Creating easy to deploy games/applications with serverless architecture and Alexa

Jaco van Cranenburgh, Jamaal Akhtar
up721426@myport.ac.uk, up774330@myport.ac.uk
School of Computing, University of Portsmouth, Buckingham Building, Lion Terrace, PO1 3HE

1. Abstract

This paper will research 'serverless computing', specifically the advantages and disadvantages of using cloud computing to create easy to deploy applications without the need to set up and manage servers. The paper describes the process of setting up an Alexa based voice application in the form of a game, using Amazon's AWS Lambda service to create a 'serverless' application, and discusses the practical limitations and benefits.

2. Introduction

Serverless computing and voice based home systems are quickly becoming larger parts of our lives every day. Although it's unlikely you would even notice if an application is making use of serverless/cloud computing, applications that make use of it are becoming increasingly more common, and the ease in which serverless computing can be utilised has been improved drastically. Voice based home systems have become much more popular in recent years, with the introduction of Alexa and Google Home. This has opened a new avenue for voice based applications and systems, from smart lightbulbs that operate via the users voice, to video games that utilise control of the game via voice commands, rather than traditional keyboard and mouse or controller. This paper will research into both serverless computing and voice based systems, discuss the benefits and limitations to both, and describe the process of implementing a voice based game using these technologies. The aim is to create a proof of concept game with as minimal setup, deployment and server management as possible, by utilising these technologies to minimise complexity through abstraction.

3. Background Research

Since the release of electronic personal assistants, a new avenue for games development has emerged. Developers are now able to utilize this new technology to create web-based games that are responsive to voice commands.

3.1 Voice Based Games

Voice-based games date back to the N64, released in 1996. An example of one game developed on the Nintendo 64 was 'Hey you, Pikachu!'. It utilized the Voice Recognition Unit, a microphone made exclusively for the Nintendo 64, included when purchasing the game. Using this technology, the game developers were able to implement software that allowed the player to essentially verbally interact with their Pikachu. The device had various reactions depending on the voice command (Bulbapedia, 2017). The VRU was released in 2000 by Nintendo as an extra to the console. It consisted of two parts, one of the parts is the actual unit which would plugs in to the console, and the second part is the microphone that you would plug into the VRU along with a strap to attach it to the controller (Nintendo, Unknown).

Another, more recent example of a voice based game is ‘Starship Commander’. This game not only utilizes voice commands but takes advantage of the new virtual reality technology. In short, ‘Starship Commander is a virtual reality title driven by human speech. The audience is given agency in the middle of a sci-fi story, as part of a military embroiled in a dark intergalactic war. You are in command of a secretive mission, and your decisions have deadly consequences.’ (Human Interact, 2017). All the voice commands and interaction are conducted through the Oculus Rift headset.

Finally, an example that is more relevant to this project. Starlanes is a fairly advanced game that has been developed for Alexa and is purely voice controlled by the player, with responses coming directly from the Alexa unit. Exploring the functionality of the game, it includes a separate web interface which allows the player to login via Amazon. It enables players to chat with each other and check up on the status of the game when the player is not near their Alexa. To sum it up ‘Starlanes is an interactive, voice-based game where the objective is to help the player’s faction dominate the Star Lanes. Using the Alexa Service, you direct the actions of your ‘thoughtship’ by speaking to the ship's computer.’ (Starlanes Wikia, 2017). We will be recreating a similar game to Starlanes in an effort to understand and analyse the web technologies that are used to create voice-active games.

3.2 Amazon Alexa

Amazon developed Alexa, an intelligent personal assistant. Alexa was first used by Amazon’s own product the Amazon Echo and Echo Dot. Alexa works similarly to Apple’s Siri and Google Assistant. Amazon describes Echo as ‘a hands-free speaker you control with your voice. Echo connects to the Alexa Voice Service to play music, answer questions, make calls, send and receive messages, provide information, news, sports scores, weather and more. All you have to do is ask’ (Amazon, 2016). It has been developed to the point where it is no longer just a source of information but is able to be used with other technology for remote control such as light bulbs operated via WiFi in your house. The market for voice-based AI in homes has greatly increased over the past few years and tens of millions of Alexa-powered machines have been sold since their debut in 2014. The U.S market is believed to make up about 70% of those sales (Angers, G. 2017). Amazon has made it easy for developers to create skills with their easy-to-use development tools. It is possible to create a simple skill within an hour without too much coding knowledge, and thus, since its release, more than 15,000 skills have been developed (Angers, G. 2017).

