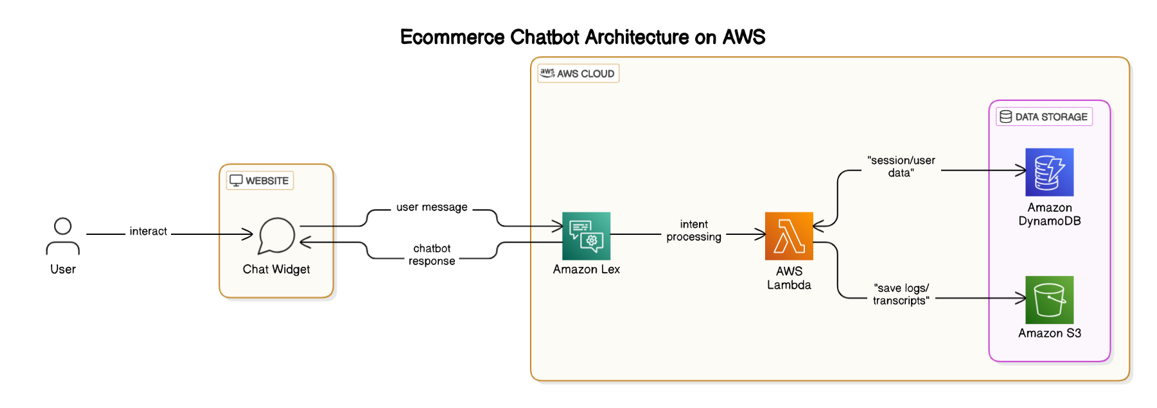
**AI Chatbot using Amazon Lex, Lambda, DynamoDB, and S3**

1. This capstone project demonstrates how to build an ecommerce chatbot that can track orders using AWS Lex V2, AWS Lambda, DynamoDB, and S3. The chatbot is also embedded into a custom HTML page which serves as the UI.

2. Below are the key components:

| **Component** | **Service / Technology** | **Role in Ecommerce Chatbot** |
| --- | --- | --- |
| **Bot UI** | HTML & JavaScript | Provides the **frontend chat interface** for users to interact with the bot on the website. |
| **NLU Engine** | Amazon Lex V2 | Handles **Natural Language Understanding** – interprets user queries and identifies intents. |
| **Business Logic** | AWS Lambda | Executes backend logic like **fetching order status from DynamoDB** and formulating replies. |
| **Authentication** | Amazon Cognito | Manages **user identity and permissions**, allowing unauthenticated or authenticated access. |
| **Hosting (optional)** | Amazon S3 | Stores and serves static files like **HTML, images, or transcripts** of conversations. |
| **Database** | Amazon DynamoDB | Stores **order details** (e.g., status and ETA) that Lambda retrieves based on order ID. |

3. Architecture:



4. Below are is the step by step guide to implement the chatbot per above architecture using components listed.

**1. IAM Setup**

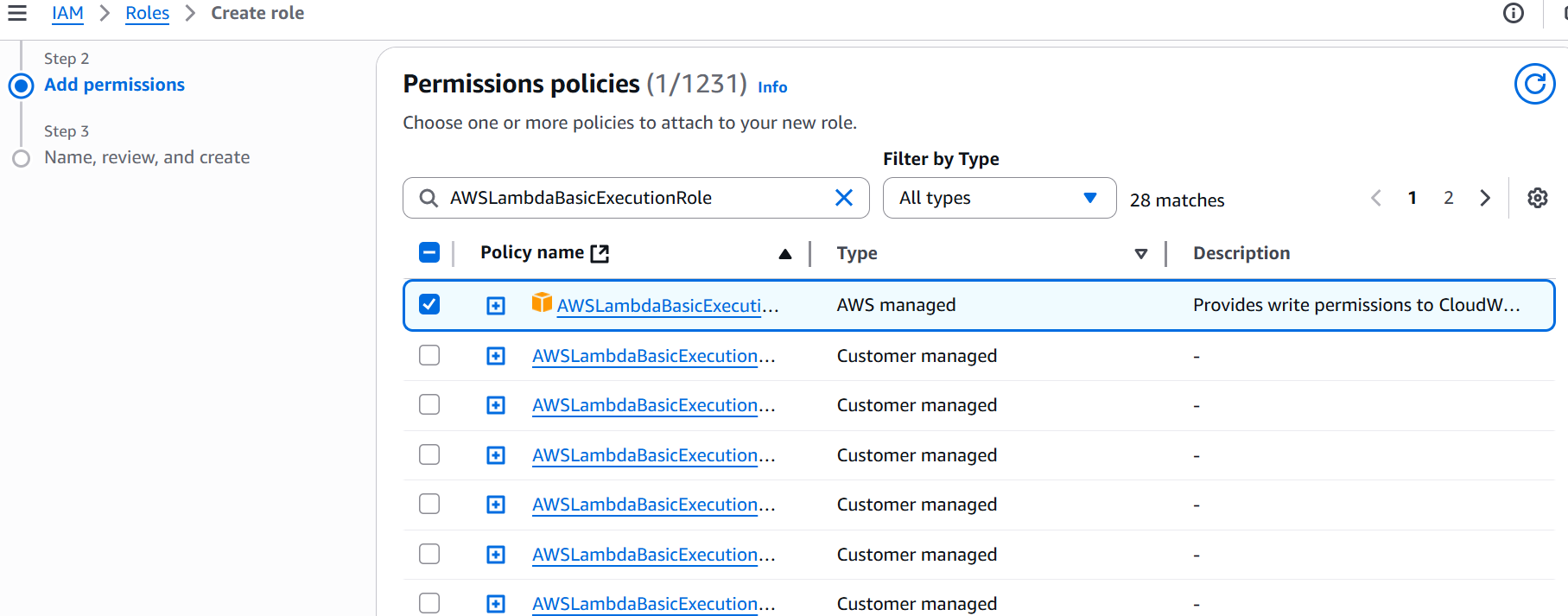
**Create IAM Role for Lambda:**

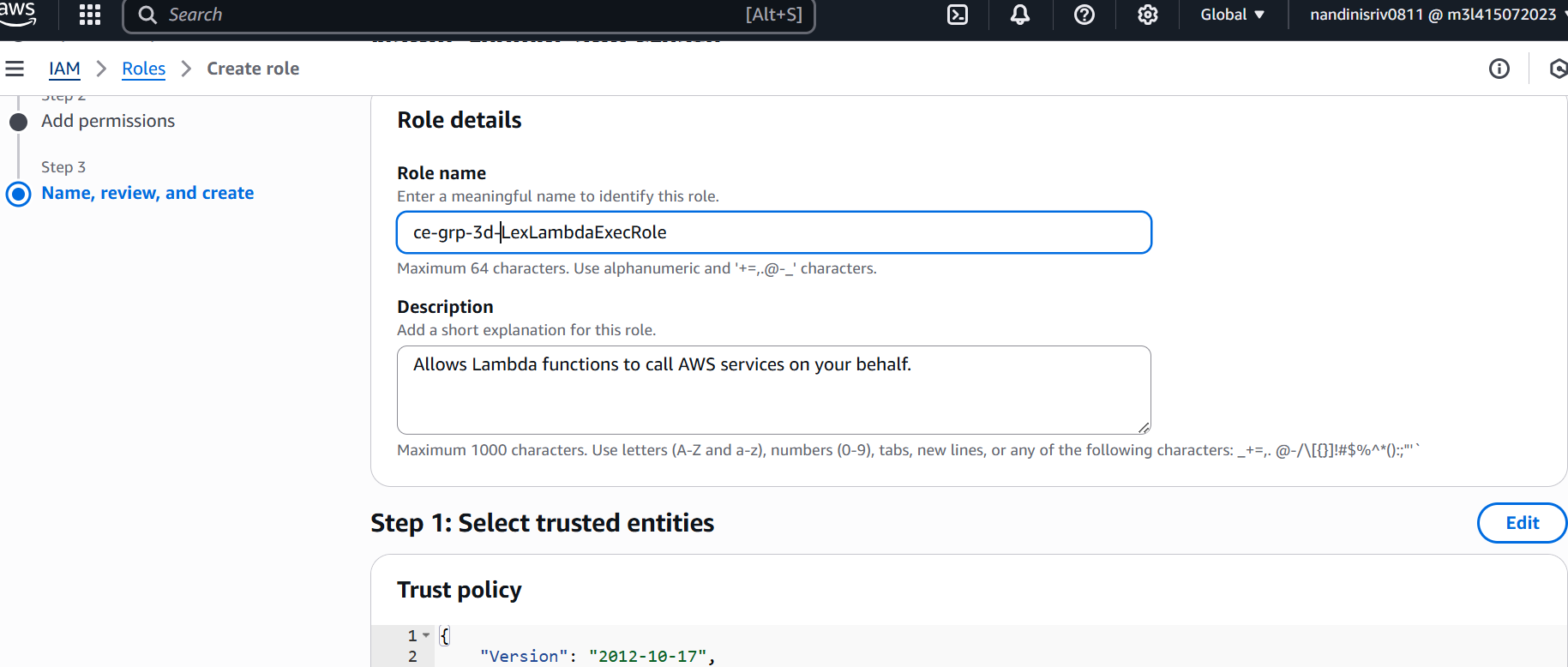
1. Go to **IAM > Roles > Create Role**.
2. Choose **Lambda** as the trusted entity.
3. Attach the following policies:
   * *AWSLambdaBasicExecutionRole*
   * *AmazonLexFullAccess*
   * *AmazonS3FullAccess*
   * *AmazonDynamoDBReadOnlyAccess*
4. Name it: ***ce-grp3d-LexLambdaExecRole***

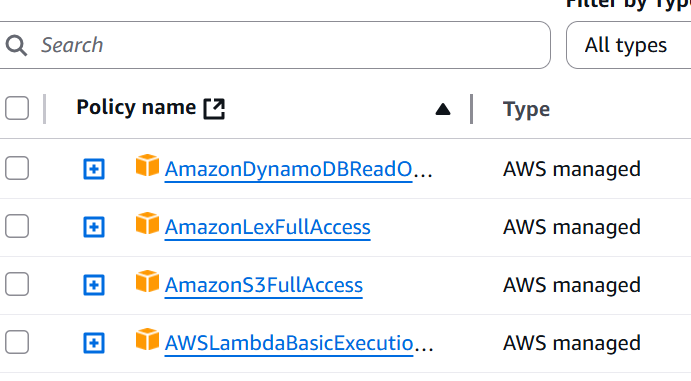
This role is **used by the Lambda function** so it can:

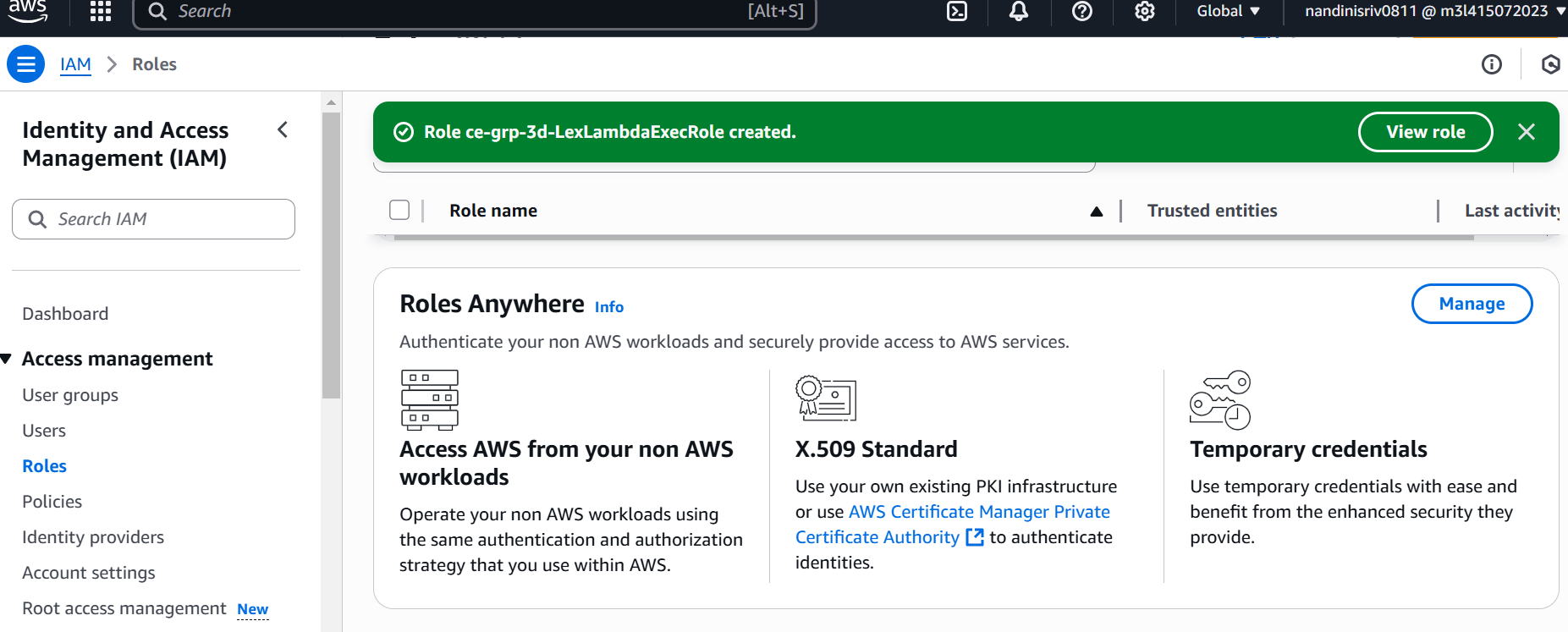
* Write logs to CloudWatch
* Be invoked by Lex
* Access S3 (to log chat transcripts)
* Read the database







