

Develop decoupled serverless backend architecture

Before going to develop the architecture, let's understand all the aws services and automation tool that we are using in this serverless architecture.

What is Serverless?

Serverless is a term that generally refers to serverless applications. Serverless applications are ones that don't need any server provision and do not require to manage servers.

A serverless infrastructure based on AWS Lambda has two key benefits:

1. We don't need to manage any virtual machines anymore.
2. Deploying new versions of our can be entirely controlled by API calls

How to process asynchronous tasks serverless:

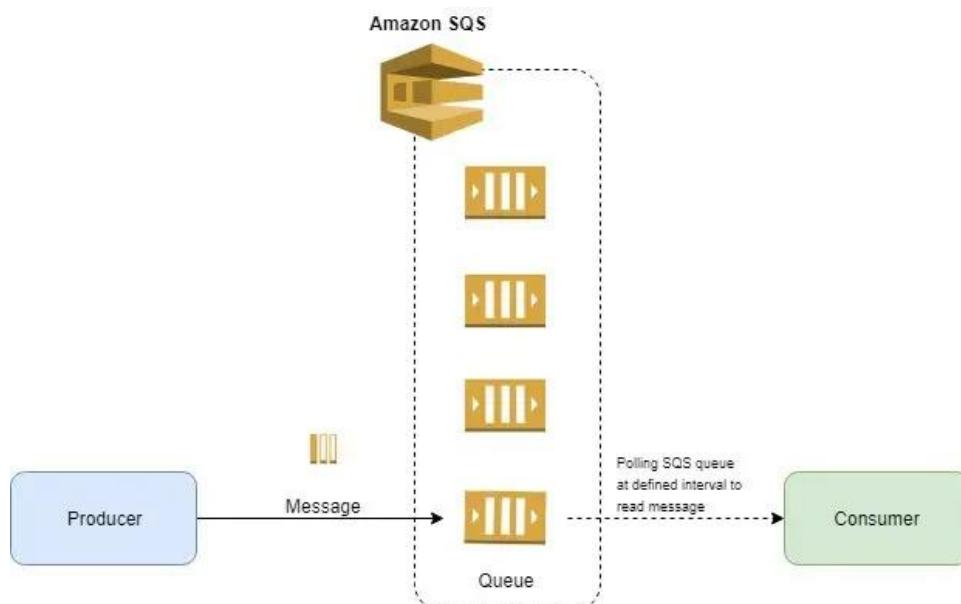
Possible use cases are: sending out massive amount of messages, emails, transcoding videofiles after uploaded, or analyzing user behavior. An SQS will be used to decouple our microservice from other parts of our system.

Firstly,

What is SQS?

Amazon Simple Queue Service (SQS) is a Fully managed message queuing for microservices, distributed systems, and serverless applications. It offers fault tolerant and scalable distributed message queues: simple to use but very powerful.

Asynchronous workflows have always been the primary use case for SQS. Using queues ensure one component can keep running smoothly without losing data when another component is unavailable or slow.



What is Lambda?

Lambda is an event-driven, serverless computing service provided by Amazon. Therefore we don't need to worry about Which AWS resources to launch, or how will you manage them. Instead, we need put the code on lambda, and it runs.

However, Amazon Lambda can only used to execute background tasks.

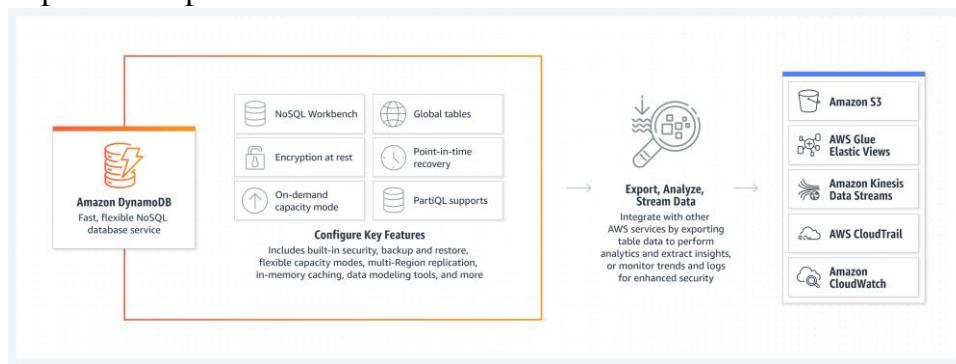
AWS Lambda function helps you to focus on your core product and business logic instead of managing operating system(os) access control, provisioning, etc.

