Article title: Serverless Architecture For Bulk Email Management

Authors: Priyatham Sai Chand Bazaru[1]

Affiliations: Mahatma Gandhi Institute of Technology[1]

Orcid ids: 0000-0002-7764-626X[1]

Contact e-mail: priyathamsaichand@gmail.com

License information: This work has been published open access under Creative Commons Attribution License <http://creativecommons.org/licenses/by/4.0/>, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. Conditions, terms of use and publishing policy can be found at <https://www.scienceopen.com/>.

Preprint statement: This article is a preprint and has not been peer-reviewed, under consideration and submitted to ScienceOpen Preprints for open peer review.

DOI: 10.14293/S2199-1006.1.SOR-.PPTDBKS.v1

Preprint first posted online: 12 December 2021

Keywords: Cloud Computing, Serverless, Email, Distributed Systems, Computer Science

SERVERLESS ARCHITECTURE FOR BULK EMAIL MANAGEMENT

A PREPRINT

© B. Priyatham Sai Chand

Student

Dept of Computer Science

Mahatma Gandhi Institute of Technology

priyathamsaichand@gmail.com

December 8, 2021

ABSTRACT

Email sending is an important tool to broadcast messages to many people to convey information. bulk emails in magnitude of thousands of emails is needed for large organization or company reaching out to its customers. Creating and maintaining a server to accomplish this is not only time-taking but not cost-effective. Products such as Gmail and Outlook are useful in basic emailing services but fail to be useful in larger volumes of emails to target people or spread information to consented individuals. This paper tries to achieve bulk email management through the use of serverless architecture to send emails through a service from the Amazon Web Services(AWS) ecosystem.

Keywords serverless · bulk emails · broadcast

I Introduction

Email has become one of the most important means of communications. Sending emails through conventional software like Gmail is limited to a mere thousands which is not sufficient to fit into the business needs and managing the insights of sent emails is crucial to companies to further enhance their approach.

Serverless computing is any computing platform that hides server usage from developers and runs code on-demand automatically scaled and billed only for the time the code is running[1]. This reduces the time and knowledge required to deploy pieces of software while offering significant reduction in pricing compared to hosting the software on dedicated hardware. Various cloud platforms such as AWS, Microsoft Azure and Google Cloud Platform have adapted to this way of thinking making it easily accessible. A robust serverless architecture for sending bulk emails of the magnitude of thousands to millions using scalable model which is implemented using AWS lambda[2] and Simple Email Service(SES)[3] for the SMTP protocol. Challenges such as able to trace the bounce emails as well as the statistics for the emails sent. The prototype is built using the React framework along with underlying services from AWS. These tightly integrated services offer the best way to simulate the architecture and help in sending customized for every recipient based on email templates along with storing bounce emails.

The effectiveness of the scalability offered by the architecture is analyzed using X-ray traces to simulate and test the bottlenecks and how to overcome them is discussed in the paper. With serverless technology, the cloud provider abstracts away the server management, provisioning servers with fine granularity, on demand, and with a pay-per-use model[4].

This paper proposes an architecture for sending and management of the emails through serverless application using the AWS services. The paper outlines the accuracy and the reliability of the design and future aspects for this architecture.

II Overview of bulk email management system

The bulk email management system is used to send emails of magnitude in thousands and more and broadcast the message to multiple email addresses with a single operation. All the required fields such as the senders' email addresses, subject, the html body and to addresses are selected and the sending is initiated with a button. Upon successful sending to the API endpoint the emails are processed and send through the AWS simple email services to all the users and the emails that were bounced are returned after the sending operation. It makes use of a Javascript based framework called ReactJS to input the fields and send it to the API hosted on the AWS service where an array of services are used to scale and perform the sending and retrieving of bounces.

Figure 1: Architecture for bulk email management

III Architecture Design

The architecture is demonstrated in the Fig 1 with the main functionalities include sending the emails and tracking the bounces. Initially, all the data needed is sent to the lambda using the API gateway which handles the request and response for the whole operation. Upon reaching the lambda JSON objects are created from the string parameters which are preprocessed, the html part of the email along with the subject are made into SES templates which can be accessed by other lambdas and the emails batched no more than 50 emails per batch and placed in a Simple queue Service[5] queue. The ordering of the messages is important hence, we use a First In First Out queue, which triggers another lambda that sends emails to all the 50 recipients using a templated email function by accessing the template that we created earlier. We set a flag to mark the end of the batches upon which the template is deleted by the end of the execution.

The emails are sent using a configuration set that triggers events such as bounce, complaints or seen to other services for logging and other analysis. The configuration set is set to send all the bounce receipts to SNS which triggers a lambda function that stores the receipts useful information as a document in a NoSQL database such as dynamoDB once all the emails are sent and all bounce receipts are stored in the database. The origin lambda retrieves all the records and sends the sent emails and bounced emails as

response to API gateway which forwards it the application which is shown to the user.

The emails that are bounced are identified by the configuration set and are sent to the SNS with email address and the type of bounce which invokes a lambda trigger that stores the Simple Notification Service(SNS)[6]notifications after filtering the message into the dynamo DB.

