

Capstone Project: Customer Feedback Application

Gathering Customer Feedback

Final Capstone Report

Daniel Kushner

CST 499 Directed Group Capstone

California State University, Monterey Bay

October 1, 2019

Executive Summary

The world is moving towards cloud based software solutions for their technology needs and the transition to the cloud has resulted in customer experience teams becoming a core focal point of getting businesses to buy in and a centerpiece of building the trust and transparency required to migrate to the cloud. These customer experience teams require more frequent and better customer feedback than ever before. But gathering that feedback across a variety of customers using various components of distributed systems in the cloud has become more complicated and costly than ever before, demanding a solution that can provide flexibility, privacy, and rapid customization at a cost effective price. These demands are what the customer feedback application aims to address.

These higher demands for customer feedback can be met with the right combination of resources that are either free and open source or following a pay for what you use model. The weight of hosting these resources can be relieved by using serverless technology. The flexibility and customization can be provided by using component driven libraries that has been open sourced. And the cost effectiveness can be achieved by using cloud providers that only charge for the resources consumed. The solution provided by the customer feedback application addresses this by using Amazon Web Services and React to deliver a customer experience tool that can be configured and deployed easily with minimal programming knowledge and experience, with any monetary cost coming into play only as your business grows.

Table of Contents

[Introduction](#)

[The Problem/Issue in Technology](#)

[Project Objectives](#)

[Project Goals](#)

[Community and Stakeholders](#)

[Evidence the Project is Needed](#)

[Functional Decomposition](#)

[Selection of design criterion](#)

[Final Deliverables](#)

[Approach/Methodology](#)

[Legal and Ethical Considerations](#)

[Budget, Timeline and Milestones](#)

[Usability Testing/Evaluation](#)

[Final Implementation](#)

[Conclusion](#)

[References](#)

Introduction

All businesses, regardless of whether they are service based or product based require customers to survive. Determining how to best serve and continue to serve those customers can be a daunting task due to both competition and customer satisfaction. The sections below will aim to help address the latter issue of customer satisfaction. The problem and solution will be outlined as well as the goals, objectives and outcomes. It will also present evidence that customer feedback is needed and discuss its impact on stakeholders.

The Problem/Issue in Technology

There are several major issues with gathering customer feedback in today's market. First is deciding how to gather customer feedback. With the internet allowing any business to reach global audiences, sending out individuals to survey clientele is no longer feasible. How does one get the feedback they need and how can it be presented in a form that is usable for making business decisions? How does one customize the feedback to conform to their businesses look, feel and product requirements? This leads to the next major issue with gathering customer feedback in today's market, what will it cost in terms of money, as well as manpower, to deploy software for gathering and visualizing customerfeedback? In today's market, with most software moving to a subscription based software as a service model, cost can be prohibitive to many businesses. Especially since many subscription based pricing models focus on volume rather than simply direct access to a set of software.

The customer feedback application will address each of these issues in multiple ways. For simplicity of self hosting and gathering the data, the stakeholder will be able to deploy the application using AWS web services UI, lowering the technical skill and manpower requirements required to use the application. Keeping with the idea of simplicity They can then customize the form and theming via JSON instead of needing to write code. Both of these help reduce manpower cost, but monetary cost is further reduced because Amazon web services uses a pay for what you use model. Visualizing and using the results is also made easy, as AWS dynamoDB allows for data to be easily downloaded as a csv file that users with basic Microsoft Excel skills can use to aggregate the data and present useful insights.

Project Objectives

- Develop an initial generic widget for gathering feedback (Completed Week 1)
- Create easy deployments using infrastructure as code (Completed week 1)
- Write clear documentation showing the application architecture and how to continue custom development of the application (Week 5)
- Create a tutorial for the non-technical user showing how to deploy the application, attach it to a web page and retrieve customer feedback data. (Week 5)
- Test the application with at least 3 individuals or individual groups (Weeks 4-7)

Project Goals

- Allow business owners to easily deploy a feedback widget to a webpage for gathering direct feedback

- Make the widget an affordable option compared to competitors by following a pay only for what you use model
- Create the widget to provide a smooth and easy user experience for the customer
- Ensure the feedback is stored in a secure location
- Prevent basic spamming and abuse, so customer data isn't compromised (i.e. you can't submit the widget over and over)
- Store the data in a way that is easy for non-technical personnel to easily export into common tools such as Excel.

