipeline demonstrates selective event routing based on compliance criticality. The ‘where’ clause filters to only high-impact events: those flagged as APRA notification candidates (regulatory requirement) OR events with High/Critical severity. This significantly reduces data volume while preserving compliance-critical events. Most low-severity events with null severity ratings are filtered out before reaching the routing stage, which is why $destination3 (APRA notifications) and $destination2 (archive) see more traffic than $destination4 (general significant events).

## CPS Pipeline

54. Your CPS Pipeline SPL should look like the following (Be aware copying below will require updating env name on second line:

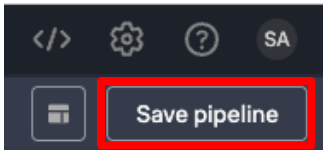
```
import route from /splunk.ingest.commands
import 'anz_critical_operations.csv' from /envs.splunk.'dmx-shw-
a602e5f186a868'.lookups

$pipeline =
| from $source

    // Extract entity_id from _raw for lookup key building
    | eval lookup_entity_id = json_extract(_raw, "entity_id")

    // Lookup enrichment: match on entity_id from events to entity_id in CSV
    | lookup 'anz_critical_operations.csv' entity_id AS lookup_entity_id
OUTPUT system_id, critical_operation_name, tolerance_level, business_unit

    // Inject lookup results into _raw so enrichment travels with the event
    | eval _raw = if(isnotnull(system_id), json_set(_raw, "system_id",
system_id), _raw)
    | eval _raw = if(isnotnull(critical_operation_name), json_set(_raw,
"critical_operation_name", critical_operation_name), _raw)
    | eval _raw = if(isnotnull(tolerance_level), json_set(_raw,
"tolerance_level", tolerance_level), _raw)
    | eval _raw = if(isnotnull(business_unit), json_set(_raw, "business_unit",
business_unit), _raw)

    // Remove temporary top-level lookup fields to avoid duplication (after
injecting into _raw)
    | fields - lookup_entity_id, system_id, critical_operation_name,
tolerance_level, business_unit

    // Extract severity from _raw and convert to numeric for filtering
    | eval cps230_incident_severity = json_extract(_raw,
"cps230_incident_severity")
    | eval severity_numeric = case(
        cps230_incident_severity == "Critical", 4,
        cps230_incident_severity == "High", 3,
        cps230_incident_severity == "Medium", 2,
        cps230_incident_severity == "Low", 1,
        true, 0
    )

    // Filtering: keep events that are APRA notification candidates OR have
high severity
```

```
    | where json_extract(_raw, "apra_notification_candidate") == true OR
severity_numeric >= 3

    // Pseudonymize reporter_user_id
    | eval reporter_user_id = json_extract(_raw, "reporter_user_id")
    | eval hashed_reporter_id = if(isnotnull(reporter_user_id),
sha256(reporter_user_id + "cps230_salt"), null)
    | eval _raw = json_set(_raw, "reporter_user_id", hashed_reporter_id)

    // Mask vendor_name
    | eval vendor_name = json_extract(_raw, "vendor_name")
    | eval masked_vendor = if(isnotnull(vendor_name), "REDACTED", null)
    | eval _raw = json_set(_raw, "vendor_name", masked_vendor)

    // Drop temporary fields used for processing
    | fields - lookup_entity_id, system_id, critical_operation_name,
tolerance_level, business_unit, reporter_user_id, hashed_reporter_id,
vendor_name, masked_vendor, cps230_incident_severity, severity_numeric

    // Send raw data to secure archive (e.g., Amazon S3 Bucket)
    | thru [ | into $destination2 ]

    | eval _raw = json_set(_raw, "participant", "hrobalin")

    // Route to different indexes based on APRA notification status
    // Route APRA notification candidate events to APRA notifications index
    | route _raw.apra_notification_candidate == true,
    [
        | eval index = "cps_apra_notify_events"
        | into $destination3
    ]

    // Route non-APRA events to significant events index
    | route _raw.apra_notification_candidate == false,
    [
        | eval index = "cps_significant_events"
        | into $destination4
    ]

    | eval index = "cps_events"
    |  into $destination;
```

