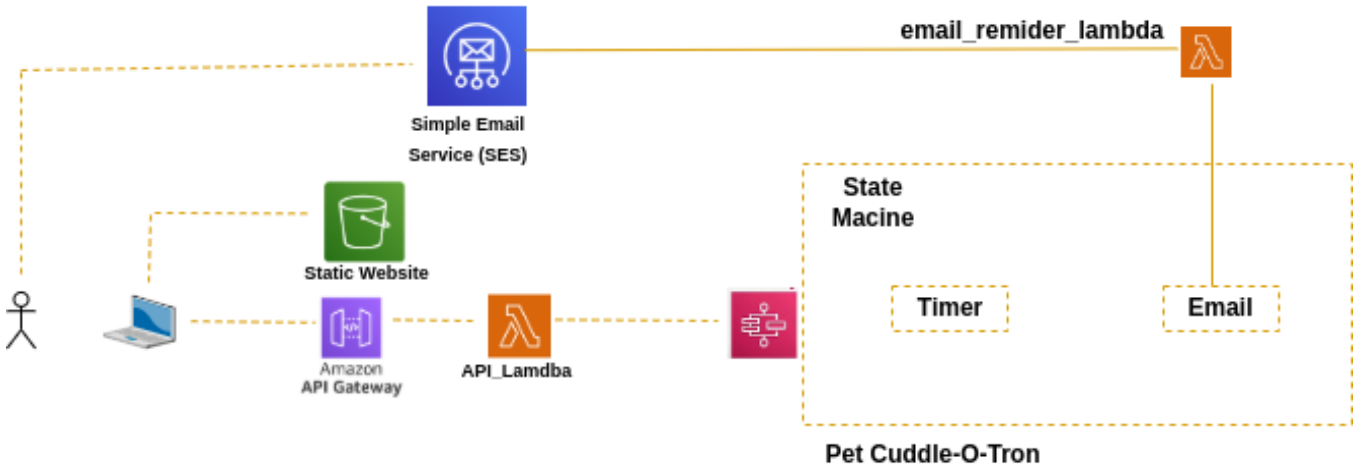
"Pet Cuddle-O-Tron" allows users to receive email reminders about events related to their pets using AWS services:



- ## STAGE 1 : Configure Simple Email service:

Stage 1 - Verifying Email Addresses for SES

The **Pet-Cuddle-O-Tron** application will send reminder messages via **SMS and Email** using **AWS Simple Email Service (SES)**. In a production environment, SES can be configured to send emails to any user of the application.

By default, SES operates in **sandbox mode**, meaning it can only send emails to **verified addresses** to prevent spam.

For this demo, instead of removing SES from sandbox mode, we will **verify the sender and receiver email addresses** to keep things simple.

Stage 1A - Verify the Application's Sending Email Address

Steps to verify the sender email address:

1. Ensure you are logged into an AWS account with **admin privileges** and are in the **us-east-1 (N. Virginia) region**.
2. Open the **SES Console**:
   👉 [AWS SES Console]
3. Click on **Verified Identities** under **Configuration**.
4. Click **Create Identity**.
5. Select the **Email Address** option.
6. Enter an email address that the application will use to send emails.
   - Example: `adrian+cuddleotron@cantrill.io`

    - You can use any valid email address, but it must be one you can access.
7. Click **Create Identity**.
8. Check your inbox for a **verification email** and click the confirmation link inside.
9. You should see a **Congratulations!** message.
10. Return to the **SES Console** and refresh the page. The email address should now be marked as **Verified**.
11. **Save this email address**, as it will be used later as the **PetCuddleOTron Sending Address**.

Stage 1B - Verify the Customer Email Address

If you want to use a different email address for testing, follow these steps:

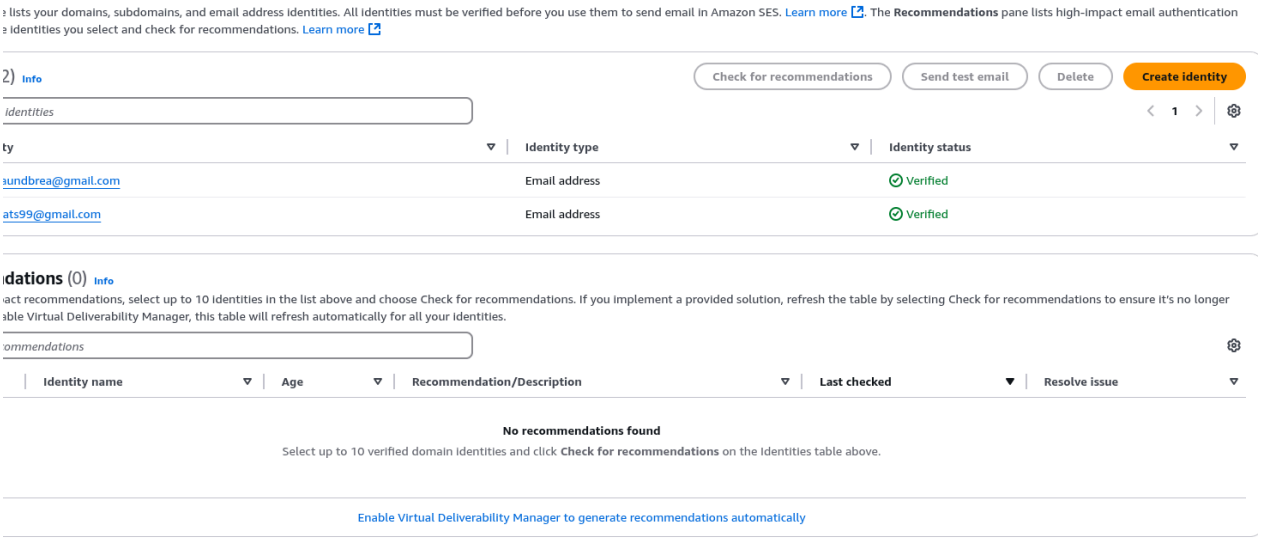
1. Go back to the **SES Console**.
2. Click **Create Identity**.
3. Select the **Email Address** option.
4. Enter an email for the test customer (for example, `adrian+cuddlecustomer@cantrill.io`).
5. Click **Create Identity**.
6. Check your inbox for a **verification email** and click the confirmation link inside.
7. You should see a **Congratulations!** message.
8. Return to the **SES Console** and refresh the page. The email should now be marked as **Verified**.
9. **Save this email address** as the **PetCuddleOTron Customer Address**.

