**Music box skill – part one**



You tube video: <https://youtu.be/eRbn0711hK0>

Music box skill – I’ve called it johns music box

References:

Dabble labs

<https://dabblelab.com/tutorials/creating-an-alexa-audio-streaming-skill>

Alexa skills:

https://developer.amazon.com/en-US/docs/alexa/custom-skills/audioplayer-interface-reference.html

**https://github.com/alexa/skill-sample-python-audio-player**

**Instructions**

Create a new Alexa Hosted skill.

Check / Change the invocation

**Now add audio player interface**. This adds the Pause, Resume intents. We’ll add code for those later.

Click Interface > audio player > switch On. Save interfaces, and build the model

Save model

Change intent json code to:

{

"interactionModel": {

"languageModel": {

"invocationName": "johns music box",

"intents": [

{

"name": "PlayAudio",

"slots": [],

"samples": [

"play my radio",

"start my radio",

"start the audio",

"play the audio",

"start the music",

"play the music",

"play my music"

]

},

{

"name": "AMAZON.PauseIntent",

"samples": []

},

{

"name": "AMAZON.ResumeIntent",

"samples": []

},

{

"name": "AMAZON.HelpIntent",

"samples": [

"how can I listen to my radio",

"how do I play my music",

"tell me how to play",

"how do I listen to my music"

]

},

{

"name": "AMAZON.StopIntent",

"samples": []

},

{

"name": "AMAZON.CancelIntent",

"samples": []

},

{

"name": "AMAZON.StartOverIntent",

"samples": []

},

{

"name": "AMAZON.FallbackIntent",

"samples": []

},

{

"name": "AMAZON.NavigateHomeIntent",

"samples": []

}

],

"types": []

