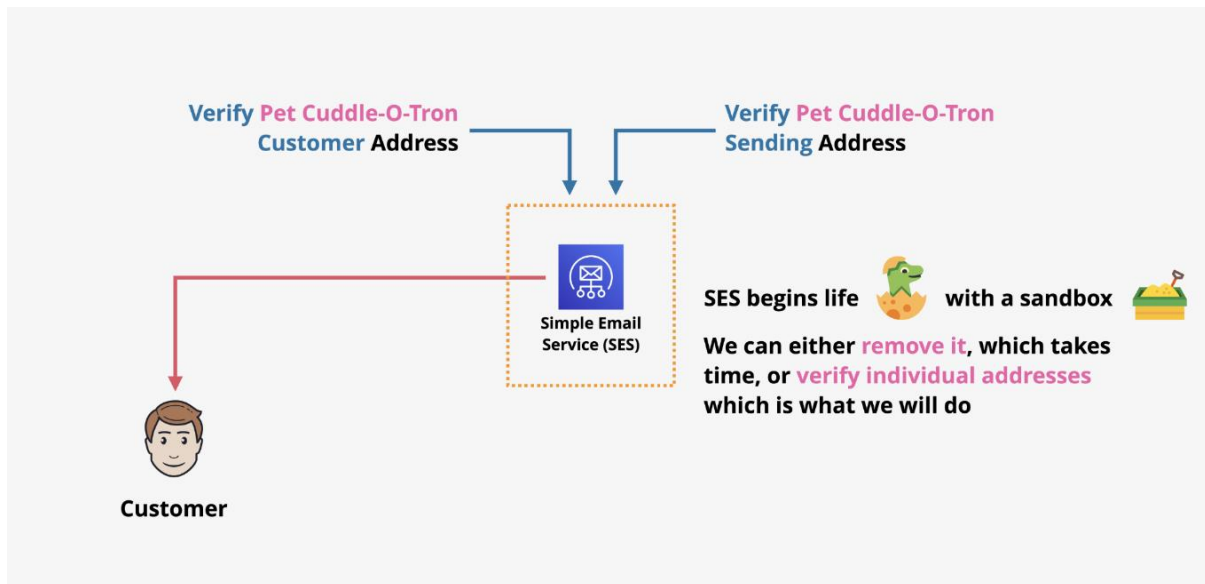


In this advanced demo you are going to implement a simple serverless application using S3, API Gateway, Lambda, Step Functions, SNS & SES.

The advanced demo consists of 6 stages :-

- STAGE 1 : Configure Simple Email service
- STAGE 2 : Add a email lambda function to use SES to send emails for the serverless application
- STAGE 3 : Implement and configure the state machine, the core of the application
- STAGE 4 : Implement the API Gateway, API and supporting lambda function
- STAGE 5 : Implement the static frontend application and test functionality
- STAGE 6 : Cleanup the account



STAGE 1A - VERIFY SES APPLICATION SENDING EMAIL ADDRESS

The Pet-Cuddle-O-Tron application is going to send reminder messages via SMS and Email. It will use the simple email service or SES. In production, it could be configured to allow sending from the application email, to any users of the application. SES starts off in sandbox mode, which means you can only sent to verified addresses (to avoid you spamming).

There is a whole [process to get SES out of sandbox mode](#), which you could do, but for this demo to keep things quick - we will verify the sender address and the receiver address.

Ensure you are logged into an AWS account, have admin privileges and are in the ap-south-1 / Mumbai Region

Move to the SES console <https://console.aws.amazon.com/ses/home?region=ap-south-1#>

Click on Verified Identities under Configuration (click on left 3 bars if you don't see this option)

Click Create Identity

Check the 'Email Address' checkbox

Ideally you will need a sending email address for the application and a receiving email address for your test customer. But you can use the same email for both.

