tx-manager Documentation

Release

unfoldingWord

Jun 21, 2017

Contents:

master: develop: **NOTE: High level Architecture documentation is here:** *tX Architecture*.

CHAPTER 1

tx-manager

Project description at **'tX Manager Module'_**.

Issue for its creation at https://github.com/unfoldingWord-dev/door43.org/issues/53.

CHAPTER 2

tX Pipeline

1. Gogs
2. Webhook
3. Request Job
4. Start Job
5. [CONVERTER]
6. Callback
7. Door43 Deploy

CHAPTER 3

Definitions

The following placeholders are used in examples in this document:

- <repo> - the machine name of a Gog's repo. This is used in the URL for the repo, such as en-obs
- <user> - the user or organization that the Gog's repo belongs to, such as richmahn (user) or door43 (org)
- <commit> - The 10 character hash string that represents the commit (revision) that is being processed

CHAPTER 4

How it Works

Request Conversion Job

Using the Pipeline and the corresponding numbers above, this describes each part of the pipeline and how each are integrated, both with each other as well as the AWS Services that are used.

NOTE: This gives URLs and bucket names for test. For development, replace the *test-* prefix from domain or bucket name with *dev-*. For production, remove the *test-* prefix from domain or bucket name.

1. Gogs (Git website)

   When a repository is updated on Gogs, the commit triggers all webhooks in the repo's settings. One of those webhooks, which our copy of Gogs sets up automatically for every new repo, is a call to https://test-api.door43.org/client/webhook (API Gateway -> Lambda function).

2. Webhook (Lambda function - API Gateway triggered)

   The webhook triggered in Gogs (#1) sends the commit payload to the AWS API Gateway *client* stage and the *webhook* method which triggers the client_webhook Lambda function.

   The webhook function expects the following variables in the payload:

   - data - the commit payload from Gogs
   - api_url* - the base URL to the tX Manager API (e.g. https://test-api.door43.org)
   - pre_convert_bucket* - the S3 bucket in which to put the zip file of the files to be converted (e.g. tx-webhook)
   - cdn_bucket* - the S3 bucket in which the zip file of the converted files is to be found in client_callback (e.g. cdn.door43.org)
   - gogs_url* - the URL to the Gogs site to verify user token (e.g. https://git.door43.org)
   - gogs_user_token* - a user token of a valid user to prove they are a user so we can track job requests (for the webclient we just have one user token for all requests, given by the API Gateway)

these variables are set up in the 'client' Stage Variables <https://us-west-2.console.aws.amazon.com/apigateway/home?region=us-west-2#/apis/94c6v76xoh/stages/client>, so dev and prod gateways can have different variables

The client_webhook function is responsible for standardizing both a manifest.json file and the resource containers from all types of repos committed to Gogs, and it will call a preprocessor (e.g. TsObsMarkdownPreprocessor) to handle this. Converters (#4) expect the files to be converted to be in a flat-level zip file, where all files to be converted (with the input file extension) are one file per chapter (Bible, OBS) and in alphabetical order for logging and display purposes. Once the files are zipped up and the zip file put at https://test-cdn.door43.org/temp/<repo>/<commit>, the client webhook function requests a job by posting a request to https://test-api.door43.org/tx/job and exits.

3. Request Job (Lamdbda function - API Gateway triggered)

   Request Job is triggered through a call to the AWS API Gateway, running the request_job lambda function. This function expects the following variables in the payload:

   - gogs_url* - the URL to the Gogs site to verify user token (e.g. https://git.door43.org)
   - api_url* - the base URL to the tX Manager API (e.g. https://test-api.door43.org)
   - data - information about the job to performed. It contains the following variables:
     - gogs_user_token - a user token of a valid Gogs user
     - cdn_bucket - the S3 bucket in which the zip file of the converted files is to placed
     - source - The URL of the archive of files to convert (e.g. https://s3-us-west-2.amazonaws.com/test-tx-webhook/preconvert/0038b1d1-bf3b-11e6-8481-ed2b5603783b.zip)
     - resource_type - The resource type (e.g. obs, ulb, udb, etc.)
     - input_format - The input format of the files (e.g. md)
     - output_format - The desired output format (e.g. html)

   these variables are set up in the 'tx' Stage Variables , so dev and prod gateways can have different variables.