Amazon has described the Alexa Skill Set as ‘The Alexa Skills Kit (ASK) is a collection of self-service APIs, tools, documentation, and code samples that makes it fast and easy for you to add skills to Alexa. ASK enables designers, developers, and brands to build engaging skills and reach customers through tens of millions of Alexa-enabled devices.’ (Amazon, 2017). Amazon have detailed documentation for advanced users and a simple step by step to create a simple skill for beginners. The ASK makes it easier than ever for developers to create applications that utilise voice control, by abstracting the voice technology away from the developer and users. Instead, developers just have to utilise three parts of the ASK, as follows;

1. **Intent Schema**

An Intent Schema is a JSON structured file that declares the set of intents your skill can accept and process (Amazon). Figure 1 shows the syntax used for the JSON intent schema.

Figure 1 - Intent Schema

```
{
 "intents": [
   {
    "intent": "string",
    "slots": [
      {
       "name": "string",
       "type": "string"
      },
      {
       "name": "string",
       "type": "string"
      }
    ]
   }
 ]
}
```

2. **Slots**

In essence, slots are variable types that help Alexa understand information about what the user is saying. Amazon provides some built-in slots like durations, dates, numbers etc but developers can create custom skill slots that are more relevant to the skill they are creating.

3. **Sample Utterances**

Sample Utterances declare which phrases the user can speak to the intents that have been defined. They are simply written as lines in a plain text file and with each utterance there can be multiple slots (Amazon).

This technology enables voice requests from the user to be converted into intents, which are sent to a server. The server then handles these requests, and returns a response which the Alexa skill then uses to create the output. In essence, Alexa handles the input/output speech, but offloads the actual computation that is triggered to a server which is defined in the skill.

There are downsides to using Alexa to build voice applications. Most importantly, it limits the audience for the application to users that own an Alexa enabled device. Amazon also controls and manages the Alexa skill for you, so there are minimal ways to modify the underlying architecture and create innovative ways to use the technology. This is fine in most cases, and it actually beneficial if you are looking to create a voice app with as little setup complexity as possible.

3.3 Google Home

Released in the same month as the Amazon Echo, Google Home is an alternative to the Amazon Echo. It utilizes Google Assistant which is commonly seen in recent Android devices. It has multiple uses similar to the Echo, i.e. the user is able to control home appliances such as lights, play music and interact with Google Assistant.

Google Home includes both first and third party services. Some examples are Spotify, Youtube, and Google Calendar. To create an app for Google Home there are three components involved, again similar to the Echo.

1. **Actions on Google developer project**

This allows the developer to manage the application, view analytics and also use the Actions Simulator, which lets the developer test the app without actually having a physical device. (Google, 2017).

2. **Action Package**

This defines the metadeta about the app including how Google Assistant invokes the actions and how it should call your fulfillment service. (Google, 2017)

3. **Fulfillment**

FulFillment is the main component, which defines the functionality of the app. It is hosted on an HTTPS web service and it is defined by the Actions Protocol in a JSON structure (Google, 2017). This works in a similar way to the Amazon Echo in regards to the JSON structure.

To conclude, both services work in a similar way, they both use JSON and both platforms utilize their own cloud services. Google allows apps to be hosted on Google Cloud and Amazon on their Amazon Web Services. One noticeable difference however is that the Google SDK works on external servers, whilst Alexa's SDK only operates on AWS.

3.4 Serverless Computing

Serverless Computing is an emerging technology which is quickly becoming popular. The name could be quite misleading, as there are still servers operating in the cloud. Essentially it is so developers do not have to set up and manage their own servers. Serverless computing works by executing code written by the developer almost in real-time, using only the resources it needs for that specific code. (Butler, B. 2017).