What is DynamoDB?

Fast, flexible NoSQL database service for single-digit millisecond performance at any scale

How it works

Amazon DynamoDB is a fully managed, serverless, key-value NoSQL database designed to run high-performance applications at any scale. DynamoDB offers built-in security, continuous backups, automated multi-Region replication, in-memory caching, and data import and export tools.

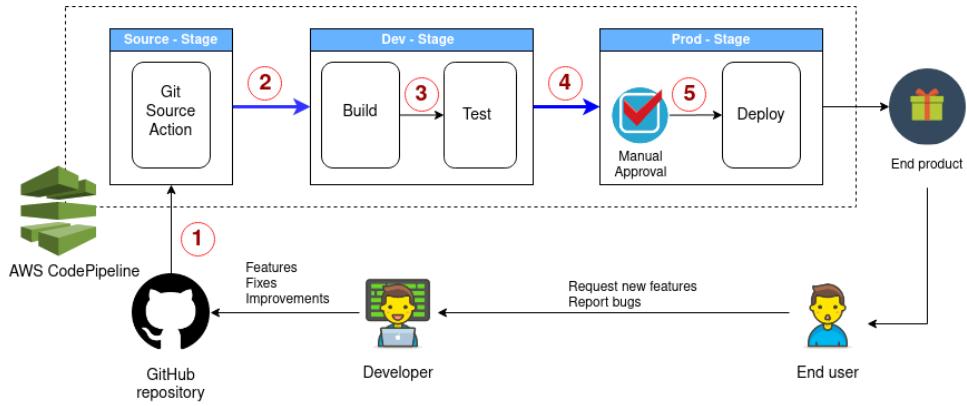


What is AWS CodePipeline?

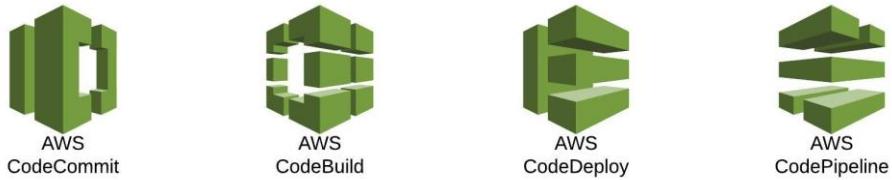
AWS CodePipeline is an Amazon Web Services product that **automates the software deployment process**, allowing developers to quickly model, visualize and deliver code for new features and updates.

AWS CodePipeline automatically builds, tests and launches an application each time the code is changed.

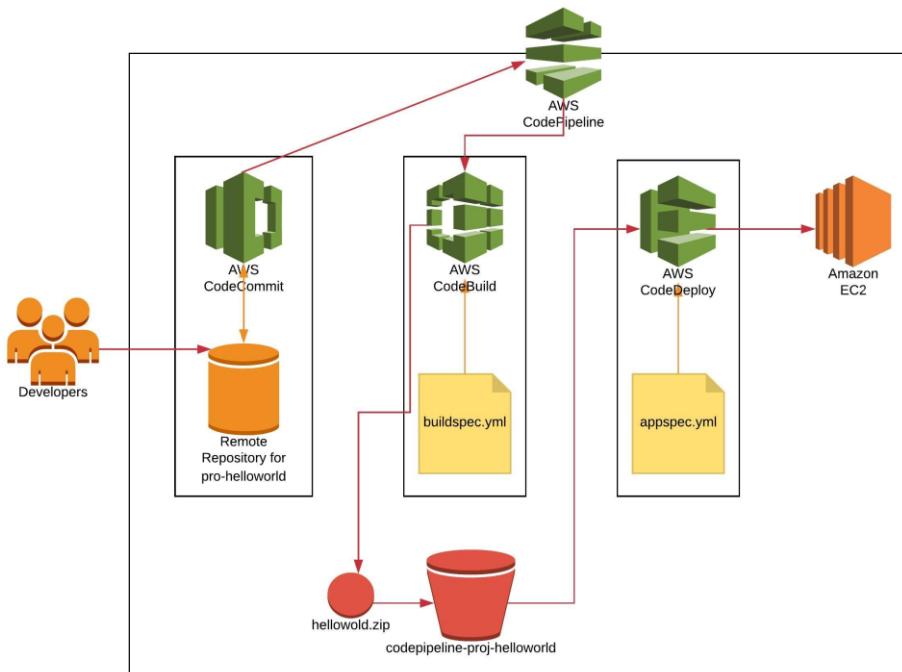
AWS CodePipeline is a continuous delivery service that enables you to model, visualize, and automate the steps required to release your software.