Stage 1 - Completion

At this point, you have successfully verified **two email addresses** for use with SES:

✅ The **PetCuddleOTron Sending Address** (used by the application to send emails).
✅ The **PetCuddleOTron Customer Address** (used to receive test emails).

These email addresses will be configured and used in later stages of the application setup.
You have now completed all the tasks for Stage 1 of this Advanced Demo Lesson! 🎉



- ## STAGE 2 : Add a email lambda function to use SES to send emails for the serverless application

In this stage, we will **create an IAM role** and a **Lambda function** that will handle sending emails using **AWS SES**.

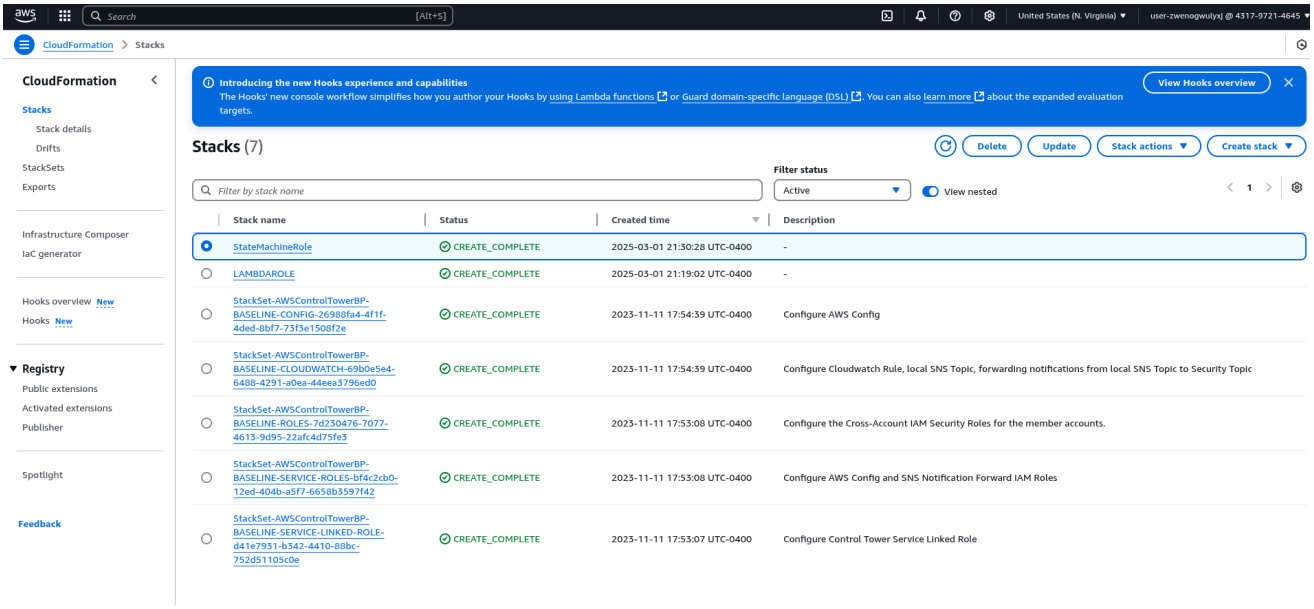
Stage 2A - Create the IAM Role for Lambda

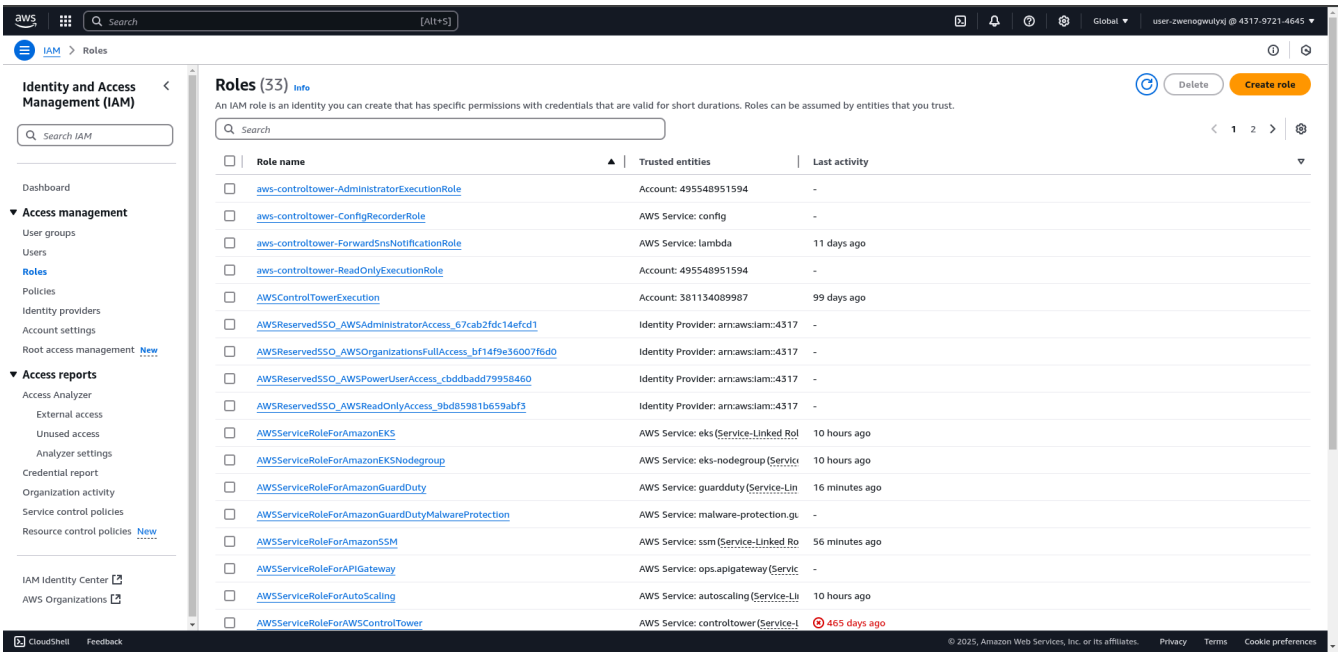
The `email_reminder_lambda` function needs permissions to interact with other AWS services like **SES, SNS, and CloudWatch Logs**. Instead of manually creating an IAM role, we will use **AWS CloudFormation** to automate the process.

