AWS Machine Learning Blog

Discovering and indexing podcast episodes using Amazon Transcribe and Amazon Comprehend

by Angela Wang and Mike Gillespie | on 20 SEP 2018 | in Amazon Comprehend, Amazon Transcribe, Artificial Intelligence, AWS Step Functions | Permalink | Comments | Share

As an avid podcast listener, I had always wished for an easy way to glimpse at the transcript of an episode to decide whether I should add it to my playlist (not all episode abstracts are equally helpful!). Another challenge with podcasts is that, although they contain a wealth of knowledge that is often not available in blogs and other text formats, there's isn't a readily available search engine like Google to index and search the content. What if we could build a tool that converts the audio to text and then build a searchable index on all of our favorite podcast feeds so users could discover information that interests them, without having to listen to a full episode?

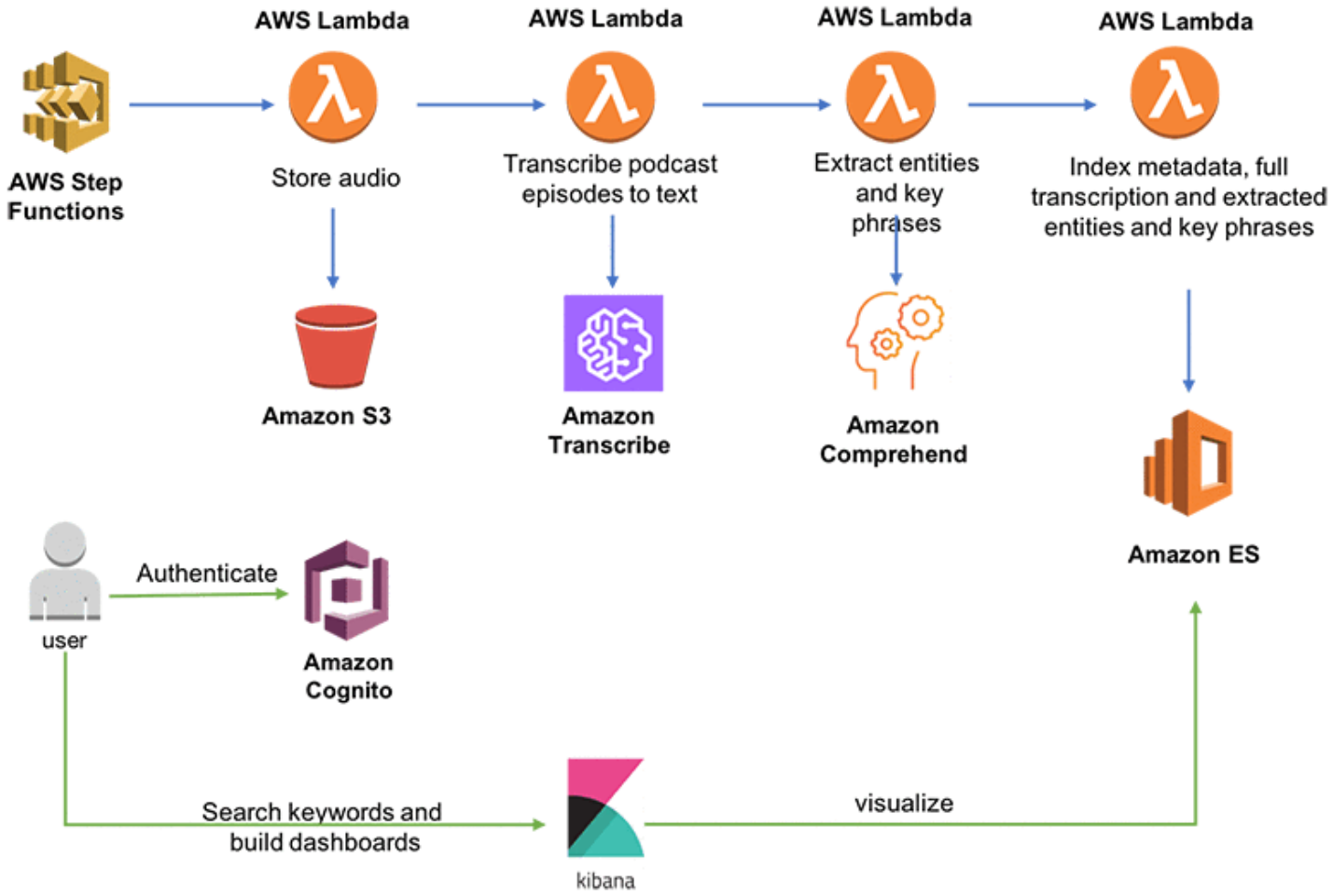
I was very excited when I heard about the launch of Amazon Transcribe and Amazon Comprehend. These machine learning (ML) application services provide pre-trained deep learning-based models as ready-to-use APIs. Amazon Transcribe performs automatic speech recognition(ASR) to transcribes audio into text. Amazon Comprehend provides a range of natural language processing (NLP) services such as entity extraction and sentiment analysis. With the help of these two services, you can now quickly put together an architecture that produces this searchable database of podcasts.

In this blog post, I'll walk you through how you can build a serverless podcast indexing application. When you do a keyword search in this application, it will show you which episodes your keyword appears in and where it appears in the full text transcript. It also will give you a link to the spot in the audio where the keyword was mentioned. In addition, it can analyze a specific podcast feed and visualize keywords and entities mentioned across episodes in that feed. For example, you can use it to compare the frequency of episodes that mention "containers" compared to "serverless". Furthermore, after an index is built, there are many things you can build on top of this data to further extend this application. For example, you may apply machine learning algorithms to build a podcast recommendation engine, so that the podcasts you like the most can be used to identify others that you might find interesting.

Note: This demo is designed to stay within the limits of the AWS Free Tier for AWS Lambda, AWS Step

Functions, Amazon Elasticsearch Service (Amazon ES), and Amazon Transcribe. As of this writing, if your account is not eligible for the Free Tier, running a single transcribe job and running the Elasticsearch cluster for about 2 hours will generate less than $2.00 in charges.

Here's a high-level diagram of the architecture:



A quick overview of the AWS services used in this application:

- **Amazon Transcribe**: Converts the audio files into a text transcription, including the timestamp of each word spoken. The demo also uses two cool features of Amazon Transcribe: speaker identification and custom vocabularies.
- **Amazon Comprehend**: Find insights and relationships in text using natural language processing (NLP). It can extract key phrases, places, people, brands, and events.
- **AWS Step Functions**: Coordinates the workflow of processing the podcasts and scheduling Lambda functions.
- **AWS Lambda**: Provides a serverless compute platform to handle the application logic, in this case written in Python.
- **Amazon Elasticsearch Service**: Provides a managed Elasticsearch cluster for searching the podcast transcripts.

- **Amazon Cognito**: Provides user authentication for the Kibana user interface to Amazon ES.
- **AWS CloudFormation**: Provides infrastructure as code in the AWS environment to simplify the deployment of the components.
- **Amazon S3**: Provides shared storage for audio and text files.

Now let's take a look at how to you can quickly deploy this application in your own AWS account to start indexing your favorite podcast feeds!

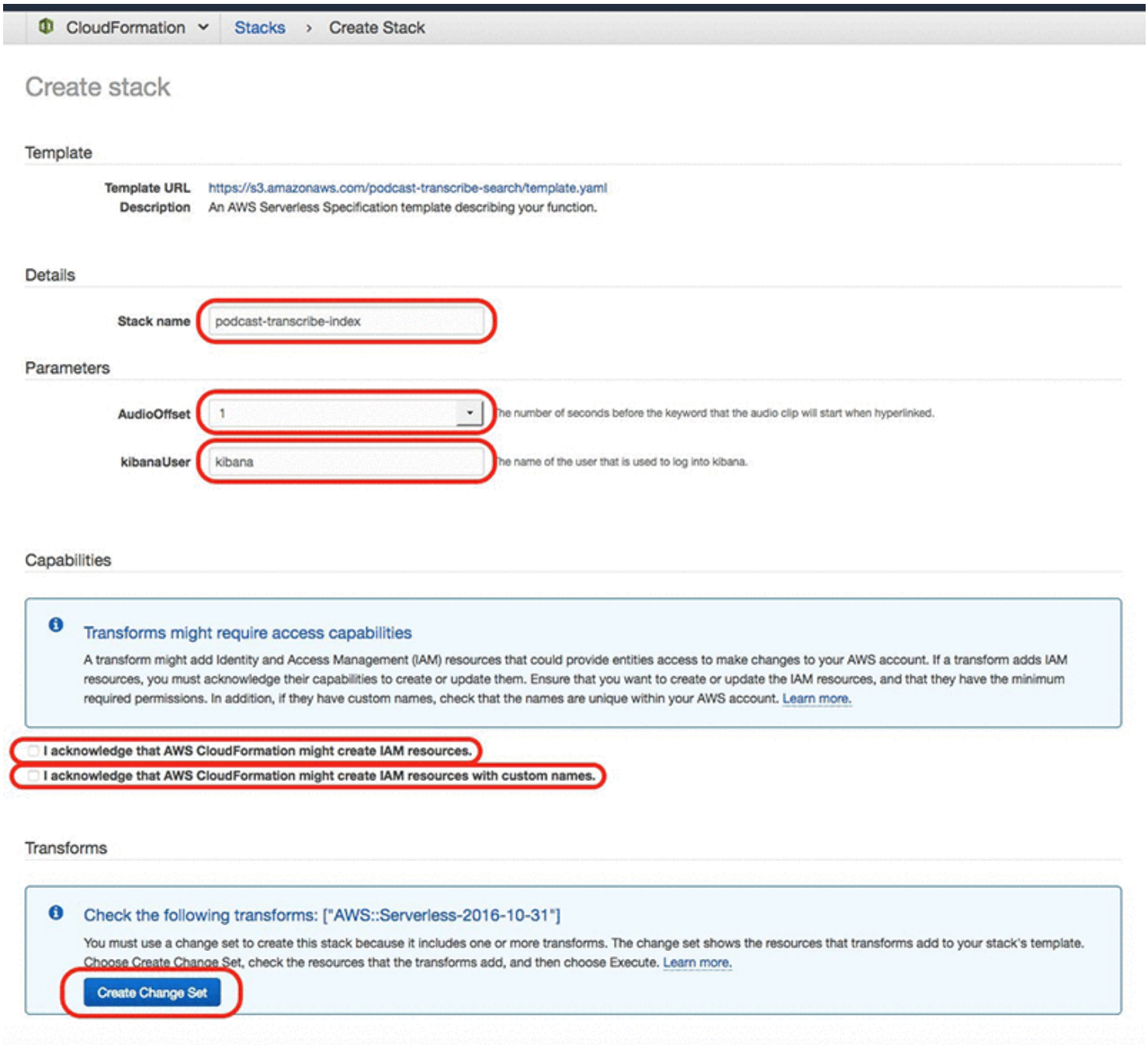
Launching the application with CloudFormation

To deploy this application in your AWS account, start by launching this CloudFormation stack in the CloudFormation console by using the following button:

In the CloudFormation console, on the **Create stack** page, complete the following:

- **Stack Name**: Provide a unique stack name for this account.
- **AudioOffset**: When you search for a keyword, the search result provides a link to the segment of audio file that mentions the keyword. This parameter controls the number of seconds to seek in the audio before the paragraph containing the keyword starts, to allow more context when hearing the playback. The default is 1 second.
- **kibanaUser**: The username of the user that is used to log into Kibana. Defaults to 'kibana'.
- Acknowledge the stack might create IAM resources by checking these boxes:
    - **I acknowledge that AWS CloudFormation might create IAM resources.**
    - **I acknowledge that AWS CloudFormation might create IAM resources with custom names.**

Choose **Create Change Set** to create the change set for this transform.