For my application email ... the email the app will send from i'm going to

use rakeshtaninki4u@gmail.com

Enter whatever email you want to use to send in the box (it needs to be a valid address as it will be checked)

Click Create Identity

You will receive an email to this address containing a link to click

Click that link

You should see a Congratulations! message

Return to the SES console and Refresh your browser, the verification status should now be verified

Record this address somewhere save as the PetCuddleOTron Sending Address

STAGE 1B - VERIFY SES APPLICATION CUSTOMER EMAIL ADDRESS

If you want to use a different email address for the test customer (recommended), follow the steps below

Click Create Identity

Check the 'Email Address' checkbox For my application email ... the email for my test customer is rakeshtaninkil@gmail.com

Enter whatever email you want to use to send in the box (it needs to be a valid address as it will be checked)

Click Create Identity

You will receive an email to this address containing a link to click

Click that link

You should see a Congratulations! message

Return to the SES console and refresh your browser, the verification status should now be verified

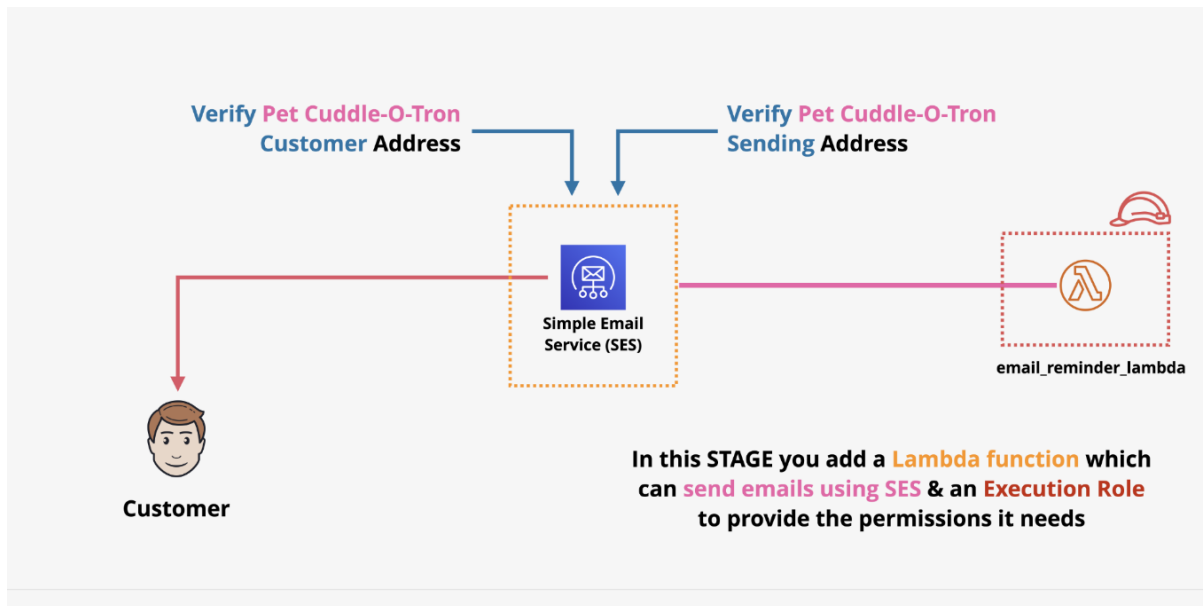
Record this address somewhere save as the PetCuddleOTron Customer Address

STAGE 1 - FINISH

At this point you have whitelisted 2 email addresses for use with SES.

- the PetCuddleOTron Sending Address.
- the PetCuddleOTron Customer Address.

These will be configured and used by the application in later stages. AT this point you have finished all the tasks needed in this STAGE of the Advanced Demo Lesson



STAGE 2A - CREATE THE Lambda Execution Role for Lambda

In this stage of the demo you need to create an IAM role which the email_reminder_lambda will use to interact with other AWS services.

You could create this manually, but its easier to do this step using cloudformation to speed things up.

Click <https://ap-south-1.console.aws.amazon.com/cloudformation/home?region=ap-south-1#/stacks/quickcreate?templateURL=https%3A%2F%2Ffrakeshtaninki-trainings.s3.ap-south-1.amazonaws.com%2Fflabs%2Faws-serverless-lab%2Flambdarolecfm.yaml&stackName=IAMRoleForLambda>

Check the I acknowledge that AWS CloudFormation might create IAM resources. box and then click Create Stack

Wait for the Stack to move into the CREATE_COMPLETE state before moving into the next

Move to the IAM Console <https://console.aws.amazon.com/iam/home?#/roles> and review the execution role which starts with *IAMRoleForLambda**

Notice that it provides SES, SNS and Logging permissions to whatever assumes this role.