The two major front-runners for this emerging technology are Amazon, who have implemented serverless computing into their Amazon Web Servers, and Microsoft who have implemented it into Azure. They both work in a similar fashion. For this project we will be using AWS as we are developing with Alexa, also owned by Amazon. This should make different components easier to connect to each other and manage.

Amazon Web Services was introduced in 2014, and since then AWS Lambda has been the fastest growing service. 'Lambda is a high-scale, provision-free serverless compute offering based on functions. It provides the cloud logic layer for your application.' (Amazon, 2017). When using Lambda developers only pay for the computing time, so when your function is idle, you don't pay for it. Where Virtual Machines usually use a lot of power when turned on, Lambda reduces that by a considerable amount. Amazon Web Services focus on 'Serverless Computing'. Serverless computing allows the developer to create applications and focus on coding without having to worry about the server side of things. ‘Serverless applications don’t require you to provision, scale and manage any servers. You can build them for nearly any type of application or backend service.’ (Amazon, 2017).

The programming model adopted by Lambda is stateless, meaning that whenever a function is called it will always return the same result and is not affected by other functions that may have changed a variable, to be used later. The reason for the requirement of ‘stateless’ code is that it ‘enables AWS Lambda to launch as many copies of a function as needed to scale to the incoming rate of events and requests. These functions may not always run on the same compute instance from request to request’. (Amazon Web Services, 2017)

The downside of this requirement is that you can not keep the state of an application in variables within the program during the session. Instead, every change of state within the application must be stored somewhere, and every time a state needs to be changed, the corresponding data store must be updated. This requirement makes it very intensive for a game that is reliant on state, and limits it to games with less reliance on states, as they would require less data store updates.
This data storage could even be another AWS service, such as DynamoDB, explained by (Hammond, E. 2018).

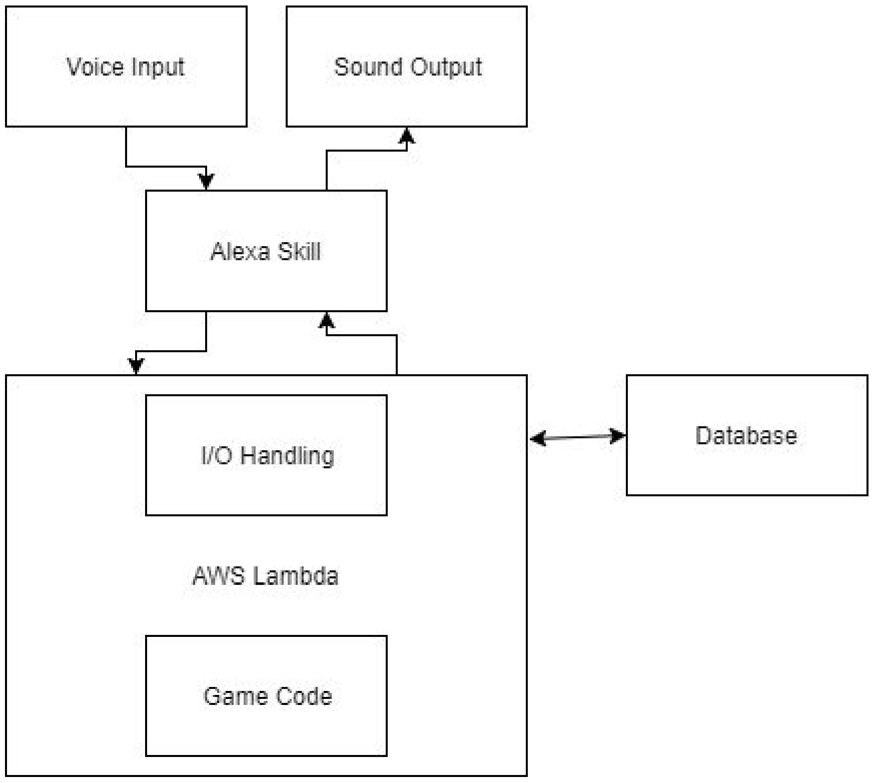
4. Implementation

We have developed a proof of concept implementation of a game, making use of Alexa and Serverless-computing to allow the interaction with the game to be voice-based. The game is based on classic text-based games such as Zork, where the user controls a character through a virtual world using only text based commands. Our game is similar in concept, except that all commands are now voice based. We implemented some basic commands; move, pickup, inventory, help and end game. Using these commands controls the ‘player’, allowing the user to move through rooms and pick up items. The purpose of this is a proof-of-concept to show the validity of using this architecture to implement a game. To achieve this, we used an Alexa skill to handle the input and output, with AWS lambda to handle the server side computation.