After the change set is generated, choose **Execute**.

It takes about 15-20 minutes to create the stack, mostly due to the time required to create and configure an Amazon ES cluster.

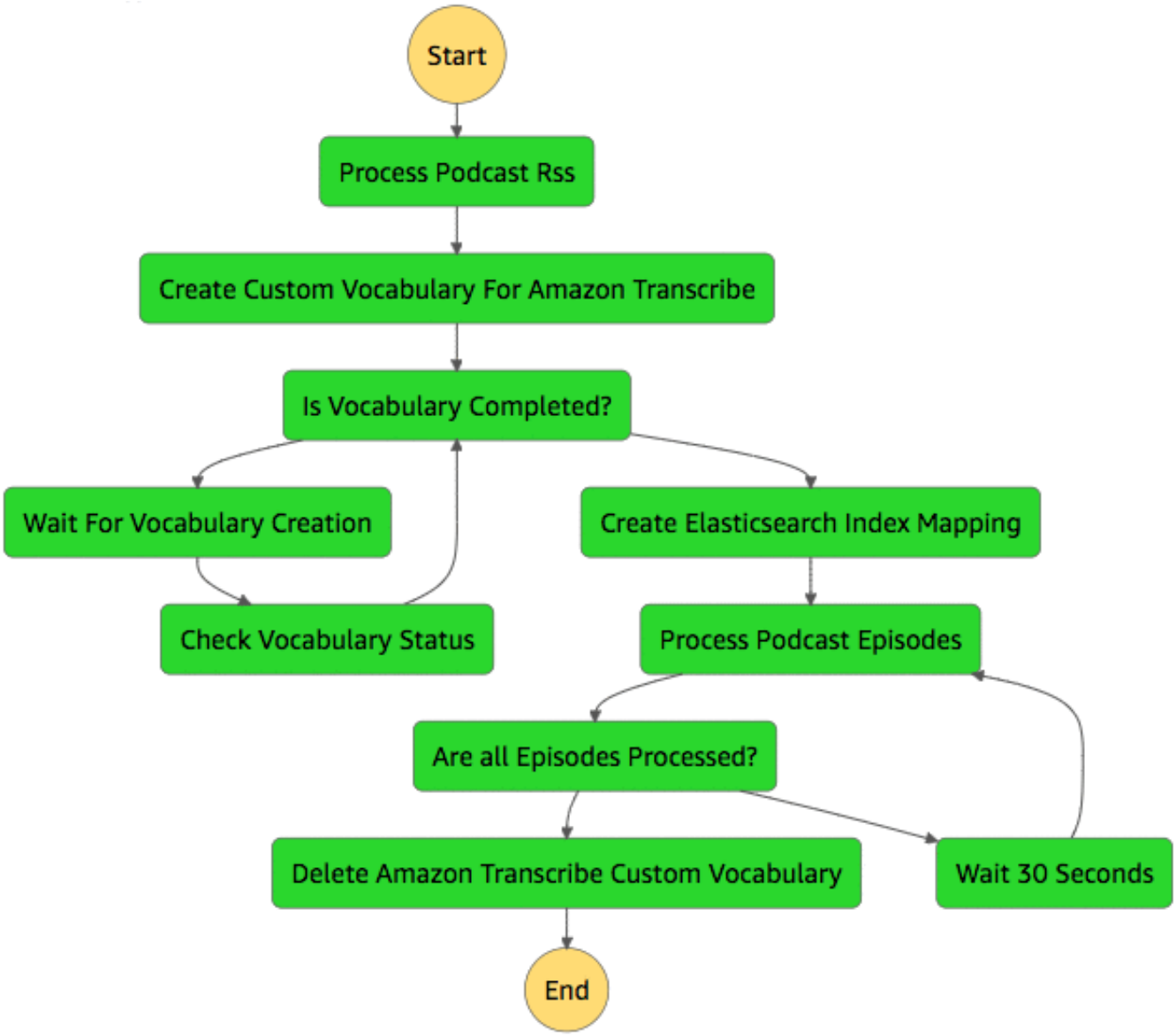
Application components walkthrough

While the AWS CloudFormation stack is being launched, let's step through the details of the components being built:

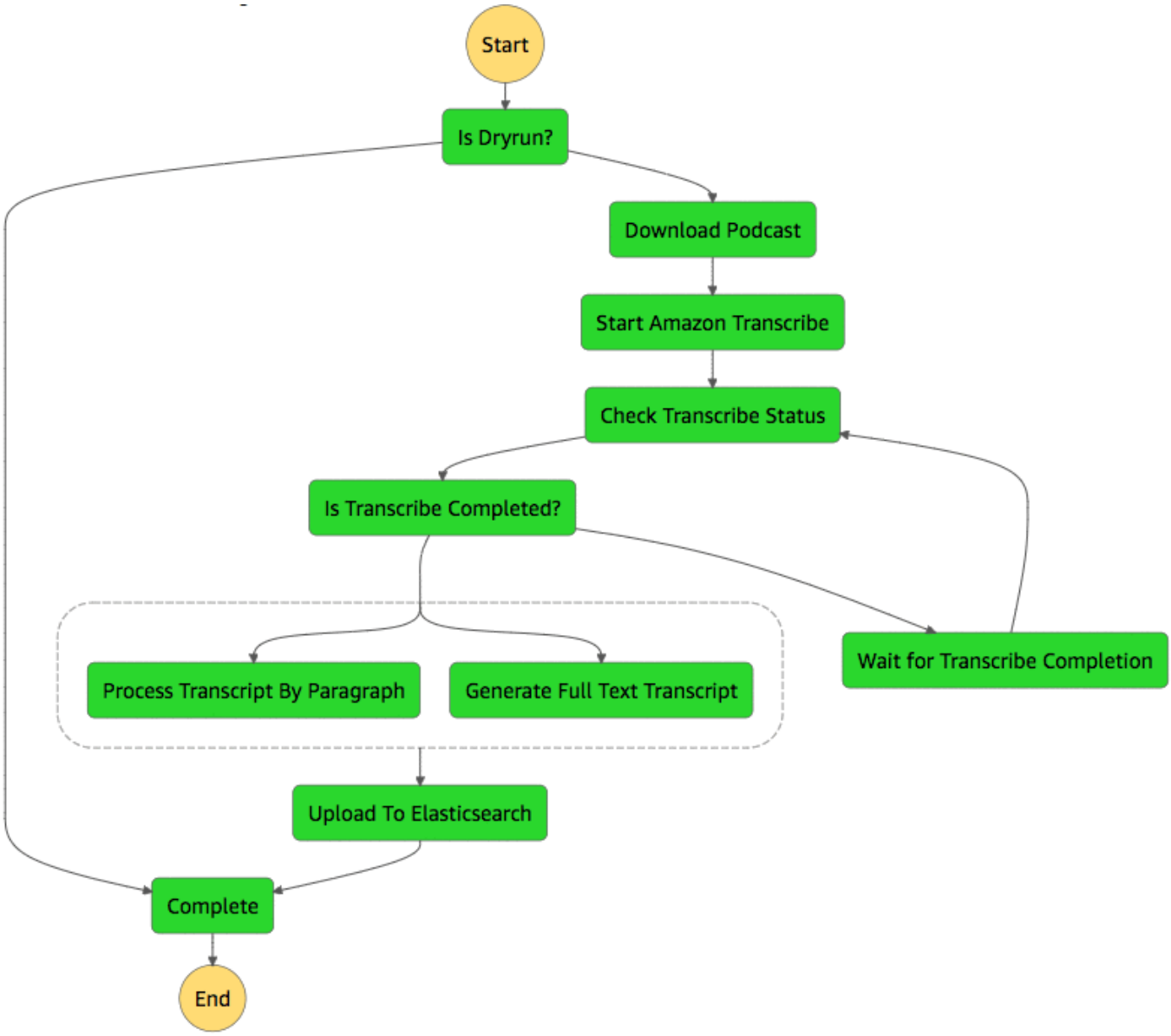
- **Amazon Elasticsearch Cluster**: This provides the persistence and visualization tools for our podcast

index. Visualization is performed by Kibana, which is included as part of Amazon ES.

- **RSS Feed Step Function State Machine**: This is a workflow that processes an RSS feed. It triggers Lambda functions that iterate through the episodes in the RSS feed and create a child state machine to process each episode.



- **Episode Step Functions State Machine**: The workflow that processes each individual podcast episode. The extracted transcription and metadata will be indexed into the Elasticsearch cluster.



Note that in this workflow we used a periodic polling model to wait for completion of the Amazon Transcribe job. This works well in the model where AWS Step Functions is keeping state and orchestrating the entire workflow. In an alternative architecture, you can leverage Amazon Transcribe's integration with Amazon CloudWatch Events and trigger subsequent processing from the "Transcribe job complete" CloudWatch Event in an event-driven model.

Each of the processes in these state machines is implemented using a Lambda function. We'll explore the implementation of these processes in more detail later.