Community and Stakeholders

The Stakeholders are the business/product owner, their customers, and the partners they work with. For the product owners and business stakeholders, their ability to gather and address customer feedback is critical to customer satisfaction and retention. Gathering customer feedback with a pay for what you use model is also critical for helping keep pricing down, allowing for potentially greater profits and helps to keep up or even undercut the competition. It's impact to the customer is just as significant, as it allows the customer to not only be heard by the product owners, but can result in more rapidly improving the user experience. Providing customers who are dissatisfied with a private outlet to send feedback may also help protect the business's public image. If the customer feels satisfied they are heard via the provided feedback widget, they may be less inclined to post negative feedback on Yelp, Google Reviews or other social media.

Evidence the Project is Needed

Petra Martišková and Roman Švec's conducted research into gathering customer feedback and found that "About 60% of customers give their feedback to an e-shop where these customers have made a purchase" (Littera Scripta, 2017), demonstrating customers actively looking to provide feedback that businesses can use to improve user experiences. But they are far from the only ones to conduct research showing the need for gathering customer feedback. In a 2011 article on Forbes.com from the Young Entrepreneurs Council states there are three good reasons to gather customer feedback. Those reasons are "Learn what your customers like and don't like", "Make customers feel important and involved", and "Constantly improve". It's clear that gathering customer feedback is a necessity in today's global economy.

Feasibility

The available options for gathering customer feedback has numerous players, the largest being Zendesk and Freshdesk. These all follow subscription based models at a flat rate, regardless of how much various features provided are used by the subscriber or their customers. This subscription based model is broken into pricing tiers and often has an enormous jump in price across each tier, resulting in the cost of the product potentially prohibiting smaller businesses and nonprofits from utilizing their product to gather customer feedback. A great step first step towards providing businesses the ability to gather customer is providing a pay for what you use model, which this capstone does by utilizing Amazing Web Services (AWS), whose business model revolves around paying for what you use. Even if a business can afford to use one of the subscription based services to gather customer feedback,

they are bound by several serious limitations that the design elements of this capstone try to address.

Perhaps the greatest advantage that this application seeks to provide, beyond the simple pay as you go model, is the ease of use and minimal maintenance it requires due to the creation of serverless technologies. With other customer feedback options, you have to set up an application to send and receive the data which requires maintenance, upgrades, and often reworking of code and expansion of hardware to scale as the company grows. Using AWS Lambda, AWS DynamoDB and AWS ApiGateway, this application is entirely serverless, allowing it to scale indefinitely without having to maintain data on hardware. It further eases the use of the product, because it's not only serverless, but can be repeatedly deployed thanks to advancements in infrastructure as code, which in this case is done with AWS CloudFormation.

The second limitation is customization of the form and the feedback widget. Zendesk's support widget and its documentation (Zendesk.com) requires one form per widget to be created and if they are used on the same page, there's no way to sort out the data collection. To address this, the choice was made to use Mozilla's react-jsonschema-form library, which allows you to define separate schemas for the form, form data and the UI and theming separately. This means I can customize each widget used and could additionally reuse components from existing schemas to create an entirely new widget instead of building it from scratch.

The third issue that separates this capstone from alternatives is what kinds of data customers can collect with their feedback widget, in particular what kind of meta-data they can gather. When using the Zendesk application, it only gathers the data directly placed in the form, and provides no expanded options for adding meta-data about the user that may be

available, such as information tied to their user account. This could be information like a username or id, email address or maybe what customer organization this user represents. To address this issue, a data object has been added to the javascript and database tables to allow them to customize this. Currently, they can change what exists within this by adding some custom javascript to their webpage, with future enhancements that will allow the extra data to be defined in a json schema similar to the out of the box features provided by the above mentioned react library.