   From the above information, tX Manager's setup_job function will determine what converter to use for this job and will save this job request to the tx-job table. Inserting this job into the DynamoDB will trigger the tX Manager Start Job lambda function.

4. Start Job (a) (Lamdbda function - DynamoDB tx-job table insert triggered)

   The Start Job lambda function is triggered by a job being inserted into the DynamoDB tx-job table (Thus is NOT triggered through a call through the API. This is to separate the Request Job from the Start Job due to the 5 minute limit of execution time of a Lambda function)

   This function will load the given record from the DB and populate a TxJob object. It will then send this to the converter determined in #3 from its input and output formats. A call to the converter is then made.

5. [CONVERTER] (Lamdbda function - tX Manager triggered)

   Each converter is responsible for converting a given input file type to a given output file type. It also can have one or more resource types it converts. It expects the URL of a zip file which it then downloads and unzips. It then converts all the files to another zip file, converting the files of the given input type to the given output type, and copies all other files as they are to the new archive. It uploads the archive to the given S3 bucket and file path.

   It also can perform checks at this point if there any warnings or errors and return those in the JSON object returned to the Start Job function (#4)

4. Start Job (b) (Lamdbda function - Return from [CONVERTER] #5)

   Once the CONVERTER returns a status of warnings and errors (if any), the Start Job function calls the call back URL if one was given so the client can know the job was completed and if it was successful or not.

6. Callback (Lamdbda function - API Gateway triggered)

   When the callback function is called, the client looks to see if the job was a success and if it was, unzips the new archive and puts its contents in the test-cdn.door43.org bucket with the key prefix of *u/<user>/<repo>/<commit>*. It puts the status of the build into a file and uploads to the same bucket with the key *u/<user>/<repo>/<commit>/build_log.json*.

7. Deploy to Door43 (Lamdbda function - S3 modified file triggered)

   The uploading of build_log.json in #6 triggers the Door43 Deploy function.

   The Door43 Deploy function is what moves the HTML files converted by #5 and placed in the CDN bucket in #6 to live.door43.org and templates it based on the door43.org layouts. It also generates header, status and navigation portions of the pages for each revision.

Register Conversion Module

In order for tX Manager to know about a conversion module and to assign a conversion request to the module, it must be registered. To register a module, it must make a call to the API Gateway with the URL https://test-api.door43.org/tx/register. It expects the following variables:

- name - the Lambda function name of the converter, usually in the form of tx-<input>2<output>_convert
- type - the type of the module, usually “conversion”
- input_format - the input format accepted by the conversion, which is the extension of the file, such as “md”
- output_format - the output format of the files to be generated, which is the extension of the file, such as “html”
- resource_types - the resource type(s) accepted by the converter, such as “obs”

See tx-md2html_register Lambda function. for an example of a module registering itself.

CHAPTER 5

Setting up as deployed in virtual environment

In IntelliJ terminal, switch to virtual environment and install requirements.

CHAPTER 6

Deploying your branch of tx-manager to AWS

For developing the tx-manager library which this repo uses for every function, you can deploy your code to a test AWS environment with apex by doing the following:

- Copy project.test.json.sample to project.test.json
- Edit project.test.json and change <username> and <branch> to your tx-manager branch
- Install apex from http://apex.run/#installation
- Set up your AWS credentials as specified at http://apex.run/#aws-credentials
- Run *apex deploy –env test* to deploy all functions, or *apex deploy –env test [function-name]* for a single function

For more information on using –env to specify a project json file, see https://github.com/apex/apex/blob/master/docs/projects.md#multiple-environments

CHAPTER 7

License

CHAPTER 8

Indices and tables

- genindex
- modindex
- search