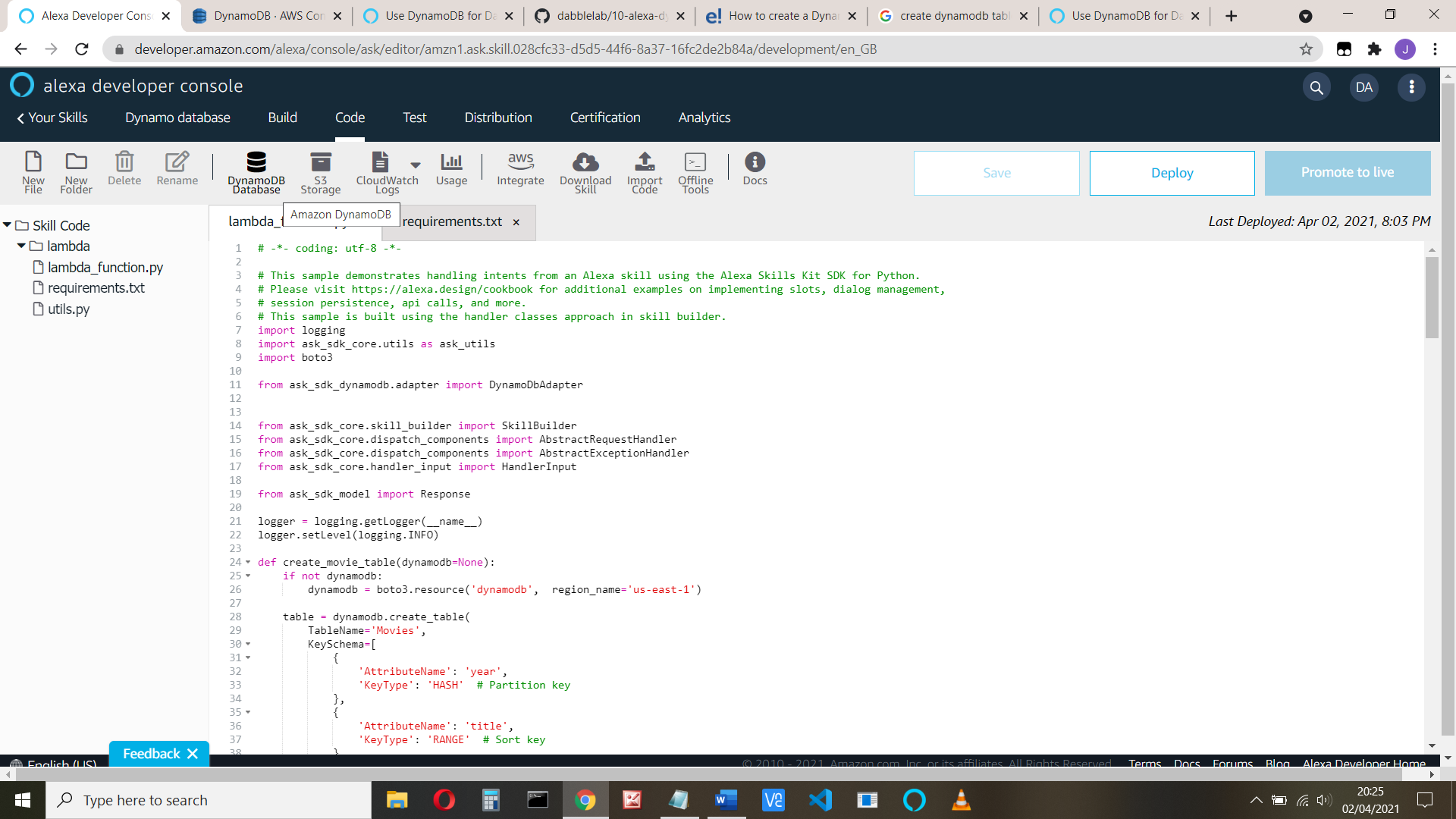
}

}

}

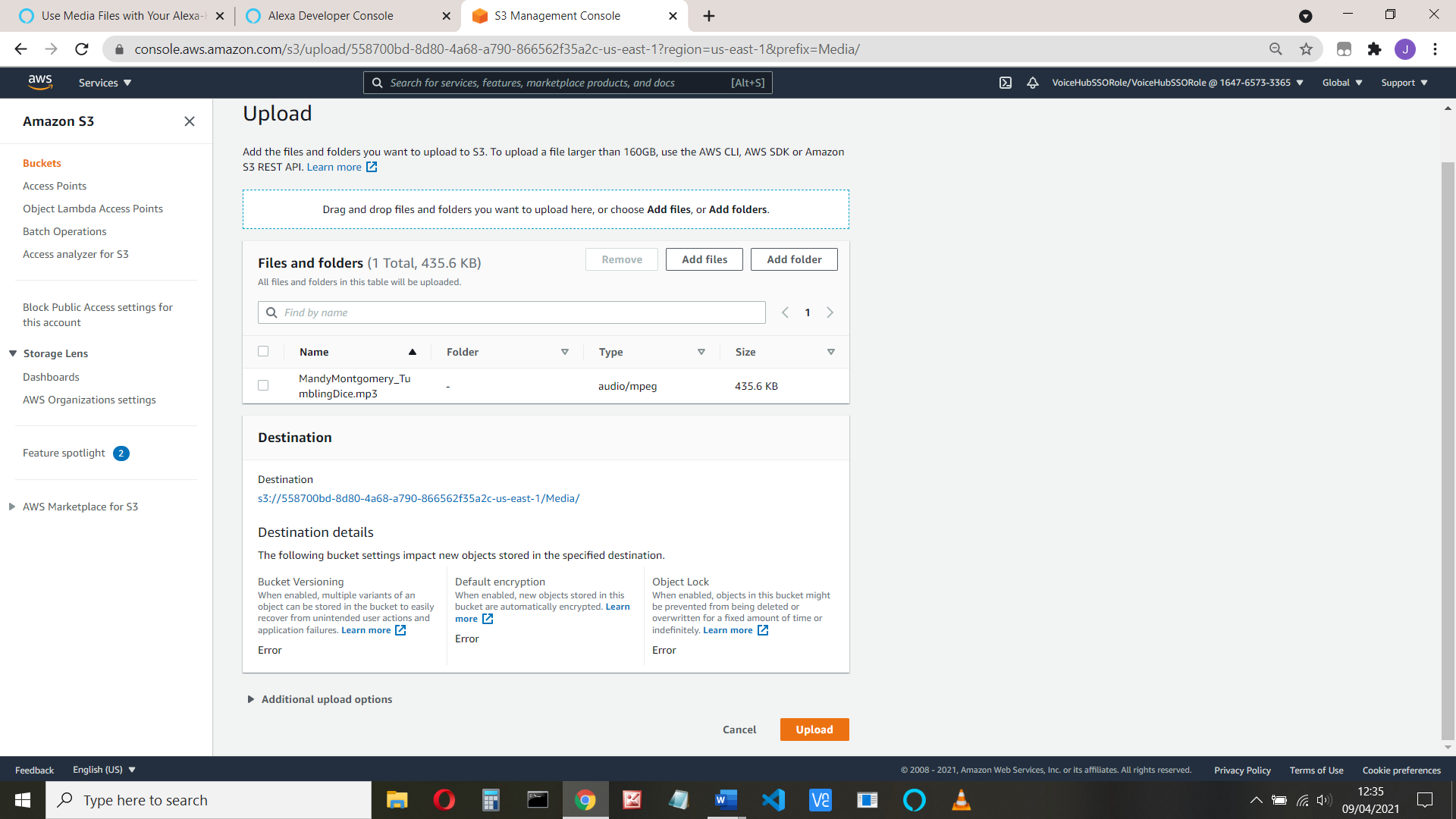
Save and build model

Upload music to S3 Storage



Click the S3 storage tab. It opens in a new window.

Upload your music and images too, if you want them.



Save the end part of the url e.g. Media/RemovalMen\_Borderline.mp3

Or click ‘key’ on the bottom left hand side.

Media/RemovalMen\_Borderline.mp3

Media/waveform108.png

Media/waveform512.png

See <https://developer.amazon.com/es-MX/docs/alexa/hosted-skills/alexa-hosted-skills-media-files.html>

The full code follows, but the important bits are:

1. Launch event – Changed to prompt to play your music.
2. AudioPlayIntentHandler – runs when requested. Executes the PlayDirective. This accesses the music stored in S3 with a presigned url, which gives the program access to the storage. This has a lifetime of 60 seconds, determined in ExpiresIn in utils.py

N.b **The token cannot exceed 1024 characters.**

1. The audioplayer intent handlers – started, nearly finished, stopped etc. All used by the Alexa code. These are added in the skill builder code

Copy the code, replace your url values, deploy and test in developer, enabling skill testing to Development. Check that is does play your music

Next video we’ll add more tracks and a music database.

# -\*- coding: utf-8 -\*-

# This sample demonstrates handling intents from an Alexa skill using the Alexa Skills Kit SDK for Python.

# Please visit https://alexa.design/cookbook for additional examples on implementing slots, dialog management,

# session persistence, api calls, and more.

# This sample is built using the handler classes approach in skill builder.

import logging

import ask\_sdk\_core.utils as ask\_utils

**from utils import create\_presigned\_url**

from ask\_sdk\_core.skill\_builder import SkillBuilder

from ask\_sdk\_core.dispatch\_components import AbstractRequestHandler

from ask\_sdk\_core.dispatch\_components import AbstractExceptionHandler

from ask\_sdk\_core.handler\_input import HandlerInput

from ask\_sdk\_model import Response

logger = logging.getLogger(\_\_name\_\_)

logger.setLevel(logging.INFO)

from ask\_sdk\_core.utils import is\_intent\_name

from ask\_sdk\_core.utils import get\_intent\_name

from ask\_sdk\_core.utils import is\_request\_type

from ask\_sdk\_model.ui import StandardCard, Image, SimpleCard

**from ask\_sdk\_model.interfaces.audioplayer import (**

**PlayDirective, PlayBehavior, AudioItem, Stream, AudioItemMetadata,**

**StopDirective)**

from ask\_sdk\_model.interfaces import display

**small\_image\_url = create\_presigned\_url("Media/waveform108.png")**

**large\_image\_url = create\_presigned\_url("Media/waveform512.png")**

audio\_data = {

"card": {

"title": 'My music',

"text": 'I like music',

}

}

card = StandardCard(

title=audio\_data["card"]["title"],

text=audio\_data["card"]["text"],

image=Image(

**small\_image\_url=small\_image\_url,**

**large\_image\_url=large\_image\_url**

)

)

class **LaunchRequestHandler**(AbstractRequestHandler):

"""Handler for Skill Launch."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("LaunchRequest")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "Hello, I can play your music, just say play my music"

return (

handler\_input.response\_builder

.speak(speak\_output)

.ask(speak\_output)

.response

)

class HelpIntentHandler(AbstractRequestHandler):

"""Handler for Help Intent."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return ask\_utils.is\_intent\_name("AMAZON.HelpIntent")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "Just say play my music"

return (

handler\_input.response\_builder

.speak(speak\_output)

.ask(speak\_output)

.response

)

class **AudioPlayIntentHandler**(AbstractRequestHandler):

# Handler for Audioplayer Play Intent

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_intent\_name("PlayAudio")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("in AudioPlayIntent")

speech\_text = "Welcome to my audio player"