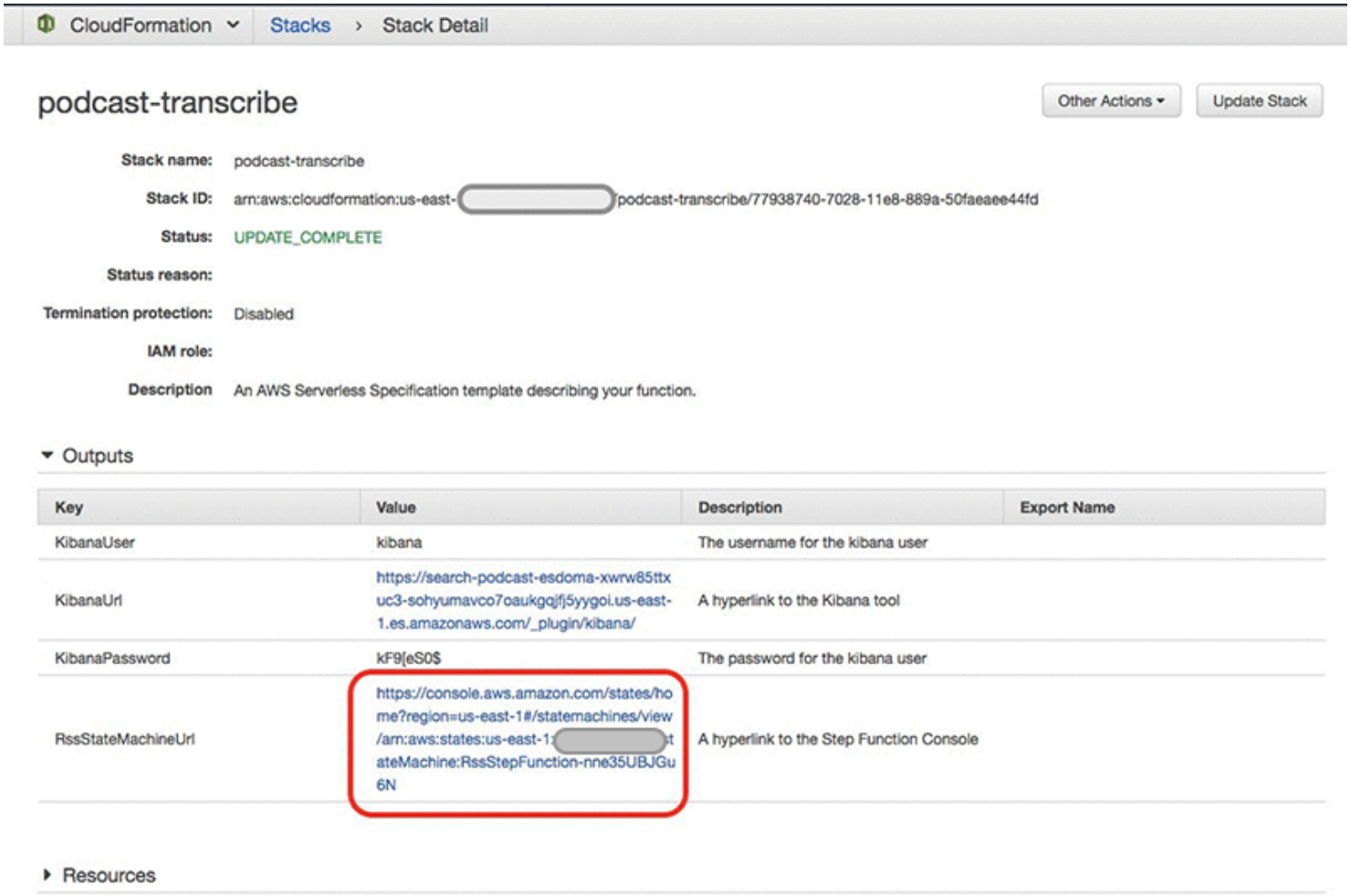
Start a podcast indexing workflow

Wait until the CloudFormation stack you created has a status of CREATE COMPLETE.

After the CloudFormation stack is created, the next step is to kick off a job to start indexing a podcast feed.

First, navigate to the **RSS Step Functions State Machine**. To do so, find the console URL by going to the output section of the CloudFormation stack. There are 4 outputs to the CloudFormation stack:

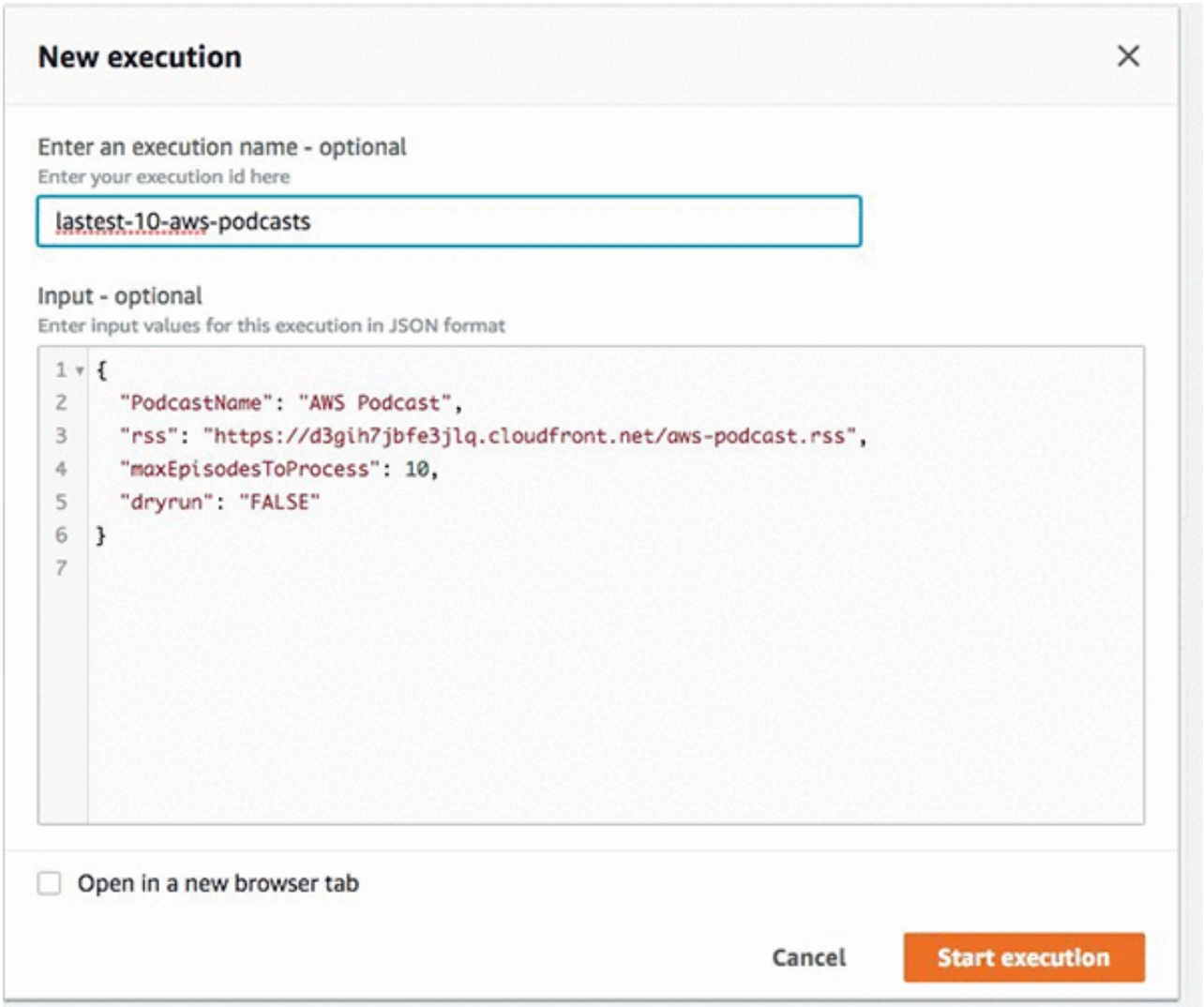
- **KibanaUser**: The username for the Kibana login.
- **KibanaPassword**: A random password for the first login into Kibana. You will be forced to change the password on the initial login.
- **KibanaUrl**: A hyperlink to the Kibana site
- **RssStateMachineUrl**: The URL to the Step Functions console that processes the RSS feed.



To go to the **RSS Step Functions State Machine**, choose the RssStateMachineUrl link in the CloudFormation Stack's Output.

Create an execution using the **Start execution** button and provide the input payload:

```
{
  "PodcastName": "AWS Podcast",
  "rss": "https://d3gih7jbfe3jlq.cloudfront.net/aws-podcast.rss",
  "maxEpisodesToProcess": 10,
  "dryrun": "FALSE"
}
```



This input will process the latest 10 episodes of the AWS Podcast.

Note: You can choose a different podcast feed by changing the `rss` input parameter. The `maxEpisodesToProcess` input parameter lets you control the number of episodes to process from this feed. This helps keep down the cost of this demo. For reference, Amazon Transcribe costs $0.0004/second, which comes to $0.36 for a 15 minute audio. The `dryrun` flag will test the state machine without calling the AI functions. Leave is to FALSE to fully process the podcast.

