Indexing Large Data Sets with Watsonx Discovery S3 Connector

Introduction

Watsonx Discovery comes with a lot of capabilities providing a lot of features and connectors to leverage from. While the Q&A with RAG Accelerator provides an efficient approach for data ingestion and indexing, indexing of very large datasets (terabytes and so) can be enhanced using connector feature provided by Watsonx Discovery. Large datasets can be stored in Amazon S3 bucket and using an S3 connector with Watsonx Discovery, documents can be directly ingested and indexed to Elasticsearch vector store.

Key tasks involved tin Indexing Large Datasets from S3:

- **Setup self-managed S3 connector** to connect your Amazon S3 bucket to Watsonx Discovery.
- **Setup Elasticsearch pipeline** for chunking and indexing of the documents.
- **Setup self-managed Content Extraction Service** to process large documents (e.g., documents exceeding 100MB).

Prerequisites

- The datasets to be indexed are present in AWS S3 bucket in pdf format.
- An AWS EC2 instance is available to execute the S3 connector.

Processing Compressed Files with AWS Lambda and S3 Connector

Since the S3 connector cannot directly process compressed files, you will first need to extract the files from the compressed archives and store them in a new S3 bucket. AWS Lambda can handle this extraction automatically when a new compressed file (such as .tar or .tar.gz) is uploaded to the source S3 bucket.

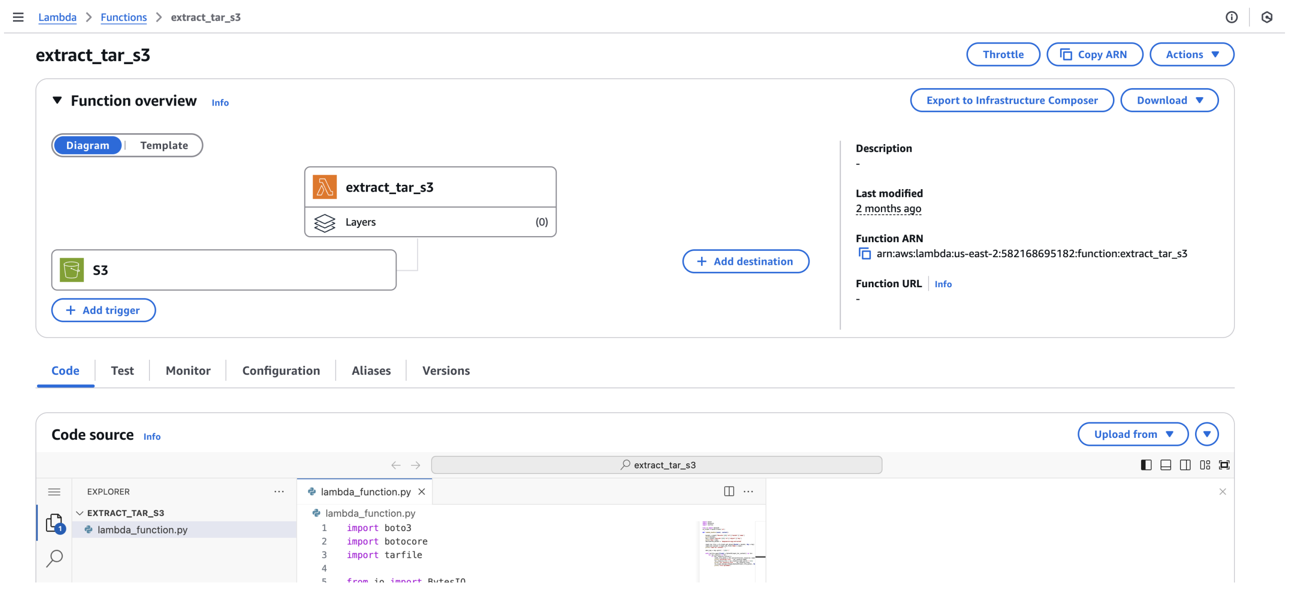
Steps Overview:

- Create AWS IAM Roles and Policies for Lambda to access and interact with S3.

- Create a Lambda function to extract and store the files from compressed archives. Here is a sample Lambda function written in Python that handles the extraction of .tar files from an S3 bucket:

```
    import boto3
    import botocore
    import tarfile

    from io import BytesIO
    s3_client = boto3.client('s3')

    def lambda_handler(event, context):

        bucket = event['Records'][0]['s3']['bucket']['name']
        key = event['Records'][0]['s3']['object']['key']
        destination_bucket = bucket+'_extracted'
        input_tar_file = s3_client.get_object(Bucket = bucket, Key =
key)
        input_tar_content = input_tar_file['Body'].read()

        dest_key = key.split('.')[0]+'/'

        with tarfile.open(fileobj = BytesIO(input_tar_content)) as
tar:
            for tar_resource in tar:
                if (tar_resource.isfile()):
                    inner_file_bytes =
tar.extractfile(tar_resource).read()
                    print('extracted file: ',tar_resource.name)
                    tar_resource_file = tar_resource.name.split('/')
[1]
                    print('tar_resource_file: ', tar_resource_file)

s3_client.upload_fileobj(BytesIO(inner_file_bytes),
                            Bucket = destination_bucket,
                            Key = dest_key+tar_resource_file)
                    print('file uploaded')
```

- Set up the trigger for Lambda to automatically execute upon file upload in the S3 bucket.
- Create a destination S3 bucket to store the extracted files.