**audio\_key = "Media/RemovalMen\_Borderline.mp3"**

**audio\_url = create\_presigned\_url(audio\_key)**

directive = **PlayDirective**(

play\_behavior=PlayBehavior.REPLACE\_ALL,

audio\_item=AudioItem(

stream=Stream(

**token=audio\_key,**

**url=audio\_url,**

offset\_in\_milliseconds=0,

expected\_previous\_token=None),

metadata=None))

handler\_input.response\_builder.speak(speech\_text).set\_card(card).add\_directive(directive).set\_**should\_end\_session(True)**

logger.info("response")

logger.info(handler\_input.response\_builder.response)

return handler\_input.response\_builder.response

class **AudioStopIntentHandler**(AbstractRequestHandler):

# Handler for Stop – come here on pause or cancel too

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return (is\_intent\_name("AMAZON.CancelIntent")(handler\_input) or

is\_intent\_name("AMAZON.StopIntent")(handler\_input) or

is\_intent\_name("AMAZON.PauseIntent")(handler\_input))

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("in AudioStopIntent and request")

logger.info(handler\_input.request\_envelope.request)

speech\_text = "Goodbye"

directive = StopDirective()

handler\_input.response\_builder.speak(speech\_text).add\_directive(

directive).set\_should\_end\_session(True)

return handler\_input.response\_builder.response

**# ########## AUDIOPLAYER INTERFACE HANDLERS #########################**

# from https://github.com/alexa/skill-sample-python-audio-player

# This section contains handlers related to Audioplayer interface

class PlaybackStartedHandler(AbstractRequestHandler):

"""AudioPlayer.PlaybackStarted Directive received.

Confirming that the requested audio file began playing.

Do not send any specific response.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackStarted")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In PlaybackStartedHandler")

return handler\_input.response\_builder.response

class PlaybackFinishedHandler(AbstractRequestHandler):

"""AudioPlayer.PlaybackFinished Directive received.

Confirming that the requested audio file completed playing.

Do not send any specific response.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackFinished")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In PlaybackFinishedHandler")

return handler\_input.response\_builder.response

class PlaybackStoppedHandler(AbstractRequestHandler):

"""AudioPlayer.PlaybackStopped Directive received.

Confirming that the requested audio file stopped playing.

Do not send any specific response.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackStopped")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In PlaybackStoppedHandler and request is")

logger.info(handler\_input.request\_envelope.request)

return handler\_input.response\_builder.response

class PlaybackNearlyFinishedHandler(AbstractRequestHandler):

"""AudioPlayer.PlaybackNearlyFinished Directive received.

Replacing queue with the URL again. This should not happen on live streams.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackNearlyFinished")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In PlaybackNearlyFinishedHandler")

audio\_key = "Media/RemovalMen\_Borderline.mp3"

audio\_url = create\_presigned\_url(audio\_key)

directive = PlayDirective(

play\_behavior=PlayBehavior.REPLACE\_ENQUEUED,

audio\_item=AudioItem(

stream=Stream(

token=audio\_key,

url=audio\_url,

offset\_in\_milliseconds=0,

expected\_previous\_token=None),

metadata=None))

handler\_input.response\_builder.set\_card(card).add\_directive(directive).set\_should\_end\_session(True)

return handler\_input.response\_builder.response

class PlaybackFailedHandler(AbstractRequestHandler):

"""AudioPlayer.PlaybackFailed Directive received.

Logging the error and stoprestarting playing with no output speech and card.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackFailed")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

request = handler\_input.request\_envelope.request

logger.info("Playback failed: {}".format(request.error))

return handler\_input.response\_builder.response

class ExceptionEncounteredHandler(AbstractRequestHandler):

"""Handler to handle exceptions from responses sent by AudioPlayer

request.

"""

def can\_handle(self, handler\_input):

# type; (HandlerInput) -> bool

return is\_request\_type("System.ExceptionEncountered")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EXCEPTION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

logger.info(handler\_input.request\_envelope)

return handler\_input.response\_builder.response

# ###################################################################

class SessionEndedRequestHandler(AbstractRequestHandler):

"""Handler for Session End."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return ask\_utils.is\_request\_type("SessionEndedRequest")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("in SessionEnded")

# Any cleanup logic goes here.

return handler\_input.response\_builder.response

class IntentReflectorHandler(AbstractRequestHandler):

"""The intent reflector is used for interaction model testing and debugging.

It will simply repeat the intent the user said. You can create custom handlers

for your intents by defining them above, then also adding them to the request

handler chain below.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("IntentRequest")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

intent\_name = ask\_utils.get\_intent\_name(handler\_input)

speak\_output = "You just triggered " + intent\_name + "."

return (

handler\_input.response\_builder

.speak(speak\_output)

# .ask("add a reprompt if you want to keep the session open for the user to respond")

.response

)

class CatchAllExceptionHandler(AbstractExceptionHandler):

"""Generic error handling to capture any syntax or routing errors. If you receive an error

stating the request handler chain is not found, you have not implemented a handler for

the intent being invoked or included it in the skill builder below.

"""

def can\_handle(self, handler\_input, exception):

# type: (HandlerInput, Exception) -> bool

return True

def handle(self, handler\_input, exception):

# type: (HandlerInput, Exception) -> Response

logger.error(exception, exc\_info=True)

logger.info("exception")

logger.info(exception)

#logger.info(ask\_utils.get\_intent\_name(handler\_input))

#The provided request is not an IntentRequest

logger.info(ask\_utils.get\_request\_type(handler\_input))

speak\_output = "Sorry, I had trouble doing what you asked. Please try again."

return (

handler\_input.response\_builder

.speak(speak\_output)

.ask(speak\_output)

.response

)

# The SkillBuilder object acts as the entry point for your skill, routing all request and response

# payloads to the handlers above. Make sure any new handlers or interceptors you've

# defined are included below. The order matters - they're processed top to bottom.

sb = SkillBuilder()

sb.add\_request\_handler(LaunchRequestHandler())

sb.add\_request\_handler(HelpIntentHandler())

sb.add\_request\_handler(AudioPlayIntentHandler())

sb.add\_request\_handler(AudioStopIntentHandler())

# ########## AUDIOPLAYER INTERFACE HANDLERS #########################

sb.add\_request\_handler(PlaybackStartedHandler())

sb.add\_request\_handler(PlaybackFinishedHandler())

sb.add\_request\_handler(PlaybackStoppedHandler())

sb.add\_request\_handler(PlaybackNearlyFinishedHandler())

sb.add\_request\_handler(PlaybackFailedHandler())

sb.add\_request\_handler(ExceptionEncounteredHandler())

sb.add\_request\_handler(SessionEndedRequestHandler())

sb.add\_request\_handler(IntentReflectorHandler()) # make sure IntentReflectorHandler is last so it doesn't override your custom intent handlers

sb.add\_exception\_handler(CatchAllExceptionHandler())

lambda\_handler = sb.lambda\_handler()

If you get an error look at the cloudwatch logs. Note if you are in the UK, click the down arrow and choose EU.

