

# AWS Embedded Transcoder Deployment Guide

Version 1.0

November 22, 2021

## **LiveRamp**

225 Bush Street, 17th Floor

San Francisco CA 94104

Tel. (866)352-3267

Contact: [awseembeddedtranscoder@liveramp.com](mailto:awseembeddedtranscoder@liveramp.com)

LIVERAMP MAKES NO WARRANTIES OR REPRESENTATIONS IN THIS GUIDE, EXPRESS OR IMPLIED, INCLUDING ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, OR, WITH RESPECT TO DATA, ACCURACY, COMPLETENESS, OR CURRENTNESS. THE SERVICE IS PROVIDED TO THE COMPANY STRICTLY ON AN "AS-IS, AS-AVAILABLE" BASIS, SUBJECT ONLY TO THE TERMS AND CONDITIONS CONTAINED IN THE APPLICABLE LEGAL AGREEMENT(S).

# Table of Contents

AWS Embedded Transcoder Deployment Guide	2
Table of Contents	3
Front Matter	5
Audience	5
Prerequisites	5
Product Support	5
Executive Summary	6
Architecture	7
CloudFormation	11
Authentication and Initialization	16
Transcoding	18
Metrics	21
API Reference	22
/init Endpoint	22
/request Endpoint	23
Errors	25
See Also	26

# Front Matter

## Audience

This document is intended for existing LiveRamp customers with data already resolved to RampID looking for a way to translate to a partner encoded RampID within their own AWS VPC. The following sections will walk you through the deployment process starting in AWS Marketplace, downloading the CloudFormation template, and instantiating the Transcoder appliance. The audience is a data engineer tasked with transcoding.

## Prerequisites

LiveRamp's Embedded Transcoder offering is available through the AWS Marketplace and deployed through a [CloudFormation](#) template that is stored in a GitHub repository.

The following prerequisites are needed:

1. Access to a public GitHub repository.
2. An AWS Marketplace account and ability to procure through the Marketplace.
3. Access to [CloudFormation](#) and the ability to create [IAM](#) roles and resources.
4. Customer data already resolved to RampID, this service will transcode from one RampID encoding to another.
5. Multi-party consent is required, both parties need to execute a LiveRamp permission order for access to be permitted to transcode to a partner RampID encoding.

## Product Support

Support for Embedded Transcoder can be found on the following: [developer site](#), [community support site](#) and by working with your LiveRamp account team. LiveRamp's Embedded Transcoder deployment is supported by our Global Support team, a 24/7 operation to provide the first line of defense on technical issues. Support for this product will be limited to interactions between the proxy and our encoding technology (including incoming requests) and from the proxy out to our core services (authorization requests and updates). Other support issues involving AWS technology will need to be directed to the AWS support team.

## Executive Summary

LiveRamp's Embedded Transcoder allows for the conversion of RampID from one encoding to RampID in another partner encoding. This conversion takes place inside an AWS Virtual Private Cloud (VPC) so that the data never leaves your control, with attendant latency and security benefits. Transcoding requires the agreement of both parties, owners of each domain.

LiveRamp's AWS Embedded Transcoder deployment invokes [SageMaker's network isolation mode](#) on the appliance, protecting your VPC and deployed LiveRamp technology. The system design forces all interactions between the encryption technology and your VPC through a proxy service. This proxy also handles authentication requirements and updates to the appliance.

This deployment guide details how to initialize the appliance using a CloudFormation template, walks you through authentication, and describes how to call the transcoding service. The document walks through additional details on the architecture and design, and provides additional resources for review.

The steps for deploying LiveRamp's Embedded Transcoder are:

1. Subscribe to LiveRamp's Embedded Transcoder product through the AWS Marketplace.
2. Execute multi-party agreements through LiveRamp for data exchange.
3. Download the template from the [public Github repository](#).
4. Run the template in AWS CloudFormation
5. Run the LiveRamp `/init` endpoint to initialize the transcoder.
6. Get a Smart Token from SM Proxy's `/request` endpoint. Tokens must be refreshed every 15 minutes by calling the `/request` endpoint and passing the `rpc` parameter the value `refresh`.

The system must also be reinitialized every day after 6 AM CST (UTC -6:00).

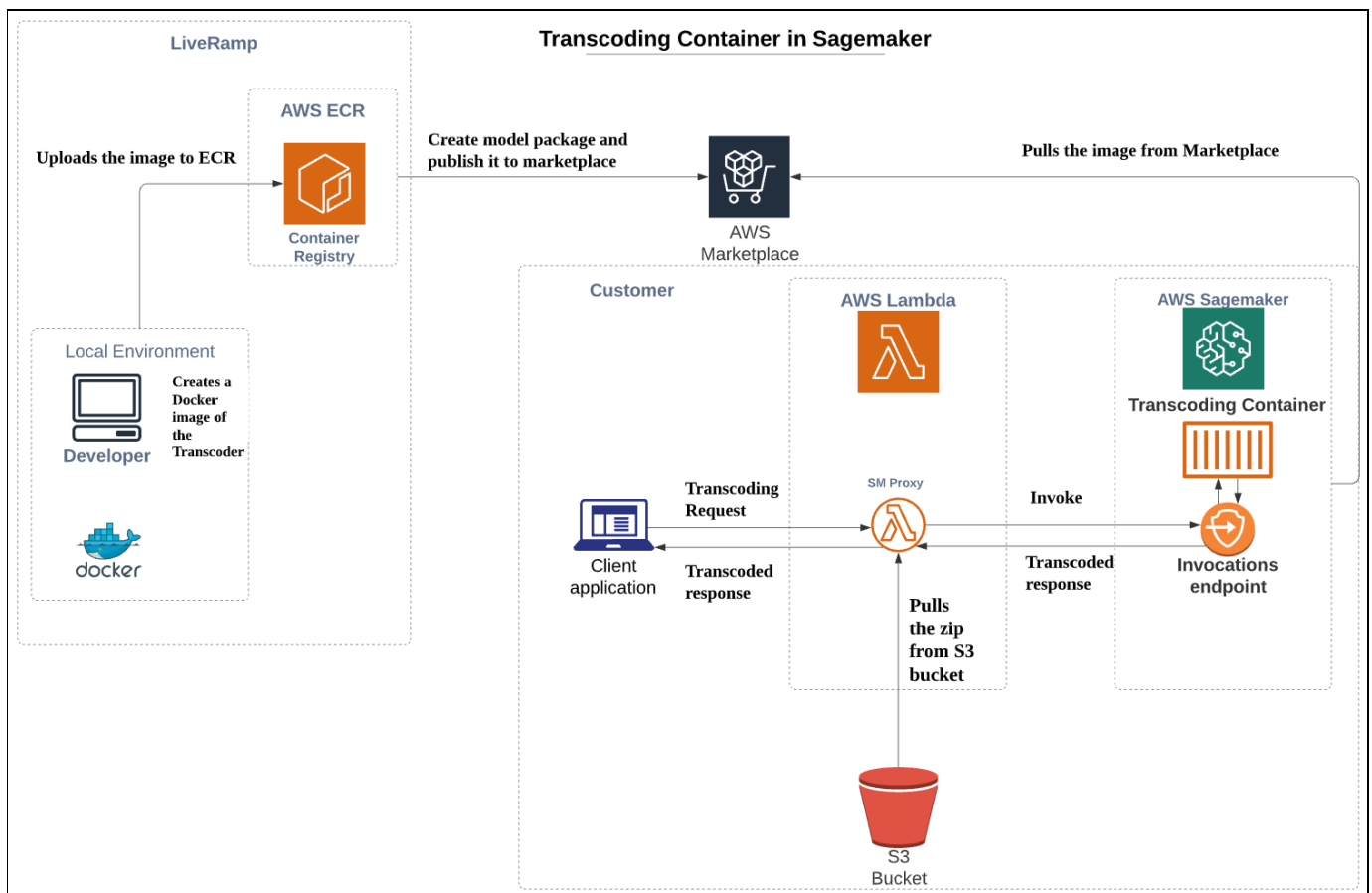
7. Use the SageMaker proxy `/request` endpoint to execute the transcoding; `client_id`, `client_secret`, and the `rpc` parameter set to `request` as parameters of the call.