Steps to create the IAM role:

1. Click this link to open **CloudFormation**:
   👉 [CloudFormation Quick Create]
2. Check the box **"I acknowledge that AWS CloudFormation might create IAM resources."**
3. Click **Create Stack**.
4. Wait until the stack reaches the **CREATE_COMPLETE** state.

5. Open the **IAM Console**:
   👉 [IAM Roles]
6. Find the new IAM role and review its permissions.
   - It allows access to **SES, SNS, and logging** services.
   - This role will be used by the Lambda function to interact with AWS services.





Stage 2B - Create the Lambda Function

Next, we will create the **email_reminder_lambda** function. This function will generate and send emails through **AWS SES** when triggered.

Steps to create the Lambda function:

1. Open the **AWS Lambda Console**:
   👉 [Lambda Console]
2. Click **Create Function**.
3. Choose **Author from scratch**.
4. Enter the function name: `email_reminder_lambda`.
5. Under **Runtime**, select **Python 3.9**.
6. Expand **Change default execution role**.
7. Select **Use an existing role**.
8. Choose the IAM role you created earlier (`LambdaRole`).
9. Click **Create Function**.

Stage 2C - Configure the Lambda Function

Now, we need to add code to the Lambda function so it can send emails.

Steps to configure the function:

1. Scroll down to **Function Code**.
2. Delete the existing code and replace it with this:

python
CopyEdit

```
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))

    # Send an email using SES
    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!'
```

3. Replace **'REPLACE_ME'** with the **PetCuddleOTron Sending Address** you verified in **Stage 1**.
4. Click **Deploy** to save the function.
5. Scroll to the top and copy the **Lambda function ARN** (Amazon Resource Name).
6. Save this ARN for later use as the **email_reminder_lambda ARN**.

Stage 2 - Completion

At this point, you have successfully:

✅ Created an **IAM Role** for the Lambda function with SES, SNS, and logging permissions.
✅ Created the **email_reminder_lambda** function.
✅ Configured the function to send emails using **AWS SES**.

You are now ready to proceed to **Stage 3** of the demo! 🚀

- ## STAGE 3 : Implement and configure the state machine, the core of the application

In this stage, we will **create an IAM role** for the state machine and then configure the **AWS Step Functions state machine** to manage the workflow of our serverless application.

Stage 3A - Create the IAM Role for the State Machine

The state machine needs permissions to interact with **AWS services** such as **CloudWatch Logs, Lambda (for sending emails), and SNS (for sending text messages)**. Instead of creating this role manually, we will use **AWS CloudFormation** to automate the process.

Steps to create the IAM role:

1. Click this link to open **CloudFormation**:
   👉 CloudFormation Quick Create
2. Check the box **"I acknowledge that AWS CloudFormation might create IAM resources."**
3. Click **Create Stack**.
4. Wait for the stack to reach the **CREATE_COMPLETE** state.
5. Open the **IAM Console**:
   👉 IAM Roles
6. Find the newly created **State Machine Role** and review its permissions:
   - It allows **logging** to CloudWatch.
   - It has permission to **invoke the email Lambda function**.
   - It can **send SMS messages via SNS**.

Stage 3B - Create the State Machine

Now, we will create the **state machine**, which will define the workflow for the serverless application.

Steps to create the state machine:

1. Open the **AWS Step Functions Console**:
   👉 Step Functions Console
2. Click the **menu (☰) in the top left** and select **State Machines**.
3. Click **Create State Machine**.
4. Select **"Write your workflow in code"** to use **Amazon States Language (ASL)**.
5. Scroll down to **Type** and select **Standard**.
6. Open this link in a new tab:
   👉 Pet Cuddle-O-Tron State Machine JSON

7. Copy all the JSON content to your clipboard.
8. Go back to the **Step Functions Console**.
9. Delete the default code snippet and **paste the copied JSON**.
10. Click the **Refresh icon** next to the visual map to update the state machine diagram.

Understanding the State Machine Flow

- The state machine starts and waits for a **specific time period** (defined by the Timer state).
- After the wait time, it triggers **an email reminder** using the **Lambda function**.
- The state machine coordinates all steps in the serverless application workflow.

Stage 3C - Configure the State Machine

Now, we need to modify the state machine to use the **email Lambda function** we created earlier.