**Part 2 Add more music and resume, next, previous and startOver intents**



Part two in action at: <https://youtu.be/3wu2-m2KZT0>

Part two (first part) coding example at: <https://youtu.be/ArT9sH1BM20>

Part two (second part) coding example at: <https://youtu.be/AU8HMycFcYQ>

**We’ll look at : play, stop, resume, startOver, next, previous, nearly finished**

Available at : **https://github.com/jallwork/alexa-hosted-python-audio-skill**

Firstly, set up the intents.

**Change the intent JSON to add the required intents (and a few others):**

{

"interactionModel": {

"languageModel": {

"invocationName": "johns music box",

"intents": [

{

"name": "PlayAudio",

"slots": [],

"samples": [

"play my radio",

"start my radio",

"start the audio",

"play the audio",

"start the music",

"play the music",

"play my music"

]

},

{

"name": "AMAZON.PauseIntent",

"samples": []

},

{

"name": "AMAZON.ResumeIntent",

"samples": []

},

{

"name": "AMAZON.HelpIntent",

"samples": [

"how can I listen to my radio",

"how do I play my music",

"tell me how to play",

"how do I listen to my music"

]

},

{

"name": "AMAZON.StopIntent",

"samples": []

},

{

"name": "AMAZON.CancelIntent",

"samples": []

},

{

"name": "AMAZON.StartOverIntent",

"samples": [

"start again",

"go to the beginning",

"start over"

]

},

{

"name": "AMAZON.PreviousIntent",

"samples": []

},

{

"name": "AMAZON.NextIntent",

"samples": []

},

{

"name": "AMAZON.FallbackIntent",

"samples": []

},

{

"name": "AMAZON.NavigateHomeIntent",

"samples": []

}

],

"types": []

}

}

}

**Upload Music**

Upload some more music to your S3 first of all. Remember the key (the url) for each.

Let’s save our music data. It’s a dictionary rather than a database. Each track\_info has genre, title, artist and url. We’ll use a multi-dimensional dictionary:

track\_info = [

{

"genre" : "Blues",

"title" : "You Left all the Water running",

"artist" :"Removal Men",

"url" : "Media/RemovalMen\_YouLeftAllTheWaterRunning.mp3"

},

{

"genre": "Rock",

"title" : "Tell Momma",

"artist" :"Removal Men",

"url" :"Media/RemovalMen\_TellMomma.mp3"

},

{

"genre":"Blues",

"title" :"Cant find my way home",

"artist" : "Mandy Montgomery",

"url" :"Media/MandyMontgomery\_CantFindMyWayHome.mp3"

},

{

"genre" :"Rock",

"title" : "Tumbling dice",

"artist" : "Mandy Montgomery",

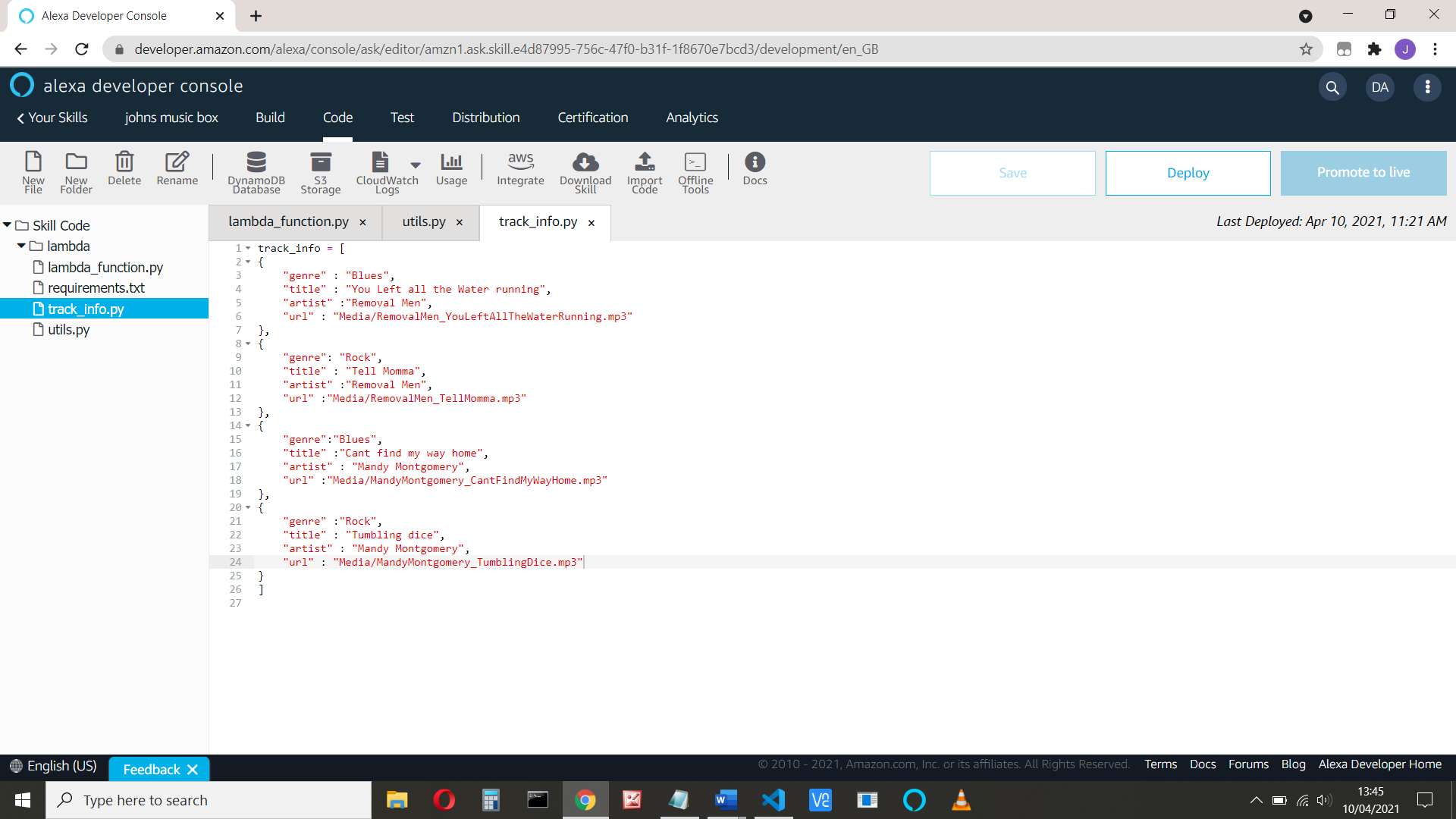
"url" : "Media/MandyMontgomery\_TumblingDice.mp3"

}

]

Save this in a **separate file** called track\_info.py in your code. (Create a new file from the top left hand corner)

Remember, you will have to change the URLs in my file to those from your S3 bucket.