Another limitation this capstone seeks to do better than competitors is the customization of the theme for the widget. Zendesk states “For more advanced customizations, you can enter in your own custom CSS” ([Knowledgebase.proprofs.com](https://knowledgebase.proprofs.com)), it requires the user to write all their own css. A core improvement that comes from the Mozilla react library allows for a series of default customized themes that can be selected out of the box as well as easy overrides for each of those. No code, no css, just specify the options in json schema files. If the user needs to expand it further, they can design or download a theme and easily tied it into the Mozilla, where it too can be used over and over via the json schema files.

Functional Decomposition

The functionality of the feedback for the user is simple and easy. A widget appears when they visit a website page. The user can then input feedback about their experience and press send. The functionality itself is slightly more complicated but very easy to deploy. To deploy the application a user needs to have an AWS account and a url to the cloudformation template. In the cloudformation console they just click deploy from that url and the infrastructure will set itself up. Once the application is deployed they need to add it to their website. To do this the

user can visit the AWS Apigateway console and copy the url and path for the GET request and add it to their website. After which the widget will load. The remaining major functions of the design are described below by figure 1 and the bulleted list that follows.

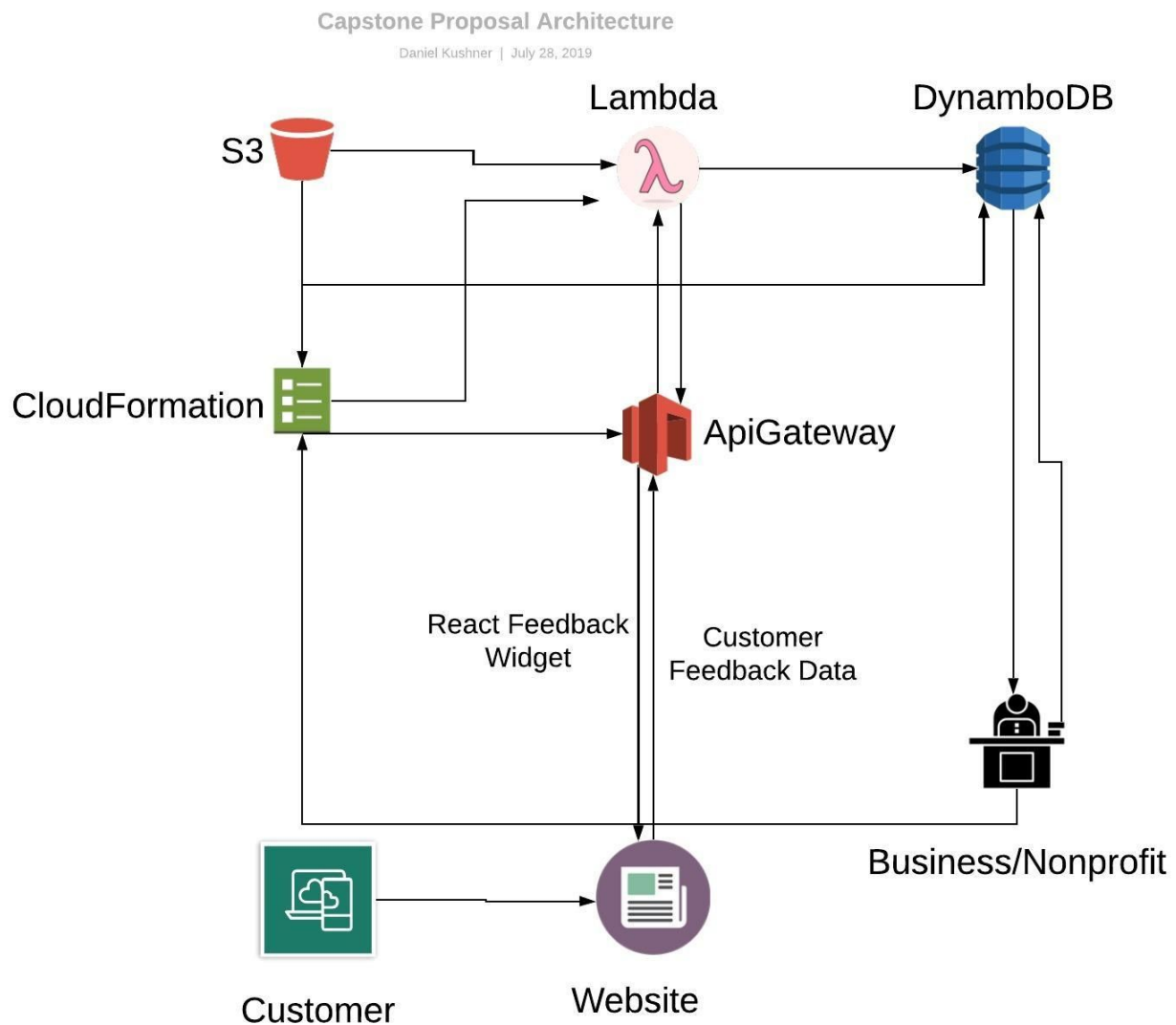


Figure 1

1. Customers visits the website and the application is loaded from the s3 bucket generated by the cloudformation template.
2. The customer submits feedback.

3. The data from the customer submission is then passed on Apigateway
4. Apigateway calls the appropriate Lambda function
5. The lambda function then stores the data in Dynamodb
6. The business can then download the data from Dynamodb or query it from other applications.

Selection of design criterion

The primary architecture was chosen due to its performance, reliability, and scalability. Since it's an AWS service reliability is managed by Amazon. The choice to use serverless technology means the performance and scalability is virtually infinite, as you do not need to worry about hardware failures or hardware capacity. The performance of the code is a non-factor because a lambda function will execute it separately each time the function is called. This means users are never sharing the same instance of executed code. For the purposes of cost, only always free options were considered and not the one year of free options AWS allows. Below is a list of components and their cost.