This is what gives lambda the permissions to interact with those services

STAGE 2B - Create the email_reminder_lambda function

Next You're going to create the lambda function which will be used by the serverless application to create an email and then send it using SES

Move to the lambda

console <https://console.aws.amazon.com/lambda/home?region=ap-south-1#/functions>

Click on Create Function

Select Author from scratch

For Function name enter email_reminder_lambda

and for runtime click the dropdown and pick Python 3.9

Expand Change default execution role

Pick to Use an existing Role

Click the Existing Role dropdown and pick LambdaRole (there will be randomness and thats ok) and Click Create Function

STAGE 2C - Configure the email_reminder_lambda function

Scroll down, to Function code


in the lambda_function code box, select all the code and delete it
paste in this code

```
import boto3, os, json

FROM_EMAIL_ADDRESS = 'REPLACE_ME'

ses = boto3.client('ses')

def lambda_handler(event, context):
    # Print event data to logs ..
    print("Received event: " + json.dumps(event))
    # Publish message directly to email, provided by EmailOnly or EmailPar
    TASK
    ses.send_email( Source=FROM_EMAIL_ADDRESS,
                    Destination={ 'ToAddresses': [ event['Input']['email'] ] },
                    Message={ 'Subject': { 'Data': 'Whiskers Commands You to attend!' },
                              'Body': { 'Text': { 'Data': event['Input']['message'] } }
                    }
    )
    return 'Success!'
```



This function will send an email to an address it's supplied with (by step functions) and it will be FROM the email address we specify.

Select `REPLACE_ME` and replace with the PetCuddleOTron Sending Address which you noted down in `STAGE1`

Click `Deploy` to configure the lambda function

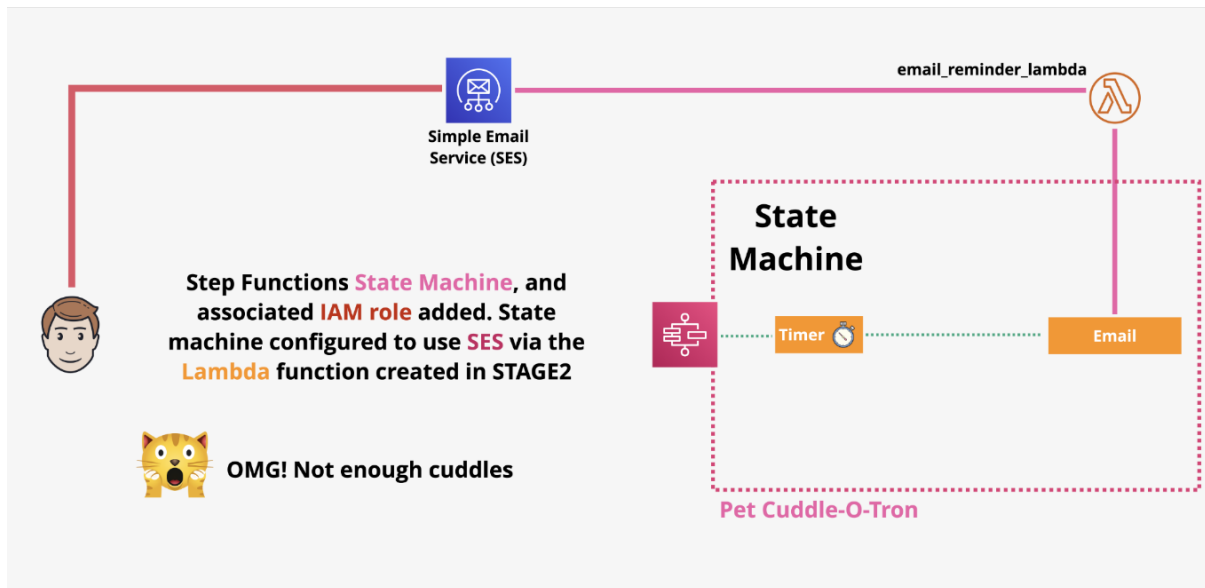
Scroll all the way to the top, and click the copy icon next to the lambda function ARN.

Note this ARN down somewhere same as the `email_reminder_lambda` ARN

STAGE 2 - FINISH

At this point you have configured the lambda function which will be used eventually to send emails on behalf of the serverless application.