To access this, we will add ‘**import track\_info as trackInfo**’ at the top of the code

We’ll use persistence attributes to load and save our meta-data and the offset if paused.

Reference: <https://developer.amazon.com/en-US/docs/alexa/hosted-skills/alexa-hosted-skills-session-persistence.html#persistence-python>

We’ll store **playback\_settings** (token, offset, URL and Enqueued state) and **track\_number** which holds the current track being played. To read them we use:

persistence\_attr = handler\_input.attributes\_manager.persistent\_attributes

and then we can use for example,

track\_number = persistence\_attr['track\_number'] offset\_in\_milliseconds=int(persistence\_attr["playback\_settings"]["offset\_in\_milliseconds"])

To save them we use

persistence\_attr['track\_number'] = track\_number

persistence\_attr["playback\_settings"]["url"] = **audio\_url**

persistence\_attr["playback\_settings"]["token"] = **audio\_key**

persistence\_attr["playback\_settings"]["offset\_in\_milliseconds"] =

handler\_input.request\_envelope.request.offset\_in\_milliseconds

handler\_input.attributes\_manager.persistent\_attributes = persistence\_attr

handler\_input.attributes\_manager.save\_persistent\_attributes()

We will also need to set them up initially. We’ll use **interceptors** to load and save our persistent data. We add more ‘from’ s at the top of the code:

from ask\_sdk\_core.dispatch\_components import AbstractRequestInterceptor

from ask\_sdk\_core.dispatch\_components import AbstractResponseInterceptor

Add the intercept code before the skill builder:

class LoadPersistenceAttributesRequestInterceptor(AbstractRequestInterceptor):

#Check if user is invoking skill for first time and initialize preset

def process(self, handler\_input):

# type: (HandlerInput) -> None

#handler\_input.attributes\_manager.delete\_persistent\_attributes()

persistence\_attr = handler\_input.attributes\_manager.persistent\_attributes

if len(persistence\_attr) == 0:

logger.info("Create attributes")

# First time skill user

persistence\_attr["playback\_settings"] = {

"token": None,

"offset\_in\_milliseconds": 0,

"url" : None,

"next\_stream\_enqueued" : False

}

persistence\_attr["track\_number"] = 0

else:

# Convert decimals to integers, because of AWS SDK DynamoDB issue

# https://github.com/boto/boto3/issues/369

pass

return

class SavePersistenceAttributesResponseInterceptor(AbstractResponseInterceptor):

#Save persistence attributes before sending response to user.

def process(self, handler\_input, response):

# type: (HandlerInput, Response) -> None

handler\_input.attributes\_manager.persistent\_attributes = persistence\_attr

handler\_input.attributes\_manager.save\_persistent\_attributes()

return

Add the sb intercept code:

# Interceptors

sb.add\_global\_request\_interceptor(LoadPersistenceAttributesRequestInterceptor())

sb.add\_global\_response\_interceptor(SavePersistenceAttributesResponseInterceptor())

All this requires access to the DynamoDb table, see:

<https://developer.amazon.com/en-US/docs/alexa/hosted-skills/alexa-hosted-skills-session-persistence.html#persistence-python>

**Add the db reference to the requirements.txt.** Note that boto3 is already there.

ask-sdk-dynamodb-persistence-adapter==1.15.0

**Add the following code to the top of lamba\_function.py**

import os

import boto3

from ask\_sdk\_dynamodb.adapter import DynamoDbAdapter

**and the code to initialise the dDB**

ddb\_region = os.environ.get('DYNAMODB\_PERSISTENCE\_REGION')

ddb\_table\_name = os.environ.get('DYNAMODB\_PERSISTENCE\_TABLE\_NAME')

ddb\_resource = boto3.resource('dynamodb', region\_name=ddb\_region)

dynamodb\_adapter = DynamoDbAdapter(table\_name=ddb\_table\_name, create\_table=False, dynamodb\_resource=ddb\_resource)

from ask\_sdk\_core.skill\_builder import CustomSkillBuilder

from ask\_sdk\_dynamodb.adapter import DynamoDbAdapter

**Add the CustomSkillBuilder** to your skill builder request handlers, and the bottom of the code.

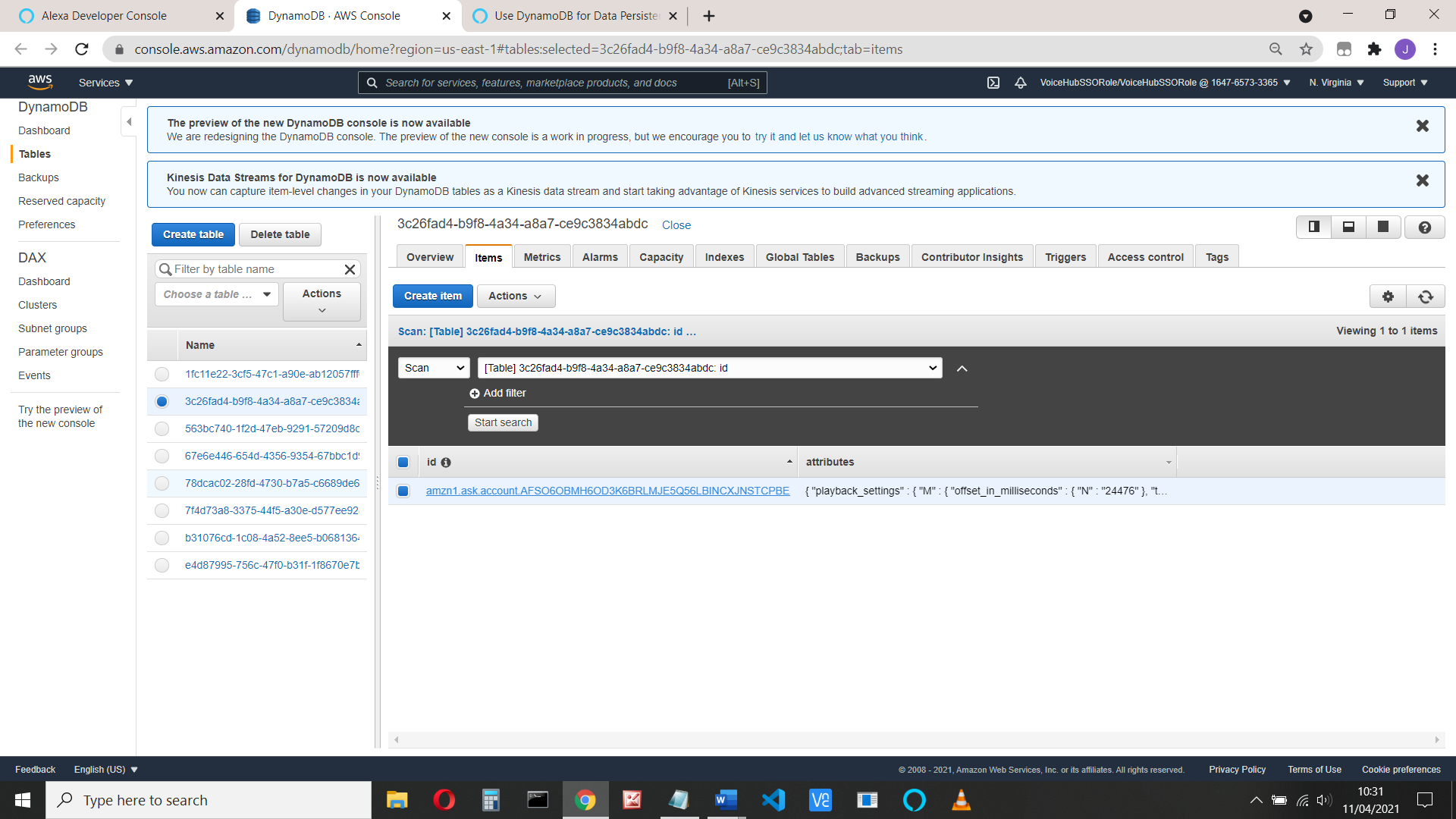
sb = CustomSkillBuilder(persistence\_adapter = dynamodb\_adapter)