AWS Codepipeline providers

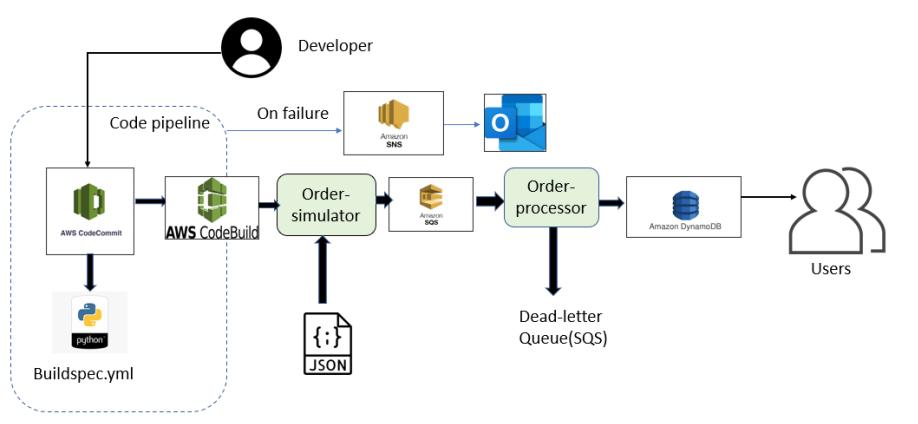


Setup and configuration



Finally After understanding all the basics of needed services to Develop decoupled serverless backend architecture We can start:

Inorder to start, lets understand the below architecture in steps -



Step 1 : Writing two Lambda functions, One Function(Order_simulator) which will create and send the messages to an SQS and second function(Order_processor) which will process all SQS messages and store in DynamoDB and SQS queue to store the badly or invalid data in dead letter queue - poll for the messages to view the exact input given

Dynamo DB to have item records for valid data.

Step 2 : Creating a Pipeline which can automate our Lambda Function

-> We need push our git code repository to code commit which contains buildspec.yaml/json and appspec.yaml/json.

Why buildspec.yaml/json need?

A buildspec is a collection of build commands and related settings, in YAML /JSON format, that codebuild uses to run a build. You can include a buildspec as part of the source code or you can define a buildspec when you create a build project.

Here is the code for buildspec.yaml file, this is for upadating our lambda function, if we creating lambda functions through codebuild, then u need to add create-function commands, u can get these here by uncommenting the lines. We can also invoke the functions by sending the response file for lambda function.

Why appspec.yaml/json need?

The application specification file (AppSpec file) is a YAML -formatted or JSON-formatted file used by CodeDeploy to manage a deployment. The AppSpec file for an EC2/On-Premises deployment must be named appspec. yml or appspec. yaml , unless you are performing a local deployment.

Now we can start the pipeline.

Step 3 : Creating an Simple Notification Service for failures occurs in codepipeline it will sent a notification to our outlook email

To create a notification rule (console)

1 . Sign in to the AWS Management Console and open the CodePipeline console at
<https://console.aws.amazon.com/codepipeline/>.

2 . Choose **Pipelines**, and then choose a **pipeline** where you want to add notifications.

