



# CROSS-PLATFORM VOICE ASSISTANT DEVELOPMENT

School of Electrical Engineering and Computing

The University of Newcastle, Australia

Jarrad Price

Supervisors: Andrew Sargeant, Michael Bourke

A thesis submitted in partial fulfilment of the requirements for the degree of Bachelor of Engineering  
(Honours) (Software)

Submitted: 6/11/2021

Word count: 20543

Signed:

A handwritten signature in black ink, appearing to be "Jarrad Price", written over a light blue horizontal line.

Student: Jarrad Price

Signed: Andrew Sargeant, Michael Bourke

Supervisors: Andrew Sargeant, Michael Bourke

# Abstract

This project had an original brief in the form of “developing a virtual assistant that can help support departments in hospitals”. This virtual assistant was to be used by NSW Health Pathology staff who have questions on operation of point of care health devices. The responses were to be gathered from existing device manuals. This voice activated assistant field was not something NSW Health Pathology had attempted before.

The relatively simple question-answer intention of the virtual assistant combined with the open nature of the scope resulted in a software product envisioned to accomplish 3 main goals:

- A voice interface assistant that answers simple “FAQ like” questions.
- An accessible management hub where the system can be modified and updated.
- A centralised codebase with automatic deployment pipelines.

Once the initial platforms of Alexa and Azure were chosen for facilitating the virtual assistant a prototype was built that confirmed the validity of this approach. It was then decided to have a Google Sheet serve as the management hub. This allowed for users to easily add new responses to the Alexa application. Having three platforms to build and connect, combined with the requirements of a continuous deployment pipeline built up the development challenge of the project. To facilitate these systems and the centralised codebase requirement, a single GitHub repository was used to host the codebases. With this repository, GitHub workflow files were used to trigger automatic deployments on events such as updates of the code.

Difficulties in development also arose by using the Google Sheet as the custom management hub with two other cloud service companies. By having an Amazon Alexa app which uses a Microsoft Azure endpoint and database, it is a challenge to consolidate functionality across the platforms. This was solved by using Google's JavaScript based scripting service, Google Apps Script, which was used to add custom code to the Google Sheet. Google Apps Script was built to make HTTP calls to the Azure endpoint which added responses to the database. Google Apps Script was also built to make calls to the GitHub API which added new questions to the Alexa application.

The results of this project prove that you can have a “platform agnostic” approach to building a full virtual assistant system. This project adds new documentation to the field on integrating multiple different platforms into a cohesive virtual assistant toolset - Amazon’s Alexa using Microsoft’s Azure and Google Sheets for management.

The next big step for this project would be deployment and testing with real-world users. The core systems for the operation of the Alexa application are in place and the user experience needs to be focused on next. Attention and an observe-update loop needs to take place on real-world users interacting with Alexa to build a system that fully behaves as expected and accomplishes the goals of the end-users. A parallel step would also be evaluating if continuing with the Google Sheet should be done or another platform is to be developed to facilitate the same functions, due to the limitations of Google Apps Script.

This projects code can be viewed at: <https://github.com/JarradPrice/poct-device-alexa-apps>

# Acknowledgements

I would like to thank and acknowledge the two NSW Health Pathology supervisors I have had for this project, Andrew Sargeant and Michael Bourke. Their enthusiasm for the project domain and amiable personalities have made the process of development and meetings be an enjoyable experience throughout all of this project. I have been lucky to have them as my supervisors, even if it has only been through the virtual realm.

Next, I would like to express my thanks to my software engineering cohort. I have made friends who have been a beacon of aid, providing help when asked much to my happiness. Without their help and conversations, I would have had a much harder time getting through this degree.

Finally, I would like to thank my friends and family for their support. A big acknowledgement goes out to my friend since university day 1, Aidan Turnbull from environmental engineering. It has been a fun ride from young bright-eyed freshman to experienced and wizened (... to a degree) graduates. I would also like to thank the rest of my friends who have kept my social life alive throughout this all, without them I would be that much duller. And of course, big love for my family who has always been there for me and providing never-ending support.

# Contents

Introduction.....	1
Purpose.....	1
Development team .....	1
Scope (brief).....	2
Definitions, acronyms, and abbreviations.....	2
Literature Research .....	3
Overview .....	3
Advancement & awareness of virtual assistants .....	4
Healthcare adoption & legal requirements.....	5
Patient information proximity .....	6
Regulation requirements .....	6
Use case studies .....	7
Study - Virtual assistant road to college .....	7
Study - Implementation of virtual assistants for education.....	8
Software Specification .....	10
Product perspective.....	10
Product functions .....	10
User characteristics .....	10
NSW Health staff.....	10
Project methodology .....	11
Constraints .....	12
Assumptions & dependencies .....	13
Requirements .....	14
Final requirements .....	14
Beginnings .....	15
Evolution.....	16
System Architecture.....	17
Final architecture .....	17
Beginnings .....	19
Evolution.....	21
GitHub repository .....	23
Introduction.....	23
System design .....	23
Automatic deployments .....	25
Journey .....	31

Amazon Alexa .....	34
Introduction.....	34
System design .....	36
Journey .....	39
Azure Functions and Cosmos .....	42
Introduction.....	42
System design .....	43
Journey .....	47
Google Apps Script and Sheets.....	51
Introduction.....	51
System design .....	51
Journey .....	55
Testing .....	57
Skill beta .....	59
Device testing.....	60
Alexa API Tests .....	61
Utterance Conflicts .....	61
Validation Pass.....	61
Amazon Certification.....	62
Static Quality Analysis .....	63
Reflection.....	65
Cut and unimplemented content .....	65
Future system expansion.....	66
Retrospect .....	70
Developer notes .....	71
Results & Conclusion .....	72
Evaluation .....	72
Contribution to the field.....	74
Final .....	74
References.....	75
Appendices.....	77
GAS Cosmos Authorisation Token Code .....	77
Interim Report.....	78
Deployment document.....	80

# Introduction

## Purpose

This report serves as project documentation for the University of Newcastle Software Engineering final year project “Set-up virtual assistants to help with supporting departments in hospitals”. This project is a collaboration between the university and NSW Health Pathology. This document will contain the following components:

- Literature Research
- Software Specification
- System Architecture
- Testing
- Reflection
- Results & Conclusion

The intended audience for this document includes the project supervisors (University of Newcastle & NSW Health Pathology) and developers, specifically those new to the field of virtual assistants. This document will provide a developer with the needed information to use this project as a base for building their own virtual assistant implementation or building on top of the developed one.

To cater to developers, certain sections of the document have been written in a different manner with a more “development log” style approach. They act as a pseudo development log by stepping through the development process, with the challenges faced and how they were overcome. These sections are targeted at developers and as such provide more resource links on discussed topics. They are meant to be used to avoid facing the same problems if also entering the domain space of the project, as they serve insights into the development steps.

## Development team

This project has been developed by a one-man team. All use of other people's work has been documented in this report and/or attributed in code comments. The work of this project has been completed by a final year software engineering student at the University of Newcastle across two semesters.

## Scope (brief)

The following is the scope that was decided upon the beginning of the project.

The current environment of NSW Health does not utilise virtual assistants in any manner. As such a quality-of-life improvement for staff will be a voice activated virtual assistant. This project aims to help staff users access information more quickly and efficiently, enabling a better workflow. The VA is to be voice activated and interfaced with through smart speakers that are to be located within the same space as the health devices. These health devices are what the staff will be using and are the subject of the questions for the VA. This project will not build its own assistant with custom built language parsing but instead use an established assistant platform and its development tools.

The final scope was expanded to include specification on what platforms are used and how platforms are to be managed, this is expanded upon in the Software Specification section.

## Definitions, acronyms, and abbreviations

Acronym/Abbreviations	Definition
API	Application Programming Interface
ASK	Alexa Skills Kit
AWS	Amazon Web Services
Chatbot	Conversation Bot
CI/CD	Continuous Integration / Continuous Delivery
CLI	Command Line Interface
Cosmos	Cosmos DB
FAQ	Frequently Asked Questions
GAS	Google Apps Script
NSW	New South Wales
NSW Health	NSW Health Pathology
PoCT	Point of Care Testing
Repo	Repository
Skill Management API	SMAPI
Software Development Life Cycle Model	SDLC
TGA	Therapeutic Goods Administration
URL	Uniform Resource Locator / Web Page Address
VA	Virtual Assistant

# Literature Research

## Overview

The literature review for this project involved procuring reports on three main areas of VAs:

1. VA advancement and growing forefront to consumers
  - This gave a widened understanding of the field that is to be built in and what is the current state of VAs, their capabilities, drawbacks, etc.
2. VA adoption in healthcare and legal requirements
  - This allowed for a more specific insight into what has already been attempted and is growing in the projects home field.
  - It was also required for understanding if the project is beholden to any laws and regulations.
3. Use case projects
  - These gave knowledge into the actual development process for a VA and enabled learning from already attempted projects.



## **Advancement & awareness of virtual assistants**

Before starting development for this project and making important decisions like what platform to utilise the advances and consumer awareness in the VA space was first delved into. Without the recent explosion of VAs and their use this project would not be possible, as this massive uptick in awareness has correspondingly included an upgrade in development tools.

At its core VAs are a paradigm shift of human-computer interaction. They represent developments in several areas, these being a composition of:

- Machine translation & chatbots
- Information extraction & retrieval
- Text summarization
- Parsing of text into parts of speech
- Topic modelling
- Sentiment analysis
- Natural language understanding and generation

The deep learning aspect of VAs contributes to rapid advancement in natural language processing, an underlying base component of processing human voice input. Although the advancements have been immense, virtual translators still cannot beat human ones. Due to still existing imperfections compared to a human, the use cases of VAs are best suited to give a general understanding, while accuracy critical applications like diplomacy or structure/tone important situations remain an inappropriate use case. [1]

Regarding consumer awareness, the increased development and application of VAs has resulted in increasingly more consumers being exposed. The first smartphone VA Siri is the current most used assistant out of the market share of Apple Siri, Google Assistant, Amazon Alexa, Samsung Bixby, and Microsoft Cortana. The second largest platform used behind smartphones is smart speakers where Amazon occupies the majority market share. Also to take into consideration is the development of VAs as a service to be integrated into any number of devices, with adoption being seen in cars, home appliances, headphones, smart watches, and smart TVs. This wide range of availability on devices builds VAs into becoming a ubiquitous experience. [2]

This combination of increased VA intelligence and consumer use results in a space that is becoming increasingly important in the eyes of prospective companies who are wishing to remain relevant and give off a vision of remaining current.

## Healthcare adoption & legal requirements

Healthcare can be classed in the “early adopter” phase of the technology adoption lifecycle for VAs. The United States has seen the healthcare industry experiment with VA functionality. Providing FAQ, appointment scheduling and post-op reporting has been the most popular services.

As the use of VAs become more prolific consumers will increasingly come to expect healthcare to also have such systems. In the current use cases that have been developed and deployed in healthcare the most common are:

- Asking about illness symptoms.
- Medication information.
- Locate care facilities and providers.

Some healthcare providers have developed their own “in-house” VA platforms. The main advantage of this is ensuring health related regulations are met. The disadvantage is a much higher development effort and consumers not having been previously exposed to the platform unlike the already established ones, Apple Siri, Google Assistant, etc. [2]

When choosing a platform for developing the healthcare VA app, there are different considerations:

- Amazon Alexa – offers a large user base in smart speakers, easy methods for healthcare providers to publish voice apps that offer general purpose healthcare information.
- Google Assistant – offers a large user base on smartphones and speakers, easy methods for healthcare providers to publish voice apps that offer general purpose healthcare information.
- Apple Siri – offers a large user base on smartphones with a focus on personal health monitoring through Apple Watch.

Initial adoption in the healthcare space was slowed by:

- Regulations
- Need for IT infrastructure for health system integration (E.g., electronic health record)
- Compliance to exchange personal health information

Australia also has legal frameworks that must be followed when developing medical technology, which must be considered for this project. The key considerations are:

- Proximity to patient information
- Regulation requirements

## **Patient information proximity**

Privacy of individuals' health information is protected by privacy acts [3]. Any health information collected must be consented to. This is a concern for this project as the VA will exist within spaces where patient health information can be processed and gathered. Due to this proximity to private patient information the VA may inadvertently collect data. When the VA is invoked and activated it collects all speech from the user until it detects they have stopped talking. This can result in unintended information being received by the VA by way of unintentional activations, or unintentional captured speech.

To ensure any accidentally captured information complies to privacy acts the designed system must:

- Not connect to any external storage systems outside of itself
- Not store queries for an extended period

## **Regulation requirements**

The Australian Therapeutic Goods Administration (TGA) requires that any software based medical devices that are defined as a “medical device” be subject to TGA regulatory requirements. [4]

By cross-checking with the document developed by TGA, it can be discerned that this project software does not fall under the regulation requirements of the TGA. The specific excluded software example provided by the document is as follows:

*“Software intended to administer or manage health processes or facilities, rather than patient clinical use cases. This includes software for the processing of financial records, claims, billing, appointment schedules, business analytics, admissions, practice and inventory management, utilisation, cost effectiveness, health benefit eligibility, population health management, and workflow.” [5]*

# Use case studies

## Study - Virtual assistant road to college

This study implemented a system for a VA to support new students with the transition to college. They argue that AI could drastically change the viability of providing students personal assistance. It was tested if a conversational AI could effectively help a potential college freshman with transitioning to college by use of personalized text messages. The applications of this study are relevant to this project as it uses a question-answer relationship with the users.

Their virtual assistant “Pounce” was developed to reduce the number of students who would fail to complete the transition from high school to college, it accomplished this by then reducing the percentage by 21%. This was in line with previous attempted projects to reduce “summer melt” (when students don’t complete the transition) but was advantageous as it had less of a burden on staff. Another advantage of the VA system was its ability to integrate with the college's information on the students’ progress in enrolling, this provided a richer dataset with better quality data on different stopping points of enrolment (E.g., accessing financial aid, submitting required paperwork, and attending orientation).

The VA was composed of four key components:

1. Topical architecture – branching message flows involving over 90 enrolment topics
2. Data sharing – they were able to personalize responses by integrating with university student’s information and management systems
3. Knowledge base – for the automation of responses they started with around 250 FAQ. This grew over the lifetime of the system
4. Text-to-email routing – when the VA could not answer an asked question they were automatically forwarded to counsellors by email. Replies were routed directly back to students. The response was then checked by staff and added to the VA knowledge base, reducing the amount of human interaction needed over time

This use case highlights one of the key features of VAs, their “growing potential” over time where they can learn as more user interaction occurs. [6]

## **Study - Implementation of virtual assistants for education**

This use case study developed a VA to provide students access to content for a course through an interactive medium. Their use of a VA is beneficial in education cases due to:

- The ability to serve questions and answers to thousands of students at the same time is perfect to counter the scarcity of teachers and lecturers in schools and universities around the globe.
- It creates a better tool for online courses to improve their students' performance.

The aspect of many users to a small amount of knowledge experts is directly relevant to this project's domain of providing device information to the many device users and bypassing having to go to the small number of device support staff.

Their development platform of choice was Google Dialogflow. Google Dialogflow is an in-browser visual flow builder that is used for building chatbots and voicebots. Its purpose is to increase the ease of creating conversational experiences across devices. An essential requirement for their system was the ability for a lecturer, who has no prior knowledge of chatbots or AI, to be capable of creating, modifying, and implementing changes to the educational tool in a simple and efficient manner. This was deemed not feasible by providing access to Dialogflow tools as the system still requires knowledge on the development process of building a bot conversation.

They experienced difficulties working with the Dialogflow platform, it imposed constraints on development as it would freeze and even close. This highlights the infancy of the platform. Fortunately, they were able to address simple modification of the bot and avoid directly working in Dialogflow by making use of an API provided by the platform that allows tying into Google sheets. With this Google sheets integration, it was possible to modify the bot in a more stable and user-friendly fashion.

This study notes that it is possible to generalize their development methodology as it is content independent. Their methodology builds on the basis that there is a database of questions which are grouped into intents. The methodology is divided into two parts, knowledge abstraction and response generation.

Knowledge abstraction is the analysis of course content which they refer to as their data, it involves three phases:

- Data gathering – generating a knowledge base. This is procuring possible questions for the bot from sources such as, discussion forums, social media interaction with students or messaging applications. The developers then classify the questions into topic categories.
- Data manipulation – storing the information in a database. Question-answer pairs are created by developers. Questions and answers are stored in two separate spreadsheets. Questions with similar answers are used later in Dialogflow to build intents.
- Data augmentation – building on data manipulation. This is for increasing the number of training examples needed for Dialogflow to be effective. Sometimes answers can be used to create questions.

Response generation is the output of intents from contexts, training examples and responses. These are implemented within the Dialogflow system. [7]

# Software Specification

## Product perspective

NSW Health does have some virtual chatbots in place but none of these feature voice interfaces, and none are geared towards point of contact health device staff. At the beginning NSW Health expressed that they are open with use cases and are happy to have multiple. What NSW Health does have is large amounts of existing procedures and videos for operating PoCT health devices. The key desire for them is to have this knowledge available as “quick snapshot answers” [8]. In the first meeting it was also expressed that they were happy for the product to be freely shaped throughout development. The current option for NSW Health staff for retrieving procedures is quite manual and requires stopping what they are doing to seek out. This product aims to address this pitfall by developing an easy-to-use VA for staff which is easily extendable.

## Product functions

This product features three key hallmarks:

- VA procedure application
- Assistant management hub
- Centralised codebase handling and auto deployment

## User characteristics

This product has one type of user, NSW Health staff.

### NSW Health staff

Experienced with the health device equipment, this user does not need walkthrough tutorials of how to operate and as such do not need replication of whole procedures repeated back to them. They will want answers to simple things they have forgotten.

These users are very likely to have experienced a VA before and therefore the concept does not need to be taught to them.

## Project methodology

This project involved fortnightly client meetings where progress was shared, and feedback received. This process lends itself to an iterative prototyping design approach. Prototyping is used for procuring customer feedback in a develop, refine and testing process loop.

The specific Software Development Life Cycle Model (SDLC) applied is Incremental Prototyping, an overview of this process is provided in Figure 1. In this model the requirements are broken into standalone modules. For this project, the individual increments were the next identified features that improved the system or that was a core requirement.

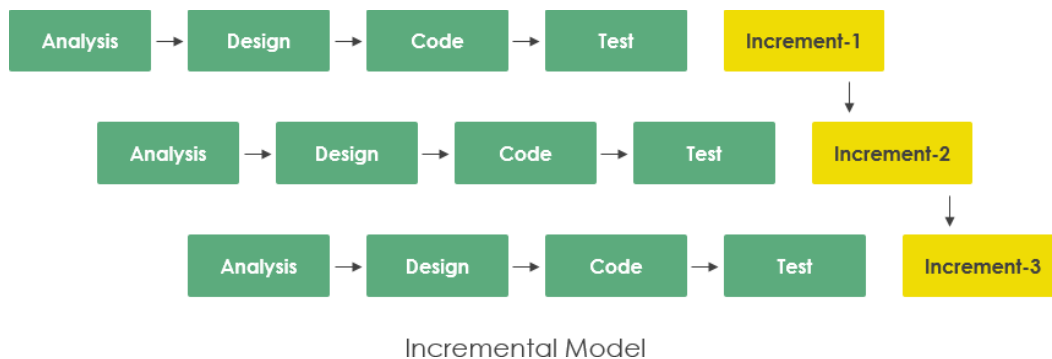


Figure 1. Incremental model approach to SDLC [9].

This approach turns the development of the modules into their own small development projects where the systems are combined as they are developed until all features are present in the system and the product is accepted by the client.

Within each increment is the following phases:

- Requirement Analysis
  - Requirements and specifications of the next feature of the system are gathered, this comes through use and testing.
- Design
  - The design of the feature is established. For code the implementation is prototyped and for new visual features they are mocked up.
- Code
  - The new feature is coded and pushed to the in-development deployment.
- Test
  - The new feature is tested by way of developer run-throughs.