Description
-
Last modified
2 months ago
Function ARN
arn:aws:lambda:us-east-2:582168695182:function:extract_tar_s3
Function URL Info
-

```
import boto3
import botocore
import tarfile

from io import BytesIO
```

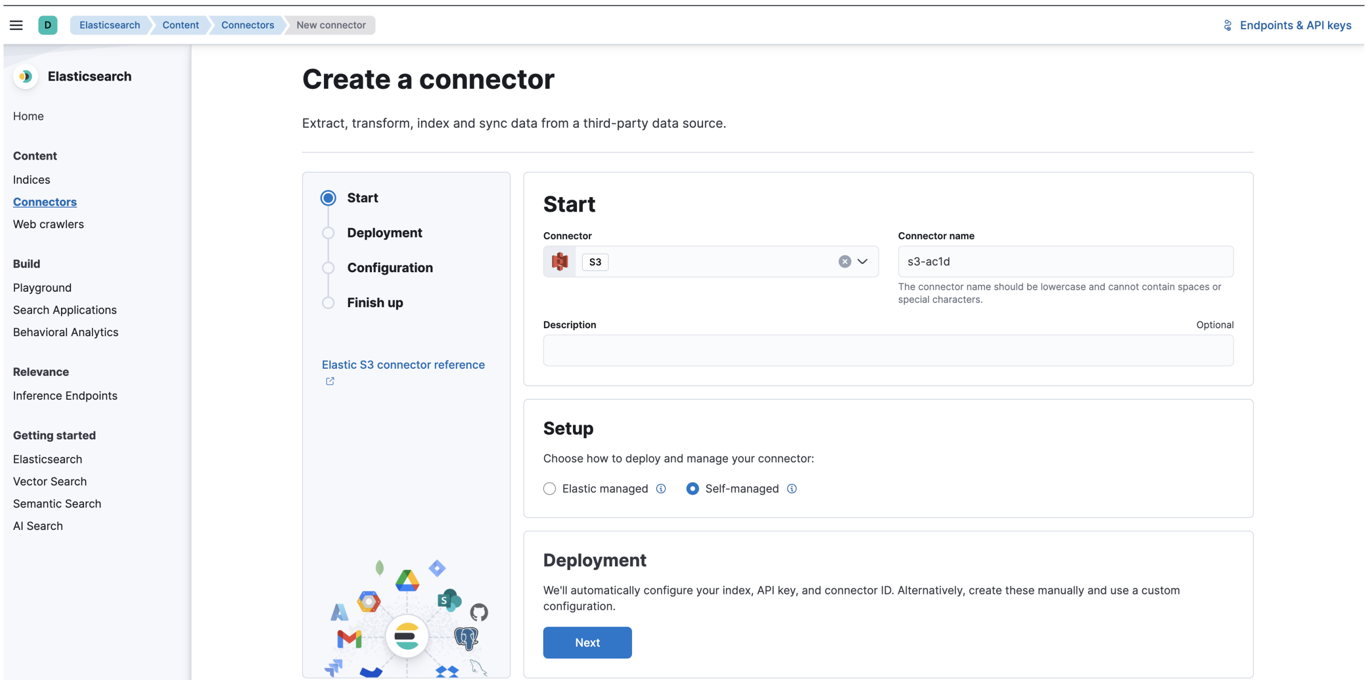
Figure: AWS Lambda Function

- Upon execution of the above created Lambda function, the destination bucket will get populated with the PDF files extracted from the .tar archive, preserving the same directory structure as the source S3 bucket.

Setting up the S3 Connector on wx-discovery Kibana

To integrate the pipeline, the S3 connector needs to be set up within wx-discovery Kibana. Follow the steps below to configure the S3 connector:

1. Navigate to Kibana:
   - Access wx-discovery Kibana and go to Elasticsearch.
   - In the left-hand menu, click on Connectors.
2. Create a New Connector:
   - Click on New Connector to begin the setup process.
   - Select S3 as the connector source.
3. Configure the Connector:
   - A default connector name will be auto-generated, which you can update as needed.
   - Choose Setup as self-managed.



Create a connector

Extract, transform, index and sync data from a third-party data source.

Start

Connector: S3

Connector name: s3-ac1d

The connector name should be lowercase and cannot contain spaces or special characters.

Description — Optional

Setup

Choose how to deploy and manage your connector:

Elastic managed [✓ Self-managed]