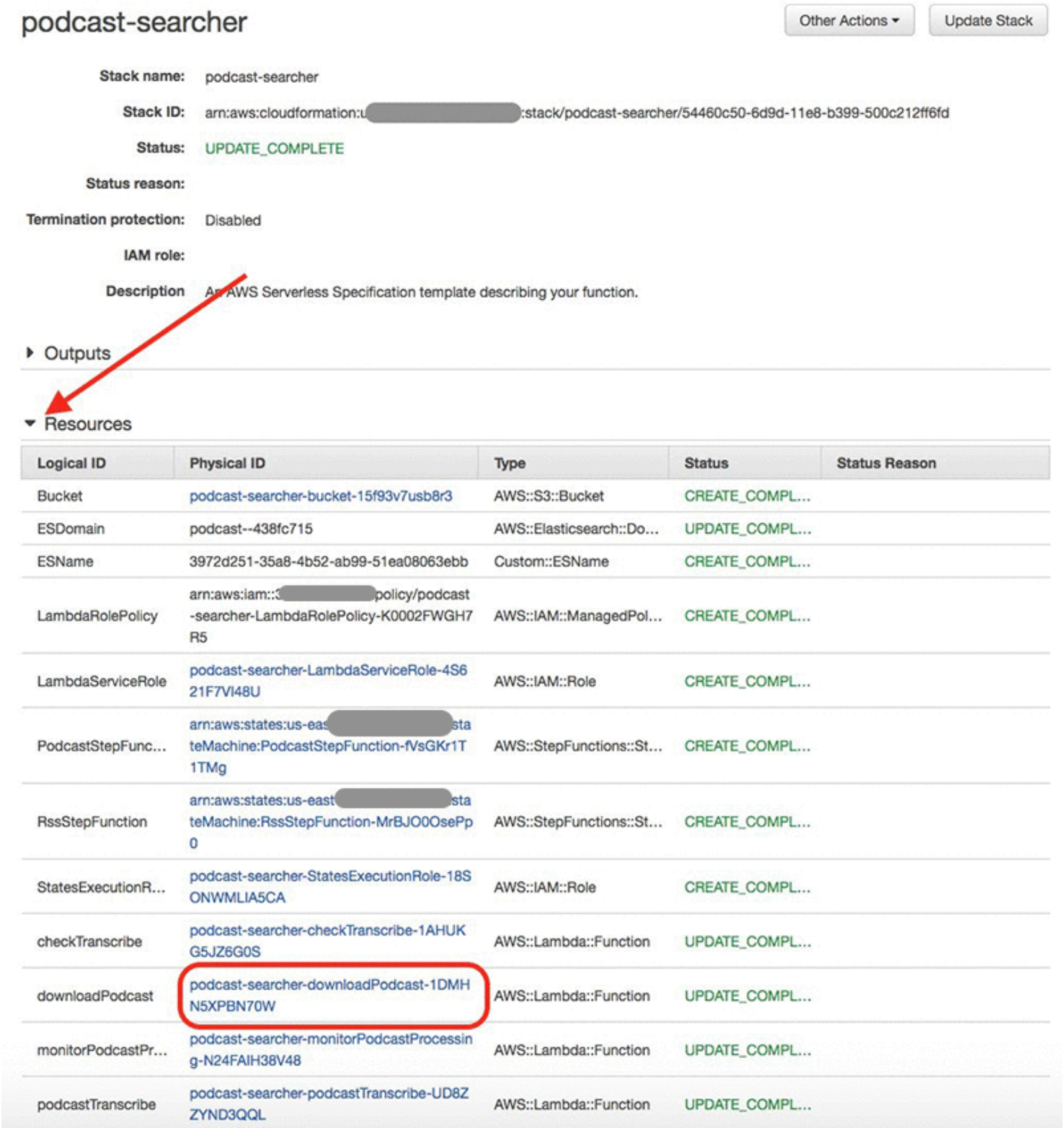
Amazon Transcribe can take about 10-15 minutes to process the 10 episodes. Although you can request to increase the number of concurrent jobs, Amazon Transcribe limits each account/AWS Region to 10 concurrent jobs by default (see Amazon Transcribe Service Limits documentation). This means that additional jobs needs to be queued and wait for earlier jobs to finish before they can be run. In our demo application, we control the rate of job submission in the Rss Step Functions State Machine's "Process Podcast Episodes" step and also leverage Exponential Backoff on the "Start Transcribe" step in the Episode Step Functions State Machine.

While this process is running, let's look at the code.

Review code of AWS Lambda functions

You can review the source code at https://github.com/aws-samples/amazon-transcribe-comprehend-podcast or in the AWS Lambda console following these instructions:

1. In the AWS CloudFormation console, expand the **Resources** section and find the Lambda function you want to review. Choose the hyperlink for the details page for the function.
2. In the AWS Lambda console, scroll to the **Function code** section to review the code.



Repeat the process for the following Lambda functions that process the **RSS Feed Step Function State Machine**:

- **processPodcastRss**: Downloads the RSS file and parses it to determine the episodes to download. This function also leverages the Amazon Comprehend **entity extraction feature** for two use cases:

    - To compute an estimate of the number of speakers in each episode. We do this by using Amazon Comprehend to find people's names in each episode's abstract. We find that many podcast hosts like to mention their guest speakers' names in the abstract. This helps us later when we use Amazon Transcribe to break out the transcription into multiple speakers. If no names are found in the abstract, we will assume the episode has a single speaker.
    - To build a domain-specific custom vocabulary list. If a podcast is about AWS, you will hear lots of expressions unique to the specific domain (e.g., EC2, S3) that are completely different from expressions found in a podcast about astronomy (e.g., Milky Way, Hubble). Providing a custom vocabulary list to Amazon Transcribe can help guide the service in identifying an audio segment that sounds like "easy too" to its actual meaning "EC2." In this blog post, we automatically generate the custom vocabulary list by using the named entities extracted from episode abstracts to make Amazon Transcribe more domain aware. Keep in mind that this approach may not cover all jargon that could appear in the transcripts. To get more accurate transcriptions, you can complement this approach by drafting a list of common domain-specific terms so that you can construct a custom vocabulary list for Amazon Transcribe.
- **createTranscribeVocabulary**: Creates a custom vocabulary for the Amazon Transcribe jobs so it will better understand when an AWS term is mentioned. The custom vocabulary is created using the method mentioned earlier.
- **monitorTranscribeVocabulary**: Polls Amazon Transcribe to determine if the custom vocabulary creation is complete.
- **createElasticsearchIndex:** Creates index mappings in Elasticsearch.
- **processPodcastItem**: Creates a child state machine execution for each episode while maintaining a maximum of 10 concurrent child processes. This function keeps track of how many processes are active and throttles the downstream calls once the maximum is hit. Amazon S3 is used to store additional state about each episode.
- **deleteTranscribeVocabulary**: Cleans up the custom vocabulary after the processing of all episodes is complete. Note that we added this step to minimize artifacts that stay around in your account after you run the demo application. However, when you build your own apps with Amazon Transcribe, you should consider keeping the custom vocabulary around for future processing jobs.

Review the Lambda functions that make up the **Episode Step Functions State Machine**.

- **downloadPodcast**: Downloads the podcast from the publisher and stages it in Amazon S3 for further processing.
- **podcastTranscribe**: Makes the call to Amazon Transcribe to create the transcription job. Notice how we pass in parameters extracted from previous steps, such as the custom vocabulary to use and number of speakers for the episode.
- **checkTranscript**: Polls the transcription job for status. Returns the status, and the step function will retry if the job is in progress.
- **processTranscriptionParagraph**: This is the most complicated function in the application. It downloads the transcription data from Amazon Transcribe and break it into paragraphs. The paragraphs are broken by speaker, punctuation, or a maximum length. The output of this function is a

file that contains all of the paragraphs in the transcription job. It also includes the start time for when the phrases were spoken in the audio file, and it includes the speaker that the paragraph is attributed to.
- **processTranscriptionFullText**: This function contains logic that is similar to **processTranscriptionParagraph**, but the output is a full text transcription in a readable format.
- **UploadToElasticsearch**: Parses the output of the previous steps and performs a bulk load of the indexes into the Elasticsearch cluster. The connection to Elasticsearch uses a SigV4 signature to perform IAM-based authentication into the cluster.

If the state machine execution you created several steps ago is not yet complete, wait a few minutes and recheck the status. Note that you will be able to see results appear in the Elasticsearch index as soon as some executions of the child workflow **EpisodeStateMachine** is complete, even while the parent **RssStateMachine** is still waiting for the rest of the episodes to finish**.**

Search and visualize podcasts with Kibana

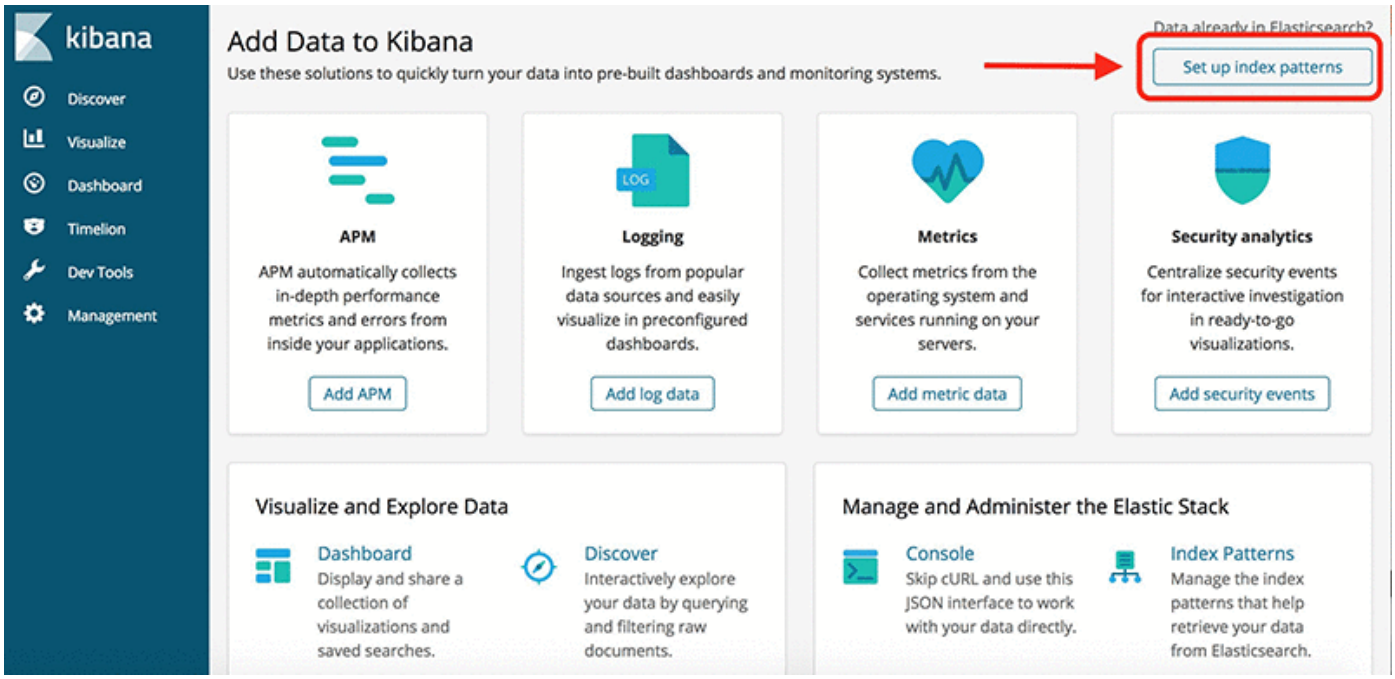
This application creates two Elasticsearch indexes:

- **Episodes index**: Each document in this index is a single episode. You can search and review attributes such as the episode's title, readable transcript of the full episode, publish date, entities extracted from the transcript, etc.
- **Paragraphs index**: Each document in this index is a single paragraph of an episode. You can search for keywords at a paragraph level, review the episode the paragraph appears in, and find the URL that directly links to the position of the paragraph in the audio.