Remember, we have **persistence\_attr['track\_number']** which holds the track number being played and **persistence\_attr['playback\_settings']** which will hold the current playback information (token, offset and url) if the music is paused for any reason.

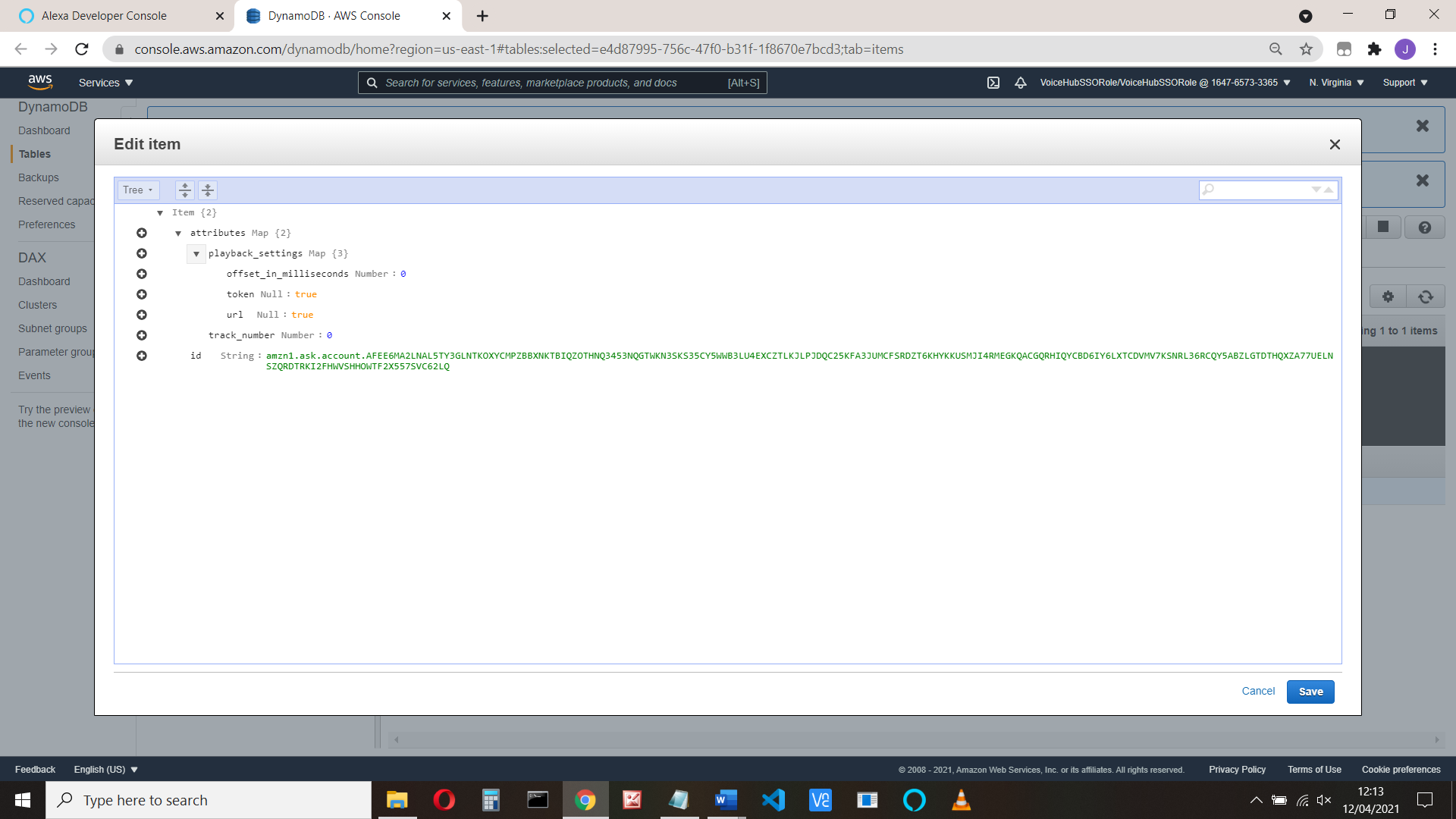
We can now find the track url using:

**audio\_key** = trackInfo.track\_info[track\_number]['url']

When the program has created some persistence data, you can view it. Click the DynamoDb icon and the database is shown:



You can then expand and edit the data



Now we want to add the resume and play commands.

**PlaybackStoppedHandler**

When we get the playback stopped, we’ll save the offset in playback\_settings.

class PlaybackStoppedHandler(AbstractRequestHandler):

"""AudioPlayer.PlaybackStopped Directive received.