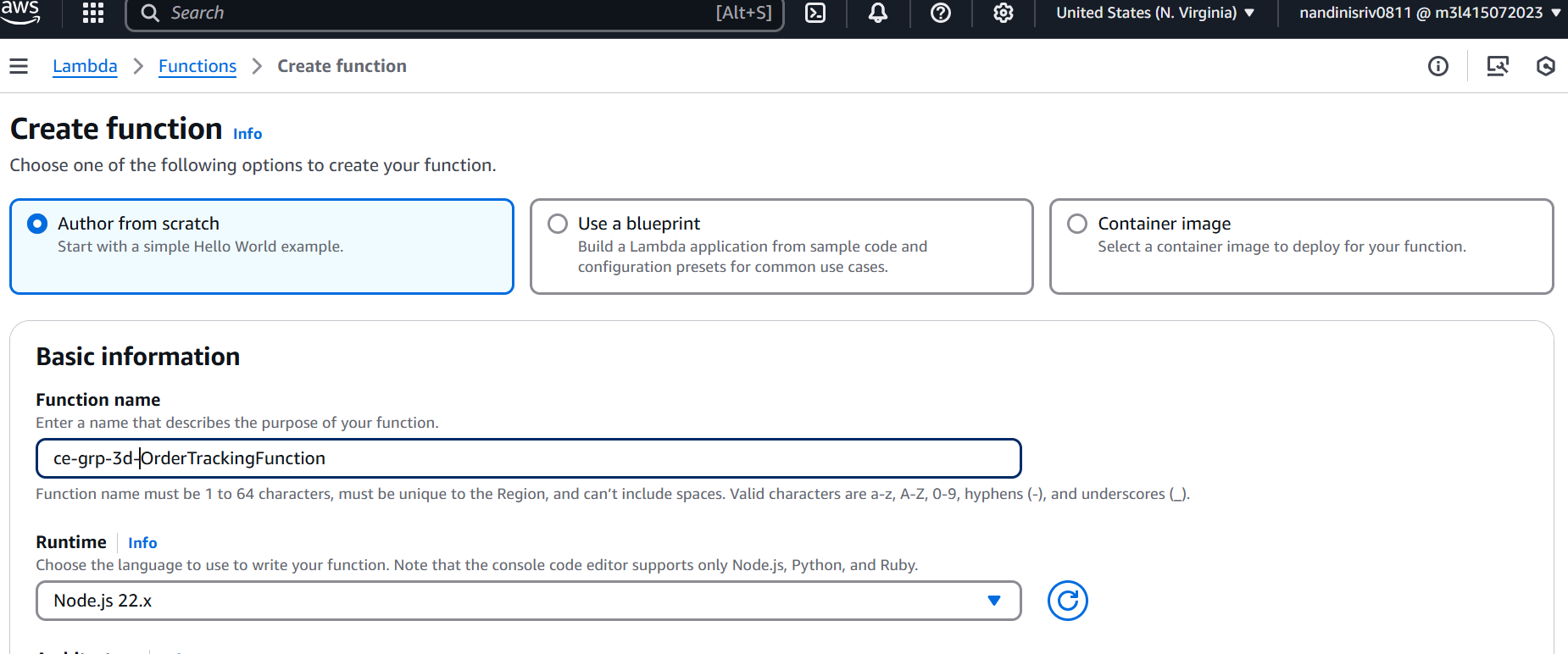


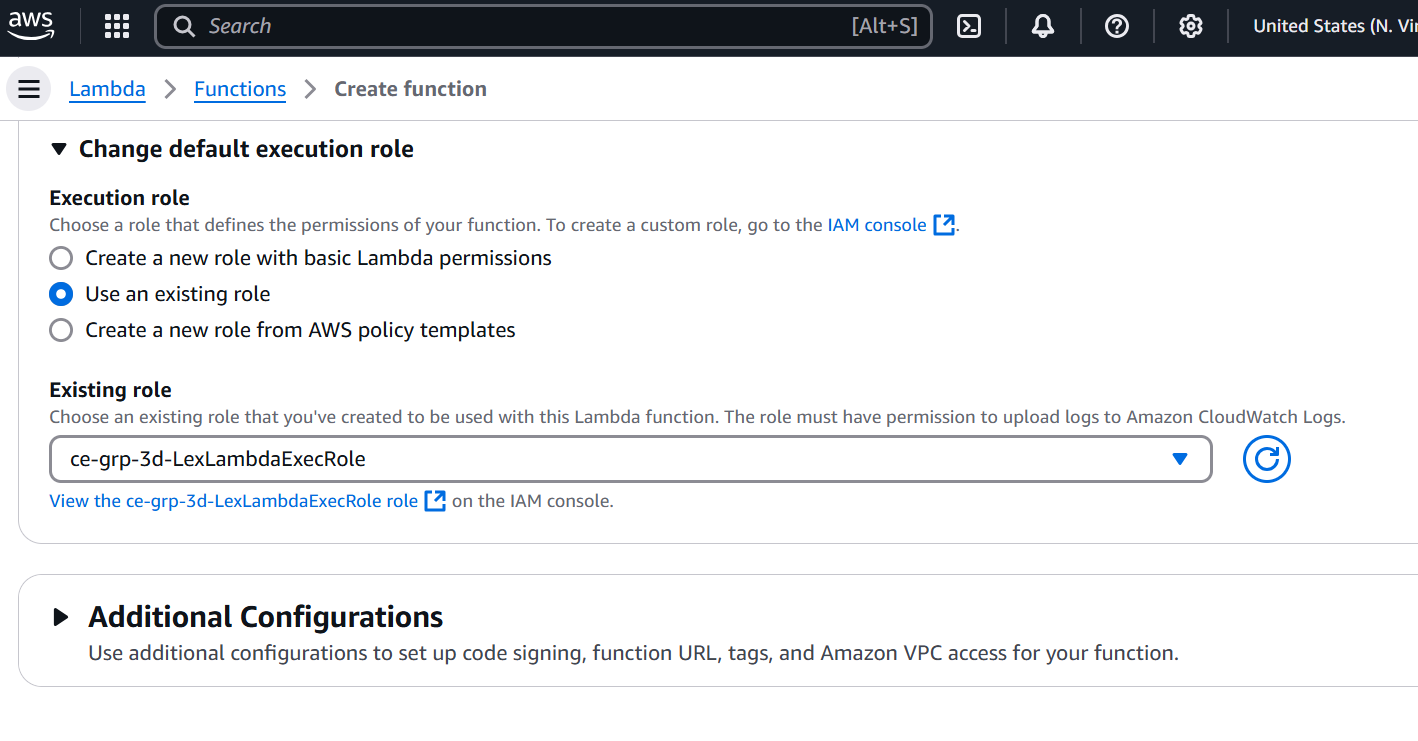


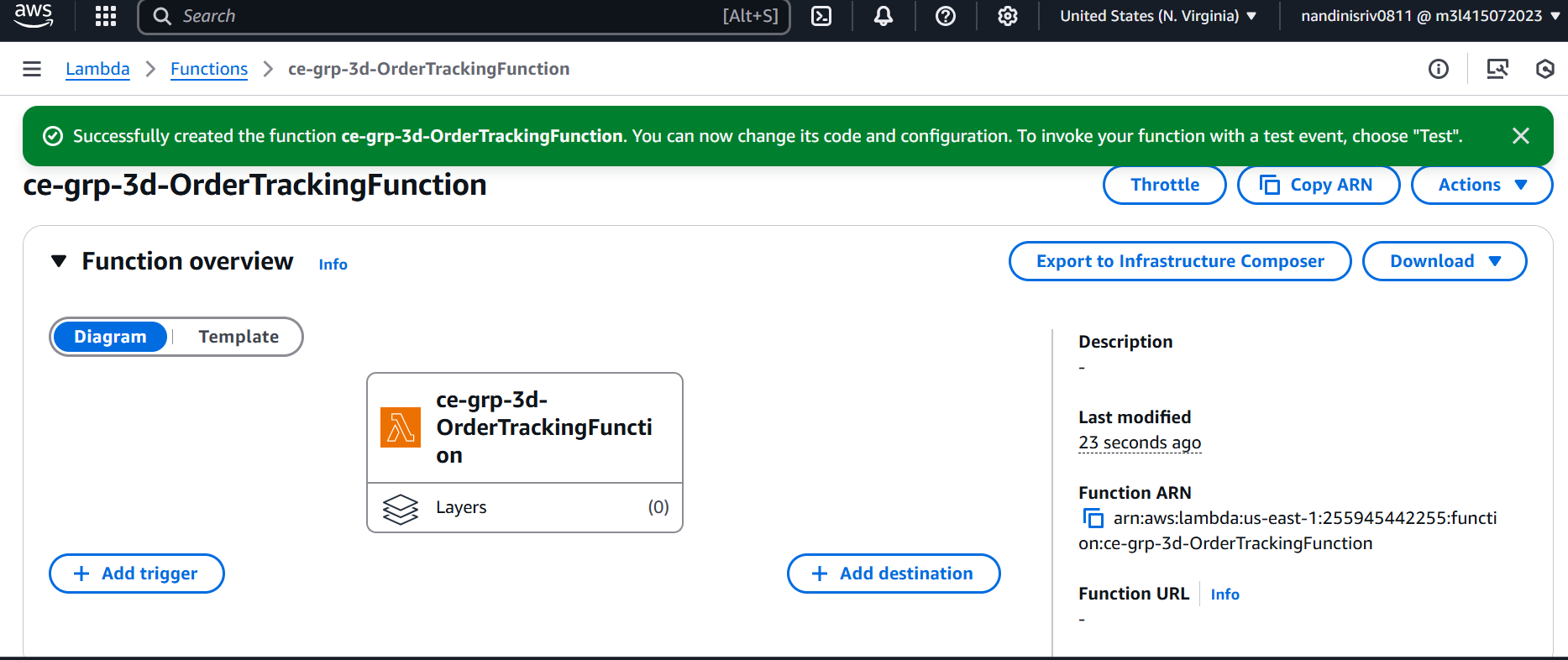
2. **Create AWS Lambda Function**

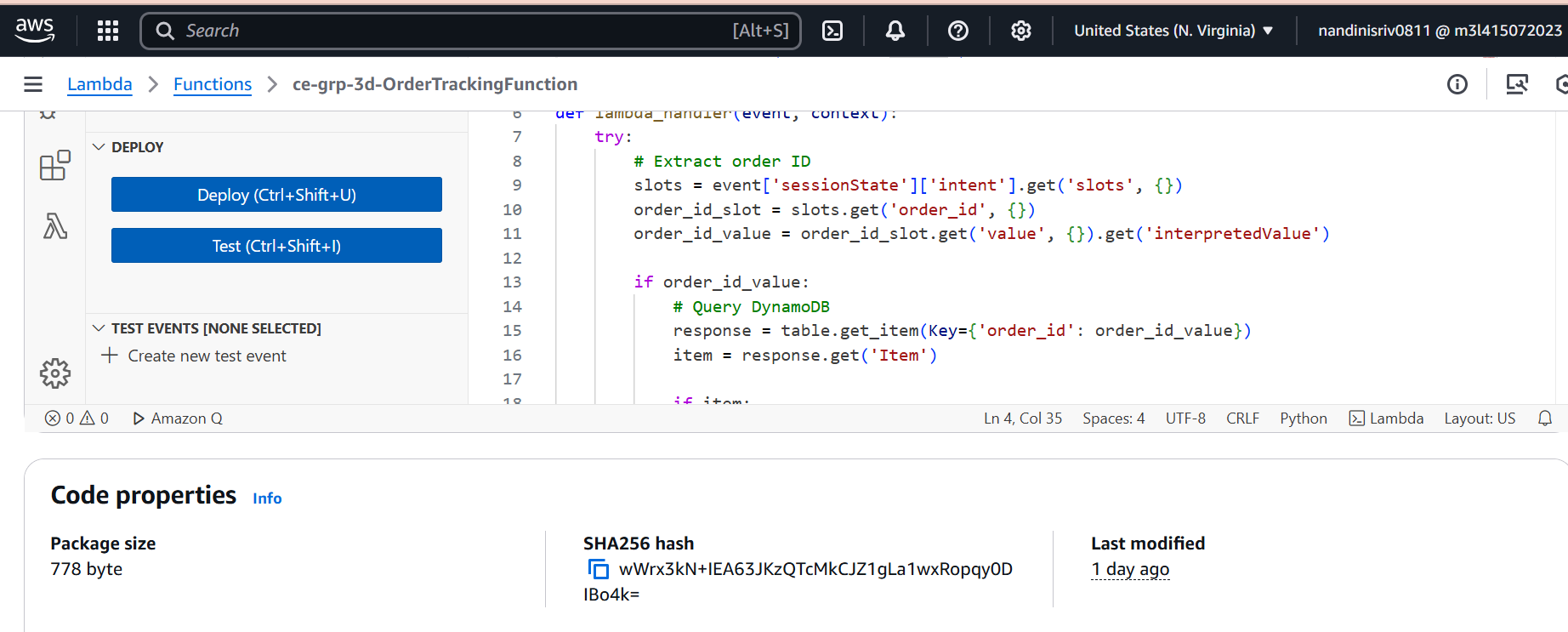
#### Function Setup:

1. Go to **Lambda > Create Function**
2. Name: *ce-grp-3d-OrderTrackingFunction*
3. Runtime: **Python 3.12 (as Lex V2’s integration is tested with Python 3.12 or lower)**
4. Execution role: Use ce-grp-3d-LexLambdaExecRole
5. In the code tab copy and paste the code from ce-grp-3d-OrderTrackingFunction.py in the github repo









## 3. **Grant Lex permission to invoke the Lambda function**

After the role ce-grp-3d-LexLambdaExecRole is attached to Lambda, run a one-time command via AWS CLI or CloudShell that directly adds a **resource-based policy** to the Lambda function.