Once the increment is complete it is joined with the previous increments. This will result in the capability of the system to expand. If a certain increment has work that extends past its original planned increment time it is divided into sub-increments.

This methodology remained the approach throughout the whole project.



## Constraints

- Time
  - The scope of the project was limited by the development time allotted. This was restricted due to the length of the project course and other workloads and obligations (university courses, work commitments) the developer was beholden to.
  - The project had at a minimum 5 hours a week put in by the developer for the first semester, this then increased to 10 hours per week in the second semester.
- Technologies
  - The project only used technologies with public access and documentation.
- Funds
  - Only software and devices within the budget of the development team was considered. This was set to a maximum of 100\$ AUD for any one item.
  - This did not consider the expiration of free tiers or trial software.
- Access to product environment
  - Due to restrictions in place over the course of the COVID-19 pandemic, access to the deployment environment of NSW Health Pathology was not available which hindered deployment.
- Public hosting
  - This project's codebase is publicly hosted and available to be viewed by anyone.
  - This added requirements of masking any sensitive information before storing it into the repository codebase. This resulted in repository secrets, token substitution and masking having to be utilised.
  - This is expanded on in the GitHub repository section.

## **Assumptions & dependencies**

### **Assumptions:**

- The system needs to be adaptable in its handling of users. Due to the nature of the system and its environment the system may garner little or a lot of use. The system and its architecture should be able to easily scale up and down.
- The system does not need to integrate with existing systems used by NSW Health. This system is built to be able to stand alone.
- NSW Health can bear any costs (within a reasonable margin) that the final deployment would require, i.e., VA devices, non-free tier subscriptions.
- The system does not have to consider and will not save sensitive patient information, and as such does not need to adhere to Australia New South Wales Government patient information laws and policies.
- Any users of the deployment documentation will be comfortable with or have introductory knowledge of the technologies used.

### **Dependencies:**

- Building VA responses for this project is dependent on existing device manuals in possession of NSW Health.
- The platforms used are dependent on the up time of used web services. This project makes use of online web services from multiple different platforms. The correct functioning relies on the stability and availability of each of these platforms at any given time.

# Requirements

## Final requirements

An overview of the final requirements is displayed in Table 1.

*Table 1. Requirement types for the project.*

<b>Business</b>	<ul style="list-style-type: none"><li>▪ Reduce time taken to use PoCT devices as opposed to pulling out device manuals</li><li>▪ Increase usability after deployment</li></ul>
<b>User/Stakeholders</b>	<ul style="list-style-type: none"><li>▪ NSW Health staff</li></ul>
<b>Functional</b>	<ul style="list-style-type: none"><li>▪ Recognise user questions and respond with the correct response</li><li>▪ Changing of questions and responses from a single separate solution</li></ul>
<b>Non-functional</b>	<ul style="list-style-type: none"><li>▪ Scalability</li><li>▪ Easy deployment</li><li>▪ Endpoint response time ideally as fast as possible</li><li>▪ Extendibility</li></ul>
<b>Implementation/Transition</b>	<ul style="list-style-type: none"><li>▪ Automatic deployment solutions for each component of the system</li><li>▪ Deployment documentation/wiki</li></ul>

The overarching requirement of this project is to develop additional support for departments in hospitals by use of a virtual assistant. The domain space of this project is NSW Health staff who wish to have quick access to procedure information on health device operation and queries. Such a device query could be:

- “What do I do if I get an INR result greater than 4?”

This information is drawn from pre-existing device manuals. This project aims to help users access such information more quickly and efficiently, enabling a better workflow. The VA is voice activated and interfaced with through smart speakers that are to be located within the same space as the health devices. It provides quick access to information with results returned without needing physical interaction. Responses are pulled from a database that is developed from information currently contained in device manual documents.

## Beginnings

The initial project brief as provided by the university was as follows:

- Set-up virtual assistants to help with supporting departments in hospitals.
  - NSWHP would like to provide additional support for departments in hospitals with virtual assistants. Outcome/objectives: While leaning towards Amazon Alexa, there is no current arrangement within NSWHP. Platforms: Alexa, Google Assistant and Siri.

Upon the first meeting with my supervisors, they conveyed that they were happy for me to shape the end products and gave their interest in the final product to be in two parts. The first part would be a voice activated voice tool (Alexa, Google Home), that is for procedural based stuff i.e., asking health device questions. This tool would feature asking questions, vetting answers, validating answers, and tracking to keep improving. The second possible part of the project would be an online chatbot/virtual assistant that people go to ask questions. They also iterated on it being an open path from ideation, to prototyping, to embedding for me.

These specific project outcomes had not been attempted by NSW Health (although they did have some chatbots working) before, so a large part will be the setup of new systems, one such being the database of responses the system will use. A mentioned desirable feature was the ease of adjusting to the new systems. This assistant (voice activated and chatbot) would have lots of procedures, its value is for little things, "what coloured tube do I need for this test" and the like for health device procedures. The core problem is the access to information with quick results to use right there and then, a quick snapshot answer.

On a second meeting I was able to further refine what the specifications of each part would be.

Part 1

- My supervisors were leaning towards Google or Amazon voice assistant solutions.

Part 2

- Having a person at a desk responding to queries would be suitable for the chatbot system. Ideally the bots will recognise if they cannot suitably respond to the user and get a live agent.
- A platform that they have experience with is Microsoft's Power Platform which has a virtual agent solution, Power Virtual Agents. This would be a good starting point for this part's research.

So, the key points after these first meetings were:

- They are comfortable with me progressing how I see fit, setting deadlines, project methodology, technology use, etc. I oversee my schedule and deliverables.
- End users will be NSW Health staff who are using the point of care devices.
- Operations manual type answers. Assists in providing information on how to operate the device. E.g., "How can I run QC (Quality Control) on a Coag XS Pro?"

## **Evolution**

Upon early analysis of the timeframe available for the project and the workload for each part, it was quickly decided that implementation of only one part would be feasible within the scope. When asked what part is the most desirable to see a working system my supervisors indicated for me to go down the path of a voice activated system. I also decided to implement the voice activated tool as this was a field that is a lot newer to developers and would provide an interesting research and development process.

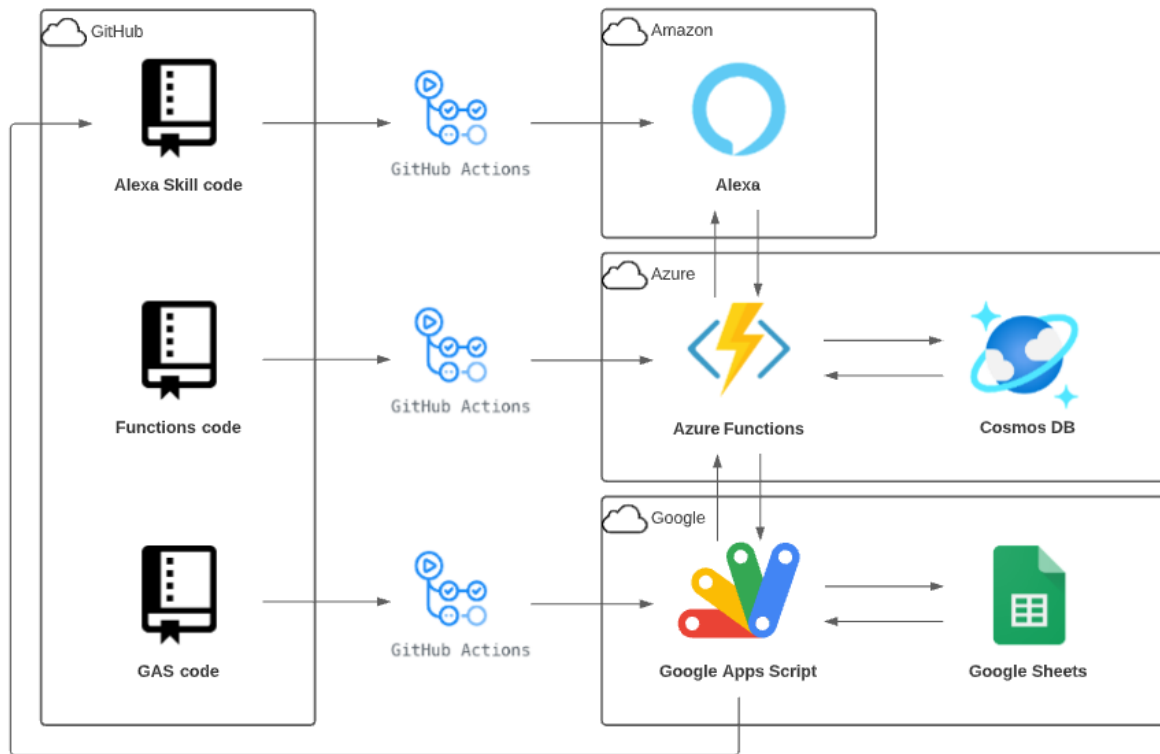
An early scope clarification was upon the type of communication that was to be achieved between the user and the system. The two that had been mentioned were operations manual type answers where it assists in providing information on how to operate the device, "How can I run QC (Quality Control) on a Coag XS Pro?", or it would be a conduit for IT services first level support where if the bot is not able to assist, then it should log a ticket in Dynamics 365 for one of IT staff to follow-up. Due to the focus on the voice activated type system the first communication of device manual knowledge was chosen as the syntax for the assistant.

Further clarification revealed that the virtual assistants will be in a room with multiple devices. Its aim is not to replicate whole procedures, but rather more simple things that the end user have forgotten i.e., information from the health device manuals FAQ.

# System Architecture

## Final architecture

The final system architecture is displayed in the diagram in Figure 2.



*Figure 2. System architecture diagram for the final project.*

The final design makes use of four platforms:

- i. GitHub
- ii. Amazon
- iii. Azure
- iv. Google

A brief of these platforms is provided below, they also have their own main sections where they are expanded on.

## **GitHub**

This platform is responsible for storing the codebase. Each of the other platforms' code is contained within one GitHub repository. Having one repository makes it easier to manage each platform's code and provides a central location for updating documentation and monitoring updates. The foremost leading advantage that GitHub provides is its relatively new auto deployment solution, GitHub Actions. This tool automates the software workflows for deployment of the system components, whenever any of a platform's code is updated by a Git commit the Actions workflow runs and deploys the new code.

## **Amazon**

Amazon's voice assistant was chosen as the platform of choice and as such Amazon hosts the Alexa skill which parses user questions. Amazon provides helpful tools for managing a skill outside of the online developer console, these have proved essential for the GitHub CI/CD management of the skill.

## **Azure**

Microsoft's Azure platform serves as the host for the endpoint and database solutions. The Azure platform features strong scalability and a subscription model that charges based on how much it is used, ideal for this project.

## **Google**

Google Sheets acts as the user-friendly interface for end users to update the questions and responses easily without having to touch any actual code. This sheet is dependent on Google's scripting tool, Google Apps Script which provides the necessary functionality to export questions to the Azure skill code and responses to the Cosmos DB. It also has features for creating skill betas and creating merge requests for the GitHub repository.

## Beginnings

Before any sort of system architecture could be designed a platform had to be chosen and the information flow within that platform researched.

My initial impression of this before meeting with my supervisors was directly using the online Google Assistant developer tools to develop an assistant. After my meeting I was provided with more context of the current working environment of NSW Health and their technology stack.

Key architecture points from early initial meetings:

- They were open to any platform that I thought would be best, but once one is chosen to commit strongly.
- They primarily make use of the Microsoft stack of products and have open dialog with Microsoft who provides lots of support. They have Microsoft representatives keen to help.

Armed with this new knowledge I researched into the big three of voice assistants, Amazon's Alexa, Google Assistant and Apple's Siri. I initially did a small amount of research into Microsoft voice assistants, Cortana or voice capabilities provided by their virtual agents in the Power suite. I pivoted to the "big three" after discovering that Microsoft does not provide a suitable platform to accomplish the project's goals. Microsoft's virtual agents do not provide any voice capabilities and their virtual assistant Cortana has no dedicated devices, they instead give the impression of desiring to turn Cortana into more of an "app" that is not a competitor to Alexa or Google Assistant [10].

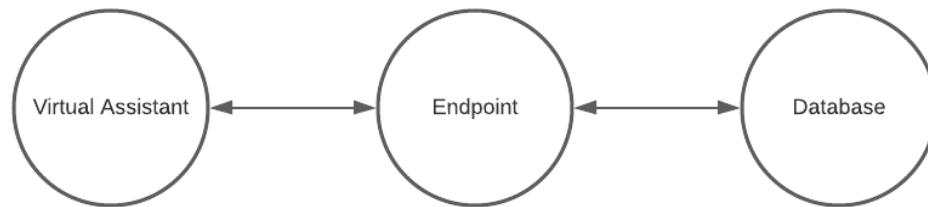
Siri was quickly eliminated as an option as the use case requires voice activated devices near the areas where the operational devices will be used by staff, and Apple does not have robust support for this with their small range of smart speakers. This left the two platforms that provide large number of devices and support, Amazon's Alexa, and Google's Google Assistant.

After comparing the two I decided that Alexa will be the best route due to a couple of reasons:

- Better and increased documentation of Alexa. Alexa has been around longer and has a larger market share for business development use so there is more documentation/discourse.
- It seems Microsoft is focusing on integration with Alexa [11] and due to the business's current majority use of Microsoft services it makes sense going with Alexa.
- From what I had seen the development front-end for both Alexa and Google Assistant are very similar, switching to another platform probably wouldn't be a huge task. You would just need to rebuild the "intents" and the coding around handling JSON request from Dialogflow/Alexa and format of response.



From this introductory exposure into the platforms, I also began to understand the basic data flow in the system. I was able to recognise three main components, the “front-end” being the assistant platform that parses the users voice, the endpoint which receives the recognised intent from the frontend, and the database which has the information that the endpoint grabs to send back to the front end. Figure 3 is a simple diagram of this process.



*Figure 3. Simple diagram of VA systems.*

So, my view of the architecture from this point became the following:

1. Alexa skills used to build intentions - "Give me step 2 for the radiometer"
2. Intention is sent as HTTP request having a JSON payload
3. This request is received by Azure Functions
4. Azure Functions uses the Dataverse database to find the specific information
5. Sends back response information to Alexa
6. Alexa gives the user the response

This first established data flow is displayed in Figure 4.



*Figure 4. First draft of the project's system architecture.*

I initially started with Dataverse as the database solution, but this was quickly replaced with Cosmos DB as I felt it was more appropriate for a new project like this, instead of trying to integrate with NSW Health’s existing Dataverse it would be easier and more appropriate to use another of Microsoft’s data store solutions - the document-based Cosmos DB.

So, by the end of semester 1 I had nailed down the following about the systems architecture:

- Amazon’s Alexa platform for the assistant host
- Azure Functions for the endpoint
- Cosmos DB platform for the database

## Evolution

In semester 2 development started and this quickly led to revisions to the architecture of the project. The Function app started with just containing one Function which served as the Alexa endpoint, receiving Alexa requests. This changed to holding two Functions, one serving as the Alexa endpoint and the other serving as a HTTP trigger where requests can be sent to get data from the database. During the same time as this was when the Google Sheet first was created. Originally it was just a “dumb” storage solution used to hold question-answer pairs where you can also keep track of what has been implemented in Alexa via colour coding. This Sheet would change from a simple memory offloading tool to a robust central management tool.

The next big change was turning the Google Sheet into an actual interactive tool by implementing a method where a sheet can be updated using data from the database. The first iteration of this had a direct connection between the Apps Script behind code of the Sheet and the Cosmos DB. The GAS would make a REST call to the Cosmos REST API. This required a hash signature in the authorization header of the call, this signature used an access key for the database [12]. This solution was deemed undesirable following consideration as:

- The process in GAS to get the authorization token using the key was unwieldy and had to have an access key in the GAS code.
- It does not follow the typical tiered design of security in layers, the GAS was connecting straight to the database, albeit through an API but without a middleman to format and document requests.

The next change was the introduction of automatic deployment with the GitHub Actions pipeline. The first automatic deployment setup was the Azure Function app, then the GAS code, and finally the Alexa code. These each provided different challenges and research to overcome, this is expanded in the GitHub repository section.

It is after this point that the final system architecture diagram was envisioned with these key changes:

- Connection service in the Azure Function app for the GAS. This provides advantages of:
  - Easier communication from GAS to Cosmos.
  - Allowing for easier validation checks on incoming entries.
  - Can also achieve functionality of creating the database items if it does not exist yet, or if a new device document is needed.
  - Provides the concept of “security in layers” [13].
  - Easier monitoring of requests, better logging.
- Having the GAS push updates to the Alexa skill code Git repository. This provides advantages of:
  - Able to manage both answers and questions in the Sheet.
  - Auto deployment of skill when changed from Sheet.

Towards the end of the project development smaller updates and iterations to the architecture design were made:

- To increase performance the Alexa endpoint would use its own database access model as opposed to calling the other Function. This increase in performance was deemed required as response time is the highest priority for Alexa responses.
- A shared repository for code was made. Instead of three separate repositories the codebase of GAS, Azure and Alexa skill were contained in one repository.
- A “main” and “dev” branch deployment difference for the Alexa skill.

# GitHub repository

## Introduction

GitHub was chosen as the de facto source control solution from day one. This was due to my previous experience with the platform and its universal acceptance. I also had a small understanding of the auto deployment and knew I wished to pursue that feature set. Advantages include:

- Git versioning
  - Can easily keep track of changes and progress.
- Source control
  - Provides an online hosting platform for the codebase. Using a public repository allows for easily sharing the code and contributing to the field. This also makes handover easier.
- GitHub Actions
  - Backbone of the CI/CD pipeline.
- Documentation/wiki
  - Able to host a wiki with the repository. This combining of code and deployment documents help with handover and contributing to the field.

## System design

The final solution involves a single repository with the structure as so:

- .github/workflows
  - Workflows that facilitate the CI/CD pipeline.
- alexa-skill
  - The ASK code.
- azure-endpoint
  - .NET code for the Azure Functions.
- spreadsheet-gas
  - The JavaScript code for the spreadsheet.
- tests
  - Scripts used by workflows.

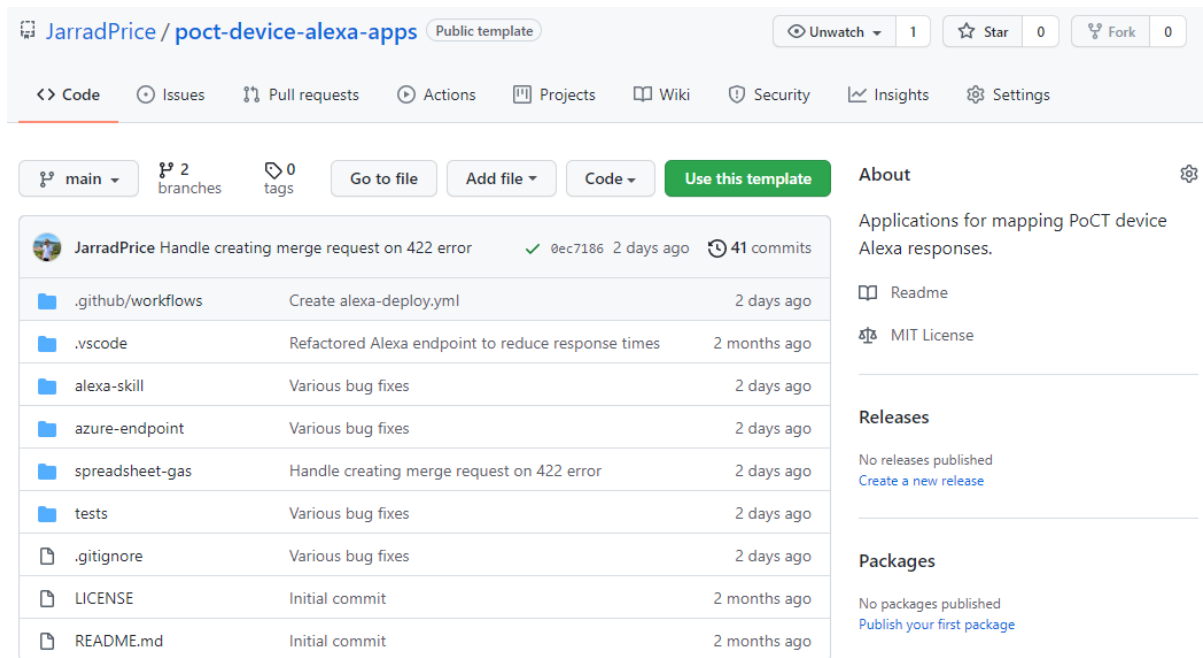


Figure 5. Screenshot of the final project GitHub repository page.