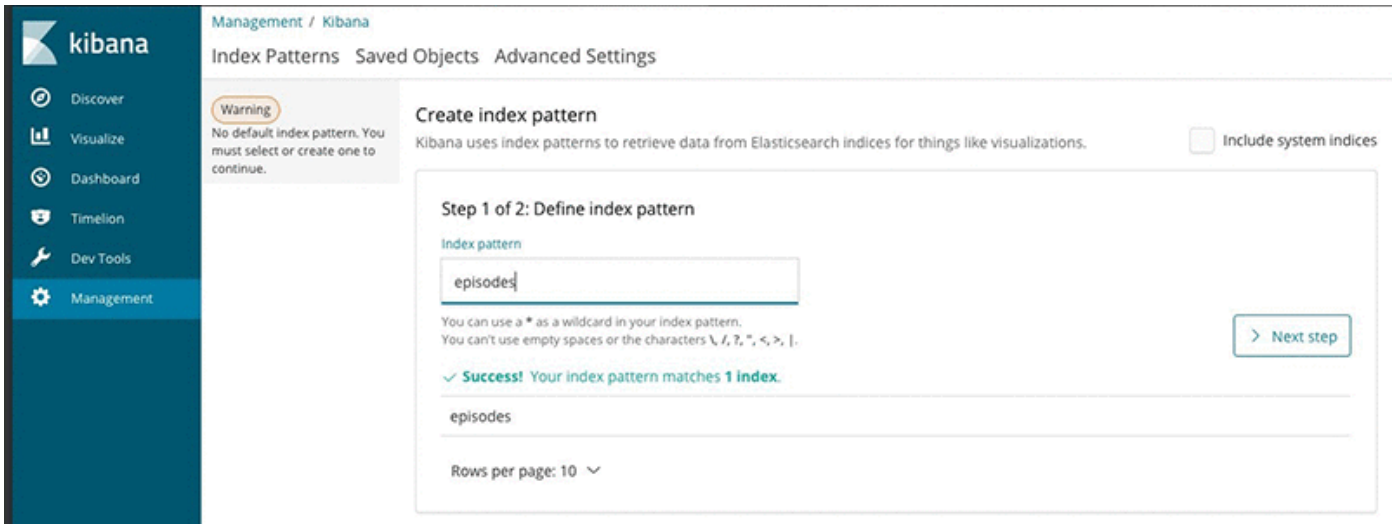
To configure Kibana to search and visualize these indexes, follow these steps:

Setting up the episode index

1. Navigate to the URL for Kibana. (You can find the KibanaUrl in the **Output** section of the CloudFormation stack.)
2. To log in to Kibana, use the username and password from the CloudFormation output. You will be required to enter a new password on the initial login.
3. After Kibana opens, choose **Set up index patterns**



4. In the index pattern textbox, type **episodes**, then choose **Next Step**:



5. In the second step, if you are asked to pick a Time Filter Field, choose “I don’t want to use the Time Filter”, then choose **Create Index Pattern**.

Searching the episode index and read the transcript

Click the **Discover** entry on the left-hand toolbar.

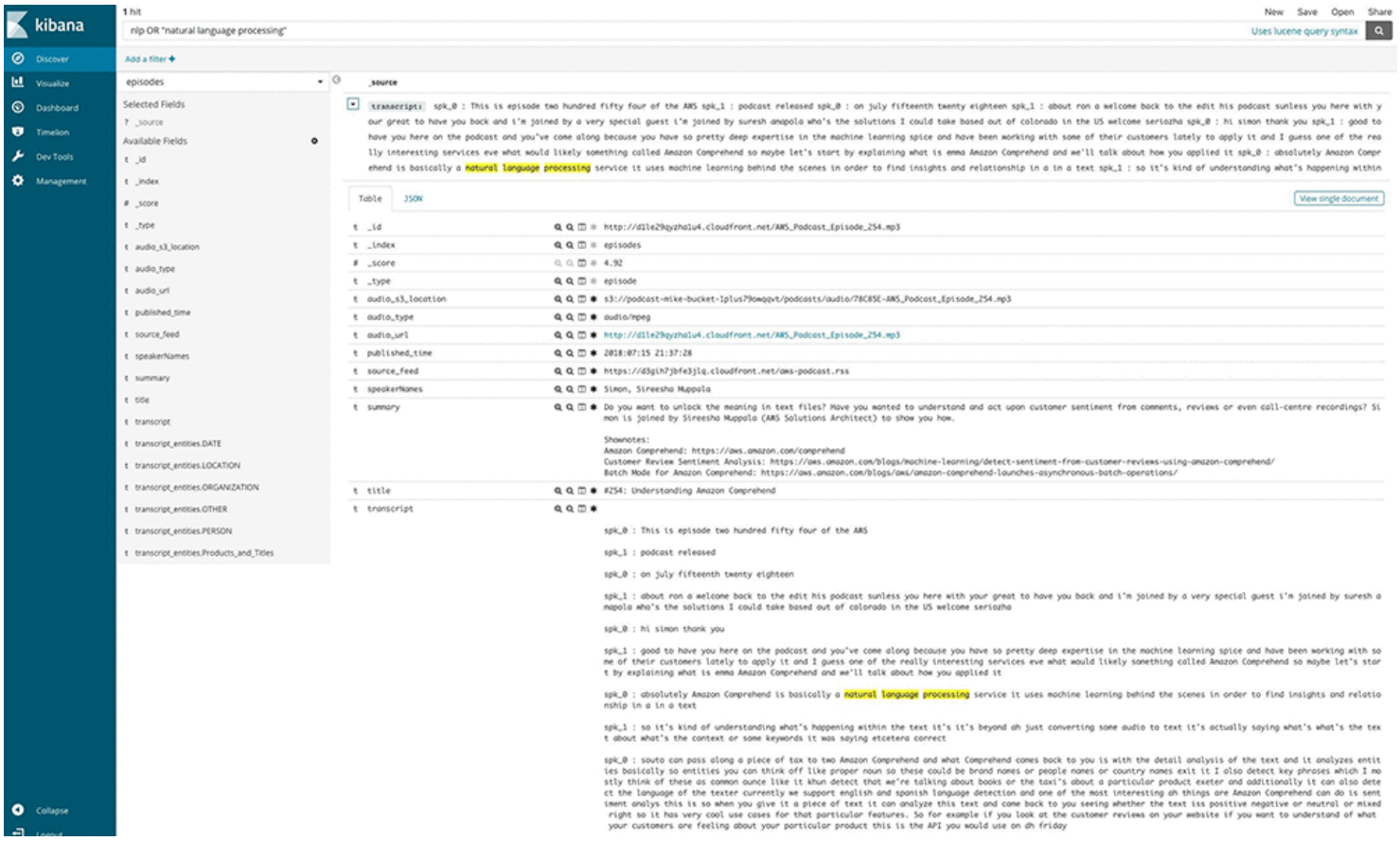
Enter search terms you are interested in, and you can explore the episodes that mention your terms and read the transcript of the episode!

To take advantage of the query capabilities of Kibana, refer to the documentation here for more supported syntaxes. See the following chart for some example queries:

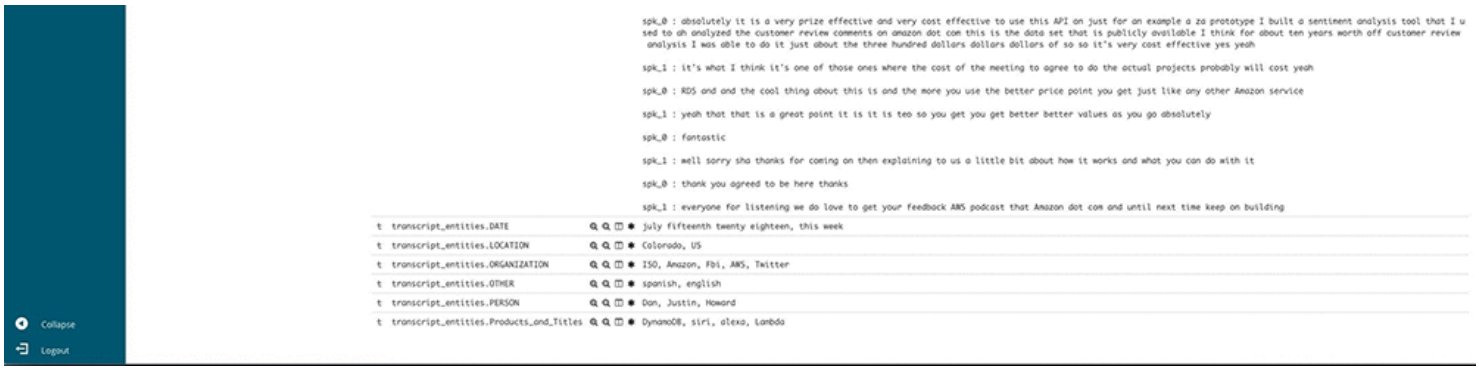
	Search Criteria	Notes
1	natural language processing	Any episode with either “natural”, “language” or “processing”
2	“natural language processing”	Any episode with phrase “natural language processing” in the title or text
3	nlp OR “natural language processing”	Any episode with either “nlp” or the “natural language processing” phrase in the title or text
4	+text:”natural language processing” -Episode:”240: AWS Answers”	Any episode with the phrase “natural language processing” except Episode 240

In the example that follows, I used the query `nlp OR "natural language processing"` to search for episodes that mention either the word “nlp” or the phrase “natural language processing”.

Use the ▼ to expand the full document to look through the transcript. If you like it, find the link to the episode from the `audio_url` field to listen to the full show!