*aws lambda add-permission \*

*--function-name ce-grp-3d-OrderTrackingFunction \*

*--statement-id AllowLexInvoke \*

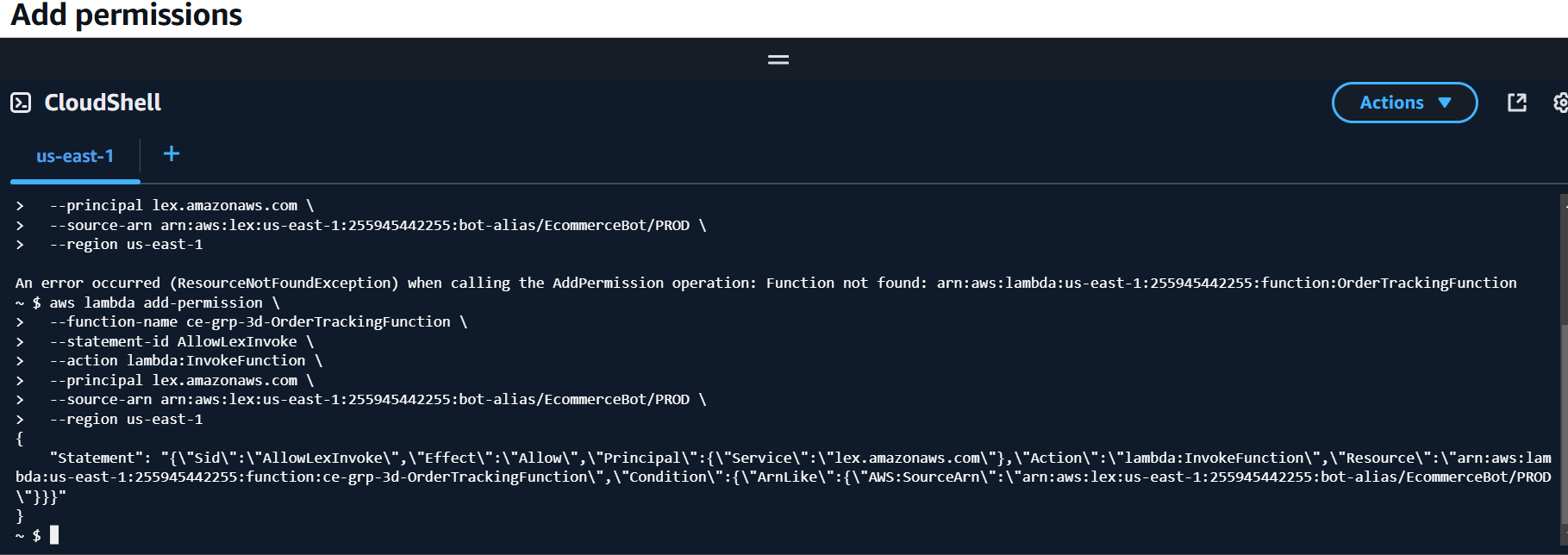
*--action lambda:InvokeFunction \*

*--principal lex.amazonaws.com \*

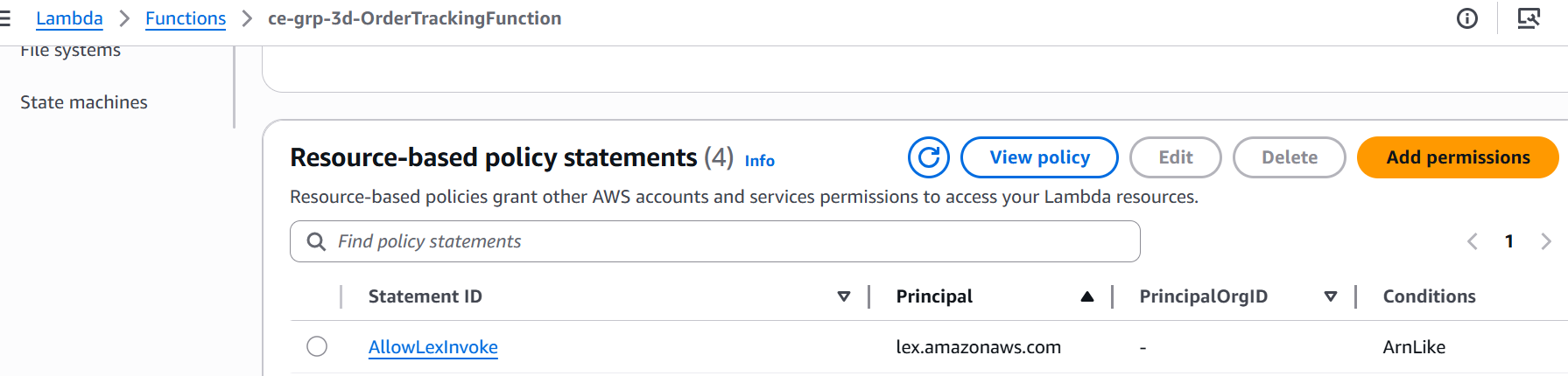
*--source-arn arn:aws:lex:us-east-1:<ACCOUNT\_ID>:bot-alias/ce-grp-3d-EcommerceBot/PROD \*

*--region us-east-1*

Replace < ACCOUNT\_ID> with your AWS account number (12 digits) – see screenshot below:



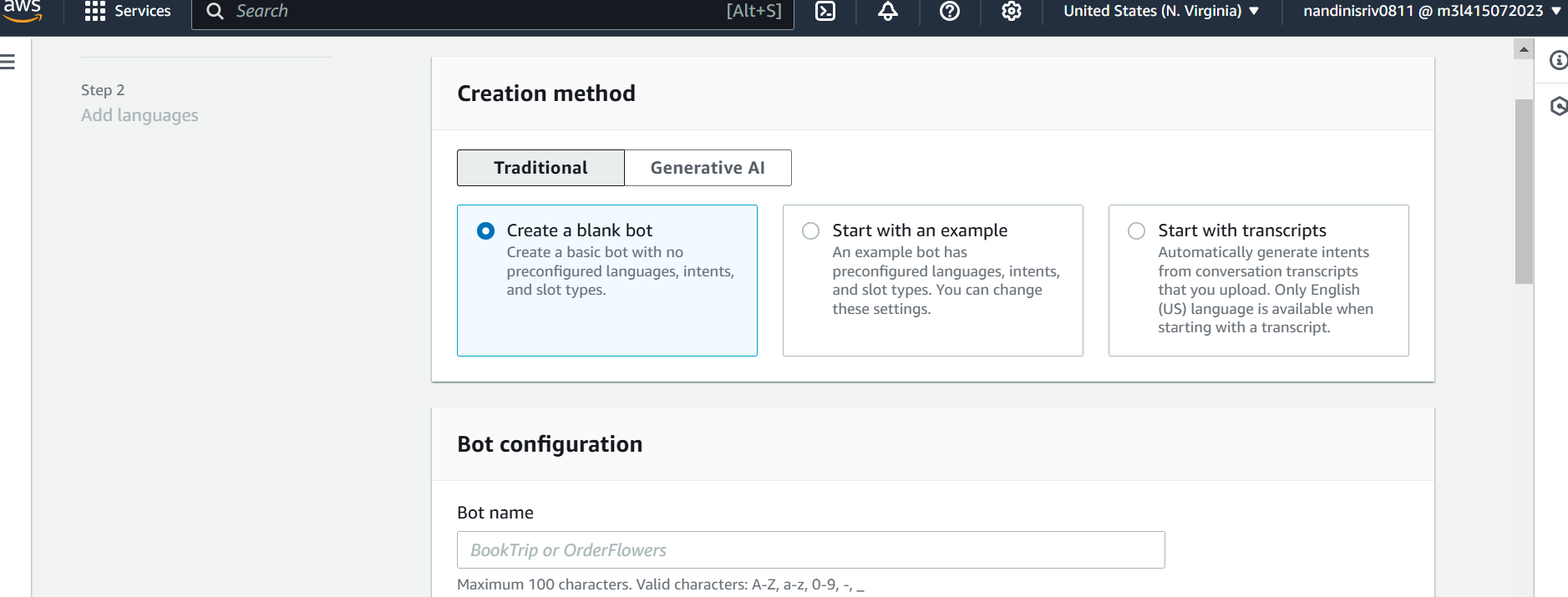
This creates a resource-based policy per below:

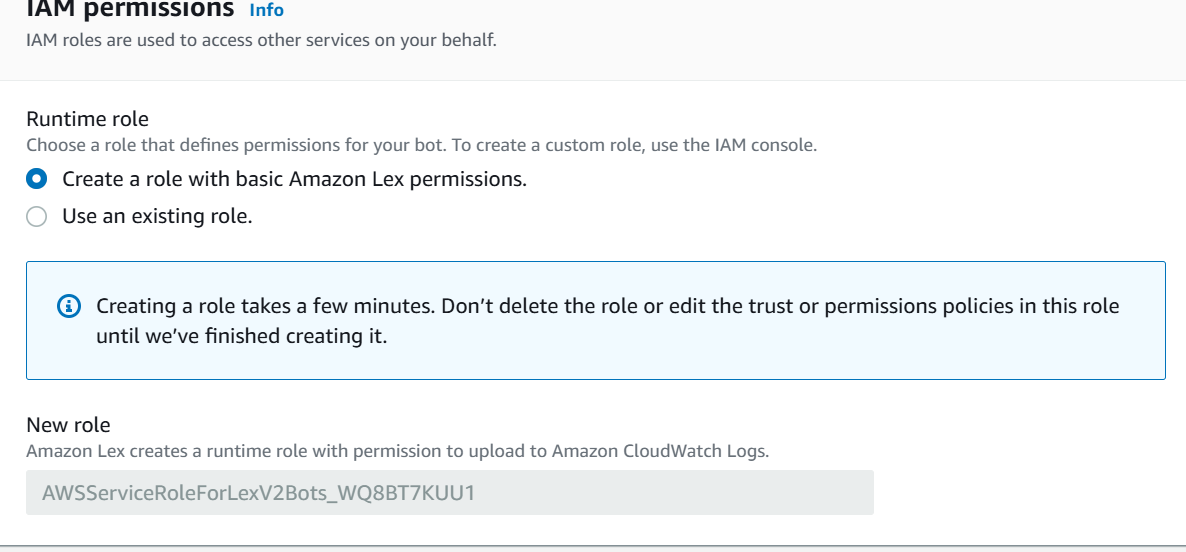


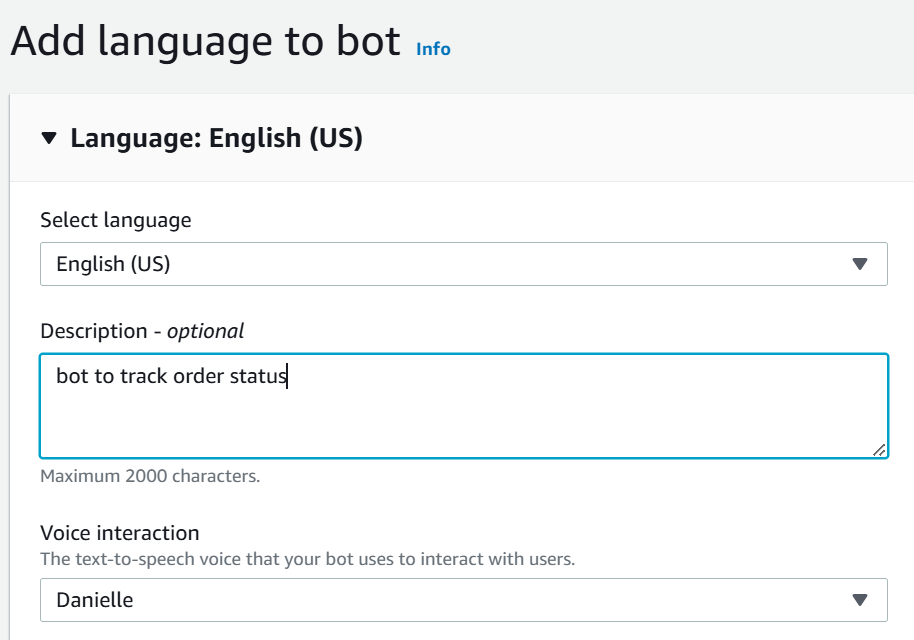
1. **Create Amazon Lex V2 Bot**

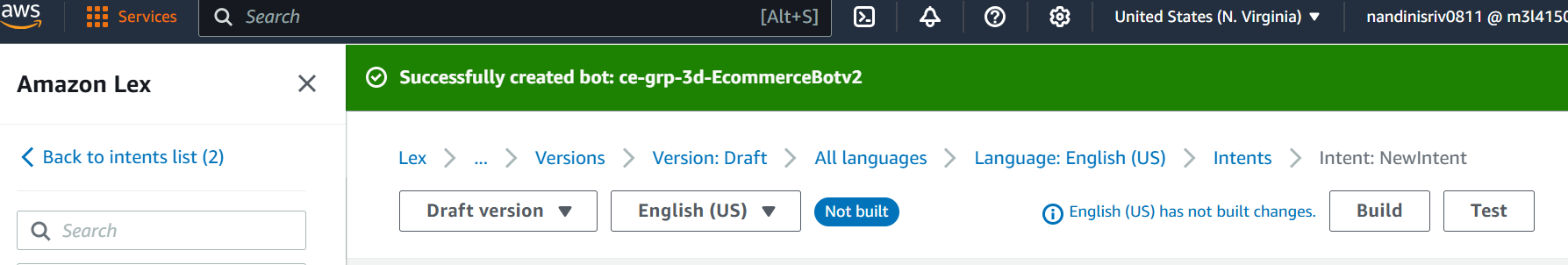
**Setup:**

1. Go to **Lex V2 > Create bot**
2. Bot name: ce-grp-EcommerceBot
3. IAM permissions: Create a new role with basic Lex permisions
4. Idle Session Timeout : 5 minutes (this defines how long Lex maintains session context - like slot values and dialog state- when the user is idle.) After this timeout, the session ends and the context is cleared.
5. Leave other settings as default









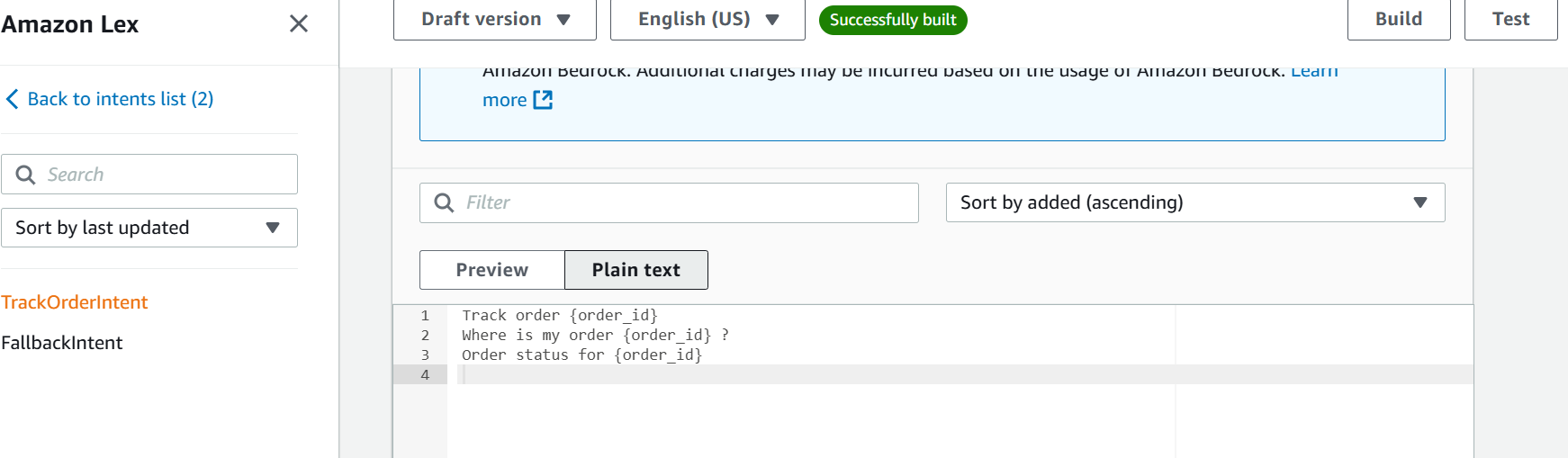
1. **Add Intent: TrackOrderIntent**

In Amazon Lex, **intents** represent the **purpose** or **goal** of what a user wants to do.