Steps to configure the state machine:

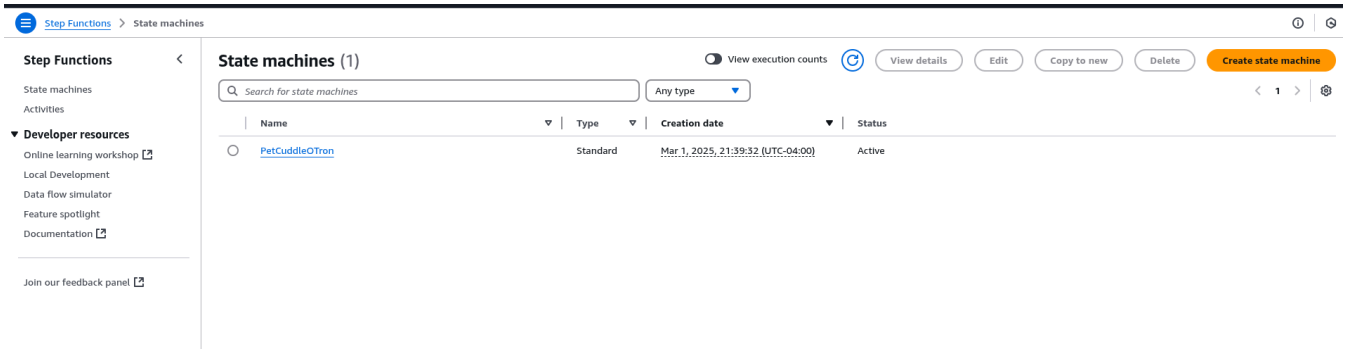
1. In the **ASL code (left panel of Step Functions Console)**, locate the `EmailOnly` definition.
2. Find the placeholder `EMAIL_LAMBDA_ARN`.
3. Replace `EMAIL_LAMBDA_ARN` with the **ARN of your `email_reminder_lambda` function** (which you saved in Stage 2).
4. Scroll down and click **Next**.
5. Enter **PetCuddleOTron** as the **State Machine Name**.
6. Under **Permissions**, choose **"Use an existing role"** and select **StateMachineRole** from the dropdown.
7. Under **Logging**, change **Log Level** to **All**.
8. Scroll to the bottom and click **Create State Machine**.
9. Once created, copy the **ARN of the state machine** (found in the top-left corner).
10. Save this ARN as your **State Machine ARN** for later use.

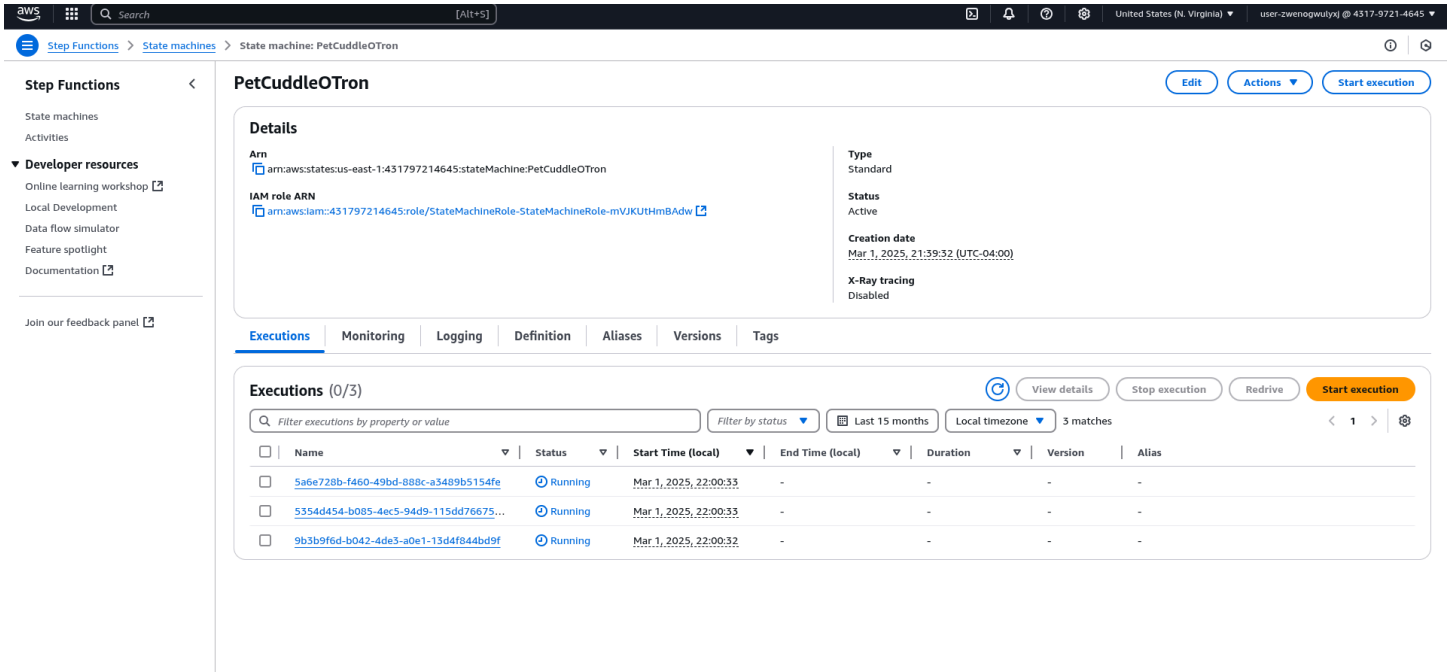
Stage 3 - Completion

At this point, you have successfully:

✅ Created an **IAM Role** for the state machine with permissions for logging, Lambda, and SNS.
✅ Created and configured the **State Machine** that controls the workflow of the serverless application.

Next, in **Stage 4**, you will set up **API Gateway** and another **Lambda function** to serve as the front end for the application. 🚀





- ## STAGE 4 : Implement the API Gateway, API and supporting lambda function

In this stage, you will set up the **API layer** for the serverless application. This API acts as the bridge between the **browser-based front end** (hosted on S3) and the **backend services** running on AWS.