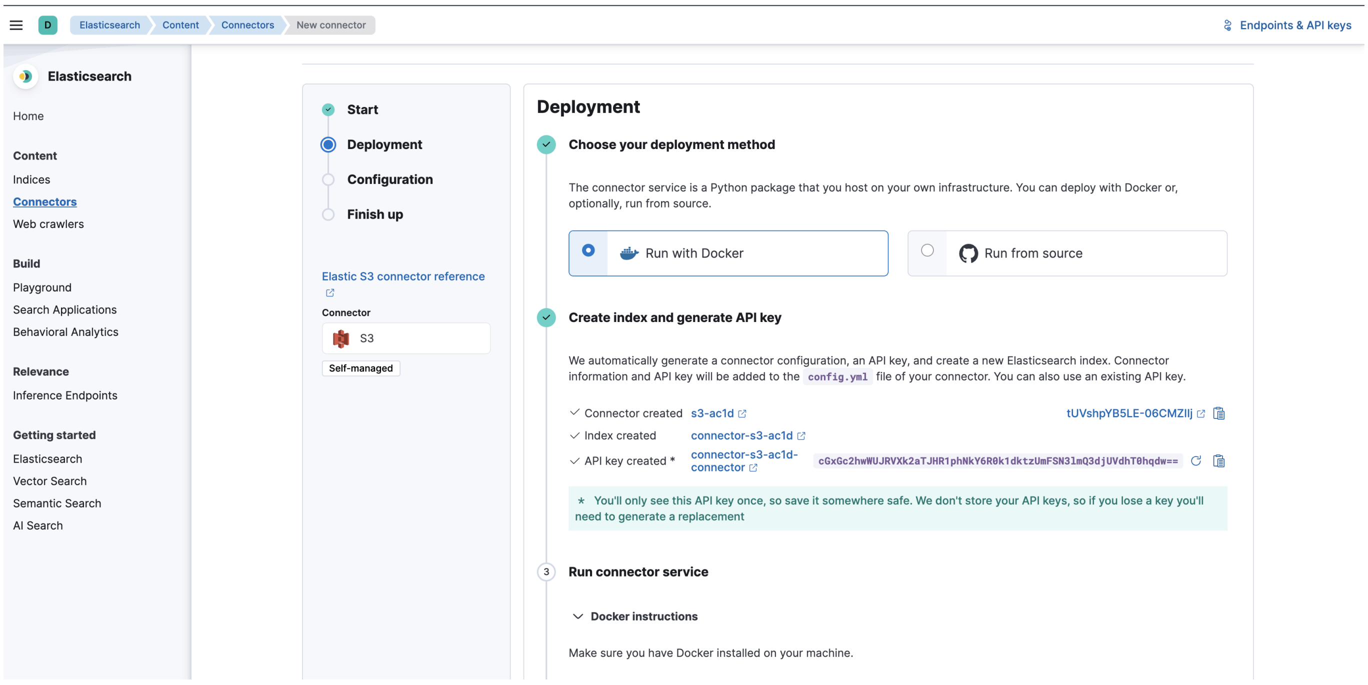
Deployment

We'll automatically configure your index, API key, and connector ID. Alternatively, create these manually and use a custom configuration.

Figure: Kibana Connector Setup Step

```
- Click Deployment and then select Next to continue.
- Choose "Run with Docker" as the deployment method.
```

4. Connector ID:
   - A new connector_id will be generated during the setup.



Deployment

Choose your deployment method

The connector service is a Python package that you host on your own infrastructure. You can deploy with Docker or, optionally, run from source.

[✓ Run with Docker] Run from source

Create index and generate API key

We automatically generate a connector configuration, an API key, and create a new Elasticsearch index. Connector information and API key will be added to the `config.yml` file of your connector. You can also use an existing API key.

✓ Connector created s3-ac1d — tUVshpYB5LE-06CMZIlj
✓ Index created connector-s3-ac1d
✓ API key created * connector-s3-ac1d-connector `cGxGc2hwWUJRVXk2aTJHR1phNkY6R0k1dktzUmFSN3lmQ3djUVdhT0hqdw==`

* You'll only see this API key once, so save it somewhere safe. We don't store your API keys, so if you lose a key you'll need to generate a replacement

Run connector service

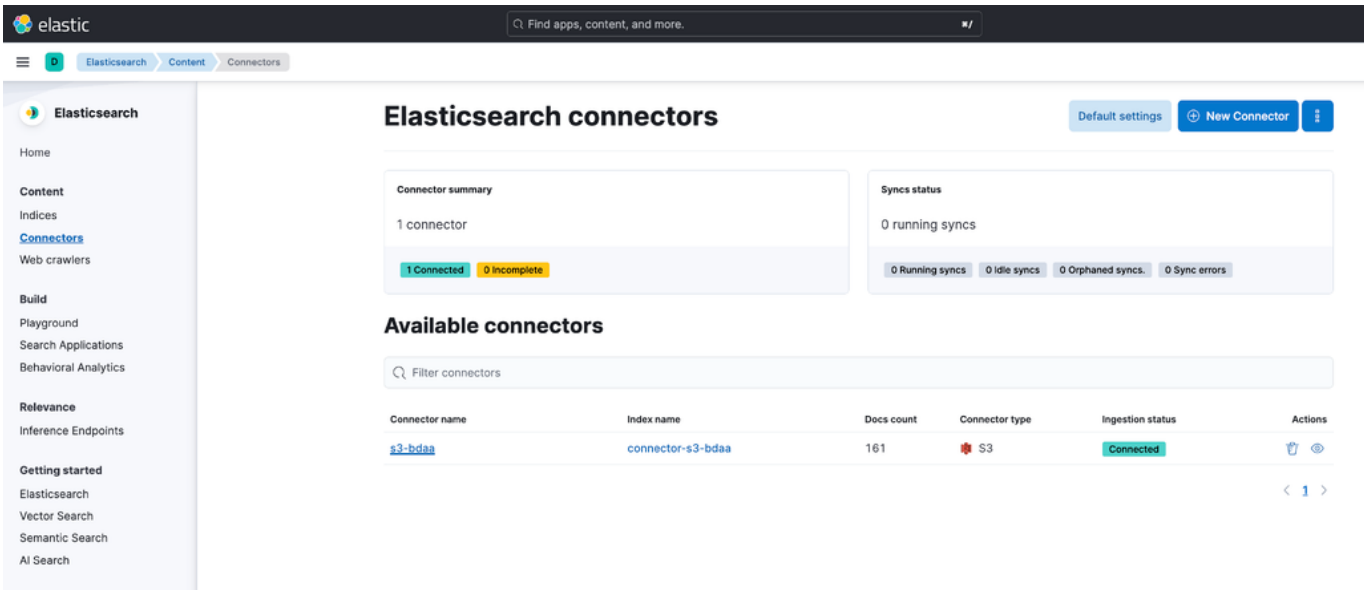
Docker instructions

Make sure you have Docker installed on your machine.