They are the core building blocks that drive how the chatbot understands and responds to user input.

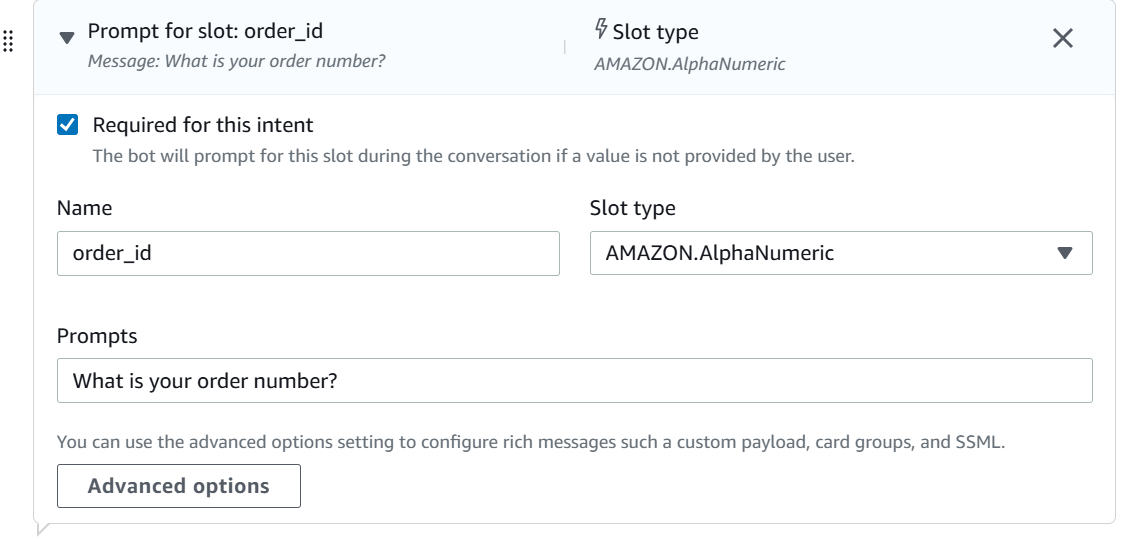
Each intent:

* Has a **name** (e.g., TrackOrderIntent)
* Has **sample utterances** (ways users say things)
* Can have **slots** (to collect details like order ID)
* Can trigger a **Lambda function** or just reply with a message
* Add Sample utterances in plain text:
  + "Where is my order {order\_id} ?"
  + "Track order {order\_id}"
  + "Check status for order {order\_id}"



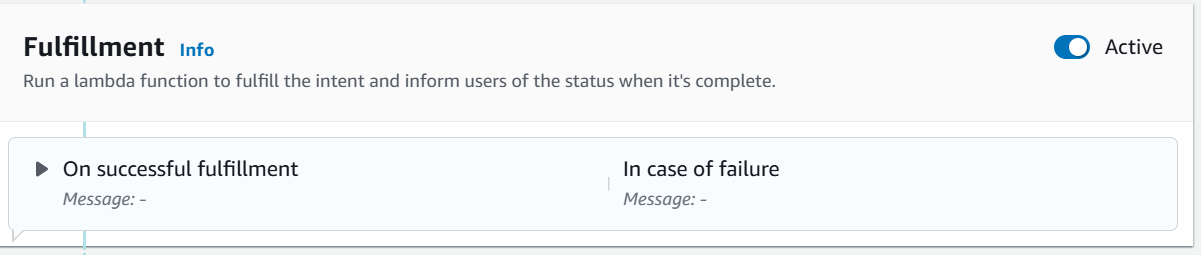
**Add Slot:**

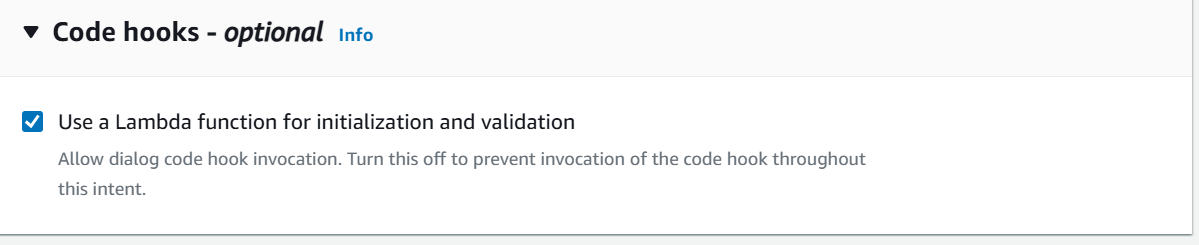
* Name: order\_id
* Slot type: AMAZON.AlphaNumeric
* Prompt: "Please enter your order ID"



**Fulfillment:**

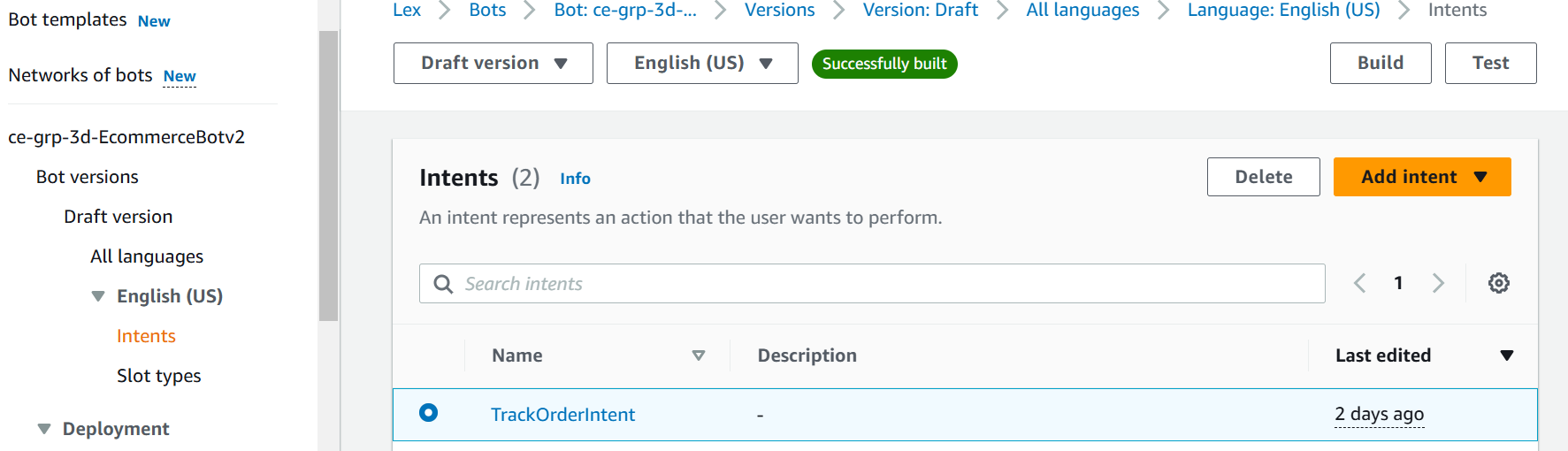
* Enable fulfilment
* Enable Use a Lambda function for initialization and validation





**Build and Deploy:**

* Build the TrackOrderIntent per the screenshot below :

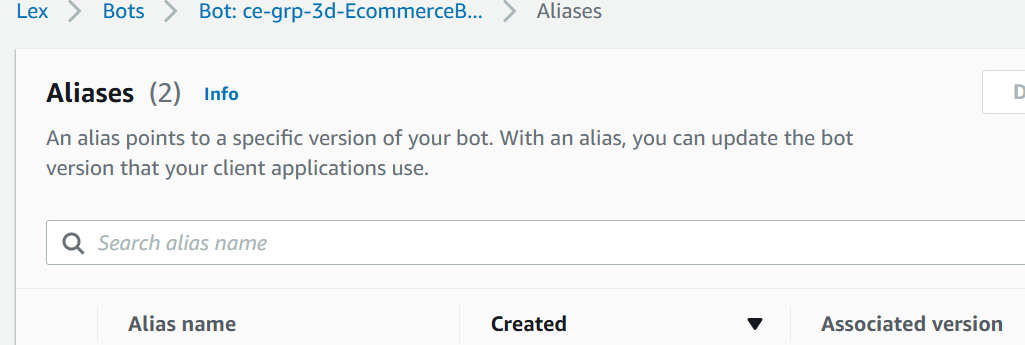


1. **Creation of an alias**

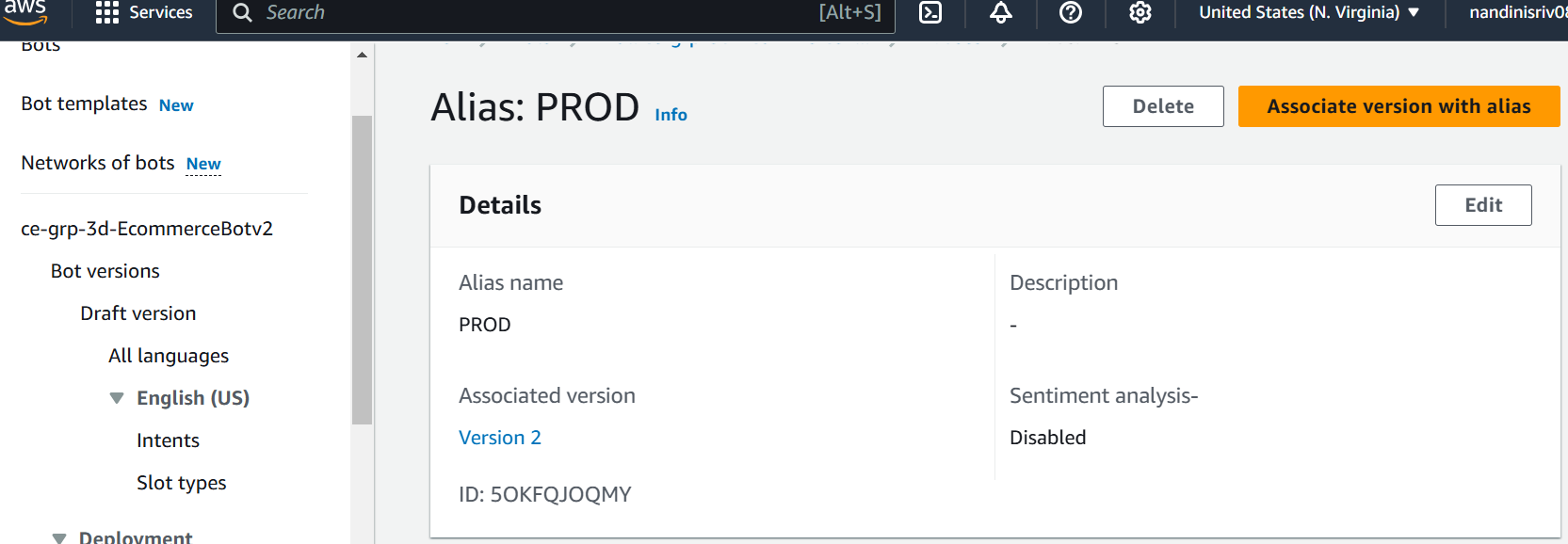
An **alias** is like a label (e.g., TEST, PROD) that points to a specific version of the bot.

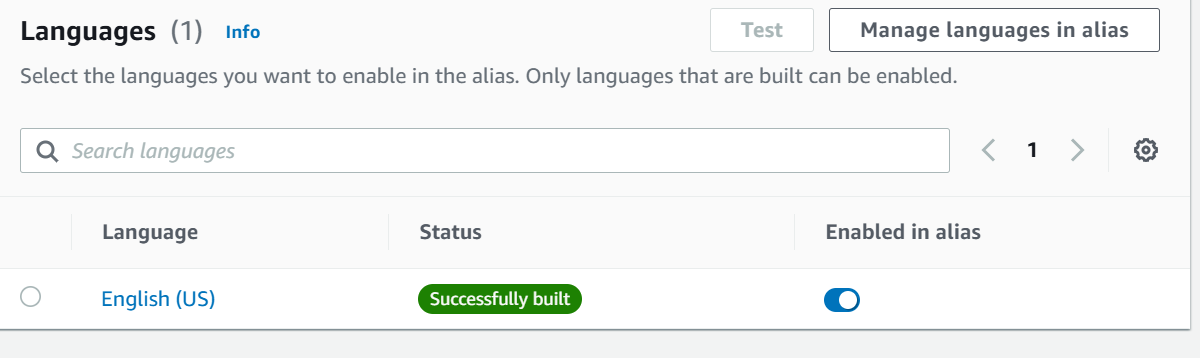
We need aliases because:

* They **decouple deployment from development**. We can update which version the alias points to without changing client apps.
* Our chatbot code (like the HTML file using the Lex SDK) **references an alias**, not a version directly. This keeps the integration flexible.
* We can **easily switch environments** by changing alias mappings (e.g., PROD points to version 3 instead of 2).