Save playback for resume

Response example is

{

"type": "AudioPlayer.PlaybackStopped",

"requestId": "unique.id.for.the.request",

"timestamp": "timestamp of request in format: 2018-04-11T15:15:25Z",

"token": "token representing the currently playing stream",

"offsetInMilliseconds": offset in milliseconds,

"locale": "a locale code such as en-US"

}

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackStopped")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In PlaybackStoppedHandler")

# Save offset\_in\_milliseconds for Resume

persistence\_attr = handler\_input.attributes\_manager.persistent\_attributes

**persistence\_attr["playback\_settings"]["offset\_in\_milliseconds"] = handler\_input.request\_envelope.request.offset\_in\_milliseconds**

handler\_input.attributes\_manager.persistent\_attributes = persistence\_attr

return handler\_input.response\_builder.response

**AudioPlayIntentHandler**

We’ll modify the AudioPlayIntentHandler so that it starts from the beginning of track if the user says ‘Play’, but from the saved point if the user says ‘Resume’. We know from the intent if the user command was Play or Resume

(e.g. if is\_intent\_name("AMAZON.ResumeIntent") )

Add a new or change AudioPlayIntent to the following

class AudioPlayIntentHandler(AbstractRequestHandler):

# Handler for Audioplayer Play Intent and RESUME

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return (is\_intent\_name("PlayAudio")(handler\_input) or

is\_intent\_name("AMAZON.ResumeIntent")(handler\_input))

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("in AudioPlayIntent")

persistence\_attr = handler\_input.attributes\_manager.persistent\_attributes

if (is\_intent\_name("PlayAudio")(handler\_input)):

logger.info("play Audio")

# first time - set track to zero, previous saved track number ignored

**track\_number = 0**

**persistence\_attr["track\_number"] = track\_number**

card = StandardCard(

title=trackInfo.track\_info[track\_number]["title"],

text=trackInfo.track\_info[track\_number]["artist"],

image=Image(

small\_image\_url=small\_image\_url,

large\_image\_url=large\_image\_url

)

)

**audio\_key = trackInfo.track\_info[track\_number]["url"]**

**audio\_url = create\_presigned\_url(audio\_key)**

persistence\_attr["playback\_settings"]["url"] = audio\_url

persistence\_attr["playback\_settings"]["token"] = audio\_key

persistence\_attr["playback\_settings"]["offset\_in\_milliseconds"] = 0

**persistence\_attr["playback\_settings"]["next\_stream\_enqueued"] = False**

# REPLACE\_ALL - replace current and enqueued streams

speech\_text = "Playing your music"

directive = PlayDirective(

play\_behavior=PlayBehavior.REPLACE\_ALL,

audio\_item=AudioItem(

stream=Stream(

token=audio\_key,

url=audio\_url,

offset\_in\_milliseconds=0,

expected\_previous\_token=None),

metadata=None))

handler\_input.response\_builder.speak(speech\_text).set\_card(card).add\_directive(directive).set\_should\_end\_session(True)

else:

# resume

**track\_number = int(persistence\_attr["track\_number"])**

**audio\_key = trackInfo.track\_info[track\_number]["url"]**

**audio\_url = create\_presigned\_url(audio\_key)**

persistence\_attr["playback\_settings"]["url"] = audio\_url

persistence\_attr["playback\_settings"]["token"] = audio\_key

card = StandardCard(

title=trackInfo.track\_info[track\_number]["title"],

text=trackInfo.track\_info[track\_number]["artist"],

image=Image(

small\_image\_url=small\_image\_url,

large\_image\_url=large\_image\_url

)

)

directive = PlayDirective(

play\_behavior=PlayBehavior.REPLACE\_ALL,

audio\_item=AudioItem(

stream=Stream(

token=audio\_key,

url=audio\_url,

**offset\_in\_milliseconds=persistence\_attr["playback\_settings"]["offset\_in\_milliseconds"],**

expected\_previous\_token=None),

metadata=None))

handler\_input.response\_builder.set\_card(card).add\_directive(directive).set\_should\_end\_session(True)

handler\_input.attributes\_manager.persistent\_attributes = persistence\_attr

return handler\_input.response\_builder.response

**Pause and Resume**

**N.b. Resume doesn’t work correctly with echo dot:**

<https://forums.developer.amazon.com/questions/68678/audio-player-not-honoring-offsetinmilliseconds.html>

When the user says Stop or Pause, a PlaybackStoppedHandler event occurs. This event contains the current offset in milliseconds for the track playing. We store this in

*persistence\_attr["playback\_settings"]["offset\_in\_milliseconds"]*

which we retrieve when we resume. The PlaybackStoppedHandler code is:

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackStopped")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

persistence\_attr = handler\_input.attributes\_manager.persistent\_attributes

**persistence\_attr["playback\_settings"]["offset\_in\_milliseconds"] = handler\_input.request\_envelope.request.offset\_in\_milliseconds**

handler\_input.attributes\_manager.persistent\_attributes = persistence\_attr

return handler\_input.response\_builder.response

**Add Next and Previous**

For the next track command, we increment the current track number, resetting to zero if it’s at the end and play the new track

**The code is**

class NextPlaybackHandler(AbstractRequestHandler):

"""

Handles Next Intent

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return (is\_intent\_name("AMAZON.NextIntent")(handler\_input))

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

persistence\_attr = handler\_input.attributes\_manager.persistent\_attributes

**track\_number = int(persistence\_attr["track\_number"])**

**next\_track = (track\_number + 1) % len(trackInfo.track\_info)**

**persistence\_attr["track\_number"] = next\_track**

track\_number = next\_track # for consistency below

audio\_key = trackInfo.track\_info[track\_number]["url"]

audio\_url = create\_presigned\_url(audio\_key)

persistence\_attr["playback\_settings"]["offset\_in\_milliseconds"] = 0

persistence\_attr["playback\_settings"]["url"] = audio\_url

persistence\_attr["playback\_settings"]["token"] = audio\_key

**persistence\_attr["playback\_settings"]["next\_stream\_enqueued"] = False**

handler\_input.attributes\_manager.persistent\_attributes = persistence\_attr

card = StandardCard(

title=trackInfo.track\_info[track\_number]["title"],

text=trackInfo.track\_info[track\_number]["artist"],

image=Image(

small\_image\_url=small\_image\_url,

large\_image\_url=large\_image\_url

)

)

directive = PlayDirective(

play\_behavior=PlayBehavior.REPLACE\_ALL,

audio\_item=AudioItem(

stream=Stream(

token=audio\_key,

url=audio\_url,

offset\_in\_milliseconds=0,

expected\_previous\_token=None),

metadata=None))

handler\_input.response\_builder.set\_card(card).add\_directive(directive).set\_should\_end\_session(True)

return handler\_input.response\_builder.response

We’ll look at the *persistence\_attr["playback\_settings"]["next\_stream\_enqueued"] = False* code soon.