Figure: Kibana Connector Setup Step

```
- Copy this connector_id and will be used in the config.yaml file for
further configuration.
```

5. Start the Connector:
   - Once the setup is complete, start the connector from the AWS Console.
     - To set up and start the connector on the AWS Console, please follow the steps as outlined in the section **Steps for Setting Up the S3 Connector Instance** below.
   - The connector status should now appear as Connected in the AWS Console.



Elasticsearch connectors

Connector summary: 1 connector — 1 Connected, 0 Incomplete

Syncs status: 0 running syncs — 0 Running syncs, 0 Idle syncs, 0 Orphaned syncs, 0 Sync errors

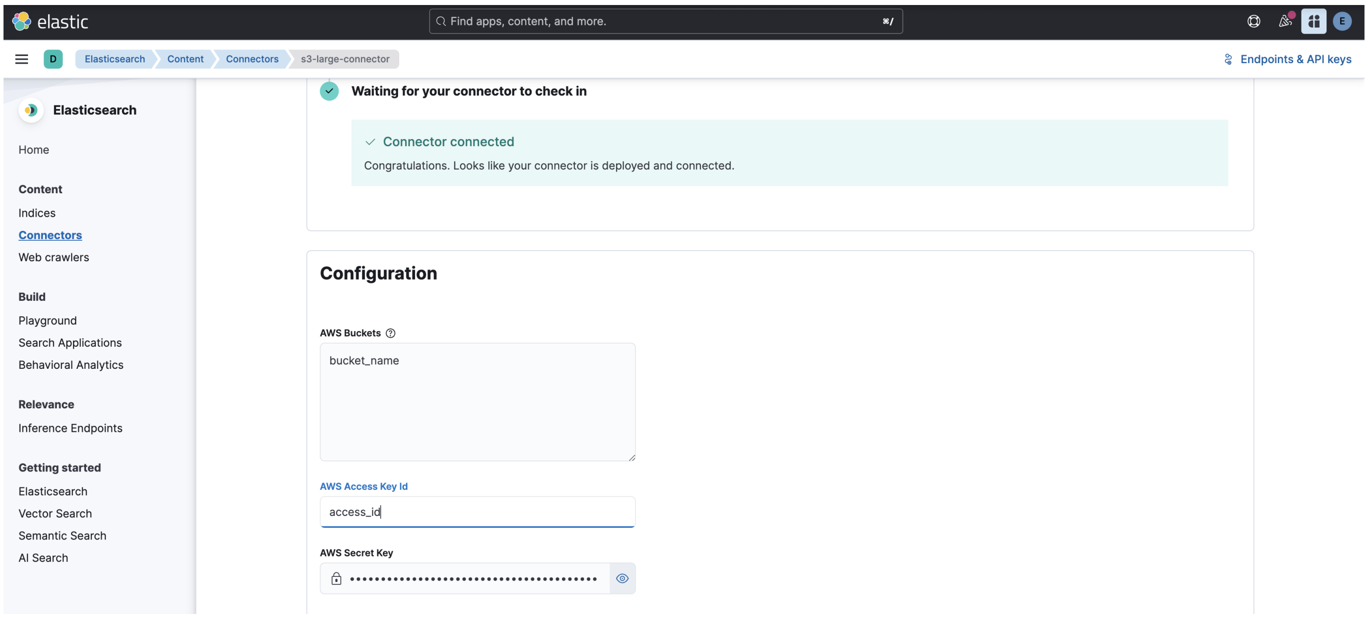
Available connectors

Connector name	Index name	Docs count	Connector type	Ingestion status	Actions
s3-bdaa	connector-s3-bdaa	161	S3	Connected	

Figure: Kibana Connector Setup Step

6. Configure the connector
   - Go to the S3 Connector configuration page.
   - Navigate to Configuration.
   - Click Edit Configuration.

   - Add the AWS S3 bucket name containing the input PDF files, along with the AWS Access Key and AWS Secret Key.
   - Save the configuration.



Waiting for your connector to check in

Connector connected

Congratulations. Looks like your connector is deployed and connected.

Configuration

AWS Buckets: bucket_name

AWS Access Key Id: access_id

AWS Secret Key: ••

Figure: Kibana Connector Setup Configuration

By following these steps, the S3 connector will be properly configured and ready for integration with the pipeline in wx-discovery Kibana.