If you scroll pass the transcript, you will also see some of the named entities extracted from the transcript. We can later use these keywords to build some visualization/tag clouds:

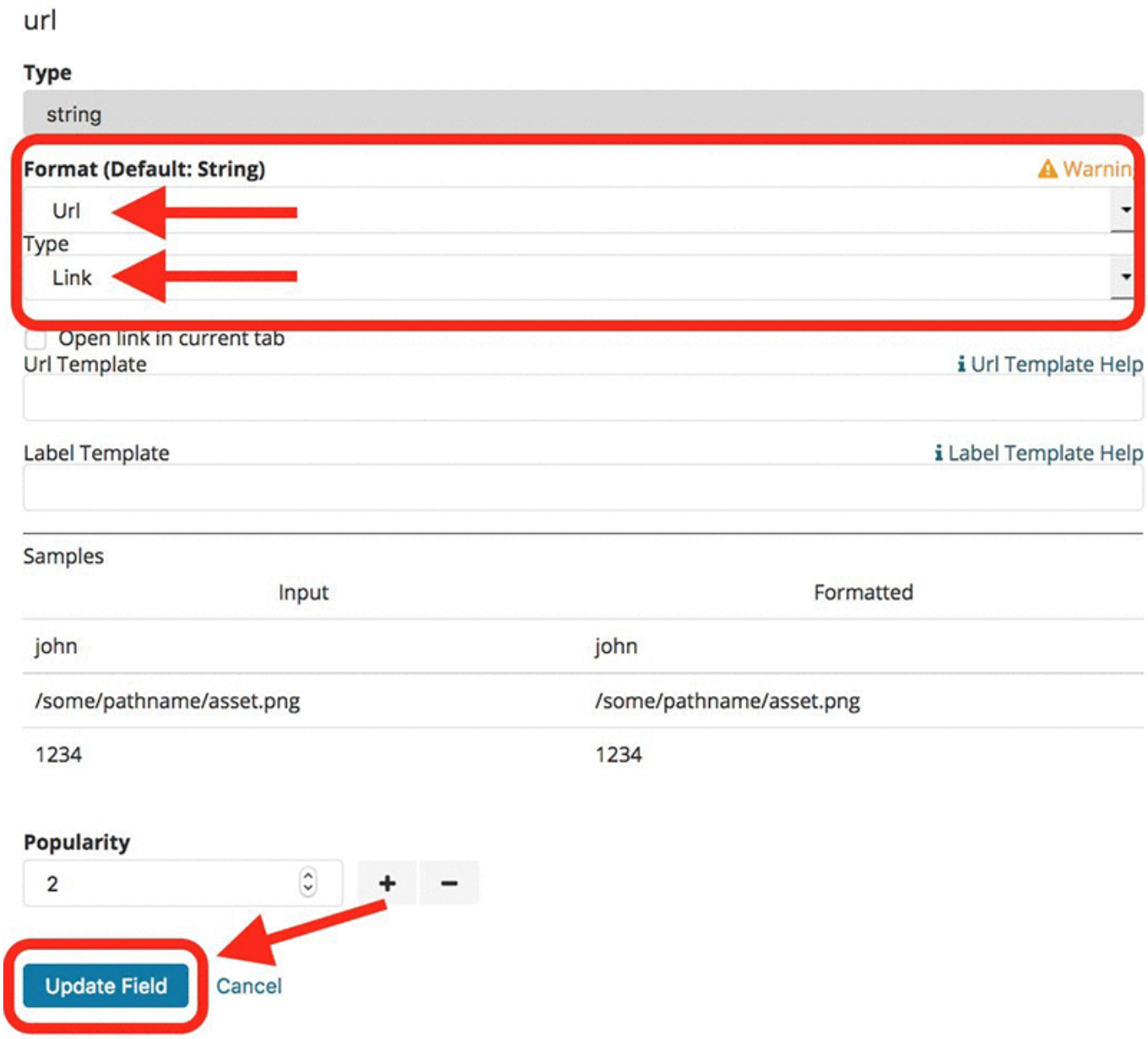


Setting up the paragraph index

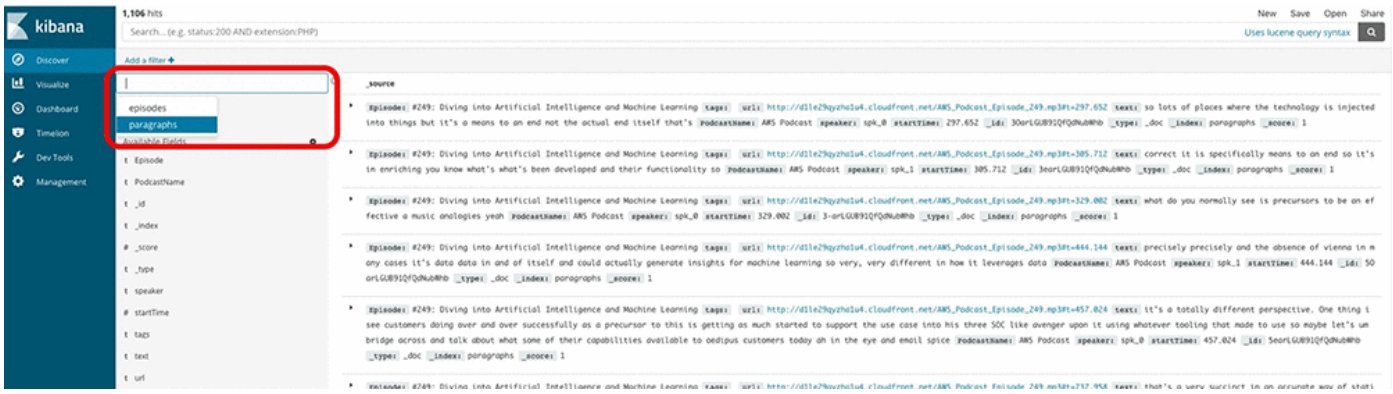
Now let's set up the second index we built. This index lets you search at a paragraph level where a keyword appears in, and links you to the exact spot in the audio where it was discussed.

1. Navigate to **Management**→ **Index Patterns** → **Create Index**.
2. In the index pattern textbox, type **paragraphs**, then choose **Next Step**.

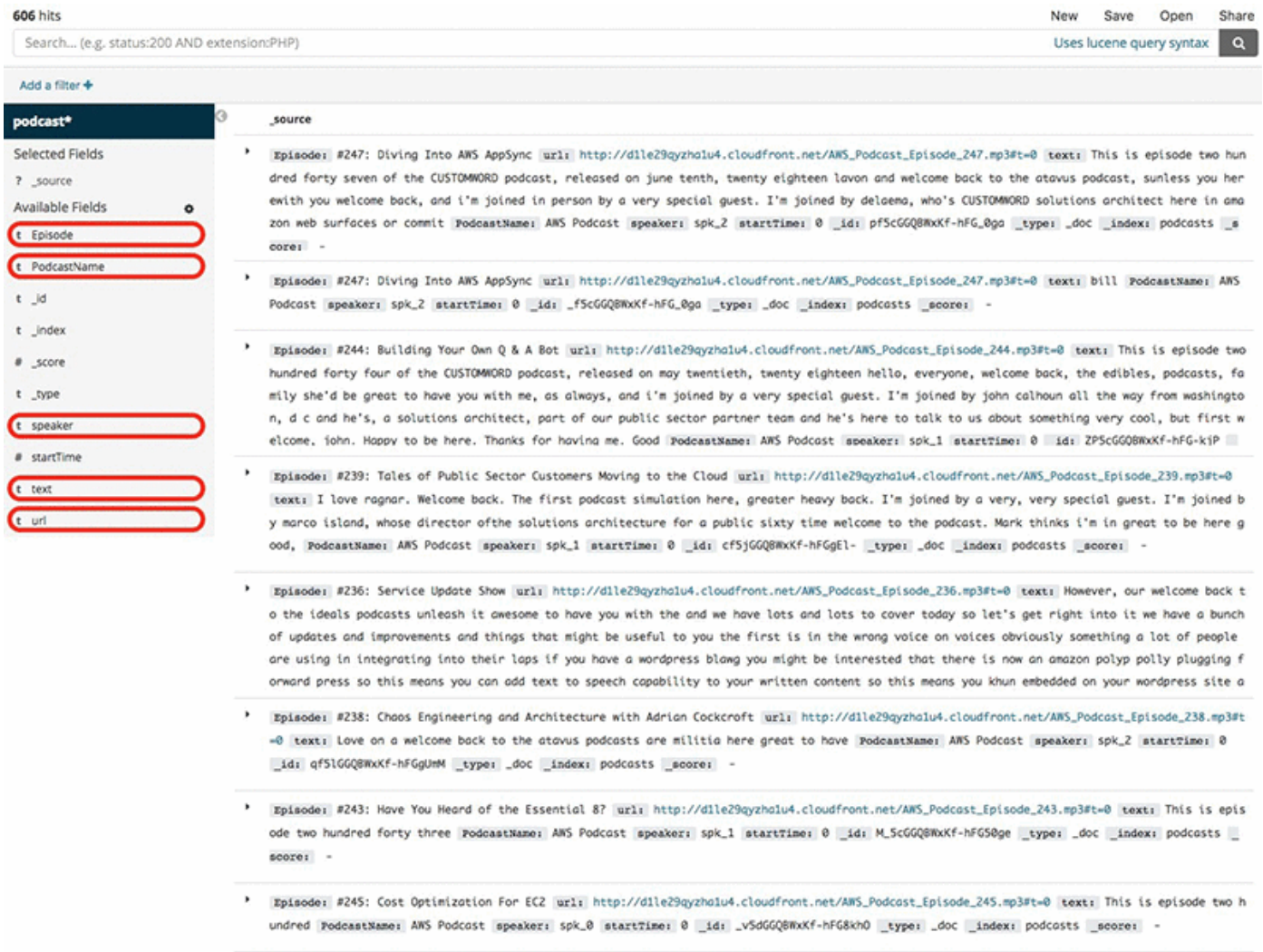
3. Accept the defaults and choose **Create Index Pattern**.
4. Scroll down to the **Url** field and choose the edit icon on the right.
5. Set the **Format** to **Url**, and the type to **Link**. Then choose **Update Field**.