Having all the project code in one repository allows for an easy central location for management.

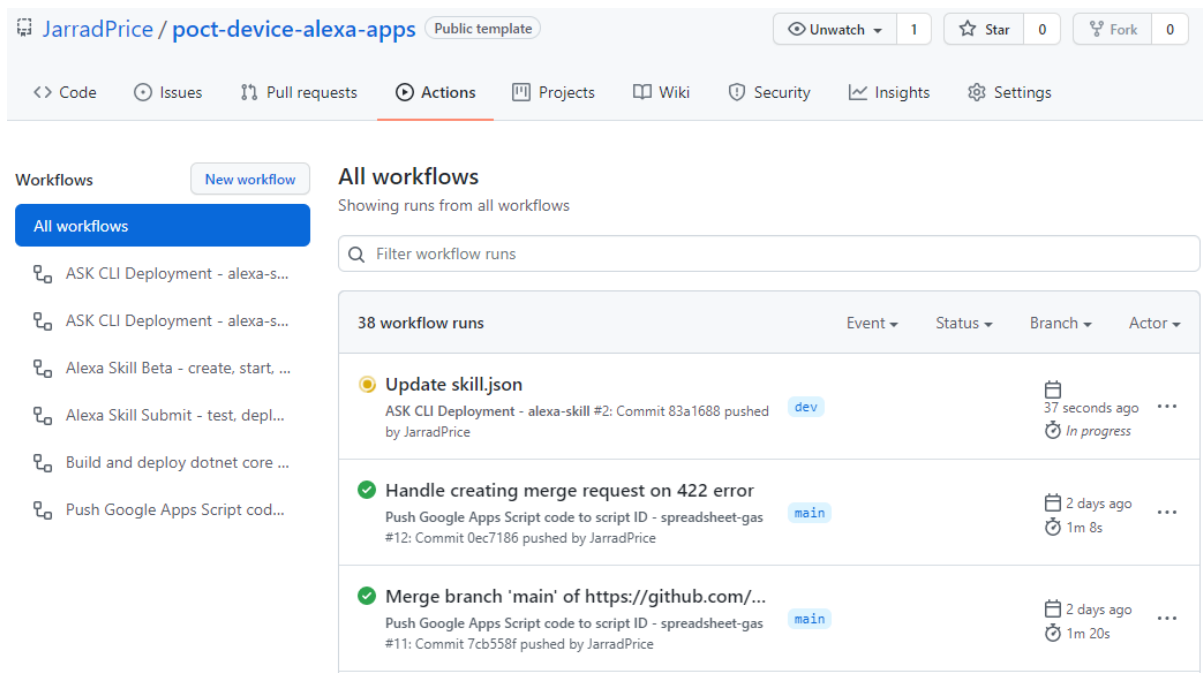


Figure 6. Screenshot of the Actions page for the project repository.

From here all the deployment can also be seen in the Actions page for the repo, as shown in Figure 6. Each main section of the repo has its own deployment workflows (ASK CLI, Azure Dotnet, Apps Script Clasp) that trigger when a commit changes any file in the folders of their concern.

The following section will go into how the automatic deployment is achieved.

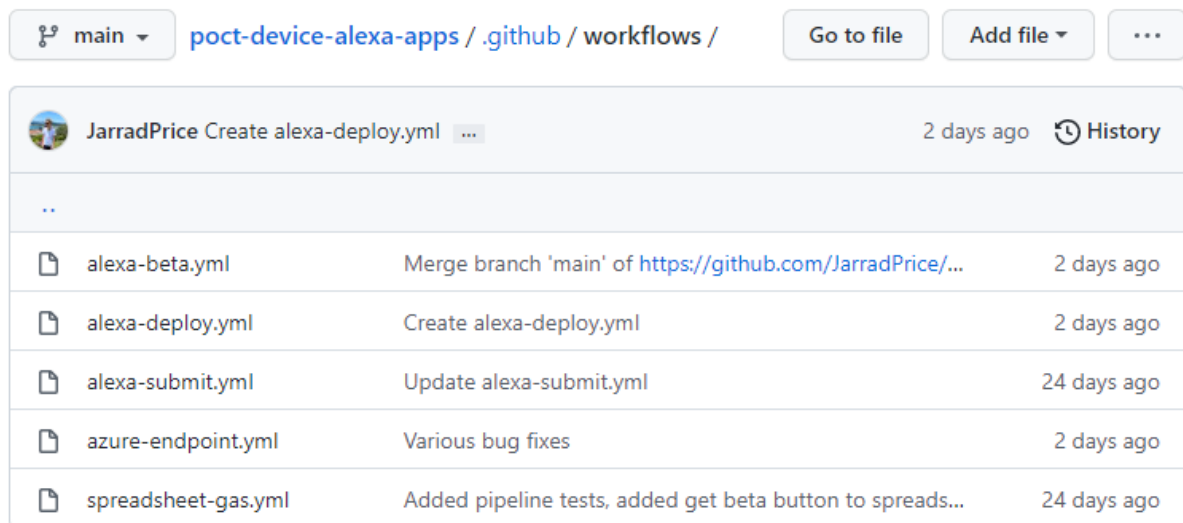
## Automatic deployments

GitHub provides the GitHub Actions service. This service aims to help automate tasks that occur in the software development life cycle. GitHub Actions are tasks that are event driven, meaning we can trigger something after a push is made to a branch in a repository. A workflow is a procedure in a repository. Within workflows jobs are performed that can perform different actions including script commands.

*“A workflow is a configurable automated process made up of one or more jobs. You must create a YAML file to define your workflow configuration.”*

*<https://docs.github.com/en/actions/learn-github-actions/workflow-syntax-for-github-actions>*

Workflows are written with the YAML syntax. These files must be located in the “.github/workflows/” directory in the repository. Within this project workflows are used to build the CI/CD processes.



The screenshot shows the GitHub interface for the repository 'poc-device-alexa-apps' in the '.github / workflows /' directory. At the top, there's a navigation bar with 'main' selected, the repository path, and buttons for 'Go to file', 'Add file', and a menu. Below this, a header for the workflow list shows 'JarradPrice Create alexa-deploy.yml' with a '2 days ago' timestamp and a 'History' link. The main content is a table listing five workflow files:

File Name	Commit Message	Time Ago
..		
alex-beta.yml	Merge branch 'main' of <a href="https://github.com/JarradPrice/...">https://github.com/JarradPrice/...</a>	2 days ago
alex-deploy.yml	Create alexa-deploy.yml	2 days ago
alex-submit.yml	Update alexa-submit.yml	24 days ago
azure-endpoint.yml	Various bug fixes	2 days ago
spreadsheet-gas.yml	Added pipeline tests, added get beta button to spreads...	24 days ago

Figure 7. Screenshot of the repository workflows.

## Alexa skill deployment

The Alexa skill codebase workflow is unique from the other workflows as it has three different workflows, one is a manual trigger only while the two others trigger for updates to the “main” and “dev” branches. The workflows are:

- alexa-deploy
- alexa-beta
- alexa-submit

As of the time of this project there is no official Amazon ASK CLI marketplace Action. Marketplace Actions are typically used for easier workflow development as you do not have to write as much workflow code yourself. No marketplace Action was used, instead the workflows were developed manually. For this the ASK CLI is used to manage the Alexa skill through the command line.

*"The Alexa Skills Kit Command Line Interface (ASK CLI) is a tool for us to manage our Alexa Skills and its related resources. With the ASK CLI, we have access to the Skill Management API, which allows us to manage Alexa Skills through the command line." <https://dzone.com/articles/pipeline-of-an-alexa-skill-with-github-actions>*

Another key thing to note is as this skill is a “custom skill” it means there is no associated AWS Lambda code that is to be deployed with it. As such we are only concerned with deploying the Alexa skill code. The effect of this reveals the youth of the auto deployment platform as we still must pass AWS variables to the ASK CLI even though we do not use them. To work around this the AWS values are simply set as “dummy”.

All these workflows share the step of token replacement. This is done by running a script that replaces placeholders in the template JSON files. This is done before the skill is deployed. The following steps must be done for token replacement:

1. Token replace of the endpoint uri in the skill.json file
2. Token replace of the skill id in ask-states.json
3. Token replace of the skill id in ask-resources.json

These are required as when hosting the codebase in a public repository it is undesirable to leak any of these values and as such they should instead be stored as repository secrets and substituted back in at time of deployment.

## **Alexa-deploy**

This workflow triggers on push events only for the “dev” branch. Also required for it to be triggered is that the push event includes a change in files for the “alexa-skill” directory. This workflow is simply for deploying the Alexa skill. It will mainly be triggered by use of the Google Sheet when questions are exported, as this then updates a file in the “alexa-skill” directory with the new questions and intents which then triggers this workflow to run.

Steps of workflow:

1. Token replacement
2. Install ASK CLI and deploy

This will update the hosted skill on Amazon with the latest code from the repository.



## Alexa-beta

This workflow is only triggered manually. This workflow is also unique as it uses inputs. These are action input parameters where data can be specified for the action to use at runtime. In this case the workflow has a required input of “Tester email” which is the email to send a beta invite to for skill testing and it also has an optional input of “Feedback email”, this is the email which is used for testers to send feedback on the beta. The feedback email defaults to “none”.

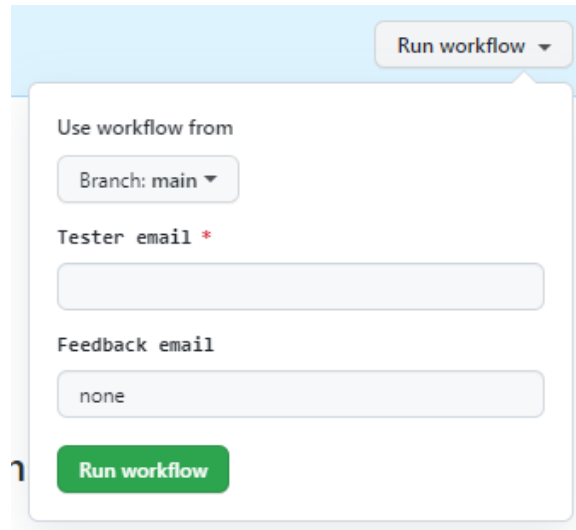
The image shows a web interface for running a workflow. At the top right, there is a button labeled "Run workflow" with a downward arrow. Below this, a modal or dropdown form is displayed. Inside the form, the text "Use workflow from" is followed by a dropdown menu currently showing "Branch: main". Below that, there are two input fields. The first is labeled "Tester email \*" with a red asterisk indicating it is required. The second is labeled "Feedback email" and contains the text "none". At the bottom of the form is a green button labeled "Run workflow".

Figure 8. Interface that appears if ran manually, this form appears for the user to enter the inputs.

An important aspect of this workflow and its inputs is the requirement to mask the inputs. The emails should not be revealed in the console after the workflow runs for privacy. Another output that must be masked is the invitation URL for the beta, if the repository is public anyone could look at the workflow run history, find the link, and join the beta. To achieve this, output from running commands is redirected.

This workflow will mainly be triggered by using the Google Sheet and the “Get Beta” button.

Steps of workflow:

1. Token replacement
2. Install ASK CLI and deploy
3. Enter the tests/skill-beta/ folder, run `beta_initialiser.sh`

The `beta_initialiser.sh` does as follows:

1. Creates beta for skill
2. If not passed “none” as feedback email, adds feedback email to beta
3. Starts beta
4. Invites tester to beta

This script uses the Alexa Skill Management API (SMAPI) which provides skill management operations. If there is a beta already running and started, the create and start command will fail but this is ignored as no easy way for checking beforehand is available in code and it is simple to ignore the failed command.

## Alexa-submit

The “main” branch pushes will run the “alexa-submit” workflow. Also required for it to be triggered is that the push event includes a change in files for the “alexa-skill” directory. This workflow is for deployment and certification submission to Amazon and as such contains the most steps out of any workflow file.

Overall, it runs tests then submits the Alexa skill to certification. This workflow has multiple jobs which each have their own steps. Workflow order:

1. Bring execution permission to be able to execute all the tests
2. Deploy the Alexa skill to the Alexa cloud in the development stage
3. Check if the interaction model uploaded has conflicts
4. Validate Alexa skill before submitting it to certification
5. Submit Alexa skill to certification
6. Store the entire code as an artifact

The tests are expanded on in the Testing section. If any preceding steps fail the whole workflow will stop and fail as that would mean the skill will not be able to be certified.

## **Azure deployment**

The “main” branch pushes will run the “azure-endpoint” workflow. Also required for it to be triggered is that the push event includes a change in files for the “azure-endpoint” directory. This folder contains the Azure Function code.

This workflow required the least development out of all the workflows. The reason for this is as it uses the official Microsoft GitHub Action found on the Action marketplace and no tests are a part of the pipeline.

Steps of workflow:

1. Setup Dotnet environment
2. Resolve project dependencies using Dotnet
3. Run Azure Functions Action

Building of the workflow was all pulled from Microsoft's official documentation at:

- <https://docs.microsoft.com/en-us/azure/azure-functions/functions-how-to-github-actions?tabs=dotnet>

## **Google Apps Script deployment**

The “main” branch pushes will run the “spreadsheet-gas” workflow. Also required for it to be triggered is that the push event includes a change in files for the “spreadsheet-gas” directory. This folder contains the GAS code for the Google Sheet.

This workflow makes use of a marketplace Action, making deployment easier. Also done is a static quality test of the code.

Steps of workflow:

1. Static quality analysis test
2. Push code using Clasp Action

## Journey

At the beginning of development there was only one repository as at that stage I was unsure of storing the Alexa skill codebase and the Google Apps Script was not yet in the scope. The first repository was for the Azure Function code. As I was already familiar with the process for automatic deployment when hosting Azure code in GitHub and with Microsoft's great documentation it was also the first setup in the CI/CD pipeline. The first setup used this guide for continuous deployment by Microsoft:

- <https://docs.microsoft.com/en-us/azure/azure-functions/functions-continuous-deployment>

At first, I was getting errors when attempting to use the workflow. The error was "Invalid version format".

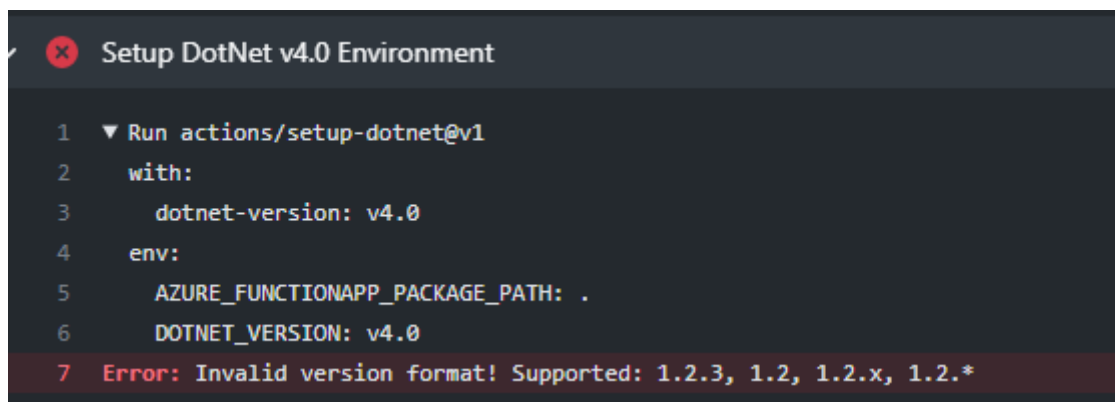


Figure 9. Error log in workflow run.

This was due to a mismatch in the Function code .NET version and the version specified in the workflow. To fix it I changed DOTNET\_VERSION: 'v4.0' to DOTNET\_VERSION: '3.1.x' in the workflow file.

Next I was getting an issue that resulted in it still deploying but a significantly increased deploy time. Upon research found a GitHub issue thread with the same issue:

- <https://github.com/Azure/functions-action/issues/86>

Looking through the thread I found a fix which was to revert the used Actions version to v1.3.2. After this the auto deployment was successfully set up.

The next repository to be set up was the Google Apps Script. Downloading and deployment of GAS code is achieved by using Google's Clasp which allows developing Apps Script projects locally:

- <https://github.com/google/clasp>

To set up the workflow for CI/CD a marketplace Action was found. From this point I learned to store sensitive variables as secrets in the repository to avoid being leaked. This was required for passing the needed values for the Action.

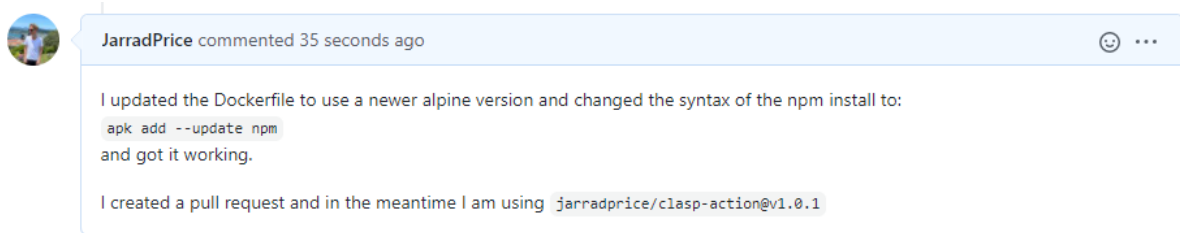
I did get error on attempting to use the following action:

- <https://github.com/marketplace/actions/clasp-action>

I believed it was to do with the Actions Docker setup and found an issue thread:

- <https://github.com/daikikatsuragawa/clasp-action/issues/3>

I did not want to use their fix of downgrading the Clasp version so instead went down the path of learning to understand the Docker file that the Action uses and seeing if I could fix the issue. After learning the workings of it, I forked the Action and modified the Docker file to use a newer alpine version and different npm install syntax. This worked. I then commented on the thread and created a pull request.



*Figure 10. Comment on GitHub issue thread.*

This request was later accepted, and the issue thread was closed.

At this point the final architecture diagram was visualised and it was discovered that I could host Alexa skill code on GitHub and get it to auto deploy to Amazon. On looking at marketplace Actions for Alexa, none seemed necessary as it was simple enough to install the ASK CLI and use it.

The following was used to learn how to set up CI/CD for the Alexa repository:

- <https://dev.to/aws-builders/devops-your-alexa-skill-3hom>

Originally, I was stuck on an issue for an extended amount of time because I was not passing AWS values as environment variables even though I was not using AWS to host my skill's endpoint. Tried at first to upload a config file for use then after that did not work tried modifying the CLI version used. It kept giving me the same error of "CliFileNotFoundError: File /home/runner/.ask/cli\_config not exists.". This was then I added the dummy variables for the AWS values, and it worked.

Once all repositories were built, I realised that it would be cleaner and easier if they were stored in one repository. I then put the GAS, Azure and skill code in a single repo with individual workflows. At this point they also had to implement workflows triggering only if the folder they are concerned about is changed. I also had to modify working directories of the workflows. This final step was doing all token substitution and needed repository secrets as the new repository was to be public. This substitution was only needed for the skill code as it had private values in the json files.