Setting Up Elasticsearch Pipeline for Chunking and Indexing Documents

To set up the Elasticsearch pipeline for chunking and indexing the documents, follow these steps:

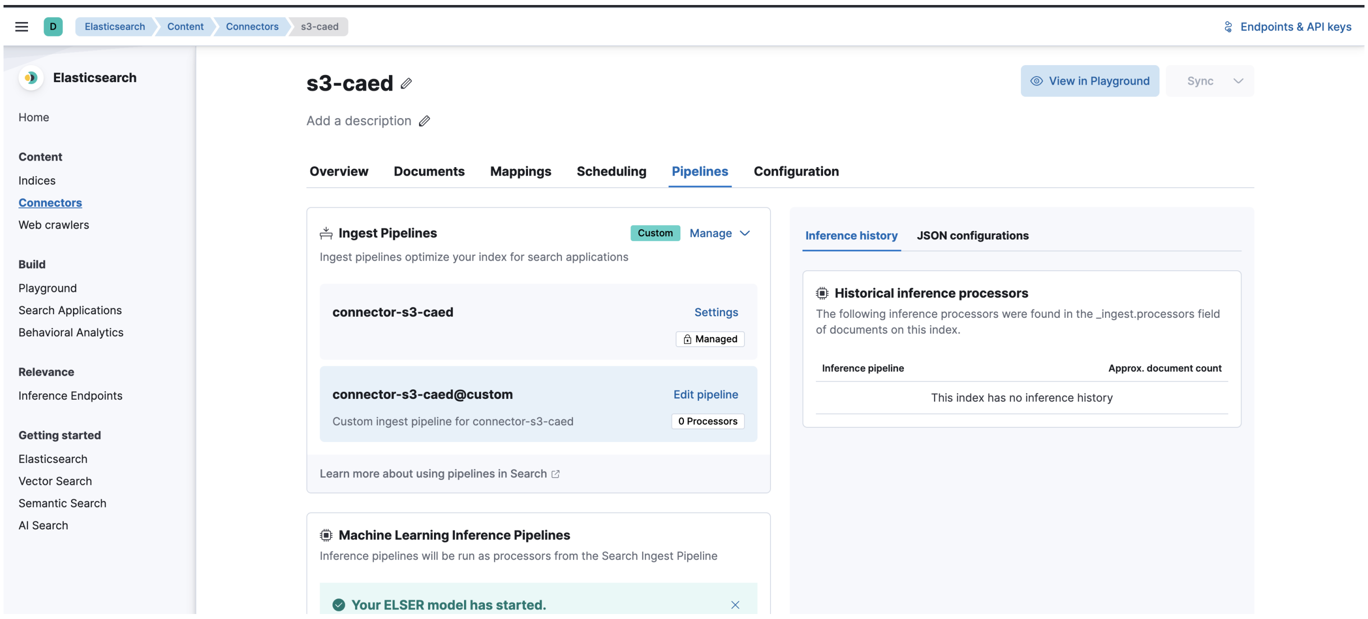
1. Create a New Elasticsearch Index:
   First, create a new Elasticsearch index with sharding enabled and an increased field limit. Optimal number of shards can be derived based on the input data size and cluster configuration. Following is the sample settings:

```
{
  "settings": {
    "number_of_shards": 7,
    "index": {
      "refresh_interval": "30s",
      "mapping.total_fields.limit": 5000000
    }
  }
}
```

2. Add the New Index to the S3 Connector:
   After creating the new index, add the shard_index to the S3 connector in Kibana. This will enable the connector to push the documents to the newly created index.

3. Add Elasticsearch Pipelines for Nested Chunking and Indexing:
   To ensure that the documents are chunked and indexed properly, add the following Elasticsearch

pipelines to the connector, based on the instructions from the Watson Developer Cloud Assistant Toolkit.


Elasticsearch > Content > Connectors > s3-caed
Endpoints & API keys

Elasticsearch
Home
Content
Indices
Connectors
Web crawlers
Build
Playground
Search Applications
Behavioral Analytics
Relevance
Inference Endpoints
Getting started
Elasticsearch
Vector Search
Semantic Search
AI Search

s3-caed
Add a description
View in Playground
Sync

Overview | Documents | Mappings | Scheduling | Pipelines | Configuration

Ingest Pipelines — Custom — Manage
Ingest pipelines optimize your index for search applications

connector-s3-caed — Settings — Managed

connector-s3-caed@custom — Edit pipeline
Custom ingest pipeline for connector-s3-caed — 0 Processors

Learn more about using pipelines in Search

Machine Learning Inference Pipelines
Inference pipelines will be run as processors from the Search Ingest Pipeline

Your ELSER model has started.

Inference history | JSON configurations

Historical inference processors
The following inference processors were found in the _ingest.processors field of documents on this index.