3 . On the pipeline page, choose **Notify**, and then choose **Create notification rule**. You can also go to the Settings page for the pipeline and choose Create notification rule.

4 . In Notification name, enter a name for the rule.

5 . In Targets, do one of the following:

- If you have not configured a resource to use with notifications, choose Create target, and then choose SNS topic. Provide a name for the topic after codestar-notifications-, and then choose Create.
- If you create the Amazon SNS topic as part of creating the notification rule, the policy that allows the notifications feature to publish events to the topic is applied for you. Using a topic created for notification rules helps ensure that you subscribe only those users that you want to receive notifications about this resource.
- If you want to use an existing Amazon SNS topic as a target, you must add the required policy for AWS CodeStar Notifications in addition to any other policies that might exist for that topic.
- Creating a Topic, Go to **SNS console** click on **Topic** -> click on **Create Topic** -> Give **Topic name** -> After click **Create Topic**. Click on **Create Subscription** in the Topic which we created. Then Give a name, select **Email as protocol**. Click on **Submit**.

6 . To finish creating the rule, choose Submit.

7 . You must subscribe users to the Amazon SNS topic for the rule before they can receive notifications.

Step 4 : Create a SQS queue and adding lambda trigger

1. Visit the SQS Console again and click on the queue created earlier:

2. Click on ‘**Lambda Triggers**’ and click on ‘**Configure Lambda function trigger**’ button.

3. Search for the lambda function created earlier by name in the drop down and select it. Click on the ‘**Save**’ button. This would successfully add the lambda trigger to the SQS queue:

4. Now to test our setup, we shall send a message to the queue (as shown earlier in the article). The message must get passed on to the lambda function.
5. Go back to the lambda console and click on the '**Monitor**' tab. Then click on the '**View logs in CloudWatch**' button. This will open a new tab and take you to CloudWatch.
6. In the CloudWatch log group page, you would find a log stream pertaining to the latest execution of the lambda function. Click on that to see the logs :
7. The log of the message object is highlighted below. Expand it to view the entire object. The message we sent will appear inside the 'body' field of the message object.

So at this stage, we have a functional setup where a message sent into SQS gets pushed to a lambda function which in turn logs it into CloudWatch. Now for the final piece of the puzzle, we have to setup a DynamoDB table and write some code to insert data into it.

Step 5 : Create a DynamoDB that can store our SQS valid messages

lets understand the below architecuture,

Here after our order_simulator lambda function sent messages to the SQS , In the Next step we need add our order_processor lambda function as a trigger to the SQS .

Then Whenever The messeges added to our SQS Queue it trigger the order_processor function, then the function will trigger and push all the valid messages to the DynamoDB. Duplications or invalid messages added to the dead letter queue.

Here we can add Trigger in two ways -

1. By Going to SQS console
2. By adding triggers in Lambda function.

Here we are adding trigger by going to SQS console

Go to the [DynamoDB console](#) and click on '**Create Table**':

2. Provide a table name and a partition key as shown below. Leave the other settings as they are by default. Scroll to the bottom of the page and click on '**Create Table**' button :
3. The table would take some time to get created. In the meantime, go back to the Lambda console. Paste the following code inside the editor and click on '**Deploy**' (like we did previously) :

The above code parses the message received from SQS and writes it into the dynamoDB table we created. It also logs some additional information to CloudWatch in case of success as well as error.

Our main Goal is not creating the resources by going to every aws console manually, because we human can make errors and also spending lot of time by visiting every console, we need a configuration as a code which can automate everything regarding our serverless infrastructure.

So Let's get Start,

Terraform for Automating the AWS Serverless Infrastructure

So Far we creating everything manually, inorder to reduce the headache of creating iam roles, policies, sqs and dynamodb by visiting every aws console, its take a lot of time.

So for that, Now we are automating everything by using automation tools to reduce. Here I am using terraform.

Let's understand what is terraform and why we are so excited to use.

Without any further , let's get started.

So let us start with creating the infrastructure required by the pipeline. We will make use of Terraform which is an open-source infrastructure as a code (IaaC)software tool.