You can go ahead and move onto stage 3 of the advanced demo.



STAGE 3A - CREATE STATE MACHINE ROLE

In this stage of the demo you need to create an IAM role which the state machine will use to interact with other AWS services.

You could create this manually, but its easier to do this step using cloudformation to speed things up.

Click <https://ap-south-1.console.aws.amazon.com/cloudformation/home?region=ap-south-1#/stacks/quickcreate?templateURL=https%3A%2F%2Ffrakeshtaninki-trainings.s3.ap-south-1.amazonaws.com%2Fflabs%2Faws-serverless-lab%2Fstatemachinerole.yaml&stackName=IAMRoleForStateMachine>

Check the I acknowledge that AWS CloudFormation might create IAM resources. box and then click Create Stack

Wait for the Stack to move into the CREATE_COMPLETE state before moving into the next

Move to the IAM Console <https://console.aws.amazon.com/iam/home?#/roles> and review the STATE MACHINE role which starts with "IAMRoleForStateMachine" and note how it gives

- logging permissions
- the ability to invoke the email lambda function when it needs to send emails
- the ability to use SNS to send text messages

STAGE 3B - CREATE STATE MACHINE

Move to the AWS Step Functions

Console <https://console.aws.amazon.com/states/home?region=ap-south-1#/homepage>

Click the Hamburger Menu at the top left and click State Machines

Click Create State Machine

Select Write your workflow in code which will allow you to use Amazon States Language

Scroll down for type select standard

Open this in a new tab <https://rakeshtaninki-trainings.s3.ap-south-1.amazonaws.com/labs/aws-serverless-lab/pet-cuddle-o-tron.json>

this is the Amazon States Language (ASL) file for the pet-cuddle-o-tron state machine

Copy the contents into your clipboard

Move back to the step functions console

Select all of the code snippet and delete it

Paste in your clipboard

Click the Refresh icon on the right side area ... next to the visual map of the state machine.

Look through the visual overview and the ASL .. and make sure you understand the flow through the state machine.

The state machine starts ... and then waits for a certain time period based on the `Timer` state. This is controlled by the web front end you will deploy soon. Then the `email` is used Which sends an email reminder

The state machine will control the flow through the serverless application.. once stated it will coordinate other AWS services as required.

STAGE 3C - CONFIGURE STATE MACHINE

In the state machine ASL (the code on the left) locate the `Email` definition.

Look for `EMAIL_LAMBDA_ARN` which is a placeholder, replace this with the `email_reminder_lambda` ARN you noted down in the previous step. This is the ARN of the lambda function you created.

Scroll down to the bottom and click next

For State machine name USE `PetCuddleOTron`

Scroll down and under Permissions select Choose an existing role and select `StateMachineRole` from the dropdown (it should be the only one, if you have multiple select the correct one and there will be random which is fine as this was created by CloudFormation)

Scroll down, under Logging, change the Log Level to All

Scroll down to the bottom and click Create state machine

Locate the ARN for the state machine on the top left... note this down somewhere safe