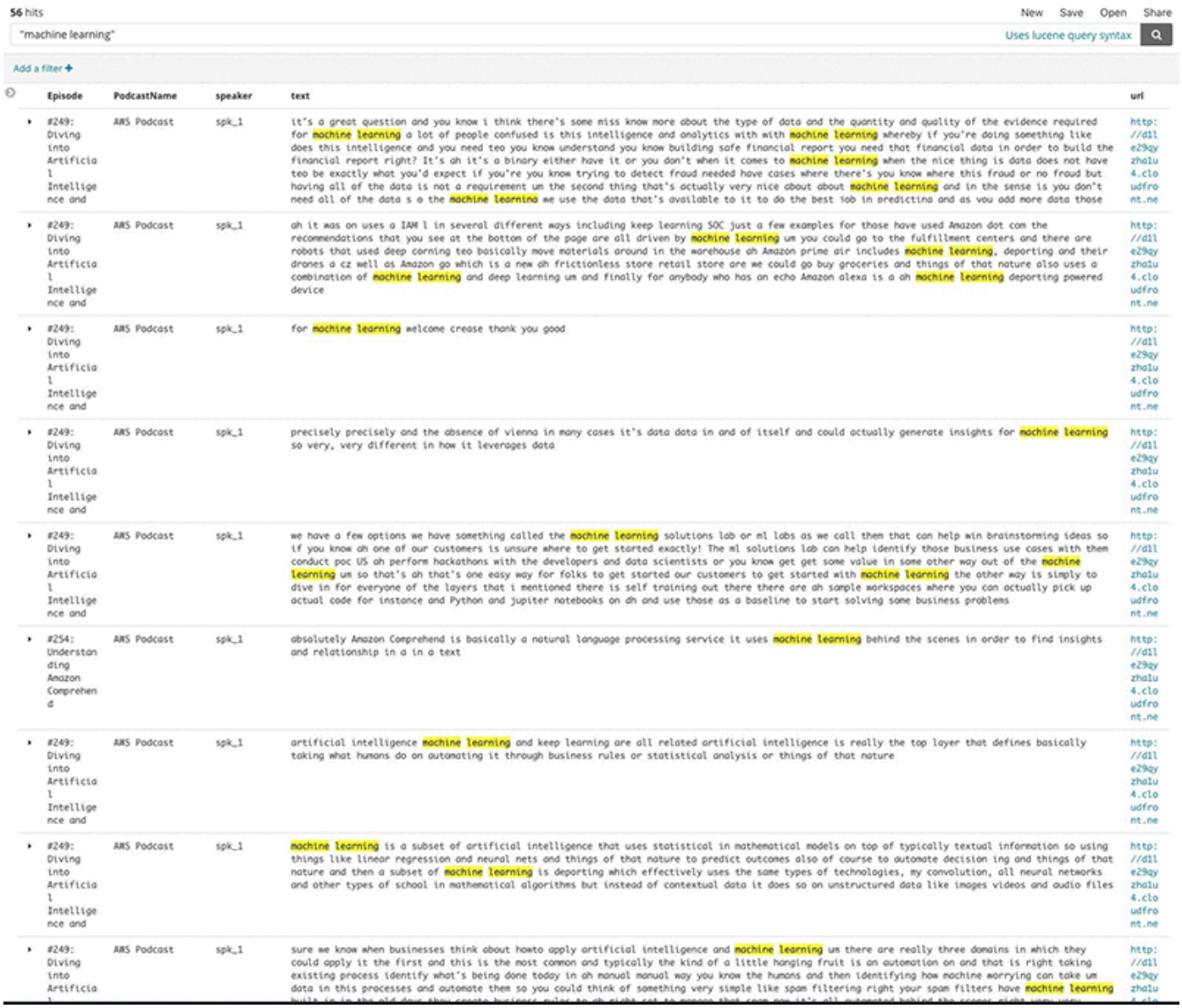
6. In the **Discover** section, ensure you have picked the paragraph



7. Now customize the search columns by adding the `PodcastName`. Repeat this for `Episode`, `speaker`, `text`, and `url`.



8. You now have a screen that looks like this:



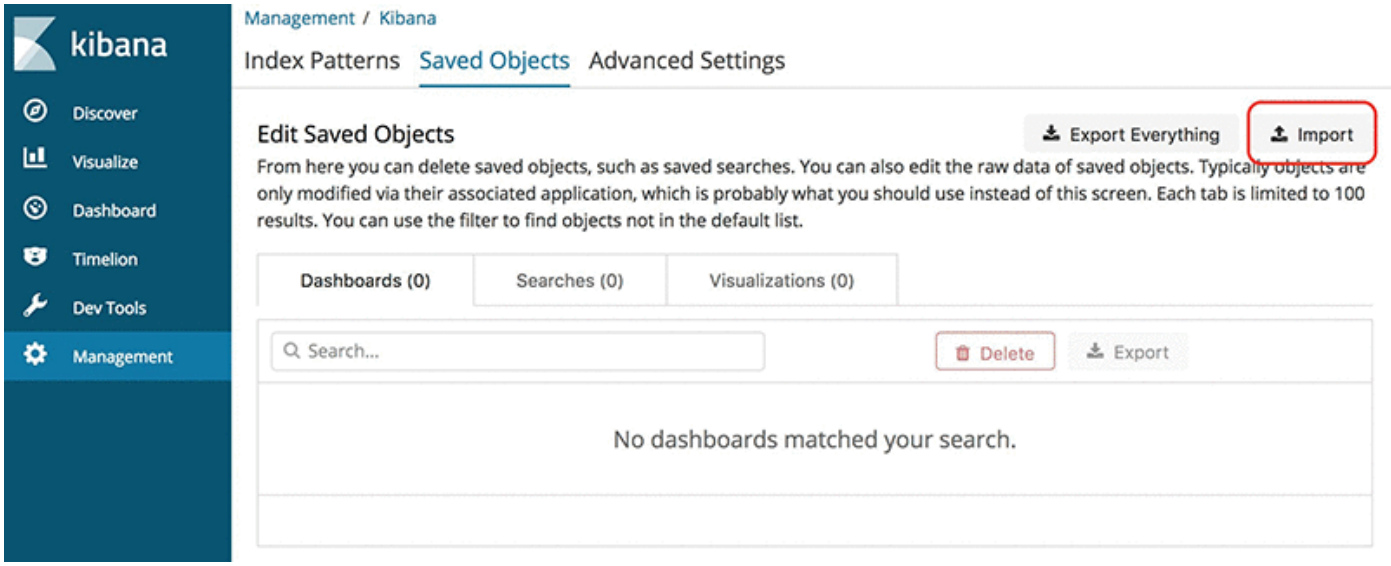
Searching the paragraph index

Now, have some fun searching for things. When you click the link in the url column, the browser will open the audio and seek to a point shortly before the word is mentioned. There are occasions where the browser will drop you slightly after, so you may need to move back a few seconds.

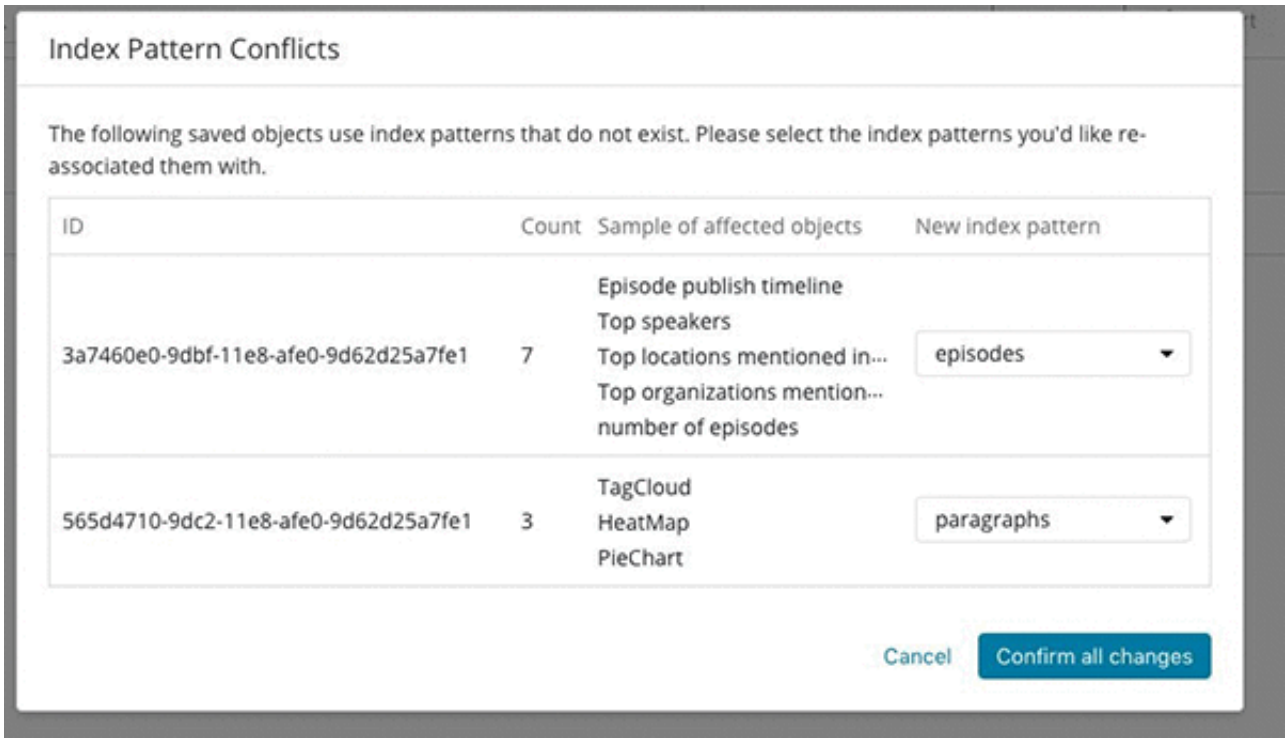
Import a Kibana Dashboard

Now for some fun visualizations!