Why infrastructure as a code?

It is fair to ask the question why to make use of infrastructure as code when we could make use of AWS services using the console? The reason is simple,

- Firstly, let say we develop the pipeline in development environment. Once happy with the pipeline we move the pipeline to test and then to prod environment. Imagine we make use of 100s of services, then we would have to recreate all those services manually first in test and then in development environment. This is a tedious and time consuming task.
- Secondly, with 100's of services, that needs to be created multiple time, there is a fair bit chance of errors that can happen and would be very difficult to debug when done manually.
- Finally, if we create services manually we can not use version control techniques i.e using Git. Infrastructure as a code (Iaac) is quite helpful in that aspect.

Note : We make use of terraform to implement infrastructure, however it is not the only tool out there in the market. Since we are using AWS we could have use AWS Cloudformation another IaaC service by AWS. I am use terraform due to its multi-cloud integration capabilities. So now let's understand briefly about terraform.

What is Terraform?

Terraform is an infrastructure as code tool that lets you build, change, and version cloud and on- prem resources safely and efficiently.

How does Terraform work?

Terraform creates and manages resources on cloud platforms and other services through their application programming interfaces (APIs). Providers enable Terraform to work with virtually any platform or service with an accessible API.

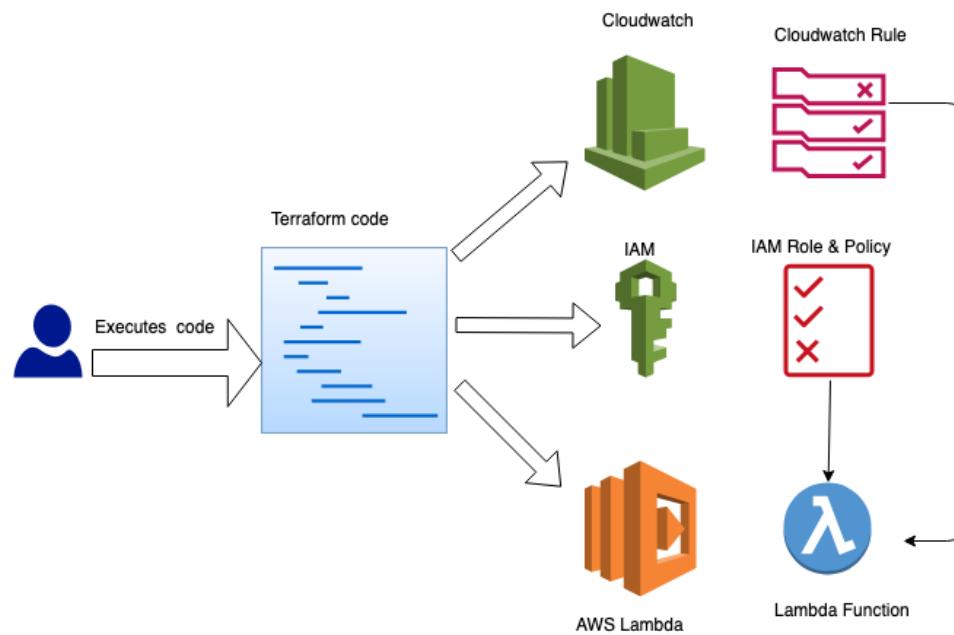
The focus is on automating as much as possible to reduce human errors and provide an efficient way. Although we would look at the pipeline in AWS, similar pipeline can be built any any cloud platform (including Google Cloud platform, Azure, etc)as they have very similar services.