Component	Free Tier	
AWS Lambda	1st Million executions free	\$0.20 Per 1 Million Requests
AWS DynamoDB	25 GB Free -	\$1.25 Per million writes \$0.25 Per million reads \$0.25 Per GB storage
AWS CloudFormation	Always free	Always free
AWS S3	none	None as they are not hosting the cloudformation template themselves
AWS Apigateway	none	\$3.50 per million per month

React-JSON-Sc hema	Open source provided by Mozilla	
Total Cost	No more than \$3.50 to start	

Final Deliverables

The final deliverables to stakeholders is a live customer feedback widget on their website deployed from their own aws accounts. Hopefully this means customers are using it and sending in feedback, that the stakeholder is retrieving the data for analysis. As the application will be in active use, it means good documentation and easy deployment is a core part of the final deliverable. For academic purposes since the application will be deployed in their own accounts and websites and contain private customer data, final deliverables include a demo website with the widget and the infrastructure deployed in an independent aws account, as well as the code for the project in a public GitHub repository. The data contained in the independent aws account for demo purposes will be the data gathered during usability testing.

Approach/Methodology

The approach to building the application was split between the infrastructure components and the application. The base infrastructure was completed using an AWS Cloudformation template to define and deploy AWS Lambda, Dynambdo, and Apigateway. The remainder of the project then revolved around building the feedback widget itself. The first step was to simply made a small square widget that successfully loaded onto a page. Afterwards the application progressed as follows:

1. Add functionality that allows the widget to expand/collapse
2. Create a basic form that loads inside the widget
3. Add a submit button

4. Configure webpack to package images and other objects into a single js file for delivery
5. Construct functionality that sends the data to the Apigateway
6. Write javascript to demonstrate the extra data object for gathering meta-data
7. Verify the data is received and can be retrieved
8. Stakeholder testing and feedback.