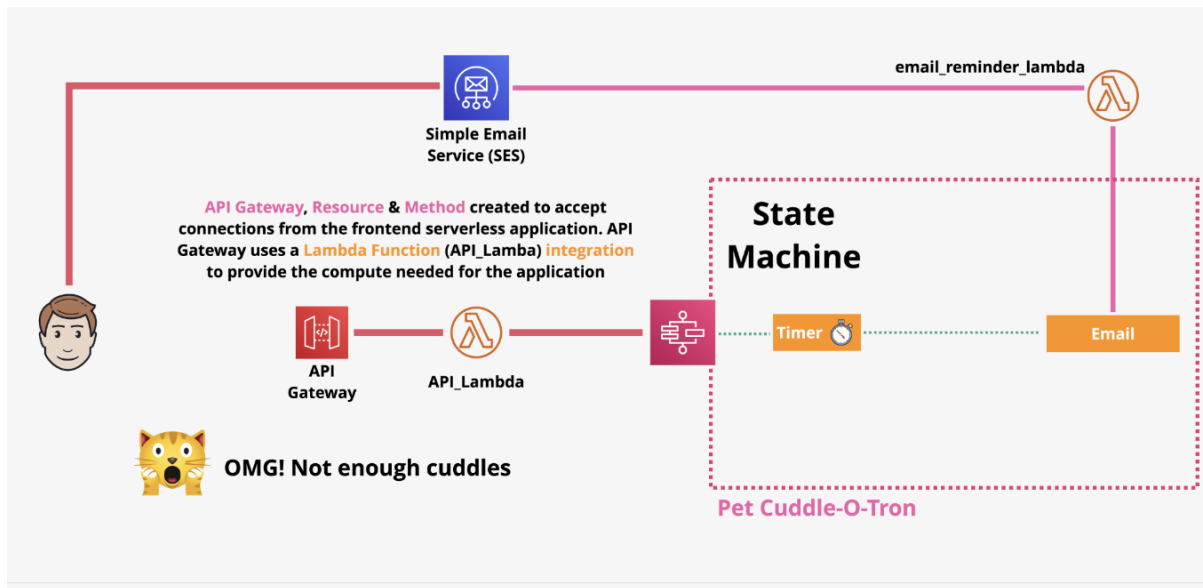
as State Machine ARN

STAGE 3 - FINISH

At this point you have configured the state machine which is the core part of the serverless application.

The state machine controls the flow through the application and is responsible for interacting with other AWS products and services.

In STAGE4 you will create the API Gateway and Lambda function which will act as the front end for the application.



STAGE 4A - CREATE API LAMBDA FUNCTION WHICH SUPPORTS APIGATEWAY

Move to the Lambda console <https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/functions>

Click On Create Function

for Function Name USE `api_lambda`

for Runtime USE Python 3.9

Expand Change default execution role

Select Use an existing role

Choose the `LambdaRole` from the dropdown

Click Create Function

This is the lambda function which will support the API Gateway

STAGE 4B - CONFIGURE THE LAMBDA FUNCTION (Using the current UI)

Scroll down, and remove all the code from the `lambda_function` text box

copy the below code into your clipboard

```
import boto3, json, os, decimal

SM_ARN = 'YOUR_STATEMACHINE_ARN'

sm = boto3.client('stepfunctions')

def lambda_handler(event, context):
    # Print event data to logs ..
    print("Received event: " + json.dumps(event))

    # Load data coming from APIGateway
    data = json.loads(event['body'])
    data['waitSeconds'] = int(data['waitSeconds'])

    # Sanity check that all of the parameters we need have come through from
    # API gateway
    # Mixture of optional and mandatory ones
    checks = []
    checks.append('waitSeconds' in data)
    checks.append(type(data['waitSeconds']) == int)
    checks.append('message' in data)

    # if any checks fail, return error to API Gateway to return to client
    if False in checks:
        response = {
            "statusCode": 400,
            "headers": {"Access-Control-Allow-Origin": "*"},
            "body": json.dumps( { "Status": "Success", "Reason": "Input failed
validation" }, cls=DecimalEncoder )
        }
        # If none, start the state machine execution and inform client of 2XX
        success :)
    else:
        sm.start_execution( stateMachineArn=SM_ARN, input=json.dumps(data,
cls=DecimalEncoder) )
        response = {
            "statusCode": 200,
            "headers": {"Access-Control-Allow-Origin": "*"},
            "body": json.dumps( {"Status": "Success"}, cls=DecimalEncoder )
        }
    return response

class DecimalEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, decimal.Decimal):
```

```
return int(obj)
return super(DecimalEncoder, self).default(obj)
```

Move back to the Lambda console.

Select the existing lambda code and delete it.

Paste the code into the lambda function.

This is the function which will provide compute to API Gateway.

Its job is to be called by API Gateway when its used by the serverless front end part of the application (loaded by S3)

It accepts some information from you, via API Gateway and then it starts a state machine execution - which is the logic of the application.

You need to locate the *YOUR_STATEMACHINE_ARN* placeholder and replace this with the State Machine ARN you noted down in the previous step.

Click Deploy to save the lambda function and configuration.

STAGE 4C - CREATE API

Now we have the `api_lambda` function created, the next step is to create the API Gateway, API and Method which the front end part of the serverless application will communicate with.

