Andrew Jones
Data Engineering @Grubhub
Mar 8, 2017 · 3 min read

Processing Email Attachments with AWS

Over the past several months, the Casper team has worked on pulling data managed by third parties into our AWS-based data infrastructure. While we have been fortunate to partner with many organizations who are able to provide an API or work with our SFTP server for the delivery of information, the only available delivery method for some partners is through email.

At first, the process of uploading attachments from emails to our data warehouse involved manually downloading each file and either uploading to an appropriate location in S3 or running a script against the file to complete the upload. Recently, we constructed a solution that allows us to automatically forward emails to S3 for our ETL system to process on its regular schedule. The rest of this post will describe how we set up our solution and the applications that have become available to us.

Configuring AWS

By leveraging AWS Workmail and Simple Email Service (SES), we are able to publish emails to an S3 bucket. We have setup a single email address in Workmail and distinguish between different types of emails by using email aliases with plus signs (“+”). For example, we will send a report from foo to emailreceiver@awsapps.com with the alias emailreceiver+foo@awsapps.com. Using this pattern allows us to focus on routing rather than spending additional time creating new Workmail accounts.

In SES, we use a rule set for inbound mail on our Workmail domain. When we want to add a new report to S3, we create a new rule for the report. Continuing with the previous example, our new rule will use the recipient emailreceiver+foo@awsapps.com and will add an S3 action. With the S3 action, we are able to route an email message to the bucket email-reports with an object key prefix of foo/. All reports that are routed to S3 by this rule will be stored at s3://email-reports/foo/ with a hash as the file name.

Parsing Attachments

When you pull an email file down from S3 and look at the contents, your initial reaction might be to close the file and never look at it again; a raw email is an unpleasant sight. Thankfully, the Python standard library provides the *email* module to allow us to crawl through an email without experiencing too much of a headache.

email provides the function *message_from_string()* which will read a raw email string and convert it into the type *email.message.Message*. *Message* is a tree-like structure that contains a hierarchy of *Message*s that describe the composition of an email (Figure 1). The object returned when running *message_from_string()* is the root of the *Message* tree. By using the *get_payload()* function, we can fetch a list of the children of the current *Message*.

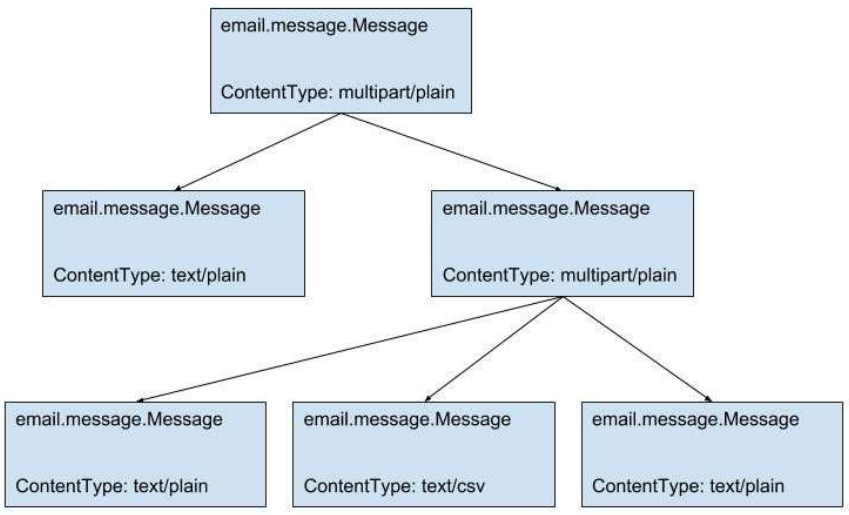

Figure 1. Illustrating an email Message hierarchy

We created the function *get_attachment()* to find the *Message* that holds our attachment. *get_attachment()* searches the *Message* hierarchy for a base64 encoded payload that matches a specified content type or is of type *text/plain*. The *get_attachment()* function can be viewed below.

```
def get_attachment(msg, content_type):
    """
    Moves through a tree of email Messages to find an atta

    :param msg: An email Message object containing an atta
    :param content_type: The type of attachment that is be
    :return: An email Message object containing base64 enc
    """
    attachment = None
    msg_content_type = msg.get_content_type()

    if ((msg_content_type == content_type or msg_content_t
            and is_base64(msg.get_payload()))):
        attachment = msg

    elif msg_content_type.startswith('multipart/'):
        for part in msg.get_payload():
            attachment = get_attachment(part, content_type
```

Results and Future Work

With this new approach, we save approximately 20 hours a month of an analyst's time, data is available within minutes rather than up to a week of delivery, and we can onboard new reports with minimal upfront work.

Up to this point, our email attachment processor has been used to extract CSV and Excel files from emails in order to store the data in our data warehouse. In the future, we hope to extend our usage to automate other manual work that originates from emails. For example, we have experimented with a set of PDF attachments that have to be merged together on a regular basis and have discussed ideas for improving our Customer Experience team's workflow in which they currently transfer knowledge from email to internal systems.

• • •

If you have questions about the implementation of this solution, are able to apply the formula to one of your problems, or just want to discuss Casper's tech, feel free to reach out! If you're looking for your next opportunity in NYC or Berlin, be sure to check out our jobs page.