Legal and Ethical Considerations

There is little legal concern as the project is using all open source or publically available resources. For legal, ethical, and security purposes, the application was built using best practices such as escaping data to ensure SQL injection and other common attacks were not likely to impact the use of the widget. The ethical considerations are obviously largely left open to the stakeholders and revolve mostly around what data is gathered and how that data is handled. The application itself used secure means to transmit the data and secure where the data is being deployed. This does not however, account for stakeholders properly protecting their own AWS accounts, which should be setup using multi-factor authentication.

The ethical concerns for what data is gathered and how it's handled is far more complicated. There is no single solution, but a common approach should be taken by all stakeholders. Per Meridith Powell, an expert on business growth, "Transparency is key" and "Trust is everything" (meridithelliotpowell.com). This means informing them what data they gather, how they use that data and what third parties may have access to their data. It should also include notifying customers when those change as well as any loss or exposure of data, and ideally the time in which their data will be kept.

Budget, Timeline and Milestones

The budget was of no concern on this project as all testing did not exceed the free tier provided by Amazon Web Services. The milestones and their statuses are listed on the table below.

Timeline	Milestone	Milestone met/not met
Week 1	Basic serverless infrastructure can be deployed via cloudformation	Completed on time without issue
Week 2	Build a customer feedback widget that can load into a page	This took an extended time to complete due to issues with CORS
Week 3	The widget can display a form with images, etc and be delivered via a single javascript file	Done, but the dynamic form component was simplified for MVP per stakeholder request
Week 4	The user can submit the data	Completed without issue
Week 5	The data is properly stored and can be retrieved by the stakeholder	Completed without issue
Week 6	User Acceptance Testing	Resulted in a single form and theme for the primary stakeholder until animations could be added to the dynamic json form building library at a later date.

Usability Testing/Evaluation

Usability testing was conducted with three separate groups of stakeholders from two different organizations using their own Amazon Web Services accounts. A demo site was provided for the smaller stakeholder and the other two stakeholders used their own development website for evaluation of the feedback widget. The testing itself consisted of the following basic actions:

1. Deploy the infrastructure for the widget in Amazon Web Services

2. Add the widget to a demo website
3. Use the widget itself
4. Extract the collected data from the database

The first step, deploying the infrastructure ran into no issues with stakeholders. Each group was able to successfully navigate using the AWS console to Cloudformation, find the template and deploy it. The only direct feedback here was a desire to have parameters for configuring the widget placed on the cloudformation screen, rather than being defined in a separate json file, so they would not have to edit multiple files stored in multiple places.

The second step, adding the widget to the demo website did not go as smooth. Two of the three had CORS (cross-origin resource sharing) issues, one of which was able to be immediately resolved, but the other I was forced to step in and help, as they lacked the technical skill to adjust this. Future enhancements will now include documentation on how to address this issue. Once the link was added, the pages had no trouble loading the widget as expected, however it was not as dynamic as expected either. The current form of the application relies on the form for the widget being defined from the server side. This resulted in the second future enhancement, redesigning the AWS Lambda function that serves the widget to allow the option of the configuration JSON being passed in the HTTP request instead of reading it from a stored JSON file.

The third step was perhaps the most engaging part, as stakeholders were most excited to actually use the part of the product customers themselves will use. Stakeholders were happy with the demo widget that was provided, and did not seem to really want or need the dynamic theming abilities provided by the React JSON schema library, as the colors and themes

for the businesses did not change often, nor where those changes made quickly or within the purview of the testers. The only part of the JSON schema they did want was dynamically changing what the form in the widget displayed, but become a secondary concern as they felt it was too difficult to use until there was an easier way to deploy multiple versions of the widget from the same infrastructure, as was mentioned in step two of the testing above. In step two, it was determined that the JSON needed to be passed as an HTTP request, but an alternative presented by a stakeholder was to continue using predefined widgets or JSON files for the widget and pass a parameter to tell the Lambda function which widget configuration should be returned. As a fun add on to the widget, a tiny bit of animation was added to transition the form submission from the form, to a thank you message. The stakeholders really enjoyed it and wanted to be able to select from a series of them in the same way the JSON could defined the widget and theming, however this was beyond the basic scope and timeline to deliver a minimum viable product and must be another future enhancement.