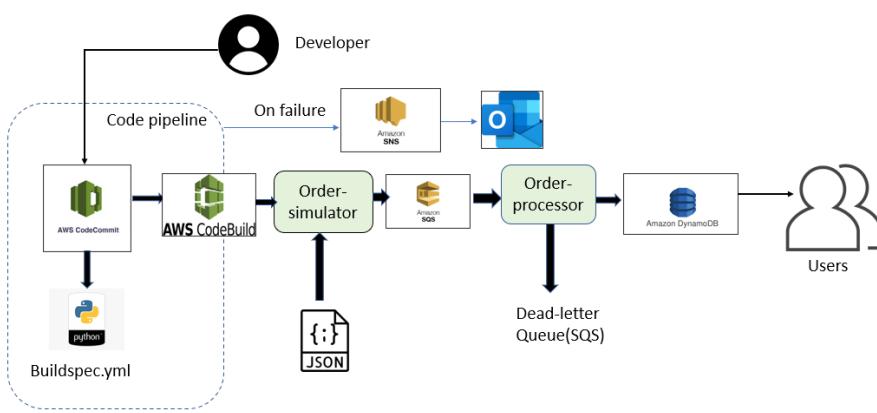
Why Terraform?

1. Manage any infrastructure
2. Track your infrastructure
3. Automate changes
4. Standardize configurations
- Collaborate

Furthermore, we would make use of Terraform to create the infrastructure as code. This helps in reducing the possibility of errors when moving the model from say development to production.

Let us start building the infrastructure





So Lets Start by installing terraform

Step 1 : Install the terraform.

```

$ wget https://releases.hashicorp.com/terraform/0.13.0/terraform_0.13.0_linux_amd64.zip
$ unzip terraform_0.13.0_linux_amd64.zip
$ mv terraform /usr/local/bin
$ terraform version
Terraform v0.13.0
  
```

Step 2 : create a main.tf configuration file and add aws provider in it.

```

provider "aws" {
  region = "us-east-2"
  # aws_secret_access_key = "9kgxJ1vbFuKz0hcZFFGVkXxuBkM+3YAwZ1FvKe"
  # aws_access_key_id = "AKIA5670WQOMGK3VBCU"
}
  
```

We should never pass the any secrets and access key, so for that configure the aws credentials in locally. Open the terminal, go to the folder and open the config file and paste the access key and secret key like show below.

```

drwxr-xr-x  2 piotrsha piotrsha  4096 Oct 13 15:46 .aws
piotrsha@piotrsha-MacBook-Pro:~/Desktop$ cd .aws/
piotrsha@piotrsha-MacBook-Pro:~/Desktop/.aws$ ls
config  config.pyc  config.save
piotrsha@piotrsha-MacBook-Pro:~/Desktop/.aws$ nano config
piotrsha@piotrsha-MacBook-Pro:~/Desktop/.aws$ 
  
```

Open the aws IAM Console -> click on users -> create a user and after that open security credentials, in that we can see our accesskey, and for secret key it will show only once, download it, then copy the secret and access key paste in terminal.

The screenshot shows the AWS IAM User Details page for a user named 'nirosha'. It displays the User ARN (arn:aws:iam::203978866841:user/nirosha), Path (/), and Creation time (2022-10-16 08:18 UTC+0530). The 'Security credentials' tab is selected, showing a 'Console sign-in link' (https://203978866841.signin.aws.amazon.com/console) and a 'Create access key' button. Below this, it lists two access keys: 'AKIAS67QWjQCMNGW3VBCU' (Created: 2022-10-16 08:18 UTC+0530, Last used: 2022-10-25 09:49 UTC+0530) and 'AKIAS67QWjQCMNGW3VBCU' (Created: 2022-10-16 08:18 UTC+0530, Last used: 2022-10-25 09:49 UTC+0530 with codecommit in us-east-1). The status for both is 'Active'.

The screenshot shows a terminal window on a Linux system (piroshash@USBLRPNHROSHUB1: ~\$aws) running the command 'aws iam create-access-key --user-name nirosha'. The output shows the creation of a new access key with the ID 'AKIAS67QWjQCMNGW3VBCU'.

Step 3 : Create a file which contains all the configurations related to codecommit which should be .tf type. Using aws_codecommit_repository resource it can create a repository for us. Provide a repository name.

The screenshot shows a Visual Studio Code editor with a Terraform configuration file named 'main.tf'. The code defines a 'resource "aws_codecommit_repository" "terraform_repo"' block with the attribute 'repository_name' set to 'terraform-repository'.

```

resource "aws_codecommit_repository" "terraform_repo" {
  repository_name = "terraform-repository"
}

```

Step 6 : Before going to next step,we need create and setup a Service role that can allow the build to execute all services such as lambda,sqs, dynamobd,deploy, cloudwatch for logs and all the required iam permissions. We add json format policies and also we can direct add the services which allows all the permissions of the policies.