Substitution used placeholders with the syntax of:

- `#{name}#`

The replacement commands can be seen in Figure 11. The commands can be understood as:

- `sed -i "s/#{EndpointUri}#/${{ secrets.SKILL_ID }}/g"`
  - **Red** - placeholder to find.
  - **Green** - replacement string, retrieved from repository secret.

```
# Must do a token replace of the skill id and endpoint uri
- name: Find and replace skill.json
  run: find skill.json -type f -exec sed -i 's/#{EndpointUri}#/${{ secrets.ALEXA_ENDPOINT_URI }}/g' {} \;
  working-directory: ./alexa-skill/skill-package
- name: Find and replace ask-states.json
  run: find ask-states.json -type f -exec sed -i 's/#{SkillId}#/${{ secrets.ALEXA_SKILL_ID }}/g' {} \;
  working-directory: ./alexa-skill/.ask
- name: Find and replace ask-resources.json
  run: find ask-resources.json -type f -exec sed -i 's/#{SkillId}#/${{ secrets.ALEXA_SKILL_ID }}/g' {} \;
  working-directory: ./alexa-skill
```

*Figure 11. Token replacement commands.*

# Amazon Alexa

## Introduction

The reasons for picking Amazon as the assistant have already been introduced in the Requirements section. Other advantages not yet mentioned that this platform offers:

- Large amount of documentation
- Free creation
- Developer tools

Also detailed in this section is how an Alexa skill works and how it can be accessed.

## Large amount of documentation

Amazon has many existing resources including, official online developer documentation, guides, tutorials, webinars and more. Also upon making an Amazon developer account Amazon will regularly send you emails with new upcoming resources and documentation. Amazon has proven themselves as a company who cares and upkeep their documentation knowledge bases. Alexa developer tools have been around a long time and has the largest market share [14] so there is more documentation and discourse.

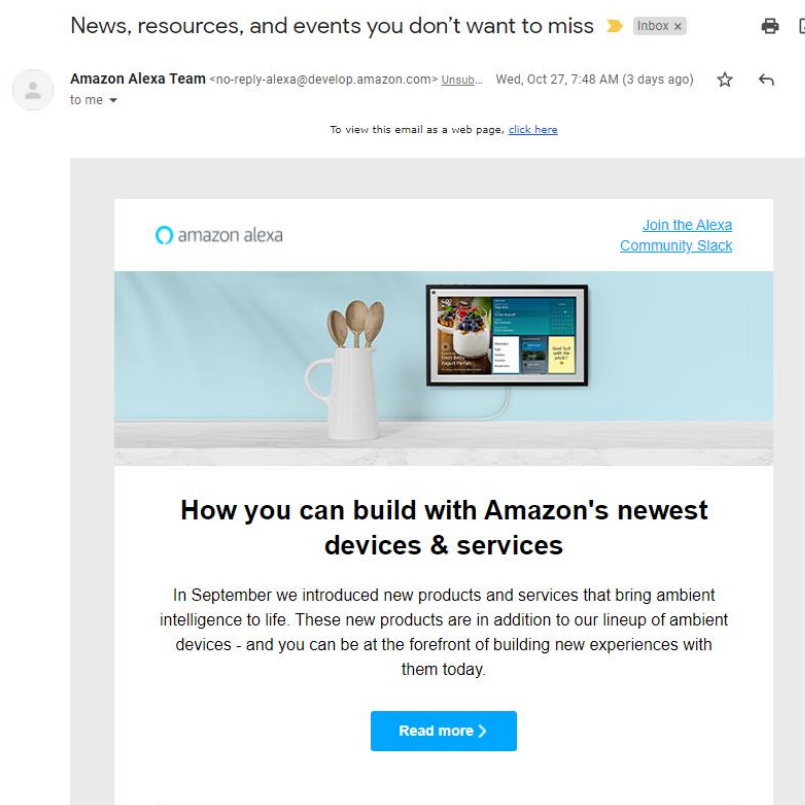


Figure 12. A previous email that Amazon has sent out to developers.

Due to this community support and early arrival to the public VA developer scene there is also an abundance of online community created question posts and tutorials.

## Free creation

Alexa development is also advantageous as it is free to create a new Alexa skill. The only thing that will incur a cost is the hosting of the skill's service endpoint where intention responses are returned. This endpoint is also up to the developer, you can use Amazon's own AWS Lambda or provide your own HTTPS endpoint.

## Developer tools

Alexa has powerful tools for creating skill's without having to do any programming, but they also provide robust tools for management through code as well. Their main codeless tool is the online developer console. This is an in-browser console where you can create, manage, test, and publish skills. It also offers analytics on the skill. This is great for quick easy access and for beginners. Their programmatic tools include the ASK Command Line Interface, Visual Studio Code Toolkit, and Skill Management API. These allow powerful management for more advanced developers who wish to manage their skill external from the developer console and opens the possibility of automation.

## How an Alexa skill works

From Amazon's official site on how a skill works [15]:

1. To launch the skill, the user says, "Alexa, open Hello World."
2. The Alexa-enabled device sends the utterance to the Alexa service in the cloud. There, the utterance is processed via automatic speech recognition, for conversion to text, and natural language understanding to recognize the intent of the text.
3. Alexa sends a JavaScript Object Notation (JSON) request to handle the intent to an AWS Lambda function in the cloud. The Lambda function acts as the backend and executes code to handle the intent. In this case, the Lambda function returns, "Welcome to the Hello World skill."



Figure 13. Diagram of skill steps, from <https://developer.amazon.com/en-US/alexa/alexa-skills-kit/start>.

In this project's case the Lambda Function is instead an Azure Function. The two main components are the interaction model and the application logic. The interaction model is the voice user interface, it is what recognises user input, processes the speech and maps to a stored intention in the system. Once it decides what the user input maps to it then sends the request to the application logic, the Azure Function, which processes and sends back a response which Alexa then plays the audio for.



## Alexa devices

The Alexa skill of this project can be used on any Alexa enabled device. More and more devices are constantly becoming available, and these can be with screens or without screens. In 2019 it was announced by Amazon that 100 million Alexa devices have been sold [16]. The entry point is lowered by the fact that Alexa devices are not just made by Amazon, any company can make an “Alexa enabled” device. Further increasing accessibility Alexa can also be installed on mobile devices with the Amazon Alexa app which also provides access to skills.

This project has been developed for primary use with smart speakers and as such does not have extra functionality that a screen device can facilitate such as images.

## System design

The design freedom for the developer for a question-answer centric skill such as the one for this project is mainly centred around the conversation experience with Alexa. The biggest design challenges come in the form of generating the questions that trigger an intent and the response that Alexa should respond with.

One of the key parts of Amazon’s design guide [17] is a loop from ideation through to publishing and into improving the skill. They stress the importance of knowing how to achieve the user’s goals.

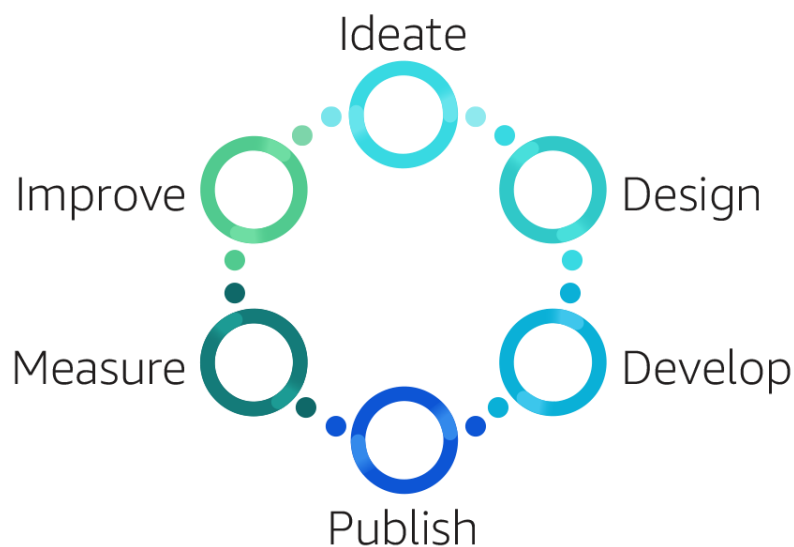


Figure 14. The Alexa skill building journey, from <https://developer.amazon.com/en-US/alexa/alexa-haus>.

One of their design principles for conversation is to “be natural” [18]. This means that responses should be relatable and not overload the user, the response would be written the way you speak. This also comes into account when writing what questions will trigger an intent.

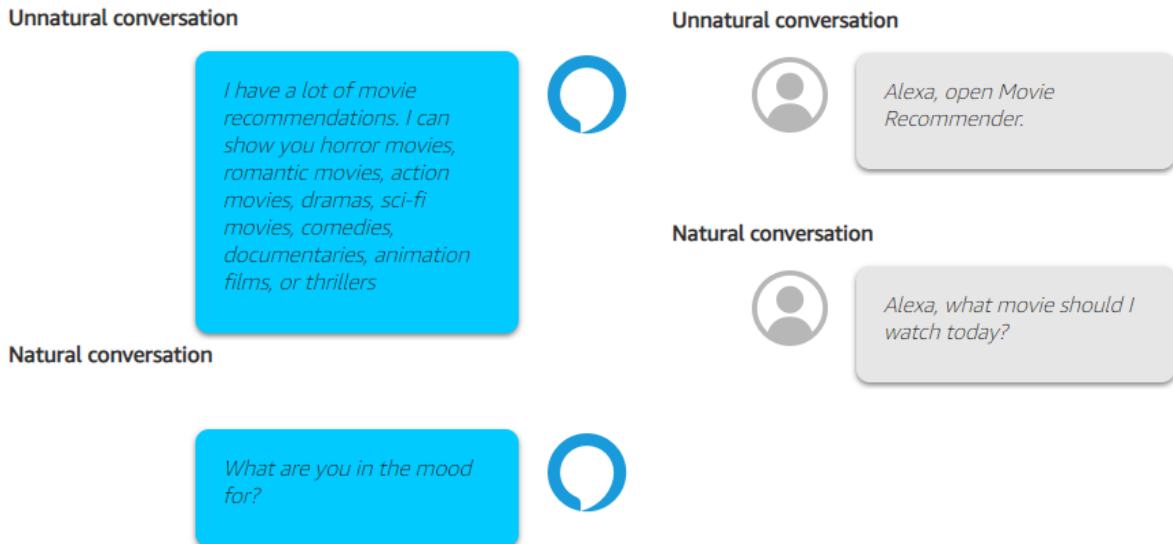


Figure 15. Examples of natural conversation, from <https://developer.amazon.com/en-US/alexa/alexa-haus/design-principles>.

This intention of natural language is admittedly an aspect of the final project that did not receive a lot of focus due to most of the development time required to be on the building of the individual platforms and the information flow between them. But due to the Google Sheet management hub, updating of the question and responses is an easy process so the “improve” step in the loop displayed in Figure 14 is very achievable. Ultimately the quality of the initial interaction with Alexa to get a desired solution response is up to the maintainer/s of the Google Sheet. Not included in this project are things that would aid the natural conversation after the initial request such as, ambient and proactive experiences, adapting to the user, and personalised and contextual conversation. The Alexa skill in this project in its current state only serves as a simple question-answer tool.

Intent	Questions			
DEVICE NAME	istat			
INR_OVER_RESULT	Actions for PoCT INR of over four			
	What do I do if I get an INR result greater than four			
	My INR result is over four			
	The INR result is more than four			
	The INR is high			
	The INR is above four			
INR_BLOOD_DROP	Which blood drop should I use			
	What blood should I use for the INR			
	How do I obtain the blood for the INR			
	Which finger should be used to get the blood drop			
	What is the method for getting blood for the INR			
INR_NOFIT_RESULT	What should I do if I get an unexpected high or low result that doesnt fit the clinical picture			
	The INR result does not fit the clinical picture			
	My INR result doesn't fit the picture			
	The INR result is different to the clinical picture			
	The INR result is higher than the clinical picture			
	The INR result is lower than the clinical picture			

Figure 16. Screenshot of questions that have been put into the Alexa skill.

Intent	Answer				
DEVICE NAME	istat				
INR_OVER_RESULT	Step 1 Immediately recheck PoCT INR. Step 2 If INR still over 4.0, collect blood for la				
INR_BLOOD_DROP	The first drop of blood should be used to achieve an accurate INR result. If an insuffic				
INR_NOFIT_RESULT	In the first instance, repeat the test on a fresh sample from another finger. Make sure				
INR_LAB_AGREE	INR results across different methods may vary due to the different thromboplastins us				
INR_LAB_DIFFERENCE	When comparing INR results from stable anticoagulated patients, they should be with				
INR_MONITOR_ANTDAB	Routine lab or PoCT INR testing is of no benefit in monitoring Dabigatran drug dose.				

*Figure 17. Screenshot of answers that will be spoken by the Alexa skill.*

The questions-answer pairs that are included in the initial project have been created with natural language in mind.

## Journey

The first step taken was creating an Amazon Developer account. This account allows you to:

- Develop Alexa skills
- Develop Android apps and put on the Amazon app store
- Build Amazon reordering experiences
- Access AWS tools, documentation, and sample code

I was just interested in the Alexa development. The next step was to follow some basic official tutorial modules from Amazon. I was able to gather enough of an understanding to create the first draft of the skill I wanted from the module located at:

- <https://developer.amazon.com/en-US/alexa/alexa-skills-kit/get-deeper/tutorials-code-samples/build-an-engaging-alexa-skill>

Early on I decided not to prototype using Amazon Lambda Functions as the endpoint. There is a great deal more tutorials using this method, but it is not what I wanted the end product to be and would have to change over eventually so thought it best to get the core information flow working.

By not using Lambda as the endpoint the skill is identified differently. When using your own custom web service that accepts requests from and sends responses to the Alexa skill your skill is known as a “custom skill”. For understanding this better the following official documentation can be used:

- <https://developer.amazon.com/en-US/docs/alexa/custom-skills/host-a-custom-skill-as-a-web-service.html>
- <https://developer.amazon.com/en-US/docs/alexa/custom-skills/understanding-custom-skills.html>

This just requires you to follow the requirements for the web service to be functional. The requirements are:

1. The service must be accessible over the internet
2. The service must accept HTTP requests on port 443
3. The service must support HTTP over SSL/TLS, using an Amazon-trusted certificate. Your web service's domain name must be in the Subject Alternative Names (SANs) section of the certificate. For testing, you can provide a self-signed certificate. For more information, see About the SSL Options
4. The service must verify that incoming requests come from Alexa. For more information, see Verify that the request was sent by Alexa
5. The service must adhere to the Alexa Skills Kit interface

When in this mode you only deploy skills and models only, and then the custom web service separately (in this case Azure Functions) because there will be a separate project directory and deployment process for both.

So, the next step was finding resources that detail an Alexa and Azure Functions connection. Luckily the endpoint requirements are simple enough that it is not an exorbitant process to use a platform that is not Amazons, and I also was able to find guides for this sort of architecture:

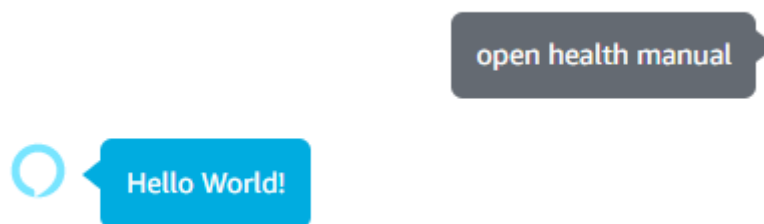
- <https://techcommunity.microsoft.com/t5/windows-dev-appconsult/build-your-first-alexa-skill-with-alexa-net-and-azure-functions/ba-p/317930>
- <https://levelup.gitconnected.com/build-your-first-alexa-skill-with-alexa-net-and-azure-functions-aa5a7ff75b55>
- <https://dev.to/tsjdevapps/alexa-goes-azure-how-to-write-a-skill-with-azure-functions-4hip>

Through these I was exposed to the Alexa.NET package. This provided a helper library for working with Alexa in C#. This introduced objects native to the skill workflow such as, SkillRequest, SkillResponse, IntentRequest, etc. This package can be found here:

- <https://github.com/timheuer/alexa-skills-dotnet>

This package makes adhering to requirement 5 of a custom web service, “The service must adhere to the Alexa Skills Kit interface” easier to meet.

With that the first skill was working with a launch request and a “Hello World!” response.



*Figure 18. First interaction testing the skill.*

Next step was creating some questions for Alexa to answer. To maintain consistency, I created some simple rules for an intent name to follow:

#### ISTAT\_INR\_OVER\_RESULT

- **Device name**
  - The intent starts with the name of the device it is for. This must be 5 characters long and only consist of letters.
- **Question identifier**
  - The question identifier is the shorthand for what the question is about. This can only consist of letters/words.
- **Separators**
  - To separate the device name from the question identifier and words from each other in the question identifier underscores are to be used.

Another important decision was the “Invocation Name” for the skill. Initially I had set the launch name for the skill to be “Health Manual”. But I was sometimes getting issues where Alexa would instead launch a skill called “Health Tips”. To avoid this, I changed the name to be more distinctively unique, becoming “point of care device manual”. The end product of the project has the invocation name being able to be changed in the Sheets management hub.

# Azure Functions and Cosmos

## Introduction

For the endpoint and database solution for this project Microsoft's platforms were chosen. To facilitate the endpoint the serverless platform Azure Functions was chosen. To facilitate the storage solution Azure Cosmos DB was chosen as the database.

These platforms were chosen due to:

- Scalability
  - Azure uses compute-on-demand. This is very relevant to the use case of this project of a service that will have requirement times that cannot be exacted, there may be large amounts of use or none over time periods. When there is more demand Azure will automatically allocate more resources to the services. This also leads it to being more cost effective as you don't pay for what you don't need.
- CI/CD support
  - Microsoft has created documentation for and actively supports continuous deployment [19]. This integration makes it easy for code updates to automatically trigger deployment to Azure. By using this form of deployment, it makes it easy for teams to work on code as you maintain a single source of truth. The use of CI/CD also makes it easy to distribute code to others and integrate it into their pipelines.
- Support between functions and cosmos
  - As they are both Microsoft solutions and are both under the Azure line of services there is extensive support and documentation for combining Functions and Cosmos. Azure provides native integration with Cosmos and Functions, it allows the option of creating triggers, bindings straight from an Azure Cosmos DB account. Just like Functions billing, Cosmos also offers payment in “Severless” form where you pay for what they call “request units” [20]. This makes an easy match between Functions and Cosmos with their handle for traffic bursts where there is not sustained traffic, exactly the scenario for this project.
- Developer and client experience with the platform
  - Both the developer and the client have previously used Microsoft's service platforms and as such makes development, deployment, and handover quicker and easier.

## System design

The final Azure services structure includes two Functions in one Function app and a Cosmos DB. The structure and what components are connected can be seen in Figure 19.