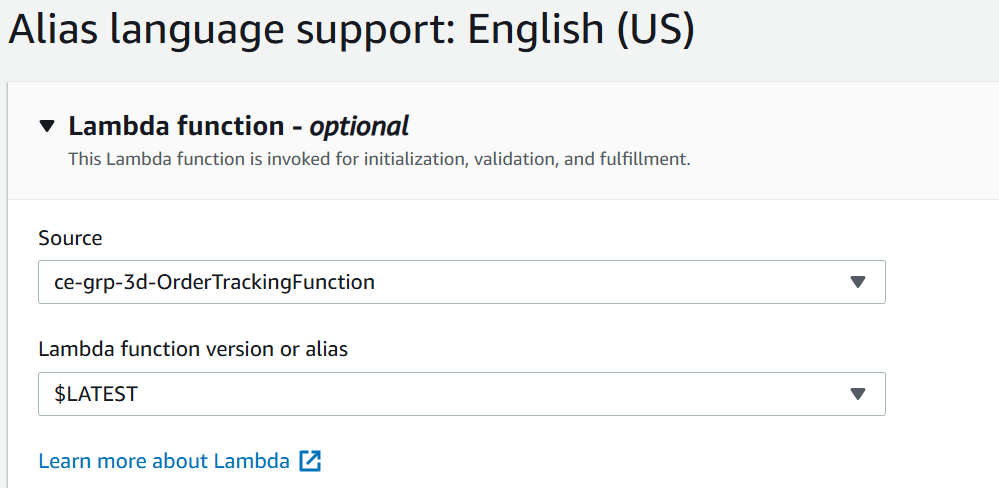
* Create an alias called PROD and enable Engish(US) per below screenshot:





1. **Link the Lambda function to the Bot Alias**

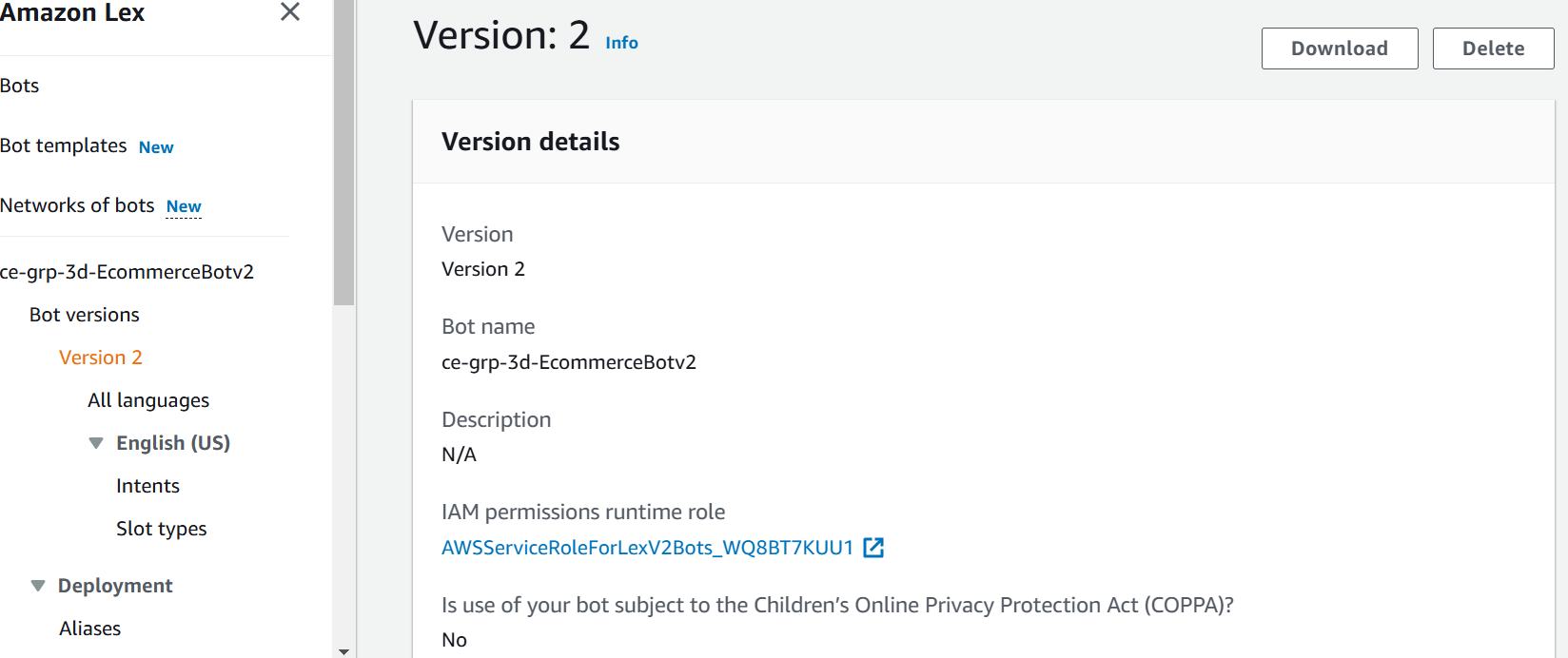
* The lambda function *ce-grp-3d-OrderTrackingFunction* is linked to the Alias *PROD* so that the lambda function is called every time the keywords in the intent are typed by the user to process a response.
* Also choose $LATEST to refer to latest version of the lambda code

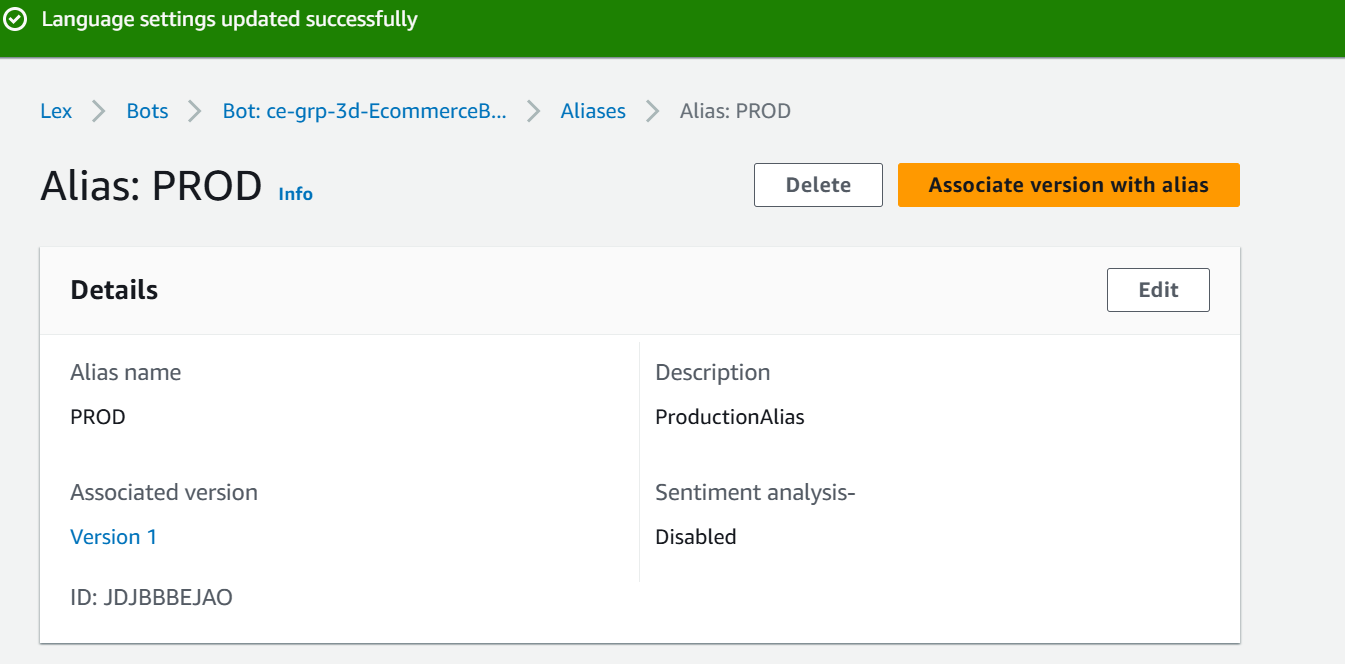


1. **Create a version and link the alias to the version**

Once we **create a version of the bot**, it becomes immutable (unchangeable). This ensures:

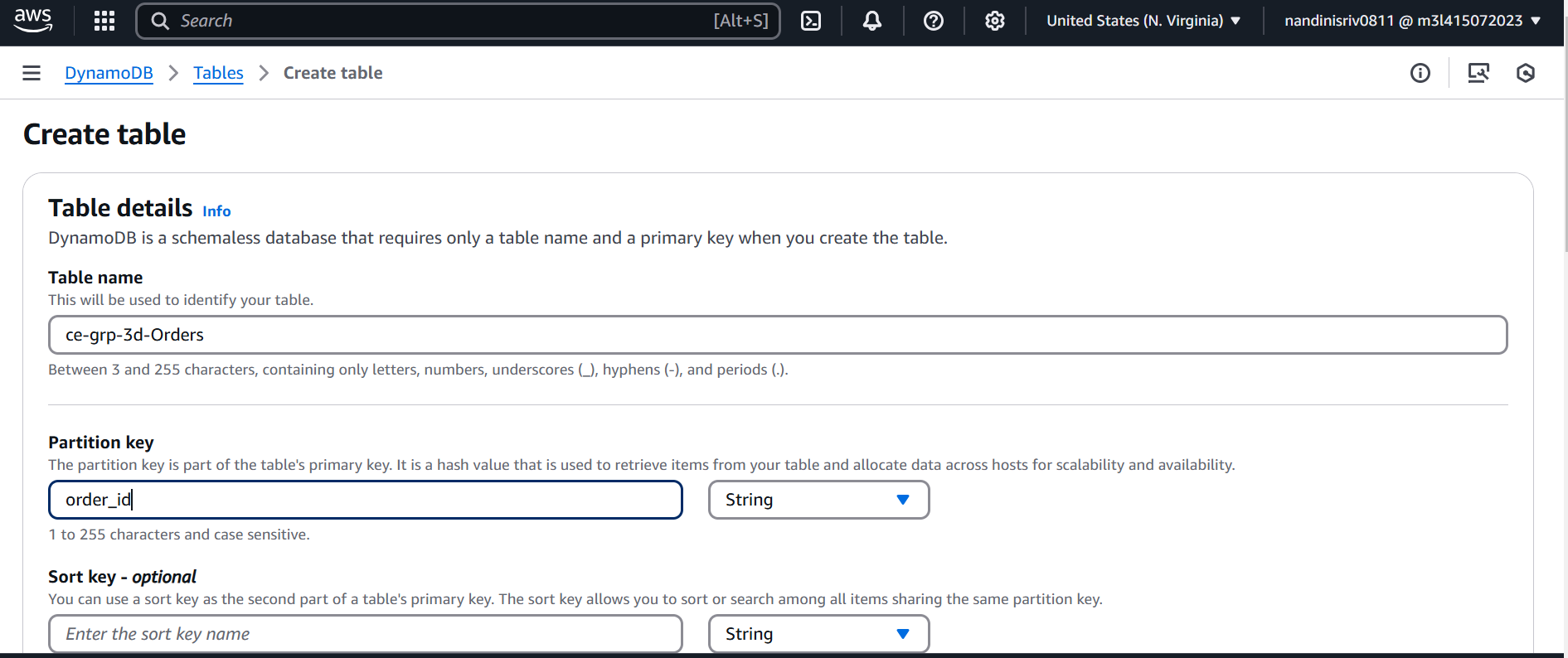
* **Stability in production**: Our deployed bot won’t break if changes are made to the draft.
* **Safe experimentation**: We can edit the **draft** freely without affecting the active version.
* **Rollback support**: We can revert to an older version if needed.



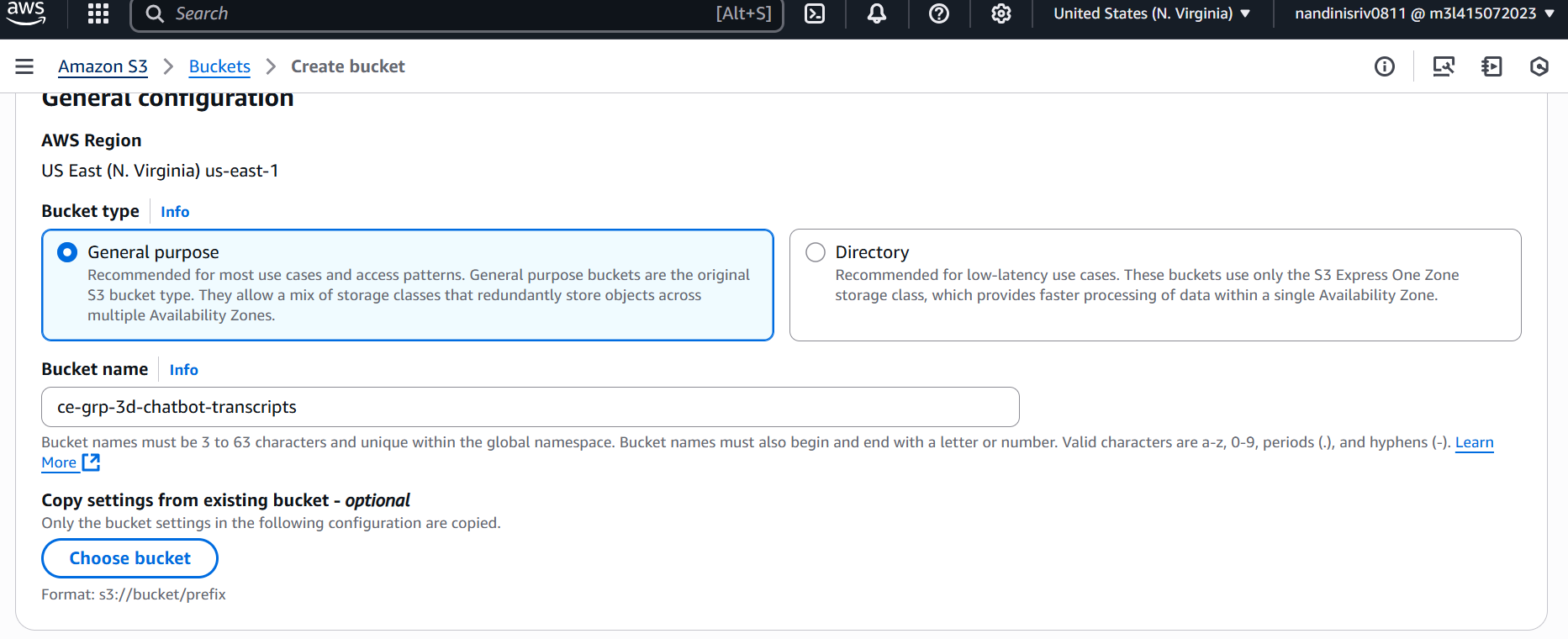


1. **Create the dynamodb table with order details per below:**

Next also insert items in the table created for various orders with different eta/status too.