**and ditto for the previous - just go back to zero:**

class PreviousPlaybackHandler(AbstractRequestHandler):

"""

Handler for Playing previous

if already at first track, it just stays there

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return (is\_intent\_name("AMAZON.PreviousIntent")(handler\_input))

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In PreviousPlaybackHandler")

persistence\_attr = handler\_input.attributes\_manager.persistent\_attributes

**track\_number = int(persistence\_attr["track\_number"])**

**next\_track = track\_number -1**

**if (next\_track) <0:**

**next\_track = 0**

**persistence\_attr["track\_number"] = next\_track**

track\_number = next\_track # for consistency below

audio\_key = trackInfo.track\_info[track\_number]["url"]

audio\_url = create\_presigned\_url(audio\_key)

persistence\_attr["playback\_settings"]["offset\_in\_milliseconds"] = 0

persistence\_attr["playback\_settings"]["url"] = audio\_url

persistence\_attr["playback\_settings"]["token"] = audio\_key

**persistence\_attr["playback\_settings"]["next\_stream\_enqueued"] = False**

card = StandardCard(

title=trackInfo.track\_info[track\_number]["title"],

text=trackInfo.track\_info[track\_number]["artist"],

image=Image(

small\_image\_url=small\_image\_url,

large\_image\_url=large\_image\_url

)

)

handler\_input.attributes\_manager.persistent\_attributes = persistence\_attr

directive = PlayDirective(

play\_behavior=PlayBehavior.REPLACE\_ALL,

audio\_item=AudioItem(

stream=Stream(

token=audio\_key,

url=audio\_url,

offset\_in\_milliseconds=0,

expected\_previous\_token=None),

metadata=None))

handler\_input.response\_builder.set\_card(card).add\_directive(directive).set\_should\_end\_session(True)

return handler\_input.response\_builder.response

We’re deliberately getting a lot of duplicate code here, look at the Alexa github example if you want neat code. **https://github.com/alexa/skill-sample-python-audio-player**

**Add the StartOver code. This starts the track from the beginning**

class StartOverHandler(AbstractRequestHandler):

"""Handler for start over."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return (is\_intent\_name("AMAZON.StartOverIntent")(handler\_input))

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In StartOverHandler")

persistence\_attr = handler\_input.attributes\_manager.persistent\_attributes

**track\_number = int(persistence\_attr['track\_number'])**

audio\_key = trackInfo.track\_info[track\_number]['url']

audio\_url = create\_presigned\_url(audio\_key)

persistence\_attr["playback\_settings"]["url"] = audio\_url

persistence\_attr["playback\_settings"]["token"] = audio\_key

**persistence\_attr["playback\_settings"]["offset\_in\_milliseconds"] = 0**

handler\_input.attributes\_manager.persistent\_attributes = persistence\_attr

directive = PlayDirective(

play\_behavior=PlayBehavior.REPLACE\_ALL,

audio\_item=AudioItem(

stream=Stream(

token=audio\_key,

url=audio\_url,

**offset\_in\_milliseconds=0,**

expected\_previous\_token=None),

metadata=None))

handler\_input.response\_builder.set\_card(card).add\_directive(directive).set\_should\_end\_session(True)

return handler\_input.response\_builder.response

**PlaybackNearlyFinished handler**

This adds the new stream to the queue. Alexa then automatically plays this track when the current one finishes.

The problem I have is that (as I use track number to get the next track) things can get out of order. When the next track is automatically played, I have a record of the previous track. To get round this I keep a record if the played track is from ENQUEUED or not and check for this in Playback Finished.

You also need to send a previous token so that Alexa can deal with a race condition. This is explained here:

<https://developer.amazon.com/en-US/docs/alexa/custom-skills/audioplayer-interface-reference.html#playlist-progression>

class PlaybackNearlyFinishedHandler(AbstractRequestHandler):

"""AudioPlayer.PlaybackNearlyFinished Directive received.

# respond to this request with a Play directive for the next stream and set ENQUEUE

# https://developer.amazon.com/en-US/docs/alexa/custom-skills/audioplayer-interface-reference.html#playbacknearlyfinished

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackNearlyFinished")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In PlaybackNearlyFinishedHandler")

persistence\_attr = handler\_input.attributes\_manager.persistent\_attributes

track\_number = int(persistence\_attr["track\_number"])

**next\_track = (track\_number + 1) % len(trackInfo.track\_info)**

**previous\_token = persistence\_attr["playback\_settings"]["token"]**

# this the previous token for the next track, i.e. at the moment it's the current one

# see playlist-progression

track\_number = next\_track # for consistency

audio\_key = trackInfo.track\_info[track\_number]["url"]

audio\_url = create\_presigned\_url(audio\_key)

persistence\_attr["playback\_settings"]["offset\_in\_milliseconds"] = 0

persistence\_attr["playback\_settings"]["token"] = audio\_key

persistence\_attr["playback\_settings"]["url"] = audio\_url

# if I update persistence\_attr["track\_number"] here, then start over (and resume) pick up wrong track (the next one)

# if not how does the next track know what to play after automatically playing next track from enqueue? So use this:

**persistence\_attr["playback\_settings"]["next\_stream\_enqueued"] = True**

# check this in playback finished. If true, then increment next track

handler\_input.attributes\_manager.persistent\_attributes = persistence\_attr

directive = PlayDirective(

**play\_behavior=PlayBehavior.ENQUEUE**,

audio\_item=AudioItem(

stream=Stream(

token=audio\_key,

url=audio\_url,

offset\_in\_milliseconds=0,

expected\_previous\_token=previous\_token),

metadata=None))

# but next track will have new url, but track\_number will be wrong

handler\_input.response\_builder.add\_directive(directive).set\_should\_end\_session(True)

return handler\_input.response\_builder.response

Now edit **PlaybackFinished**

class PlaybackFinishedHandler(AbstractRequestHandler):

"""AudioPlayer.PlaybackFinished Directive received.

Sent when the stream Alexa is playing comes to an end on its own.

Note: You can't send a new Play directive from here.

The response cannot include any standard properties outputSpeech, card, or reprompt.

Or any AudioPlayer directives. or directives from other interfaces, such a Dialog directive.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackFinished")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In PlaybackFinishedHandler")

# reset any attributes? e.g.

persistence\_attr = handler\_input.attributes\_manager.persistent\_attributes

**if persistence\_attr["playback\_settings"]["next\_stream\_enqueued"] == True:**

# track ended naturally, enqueued so stored track\_number is wrong.

**persistence\_attr["playback\_settings"]["next\_stream\_enqueued"] = False**

**track\_number = int(persistence\_attr["track\_number"])**

**next\_track = (track\_number + 1) % len(trackInfo.track\_info)**

**persistence\_attr["track\_number"] = next\_track**

handler\_input.attributes\_manager.persistent\_attributes = persistence\_attr

return handler\_input.response\_builder.response

And I think that’s it

There are lots of other things you can try:

1. Request a track
2. Go to a specific track.
3. Add a playlist
4. Request a genre of music or specific artist
5. Set loop on / off
6. Add touch controls for devices with a screen

Happy programming!

Video of it working is at: You tube video: <https://youtu.be/3wu2-m2KZT0>

All the files needed are here:

<https://github.com/jallwork/alexa-hosted-python-audio-skill/tree/main/part2>