The final step of testing the product was that the stakeholders were able to retrieve the data so it can be used for analysis. There was zero issues here, as all stakeholders were able to go to the AWS DynamoDB table and download the data as a CSV. There is currently one major flaw that will need to be addressed in a future enhancement. DynamoDB has a limit to the amount of data that can be pulled into a csv file at once when using the AWS console UI, and therefore the user has to filter it down from the UI instead of once it's downloaded and in excel or inside another tool. There were requests for perhaps putting the data into something for analytics such as google analytics, and ELK (Elasticsearch, Logstash, Kibana) stack, or adding a Lambda function for returning the desired results. There was no general consensus on how to

resolve this issue, so there is currently no planned future enhancement to address this problem at this point in time.

Additional testing was also done by writing unit tests. The unit tests were written using the test framework jest. These tests cover the functionality of the react components and their states as well as the functions inside the lambda functions primary javascript file. It does not cover testing of the infrastructure or the deployment process.

Final Implementation

The final implementation went essentially as planned but had one major for one of the stakeholders, which was to remove the json schema for the time being until there was sufficient time to extend it to include animations from an animation library that must first be developed.

The final implementation consists of two primary pieces, the client side and the server side. The client side being react libraries with the widget itself and the server side being the serverless infrastructure that would load the widget and store the transmitted data.

The client side library consists of css, images, fonts and components as well as a series of web pack configurations for different development environments (development and production environments). The css, images and fonts are loaded into the components, which is broken into four categories. The radio buttons, the submit button, the feedback component and the feedback comment component. These are just elements on the widget itself. In figure 2 below, the user is first asked if they found what they were looking for and given a happy and frowning face to select from.

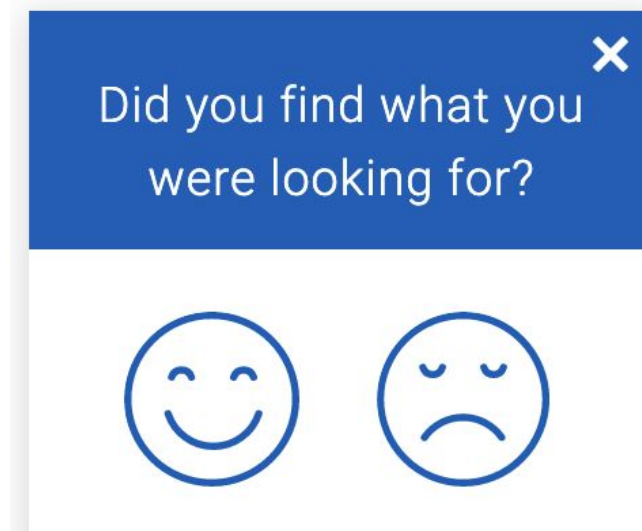


Figure 2

If the user selects the happy face it shows a thank you message and submits a generic statement saying the user is satisfied with the experience, per figure 3 below

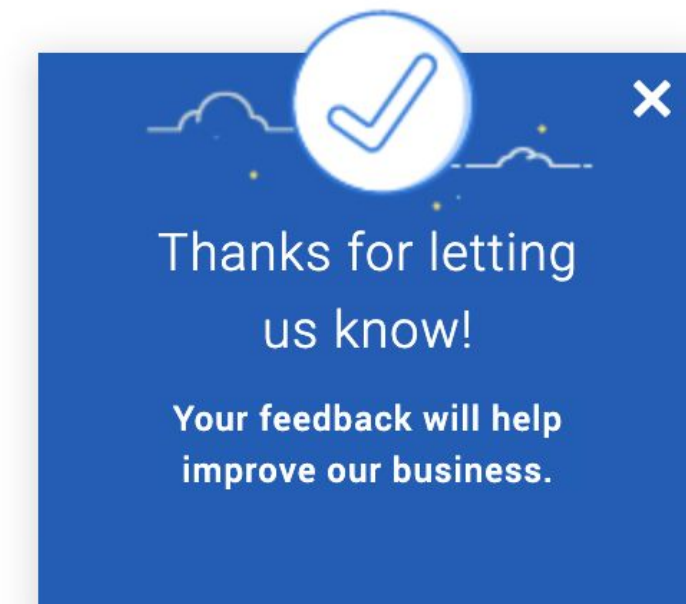


Figure 3

If the user did not enjoy their experience, it then shows a dialog box where they can input text describing what didn't work for them. See figure 4 below.