A typical use case for Transcoding is the ability to unlock collaboration use cases by translating an encoded RampID to a partner's encoding (or vice versa). By leveraging LiveRamp's Embedded Transcoder offering, you're able to translate across identity spaces to drive actionable insights.

The Embedded Transcoder allows you to connect your LiveRamp identities with external datasets to solve problems of interest. Brands and platforms want to connect, control, and activate data quickly and securely across the advertising technology marketplace but are faced with security concerns and latency issues making it difficult to connect and take action on customer data.

## Architecture

The LiveRamp Embedded Transcoder is deployed by running a CloudFormation template in AWS using the `/init` endpoint, as shown in Figure 1 below. This template stands up two Virtual Private Clouds, one for the customer and a second for LiveRamp's ID-API authentication. To obtain the needed credentials in the form of a Java Web Token (JWT) a call is made to the `/auth` endpoint. The session token is valid for 15 minutes from issuance. The third service call is to the `/request` endpoint, which passes the token and specifies the data to be processed.

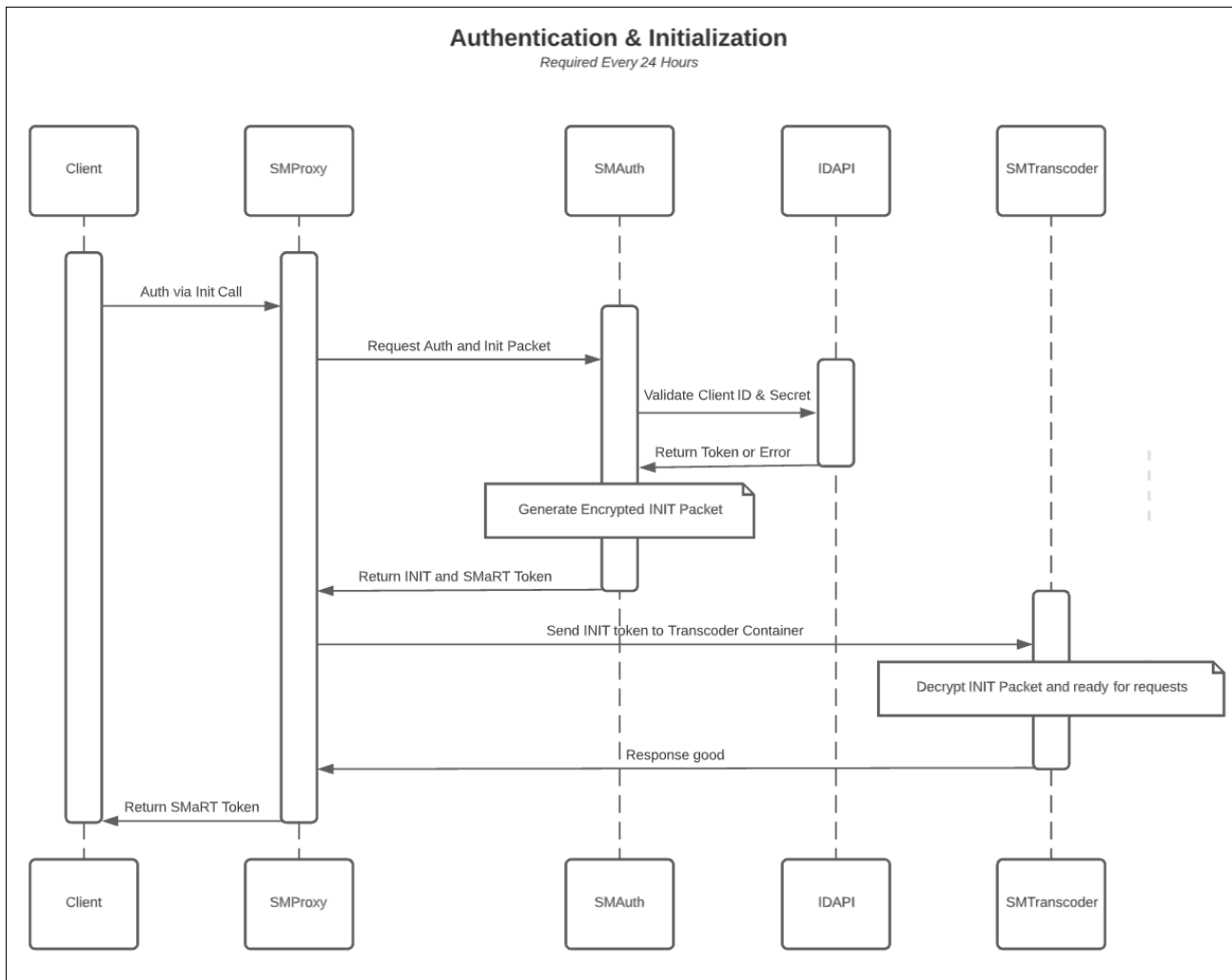


**Figure 1. AWS Transcoder system diagram.**

Figure 2 below shows the steps involved in authentication and initialization of the transcoding process from a client/server perspective. The first step deploys the template in CloudFormation to create the system shown above. To initialize the system a service call is made to the `/init` endpoint. The next step is to authenticate the client by placing a service call to the `/auth` endpoint, which returns a JWT token. A request for transcoding is made to the `/request` endpoint passing the JWT, setting the RPC parameter to