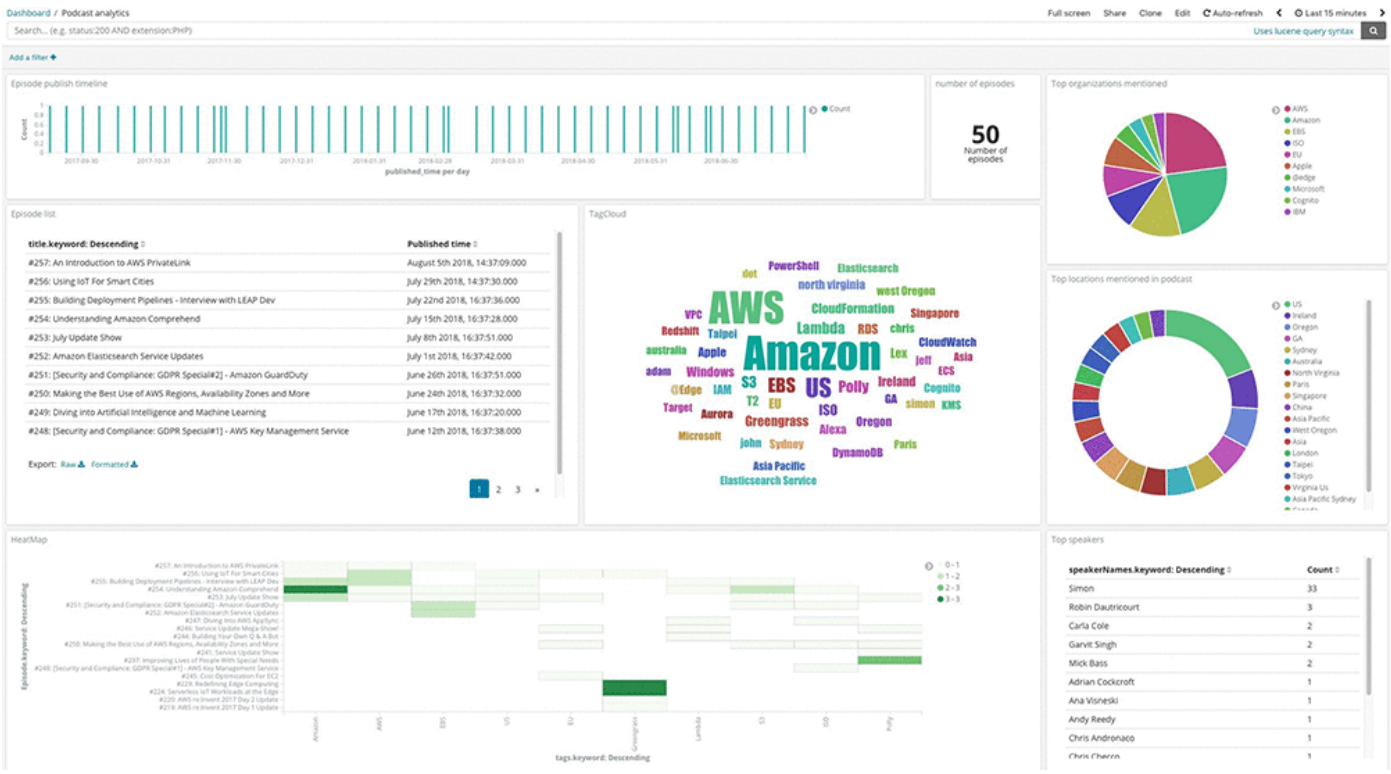
1. Download the Kibana Dashboard file here
2. On the Kibana home page, choose **Management** on the left side toolbar.
3. On the management page, choose **Import**and select kibana.json, which was downloaded in #1.



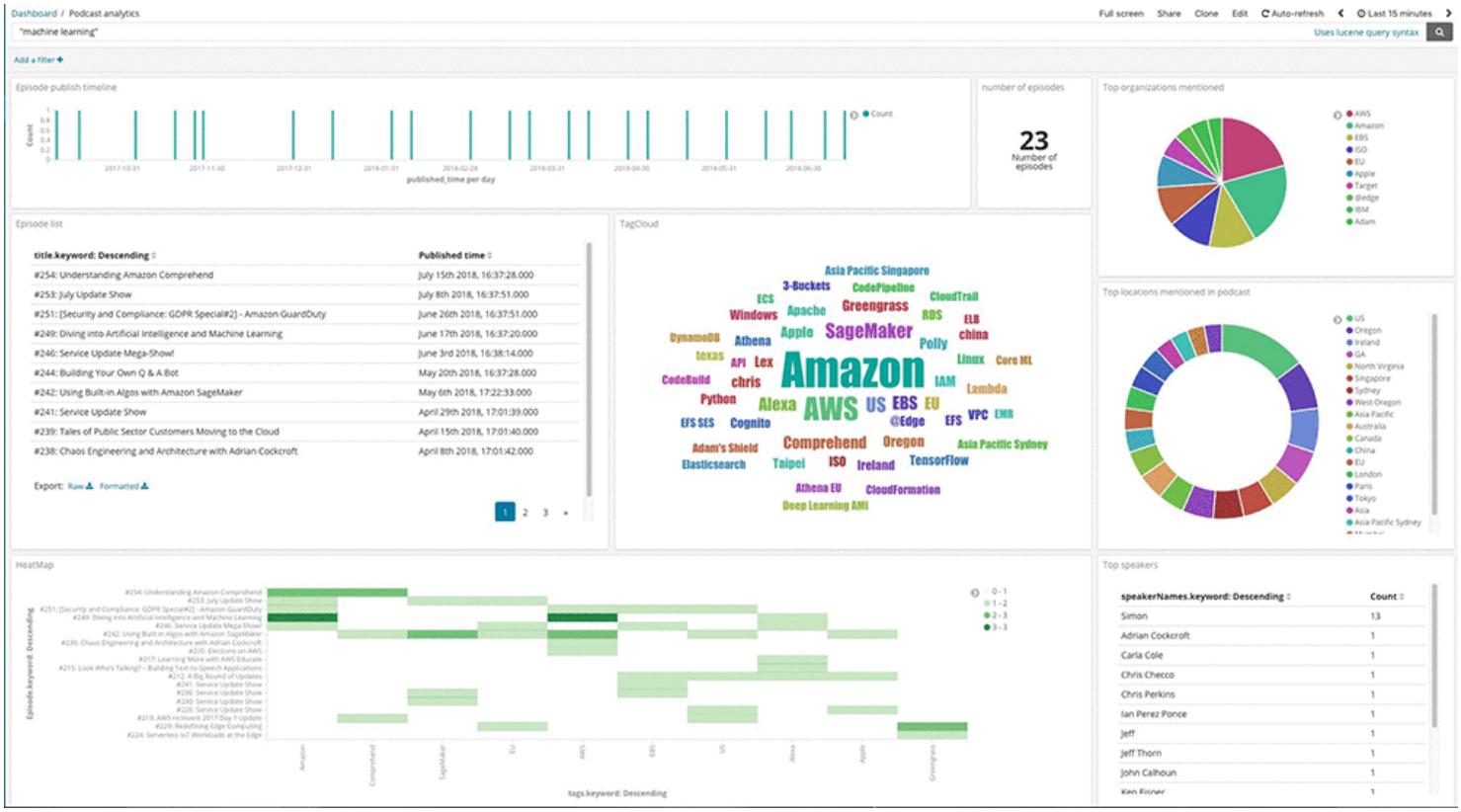
4. You need to map the visualization objects to the indices in your Kibana application as shown in the following screenshot. Confirm **Yes, overwrite all objects** when the pop-up appears.



5. You can now find the imported dashboard under **Dashboard→ Podcast analytics.**



You can do analysis using the dashboard by leveraging the search functionality. For example, by using the search term `machine learning`, we update the analytics to visualize only episodes that contains the phrase “machine learning.”



When you have finished exploring the demo application, remember to delete the CloudFormation stack to prevent unnecessary charges.

Conclusion

In this blog post, we showed you how to leverage the power of Amazon Transcribe and Amazon Comprehend to transcribe podcast episodes, extract keywords and entities from the podcast, and build a search index and dashboard using Amazon Elasticsearch Service. Combined with the serverless compute platform using AWS Lambda, you can build a system like this quickly with zero servers to manage, while AWS Step Functions makes the orchestration of various steps easily manageable in a visualizable workflow.

With the range of machine learning capabilities provided by the AWS platform, there are many ways you can take this demo further. You can, for example:

- Build a recommendation engine for podcast episodes by training a model based on user's ratings on episodes they have listened to. See https://aws.amazon.com/blogs/machine-learning/build-a-movie-recommender-with-factorization-machines-on-amazon-sagemaker/for some approaches on building recommendation engines.
- Group the episodes using Neural Topic Modeling. See https://aws.amazon.com/blogs/machine-learning/introduction-to-the-amazon-sagemaker-neural-topic-model/for an example on doing topic modeling using Amazon SageMaker. Amazon Comprehend also provide a ready-to-use API for topic modeling: https://docs.aws.amazon.com/comprehend/latest/dg/topic-modeling.html.

About the authors

Angela Wang is a Solutions Architect at Amazon Web Services based in Chicago. She helps customers build scalable and resilient architectures on the AWS cloud and is especially passionate about serverless technologies. Outside of work, she loves climbing, backcountry skiing, traveling – while listening to lots of podcasts along the way.

Mike Gillespie is a solutions architect at Amazon Web Services. He works with the AWS customers to provide guidance and technical assistance helping them improve the value of their solutions when using AWS. Mike specializes in helping customers with serverless, containerized, and machine learning applications. Outside of work, Mike enjoys being outdoors running and paddling, listening to podcasts, and photography.

AWS Podcast

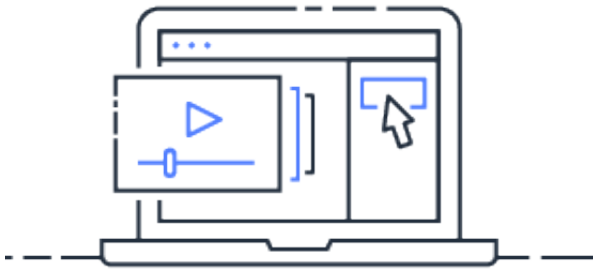
Subscribe for weekly AWS news and interviews

Learn more »

AWS Partner Network

Find an APN member to support your cloud business needs

Learn more »



AWS Training & Certifications

Free digital courses to help you develop your skills

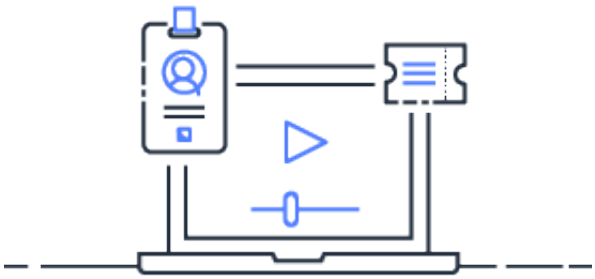
Learn more »

Resources

Getting Started
What's New
Top Posts
Official AWS Podcast
AWS Case Studies

Follow

Twitter
Facebook
LinkedIn
Twitch
RSS Feed
Email Updates



AWS Events
Discover the latest AWS events in your region
Learn more »

Related Posts

Intuit: Serving Millions of Global Customers with Amazon Connect

Analyze content with Amazon Comprehend and Amazon SageMaker notebooks

Amazon Comprehend now supports resource tagging for custom models

Amazon Comprehend now support KMS encryption

AI Powered Speech Analytics for Amazon Connect (Preview)

Newstag improves global video news discoverability using AI language services on AWS

Bridgeman Images uses Amazon Translate to establish their business globally

Use AWS Machine Learning to Analyze Customer Calls from Contact Centers (Part 2): Automate, Deploy, and Visualize Analytics using Amazon Transcribe, Amazon Comprehend, AWS CloudFormation, and Amazon QuickSight