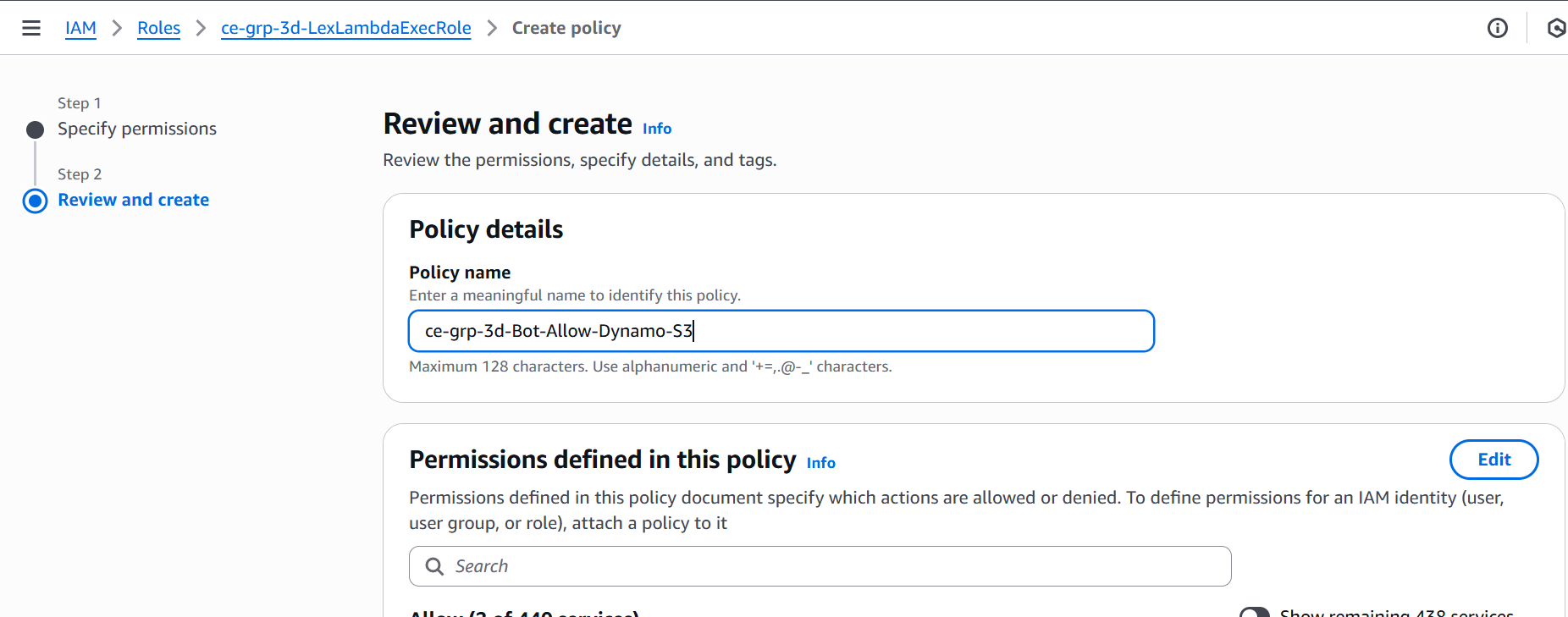


1. **Create S3 bucket to store chat transcripts:**



1. **Enable permissions for Lambda to write to the S3 bucket and read dynamo db table:**

* Add new inline policy to the Role *ce-grp-3d-LexLambdaExecRole* and name it **ce-grp-3d-Bot-allow-Dynamo-S3**



*{*

*"Version": "2012-10-17",*

*"Statement": [*

*{*

*"Effect": "Allow",*

*"Action": [*

*"dynamodb:GetItem"*

*],*

*"Resource": "arn:aws:dynamodb:us-east-1:255945442255:table/ce-grp-3d-Orders"*

*},*

*{*

*"Effect": "Allow",*

*"Action": [*

*"s3:PutObject"*

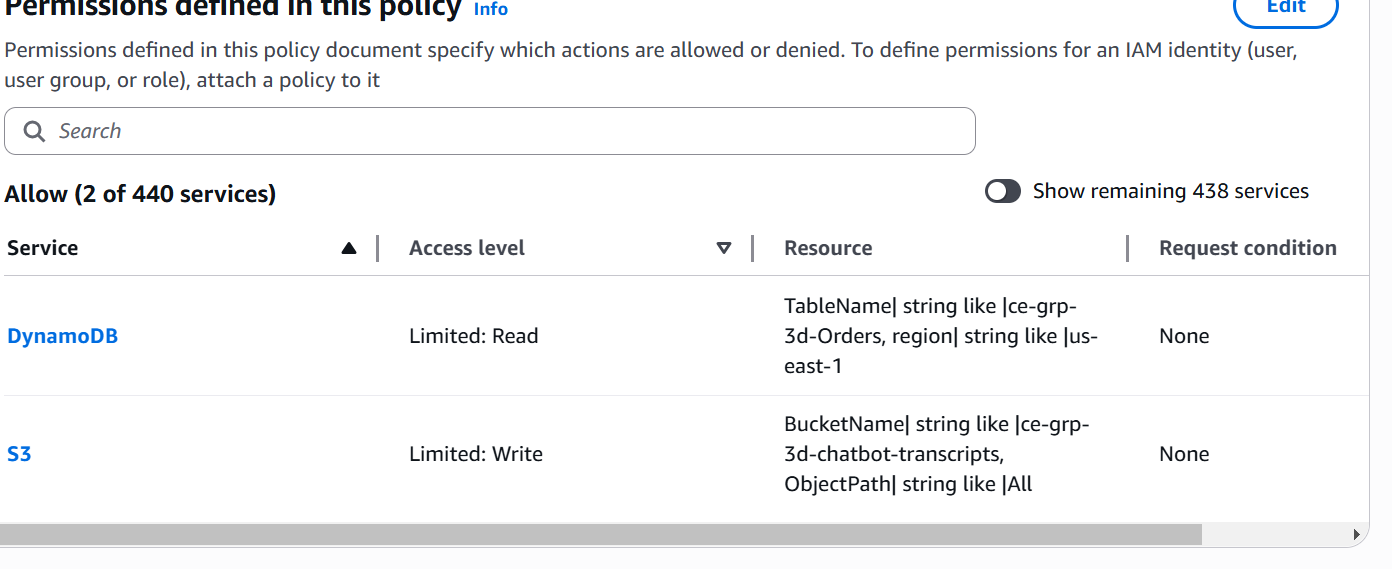
*],*

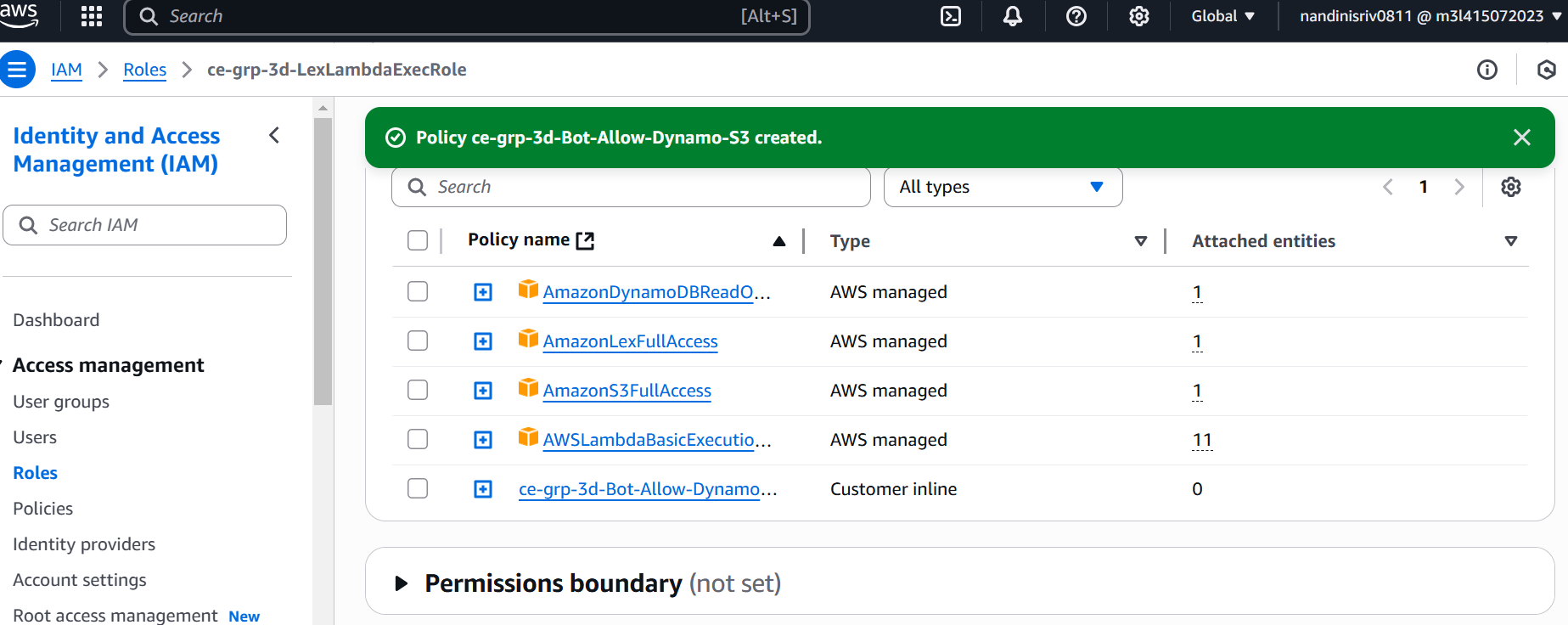
*"Resource": "arn:aws:s3:::ce-grp-3d-chatbot-transcripts/\*"*