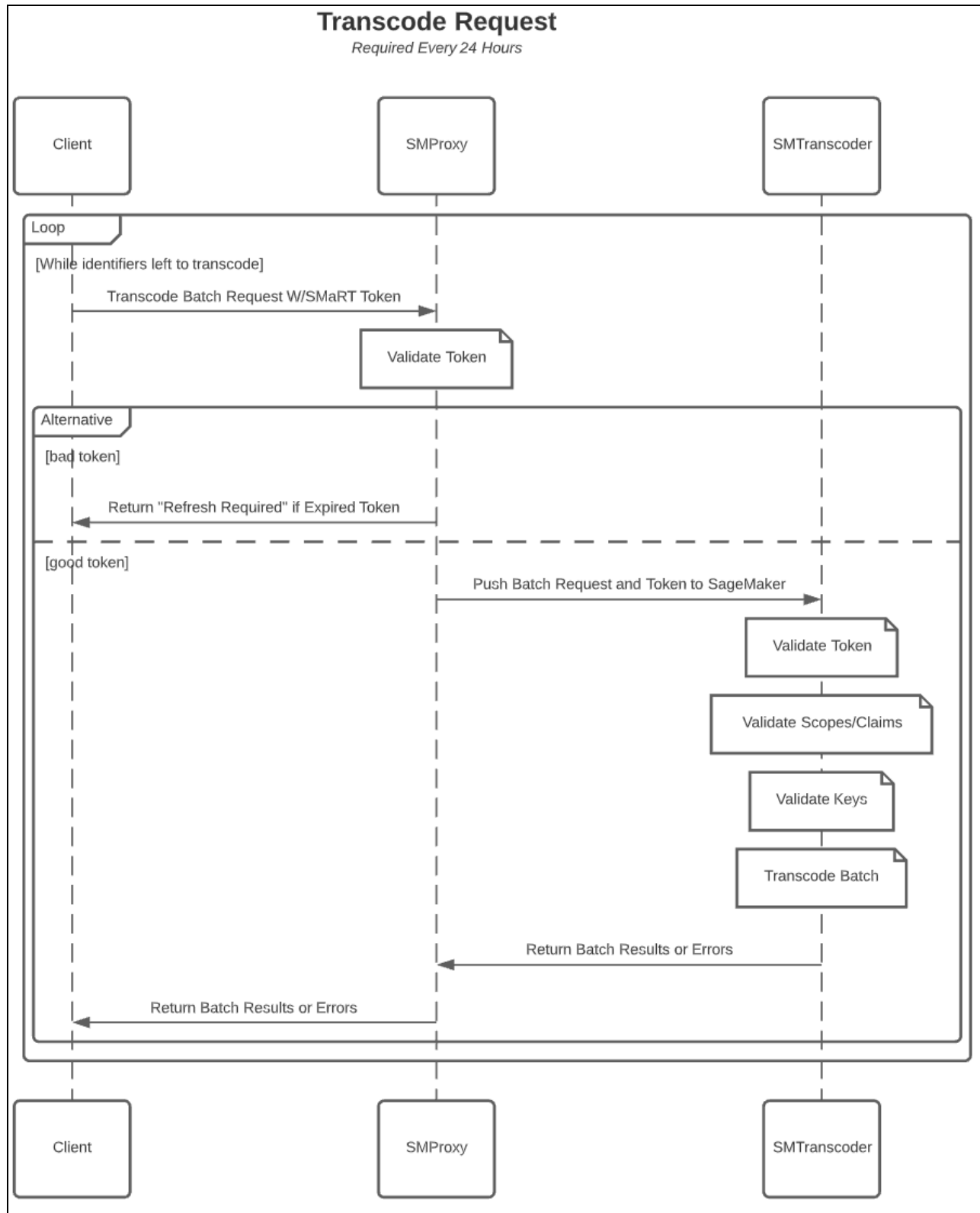
request, and specifying the Ramp IDs to be transcoded into the new domain.

**NOTE:** In Figures 2-4 the sequence of events goes from the top to the bottom.



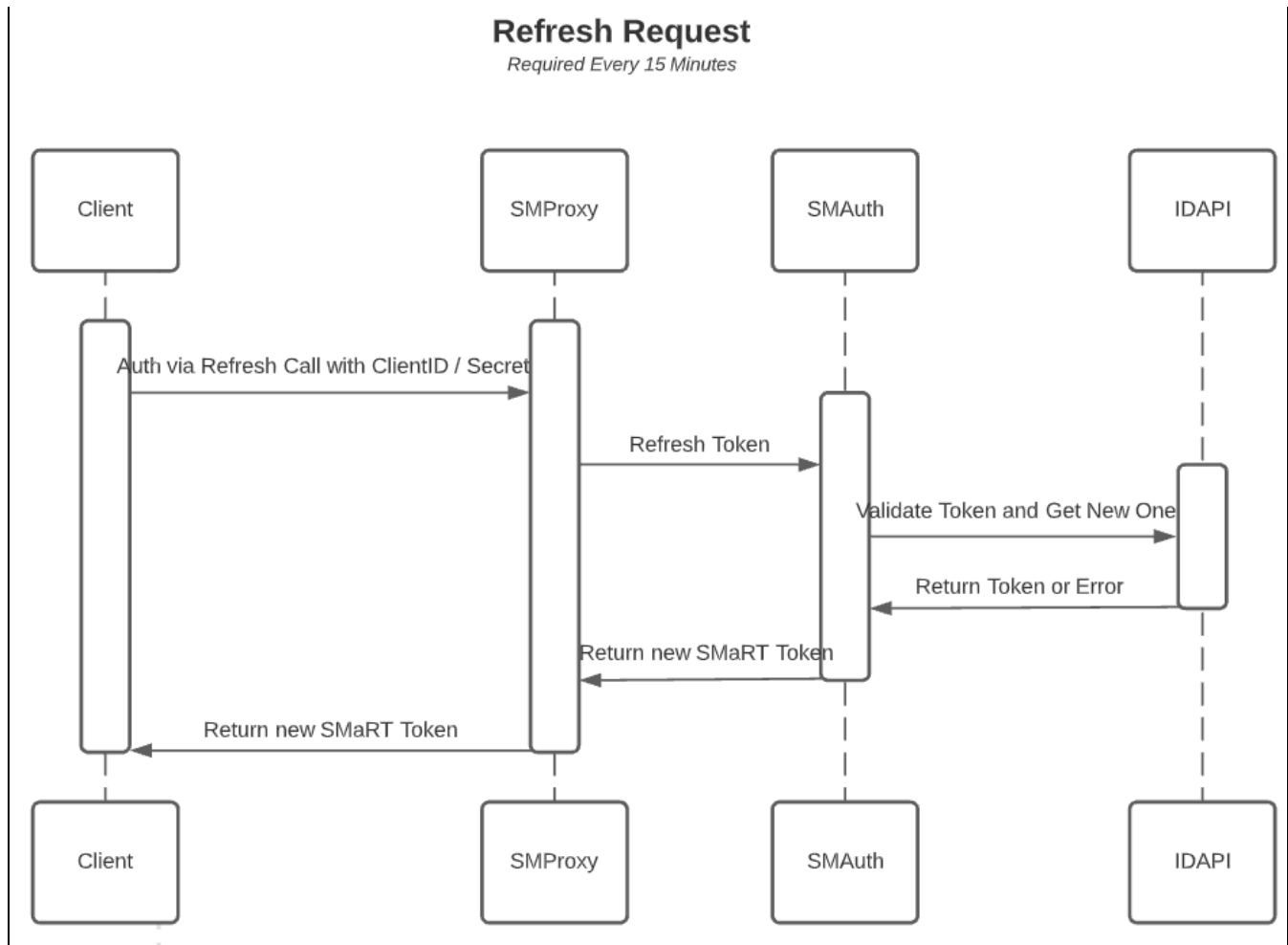
**Figure 2.** Authentication and initialization of the transcoding process.