Move to the API Gateway

console <https://console.aws.amazon.com/apigateway/main/apis?region=ap-south-1>

Click APIs on the menu on the left

Locate the REST API box, and click Build (being careful not to click the build button for any of the other types of API ... REST API is the one you need) If you see a popup dialog Create your first API dismiss it by clicking ok

Under Create new API ensure New API is selected.

For API name* enter petcuddleotron

for Endpoint Type pick Regional

Click create API

STAGE 4D - CREATE RESOURCE

Click the Actions dropdown and Click Create Resource

Under resource name enter petcuddleotron

make sure that Configure as proxy resource is **NOT** ticked - this forwards everything as is, through to a lambda function, because we want some control, we **DONT** want this ticked.

Towards the bottom **MAKE SURE TO TICK** Enable API Gateway CORS.

This relaxes the restrictions on things calling on our API with a different DNS name, it allows the code loaded from the S3 bucket to call the API gateway endpoint.

if you DONT check this box, the API will fail

Click Create Resource

STAGE 4E - CREATE METHOD

Ensure you have the /petcuddleton resource selected, click Actions dropdown and click create method

In the small dropdown box which appears below /petcuddleton select POST and click the tick symbol next to it.

this method is what the front end part of the application will make calls to.

Its what the api_lambda will provide services for.

Ensure for Integration Type that Lambda Function is selected.

Tick for *"Use Lambda Proxy integration"*

Make sure ap-south-1 is selected for Lambda Region

In the Lambda Function box.. start typing api_lambda and it should autocomplete, click this auto complete (**Make sure you pick api_lambda and not email reminder lambda**)

Make sure that Use Default Timeout box **IS** ticked.

Make sure that Use Lambda Proxy integration box **IS** ticked, this makes sure that all of the information provided to this API is sent on to lambda for processing in the event data structure.

if you don't tick this box, the API will fail

Click Save

You may see a dialogue stating You are about to give API Gateway permission to invoke your Lambda function:. AWS is asking for your OK to adjust the resource policy on the lambda function to allow API Gateway to invoke it. This is a different policy to the execution role policy which controls the permissions lambda gets.

STAGE 4F - DEPLOY API

Now the API, Resource and Method are configured - you now need to deploy the API out to API gateway, specifically an API Gateway STAGE.

Click Actions Dropdown and Deploy API

For Deployment Stage select New Stage

for stage name and stage description enter prod

Click Deploy

At the top of the screen will be an Invoke URL .. note this down somewhere safe, you will need it in the next STAGE.

This URL will be used by the client side component of the serverless application and this will be unique to you.