*}*

*]*

*}*

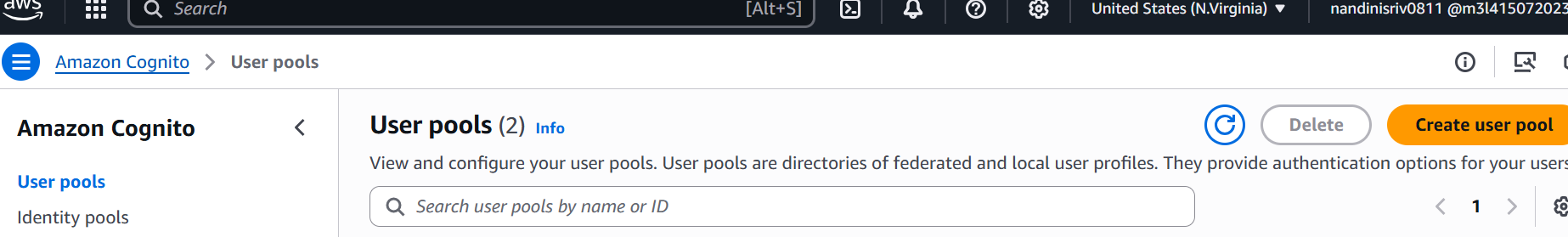


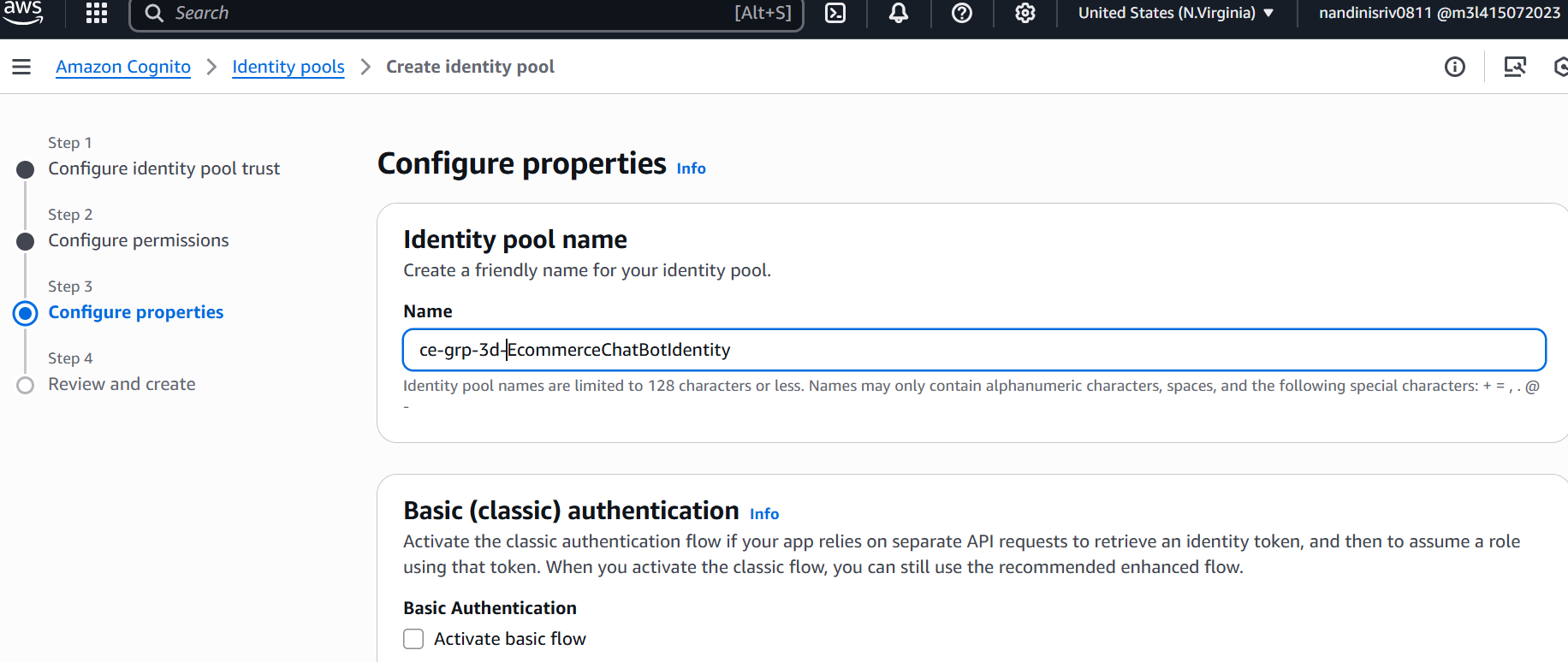


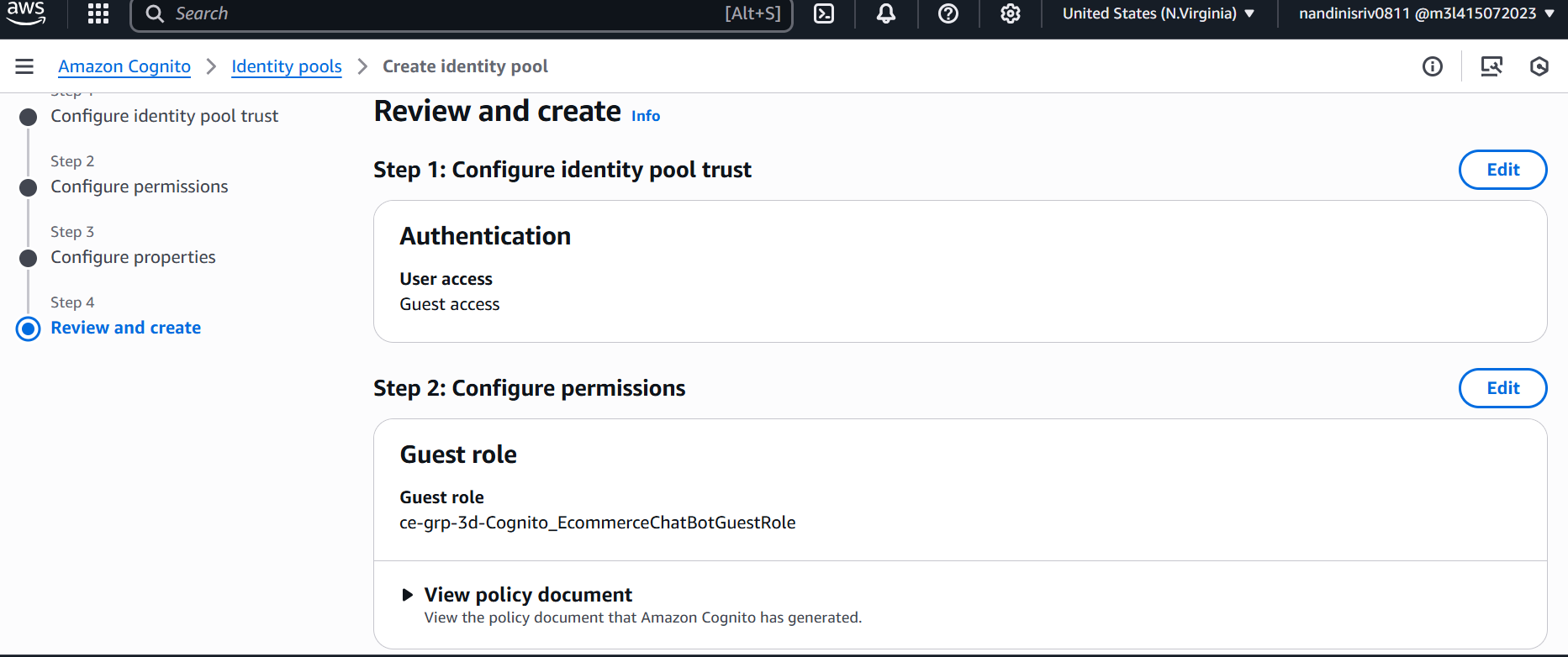
1. **Setup Cognito Identity Pool for authentication**

* Amazon Cognito is used for **authentication and access control in the chatbot.**
* Lex V2 requires AWS credentials to call its RecognizeText or RecognizeUtterance API.  
  Cognito generates these temporary credentials for your users securely — without exposing AWS secrets.
* Cognito is the secure bridge between your static frontend chatbot and AWS services like Lex and S3, with minimal setup and maximum security.

1. Go to **Amazon Cognito → Federated Identities**
2. Click “Create new identity pool”
   * Name: *ce-grp-3d-EcommerceChatBotIdentity*
   * Enable Guest access to allow website users to access the chatbot
   * Create new IAM role
3. Click “Create Pool”









1. **Setup Cognito Role to allow invoking Lex**

 Go to **IAM Console > Roles**

* Find Cognito role (ce-grp-3d-EcommerceChatBotIdentity)
* Go to the **Permissions** tab
* Click “Add inline policy”
* Switch to the JSON tab and paste below policy to allow
* REPLACE :<AWS\_ACCOUNT\_ID> with 12-digit AWS account ID
* REPLACE <BOT ID> with unique id found on clicking the properties of the bot (Lex Console->Bots->ce-grp-3d-EcommerceBot)
* REPLACE <ALIAS ID> with unique id found on clicking the properties of the bot alias (Alias->PROD)

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"cognito-identity:GetCredentialsForIdentity",

"cognito-identity:GetId"

],

"Resource": "\*"

},

{

"Effect": "Allow",

"Action": "lex:RecognizeText",

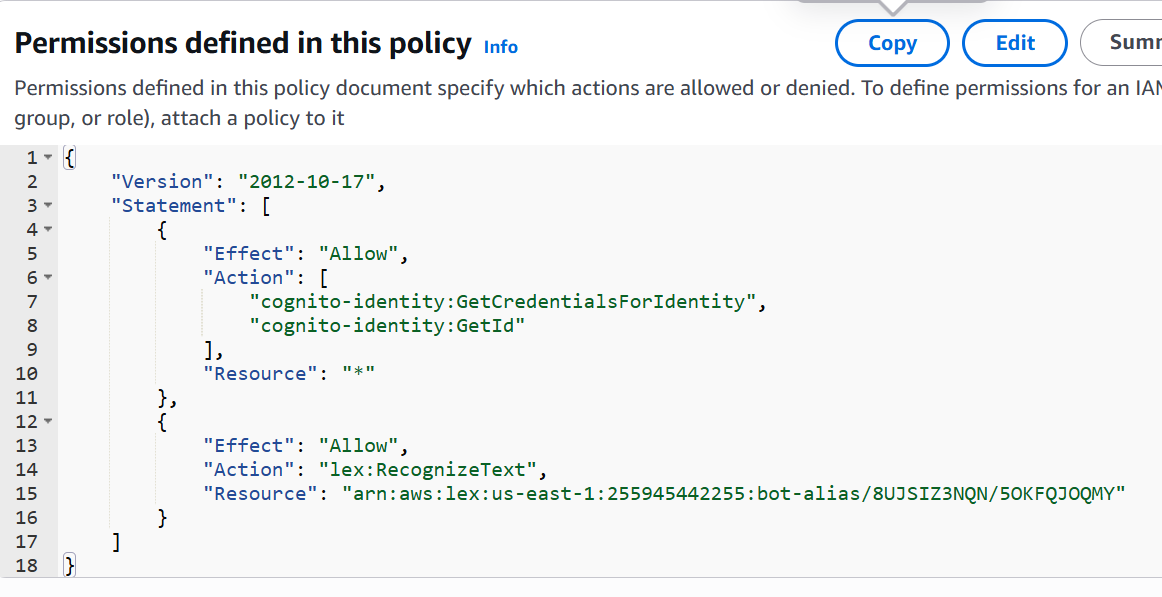
"Resource": "arn:aws:lex:us-east-1:<AWS\_ACCOUNT\_ID>:bot-alias/<BOT ID>/<ALIAS ID>"

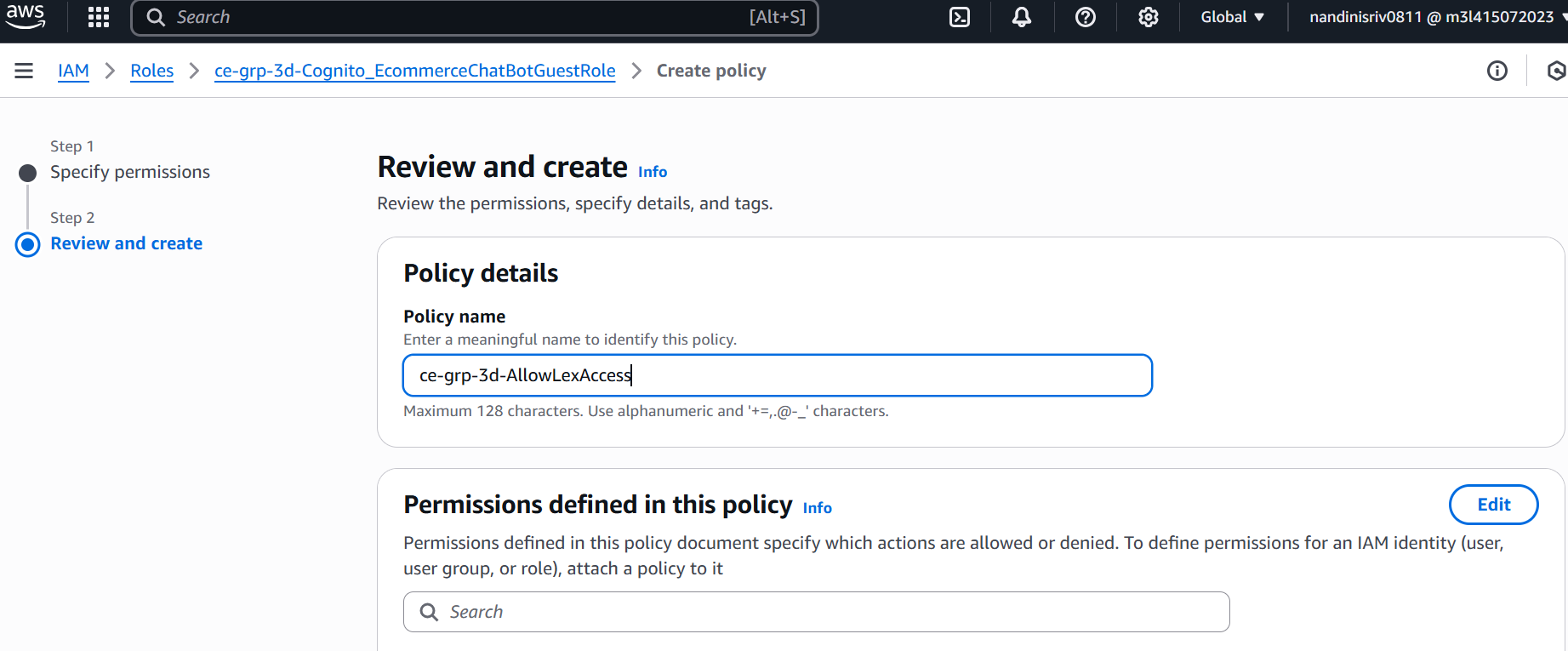
}

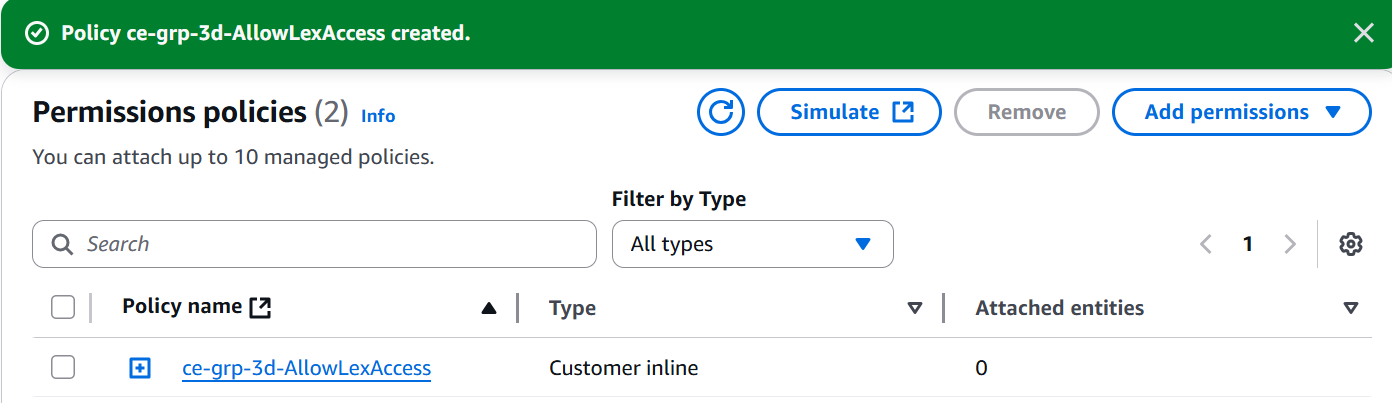
]

}

1. **Setup Cognito Role to allow invoking Lex**







1. **Integrating with UI**

* To call the chatbot from the front-end, amend the html file in the repo: Chatbot\_html per below:

1. Go to the <script> section of the html page and amend the below highlighted fields.
   * Replace region with region linked to AWS account.
   * Replace Cognito Identity Pool ID with the pool id of the Cognito pool – ce-grp-EcommerceChatBotIdentity