In Figure 3 below the actual request, the process is illustrated. Authentication, initialization, and transcoding operations are performed as long as the system is initialized less than 24 hours before the request. The session token expires after 15 minutes, after which a refresh request for a new token must be issued.



**Figure 3.** A schematic of the transcode request process.

Figure 4 below shows the schematic for a refresh request.



**Figure 4.** The transcode refresh request process.



# CloudFormation

The LiveRamp Embedded Transcoder is deployed by running a template in the form of a YAML file within AWS CloudFormation. The file is contained inside a ZIP archive obtainable from a public GitHub repository. The template installs a proxy service written as an AWS Lambda function, and a SageMaker instance inside a Virtual Private Cloud. The VPC settings place the Transcoder deployment into network isolation mode, which only allows incoming requests through the proxy server to be processed.

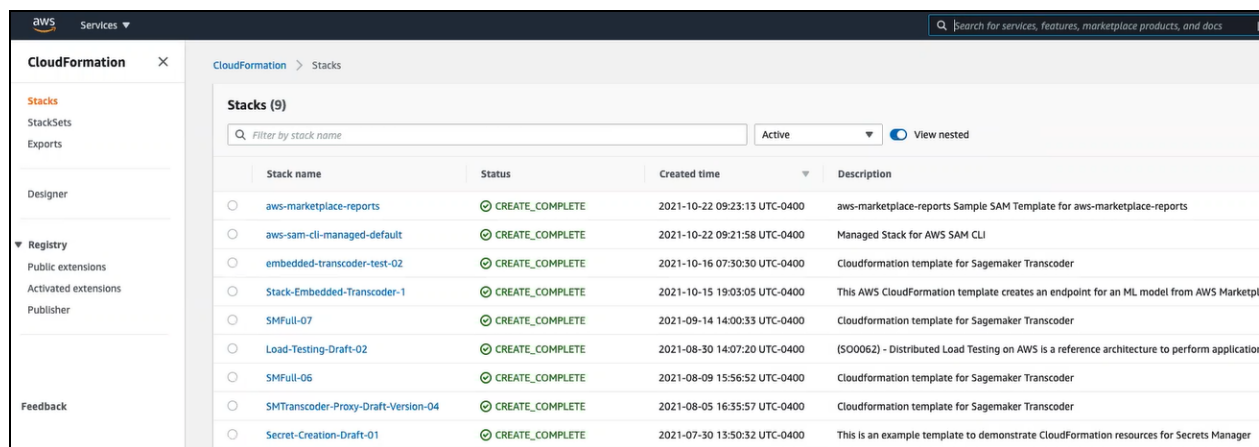
To deploy the AWS Embedded Transcoder, do the following:

1. Go to the Github repository located [here](#).
2. Download the `CloudFormation_Template_XXX_Transcodes.YAML` template desired to the storage system of your choice.

Storage targets include either cloud or local storage.

**NOTE:** Select `XXX=10k` for up to 10,000 transcoding operations per second, `XXX=100K` for up to 100,000 transcoding operations per second, or `XXX=1Million` for one million (1,000,000) transcoding operations per second.

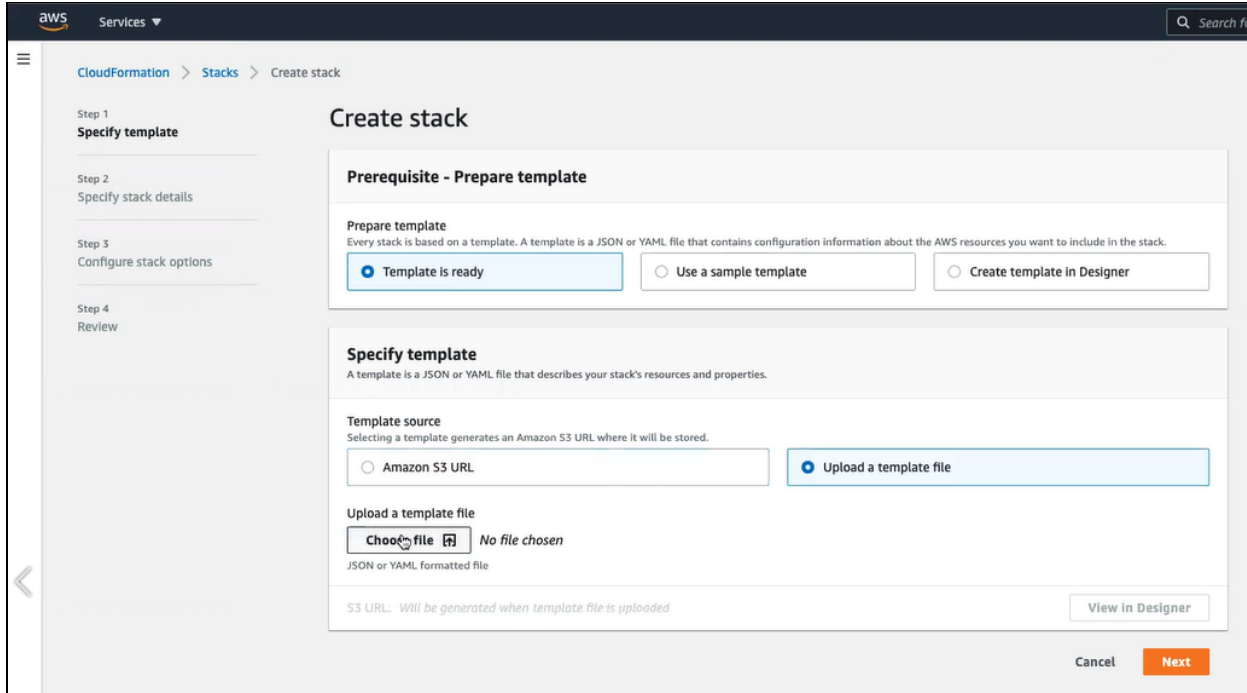
3. Log into an AWS CloudFormation account that allows the user to create stacks from project templates and click on Stacks in the navigation pane as shown in Figure 5 below.