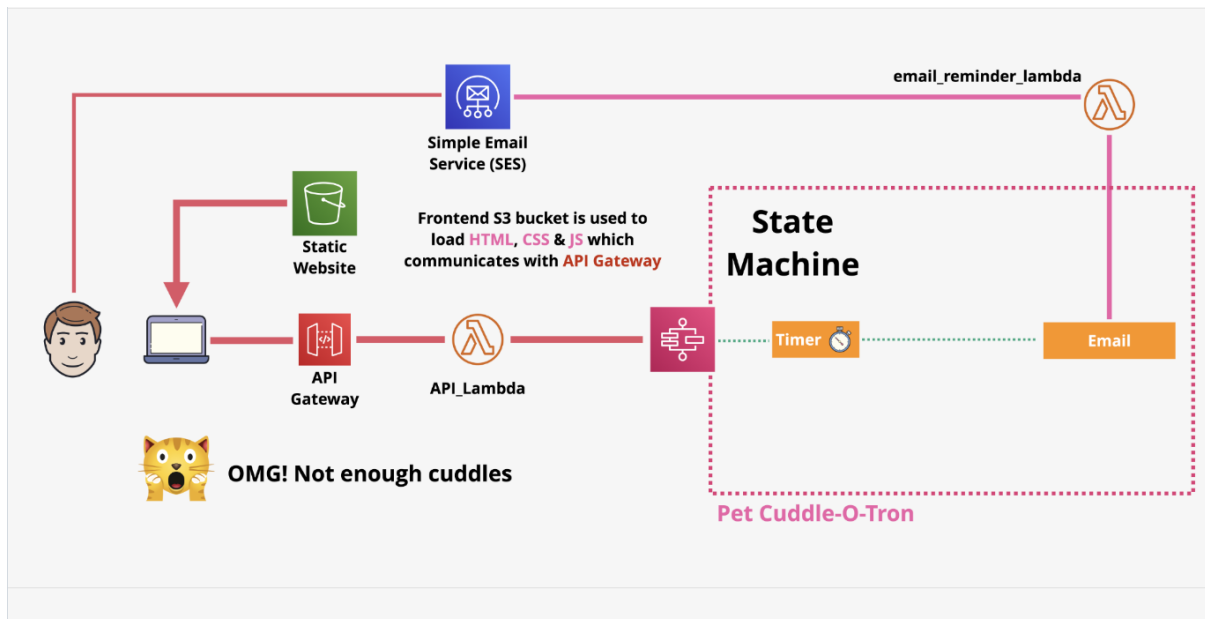
STAGE 4 - FINISH

At this point you have configured the last part of the AWS side of the serverless application.

You now have :-

- SES Configured
- An Email Lambda function to send email using SES
- A State Machine configured which can send EMAIL after a certain time period when invoked.
- An API, Resource & Method, which use a lambda function for backing deployed out to the PROD stage of API Gateway

In STAGE5 of this advanced demo you will configure the client side of the application (loaded from S3, running in a browser) so that it communicates to API Gateway.



STAGE 5A - CREATE THE S3 BUCKET

Move to the S3 Console <https://s3.console.aws.amazon.com/s3/home?region=ap-south-1>

Click Create bucket

Choose a unique bucket name Ensure the region is set to ap-south-1

Scroll Down and **UNTICK** Block all public access

Tick the box under Turning off block all public access might result in this bucket and the objects within becoming public to acknowledge you understand that you can make the bucket public.

Scroll Down to the bottom and click Create bucket

STAGE 5B - SET THE BUCKET AS PUBLIC

Go into the bucket you just created.

Click the Permissions tab.

Scroll down and in the Bucket Policy area, click Edit.

in the box, paste the code below

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicRead",
      "Effect": "Allow",
      "Principal": "*",
      "Action": ["s3:GetObject"],
      "Resource": ["REPLACEME_PET_CUDDLE_O_TRON_BUCKET_ARN/*"]
    }
  ]
}
```