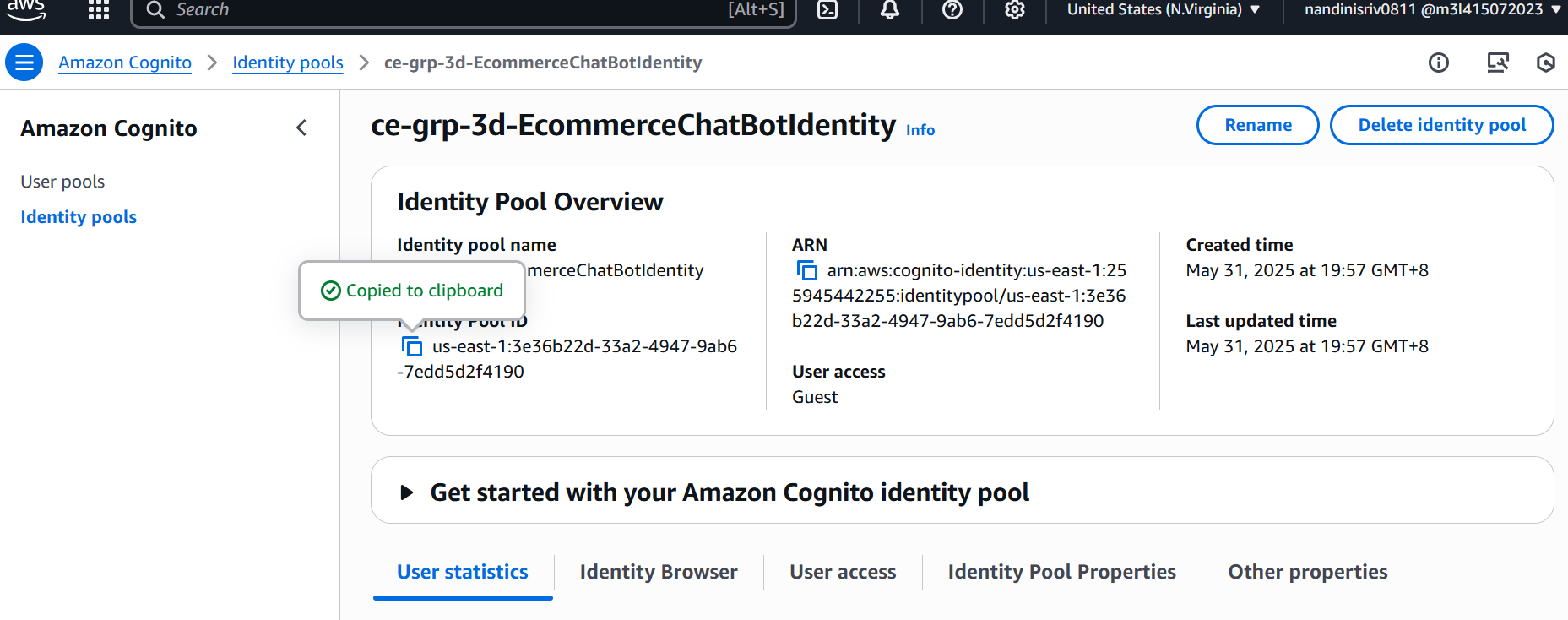
*<script>*

*AWS.config.region = 'us-east-1';*

*AWS.config.credentials = new AWS.CognitoIdentityCredentials({*

*IdentityPoolId: 'us-east-1:3e36b22d-33a2-4947-9ab6-7edd5d2f4190'*

*});*



1. Make the additional changes in the UI script:

*try {*

*AWS.config.credentials.get(async function () {*

*const params = {*

*botId: '8UJSIZ3NQN',*

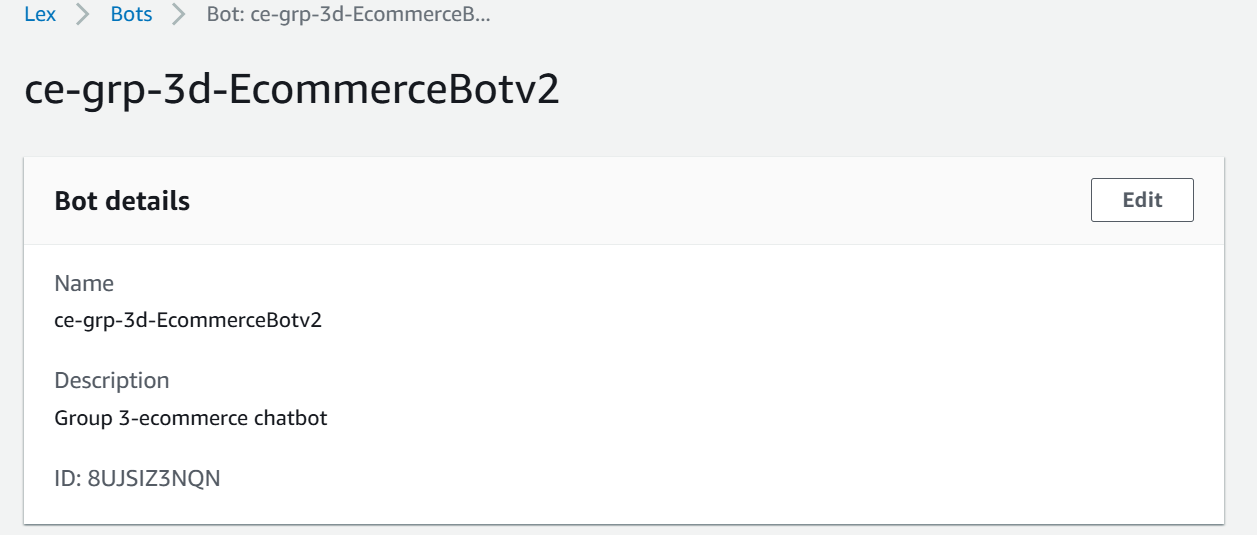
*botAliasId: '5OKFQJOQMY',*

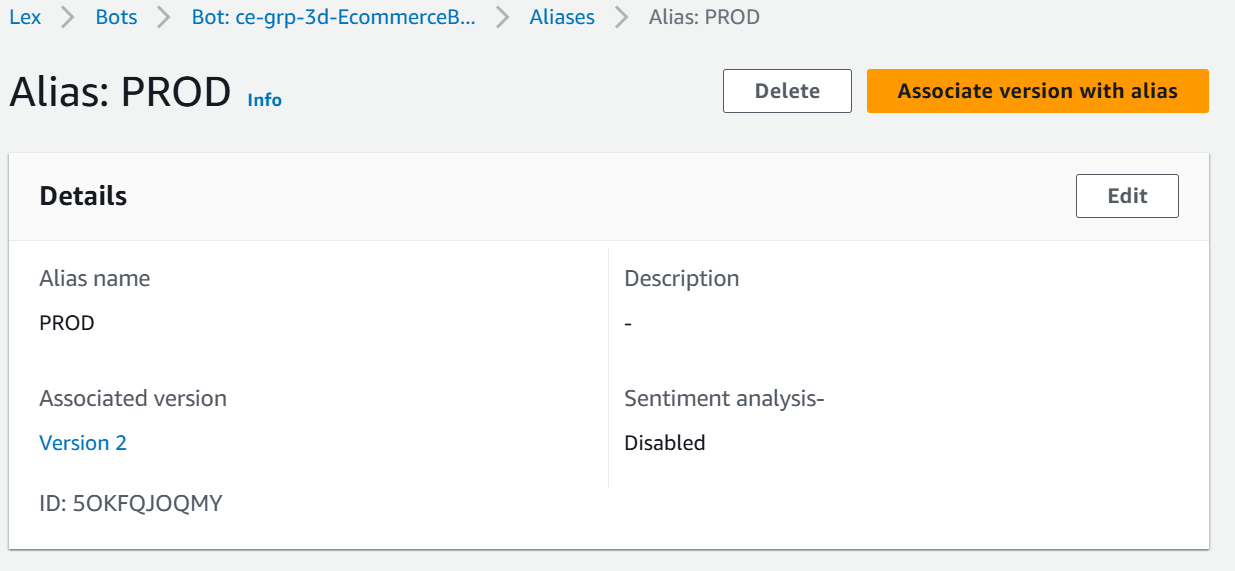
*localeId: 'en\_US',*

*sessionId: userId,*

*text: message*

*};*

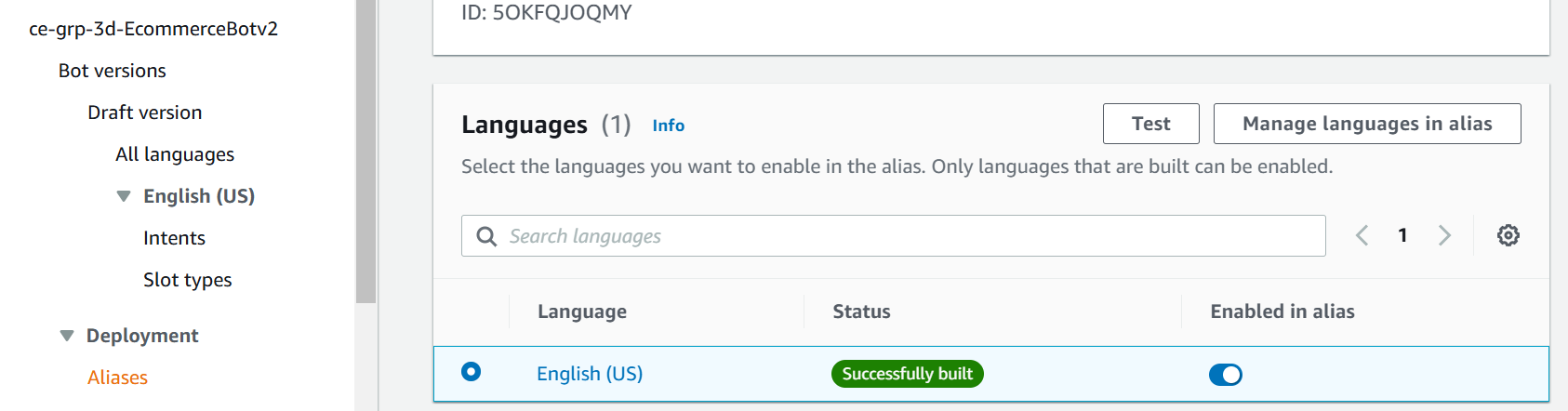




1. **Testing the chatbot**

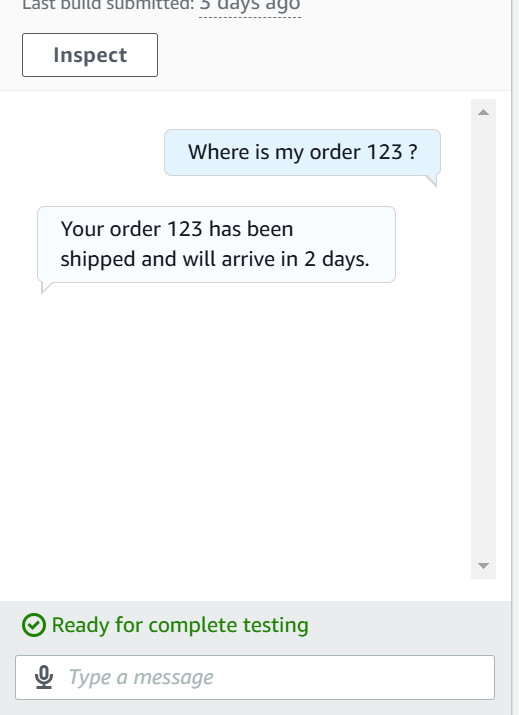
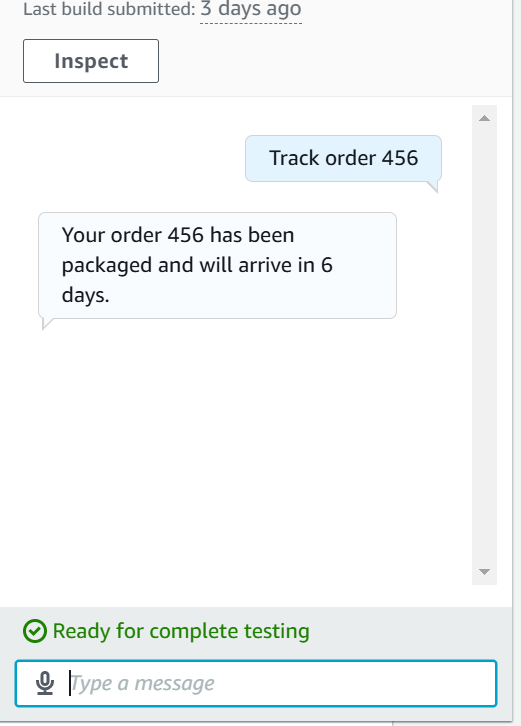
1. Stage 1: Test the code through Amazon Lex to test business logic provided by Lambda

* Go to AWS Lex and select bot created (e.g. *ce-grp-3d-EcommerceBotv2*)
* Go to Deployment->Alias & select created Alia (e.g. *PROD*)
* Select Language and click ‘**Test**’ per below screenshot.

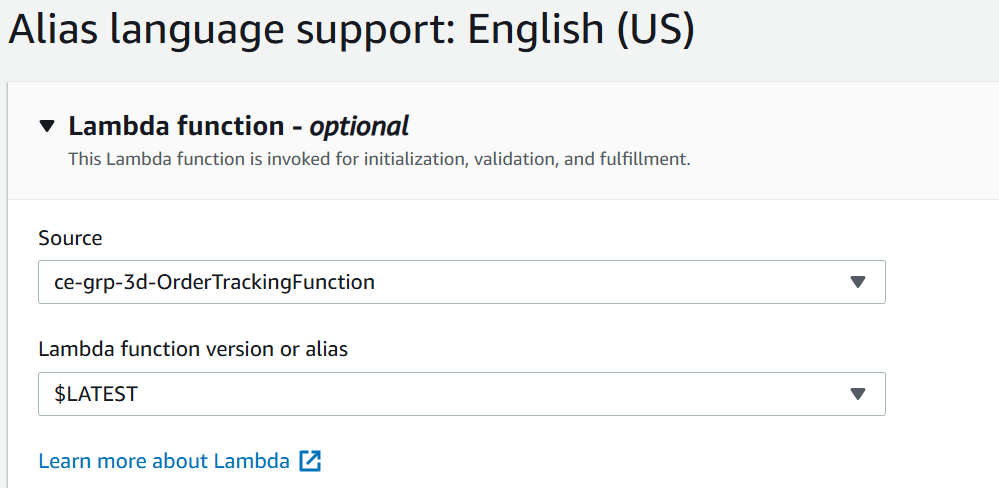


* 1. This leads to the chatbot being displayed and tests can be done using the utterances defined in the chatbot’s intent. e.g. “Where is order 123 ?”.
* The expected result is that the order status is retrieved by the lambda function associated with the Bot’s Alias in this case by Lambda function *–*

*ce-grp-3d-OrderTrackingFunction*

* In case desired output is not received, errors can be investigated and debugged by clicking on the ‘Inspect’ button on Lex’s Chatbot interface
* Should there be lambda errors faced, lambda code can be fixed and deployed only. There is no need to rebuild the Alias/version of the chatbot as we had linked the alias->language to the latest version of the lambda function – so the latest code will be picked up.



Stage 2: Once the Business Logic provided by Lambda has been tested using AWS Lex, the bot test can be conducted using the html UI (for the website). Below are the expected outcomes:

* Should errors be faced/ desired outcome not received, like the Lex Bot this can be debugged by right-click of the Bot and clicking ‘Inspect’ in the pop-up menu.
* Note, to get the background note that supporting files like images (e.g. ecommerce\_background) should be placed in the same area as the UI file(s)