**Figure 5.** Open the CloudFormation Stacks section to use the deployment template.

4. Click on the Stack Actions drop-down menu at the top right of the Stacks section and select the create stack With new resources selection.
5. On the Create Stack page shown in Figure 6 below select the Template source (Amazon S3 URL or Upload a template file), click the Choose file button, select your template file, and then click the

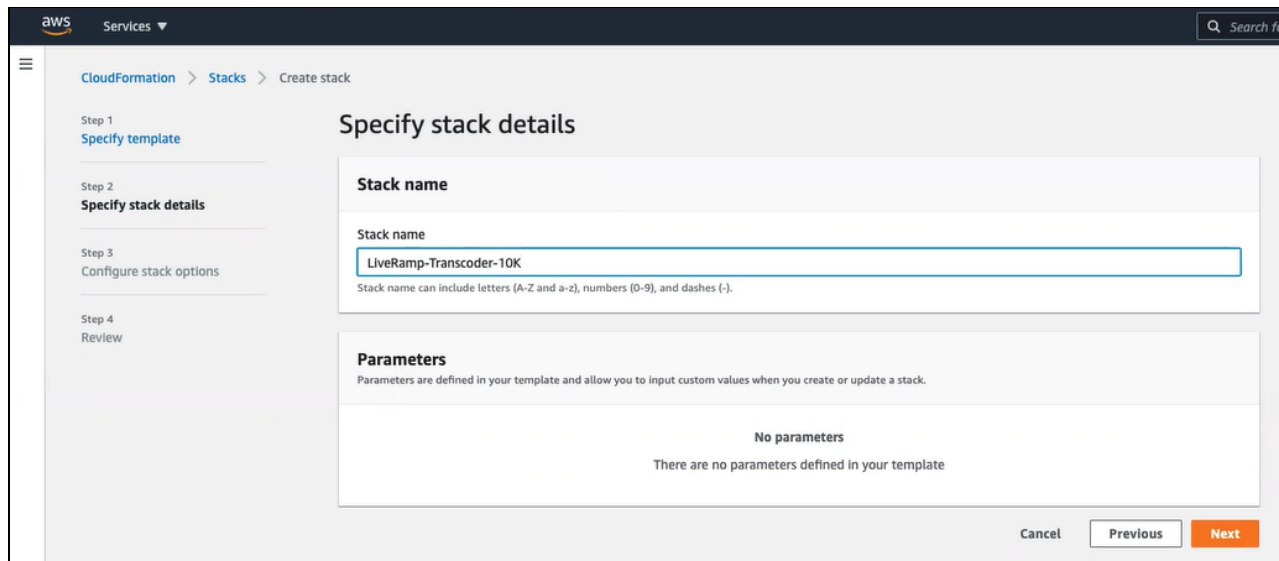
Next button.



The screenshot shows the AWS CloudFormation console with the 'Create stack' wizard. The left sidebar indicates the current step is 'Step 1: Specify template'. The main content area is titled 'Create stack' and includes a 'Prerequisite - Prepare template' section with three radio buttons: 'Template is ready' (selected), 'Use a sample template', and 'Create template in Designer'. Below this is the 'Specify template' section, which includes a 'Template source' section with radio buttons for 'Amazon S3 URL' and 'Upload a template file' (selected). The 'Upload a template file' section shows a 'Choose file' button and a 'No file chosen' message. At the bottom right, there are 'Cancel' and 'Next' buttons.

**Figure 6.** The CloudFormation Create stack page.

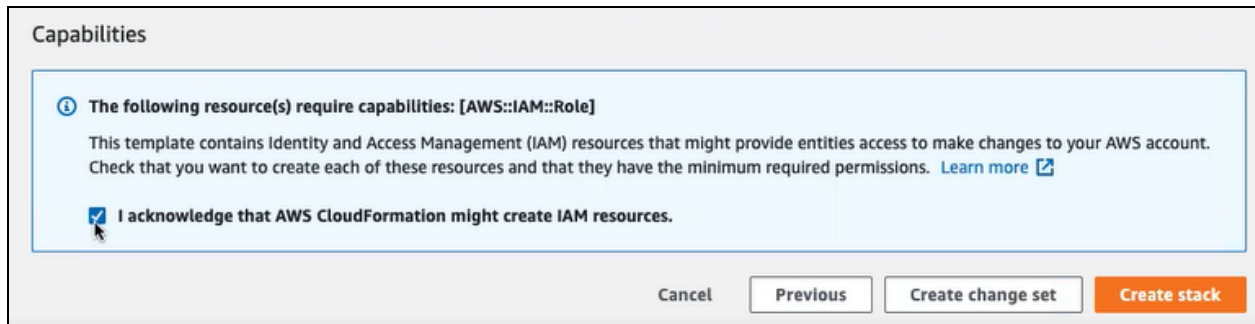
6. On the Specify stack details page, shown in Figure 7 below, enter the name of your new transcoder deployment, then click the Next button.



The screenshot shows the AWS CloudFormation console with the 'Specify stack details' page. The left sidebar indicates the current step is 'Step 2: Specify stack details'. The main content area is titled 'Specify stack details' and includes a 'Stack name' section with a text input field containing 'LiveRamp-Transcoder-10K'. Below this is a 'Parameters' section with a message stating 'No parameters' and 'There are no parameters defined in your template'. At the bottom right, there are 'Cancel', 'Previous', and 'Next' buttons.

**Figure 7.** The Specify stack details step is where you name your deployment.

7. On the Configure stack options page (Step 3) accept the default selections and click the Next button.
8. Complete the wizard by scrolling down to the Capabilities section and enabling the **I acknowledge that AWS CloudFormation might create IAM resources** check box; then click the Create stack button as shown in Figure 8 below.

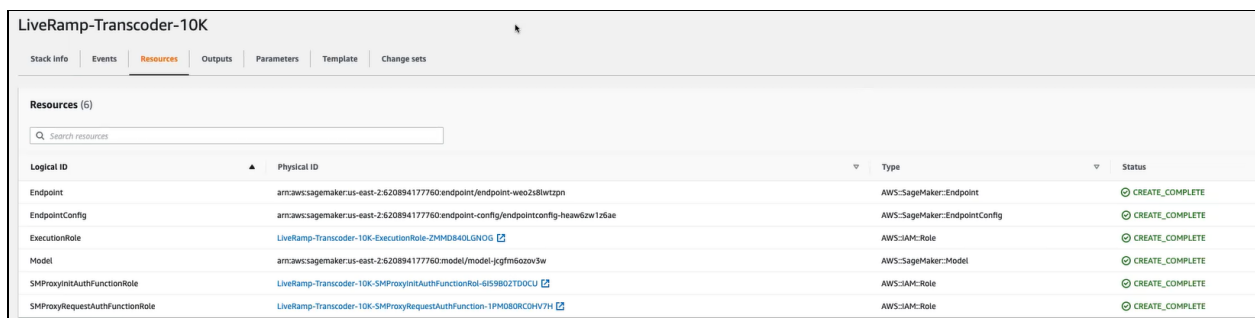


**Figure 8.** Enable the acknowledgment and click the Create stack button to initiate the deployment.

When the wizard closes you are returned to the Stacks section and the new deployment will be listed as the latest event, along with its status as `CREATE_IN_PROGRESS`. The staging time is variable, depending on several factors such as datacenter used, network traffic, the template that you chose, and more. A typical deployment will take from about 7 to 15 minutes and completion changes the status of the deployment to `CREATE_COMPLETE`.