4.1 Inter-system Communication

Figure 2 shows the inter-system communication.

Figure 2. Flow of data between components



The Alexa skill handles all input and output, and acts as a gateway between the user and the AWS Lambda instance. The Lambda instance hosts all the game code. The input and output is handled on AWS Lambda, and is used to decide what game code needs to be ran. Output is then passed back through the Alexa skill where it is then sent to the user as speech. We have included in the diagram a database, although we do not actually make use of a database in our implementation. However, if needed, an application could connect a database to AWS Lambda to gain the ability to store information.

4.2 Alexa Input/Output

All requests to the game have to be made via Alexa, making it a requirement that the user owns an Alexa enabled device, such as the Echo Dot, a device that allows users to talk to Alexa Skills. The Alexa skill then handles all voice requests, converting the Sample Utterances into Intents.

Figure 3. Sample Utterances

```
StartGame start new game
StartGame start game
StartGame start a new game
SaveGame save the game
SaveGame save game
SaveGame save
EndGame end the game
EndGame end game
EndGame end
Move move
Pickup pick up
Pickup pick up item
```

Figure 4. Intents

```
"intents": [
  {
   "intent": "StartGame"
  },
  {
   "intent": "SaveGame"
  },
  {
   "intent": "EndGame"
  },
  {
   "intent": "Move"
```

```
Pickup take item
Pickup take
Inventory inventory
Inventory items
Help help
Description description
```

```
  },
  {
   "intent": "Pickup"
  },
  {
   "intent": "Inventory"
  },
  {
   "intent": "Help"
  },
  {
   "intent": "Description"
  }
]
```

The Sample Utterances define which voice input triggers which Intent. The first word per line in Figure 3 defines which Intent should be triggered. The subsequent words define which words need to be spoken. It accepts any number of utterances per intent, meaning that the voice input does not have to be strict. For example, we mapped both ‘Take’ and ‘Pickup’ to our ‘Pickup’ Intent. The Intents themselves are stored in JSON format. This is all handled and hosted by Amazon and uses their Voice-to-text technology. This means that setting up the Alexa Skill was easy and allowed us to focus on the code of the game.

Upon using the Alexa skill, and creating an Intent, the skill then creates a service request. The service request is in JSON format and contains data about the session, application, user, and type of request. Figure 5 shows the service request created by calling the ‘move’ function in our prototype.

Figure 5. Service Request

```
{
  "session": {
    "new": true,
    "sessionId": "SessionId.72ccb595-f3eb-4ce0-9252-cac2babd282c",
    "application": {
      "applicationId": "amzn1.ask.skill.d52fe833-3fe0-4303-8a2b-7a116355dcfd"
    },
    "attributes": {},
    "user": {
      "userId":
"amzn1.ask.account.AGIWYEQVJEKZV6GAIOZZSO3DFV5LZPEWQCP5FI7EPOH5P76EZEB
YJKXJBFIWJV7KJV5VZSO3KNALUBD7UQDRYSDCW3GPOVENLETVRPHVERRKNUAH
XZSFT7RJ33BGEVT5UWCDXQUOT325TOWYXVUV3TJHBK45EXAKC5LMJYRZ2UDNRG
6B55F7OTQG5UAP3ZT54MEXBPMLEBBJPUA"
    }
  },
  "request": {
    "type": "IntentRequest",
    "requestId": "EdwRequestId.dd19ad1f-0b48-46d4-832b-f94e4cafba44",
    "intent": {
      "name": "Move",
      "slots": {}
    },
    "locale": "en-GB",
    "timestamp": "2018-02-08T14:26:27Z"
  },
  "context": {
    "AudioPlayer": {
      "playerActivity": "IDLE"
    },
    "System": {
      "application": {
        "applicationId": "amzn1.ask.skill.d52fe833-3fe0-4303-8a2b-7a116355dcfd"
      },
      "user": {
        "userId":
"amzn1.ask.account.AGIWYEQVJEKZV6GAIOZZSO3DFV5LZPEWQCP5FI7EPOH5P76EZEB
YJKXJBFIWJV7KJV5VZSO3KNALUBD7UQDRYSDCW3GPOVENLETVRPHVERRKNUAH
XZSFT7RJ33BGEVT5UWCDXQUOT325TOWYXVUV3TJHBK45EXAKC5LMJYRZ2UDNRG
6B55F7OTQG5UAP3ZT54MEXBPMLEBBJPUA"
      },
      "device": {
        "supportedInterfaces": {}
      }
    }
  },
  "version": "1.0"
}
```