The API is powered by **API Gateway**, which will handle incoming requests and pass them to a **Lambda function** (`api_lambda`) for processing. This Lambda function will then **trigger the Step Functions state machine**, which controls the flow of the application.

Stage 4A - Create the API Lambda Function

First, we need to create the Lambda function that will **handle API Gateway requests**.

Steps to create the Lambda function:

1. Open the **Lambda Console**:
   👉 AWS Lambda
2. Click **Create Function**.
3. Set the following details:
   - **Function Name**: `api_lambda`
   - **Runtime**: `Python 3.9`
4. Expand **Change default execution role**.
5. Select **Use an existing role**.
6. Choose **LambdaRole** from the dropdown.
7. Click **Create Function**.

This function will handle requests from API Gateway and start the **Step Functions workflow** when needed.

Stage 4B - Configure the Lambda Function

Now, we will **add the code** to the `api_lambda` function.

Steps to configure the Lambda function:

1. Scroll down to the **Function code** section.
2. Delete all existing code from the `lambda_function` text box.
3. Open this link in a new tab:
   👉 [API Lambda Function Code]
4. Copy all the code from the file.
5. Return to the **Lambda console** and **paste the copied code** into the function.
6. Locate the placeholder **`YOUR_STATEMACHINE_ARN`** in the code.
7. Replace **`YOUR_STATEMACHINE_ARN`** with the **State Machine ARN** that you noted down in **Stage 3**.
8. Click **Deploy** to save the function.

Now, this Lambda function will process API Gateway requests and start the **state machine execution**.

Stage 4C - Create the API Gateway

With the `api_lambda` function ready, we can now create the **API Gateway** that will serve as the **entry point** for our application.

Steps to create the API Gateway:

1. Open the **API Gateway Console**:
   👉 [API Gateway]
2. Click **APIs** on the left menu.
3. Locate the **REST API** box and click **Build**.
   - Be careful **not to select the other API types (HTTP, WebSocket, etc.)**.
4. If a **popup appears**, click **OK** to dismiss it.
5. Under **Create new API**, ensure **New API** is selected.
6. Set the following details:
   - **API Name**: `petcuddleotron`
   - **Endpoint Type**: `Regional`
7. Click **Create API**.

Stage 4D - Create an API Resource

Now, we need to create an **API resource** that represents an endpoint within API Gateway.

Steps to create the resource:

1. Click the **Actions** dropdown and select **Create Resource**.
2. Set the following details:
   - **Resource Name**: `petcuddleotron`
   - **Do NOT** check the **Configure as proxy resource** box.
   - **Check** the box for **Enable API Gateway CORS**.
3. Click **Create Resource**.

CORS (Cross-Origin Resource Sharing) is necessary because the front end (hosted on S3) will be calling this API. Without CORS, the browser will block the request.

Stage 4E - Create an API Method

Next, we will create a **POST method** for the `/petcuddleotron` resource. This method will be responsible for **handling API requests** from the front end.

Steps to create the method:

1. Make sure `/petcuddleotron` is selected in the left panel.
2. Click the **Actions** dropdown and select **Create Method**.
3. In the small dropdown below `/petcuddleotron`, select **POST**, then click the ✔️ **(tick mark)**.
4. Set the following details:
   - **Integration Type**: `Lambda Function`
   - **Lambda Region**: `us-east-1`
   - **Lambda Function**: Start typing `api_lambda`, then select it from the autocomplete list.
   - **Ensure** you select `api_lambda` and **not** `email_reminder_lambda`.
5. Make sure the following options are **enabled**:
   - ✅ **Use Default Timeout**
   - ✅ **Use Lambda Proxy Integration** (This ensures that all API data is forwarded correctly to Lambda).
6. Click **Save**.
7. You may see a **permission confirmation dialog**. Click **OK** to allow API Gateway to invoke the Lambda function.

Now, API Gateway has a **POST method** that connects to the `api_lambda` function.

Stage 4F - Deploy the API

Now that the API is configured, we need to **deploy** it so that it becomes accessible via an **Invoke URL**.

Steps to deploy the API:

1. Click the **Actions** dropdown and select **Deploy API**.
2. Set the following details:
   - **Deployment Stage**: `New Stage`
   - **Stage Name**: `prod`
   - **Stage Description**: `prod`
3. Click **Deploy**.
4. At the top of the page, you will see an **Invoke URL**.
   - **Copy and save this URL**, as it will be needed in the next stage.

Stage 4 - Completion

At this point, you have successfully:

✅ Created the **`api_lambda` function**, which will start the state machine.
✅ Configured the **Lambda function** to handle API Gateway requests.
✅ Set up **API Gateway** with a **POST method** that routes requests to Lambda.
✅ **Deployed the API** to the `prod` stage.

Now, you have a fully functional **backend API** that the front end can communicate with.

In **Stage 5**, you will configure the **client-side application** (hosted on S3) to interact with the API Gateway. 🚀

Lambda > Functions > api_lambda

Successfully updated the function **api_lambda**.

api_lambda

Throttle | Copy ARN | Actions

Function overview Info

Export to Infrastructure Composer | Download

Diagram | Template

api_lambda

Layers (0)

+ Add trigger

+ Add destination

Description

-

Last modified

Copied

Function ARN

arn:aws:lambda:us-east-1:431797214645:function:api_lambda

Function URL Info

-

Code | Test | Monitor | Configuration | Aliases | Versions

Code source Info

Upload from

EXPLORER

API_LAMBDA

lambda_function.py

```
# This code is a bit ...messy and includes some workarounds
# It functions fine, but needs some cleanup
# Checked the DecimalEncoder and Checks workarounds 20200402 and no progression towards fix

import boto3, json, os, decimal
```

CloudShell | Feedback | © 2025, Amazon Web Services, Inc. or its affiliates. | Privacy | Terms | Cookie preferences

---

aws | Search [Alt+S] | Copied | United States (N. Virginia) | user-zwenogwulyxj @ 4317-9721-4645

Lambda > Functions > api_lambda

Successfully updated the function **api_lambda**.