**NOTE:** All elements of the deployment must be successfully deployed or the system rolls back the resources to their previous state.

When the deployment is complete all of the resources will show a `CREATE_COMPLETE` status as shown in Figure 9 below. The Endpoint resource is the last resource to be completed.

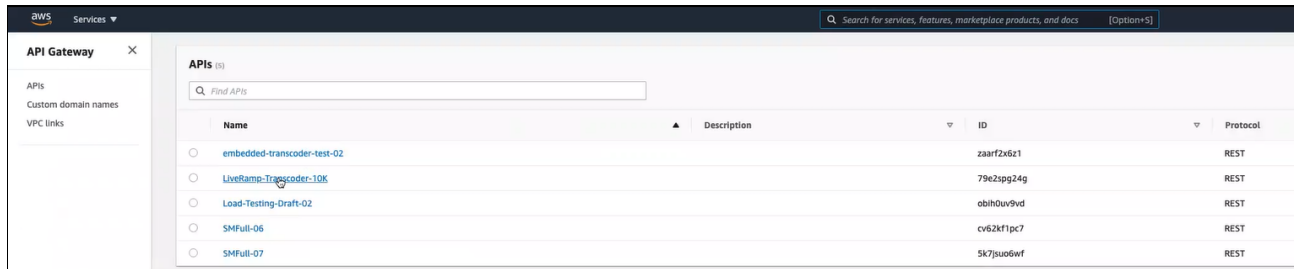


Logical ID	Physical ID	Type	Status
Endpoint	arn:aws:sagemaker:us-east-2:620894177760:endpoint/endpoint-weo2s8lwtzpn	AWS::SageMaker::Endpoint	CREATE_COMPLETE
EndpointConfig	arn:aws:sagemaker:us-east-2:620894177760:endpoint-config/endpointconfig-headwz1z6ae	AWS::SageMaker::EndpointConfig	CREATE_COMPLETE
ExecutionRole	LiveRamp-Transcoder-10K-ExecutionRole-2HMD840LQNDG	AWS::IAM::Role	CREATE_COMPLETE
Model	arn:aws:sagemaker:us-east-2:620894177760:model/model-jcgmiozov3w	AWS::SageMaker::Model	CREATE_COMPLETE
SMPProxyAuthFunctionRole	LiveRamp-Transcoder-10K-SMPProxyAuthFunctionRole-6S98Q2TDOCU	AWS::IAM::Role	CREATE_COMPLETE
SMPProxyRequestAuthFunctionRole	LiveRamp-Transcoder-10K-SMPProxyRequestAuthFunction-1PM080RCQHV7H	AWS::IAM::Role	CREATE_COMPLETE

**Figure 9.** All of the completed resources for the transcoder deployment are shown here.

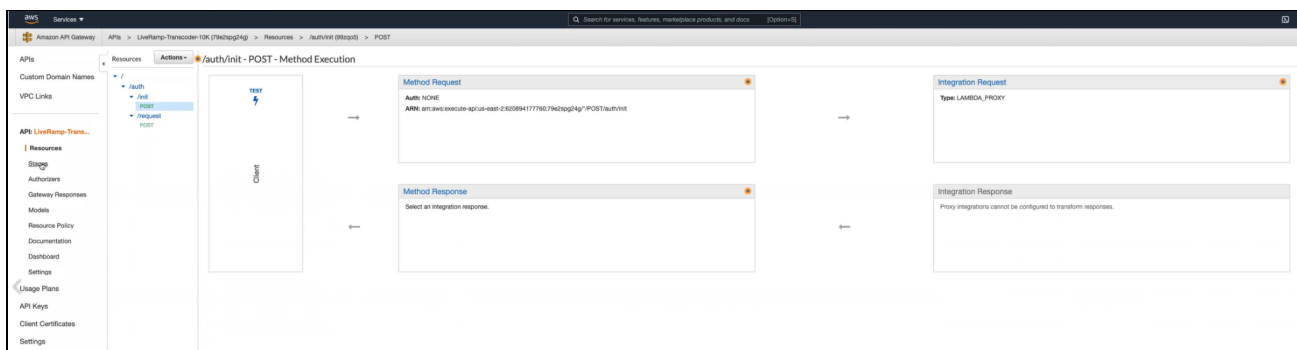
To determine the transcoder endpoints from the AWS API Gateway:

1. Log into AWS API Gateway and select the transcoder deployment from the list of the available APIs as shown in Figure 10 below.



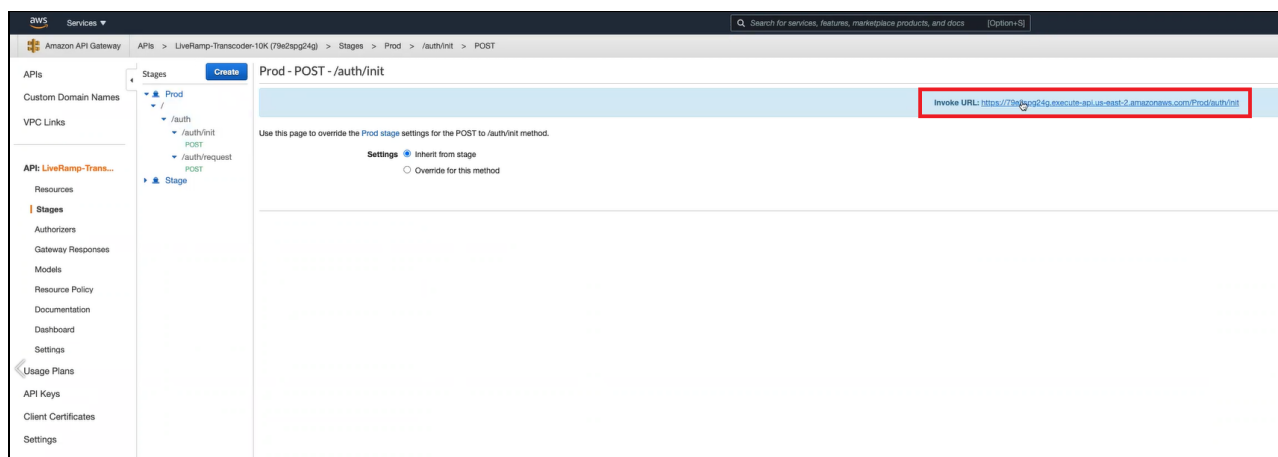
**Figure 10.** Select the transcoder deployment in AWS API Gateway to open the endpoint's methods.

2. On the Resources page click on the Stages link in the left hand navigation pane, as shown in Figure 11 below.



**Figure 11.** The installed endpoints for the transcoder.

3. Click on the Prod environment, then click on the endpoint you want to invoke as shown in Figure 12 below.  
The URL to invoke the endpoint is listed in the blue stripe at the top of the page shown boxed in red.