Inference pipeline	Approx. document count
This index has no inference history	

Figure: Elasticsearch pipeline

4. Update the Script Pipeline Processor:
   Update the Script Pipeline Processor with the following source code to perform the chunking of documents, handle overlapping passages and handle empty body content:

```
if(ctx['body'] != null)
{
  String[] envSplit = /((?<!M(r|s|rs)\.)(?<=\.)|(?<=\!)|(?<=\?))/
                  .split(ctx['body']);
  ctx['passages'] = [];

  StringBuilder overlappingText = new StringBuilder();

  int title_subs_idx = 200;
  if(ctx['body'].splitOnToken('_')[0].length() < 200)
    title_subs_idx = ctx['body'].splitOnToken('_')[0].length();
  int i = 0;
  while (i < envSplit.length) {
      StringBuilder passageText = new StringBuilder(envSplit[i]);
      int accumLength = envSplit[i].length();
      ArrayList accumLengths = [];
      accumLengths.add(accumLength);

      int j = i + 1;
      while (j < envSplit.length && passageText.length() +
envSplit[j].length() < params.model_limit) {
          passageText.append(' ').append(envSplit[j]);
          accumLength += envSplit[j].length();
          accumLengths.add(accumLength);
          j++;
      }
      ctx['passages'].add(['text': overlappingText.toString() + 
passageText.toString(), 'title': ctx['body'].splitOnToken('_')
[0].substring(0, title_subs_idx), 'url': ctx['filename']]);
      def startLength = passageText.length() * (1 - 
params.overlap_percentage) + 1;

      int k = Collections.binarySearch(accumLengths, 
(int)startLength);
      if (k < 0) {
          k = -k - 1;
      }
      overlappingText = new StringBuilder();
      for (int l = i + k; l < j; l++) {
          overlappingText.append(envSplit[l]).append(' ');
      }

      i = j;
  }
}
else{
  ctx['passages'] = [];
}
```

5. Enable Bulk Request Processing
   To optimize document indexing, enable bulk request processing in the pipeline. This allows Elasticsearch to process multiple document chunks in a single request, improving performance.
6. Pipeline Completion
   After the pipeline setup is complete, the S3 connector will chunk and index the documents into Elasticsearch as per the defined settings.

Steps for Setting Up the S3 Connector Instance

This guide outlines the process for setting up the S3 connector instance without Content Extraction.

Prerequisites:

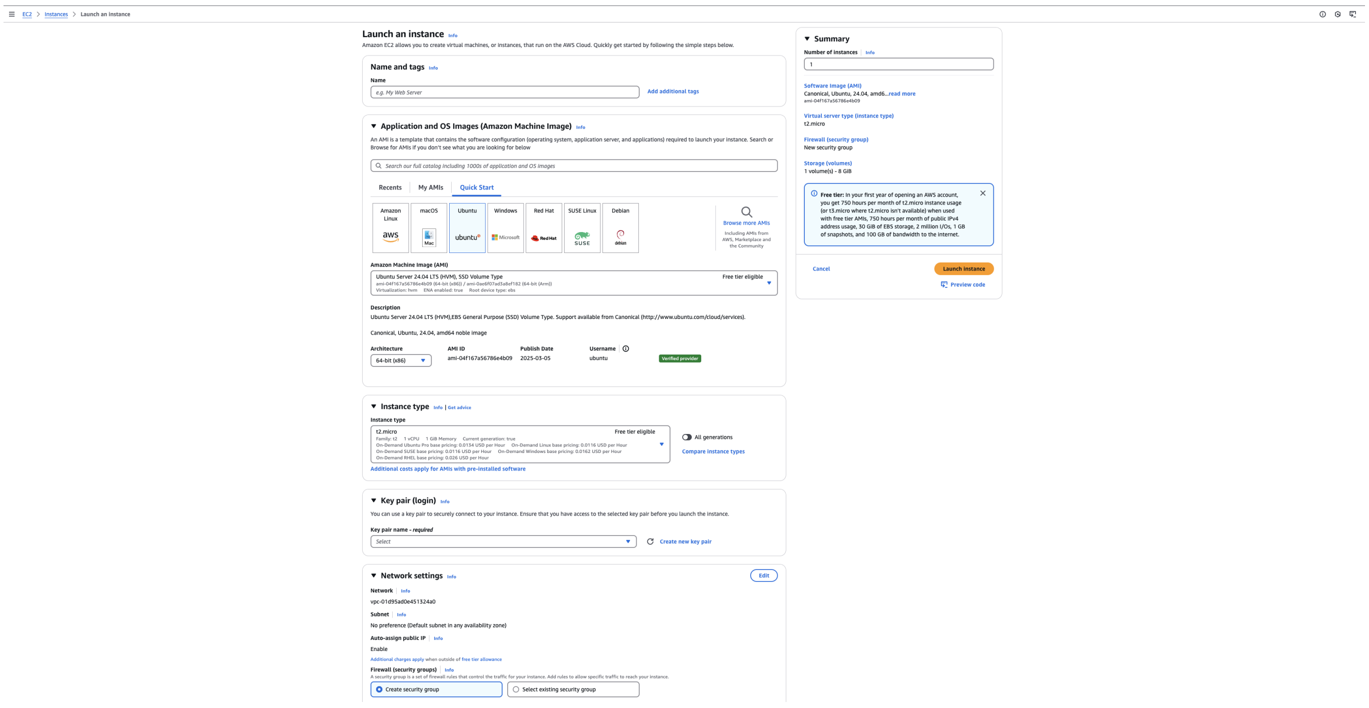
- AWS Console Access: You need access to the AWS EC2 console to launch an instance.
- Key Pair: Ensure you have the key pair for SSH access.
- Security Group: SSH access is required.

NOTE: All commands provided are based on Ubuntu OS. If using a different operating system, the commands may need to be updated accordingly.

Steps for Setup:

1. Launch an Instance on AWS EC2
   - Go to AWS EC2 service.
   - Launch a new instance based on your connector requirements (this will depend on CPU, memory, and the number of instances you need).
   - Select the Application and OS Image - Ubuntu

- Select the Instance Type (based on connector requirements).
- Choose the key pair.
- Select the appropriate security group (for SSH access).



Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags Info

Name: e.g. My Web Server — Add additional tags

Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Recents | My AMIs | Quick Start

Amazon Linux | macOS | Ubuntu | Windows | Red Hat | SUSE Linux | Debian

Browse more AMIs — Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Ubuntu Server 24.04 LTS (HVM), SSD Volume Type — Free tier eligible
ami-04f167a56786e4b09 (64-bit (x86)) / ami-0ae6f07ad3a8ef182 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Ubuntu Server 24.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).

Canonical, Ubuntu, 24.04, amd64 noble image

Architecture	AMI ID	Publish Date	Username	
64-bit (x86)	ami-04f167a56786e4b09	2025-03-05	ubuntu	Verified provider

Instance type Info | Get advice

Instance type

t2.micro — Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Ubuntu Pro base pricing: 0.0134 USD per Hour On-Demand Linux base pricing: 0.0116 USD per Hour
On-Demand SUSE base pricing: 0.0116 USD per Hour On-Demand Windows base pricing: 0.0162 USD per Hour
On-Demand RHEL base pricing: 0.026 USD per Hour

All generations — Compare instance types

Additional costs apply for AMIs with pre-installed software

Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*: Select — Create new key pair

Network settings Info — Edit

Network: vpc-01d95ad0e451324a0

Subnet: No preference (Default subnet in any availability zone)

Auto-assign public IP: Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups) Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

[✓ Create security group] Select existing security group

Summary

Number of instances: 1

Software Image (AMI): Canonical, Ubuntu, 24.04, amd6...read more
ami-04f167a56786e4b09

Virtual server type (instance type): t2.micro

Firewall (security group): New security group

Storage (volumes): 1 volume(s) - 8 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

Cancel — Launch instance — Preview code

Figure: AWS EC2 instance

2. Access the Instance

   - Once the instance is up and running, SSH into the instance.

3. Inside the Container: Install Prerequisites

   - Update the system and install required tools:

```
sudo apt install git
sudo apt-get install make
```

4. Clone the Connector Repository

   - Clone the connector repository from GitHub:

```
git clone https://github.com/elastic/connectors
cd connectors &amp;&amp; git checkout v8.17.0
```

5. Create the config.yml File

   - Inside the connectors directory, create the config.yml file:

```
touch config.yml
```

  - Add the following configuration details to the config.yml file:

```
connectors:
  - connector_id: ""
    service_type: "s3"
elasticsearch:
  host: ""
  username: ""
  password: ""
  verify_certs: true
  ca_certs: /home/ubuntu/connectors/cert.crt
```

6. Create the cert.crt File

  - You need to create a certificate file (cert.crt) as specified in the configuration: Ensure the certificate file is available at the path /home/ubuntu/connectors/cert.crt.

7. Install Python and Prerequisites

  - Before starting the connector, install the necessary prerequisites:

```
sudo apt update
sudo apt upgrade
sudo apt install python3.12-venv
```

8. Start the Connector

  - To start the connector, run the following commands:

```
make install
make run
```

9. Verify the Connector in Kibana (wx-discovery)

  - Go to Kibana under the wx-discovery environment.
  - The connector should now show as "connected".

Important Note: File Size Limitation for Above Connector

By default, the Above connector is designed to handle a maximum file size of 10MB. If you attempt to process a file larger than this size, you will encounter issues.

What to Do If Your File Exceeds the Limit

If your input file size exceeds the 10MB limit, you will need to make additional updates to the S3 connector code or configuration to accommodate larger files. This will ensure that your files are processed without errors.

Updating S3 Connector to Process Files Up to 100MB

Follow these steps to update the connector to handle files up to 100MB:

1. Update the DEFAULT_MAX_FILE_SIZE
   - Navigate to the connectors directory:

     ```
     cd connectors
     ```

   - Open the config.py file for editing:

     ```
     vi connectors/config.py
     ```

   - Update the DEFAULT_MAX_FILE_SIZE to 100MB (in bytes: 104857600):

     ```
     DEFAULT_MAX_FILE_SIZE = 104857600
     ```

2. Execute the Connector for Document Ingestion in the Elasticsearch Pipeline Run the following commands to install and start the connector after making the configuration changes:

   ```
   make install
   make run
   ```

Important Note:

For documents exceeding 100MB, it cannot be processed directly by the connector as HTTP has a max file size limit of 100MB. To process these larger files, you must set up and integrate a self-managed Content Extraction Service.

Setting Up Content Extraction for S3 Connector

To handle documents larger than 100MB, you must set up a self-managed Content Extraction Service.

Steps to Set Up the Content Extraction Service:

1. Install Docker
   Ensure that Docker is installed on the system running the S3 connector. If Docker is not installed, you can install it using the following command:

   ```
   sudo apt install docker.io
   ```

2. Start the Data Extraction Service in Docker
   Start the Data Extraction Service Docker container, which will run on port 8090. The container will allow you to extract content from documents larger than 100MB.

   Run the following command to start the extraction service:

   ```
   sudo docker run -d -p 8090:8090 -it -v /app/files:/app/files --name
   extraction-service docker.elastic.co/integrations/data-extraction-
   service:0.3.4
   ```

   This command pulls and runs the data-extraction-service Docker image, maps the necessary directories, and exposes port 8090 for the service.

