Building an Email Archiving System

The Challenges and of Course the Solution

About a year ago I wrote a blog (https://www.sparkpost.com/blog/archiving-emails/) on how to retrieve copies of emails for archival and viewing but I did not broach the actual storing of the email or related data; and recently I wrote a blog on storing all of the event data (i.e. when the email was sent, opens, clicks bounces, unsubscribes, etc) on an email for the purpose of auditing (place link to blog that is targeted to be published in November, "building a robust email event archival system), but chose not to create any supporting code.

With the increase of email usage in regulatory environments I have decided it is time to start a new project that pulls all of this together with code samples on how to store the email body and all of its associated data. Over the next year I will continue to build on this project with the aim to create a working storage and viewing application for archived emails and all log information produced by SparkPost. SparkPost does not have a system that archives the email body but it does make building an archival platform fairly easy.

In this blog series I will describe the process I went through in order to store the email body onto S3 (Amazon's Simple Store Service) and all relevant log data in MySQL for easy cross referencing. Ultimately, this is the starting point for building an application that will allow for easy searching of archived emails, then displaying those emails along with the event (log) data. The code for this project can be found in the following github repository: https://github.com/jeff-goldstein/PHPArchivePlatform.

This first entry of the blog series is going to describe the challenge and lay out an architecture for the solution. The rest of the blogs will detail out portions of the solution along with code samples.

The first step in my process was to figure out how I was going to obtain a copy of the email sent to the original recipient. In order to obtain a copy of the email body, you need to either:

1. Capture the email body before sending the email
2. Get the email server to store a copy
3. Have the email server create a copy for you to store

If the email server is adding items like link tracking or open tracking, you can't use #1 because it won't reflect the open/click tracking changes.

That means that either the server has to store the email or some how offer a copy of that email to you for storage. Since SparkPost does not have storage mechanism for email bodies, but does have a way to create a copy of the email, we will have SparkPost send us a duplicate of the email for us to store in S3.

This is done by using SparkPost's *Archive* feature. SparkPost's *Archive* feature gives the sender the ability to tell SparkPost to send a duplicate of the email to one or more **email address** and use the same tracking and open links as the original. SparkPost documentation defines their *Archive* feature in the following manner:

> *Recipients in the archive list will receive an exact replica of the message that was sent to the RCPT TO address. In particular, any encoded links intended for the RCPT TO recipient will be identical in the archive messages*

The only differences from the RCPT TO email is some of the headers will be different since the the target address for the archiving email is different; but the body of the email will be an exact replica!

If you want a deeper explanation here is a link to the SparkPost documentation on creating duplicate (or *archive*) copies of an email.

As a side note, SparkPost actually allows you to send emails to *cc*, *bcc*, and *archive* email addresses. For this solution, we are focused on the *archive* addresses.

**** Notice * Archived emails can ONLY be created when injecting emails into SparkPost via SMTP!***

Now that we know how to obtain a copy of the original email, we need to look at the log data that is produced and some of the subtle nuances within that data. SparkPost tracks everything that happens on its servers and offers that information up to you in the form of message-events. Those events are stored on SparkPost for 10 days and can be pulled from the server via a RESTful api called message-events, or you can have SparkPost push those events to any number of collecting applications that you wish. The push mechanism is done through webhooks and is done in real time.

Currently there are 14 different events that may happen to an email. Here is a list of the current events:

Bounce	Click	Delay
Delivery	Generation Failure	Generation Rejection

Initial Open	Injection	Link Unsubscribe
List Unsubscribe	Open	Out of Band
Policy Rejection	Spam Complaint	

* Follow this link for an up to date reference guide for a description of each event along with the data that is shared for each event.

Each event has numerous fields that match the event type. Some fields like the *transmission_id* are found in every event, but other fields may be more event specific; for example, only *open* and *click* events have geotag information.

One very important message event entry to this project is the *transmission_id*. All the message event entries for the original email, archived email and any cc and bcc addresses will share the same transmission_id.

There is also a common entry called the *message_id* that will have the same id for each entry of the *original* email and the *archived* email. Any *cc* or *bcc* addresses will have their own id for the *message_id* entry.

So far this sounds great and frankly fairly easy, but now is the challenging part. Remember, in order to get the archive email, we have SparkPost send a duplicate of the original email to another email address which corresponds to some inbox that you have access to. But in order to automate this solution and store the email body, I'm going to use another feature of SparkPost's called *Inbound Email Relaying*. What that does, is take all emails sent to a specific domain and process them. By processing them, it rips the email apart and creates a json structure which is then delivered to an application via a webhook. See Appendix A for a sample json.

If you look real carefully, you will notice that the json structure from the *inbound relay* is missing a very important field; the *transmission_id.* While all of the outbound emails have the *transmission_id* with the same entry which binds all of the data from the *original email*, *archive*, *cc* and *bcc* addresses; SparkPost has no way to know that the email captured by the inbound process is connected to any of the outbound emails. The inbound process simply knows that an email was sent to a specific domain and to parse the email. That's it. It will treat any email sent to that domain the same way, be it a reply from a customer or the archive email send from SparkPost.

So the trick is; how do you glue the outbound data to the inbound process that just grabbed the archived version of the email? What I decided to do is to hide a unique id in the body of the email. How this is done is up to you, but I simply created an input field with the hidden tag turned on.