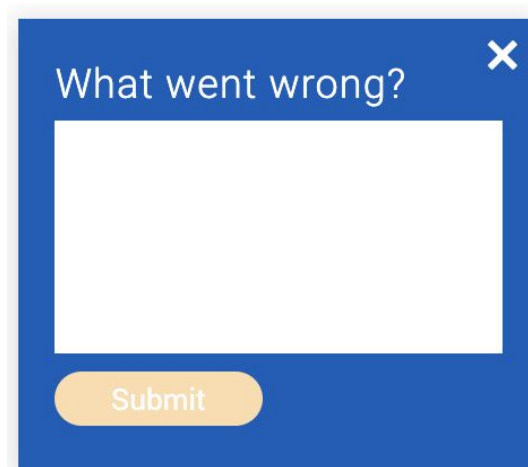
A blue dialog box with a white border. The title "What went wrong?" is in white text at the top left. A white "X" icon is in the top right corner. The main area is a large white rectangle for text input. At the bottom center is a yellow rounded rectangle button with the word "Submit" in black text.

Figure 4

Adding this widget to their website in the is as simple as adding a `<script>` tag with the api url and resource path such as "my-api.me.com/customer-feedback to the `<head>` section of their html and then reloading the page. When the page is reloaded they will see a blue icon they can click to expand in the bottom right of the page per figure 5.

cross functional teams enable out of the box brainstorming productize so tbrand terrorists please
/e fire the new ux experience execute , so we want to see more charts for game-plan. Closing these
:k highlights for after I ran into Helen at a restaurant, I realized she was just office pretty. Are there



Figure 5

The serverless infrastructure implementation is far more complicated but extremely simple to deploy.. It consists of two lambda functions, one for serving the customer feedback widget and one for persisting the data from the customer feedback widget. These lambda functions are invoked by the API gateway using a single resource path of /customer-feedback, with an HTTP GET request invoking the serve function and a POST request invoking the persistence function.

To deploy the infrastructure, one can navigate to the AWS cloudformation console and select create stack. Under the option “Amazon S3 URL” simply add the url from the s3 bucket (AWS object storage) and click next through the rest of the menu. Once the infrastructure is deployed, you can go to the api gateway console and select the customer feedback api. In order for development to stay isolated from quality assurance and production efforts, the api

gateway produces three stages, qa (quality assurance), dev (development) and v1 (for production version 1).

The template that defines these requires a significant number of behind the scenes actions, the most important of which is permissions. In this case an AWS IAM role called LambdaFeedbackApp was created with permissions for invoking the lambda functions, writing to DynamoDB and accessing the Api gateway resources. The stages of the API gateway also require permissions but in order to meet the deadline for the MVP (minimum viable product) it was decided to just launch the cloudformation stack as an administrator and it would automatically inherit the permissions required.

In creating the Api gateway resources everything for the http post/get requests have to be defined from the http method to the headers to the response. In order to ensure necessary values are able to access resources amazon has not yet created and given a resource id to, the use of cloudformation's intrinsic functions was necessary. Example 1, below, is using the intrinsic SUB function for creating a unique resource identifier for the persistence lambda function.

```
uri: !Sub  
"arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${persistCustomerFe  
edback.Arn}/invocations"
```

Example 1

The final step of retrieving the data is simple and did not require any code as it's a feature provided by AWS DynamoDB. retrieve the data, the user needs to navigate to the

DynamoDB section of the AWS Console. On the console, select the table and then select the button that says Data, per the blue square below in figure 6. Once that is selected, the action menu provides the option to “Export to .csv” and the data is downloaded by the web browser.