The most important aspects of the service request are the session ID and request type. The session ID enables the AWS lambda server to recognise and maintain a session between the user and the server. The request type tells the server what function it needs to carry out. In this case, the AWS Lambda instance returns the Service Response shown in Figure 6.

Figure 6. Service Response

```
{
  "version": "1.0",
  "response": {
    "outputSpeech": {
      "text": "The room is cramped, with barely any space to move at all. In the corner you see a
Key",
      "type": "PlainText"
    },
    "card": {
      "content": "The room is cramped, with barely any space to move at all. In the corner you see a
Key",
      "title": "move"
    },
    "reprompt": {
      "outputSpeech": {
        "text": "I didn't get that.",
        "type": "PlainText"
      }
    },
    "speechletResponse": {
      "outputSpeech": {
        "text": "The room is cramped, with barely any space to move at all. In the corner you see a
Key"
      },
      "card": {
        "content": "The room is cramped, with barely any space to move at all. In the corner you see a
Key",
        "title": "move"
      },
      "reprompt": {
        "outputSpeech": {
          "text": "I didn't get that."
        }
      },
      "shouldEndSession": false
    }
  },
  "sessionAttributes": {
    "RoomDescription": "cramped, with barely any space to move at all",
    "PlayerItems": [
      "testplayeritem"
    ],
    "RoomItems": [
      "a Key"
    ]
  }
}
```

This response gives the information needed to allow the skill to output speech, using the 'speechletResponse' attribute. These requests will be sent back and forth between the user, through the Alexa skill, to the AWS Lambda instance. The AWS Lambda instance processes the Service Request inputs and responds with a Service Response, determined by the code of the game.

4.3 Session Attributes

We used Python as the programming language for the game. We chose to program the game in an Object-Oriented fashion. However, we later discovered that all code hosted on AWS Lambda must be stateless due to its 'serverless' architecture. Being 'serverless' means that it is much easier to setup and manage the server, as you do not have to set up a server and any code required to manage connections. Instead, Amazon handles this for you. The code itself is not guaranteed to run on the same computational server every time, and therefore, variables may be lost in between requests. This lead us to rethink our game code design. We opted to continue in an object-oriented fashion, however, due to this new stateless requirement, we designed it to recreate all objects upon every request, through use of ‘sessionAttributes’.

The ‘sessionAttributes’ are a collection of attributes that are passed in both the Service Request and Service Response. By adding attributes to this, it enables us to store data during a session, and recreate variables and objects that are needed for the game upon every new request. You can see in Figure 6 that the ‘sessionAttributes’ has stored the Room Description, Player Items, and Room Items for the current state of the game. When the user makes a new request, this is sent back to the AWS Lambda instance, allowing the game code to recreate the state of the game, and then conduct the Intent. These session attributes are sent back and forth every request the user makes, and is not changed or removed unless done so in the game code. Essentially, this allows us to replicate the object-oriented structure within these session attributes and use them to replace instances of a class. Then, when you want to call a method on a class, you instantiate that class with all the information from the session attributes and save the resulting states in the session attributes too. This is a way to work around the stateless requirement. This use of the JSON data sent between AWS Lambda and the Alexa skill allows many more applications to run on a stateless, serverless architecture, without needing to save this information to a SQL database.