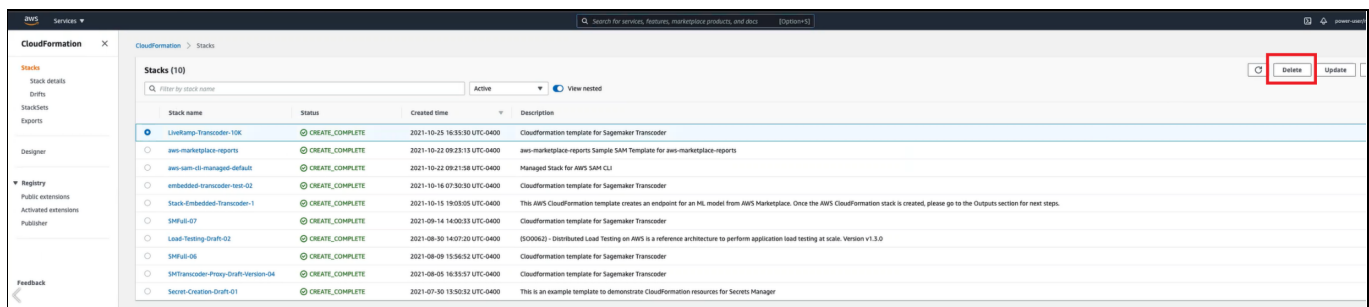


**Figure 12.** The Invoke URL endpoint appears in the Stages section for each method.

**CAUTION:** The creation of a stack deploys AWS resources that generate charges for an organization. You should use the template that is the appropriate size, and delete stacks when the work is completed to minimize your costs.

To delete the transcoder deployment:

1. Log into the CloudFormation console with appropriate privileges to delete a stack.
2. Select the deployment stack and click on the Delete button, as shown in Figure 13 below.



**Figure 13.** To delete the transcoder, delete its stack in CloudFormation.

## Authentication and Initialization

The Embedded Transcoder uses SageMaker Authentication (SM Auth) to validate user requests. A user of the transcoding appliance calls the `/init` endpoint to activate the system by calling an AWS Lambda function within LiveRamp's VPC. The `init_token` Lambda function returns the required encryption keys.

A call to the `/auth` endpoint calls a second `auth_token` Lambda function in the LiveRamp VPC, which then places a service call to the Identity API to obtain a session key in the form of a smart token that is used in the actual transcoding service request. That session key contains domain information used in the transcoding process. It is called a "Smart Token" because it contains information required by the specific transcoding process that the customer is performing.

The `auth_token` performs the following operations when called by SM Proxy auth service which resides in the Customer's VPC

1. Reads the input from the SM Proxy service `auth` request for client credentials.
2. Calls the ID-API `/token` endpoint with those credentials and fetches the ID-API authentication token.
3. Calls the Identity API `/domains` endpoint using the ID-API authentication token and fetches the domain keys needed for transcoding.
4. Creates a Scopes object that indicates the transcoding directions for the domains.
5. Wraps the ID-API token and Scopes object along with expiration time stamp, and encodes that data to generate a JWT token called a Smart Token.

A cURL request for the `auth_token` would take the following format:

```
curl --location --request POST
'https://<hostname>.execute-api.<region>.amazonaws.com/Stage/token/r
equest' \
--header 'Content-Type: application/json' \
--data-raw '{
    "httpMethod": "POST",
    "client_id": "<insert client id here>",
    "client_secret": "<insert client secret here>"
}'
```

The `init_token` LiveRamp Lambda function performs the following operations when called by SM Proxy init service which resides in the customer's VPC:

1. Reads the input from the SM Proxy service `init` request for client credentials.
2. Calls ID-API `/token` endpoint with those credentials and fetches the ID-API authentication token.
3. Calls the `/domains` endpoint using the ID-API authentication token and fetches the domain keys needed for transcoding.
4. Creates a `Scopes` object that contains the transcoding directions for the domains.
5. Wraps the ID-API token and `Scopes` object along with expiration time stamp, and encodes that data to generate a JWT token called a SMART TOKEN.
6. The `init_token` Lambda function encrypts the domain keys, generates cipher text, a session key, and an initialization vector.
7. The `init_token` Lambda function encodes the data from Step 6 to generate a JWT token called the `init_token`.

The cURL request for the `init_token` is:

```
curl --location --request POST
'https://<hostname>.execute-api.<region>.amazonaws.com/Stage/token/i
nit' \
--header 'Content-Type: application/json' \
--data-raw '{
    "httpMethod": "POST",
    "client_id": "<insert client id here>",
    "client_secret": "<insert client secret here>"
}'
```

Only the calls that are needed for initialization and refreshing the token communicate with LiveRamp's APIs. All transcoding procedures run within the client's VPC.

## Transcoding

The SageMaker Proxy function is a Lambda routine that resides in the customer's VPC. It has two functions: `request_auth` and `init_auth`. SageMaker Proxy accepts only incoming service requests to perform transcoding operations. The transcoding operation is performed through AWS SageMaker.

The SageMaker `request_auth` function receives an input request from the client through its `/request` endpoint with the following parameters.

1. `smart_token` - a Java Web Token (JWT) used for authentication.
2. `transcode_data` - contains the transcode data in the form of a JSON array of JSON objects to be input.
3. `rpc` - that parameter indicates if it is an init, request or refresh.

The `request_auth` function performs the following validations before performing the real transcoding operation:

1. Validates the operation only if the `smart_token` is present in the input request body.
2. Validates the operation contains the `transcode_data` in the input request body.
3. Validates that the `rpc` parameter is present in the input request body and the value is either request or refresh.

If any of these validations fail then the `request_auth` function sends an appropriate error response to the client. The common errors included:

- 400 - Bad request.
- 401 - Unauthorized.
- 405 - Method not allowed.
- 500 - Internal Server Error.

The `request_auth` steps performed when RPC is refresh includes the following:

1. Validate the input request body to read the `client_id` and the `client_secret` parameters.
2. Call the SM Auth Request Service and pass those credentials to receive a new `smart_token`.
3. Send the `smart_token` back to the client.



The cURL call to the `request_auth` Lambda function is:

```
curl --location --request POST
'https://<hostname>.execute-api.<region>.amazonaws.com/Stage/auth/request' \
--header 'Content-Type: application/json' \
--data-raw '{
    "httpMethod": "POST",
    "client_id": "<insert client id here>",
    "client_secret": "<insert client secret here>",
    "rpc" : "refresh"
}'
```

The `request_auth` execution steps when RPC is the request type is:

1. Reads the input request body for the `smart_token` and `transcode_data` parameters.
2. Validates the token for signature and expiry.  
If invalid or expired it returns an appropriate error response to the client.
3. If the token is valid it calls the SageMaker endpoint and passes the transcode data and Smart Token as parameters.
4. The `request_auth` function receives the transcoded links from SageMaker and sends the response back to the client.