```
} ]
```

Replace the `REPLACEME_PET_CUDDLE_O_TRON_BUCKET_ARN` (being careful NOT to include the `/*`) with the bucket ARN, which you can see near to `Bucket ARN`. Click `Save Changes`

STAGE 5C - ENABLE STATIC HOSTING

Next you need to enable static hosting on the S3 bucket so that it can be used as a front end website.

Click on the `Properties` Tab

Scroll down and locate `Static website hosting`

Click `Edit`

Select `Enable` Select `Host a static website`

For both `Index Document` and `Error Document` enter `index.html`. Click `Save Changes`

Scroll down and locate `Static website hosting` again.

Under `Bucket Website Endpoint` copy and note down the bucket endpoint URL.

STAGE 5D - DOWNLOAD AND EDIT THE FRONT END FILES

Download and extra this ZIP file https://rakeshtaninki-trainings.s3.ap-south-1.amazonaws.com/labs/aws-serverless-lab/serverless_frontend.zip

Inside the `serverless_frontend` folder are the front end files for the serverless website :-

- `index.html` .. the main index page
- `main.css` .. the stylesheet for the page
- `whiskers.png` .. an image of whiskers !!
- `serverless.js` .. the JS code which runs in your browser. It responds when buttons are clicked, and passes and text from the boxes when it calls the API Gateway endpoint.

Open the `serverless.js` in a code/text editor.

Locate the placeholder `REPLACEME_API_GATEWAY_INVOKE_URL`. replace it with your API Gateway Invoke URL at the end of this URL.. add `/petcuddleotron` it should look something like this `https://somethingsomething.execute-api.ap-south-1.amazonaws.com/prod/petcuddleotron` Save the file.

STAGE 5E - UPLOAD AND TEST

Return to the S3 console Click on the objects Tab.

Click upload

Drag the 4 files from the serverless_frontend folder onto this tab, including the serverless.js file you just edited. **MAKE SURE ITS THE EDITED VERSION**

Click upload and wait for it to complete.

Click Exit

Verify All 4 files are in the objects area of the bucket.

Open the PetCuddle0Tron URL you just noted down in a new tab.

What you are seeing is a simple HTML web page created by the HTML file itself and the main.css stylesheet.

When you click buttons .. that calls the .js file which is the starting point for the serverless application

Ok to test the application

Enter an amount of time until the next cuddle ...I suggest 120 seconds Enter a message, i suggest HUMAN COME HOME NOW

then enter the PetCuddle0Tron Customer Address in the email box, this is the email which you verified right at the start as the customer for this application.

before you do the next step and click the button on the application, if you want to see how the application works do the following open a new tab to the step functions console <https://console.aws.amazon.com/states/home?region=ap-south-1#/statemachines>

Click on PetCuddle0Tron

Click on the Logging tab, you will see no logs Click on the Executions tab, you will see no executions..

Move back to the web application tab (s3 bucket)

then click on Email Minion Button to send an email.

Go back to the Step functions console make sure the Executions Tab is selected click the Refresh Icon Click the execution

Watch the graphic .. see how the Timer state is highlighted The step function is now executing and it has its own state ... its a serverless flow. Keep waiting, and after 120 seconds the visual will update showing the flow through the state machine

- Timer .. waits 120 seconds
- Email invokes the lambda function to send an email
- NextState in then moved through, then finally END

Scroll to the top, click ExecutionInput and you can see the information entered on the webpage. This was send it, via the JS running in browser, to the API gateway, to the api_lambda then through to the statemachine

Click PetCuddle0Tron at the top of the page

Click on the Logging Tab

Because the roles you created had CWLogs permissions the state machine is able to log to CWLogs Review the logs and ensure you are happy with the flow.

STAGE 5 - FINISH

At this point that's everything .. you now have a fully functional serverless application

- Loads HTML & JS From S3 & Static hosting
- Communicates via JS to API Gateway
- uses `api_lambda` as backing resource
- runs a statemachine passing in parameters
- state machine sends email
- state machine terminates

No servers were harmed, or used even, in this production :)

That's everything for this advanced demo, in STAGE6 you will clear up all of the services used for this advanced demo.

Move to the S3 console <https://s3.console.aws.amazon.com/s3/home?region=ap-south-1>

Select the bucket you created

Click Empty, type or copy/paste the bucket name and click Empty, Click Exit

Click Delete, type or copy/paste the bucket name and click Delete, Click Exit

Move to the API Gateway console [https://us-east-](https://us-east-1.console.aws.amazon.com/apigateway/main/apis?region=ap-south-1)

[1.console.aws.amazon.com/apigateway/main/apis?region=ap-south-1](https://us-east-1.console.aws.amazon.com/apigateway/main/apis?region=ap-south-1)

Check the box next to the petcuddleotron API

Click Actions and then Delete

Click Delete

Move to the lambda console [https://ap-south-](https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/functions)

[1.console.aws.amazon.com/lambda/home?region=ap-south-1#/functions](https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/functions)

Check the box next to `email_reminder_lambda`, click Actions, Click Delete, Click Delete

Check the box next to `api_lambda`, click Actions, Click Delete, Click Delete

Move to the Step Functions console [https://ap-south-](https://ap-south-1.console.aws.amazon.com/states/home?region=ap-south-1#/statemachines)

[1.console.aws.amazon.com/states/home?region=ap-south-1#/statemachines](https://ap-south-1.console.aws.amazon.com/states/home?region=ap-south-1#/statemachines)

Check the box next to PetCuddle0Tron, Click Delete, then Delete state machine

optional, it might save time if you want to use SES in the future, and it doesn't cost anything to keep these active Go to the SES console and verified

identities <https://ap-south-1.console.aws.amazon.com/ses/home?region=ap-south-1#/verified-identities>

Select one of the identities, Click Delete, then click Confirm

Pick the other verified identity, Click Delete, then click Confirm

Move to the cloudformation console <https://ap-south-1.console.aws.amazon.com/cloudformation/home?region=ap-south-1#/stacks?filteringText=&filteringStatus=active&viewNested=true>

Check the box next to SMROLE , click Delete then Delete Stack

Check the box next to LAMBDA_ROLE , click Delete then Delete Stack

AT this point you have removed all infrastructure used for this AdvancedDemo and have completed the advanced demo itself.

Good job !!