The downside to this solution, however, is that this data will not persist between sessions. This means that in order to save the state of the game to be played another day, or even later in the same day, would still require traditional uses of databases to create more persistent data storage. Our implementation does not utilise this, but any application could utilise both these ideas together to create robust, persistent, and serverless applications.

4.4 AWS Lambda Backend

The JSON Data is handled in the python code hosted on AWS Lambda, where it is converted to the relevant objects needed for the game, to ensure the game state is correct. Figure 7 shows the input and output handling.

Figure 7. Input and output handling

```
def lambda_handler(event, context):
    if event["session"]["new"]:
        sessionstarted({"requestId": event["request"]["requestId"]}, event["session"])

    if event["request"]["type"] == "LaunchRequest":
        return onlaunch(event["request"], event["session"])
    elif event["request"]["type"] == "IntentRequest":
        return onintent(event["request"], event["session"])
    elif event["request"]["type"] == "SessionEndedRequest":
        return sessionend(event["request"], event["session"])

def speechletresponse(title, output, reprompt, shouldendsession):
    return {
        "outputSpeech": {
            "type": "PlainText",
            "text": output
        },
        "card": {
            "type": "Simple",
            "title": title,
            "content": output
        },
        "reprompt": {
            "outputSpeech": {
                "type": "PlainText",
                "text": reprompt
            }
        },
        "shouldEndSession": shouldendsession
    }

def buildresponse(sessionattributes, speechlet):
    return {
        "version": "1.0",
        "sessionAttributes": sessionattributes,
        "response": speechlet
    }

def savesession(p):
    playeritems = []
    for i in p.items:
        playeritems.append(i.name)

    roomitems = []
    for i in p.room.items:
        roomitems.append(i.name)

    sessionattributes = {"RoomDescription": p.room.description,
        "RoomItems": roomitems,
        "PlayerItems": playeritems
    }
    return sessionattributes
```

Figure 8. Example game function

```
def move(session):
    p = Player.Player(session['attributes']['RoomDescription'], session['attributes']['RoomItems'],
session['attributes']['PlayerItems'])
    p.move()
    sessionattributes = {}
    title = "move"
    speechoutput = "The room is " + str(p.room.description) + ". In the corner you see "
    for i in p.room.items:
        speechoutput += i.name
    reprompt = "I didn't get that."
    shouldendsession = False
    sessionattributes = savesession(p)
    return buildresponse(sessionattributes, speechletresponse(
        title, speechoutput, reprompt, shouldendsession))
```

In figures 7 and 8, you can see how input is handled, used to determine the state of the game and the intent, and how it saves the session back into the JSON format for the response. In figure 7, the input is handled in function ‘lambda_handler’ and used to determine the type of request. If the type is an Intent, then it calls ‘onintent’ to determine what game code must be ran. An example game code function is shown in figure 8, where the state of the game is reconstructed using the session attributes from the input. Once it conducts the required code, which may alter the state of the game, the session attributes are re-created with the new states, through the ‘savesession’ function, and the response is built.

5. Evaluation

To demonstrate the validity of using AWS Lambda and Alexa, along with the use of session attributes to store game state, Figure 9 shows a transcript of a game session. It demonstrates using a few simple instructions included in the game. This includes moving into rooms and picking up items. By using the ‘inventory’ intent, you can see that the items were successfully picked up, and stored across multiple requests.