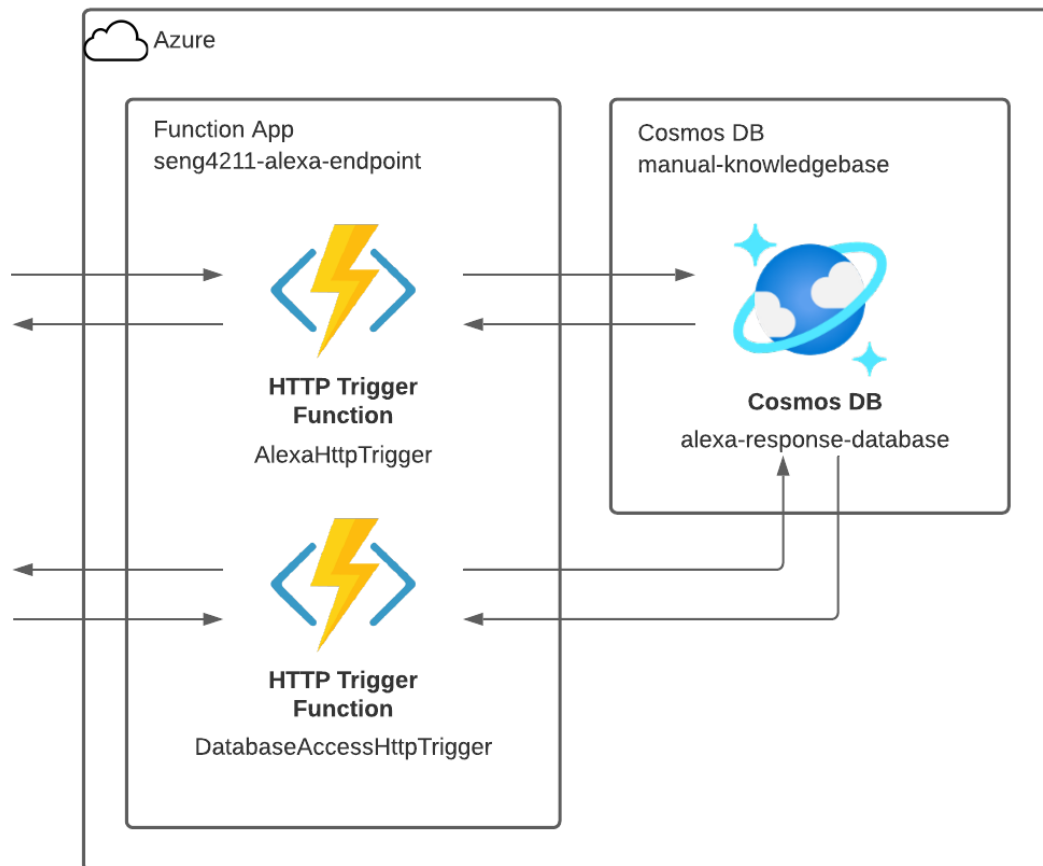


Figure 19. Diagram of Azure components.

The two Functions are HTTP triggers, meaning they run when a HTTP request is made to them. Their configuration is set to require their Function authorisation key to be passed with the request for it to be accepted.



## **AlexaHttpTrigger**

This serves as the endpoint for the Alexa skill. The skill sends a request to this Function, and it then parses the request and queries the database for the parsed intent.

The Function does the following when receiving a request:

1. Check the passed skill ID to verify the request came from our skill, if not drop the request
2. Start parallel task to get all database JSON as to decrease response time
3. Convert the passed JSON to an Alexa.NET SkillRequest object for easy management
4. Get the request type
  - i. If launch request send back success
  - ii. If session end request send back success and end session
  - iii. If intent request continue parsing

If intent request:

5. Wait for database JSON task to be complete then parse intent query
6. Get intent name as well as device ID from the passed intent
7. Get device JSON using device ID
8. Get intent JSON using intent name
9. Get response from intent JSON
10. Return response

### DatabaseAccessHttpTrigger

This is used for accessing the Cosmos DB outside of the Azure cloud. This is utilised by the Google Sheet to get, update, and add database items. The request must include JSON in the body to be accepted. The form that the JSON must follow is displayed in Figure 20.

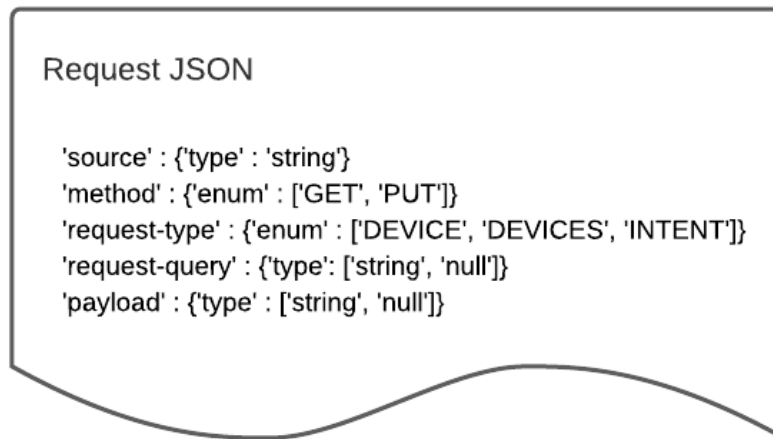


Figure 20. Structure of request JSON for DatabaseAccessHttpTrigger.

- source
  - Where the request has come from.
  - When from the google sheet will be “Google App Script”.
- method
  - If getting an item from the database or inserting/updating.
  - If a PUT request function does not use request-type attribute, only a device object can be sent to be put into the database, no support for sending multiple devices at once or a single intent to be updated.
- request-type
  - The type of object to get/put.
  - DEVICE - json of one device.
  - DEVICES - json of multiple devices.
  - INTENT - json of one intent from a device.
- request-query
  - When a DEVICE request-type or PUT method used to send id of device.
  - When a DEVICES or INTENT request-type used to send SQL query.
- payload
  - When a PUT request this is the json of the object to insert into the database.

### Alexa-response-database

This holds the JSON for the health devices which contains each device's intents mapped to responses.

The structure of the document database is displayed in Figure 21.

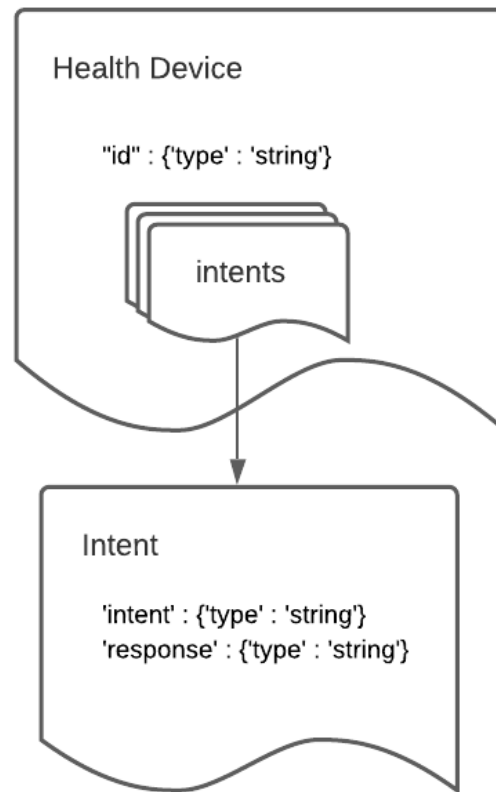


Figure 21. Structure of Cosmos DB JSON.

The structure is simplistic but serves the purpose of the project. In the future the structure could be easily updated to support storing more information if expanding use cases of the system.

## Journey

The Azure Function was very simple to begin with, just containing one Function HTTP trigger. This trigger was for the Alexa skill endpoint. First functionality tests had the Function app running locally. This was achieved using an extension called “ngrok”:

- <https://marketplace.visualstudio.com/items?itemName=philnash.ngrok-for-vscode>

This allowed exposing the local server for Alexa to access as an endpoint. With this from the outside, any service can reach the Azure Function exposed by the local web server on the used port using the address provided by ngrok. I then copied this address and put it into the Alexa developer console as the HTTPS endpoint adding “/api/alexa” to the end for specifying the trigger.

Service Endpoint Type

Select how you will host your skill's service endpoint.

☐ AWS Lambda ARN (Recommended) ⓘ

☒ HTTPS ⓘ

Default Region ⓘ (Required)

Figure 22. Alexa developer console service endpoint configuration.

This then worked and I could get local development and testing working.

The first device added to the database was the iSTAT. I originally had trouble deciding how to connect the Function and the Cosmos DB. I first wanted to make use of input bindings:

- <https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-cosmosdb-v2-input?tabs=csharp#example>

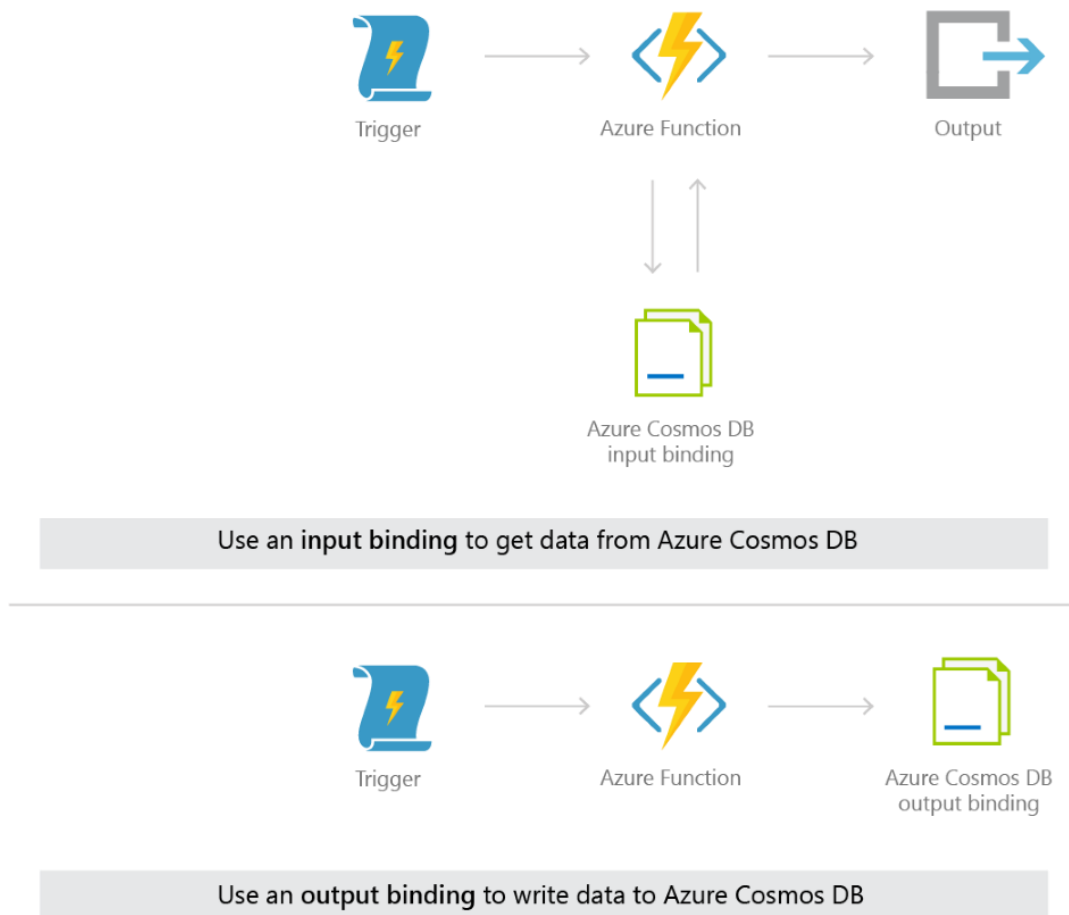


Figure 23. Cosmos bindings for an Azure Function, from <https://docs.microsoft.com/en-us/azure/cosmos-db/sql/serverless-computing-database#why-choose-azure-functions-integration-for-serverless-computing>.

At that point in time, I did not fully understand how to utilise the bindings and only wanted one HTTP trigger, so I discarded this method. On reflection, using input and output bindings for connecting to the Cosmos DB would have been an acceptable route for the DatabaseAccessHttpTrigger.

Instead of using bindings I connected to Cosmos more “conventionally”. This Microsoft tutorial was followed:

- <https://docs.microsoft.com/en-us/azure/cosmos-db/sql/sql-api-get-started>

This worked well and the same methodology was used in the final implementation. To test the Alexa endpoint without having to use the Alexa skill an extension called “Thunder Client” was installed:

- <https://marketplace.visualstudio.com/items?itemName=rangav.vscode-thunder-client>

At this point I decided what the representation of a database item would be. It was decided to be a JSON document for each device. The draft JSON object structure was:

```
{
  "id": "machine_id"
  "faqs": [
    { "question": "answer" }
  ]
}
```

This quickly changed to the final structure of:

```
{
  "id": "istat",
  "intents": [
    {
      "id": "INR_BLOOD_DROP",
      "response": "Step one, immediately re-check PoCT INR. step 2 if INR still over 4.0, collect blood for laboratory INR. step 3 consult with the patient's Doctor or a Haematologist immediately."
    }
  ]
}
```

The only difference in the final design is small naming changes.

The next step was to start getting responses for specific intents, as to query the correct response. Cosmos supports a SQL query syntax so that was utilised by using this link as reference:

- <https://github.com/Azure/azure-cosmos-dotnet-v3/blob/master/Microsoft.Azure.Cosmos.Samples/Usage/Queries/Program.cs#L189-L212>

At this point, I learnt that best practice is to verify that the request came from your skill:

- <https://developer.amazon.com/en-US/docs/alexa/custom-skills/handle-requests-sent-by-alexa.html#request-verify>

This was done by copying the skill id into the code and checking the contained id in the request against it.

Once the final architecture was envisioned a new HTTP trigger was created, this being DatabaseAccessHttpTrigger. The intended use of this Function was built as:

- Getting responses in the endpoint
- Getting entries in the apps script
- Creating the database
- Creating entries/documents

I also made a “CosmosDBService” class which acted as an intermediary between Cosmos and the trigger Function. This class was then used by both triggers when it was decided for the Alexa endpoint trigger to get the database JSON itself as opposed to using the DatabaseAccessHttpTrigger to get the database documents. This was done to reduce response time for AlexaHttpTrigger.

The final alterations to the Functions were to store private values as application settings that can be retrieved at runtime in code. This is required for security when publicly hosting code and as to be able to be deployed by a different user. The needed settings are variables for Cosmos connection and one for the Alexa skill id when verifying who the request came from in the AlexaHttpTrigger.

# Google Apps Script and Sheets

## Introduction

The choice of Google Apps Script and Sheets grew naturally from development, this platform was not in the first scope of the project.

Advantages of Google Sheets:

- Ubiquitous
  - Google is a popular platform with anyone who has a Gmail account having Google Drive storage where a Google Sheet can be stored.
- Ease of sharing and accessible
  - A Google Sheet can be shared at any time by copying its share link. This is then available in any browser.
- Extendable
  - Using Google Apps Script allows extending Sheets functionality.

## System design

The Sheet contains the following custom added functionality:

- “CONFIG” sheet to place variables for the environment.
- “EXPORT TOOLS” button in the toolbar to access the custom GAS functions.
- “Export Questions” button which upon selection exports the question-intent mapping pairs in the “Questions” sheet as a JSON which is then committed to the “dev” branch of the repository which triggers the auto deployment updating the Alexa skill.
- “Export Answers” button which opens a sidebar where answer-intent mapping pairs can be exported to the Cosmos DB; the “Answers” sheet is used to assist that process.
- “Get Beta” button which prompts for the testers email and optionally a feedback email. This sends a beta invite for testing the Alexa skill to the tester's email.
- “Create Merge Request” button which creates a merge request to merge the “dev” branch into “main”.

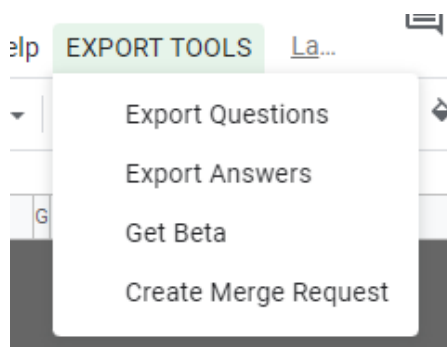


Figure 24. Screenshot of the EXPORT TOOLS options.

Usage of the Sheet should follow down that list of buttons. First the user should create new questions for a device with an intent name. They should then export the questions, triggering the CI/CD pipeline to update the skill.



	A	B	C	D	E	F
1	<b>Instructions</b>					
2	<b>ADD INTENT:</b> If adding to a device, append row					
3	<b>ADD DEVICE:</b> If creating a new device leave space after last device. New device must start with DEVICE NAME row.					
4						
5	Should aim for a <b>MINIMUM</b> of 5 phrasings that trigger an intent					
6	<b>Formatting Rules</b>					
7	<b>DEVICE NAME</b> must be 5 characters long and only consist of letters					
8	<b>Intents</b> can only consist of letters and words must be separated by underscores, no duplicates allowed					
9	<b>Questions</b> can consist of only Unicode characters, spaces, periods for abbreviations, possessive apostrophes, curly braces and hyphens.					
10						
11	<b>Intent</b>		<b>Questions</b>			
12	DEVICE NAME		istat			
13	INR_OVER_RESULT		Actions for PoCT INR of over four			
14			What do I do if I get an INR result greater than four			
15			My INR result is over four			
16			The INR result is more than four			
17			The INR is high			

press me!  
**Validate Cells**

= valid  
 = invalid, check rules

+ ☰
READ ME ▾
CONFIG ▾
Questions ▾
Answers ▾

Figure 25. Screenshot of the Google Sheet Question sheet.

	A	B	C	D
1				
2	<b>Key</b>			
3	Exported		<- remember to mark as exported when a	
4				
5				
6	<b>Intent</b>	<b>Answer</b>		
7	DEVICE NAME	istat		
8	INR_OVER_RESULT	Step 1 Immediately recheck PoCT INR. Step 2 If INR still c		
9	INR_BLOOD_DROP	The first drop of blood should be used to achieve an accur		
10	INR_NOFIT_RESULT	In the first instance, repeat the test on a fresh sample from		
11	INR_LAB_AGREE	INR results across different methods may vary due to the c		
12	INR_LAB_DIFFERENCE	When comparing INR results from stable anticoagulated pi		
13	INR_MONITOR_ANTDAB	Routine lab or PoCT INR testing is of no benefit in monitor		
14	CART_ANALYSIS_TIME	The analysis time for each cartridge is, cTnI, BNP, beta hC		
15	CART_STORAGE	Cartridges should be stored at temperatures between 2 an		
16	CART_PREP	Equilibrate a single cartridge for 5 minutes or a box of carti		
17	CART_STABLE_TIME	iSTAT cartridges have a 14 day room temperature expiry e		
18	CART_RETURN	Do not return cartridges to the refrigerator after room temp		
19	CART_BROKEN	If the pouch has been punctured, the cartridge should not l		
20	CART_HANDLING	Do not touch the electrodes at the top of the cartridge and		
21	PRE_REQUIRED_INFO	Operator ID, Patient ID, Cartridge Lot Number		
22	DEVICE_POSITION	On a flat surface.		
23	CART_LOCKED	Do not try to remove the cartridge, as this will damage the		
24	NOFIT_RESULT	Retest by using a fresh sample		
25	SCREEN_ON_TIME	2 minutes or, if Enter Cartridge is showing ENTER CARTR		

**Update Database Responses**

istat	abln	gemfi
<b>Intent</b>		
<b>Response</b>		
INR_OVE R_RESUL T	Step 1 Immediately recheck PoCT INR. Step 2 If INR still over 4.0, collect blood for laboratory INR. Step 3 Consult with the patient's Doctor or a Haematologist immediately.	
INR_BLO OD_DRO P	The first drop of blood should be used to achieve an accurate INR result. If an insufficient sample was collected the first time a different finger should be used to collect the sample. Care should be taken not to squeeze the finger to obtain the drop of blood.	
INR_NOF IT_RESU LT	In the first instance, repeat the test on a fresh sample from another finger. Make sure that the quality control is within the target limits. If the repeated test result is the same, then a sample should be sent to the laboratory for verification. Notify the medical officer immediately.	
INR_MON ITOR_AN TDAB	Routine lab or PoCT INR testing is of no benefit in monitoring Dabigatran drug dose.	
CART_A NALYSIS _TIME	The analysis time for each cartridge is, cTnI, BNP, beta hCG cartridges are 10 minutes. CG4, CG8plus, CHEM8 cartridges are 2 to 3 minutes. PT INR cartridges are dependent on clotting time	

+ ☰
READ ME ▾
CONFIG ▾
Questions ▾
Answers ▾

Figure 26. Screenshot of the Google Sheet Answers sheet.

Next the user will go to the “Answers” sheet and add a new response for the newly created intent. They should then export the answer, this will update the database.

After this the Alexa skill and the database will be updated with the new intent. The user will then select “Get Beta” to test the skill. This will send a link to the user's entered email which is used to enable the beta skill on their Amazon account.