Code | Test | Monitor | Configuration | Aliases | Versions

Code source Info

Upload from

EXPLORER

API_LAMBDA

lambda_function.py

DEPLOY

Deploy (Ctrl+Shift+U)

Test (Ctrl+Shift+I)

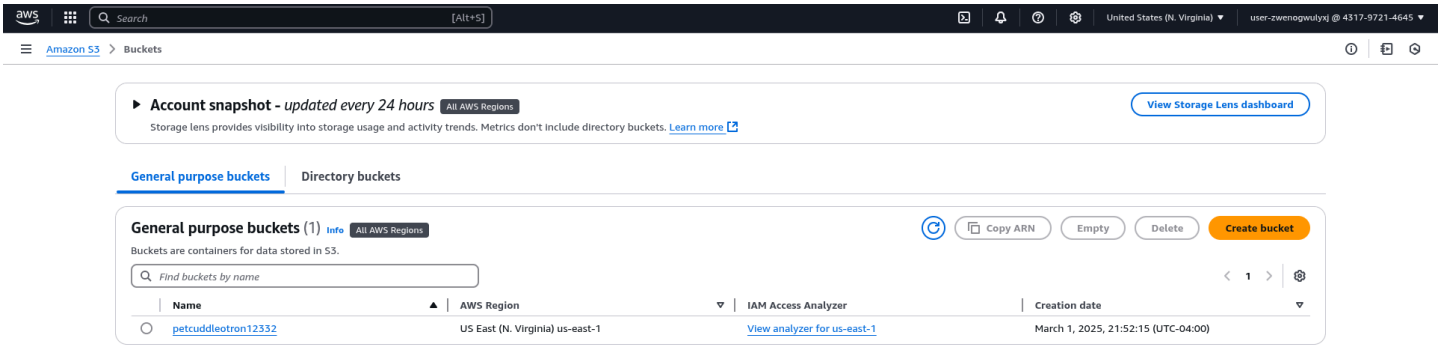
TEST EVENTS [NONE SELECTED]

Create new test event

```
# Checked the DecimalEncoder and Checks workarounds 20200402 and no progression towards fix

import boto3, json, os, decimal

SM_ARN = 'arn:aws:states:us-east-1:431797214645:stateMachine:PetCuddleOTron'

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:
```

Amazon Q Tip 1/3: Start typing to get sugg

CloudShell | Feedback | © 2025, Amazon Web Services, Inc. or its affiliates. | Privacy | Terms | Cookie preferences



- ## STAGE 5 : Implement the static frontend application and test functionality

In this stage, you will set up the **front-end** for the serverless application. The front end is a simple **static website** hosted on **Amazon S3**, which communicates with the API Gateway using JavaScript.

Stage 5A - Create an S3 Bucket

To host the front-end files, you need to create an **S3 bucket**.

Steps to create the bucket:

1. Open the **S3 Console**:
   👉 [AWS S3]
2. Click **Create Bucket**.
3. Choose a **unique bucket name** (bucket names must be globally unique).
4. Make sure the **Region** is set to **US East (N. Virginia) (us-east-1)**.
5. Scroll down and **UNTICK** the box for **Block all public access**.
6. Tick the checkbox confirming that you understand the bucket could become public.
7. Scroll down and click **Create Bucket**.

This bucket will store the static files for the front-end application.

Stage 5B - Make the Bucket Public

Since this is a public website, you need to configure the bucket so that **anyone can access** the front-end files.

Steps to make the bucket public:

1. Open the **S3 Console** and go to the bucket you just created.
2. Click on the **Permissions** tab.
3. Scroll down to the **Bucket Policy** section and click **Edit**.

Copy and paste the following policy into the box:
json
CopyEdit

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

4.
5. Replace `REPLACEME_PET_CUDDLE_O_TRON_BUCKET_ARN` with your **actual bucket ARN** (you can find it in the bucket details).
6. Click **Save Changes**.

Now, anyone can access the files in this bucket.

Stage 5C - Enable Static Website Hosting

To turn this S3 bucket into a **website**, you need to enable **static hosting**.

Steps to enable website hosting:

1. Go to the **Properties** tab of your bucket.
2. Scroll down to **Static website hosting**.
3. Click **Edit**.

4. Select **Enable**.
5. Choose **Host a static website**.

For both **Index Document** and **Error Document**, enter:
diff
CopyEdit
`index.html`

6.
7. Click **Save Changes**.
8. Scroll down again and **copy** the **Bucket Website Endpoint URL**.

This URL is where your front-end website will be accessible.

Stage 5D - Download and Edit the Front-End Files

Now, you need to download and modify the **front-end files** so that they can connect to your API Gateway.

Steps to download and edit the front-end files:

1. Download the front-end files from this link:
   👉 Front-End ZIP
2. Extract the ZIP file. Inside, you will see these files:
   - `index.html` – The main web page.
   - `main.css` – Stylesheet for the page.
   - `whiskers.png` – An image used in the app.
   - `serverless.js` – JavaScript file that handles API requests.
3. Open `serverless.js` in a **code editor** (e.g., VS Code, Notepad++, or any text editor).
4. Find the placeholder **`REPLACEME_API_GATEWAY_INVOKE_URL`**.
5. Replace it with your **API Gateway Invoke URL** (from Stage 4F).
6. At the end of the URL, add `/petcuddleotron`.

Example:
bash
CopyEdit
`https://your-api-id.execute-api.us-east-1.amazonaws.com/prod/petcuddleotron`

   -
7. Save the file.