## Save Your Pipeline

55. Save your pipeline by clicking on Save pipeline in the top right corner of the screen. Give your pipeline a suitable name, such as **cps_events_send_to_index_<yourName>** or something similar. As before, be sure to include your name or initials somewhere in the pipeline name to

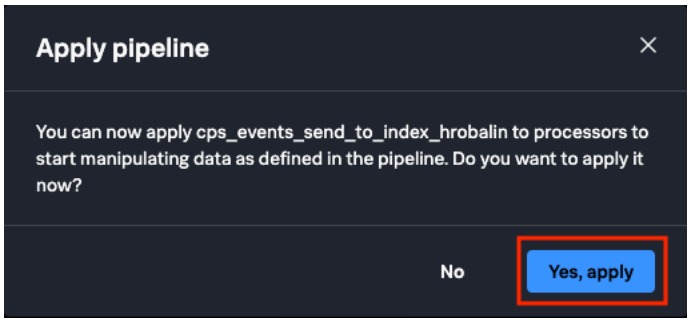
avoid confusion among those who are sharing the same environment as you!



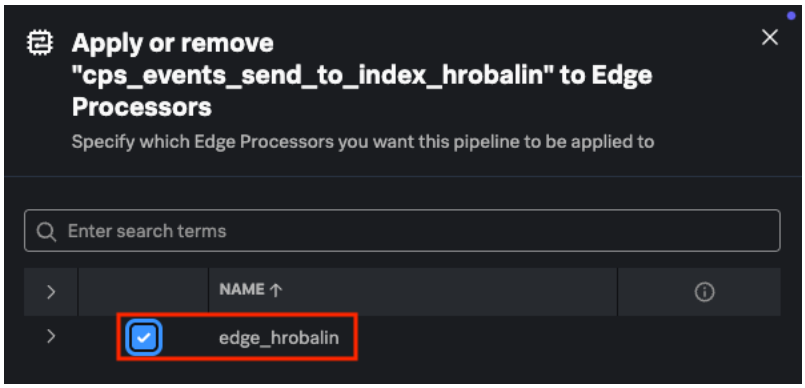


## Apply the Pipeline to Your Edge Processor

56. Now an “Apply Pipeline” dialog box will appear, click on “**Yes, apply**”.

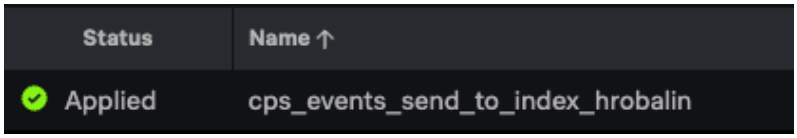



57. Check the box next to the name of the Edge Processor you created earlier and click on **Save**.



## Confirm  CPS Pipeline is applied

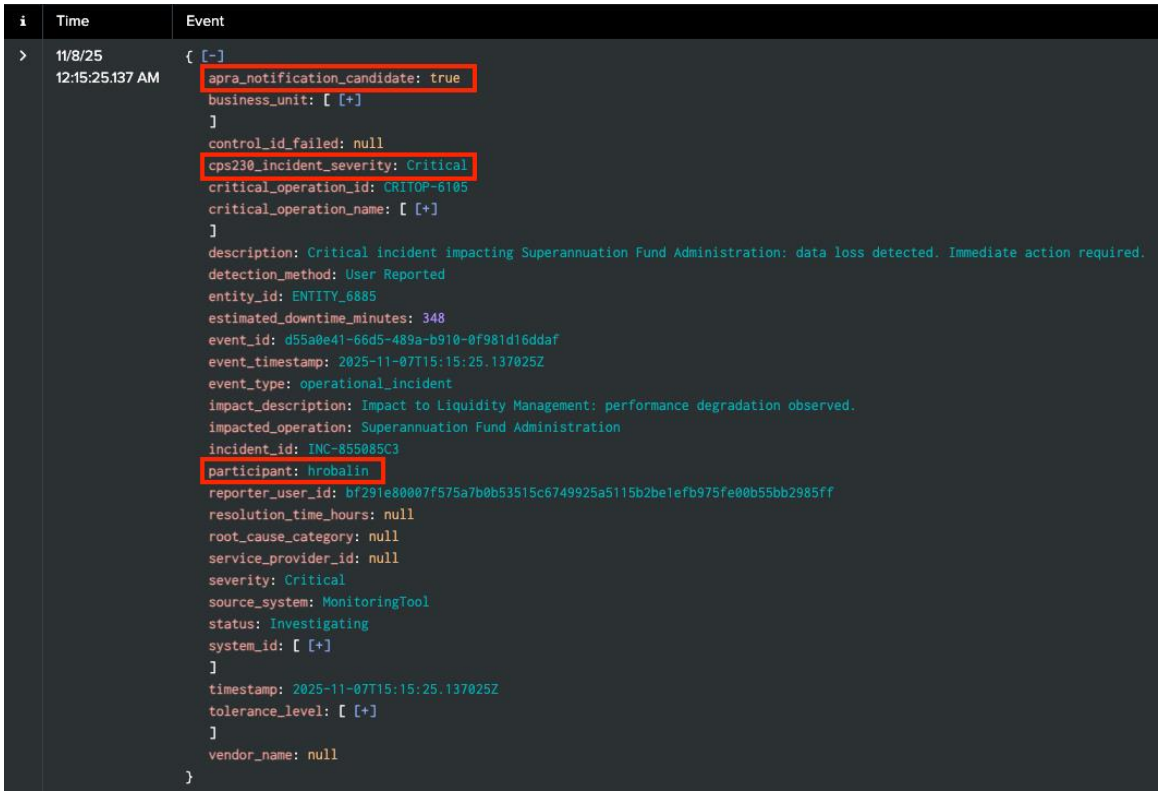
58. Make sure the CPS Events Pipeline is applied. Navigate to your Pipelines, you should see your pipeline name, and a status if Applied or Unapplied. See image below:




## Check Your Data in Splunk Cloud

59. Log in to Splunk Cloud and open up the **Search & Reporting** app. Run the following search over the **last 15 minutes** and verify that you now see the redacted events.

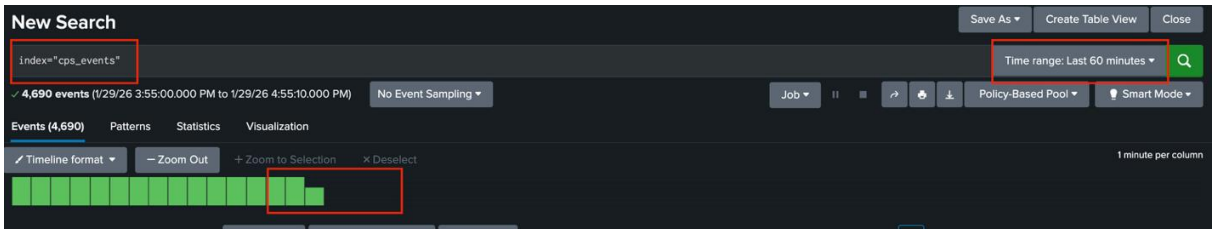
```
index=cps_apra_notify_events participant=hrobalin
```
← **Update the name!**




Note that since our search results will only contain those events that match our IP address range you may need to wait up to a couple of minutes for events matching that range to arrive at Splunk. If you don't get any results immediately try re-running the search after a minute or two.

60. You will notice that if you do a search for index=cps_events you will see no results for last 15 mins. If we expand the range to an hour the answer becomes obvious. We do get events, until we enabled the Edge Processor Pipeline for CPS. This is when our filtering becomes effective, and we can see a way that some events are not routed for storage, others are enriched before ingestion.

See image below for example.




61. You will notice that if you do a search for index=cps_significant_events you will see no results. Even if you increase the time range. The reason is that so far the pipeline has not found an event that meets all the requirements in the ingestion data to allow to get ingested into Splunk.

**You've finished!**

# Get Started with Splunk Data Management today

### Request access to Edge or Ingest Processor

Fill out this form to request access to Data Management Pipeline Builders. If you already have access, you can directly navigate to it using the instructions on this form.

### Join the Slack channel

Get direct access to the Data Management team, ask questions, get help, and collaborate with the Community. Request access here.

### Splunk Adoption Hub

Access curated Splunk resources to get started and maximize value—all in one place. Tailored to your needs and regularly updated by Splunk experts.