The cURL call to the `request_auth` Lambda function is:

```
curl --location --request POST
'https://<hostname>.execute-api.<region>.amazonaws.com/Stage/auth/request' \
--header 'Content-Type: application/json' \
--data-raw '{
    "httpMethod": "POST",
    "transcode_data" : [],
    "smart_token" : "<insert smart token here>",
    "rpc" : "request"
}'
```

The `/init_auth` function fetches the domain keys from the SageMaker Auth service and passes those

keys to Transcoder. Domain keys are retained for 24 hours.

The `/init_auth` function receives an input request from the client service that typically contains the following parameters:

1. `client_id` - used for authentication.
2. `client_secret` - contains the client secret that has access to ID-API.
3. `rpc` - that indicates if it is an `init`, `request`, or `refresh`.

Then `/init_auth` performs the following validations before performing the real operation.

1. Validates if `client_id` is present in the input request body.
2. Validates if `client_secret` is present in the input request body.
3. Validates if `RPC` is present in the input request body and the value is `init`.

If any of these validations fail then an error is returned to the client.

The `init_auth` execution steps when `RPC` is set to `init` are:

1. Calls the SM Auth Init service with the client credentials.
2. Validates the token for signature and expiry. If invalid or expired it returns an appropriate error response to the client.
3. If the token is valid it invokes the SageMaker endpoint and passes the `init` token to receive a response that indicates the status of Transcoding Initialization.
4. It extracts the smart token out of `init` token.
5. Sends the `smart_token` and initialization response status back to the client.

The cURL request for the `init_token` endpoint in a staging environment is:

```
curl --location --request POST
'https://<hostname>.execute-api.<region>.amazonaws.com/Stage/auth/in
it' \
--header 'Content-Type: application/json' \
--data-raw '{
```

```
"httpMethod": "POST",  
"client_id": "<insert client id here>",  
"client_secret": "<insert client secret here>",  
"rpc": "init"  
}'
```

## Metrics

There are multiple logs and metrics captured throughout the system. Some logs are stored only in the account where the appliance is deployed and if an issue arises, it is at the discretion of the AWS Account Owner on the logs and metrics to communicate to LiveRamp. Other metrics are communicated to LiveRamp automatically, including usage metrics (number of successful transcoding calls by domain) to assist with billing and authorization metrics (token validation failed) for example.

Other metrics may be stored in AWS Cloudwatch on the Account of the deployed appliance. If troubleshooting an issue with LiveRamp, LiveRamp may request copies/screenshots/dumps of logs and metrics in order to better assist with troubleshooting.

Additional metrics, measuring Transcoder performance, are under active development.

## API Reference

The Embedded Transcoder has two API endpoints: `/init` and `/request`. They are documented in the following two sections.

### `/init` Endpoint

Part	Purpose	Value
Title	Name	init
Path	URL	<code>/Prod/auth/init</code>
Method	OpType	POST
ShDesc	for SEO	Initialize SageMaker Transcoder
Desc	Context	
Parameters	Config.	<code>clientID (String); client_secret (String); rpc (String, default=int)</code>
Request	Input	<code>/prod/auth/init</code>
cURL		<pre>curl --location --request POST 'https://&lt;hostname&gt;.execute-api.&lt;region&gt;.amazon aws.com/Prod/auth/init' \ --header 'Content-Type: application/json' \ --data-raw '{ "httpMethod": "POST", "client_id": "&lt;insert client id here&gt;", "client_secret": "&lt;insert client secret here&gt;", "rpc": "init" }'</pre>
Response	Output	<pre>{ "transcoder_response": "Transcoder Initialization successful", "smart_token": "xxxxxxxxxxxxx" }</pre>
Status & Errors	Output	<pre>{ "200": "Success", "400": "Bad Request", "401": "Not authenticated", "500": "Server Error" }</pre>
Tags	Category	

Status	Status	Production
Authentication Method	AAA	Yes. Need client credentials within the input request body to initialize Sagemaker Transcoder.
See Also	Relationships	
Version	Status	"1.0.0"

## /request Endpoint

Part	Purpose	Value
ID	ID	
Title	Name	request
Path	URL	/prod/auth/request
Method	OpType	POST
ShDesc	for SEO	To transcode data using a smart token and to refresh that smart token
Desc	Context	
Parameters	Config.	<p>To transcode data: smart_token (String); transcode_data (JSON array); rpc (String, default= request)</p> <p>To refresh the smart_token: client_id (String); client_secret (String ), and rpc (String, default = refresh)</p>
Request	Input	/prod/auth/request
cURL		<pre>curl --location --request POST 'https://&lt;hostname&gt;.execute-api.&lt;region&gt;.amazonaws.com/Prod/auth/request' \ --header 'Content-Type: application/json' \ --data-raw '{ "httpMethod": "POST", "client_id": "&lt;insert client id here&gt;", "client_secret": "&lt;insert client secret here&gt;", "rpc" : "refresh" }'  curl --location --request POST 'https://&lt;hostname&gt;.execute-api.&lt;region&gt;.amazonaws.com/Prod/auth/request' \ --header 'Content-Type: application/json' \ --data-raw '{ "httpMethod": "POST",</pre>

		<pre>"transcode_data" : [], "smart_token" : "&lt;insert smart token here", "rpc" : "request" }'</pre>
Response	Output	An array of transcoded links when RPC in the input request body is "request". A smart token when RPC in the input request body is "refresh".
Status & Errors	Output	<pre>{ "200": "Success", "400": "Bad Request", "401": "Not authenticated", "500": "Server Error" }</pre>
Tags	Category	
Status	Status	Production
Authentication Method	AAA	Yes. Need a smart token within the input request body to transcode the data. Need client credentials within the input request body to refresh the smart token.
See Also	Relationships	
Version	Status	"1.0.0"

## Errors

When working with the API you may sometimes receive an error code in the response. Refer to the following table for a description of errors you may encounter.

Code	Description
E000	Failed Transcoding for Unknown Reason...
E002	Unable to process transcode due to a security exception. Most likely a bad key or unauthorized transcode.
E003	Invalid argument size for batch transcoding
E004	Unsupported Transcoding Type for now 'Other to Native
E005	Unable to parse and validate SMaRT Token
E006	Unable to parse request body
E007	Invalid Transcoding type
E008	Transcode type not yet supported

I000	Unable to create RampID for a partner. Invalid Link Operation
I001	Error Decoding RampID
I002	Error Encoding RampID
S001	Not authorized for this procedure or invalid token
S002	Unable to create private key
S003	Unable to create public key
S004	Unable to decrypt via Private/Public Algorithm
S005	Unable to decrypt via Bulk Algorithm
S006	Unable to decrypt and load key packet
S007	Invalid or missing key for domain sent (value)
S008	Invalid or missing key for domain sent (target)
S009	Not Authorized to go from domain (value)
S010	Not Authorized to go to domain (target)
S011	Not able to validate access
S012	Not able to validate token

## See Also

[AWS Network isolation](#)

[AWS SageMaker documentation](#)

[Embedded Transcoder GitHub repository](#)

[LiveRamp's API Developer Hub](#)