This ensures that when users click buttons on the front end, the requests are sent to the correct API Gateway endpoint.

Stage 5E - Upload and Test the Front-End

Now, you will upload the **edited files** to your S3 bucket and test the application.

Steps to upload the files:

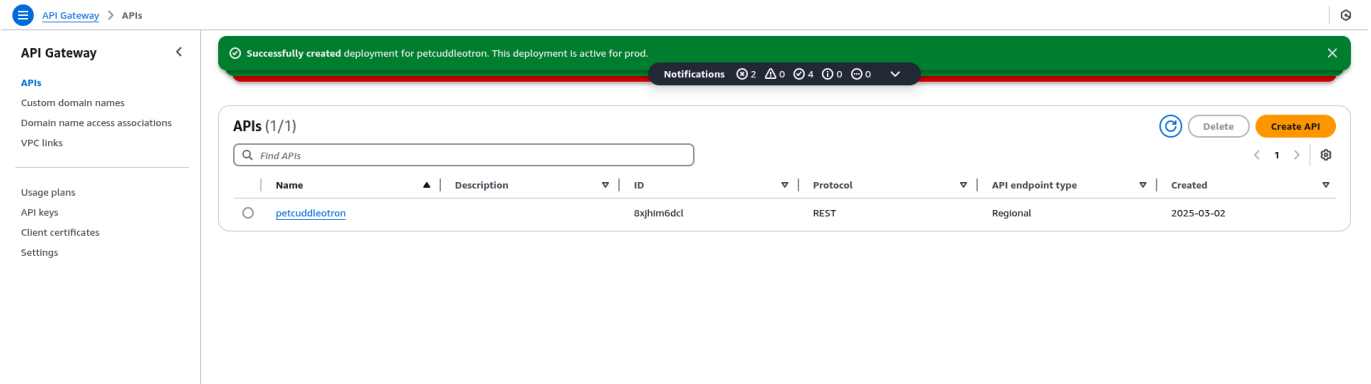
1. Go to your **S3 bucket**.
2. Click on the **Objects** tab.
3. Click **Upload**.
4. Drag and drop all **four files** (`index.html`, `main.css`, `whiskers.png`, and the modified `serverless.js`) into the upload area.
5. Click **Upload** and wait for it to complete.
6. Click **Exit** and verify that all four files are in the bucket.

Steps to test the application:

1. Open the **Bucket Website Endpoint URL** (copied in Stage 5C) in a web browser.
2. You should see a **simple web page** with buttons and input fields.
3. Enter a **time delay** (e.g., 120 seconds).
4. Enter a **message** (e.g., `"HUMAN, COME HOME NOW!"`).
5. Enter the **email address** that you verified earlier in **SES**.
6. Before clicking the button, open the **Step Functions Console** in a new tab:
   👉 [Step Functions Console]
7. Click on **PetCuddleOTron**.
8. Click on the **Executions** tab (you should see no executions yet).
9. Go back to the **web application** and click the **"Email Minion"** button.
10. Switch back to the **Step Functions Console** and click **Refresh**.
11. Click on the new execution and watch the flow:
    - **Timer state** will activate (waits for the delay).
    - **Email state** will trigger the **email Lambda function**.
    - The state machine will **end** after processing.

Check the logs:

1. Click on **Execution Input** in Step Functions to see the data sent from the web app.
2. Click on **Logging** to review CloudWatch logs.



Stage 5 - Completion

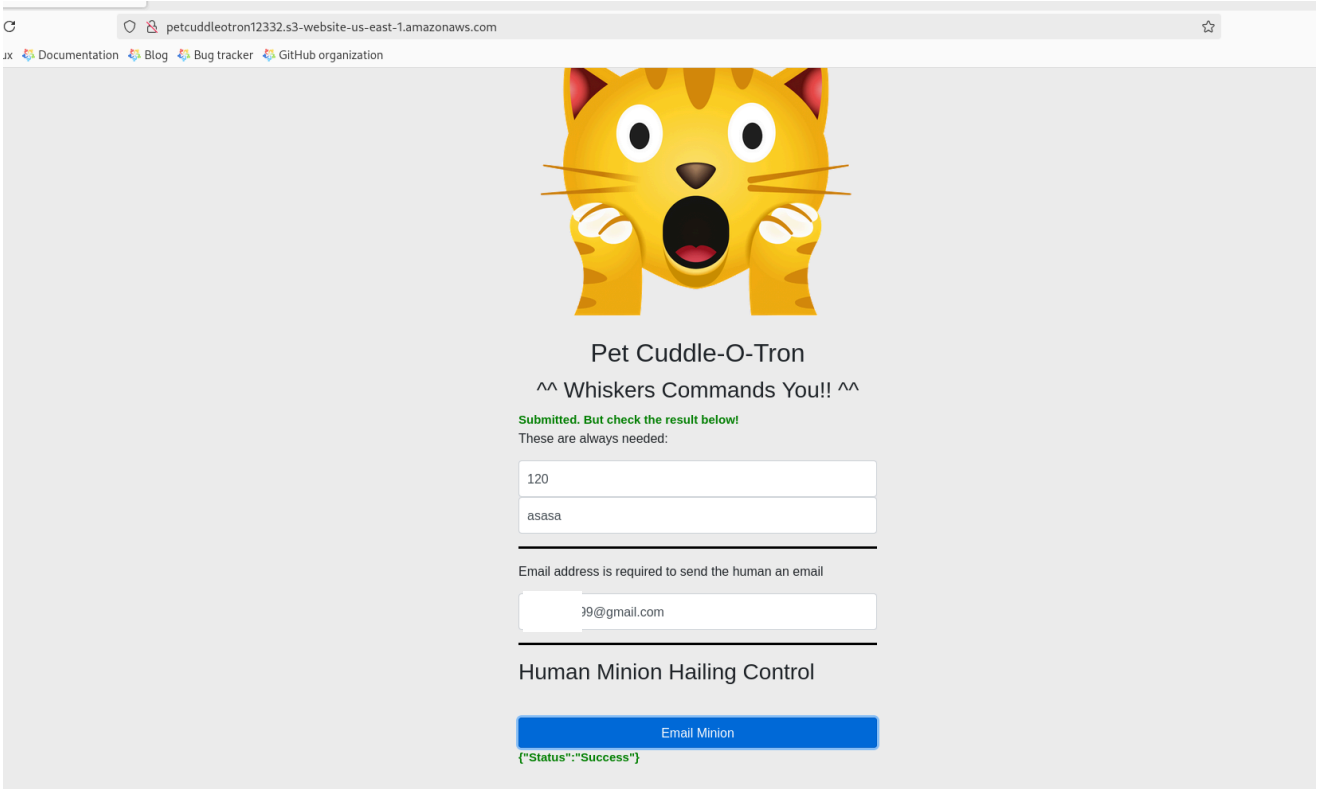
At this point, you have a **fully functional serverless application**:

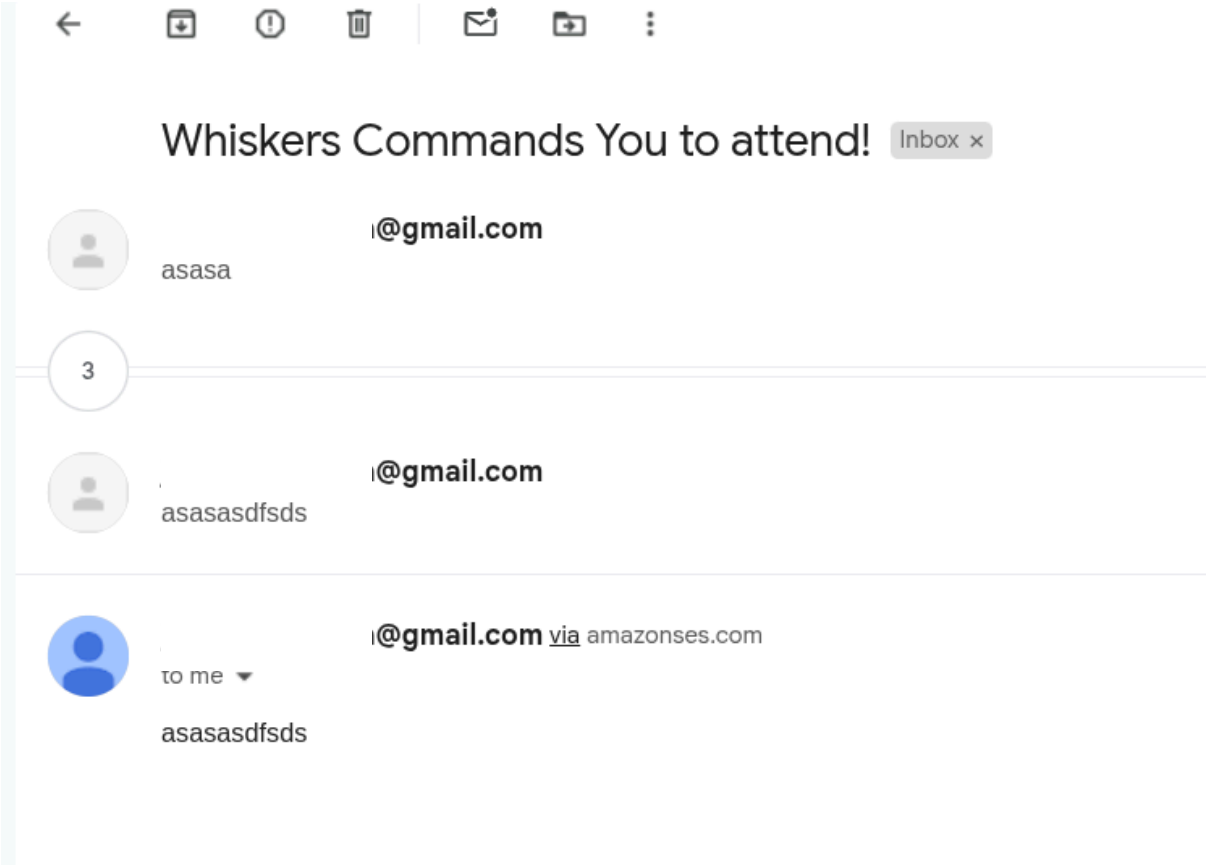
✅ **Front-end HTML & JavaScript** is hosted on **S3**
✅ **JavaScript communicates with API Gateway**
✅ **API Gateway routes requests to `api_lambda`**
✅ **`api_lambda` triggers Step Functions**
✅ **Step Functions sends an email using Lambda & SES**
✅ **The application runs entirely serverless** 🚀

This completes the main setup! 🎉

In **Stage 6**, you will clean up all AWS resources used in this demo.

file:///home/alejandroM/Documents/DevOps-Project2025/AWS-MINIPROJECT/Screenshot%20from%202025-03-01%2022-15-45.png


petcuddleotron12332.s3-website-us-east-1.amazonaws.com
Documentation
Blog
Bug tracker
GitHub organization
Pet Cuddle-O-Tron
^^ Whiskers Commands You!! ^^
Submitted. But check the result below!
These are always needed:
120
asasa
Email address is required to send the human an email
Human Minion Hailing Control
Email Minion
{"Status":"Success"}


Whiskers Commands You to attend!
Inbox
@gmail.com
asasa
3
@gmail.com
asasasdfsds
@gmail.com via amazonses.com
to me
asasasdfsds