Charger - An Airtable to Stripe Autodebitor

Charger is a Buy More Time specific implementation of the charger program at <https://github.com/DnOberon/charger>. This implementation has the bare minimum of changes from the base program at time of writing.

This documentation runs over how the program works, deployment, and how to react to changes from external programs. This documentation isn’t meant to be all inclusive, it is still suggested you learn how the code works on a base level and experiment with the software if attempting to make any changes.

## How it Works

Charger a program written in Go and can be compiled to run on Windows, Mac or Linux. Currently no releases have been created - meaning you must compile from source or use the Go toolchain to run this program on your local development machine. Setting up this toolchain and environment is too extensive to be covered in this documentation - visit <https://golang.org/> for instructions and documentation.

This program runs on a loop, once started it will end either on irrecoverable error or until the user terminates the program manually.

Each loop this program does the following things.

1. Fetch all unpaid invoice records from the configured Airtable table. Invoices are determined “unpaid” if the PAID\_COLUMN’s value is not “true” or “t”
2. For each unpaid record, attempt to do the following
   1. Verify that the DATE\_COLUMN value indicates a date that’s either that day or in the past, if not, the program skips the record.
   2. Attempt to pull the customer’s payment methods from Stripe using the value from the STRIPE\_CUSTOMER\_ID\_COLUMN.
   3. Charge the customer’s payment method through Stripe
   4. Update the record in Airtable with the charge status (whether or not it failed and any messages with that failure)
3. Waits for X seconds and then starts the loop over again.

## Deployment

Charger has been designed to be as easy to launch an updated service as possible. This is a brief description of how continuous deployment of Charger works and a step by step on triggering a deployment.

Charger is run on AWS’s ECS (Elastic Container Service) system and utilizes the ECR (Elastic Container Registry) and Github Actions to handle automatic deployments. On deployment trigger, Github will upload a new Docker image of our application to ECR and then trigger its deployment to ECS. Github will also handle injection of encrypted secrets into our application as well as any errors that might occur.

It is recommended you become familiar with Github Actions and the task-definition.json file at the repository’s root.

Triggering a deployment

1. Navigate to the Charger repository - <https://github.com/Buy-More-Time/charger>
2. Navigate to the Actions heading.
3. Under the workflows header to the left, select the “Deploy to Amazon ECS” workflow.
4. Select “Run Workflow” on the table to the right
5. Monitor workflow for completion

## Handling Changes From External Services

At some point one of the bits of information that Charger relies on will change. Whether or not that’s the Airtable API information, the table’s information, or your Stripe information you need to be able to adapt to each. Because Charger comes with a completely deployed CI/CD system, reacting to changes should be simple. Here are the steps for reacting to each.

### Changes to Stripe API Key

1. Retrieve new API key - <https://stripe.com/docs/keys>
2. Navigate to the Charger repository - <https://github.com/Buy-More-Time/charger>
3. Navigate to Settings -> Secrets inside the Github Repository <https://docs.github.com/en/actions/configuring-and-managing-workflows/creating-and-storing-encrypted-secrets>
4. Modify the STRIPE\_API\_KEY secret, replacing the value with the key you retrieved in step 1
5. Deploy Application (see [Deployment)](#_nw0qb8io9fko)

### Changes to Airtable API Key OR BaseID

1. Retrieve new API key - <https://support.airtable.com/hc/en-us/articles/219046777-How-do-I-get-my-API-key->
2. Retrieve Airtable BASE ID by
   1. Navigating to https://airtable.com/api ,
   2. Selecting the Base to which the Airtable table belongs
   3. Retrieving the Base ID by extracting it from the Authentication sections’s CURL command
3. Navigate to the Charger repository - <https://github.com/Buy-More-Time/charger>
4. Navigate to Settings -> Secrets inside the Github Repository <https://docs.github.com/en/actions/configuring-and-managing-workflows/creating-and-storing-encrypted-secrets>
5. Modify the AIRTABLE\_API\_KEY or AIRTABLE\_BASE\_ID secret respectively, replacing the value with the key you retrieved in step 1 and step 2.
6. Deploy Application (see [Deployment)](#_nw0qb8io9fko)

### Changes to Airtable Column Names

1. Write down all new column names and which part they belong to (e.g the name of the Currency Code column)
2. Navigate to the Charger repository - <https://github.com/Buy-More-Time/charger>
3. Navigate to Settings -> Secrets inside the Github Repository <https://docs.github.com/en/actions/configuring-and-managing-workflows/creating-and-storing-encrypted-secrets>
4. The secret you will need to change corresponds to the column that’s changed inside Airtable. Below you’ll find the name of the secret followed by a description of the column it should correspond to. Once you have matched the column you need to change with the secret, enter the new column name in for that secret’s value
   1. | STRIPE\_CUSTOMER\_ID\_COLUMN |- Stripe Customer ID for the invoiced person or organization |
   2. | INVOICE\_AMOUNT\_COLUMN | - Invoice Amount |
   3. | CURRENCY\_CODE\_COLUMN | - Three Digit Currency Code (currently only `usd` is accepted)|
   4. | PAID\_COLUMN | - Either "true" or anything else. Indicates whether or not a record was paid |
   5. | NOTES\_COLUMN | Will record a payment reference number on success, or error information on issues |
   6. | DATE\_COLUMN | - Date on which the invoice should be charged |
5. Deploy Application (see [Deployment](#_nw0qb8io9fko))