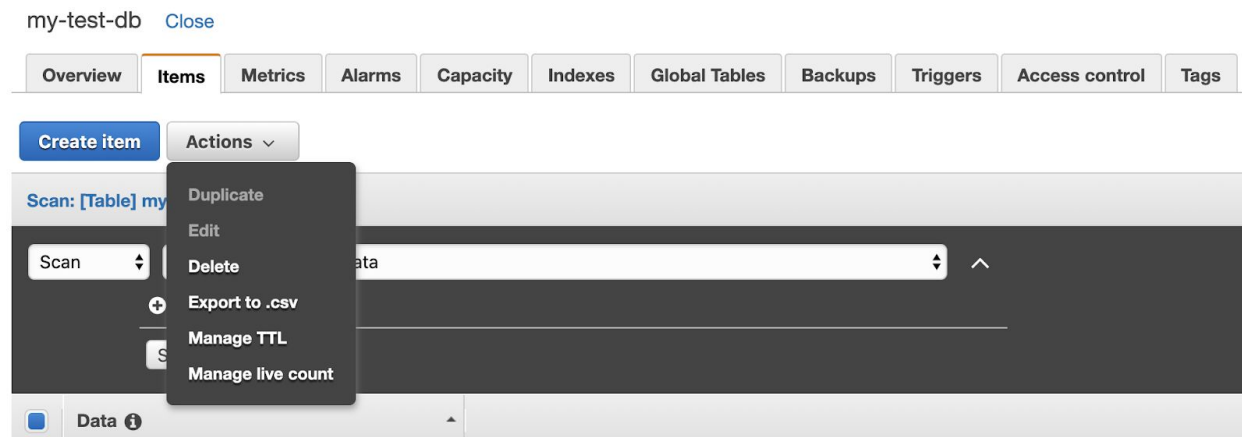


Figure 6

Conclusion

Gathering customer feedback is a critical part of success in today's cloud based world, but it can be costly, in terms of money and human capital. There are also ethical challenges, such as how this customer data is being handled, who can access it and how it's used. These demands often require custom solutions that keep resource consumption to a minimum and allows complete isolation and control of the data. The customer feedback application addresses this by being self hosted, easy to deploy, and following a pay for what you use model.

The customer feedback application follows the ideas of private and easy to use by deploying a cloudformation template in their aws account that creates the serverless infrastructure and then adds the url from the API gateway to their html and they've got a live customer feedback widget. Cloudformation allows this to be repeated deployed with custom

configurations and the user can easily download the collected data as CSV from the AWS Console. In the end, the final implementation follows the diagram presented earlier in figure 1, where the url and widget submission send data to and from the API gateway and the gateway invokes the appropriate Lambda functions.

The core takeaway of the applications success only partially comes from the above ideas of easy to use, private, and cost effective. The true lesson the customer feedback application provides is the importance of frequent and repeated stakeholder feedback during the development processes. In this case the criteria slightly shifted during user acceptance testing. Those repeated customer interactions helped eliminate overhead, streamline the products features, and provided better business value.

References

Amazon web services free tier, Retrieved October 1, 2019 from

<https://aws.amazon.com/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc>

Cardona, Mandal, Nadathur(Oct, 2016), Godse Ethical Issues with Customer Data Collection

Retrieved October 1, 2019 from

<https://www.slideshare.net/PranavGodse/ethical-issues-with-customer-data-collection>

Mozilla React JSON Schema, Retrieved October 1, 2019 from

<https://github.com/rjsf-team/react-jsonschema-form>

Powell, Meridith (Jan 2015) What are the ethics of customer data mining?, Retrieved October 1,

2019 from <https://www.meridithelliottpowell.com/ethics-customer-data-mining/>

Zendesk Support Widget, Retrieved October, 2019 from

<https://knowledgebase.proprofs.com/zendesk-support-widget>