After they have tested the skill and deem it production ready, they select “Create Merge Request”. This then requires the repository maintainer to accept the merge request, updating the skills intents and triggering the skill certification CI/CD pipeline. These steps can also be seen in this video:

- <https://youtu.be/PokfskWi3WA>

The GAS behind code for the Sheet follows this structure:

- spreadsheet-gas
  - Services
  - Sidebar

The “Services” subfolder contains three JavaScript files. One is for holding functions to facilitate connection to the Azure Function API. Another is for holding functions to commit to GitHub and the last is for holding functions to facilitate interaction with the Sheet.

The “Sidebar” subfolder contains HTML files for the sidebar that appears in the Sheet when exporting answers to the database.

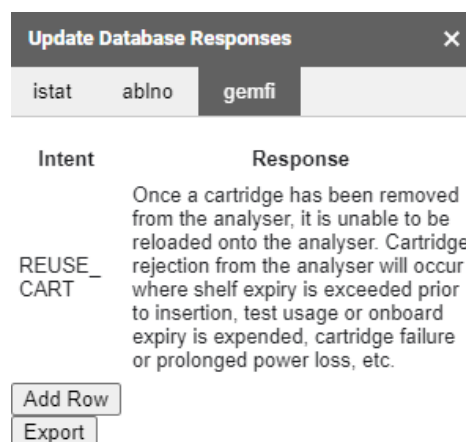


Figure 27. Screenshot of custom sidebar in Google Sheet.

There are three HTML files. One is for holding the JavaScript code used by the sidebar, one is for the base sidebar HTML document and the last is for the styling of the sidebar. The JavaScript and styling html files are inserted into the base HTML file by use of GAS specific syntax.

In the base directory of “spreadsheet-gas” is three JavaScript files.

- Main.js
  - Initialises the toolbar button in the Sheet to start functions. Gets global configuration values from “CONFIG” sheet.
- global.js
  - Holds variables used throughout the JavaScript functions. Initialized in Main.js.
- GitHubAPIRequest.js
  - Used for holding the GitHub API functions.

## Journey

The Google Sheet was first created to store question-answer pairs. It was originally just used to keep track of what has been implemented in Alexa via colour coding of the cells. A green cell meant that it has been put in the skill.

After using this I next had the thought of trying to find a method where I could automatically update the Sheet with the database items and add new items from the sheet. After research I found one method which uses the Cosmos REST API:

- <https://docs.microsoft.com/en-us/rest/api/cosmos-db/querying-cosmosdb-resources-using-the-rest-api>
- <https://h-savran.blogspot.com/2019/02/cosmos-db-rest-api-with-postman.html>

Then within Google Apps Script code for the Sheet I could make the HTTP calls to the API. I spent a large deal of time working out how to correctly build the required authorisation header that must be passed in the request using the Apps Script environment. Many Stack Overflow threads were scoured over to figure it out:

- <https://stackoverflow.com/questions/63592490/sha-256-encryption-within-google-sheets-using-app-scripts>
- <https://stackoverflow.com/questions/63406095/how-to-use-node-js-buffer-in-google-apps-script>
- <https://stackoverflow.com/questions/45027112/hmac-signing-in-google-apps-script-when-secret-and-digest-are-base64-encoded>
- <https://stackoverflow.com/questions/30369714/hmac-in-node-js-crypto-vs-google-apps-script-gas>
- <https://stackoverflow.com/questions/49251949/google-app-script-vs-php-in-encoding-base64/49267366#49267366>

The solution made use of the Apps Script specific encoding functions. This GitHub gist was also helpful in forming the full functionality:

- <https://gist.github.com/sajeetharan/c2c1fbc48bf24e3b321323b34232f5a8>

Though not used in the final project GAS code if others ever wish to accomplish the same thing the code for generating the token is in the appendix - GAS Cosmos Authorisation Token Code.

This then enabled the Sheet to import and export database JSON from the cells. This code was used for getting JSON from a sheet:

- <https://gist.github.com/pamelafox/1878143>

This method of accessing the database was discarded once the DatabaseAccessHttpTrigger Function was built, as HTTP calls were instead made to that. This greatly simplified the process as the requirements of the request were fully up to me. Authorisation was changed to simply using the Function key which is pulled from the “CONFIG” sheet and putting it in the URL as a parameter.

The next step was wanting to use a custom HTML screen to modify the database as opposed to the sheet’s cells. This was desired as it avoided handling errors when the database is imported and potentially overwrites changes that were not exported in the sheets cells. I ended up using a custom sidebar, which is a user-interface element that GAS provides for use. The sidebar offers the advantage of easily being able to copy the values from the “Answers” sheet cells into the sidebar table cells.

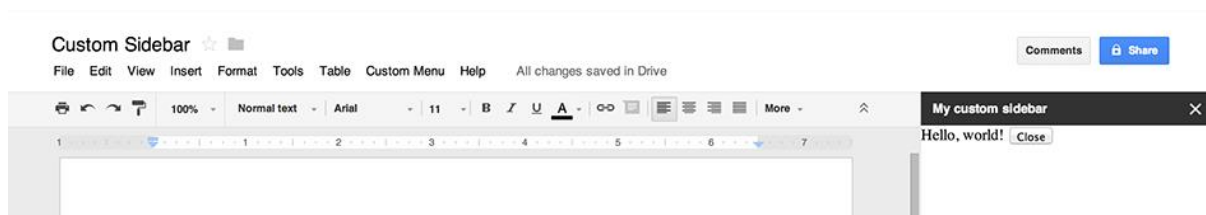


Figure 28. Custom sidebar example in Google Sheets, from [https://developers.google.com/apps-script/guides/dialogs#custom\\_sidebars](https://developers.google.com/apps-script/guides/dialogs#custom_sidebars).

I then created the HTML table in the sidebar which is where the imported JSON occupies and can be modified or added to. The JSON was easy to transpose into the table as it is from a known structure in the database.

At this time validation checking was added so the user cannot export unless the inserted text follows the format rules. This is required to stop any values that break the JSON syntax or are not accepted as a response by Alexa.

The final change to get the key functionality as a management hub was to implement pushing updated questions to the GitHub repository. This took advantage of the CI/CD pipeline which will then update the Amazon hosted skill. The code for pushing to GitHub was directly taken from:

- <https://gist.github.com/pamelafox/ea0474daa137f035b489bf78cc5797ea>

This involved generating a GitHub personal access token to be used by the GAS code.

*“Personal access tokens (PATs) are an alternative to using passwords for authentication to GitHub when using the GitHub API or the command line.”*

*<https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>*

At this point the “CONFIG” page was fully built out to hold all the needed private variables.

The final touch to the Sheet was adding the functionality of getting the skill beta and creating a merge request. These both used the GitHub API with HTTP requests to start repository workflows.

# Testing

Testing of the project through and after development was achieved using the following methods:

- Beta testing of the Alexa skill
- Checking if the uploaded skill interaction model has conflicts
- Validate skill
- Submit skill to Amazon certification
- Static quality analysis test on the GAS spreadsheet code

With the integration of testing and automatic deployment, there will seldom be someone directly watching the tests run. As such logs are kept in the “Actions” page of the repository. From here the status of each workflow can be seen, showing if it was successful or not.

The screenshot shows the GitHub Actions interface for the repository `JarradPrice / poct-device-alexa-apps`. The `Actions` tab is selected, showing a list of workflow runs. The left sidebar lists several workflows, including `ASK CLI Deployment - alexa-s...`, `Alexa Skill Beta - create, start, ...`, `Alexa Skill Submit - test, depl...`, `Build and deploy dotnet core ...`, and `Push Google Apps Script cod...`. The main area displays `34 workflow runs` for the `Alexa Skill Beta - create, start, update, ...` workflow. The runs are listed in a table with columns for `Event`, `Status`, `Branch`, and `Actor`. The first two runs are successful (green checkmarks) and the third is failed (red X). Each run entry includes the workflow name, a description, the time it was run, and a duration.

Event	Status	Branch	Actor
Alexa Skill Beta - create, start, update, and invite emails #3: Manually run by JarradPrice	Success	22 hours ago	53s
Alexa Skill Beta - create, start, update, and invite emails #2: Manually run by JarradPrice	Success	22 hours ago	1m 2s
Alexa Skill Beta - create, start, update, and invite emails #1: Manually run by JarradPrice	Failure	22 hours ago	14s

Figure 29. Screenshot of the workflow runs of the repository.

Clicking on a run will show each “Job” for that workflow. Where you can then go into the logs for that job.

Summary

Jobs

Checkout

Deploy

Check Utterance Conflicts

Validation test

Submit

Store Artifacts

Manually triggered 20 days ago

Status

Total duration

Artifacts

JarradPrice 5abc663

Failure

3m 15s

—

alex-submit.yml

on: workflow\_dispatch

Checkout

Deploy

1m 15s

Check Utterance Conflicts

Validation test

Submit

Store Artifacts

—

+

Annotations

1 error

Validation test

Process completed with exit code 1.

Figure 30. Screenshot of the summary view for a workflow run.

On workflows that contain tests, at the end the code that existed at that point in time is stored as an artifact which can be downloaded later. By using this or the GitHub commit history page a developer can know what tests are being run on what code and the results.

## Skill beta

By using a beta it is possible to test the Alexa skill without having to submit it to Amazon for certification and avoid having it publicly available in the development phase. Testers can be added to the beta by adding their emails to the beta testers list. An email is then sent with an invitation link to join the beta and have the in-development skill available to them. You can have a maximum of 500 testers.

A beta can be started from the Alexa developer console. This was how original testing was achieved for this project.

The screenshot shows the 'Beta Test' section of the Alexa developer console. At the top, there's a green banner stating 'Your Beta Test is Active' for 'Health App Prototype'. Below this, there's a form to add 'Beta Test Administrator Email Address' with an 'Add' button. A table titled 'Manage Access to your Skill Beta Test (0 total testers, 0 active testers)' has columns for 'EMAIL ADDRESS', 'STATUS', and 'ACTIONS'. A 'Add beta testers' link is visible. At the bottom right, a blue button indicates 'Beta testing is enabled'.

**Beta Test**

Give access to skill beta testers by adding their email addresses. Once Beta Test is enabled, you can continue to edit your skill until you are ready to submit your skill for Amazon review.

**Your Beta Test is Active**  
Health App Prototype Beta Test is available to 0 testers. They will have access to this skill through Saturday, January 08, 2022.

[Copy invite URL](#) [End Test](#)

Beta Test Administrator Email Address

[Add](#)

Manage Access to your Skill Beta Test (0 total testers, 0 active testers)

<input type="checkbox"/>	EMAIL ADDRESS ▲	STATUS	ACTIONS
<input checked="" type="checkbox"/>	Add beta testers		

< 0 – 0 of 0 Testers > [View all](#)

[Beta testing is enabled](#)

Figure 31. Alexa developer console beta form.

Through these beta tests, it was possible to easily add the project supervisors to give feedback. From this an error was discovered where Alexa would say that there were “problems connecting to the network” (Figure 32).



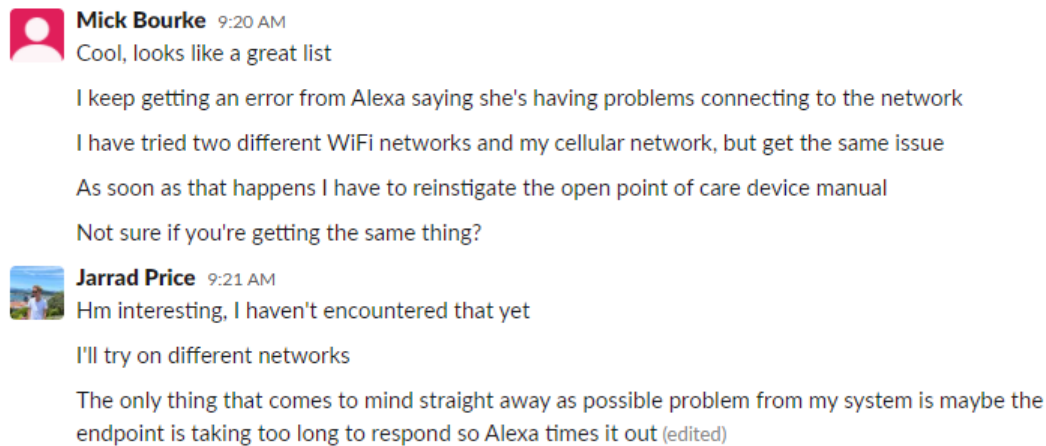


Figure 32. Screenshot of conversation with supervisor on skill beta.

The initial thought process was that the Azure Functions endpoint was taking too long to retrieve and send back the response, which would result in Alexa giving up. After research it was found that this was not the issue as “Alexa waits for your response for 8 seconds before timing out” [21] and the average latency times of the endpoint were around 6 seconds.

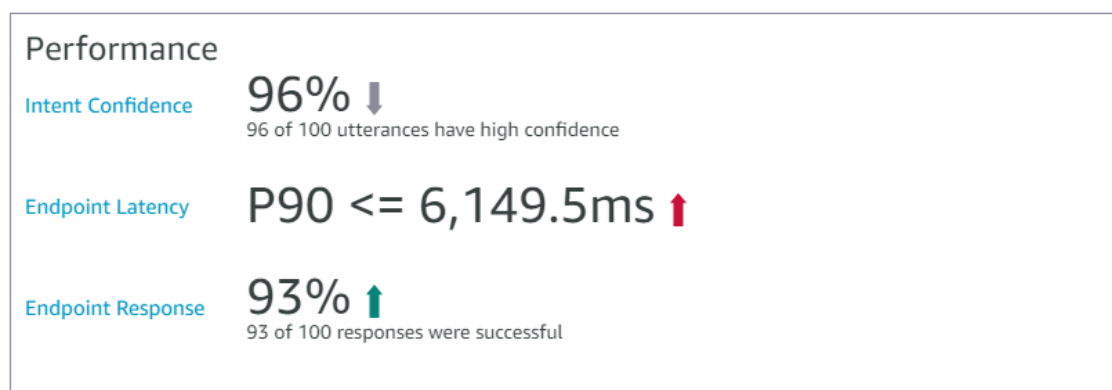


Figure 33. Endpoint performance data available on the analytics page in the skills developer console.

And this time was further reduced with the restructuring of the Alexa endpoint to use its own database service to retrieve database items as opposed to using another HTTP trigger.

## Device testing

During beta testing the skill was tested, in the developer console, with the iOS Amazon Alexa app and on an Echo Dot. To test on a physical Amazon smart speaker (Echo Dot) the same account that enabled the beta from the invite link must be used to set up the device. Also important is that the language chosen matches the language that the skills interaction model is, in this project's case English (Australia). If it is set to English (United States) it will not work, this was discovered in testing as the device was originally configured with that language and attempting to activate the skill would simply result in a sound with the skill not being launched.

A video of the project skill on the Echo Dot after successful configuration can be viewed at:

- <https://youtu.be/Uku08ChOeWI>

## Alexa API Tests

As part of the automatic deployment the Alexa Skills Kit Command Line Interface (ASK CLI) is used to access the Skill Management API (SMAPI) which is used to test the Alexa skill.

### Utterance Conflicts

The first test is to detect utterance conflicts. An utterance conflict is where multiple “utterances”, also known as phrasings of questions, map to more than one intent. These can be viewed in the developer console, but for this project are also checked in the command line during automatic deployment.

It is set up in such a manner that if any conflicts are detected the workflow run will fail and not continue to submit the skill.

```
##### Checking Conflicts #####  
Checking conflicts for locale: en-AU  
#####  
No Conflicts detected
```

Figure 34. Workflow log when there are no utterance conflicts are detected.

If conflicts are detected the workflow logs will display the number of conflicts.

### Validation Pass

The next test performed is a validation test. This is for automatically detecting if the skill is ready to submit to Alexa for certification.

```
##### Checking validations #####  
Checking validations for locale: en-AU  
#####  
Id of validation: a2e59219-0f7b-4820-96c0-126d2ab29e57  
Current status: "SUCCESSFUL"  
Validation pass
```

Figure 35. Workflow log when there is a successful validation.

A failed validation is shown in Figure 36. This error was due to an incorrect file path being used to target the location of the interaction model.

```
"response": {  
  "message": "Unsupported locale. The accepted locales are: de-DE, en-AU, en-CA,  
en-GB, en-IN, en-US, es-ES, es-MX, fr-CA, fr-FR, it-IT, or ja-JP"  
}  
}  
Id of validation:  
error: required option '-s,--skill-id <skill-id>' not specified  
Current status:  
Validation errors:  
Error: Process completed with exit code 1.
```

Figure 36. Workflow log when there is an unsuccessful validation.

## **Amazon Certification**

This is the final test of the skill. This test is not undertaken by the command line but rather the skill is sent to Amazon for official certification. The previous validation test does not perform the full set of testing that the certification process does, ultimately certification is up to Amazon. Amazon provides a checklist to ensure the best chance of certification [22]:

1. It must meet Alexa policy guidelines
2. It must meet the security requirements for the method of hosting the skill service (for this project Azure Functions)
3. Requirements for skills that allow purchases - not relevant for this project
4. Requirements for Protected Health Information skills - not relevant for this project, see Healthcare adoption & legal requirements section
5. The information presented in the skill must accurately reflect the core functionality in the skill
6. It must pass voice interface and user experience tests, verifying the quality of the skill
7. Requirements for skills that use reminders - not relevant for this project

After the certification review an email will be sent to the account that the skill was created on.

## Static Quality Analysis

The aim of quality tests is to ensure that the code that exists is, understandable, maintainable, and clear. Clean code is desirable. By integrating objective tools to judge code quality throughout development it is easy to detect and prevent problems. Problems can be things such as:

- Duplicate functions
- Complex methods
- Low code quality
- Non-standard coding style

The following checks can be run at any time in the code development environment terminal. The tool used for judging this quality is ESLint:

- <https://eslint.org/>

The goal of this tool is for avoiding bugs and making more consistent code. This tool can even be used to automatically fix most errors.

To display and check the code quality the “eslint-detailed-reporter” npm package is installed which generates easy to use HTML reports. The next configuration steps was adding new scripts to the package.json, these being:

- "lint": "eslint --format junit --output-file reports/eslint/eslint.xml --ext .js ../../spreadsheet-gas/"
- "lint-html": "eslint -f node\_modules/eslint-detailed-reporter/lib/detailed.js -o reports/eslint/report.html --ext .js ../../spreadsheet-gas/"

Now by running “npm run lint” then “npm run lint-html” a report.html file is generated in a reports/eslint/ folder. After running this on the latest project build, the output given is displayed in Figure 37.

# ESLint Report - Error

## Summary

6 problems (6 errors, 0 warnings)

### Top errors

Rule	Count
null	6

### Most Problems

File Path	Errors	Warnings
C:\Users\jarraduni\Documents\GitHub\poc-device-alexa-apps\spreadsheet-gas\GitHubAPIRequest.js	1	0
C:\Users\jarraduni\Documents\GitHub\poc-device-alexa-apps\spreadsheet-gas\global.js	1	0
C:\Users\jarraduni\Documents\GitHub\poc-device-alexa-apps\spreadsheet-gas\Main.js	1	0
C:\Users\jarraduni\Documents\GitHub\poc-device-alexa-apps\spreadsheet-gas\Services\ApiDriver.js	1	0
C:\Users\jarraduni\Documents\GitHub\poc-device-alexa-apps\spreadsheet-gas\Services\GitHubClient.js	1	0

Figure 37. ESLint output.

The top area of the report provides a summary. It will show the most common errors and warnings if there are any to display. Also available is seeing the files with the most problems, clicking on one takes you to the position of that file in the details list. The details list is what occupies the rest of the page.