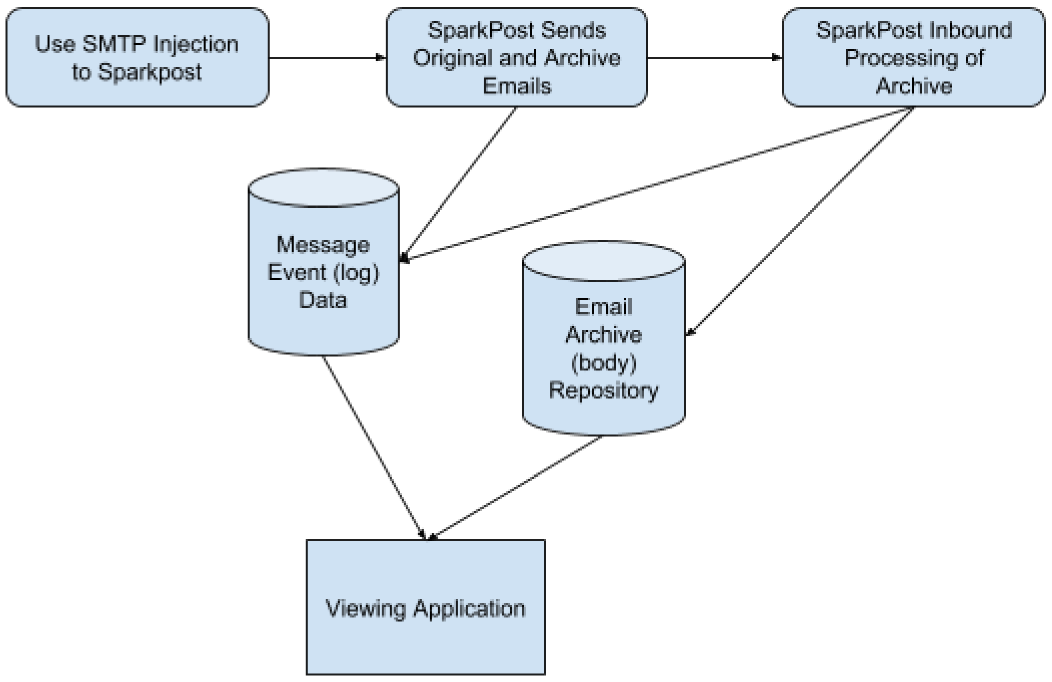
```
<input name="ArchiveCode" type="hidden" value="<<UID>>">
```

I also added that field into the meta data block of the X-MSYS-API header which is passed to SparkPost during injection. This hidden UID will end up being the glue to the whole process, and is a main component of the project and will be discussed in depth in the following blog posts.

Now that we have the UID that will glue this project together and understand why it's necessary, I can start to build the vision of the overall project and corresponding blog posts.

1. Capturing and storing the archive email along with a database entry for searching/indexing
2. Capture all message event data
3. Create an application to view the email and all corresponding data

Here is a simple diagram of the project:


Use SMTP Injection to Sparkpost
SparkPost Sends Original and Archive Emails
SparkPost Inbound Processing of Archive
Message Event (log) Data
Email Archive (body) Repository
Viewing Application

The first drop of code will cover the archive process and storing the email onto S3, while the second code drop will cover storing all of the log data from message-events into MySQL. You can expect the first two code drops and blog entries sometime in early 2019. If you have any questions or suggestions, please feel free to pass them along.

Happy Sending.

Appendix A:

```
[{
"msys": {
  "relay_message": {
    "rcpt_to": "archive@geekwithapersonality.com",
    "friendly_from": "bumpy@mail.geekwithapersonality.com",
    "customer_id": "122",
    "content": {
      "subject": "test Tue, 04 Dec 2018 17:28:12 -0800",
      "html": <html_body>,
      "headers": [{
        "Return-Path": "<msprvs1=17877Mi_gE5A3=bounces-122@sparkpostmail.com>"
      }, {
        "Authentication-Results": "b.mta1vsmtp.cc.prd.sparkpost…."
      }, {
        "Received": "from [10.92.21.198] ([10.92.21.198:26352] helo=mta…."
      }, {
        "DKIM-Signature": "v=1; a=rsa-sha256; c=relaxed\/relaxed; d=mail.geekwitha; …."
      }, {
        "X-MSFBL": "iGIF1akiy0VZT34sRlvFVNKusa4IkwzCcN…."
      }, {
        "Authentication-Results": "a.mta1vsmtp.cc.prd.sparkpost smtp.user=<hidden>; …."
      }, {
        "Received": "from [73.189.53.88] ([73.189.53.88:59789] helo=jeff-...."
      }, {
        "Date": "Tue, 04 Dec 2018 17:28:12 -0800"
      }, {
        "To": "jeff.goldstein@sparkpost.com"
      }, {
        "From": "bumpy@mail.geekwithapersonality.com"
      }, {
        "Subject": "test Tue, 04 Dec 2018 17:28:12 -0800"
      }, {
        "Message-Id": "<20181204172812.046850@jeff-gs-macbook-pro.local>"
      }, {
        "X-Mailer": "swaks v20170101.0 jetmore.org\/john\/code\/swaks\/"
      }, {
        "MIME-Version": "1.0"
      }, {
        "Content-Type": "text\/html"
      }, {
        "Cc": "austein@hotmail.com"
      }, {
        "List-Unsubscribe": "<mailto:unsubscribe@unsub.spmta.com?subject=unsubscribe:..."
      }, {
        "List-Id": "<spc-122-0>"
      }],
      "email_rfc822": <rfc822 compliant email>,
      "email_rfc822_is_base64": false,
      "to": ["jeff.goldstein@sparkpost.com"],
      "cc": ["austein@hotmail.com"]
    },
    "msg_from": "msprvs1=17877Mi_gE5A3=bounces-122@sparkpostmail.com",
    "webhook_id": "66600222946105557",
    "protocol": "smtp"
  }
}
}]
```