All the bounced emails are automatically updated to a email suppression list which saves the names of bounced emails and accepts them but doesn't send the emails after it is declared bounce once. Thereby, saving the bounce rate of the service.

Additionally, All the other statistics are sent to Cloudwatch metrics which can be used to analysis the statistics of sending emails at different time frames and track the bounce rate and can be useful in finding out the effectiveness of the campaign by the number of people who opened the email.

The architecture keeping in mind the constraints on SES service. The advantages of this is two folds. It helps to work within the limits of the service. Moreover, these constraints can be fine tuned to deal with the limits of SMTP server if an independent server or other services are used.

IV Constraints

The architecture is based a few constraints from the SES service and other bottlenecks that affect the sending of emails. The SES through which the emails are initially sent through sandbox to test and later needs to be upgraded by the AWS authority to permit the user to request increase in the number of emails to be sent. For the purpose of this architecture 50,000 emails have been requested which was granted by the authority.

- 14 emails/second
- 50,000 emails per day
- request data 5kb

There are two performance monitoring services available to AWS application developers: CloudWatch and X-ray.

V Analysis

Basic message transfer from Author to Recipients is accomplished by using an asynchronous store-and-forward communication infrastructure in a sequence of independent transmissions through some number of mail transfer agents[7]. Two of the most common ways to analyze the architecture was to consider the time required to send all the emails and the cost associated with it. Both of the metrics are analysed in the following section.

VA Performance

There are two performance monitoring services available to AWS application developers: CloudWatch and X-ray.[8] The application can be analyzed to see the time required to send the emails and the time taken to run the functions between the request and response to the API endpoint. This can be seen through the cloudwatch[9] and X-ray.X-Ray is a monitoring and display service that automatically samples the entry and exit of function instances, called segments, using unique trace identifiers (`trace_id`)[8][10].

VB Cost

The cost of the services can also be a motivating factor to implement such an architecture. In our use case The following architecture can be tested with the AWS free tier and later expanded to the needs of the users. With \$0.01 dollars per 1,000 emails but one needs to be wary about the associated services like the SES, SNS and the cloudwatch metrics to analyze the emails add up to cost but since they are calculated based on the GB of data it needs to pre-process the rates will not exceed the cost of emails.

Here we take the example, provided by the SES platform to estimate the cost required to implement the architecture. You use Amazon SES to send about 250,000 emails per month. You receive 1,000 emails per month. You don't use dedicated IP addresses. Every message you send and

receive is 32KB in size which results in a total of \$25.98 per month [11] which is significantly less than competitors such as SendGrid or MailChimp who offer their own SMTP server or schedule emails to fit into the constraints of other providers to carry out email campaigns.

Amazon provides sample pricing calculations, but as your workload varies, so will the billing.[12] The SQS requests can exceed the free tier if not monitored carefully and add up to additional costs for the next 1000 requests or more based on the usage of the system.

VI Conclusion

To conclude, the architecture has been effective in sending bulk emails of various volumes with runtime that can be considered as sufficient or optimal and cost-effective. The primary goal of the architecture to be able to manage the entire system without a server being provisioned directly i.e. serverless is achieved and is promising to further develop and enhance its capabilities. The statistics displayed can be used for reach through the customers and the key idea of the architecture is to make use of serverless functions and services to manage emails with an optional fronted interface which was the case in this paper.

References

- [1] Paul Castro et al. "The rise of serverless computing". In: *Communications of the ACM* 62.12 (2019), pp. 44–54.
- [2] *AWS lambda*. URL: <https://aws.amazon.com/lambda/> (visited on 11/22/2021).
- [3] *Amazon Simple Email Service*. URL: <https://aws.amazon.com/ses/> (visited on 11/22/2021).
- [4] Simon Eismann et al. "Serverless applications: Why, when, and how?" In: *IEEE Software* 38.1 (2020), pp. 32–39.
- [5] *Amazon Simple Queue Service*. URL: <https://aws.amazon.com/sqs/> (visited on 11/22/2021).
- [6] *Amazon Simple Notification Service*. URL: <https://aws.amazon.com/sns/> (visited on 11/22/2021).
- [7] Dave Crocker. *Internet Mail Architecture*. RFC 5598. July 2009. DOI: 10.17487/RFC5598. URL: <https://rfc-editor.org/rfc/rfc5598.txt>.
- [8] Wei-Tsung Lin et al. "Tracking causal order in AWS lambda applications". In: *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 50–60.
- [9] *Amazon CloudWatch*. URL: <https://aws.amazon.com/cloudwatch/> (visited on 11/22/2021).
- [10] *Amazon X-Ray*. URL: <https://aws.amazon.com/xray/> (visited on 11/22/2021).
- [11] *Amazon SES Pricing*. URL: <https://aws.amazon.com/ses/pricing/> (visited on 11/23/2021).

-
- [12] Adam Eivy and Joe Weinman. “Be wary of the economics of” Serverless” Cloud Computing”. In: *IEEE Cloud Computing* 4.2 (2017), pp. 6–12.