ESLint is a JavaScript quality analyser so it tests the Google Apps Script code for the Sheet. Since GAS has its own custom global variables, also installed is a plugin to avoid misidentified errors in code. This was done with the “eslint-plugin-googleappsscript” plugin:

- <https://github.com/selectnull/eslint-plugin-googleappsscript>

## Details

Filters:

All

Warnings

Errors

▶ C:\Users\jarraduni\Documents\GitHub\poc-device-alexa-apps\spreadsheet-gas\GitHubAPIRequest.js	1 problem (1 error, 0 warnings)
▶ C:\Users\jarraduni\Documents\GitHub\poc-device-alexa-apps\spreadsheet-gas\global.js	1 problem (1 error, 0 warnings)
▶ C:\Users\jarraduni\Documents\GitHub\poc-device-alexa-apps\spreadsheet-gas\Main.js	1 problem (1 error, 0 warnings)
▶ C:\Users\jarraduni\Documents\GitHub\poc-device-alexa-apps\spreadsheet-gas\Services\ApiDriver.js	1 problem (1 error, 0 warnings)
▶ C:\Users\jarraduni\Documents\GitHub\poc-device-alexa-apps\spreadsheet-gas\Services\GitHubClient.js	1 problem (1 error, 0 warnings)
▶ C:\Users\jarraduni\Documents\GitHub\poc-device-alexa-apps\spreadsheet-gas\Services\SheetsDriver.js	1 problem (1 error, 0 warnings)

Figure 38. Details section of ESLint report.

Clicking on any detail in the list opens the summary for that file. Also available is the option to click the “SourceCode” header tab and it will show the exact location in the file of the found error, displaying the code. This provides fast analysis and identification.

# Reflection

This section will encompass project reflection. For reflection the cut content and unimplemented/intended features will be detailed as well as areas of possible future system expansion if the project was to be worked on further. There will also be notes on what would have been done differently and key points for developers who wish to undertake the same project.

## Cut and unimplemented content

The main content that was not completed is increasing the robustness or “cleverness” of conversations with Alexa. This would involve things like multiphase conversations and fallback intents that would guide the user to the right response.

Early planning theorised two types of intentions from a user:

1. They know exactly what information they want and as such will ask directly - "what happens if x"
2. They do not know the exact wording of how to get the information from Alexa or there is multiple similar information for different machines from wording - "tell me about x", "how to do x", "what is step x"

For the first type there are the hardcoded intents built in Alexa that can quickly be pulled from the knowledge base once recognised, an intent name with its question set exists in the skill and maps directly to a response with the same intent name in the database.

For the second type the skill would need a multiphase conversation where Alexa has follow-up questions e.g., "what is the machine for that one?", "what are you trying to do?". So if an intent is not recognised Alexa asks one of the following:

- What machine the question is for, with Alexa listing available devices
- What area it is, a procedure, general information, etc

This ability would provide a better user experience by being able to successfully complete a user's request even if they do not know exactly how to get what they want. The main reason that can be attributed to why this was not available in the final version of the system is due to the introduction of the management hub Google Sheet into the scope. Once this Sheet became a cornerstone of the project this feature was sidelined as the difficulty of supporting these multiphase conversations increases greatly when also having the requirements of providing an easy-to-use interface where the Alexa skills intents can be updated. To add smarter fallback intents, or intents that trigger a wait for inputs from the user, the complexity of the Sheet would increase by a large degree.

Finally, as described in the Amazon Alexa - System design section since the engineering challenge of combining three different platforms into creating a cohesive experience with CI/CD features and testing, there was no time left over for testing and verifying the usability of the Alexa skill for end-users (NSW Health staff) to an acceptable degree. This was a majority restriction of the time frame and would easily be fixed with future development of the project. This is a detriment to the project but does not pull down the quality greatly as due to the nature of the skill being a simple question-answer response VA, it can be taken as acceptable due to the small amount of beta and device testing that was achieved.

## **Future system expansion**

The following features have been identified as possible routes for adding to the system:

- Better logging
- Metrics email notifications
- GAS code refactoring
- Skill image responses

### **Better logging**

The current system does have logging in place but is disjoint and could be expanded further. Logs are available at these locations:

- GitHub Actions page
- Azure console
- Alexa console

To track the logs of the complete journey of information in the system it requires visiting each of these locations. It would be preferable to consolidate all these logs in one location, these could then all be available by request from the Sheet management hub, emailed, or stored in the database. These are possible solutions for having a one access location.

Logging could also be extended to give more information. The logging of pipeline events, Azure events, Alexa events and Google Sheet events could all be more robust and notify users of errors with full logs if any occur.

By combining all log sources and providing increased log information and then sending to one central location it would provide a system that is easier to be tracked and debugged.

## Metrics email notifications

An important dataset to use for judging the success of the skill is the skill metrics, available in the Alexa developer console.

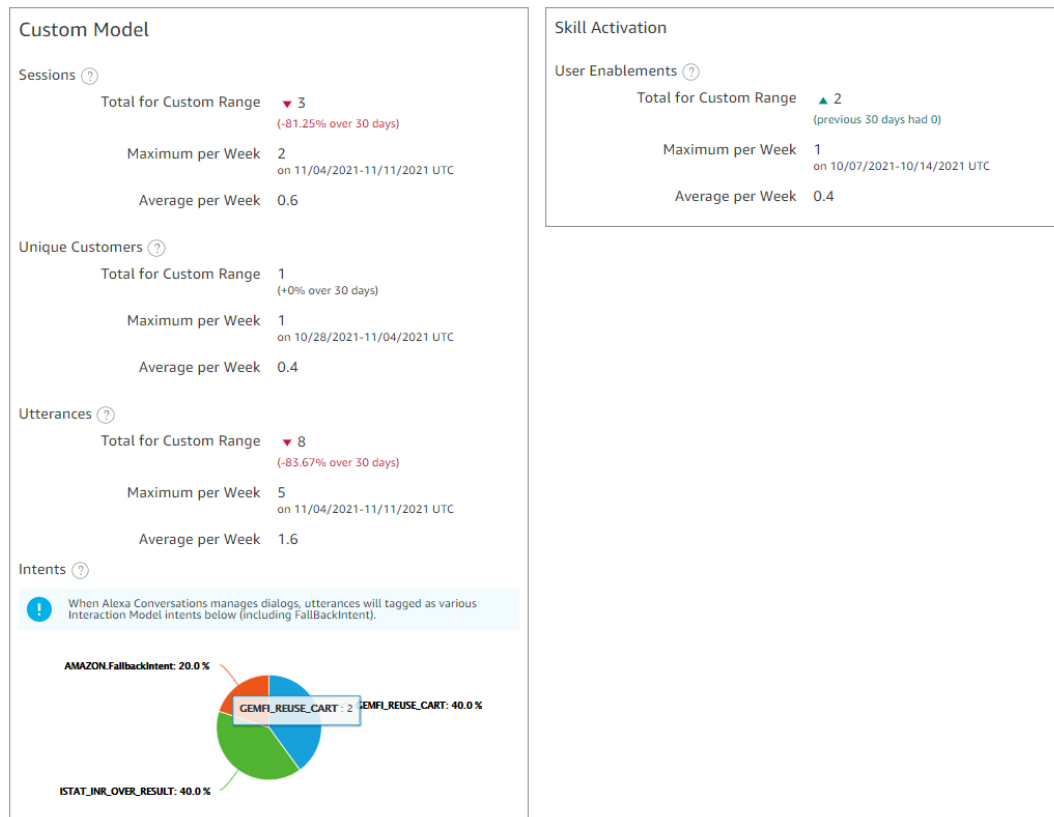


Figure 39. Part of the analytics page in the Alexa developer console.

With these you can ascertain the most popular intents, questions that could not be mapped to an intent, endpoint latency, and more. These allow the user to ensure system quality and reveal areas that could be improved.

The automatic deployment nature of this project results in not needing to visit the skill developer console as much, but it is still required to view these metrics. Ideally, users would not have to access this console after the first setup of the skill. This would mean providing other methods to get metrics data.

This is possible using the Alexa Skill Management API (SMAPI). This API is already used in the GitHub deployment workflows so has already been introduced into the system. The spot that makes most sense to retrieve metrics would be the Google Sheet management hub. This would build more into the concept of the Sheet being a “one stop shop” for all the needs of the system.

To build this GAS code would call the SMAPI RESTful HTTP interfaces to retrieve metrics data on request of the user. Another option would be to set up an Azure Function to automatically email metrics to specific addresses at a set schedule for streamlined user updating, such an implementation can be seen with this tutorial:



- <https://levelup.gitconnected.com/email-yourself-daily-alexa-skill-metrics-updates-using-lambda-smapi-and-ses-9c16ac97c1f8>

### **GAS code refactoring**

As the requirements and features of the Google Sheet evolved as the project progressed there was no original structure planning or documentation. This resulted in functions and file structure that is not optimal. A future endeavour could be refactoring the GAS codebase to have clear separation of concerns and project folder structure.

This would be important moving forward in the project if wishing to expand the Sheets functionality, but this is something that may not be applicable if moving away from the Google Sheet as a management hub, as detailed in the Retrospect section.

## Skill image responses

Skills can send cards back along with the response. These cards contain text and can also contain images. These cards can be seen on the Amazon Alexa app, desktop web browsers and smart speakers that include screens.

The benefits of cards are that they can provide additional information that does not have to be as concise as the voice response. For this specific project use case, images would be helpful to show what aspect of the health device the response is referring to.

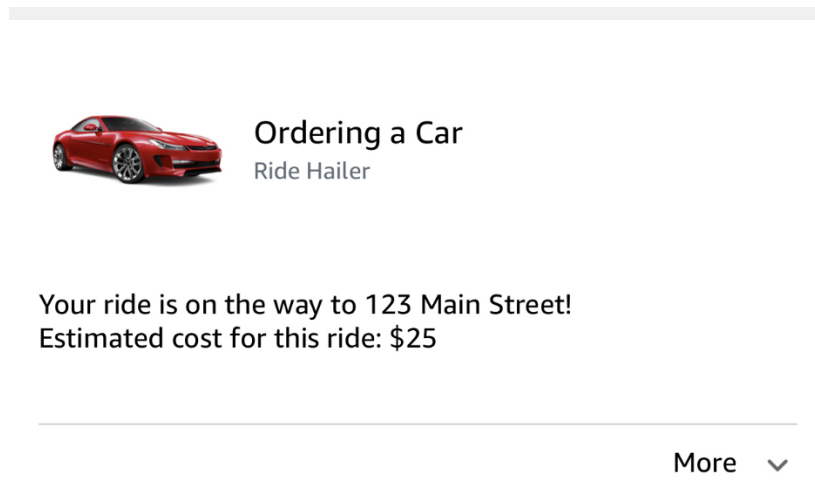


Figure 40. Example of a card with an image, from <https://developer.amazon.com/en-US/docs/alexa/custom-skills/include-a-card-in-your-skills-response.html>.

The returned response card from the endpoint must contain URLs for a large and small version of the image for compatibility across devices with different screens. This requires storing two sizes of the image in your chosen hosting service.

This was not implemented in the project but was considered for implementation at one point as a proof of concept, ultimately this did not happen as the physical testing device did not have a screen and the procurement and storing process for the images would require dedicated research. The used storage solution Cosmos DB also does not have anything inherently built in to manage image attachments, another service might be considered to accomplish the task. It would also require increased development into the Sheet hub to be able to include specifying/uploading images for responses if continuing to utilise that aspect of the project.

## Retrospect

When reflecting on the project there are a few aspects that I would have done differently. One is that although simple and easy to understand for users, using the Google Sheet as a management hub does have its limitations. Limitations include:

- The scripting language that must be used to add functionality is restricted to Google Apps Script which is cloud-based JavaScript. This comes with hindrances of external calls being subject to Google's restrictions and not being able to install node libraries or the like. Being restricted to Google cloud solution for scripting reduces the design space.
- Having to structure sheets correctly for correct functioning. For the GAS to successfully get data from sheets they must be formatted correctly. This means things such as the "CONFIG" sheet having correct values in the right cells and the "Questions" sheet having devices separated by a blank row. Breaking the formatting will either result in errors that are presented to the user or in the worst case the errors will result in bugs that propagate through the system if pushed down the pipeline.
- If there becomes many devices/intents in the Sheet it will be harder and more unwieldy to manage all of the Sheet's data. This is not of a large concern as the simple layout reduces this impact, but other more custom designs would be better.
- If wanting to implement more advanced conversation Alexa features like multiphase conversations, waiting for user input, including images, etc, the Sheet would have to become more complex and thus lose its main appeal. Implementing such features would require careful thought and a diminishing return in the amount of effort required to implement.

To not have these limitations, if done again instead of using Google Sheets as the management hub I would instead build a static web app, most likely using Azure's Static Web Apps solution to integrate with the other used Azure services, Functions and Cosmos. Using this platform would have a longer setup time to build the hub but would have none of the limitations that the Google Sheet has. Other advantages of using an Azure Static Web App are:

- Integrates easily into CI/CD workflows
- Under one subscription with Functions and Cosmos
- Layout can be built to be fully custom to best support functionality in a skill management hub
- Easily use node packages to enable quick access and reduce development time for utilising third-party endpoints and APIs
- Easier distribution and authentication of users. Can simply share a URL and have a sign-in screen to ensure only the allowed users can access the hub

If doing this project again I would go this route as the limitations of Google Apps Script was hindering when building functionality into the Google Sheet.

Other changes that would be implemented in retrospect would be:

- Submitting skill to certification earlier. To have your Alexa skill in the Alexa skill store it must be sent to certification to Amazon. This process can take two weeks to complete and if your skill does not meet the requirements you will have to fix it and resubmit. The skill in this project did not reach a certification ready state in time to be included in the Alexa skill store before the writing of this document.
- A great way for assessing the system is through usability tests. This would have been done early to assess parts of the system such as the Google Sheet and interactions with Alexa. This would have been achieved by sharing the skill beta with more people and doing surveys which would contain walkthroughs of using the management hub with a questionnaire afterwards.
- Another beneficial process to begin early would be deploying test runs in the actual intended environment of NSW Health. This would have guided design and development decisions more accurately to match the intended use cases and user interactions.

## Developer notes

This is for developers to note if attempting the same sort of project. If wishing to follow suit and attempt this project or one of a similar nature I have the following notes for prospective developers:

- Be aware that many of the platforms and libraries are not fully matured. The use of Alexa development and integration with CI/CD pipelines are relatively recent in software stacks and as such are prone to having bugs. It is best to keep up to date with versions and check forums such as GitHub's issue tracking threads for the latest developments.
- Fully map out use cases before undertaking development. If I had understood my users better and how I wanted updating of the skill to be done externally at the beginning, I would have not gone through with the Google Sheets path. Understand your users and how they will use the system as much as possible.
- If using Alexa only go down the route of using Azure as your cloud service for the endpoint and storage if you have good reasons to do so, otherwise use Amazon's services as it will be easier to integrate with Alexa and more documentation is available.

These should be used as well as the previous sections to be best equipped when entering this project's domain.

# Results & Conclusion

## Evaluation

First to assess the success of the final solution the original project outcomes, aims and deliverables will be cross-checked.

These have been taken from the interim report document for the project which can be accessed in part in the appendix - Interim Report.

Project outcome/aim	Completed?
The VA is to be voice activated and interfaced with through smart speakers.	Yes. Custom skill is activated with a launch phrase on Alexa enabled devices.
Provide quick access to information with results returned without needing physical interaction.	Yes. Information is returned after a voice query in under 8 seconds (hard requirement by Alexa).
Responses are to be pulled from a database that is developed from information currently contained in device manual documents.	Yes. The final project has two devices mapped. The two device documents used are iSTAT [23] and ABL90 [24].
The project will result in an Alexa skills app with accompanying Azure Function endpoint that facilitates communication between Alexa and a Cosmos DB knowledge base.	Yes. These platforms have been set up and connected.
Building the Alexa skills application will result in a skill with a custom model, intents, and sample utterances.	Yes.
Development of an Azure Function application requires creating Function apps and programming their .NET classes. The outcome of this will be the hosted Functions on Azure.	Yes.
The knowledge base system outcome will be a document database containing information on the health devices.	Yes.

All outcomes have been completed. This is a success.

<b>Project deliverable</b>	<b>Completed?</b>
Chosen platform.	Yes. Alexa.
Initial technology documentation/learning documents.	Yes. Interim report.
Environments set up and platforms connected. <ul style="list-style-type: none"> <li>Alexa talking to Azure Functions and Azure Functions talking to Alexa and Azure Cosmos DB.</li> </ul>	Yes. All systems are connected and transfer information between.
Alexa responding to sample queries.	Yes. Alexa can respond to any queries added into the system.
Development and integration of health device information into Alexa app. <ul style="list-style-type: none"> <li>Health device manual converted to database.</li> <li>Alexa updated</li> <li>Endpoint updated</li> <li>Use case tests</li> </ul>	Majority yes. Use case tests were not performed in the final project, other testing was performed instead.
Alexa running the developed health app on a device.	Yes. The Alexa skill was run on a mobile device and Echo Dot.
System deployed within NSW Health workspace.	No. Deployment documentation has been completed but the system has not been deployed in the NSW Health workspace.
NSW Health tests of application.	No. Small tests have been done by NSW Health supervisor but not to any reasonable degree.

That is a completion ratio of 6.5 deliverables completed out of 8 intended. This is regarded as a success.

By this analysis the project overall can be regarded as a success.

## Contribution to the field

This project was undertaken due to desire by NSW Health. They believed it to be something that would benefit their departments, this affirms the growing awareness of VAs. By simply being requested to undertake this project we can acknowledge how VA is expanding into healthcare more.

This project also highlights that a VA system can be built for healthcare without being subject to the legal requirements that software falls under when being used in certain circumstances in the healthcare space. The VA in this project and its nature of simply helping with “FAQ like” use cases does not interact with sensitive information. It reveals the many possibilities that a VA can be catered for.

This project follows a very similar implementation for the knowledge base of the VA as the Study - Implementation of virtual assistants for education use case. That study, like this one, used Google Sheets to modify the VA. But unlike that project this one does not use Dialogflow as the VA platform and as such the connection had to be designed and made from scratch. This project proves that using a Google Sheet to manage a VA does not have to be tied to Google’s services and can be more platform agnostic.

Another contribution that this project provides is documentation on integrating multiple different platforms into one cohesive VA system - Amazon, Microsoft, and Google. On research during development of this project, there was small documentation found on using Azure with Alexa and none found for connecting Sheets with Alexa. This project by way of its publicly hosted repository codebase and being under a MIT license [25] allows any developer to download the code and use it freely.

Lastly, the use of a custom external management hub (Google Sheets with Google Apps Script) for the VA in this system is also an aspect with sparse documentation found online. This project provides a guide on building your own management hub as opposed to using Amazon’s developer console combined with Azure’s portal or using a third-party solution.

## Final

This project has met the original aims and requirements as well as NSW Health supervisors having also expressed their approval of the final product. The final system has been tested and confirmed to be functioning correctly. The only steps that did not occur was deployment and testing in the NSW Health environment, but this has been deemed acceptable as appropriate documentation for deployment has been composed including the repository wiki and this report. Using these facilitates NSW Health deploying the solution and hosting it on their own systems, where they can then extend the project according to their wishes.