Figure 9. Game session transcript

Input: move
Output: The room is cramped, with barely any space to move at all. In the corner you see a Rake

Input: pickup
Output: You picked up the items

Input: inventory
Output: In your inventory you have, a Rake,

Input: move
Output: The room is smelly, like old oatmeal. In the corner you see a Gun

Input: pickup
Output: You picked up the items

Input: inventory
Output: In your inventory you have, a Rake, a Gun,

Input: end
Output: Game ended

Using AWS Lambda, rather than creating a server to host the game code, proved much easier and more time efficient. It resulted in more time to be dedicated to developing the game, without having to deal with the need for traditional ‘always on’ servers. This reduces both operational cost and complexity. When setting up AWS Lambda for this application, we simply had to sign up with AWS and create a new instance, with everything else being managed by Amazon. The game code could even be written directly into the browser, with the option of uploading a ZIP file. Overall this was very user friendly, but still came with some drawbacks. Although we managed to overcome the limitations to stateless programming, applications that exceed the complexity of our simple game may struggle to find relevant and useful ways to circumvent it.

Using Alexa for the input/output proved to be the simplest part of the system architecture. Creating a skill can be done in minutes and only requires a link to the AWS Lambda Instance. Creating Intents and Sample Utterances was simple due to the easy to understand data structure (JSON). Due to both Lambda and Alexa being created and owned by Amazon, the process of connecting the two was incredibly simple, through use of their ARN (Amazon Web Services, 2017).

The part with the most complexity was using session attributes to store state. However now that it has been implemented, it is very easy to extend this functionality and write code that can make use of it. We found it to be a very lightweight work-around to the stateless requirement, and an interesting use of the JSON data format. Due to JSON data being structured, much like Object Oriented design, it allowed the game state to be reconstructed with relative ease, allowing data to persist throughout a session without having to write to a database or file system.

As a bonus, we have found that original classic text-based games such as Zork could potentially be ran on this architecture. This could be done through the creation of an emulator that can handle input and output from Alexa, and runs on an AWS Lambda instance. This potentially opens up the game to a whole new audience who would prefer voice commands to text commands.

Future work for our game would include connecting to a database (likely DynamoDB, another AWS Service for simplicity) to ensure the user can save their progress between sessions, and adding more content and functionality to the game. This would further solidify our proof of concept as a viable solution, and act as a guide to creating simple, easy to setup, scale and deploy applications.

6. Conclusion

Overall, this proof of concept for creating a simple, easy to deploy application through the use of these technologies was a success. Any application built using AWS Lambda following this architectural structure and use of session attributes is likely to be very lightweight and easy to set up, potentially making it a go-to design for small, scalable applications. However, it is important to factor in the stateless code requirement and determine if session attributes are a viable solution for the application. If it is not viable, the application could still use traditional ‘always on’ servers, link into Alexa and gain all the functionality that Alexa Skills give.

7. References

1. Bulbapedia (2017) *Hey You, Pikachu!* Retrieved from https://bulbapedia.bulbagarden.net/wiki/Hey_You,_Pikachu!
2. Nintendo Wiki (Unknown) *Voice Recognition Unit*. Retrieved from http://nintendo.wikia.com/wiki/VRU
3. Human Interact (2017) *Starship Commander*. Retrieved from http://human-interact.com/starship-commander/
4. Starlanes (2017) *Starlanes Wikia*. Retrieved from http://starlanes.wikia.com/wiki/Starlanes_Wikia
5. Anders, G. (2017). Alexa, Understand Me. *MIT Technology Review*, 120(5), 26-31.
6. Amazon (2017) *Alexa Skills Set*. Retrieved from https://developer.amazon.com/alexa-skills-kit
7. Amazon (2017) *Serverless Architectures with AWS Lambda*. Retrieved from https://d1.awsstatic.com/whitepapers/serverless-architectures-with-aws-lambda.pdf
8. Butler, B. (2017) *Serverless Explainer, The next generation of cloud infrastructure.* Retrieved from https://www.networkworld.com/article/3187093/cloud-computing/serverless-explainer-the-next-generation-of-cloud-infrastructure.html
9. Amazon Web Services (2017) *Amazon Resource Names (ARNs) and AWS Service Namespaces*. Retrieved from https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html
10. Amazon Web Services (2017) *Programming Model*. Retrieved from https://docs.aws.amazon.com/lambda/latest/dg/programming-model-v2.html
11. Hammond, E. (2018) *Persistence Of The AWS Lambda Environment Between Function Invocations*. Retrieved from https://alestic.com/2014/12/aws-lambda-persistence/