3. Update the S3 Connector Configuration
   Now, update the S3 Connector configuration to integrate with the Data Extraction Service.

   - Navigate to the connectors directory:

     ```
     cd connectors
     ```

   - Open the config.py file for editing:

     ```
     vi connectors/config.py
     ```

   - Add the Content Extraction Service configuration block: Add the following configuration for the extraction_service to connect the S3 connector to the extraction service:

     ```
     # data-extraction-service settings
     extraction_service:
       host: http://localhost:8090
       timeout: 900
       use_file_pointers: true
       shared_volume_dir: '/app/files'
     ```

4. Enable Content Extraction in the S3 Connector
   Next, enable the content extraction feature in the S3 connector's configuration file.

   - Navigate to the sources/s3.py file in the connector directory:

     ```
     cd connector/sources
     vi s3.py
     ```

   - Update the default configuration to enable the text extraction service by setting use_text_extraction_service to True:

     ```
     use_text_extraction_service = True
     ```

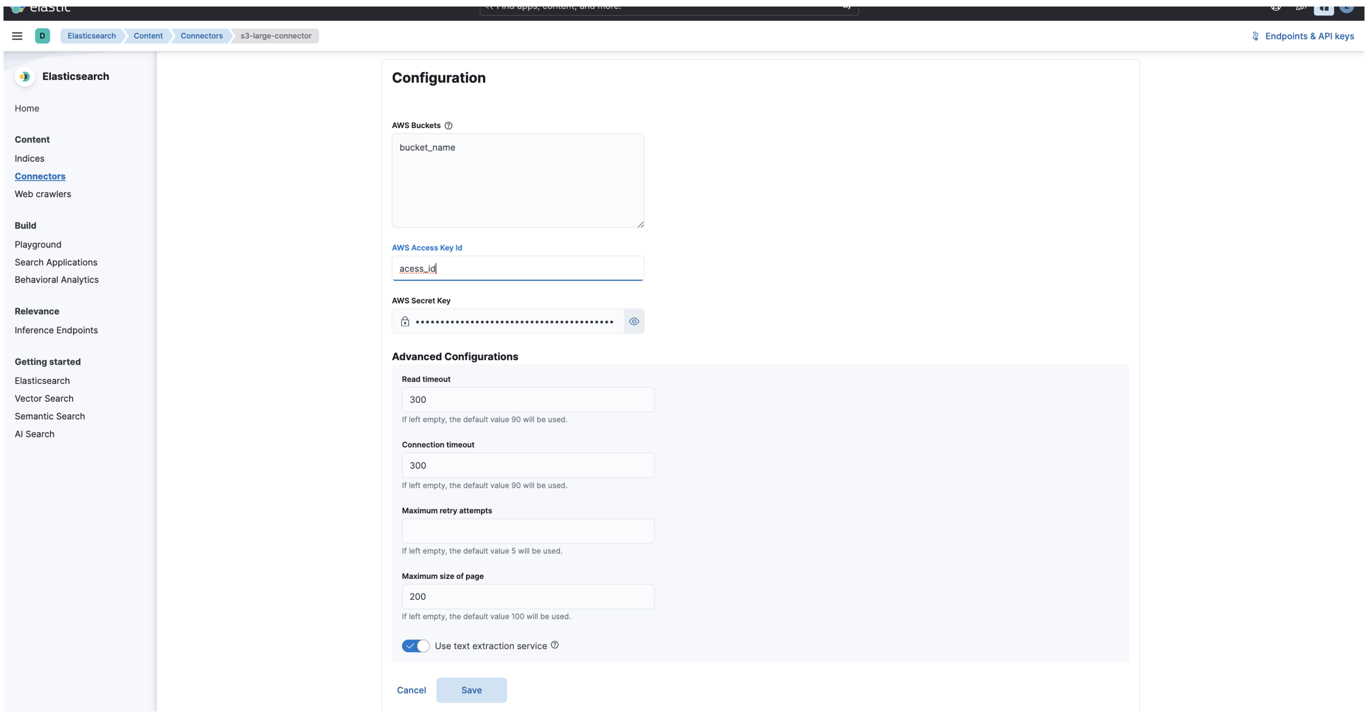
5. Execute the Connector for Document Ingestion in Elasticsearch Pipeline
   After making the necessary updates to the configuration, run the following commands to install and start the S3 connector for document ingestion:

   ```
   make install
   make run
   ```

Enabling Content Extraction in Kibana

After making the necessary configuration changes to the S3 connector, you'll need to enable content extraction in Kibana for the connector to properly process documents.

1. In Kibana UI
   - Go to the S3 Connector configuration page.
   - Navigate to Configuration.
   - Click Edit Configuration.
   - Enable the toggle for “Use Text Extraction Service”.
   - Save the configuration.



Configuration

AWS Buckets: bucket_name

AWS Access Key Id: acess_id

AWS Secret Key: ••

Advanced Configurations

Read timeout: 300
If left empty, the default value 90 will be used.

Connection timeout: 300
If left empty, the default value 90 will be used.

Maximum retry attempts:
If left empty, the default value 5 will be used.

Maximum size of page: 200
If left empty, the default value 100 will be used.

[✓] Use text extraction service

Figure: Kibana Connector Setup Configuration

2. Start the Connector in Kibana
   Once content extraction is enabled in Kibana, start the connector to begin processing documents with content extraction enabled.

License