# References

- [1] S. S. M. O. S. E.-S. Glennn Moy, “Recent Advances in Artificial Intelligence and their Impact on Defence,” DST Headquarters, Canberra, 2020.
- [2] A. M. Bret Kinsella, “Voice Assistant Consumer Adoption in Healthcare,” Orbita, 2019.
- [3] D. K. S. L. Luke Dale, “INDUSTRY FOCUS: LEGAL CONSIDERATIONS FOR MEDICAL TECHNOLOGY BUSINESSES,” 17 January 2020. [Online]. Available: <https://hwlebsworth.com.au/industry-focus-legal-considerations-for-medical-technology-businesses/>.
- [4] Department of Health Therapeutic Goods Administration, “Regulation of software based medical devices,” 24 May 2021. [Online]. Available: <https://www.tga.gov.au/regulation-software-based-medical-devices>.
- [5] Department of Health Therapeutic Goods Administration, “Examples of regulated and unregulated software (excluded) software based medical devices,” 2021.
- [6] L. C. & G. H. Page, “How an Artificially Intelligent Virtual Assistant Helps Students Navigate the Road to College,” 2017. [Online]. Available: <https://doi.org/10.1177/2332858417749220>.
- [7] D. G. L. G. V. D. I. C. J. R. Roberto Reyes, “Methodology for the Implementation of Virtual Assistants for Education Using Google Dialogflow,” Tecnologico de Monterrey, Monterrey, 2019.
- [8] A. Sargeant, Interviewee, *UoN Introduction / Set-up virtual assistants to help with supporting hospital in the home*. [Interview]. 10 March 2021.
- [9] Visual Paradigm, “What is a Software Process Model?,” [Online]. Available: <https://www.visual-paradigm.com/guide/software-development-process/what-is-a-software-process-model/>.
- [10] T. Warren, “Microsoft no longer sees Cortana as an Alexa or Google Assistant competitor,” 18 January 2019. [Online]. Available: <https://www.theverge.com/2019/1/18/18187992/microsoft-cortana-satya-nadella-alexa-google-assistant-competitor>.
- [11] Internet of Business (IoB), “Microsoft, Amazon roll out Alexa-Cortana unification programme,” [Online]. Available: <https://internetofbusiness.com/microsoft-amazon-roll-out-alexa-cortana-unification-strategy/>.
- [12] docs.microsoft.com, “Access control in the Azure Cosmos DB SQL API,” 24 April 2021. [Online]. Available: <https://docs.microsoft.com/en-us/rest/api/cosmos-db/access-control-on-cosmosdb-resources>.
- [13] A. Mersch, “What is Layered Security & How Does it Defend Your Network?,” 4 October 2021. [Online]. Available: <https://blog.totalprosource.com/what-is-layered-security-how-does-it-defend-your-network>.



- [14] Research and Markets, “Global Smart Speakers Markets Report 2021,” 24 May 2021. [Online]. Available: <https://www.globenewswire.com/en/news-release/2021/05/24/2234471/28124/en/Global-Smart-Speakers-Markets-Report-2021-Alexa-Google-Assistant-Siri-Cortana-Others-Market-is-Expected-to-Reach-17-85-Billion-in-2025-at-a-CAGR-of-26.html>.
- [15] Amazon, “Module 1: Why Build Alexa Skills,” [Online]. Available: <https://developer.amazon.com/en-US/alexa/alexa-skills-kit/get-deeper/tutorials-code-samples/build-an-engaging-alexa-skill/module-1>.
- [16] D. Bohn, “AMAZON SAYS 100 MILLION ALEXA DEVICES HAVE BEEN SOLD — WHAT’S NEXT?,” 4 January 2019. [Online]. Available: <https://www.theverge.com/2019/1/4/18168565/amazon-alexa-devices-how-many-sold-number-100-million-dave-limp>.
- [17] Amazon, “Alexa Design Guide,” [Online]. Available: <https://developer.amazon.com/en-US/alexa/alexa-haus>.
- [18] Amazon, “Design Principles,” [Online]. Available: <https://developer.amazon.com/en-US/alexa/alexa-haus/design-principles>.
- [19] Microsoft, “Continuous deployment for Azure Functions,” [Online]. Available: <https://docs.microsoft.com/en-us/>.
- [20] Microsoft, “Azure Cosmos DB pricing,” [Online]. Available: <https://azure.microsoft.com/en-au/pricing/details/cosmos-db/>.
- [21] Amazon, “Response Events,” [Online]. Available: <https://developer.amazon.com/en-US/docs/alexa/device-apis/alexa-response.html#deferred>.
- [22] Amazon, “Certification Requirements,” [Online]. Available: <https://developer.amazon.com/en-US/docs/alexa/custom-skills/certification-requirements-for-custom-skills.html#submission-checklist>.
- [23] NSW Health Pathology, “Abbott iSTAT,” [Online]. Available: <https://www.pathology.health.nsw.gov.au/clinical-services/point-of-care-testing/resources/abbott-istat>.
- [24] NSW Health Pathology, “Radiometer ABL90 Flex Series,” [Online]. Available: <https://www.pathology.health.nsw.gov.au/clinical-services/point-of-care-testing/resources/radiometer-abl90-flex-series>.
- [25] Open Source Initiative, “The MIT License,” [Online]. Available: <https://opensource.org/licenses/MIT>.
- [26] NSW Government Health Pathology, “NSWHP-POCT-INFO-0107-3,” 2018.

# Appendices

## GAS Cosmos Authorisation Token Code

```
var text = (verb || "").toLowerCase() + "\n" +
            (resourceType || "").toLowerCase() + "\n" +
            (resourceId || "") + "\n" +
            date.toLowerCase() + "\n" +
            "" + "\n";

var body = Utilities.base64Decode(Utilities.base64Encode(text));
var key = Utilities.base64Decode(masterkey);
var out = Utilities.computeHmacSignature(Utilities.MacAlgorithm.HMAC_SHA_256,
body, key);
var signature = Utilities.base64Encode(out)

var MasterToken = "master";
var TokenVersion = "1.0";

res = encodeURIComponent("type=" + MasterToken + "&ver=" + TokenVersion +
"&sig=" + signature);
```

# Interim Report

This appendix section contains part of the interim report that was completed at the end of semester 1 for this project.

## Project background & aims

The purpose of this project is to develop additional support for departments in hospitals by use of a virtual assistant (hereafter referred to as a VA). The domain space of this project is NSW Health staff who wish to have quick access to procedure information on health device operation and queries. Such a device query could be:

- “What do I do if I get an INR result greater than 4?”

This information is drawn from pre-existing device manuals. More examples of possible questions and their corresponding responses is provided in Figure 41, by way of a FAQ page straight from a health devices manual.

### Fact Sheet i-STAT Cartridges FAQs

---

#### What is the analysis time for the different cartridges?

The analysis time is:

- cTnI, BNP, beta-hCG cartridges - 10 mins
- CG4, CG8+,CHEM8 cartridges – 2-3 mins
- PT-INR cartridges – dependent on clotting time, typically 2-4 mins

#### How should the i-STAT cartridges be stored?

Cartridges should be stored at temperatures between 2 and 8 C. Do not expose to temperatures above 30°C. Do not use after the expiration date on the cartridge pouch and box. Cartridges have a room temperature storage expiry. (see below)

Figure 41. Excerpt from NSW Health Pathology PoCT Service Device [26].

This project aims to help users access such information more quickly and efficiently, enabling a better workflow. To achieve this type of access a virtual VA is to be developed.

The VA is to be voice activated and interfaced with through smart speakers that are to be located within the same space as the health devices. It will provide quick access to information with results returned without needing physical interaction. As mentioned, responses are to be pulled from a database that is developed from information currently contained in device manual documents.

This use of VAs has not been attempted by NSW Health before, it is a new development space. To begin considering such a solution, background research is to be undertaken on global experience on the use of VAs in healthcare, as to learn and build on established work. Prior to development research and training must be done on the chosen platforms assistant development suite along with all components of a finalized system, drawing guidance from studied existing solutions.

While reviewing literature and developing this solution this project will aim to establish if modern virtual assistant development platforms provide enough accessibility and extensibility to develop a robust voice assistant without being a subject expert. *What is the development flow for a modern virtual assistant platform?*

## Project deliverables

### Outcomes

To achieve the outcome of having a VA with customized responses, the project will result in an Alexa Skills app with accompanying Azure Function endpoint that facilitates communication between Alexa and a Cosmos DB knowledge base. An overview of the interacting systems can be seen in Figure 42.



Figure 42. Interacting systems for design.

These systems require different development methodologies. Building the Alexa Skills application makes use of the Alexa Developer Console, this provides an online GUI which streamlines the creation process. The outcome from this is a Skill with a custom model, intents, and sample utterances. Development of an Azure Function application requires creating Function apps and programming their .NET classes on the Azure Platform. The outcome of this will be the hosted Functions on Azure with their code base. The knowledge base system outcome will be a document database containing information on the health devices.

### Key milestones & deliverables

- Chosen platform.
- Initial technology documentation/learning documents.
- Environments set up and platforms connected.
  - Alexa talking to Azure Functions and Azure Functions talking to Alexa and Azure Cosmos DB.
- Alexa responding to sample queries.
- Development and integration of health device information into Alexa app.
  - Health device manual converted to database.
  - Alexa updated.
  - Endpoint updated.
  - Use case tests.
- Alexa running the developed health app on a device.
- System deployed within NSW Health workspace.
- NSW Health tests of application.

# Deployment document

This has been pulled from the wiki for this projects GitHub page:

- <https://github.com/JarradPrice/poct-device-alexa-apps/wiki/Deployment>

To successfully deploy this project and enable its DevOps CI/CD pipeline each platform has to be set up and the correct repository secrets added. This pipeline makes use of **15** secrets, these values are stored as repository secrets as to not leak potentially sensitive information from being a public repo.

We will setup the platforms in this order:

1. GitHub
2. Alexa
3. Azure
4. Google Apps Script/Sheet

## Prerequisite Accounts

These accounts are required.

- GitHub
- Amazon Developer
- Microsoft Azure
- Google

## Setup

### GitHub

The first step for everything is to create your own repository from this repository <https://github.com/JarradPrice/poct-device-alexa-apps/generate>. When creating ensure to tick the 'Include all branches' checkbox.

As we go through each following step there will be secrets to add to your new repository. To add a secret go to the repository page in GitHub, select Settings > Secrets > Add a new secret.

### Alexa

For Alexa we need 7 secrets.

1. First we must add some 'dummy' secrets, add ALEXA\_ACCESS\_KEY\_ID and ALEXA\_SECRET\_ACCESS\_KEY with the values of 'dummy'

These two secrets would normally be retrieved from AWS when hosting using Amazon, but since this is a self-hosted skill we do not have these but we must include something as there is a bug in the current ASK CLI where they must be passed even if not used.

2. Create a new custom skill <https://developer.amazon.com/alexa/console/ask/create-new-skill>
  - Skill name - can be anything
  - Primary locale – English (AU)
  - Choose a model to add to your skill – Custom

- Choose a method to host your skill's backend resources – Provision your own
  - Choose a template to add to your skill – Start from scratch
3. Leave the skill and in your skills list select Copy Skill ID, copy this and it becomes the ALEXA\_SKILL\_ID secret. Save this value temporarily  
The next steps require that node be installed <https://nodejs.org/en/>
  4. Open cmd and run the following command  
npm install -g ask-cli
  5. Next run  
ask configure
  6. In the terminal select [default] under 'Profile'
  7. This will open a new tab in your browser where you are to sign in using the same account that the Alexa skill is hosted with
  8. It will ask if you want to link your AWS account, input 'n'
  9. Now we need to navigate to the cli\_config in your file explorer, it should be something like:  
C:\Users\\.ask\cli\_config
  10. Navigate to that path and open with a text editor, copy the values from these keys into repo secrets:
    - access\_token becomes ALEXA\_ACCESS\_TOKEN
    - refresh\_token becomes ALEXA\_REFRESH\_TOKEN
    - vendor\_id becomes ALEXA\_VENDOR\_ID

## Azure

For Azure we are to setup a Cosmos database and a Function app.

### Cosmos DB

1. Create a new database in the Azure portal <https://portal.azure.com/#create/Microsoft.DocumentDB>
2. Click 'Create' for the 'Core (SQL) - Recommended' API
3. Step through the creation process, these options are up to you
4. Once the deployment is complete go to the new database page, a 'Go to resource' button should appear after deployed
5. Next click the 'Data Explorer' tab
6. Click 'New Container'
7. Add the needed information
  - Set database id as alexa-response-database
  - Set container id as responses
  - set partition key as /id

- Other options are up to you
- 8. Next go to the 'Keys' tab in the sidebar, it will be under the 'Settings' heading
- 9. Copy and keep these values somewhere temporarily
- URI
- PRIMARY KEY

## Function

1. Create a new function app in the Azure portal <https://portal.azure.com/#create/Microsoft.FunctionApp>
- Publish: Code
- Runtime Stack: .NET
- Version: 3.1
- Other options are up to you

Next we need to setup some application settings for the Function app.

1. Once the deployment is complete go to the new Function page, a 'Go to resource' button should appear after deployed
2. Go to the 'Configuration' tab in the sidebar, it will be under the "Settings' heading
3. Add these as new application settings
  - COSMOS\_DATABASE\_ID with value alexa-response-database
  - COSMOS\_CONTAINER\_ID with value responses
  - COSMOS\_URI with value of the copied URI before
  - COSMOS\_PRIMARY\_KEY with value of the copied PRIMARY KEY before
  - ALEXA\_SKILL\_ID with value of the Alexa skill ID obtained in the previous section
4. Click 'Save'

Lastly we need to add the Function title and the publish profile as a GitHub secret.

1. Copy the title of the Function and add as AZURE\_NAME secret
2. Go to the 'Deployment Center' tab in the sidebar, it will be under the 'Deployment' heading
3. Click 'Manage publish profile'
4. Click 'Download publish profile'
5. Open this file in a text editor and copy its contents
6. Add this as a secret to GitHub with the name AZURE\_PUBLISH\_PROFILE

We now need to deploy the Functions code stored in the repository to complete the next steps of setup.

1. Go back to the repository page, select the 'Actions' page

2. Click 'Build and deploy dotnet core app to Azure Function App - azure-endpoint' in the list of workflows. Click 'Run workflow', 'Run workflow'
3. After this workflow has completed navigate back to the Functions page and go to the 'Functions' tab in the sidebar, it will be under the 'Functions' heading
4. Click 'Get Function Url', copy and add this as ALEXA\_ENDPOINT\_URI secret

## Google

The Google technology used for this project is their spreadsheet service, Google Sheets, and their scripting platform, Google Apps Script, for providing the extra functionality in the Sheet.

## Google Sheet

These steps are to get the Google Sheet for easily managing the Alexa Skill.

1. Sign into your Google Account
2. Open [this](#) link which will take you to the already formatted Sheet
3. Click File > Make a copy, saving to your Google Drive

Completing setup of the sheet involves filling the fields in the 'CONFIG' page as shown in the image below.

	A	B	C	D	E	F	G	H	I	J
1										
2	Configuration Sheet							Description		
3	ONLY CHANGE THIS SHEET IF YOU KNOW WHAT YOU ARE DOING									
4										
5										
6										
7	Alexa Variables									
8	INVOCATION_NAME	point of care device manual	Name used to invoke Alexa							
9										
10	Azure Variables									
11	AZURE_HOST		Function app name							
12	API_HOST		API Function name							
13	API_KEY		API Function key							
14										
15	GitHub Variables									
16	Owner		Owner of repo							
17	Repo		Name of repo							
18	Username		Username for commit							
19	Token		Personal access token for commit							
20	Branch	dev	Branch to commit to							
21	Commit Email		Email for commit							
22										

First the Azure variables.

1. AZURE\_HOST is the title of the Azure Function created earlier



2. API\_HOST is DatabaseAccessHttpTrigger
3. API\_KEY can be retrieved by navigating to the 'DatabaseAccessHttpTrigger' page in the Azure portal for the Function, next click 'Function Keys' then copy the default key

Next the GitHub variables.

1. Open [this](#) page to create a new personal access token, only the first scope 'repo' is needed  
**NOTE: You can set the expiration date to whatever you want but note down the date as when the token expires the Google Sheet will stop functioning correctly**
2. Copy the created token and paste it into the "Token" field in the Configuration Sheet
3. Now to fill the other GitHub variables:
  - Owner is the GitHub username of the account hosting the repository
  - Repo is the name of the repository
  - Username is the GitHub username of the account which generated the access token
  - Token was created in step 1
  - Branch is the name of the branch to commit to (by default 'dev')
  - Commit email is the email of the GitHub account which generated the access token

The final step is to enable the Google Apps Script and export answers. This is required as database entries are not included in the repository template.

1. Click EXPORT TOOLS > Export Answers
2. An 'Authorization Required' dialog will appear, click Continue, Google will say that it hasn't verified the app, click Advanced > Go to SheetsManagementServices (unsafe)
3. Google will display the required permissions click 'Allow'  
**NOTE: The code that the Apps Script is running can be verified at any time by going to Extensions > Apps Script in the Sheet**
4. Afterwards again click EXPORT TOOLS > Export Answers and a sidebar will pop up. Use the Answers sheet to add entries to the database

## Google Apps Script
















These steps are not necessary for the functioning of the system but enable automatic deployment workflow for developing the Sheet's behind code scripts. For Google Apps Script we need 6 secrets.

1. Open [this](#) page and enable 'Google Apps Script API'
2. Open cmd and run the following command  
npm install -g @google/clasp
3. Then run  
clasp login
4. This will open a new tab in your browser where you are to sign in using the same account that the Google sheet is hosted with
5. Clasp will ask for Account permissions, click 'Allow'

6. Next after signing in the console should output the location of the clasprc file, it should be something like:  
C:\Users\<username>\.clasprc.json
7. Navigate to that path and open with a text editor, copy the values from these keys into repo secrets:
  - access\_token becomes GAS\_ACCESS\_TOKEN
  - refresh\_token becomes GAS\_REFRESH\_TOKEN
  - id\_token becomes GAS\_ID\_TOKEN
  - clientId becomes GAS\_CLIENT\_ID
  - clientSecret becomes GAS\_CLIENT\_SECRET
8. To get the script ID navigate back to the Google Sheet and click Extensions > Apps Script then in the sidebar click Project Settings > Script ID > Copy, this becomes GAS\_SCRIPT\_ID

### **Final Steps**

At the end your secrets list should look like this:

Repository secrets			
	ALEXA_ACCESS_KEY_ID	Updated 1 hour ago	<button>Update</button> <button>Remove</button>
	ALEXA_ACCESS_TOKEN	Updated 1 hour ago	<button>Update</button> <button>Remove</button>
	ALEXA_ENDPOINT_URI	Updated 1 minute ago	<button>Update</button> <button>Remove</button>
	ALEXA_REFRESH_TOKEN	Updated 1 hour ago	<button>Update</button> <button>Remove</button>
	ALEXA_SECRET_ACCESS_KEY	Updated 1 hour ago	<button>Update</button> <button>Remove</button>
	ALEXA_SKILL_ID	Updated 1 hour ago	<button>Update</button> <button>Remove</button>
	ALEXA_VENDOR_ID	Updated 1 hour ago	<button>Update</button> <button>Remove</button>
	AZURE_NAME	Updated 20 minutes ago	<button>Update</button> <button>Remove</button>
	AZURE_PUBLISH_PROFILE	Updated 24 minutes ago	<button>Update</button> <button>Remove</button>
	GAS_ACCESS_TOKEN	Updated 8 minutes ago	<button>Update</button> <button>Remove</button>
	GAS_CLIENT_ID	Updated 7 minutes ago	<button>Update</button> <button>Remove</button>
	GAS_CLIENT_SECRET	Updated 7 minutes ago	<button>Update</button> <button>Remove</button>
	GAS_ID_TOKEN	Updated 7 minutes ago	<button>Update</button> <button>Remove</button>
	GAS_REFRESH_TOKEN	Updated 8 minutes ago	<button>Update</button> <button>Remove</button>
	GAS_SCRIPT_ID	Updated 4 minutes ago	<button>Update</button> <button>Remove</button>

Finally navigate to the Actions screen for the repository.

1. Click 'Push Google Apps Script code to script ID - spreadsheet-gas' in the list of workflows. Click 'Run workflow', 'Run workflow'
2. Click 'Alexa Skill Beta - create, start, update, and invite emails' in the list of workflows. Click 'Run workflow', enter the email to send link to in 'Tester email' input field then 'Run workflow'

If any of these workflow runs fail, click on it to see if you can discern the reason in the error logs. When in doubt re-insert relevant repository secrets and redo steps for workflow targeted platform.

The next steps to start using the system fully is to use the Google Sheet, read the READ ME and watch the tutorial video.