The screenshot shows a Visual Studio Code editor with a Terraform configuration file named 'iam-roles-policies.tf'. The code defines a 'resource "aws_iam_role" "servicerole-codebuild"' block with an 'assume_role_policy' block containing a single statement that allows the service principal 'codebuild.amazonaws.com' to assume the role.

```

resource "aws_iam_role" "servicerole-codebuild" {
  name = "servicerole-codebuild"
  assume_role_policy = <>EOF
  {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": "codebuild.amazonaws.com",
          "Service": "events.amazonaws.com",
          "Service": "lambda.amazonaws.com",
          "Service": "codecommit.amazonaws.com",
          "Service": "codebuild.amazonaws.com",
          "Service": "codepipeline.amazonaws.com",
          "Service": "codedeploy.amazonaws.com",
          "Service": "sns.amazonaws.com",
          "Service": "sqs.amazonaws.com",
          "Service": "dynamodb.amazonaws.com"
        }
      }
    ]
  }
}

```

Step 5 : Next Create a file for Codebuild. Provide all the neccesary attributes such as build name, artifacts, source and service role that contain required iam permissions, make sure all the permissions are provided are not. Below is the code for my codebuild.

```

Oct 25 13:01
terraform-codebuild.tf - automate-tf - Visual Studio Code
Help
main.tf  terraform-codedeploy.tf  terraform-codepipeline.tf  terraform-codebuild.tf  appspec.yml  terraform-codecommit.tf  terraform-s3.tf  role.json
1 //Code Build.....
2
3 resource "aws_codebuild_project" "terraform-codebuild" {
4   name          = "terraform-codebuild"
5   description   = "terraform_codebuild_project"
6   build_timeout = "5"
7   #service_role = aws_iam_role.servicerole-codebuild
8   service_role  = aws_iam_role.servicerole-codebuild.arn
9
10  artifacts {
11    type = "CODEPIPELINE"
12  }
13
14  cache {
15    type    = "S3"
16    location = aws_s3_bucket.terraform-codepipeline-s3.bucket
17  }
18
19  environment {
20    compute_type      = "BUILD_GENERAL_SMALL"
21    image             = "aws/codebuild/standard:1.0"
22    type              = "LINUX_CONTAINER"
23    image_pull_credentials_type = "CODEBUILD"
24  }
25
26  logs_config {
27    cloudwatch_logs {
28      group_name = "log-group"
29      stream_name = "log-stream"
30    }
31
32    s3_logs {
33      status  = "ENABLED"
34      location = $(aws s3 bucket.terraform-codepipeline-s3.id)/build-log
35    }
36  }
37
38 source {
39   type     = "CODEPIPELINE"
40   location = "https://git-codecommit.us-east-2.amazonaws.com/v1/repos/terraform_repository"
41   git_clone_depth = 1
42
43   git_submodules_config {
44     fetch_submodules = true
45   }
46
47   source_version = "refs/heads/master"
48
49   tags = [
50     Environment = "Test"
51   ]
52 }
53
54 resource "aws_s3_bucket" "terraform-codepipeline-s3" {
55   bucket = "terraform-codepipeline-s3"
56 }
57
58 resource "aws_s3_bucket_acl" "terraform_codepipeline-s3" {
59   bucket = aws_s3_bucket.terraform-codepipeline-s3.id
60   acl    = "private"
61 }
62

```



```

Oct 25 13:02
terraform-codebuild.tf - automate-tf - Visual Studio Code
Help
main.tf  terraform-codedeploy.tf  terraform-codepipeline.tf  terraform-codebuild.tf  appspec.yml  terraform-codecommit.tf  terraform-s3.tf  role.json
1
2
3 source {
4   type     = "CODEPIPELINE"
5   location = "https://git-codecommit.us-east-2.amazonaws.com/v1/repos/terraform_repository"
6   git_clone_depth = 1
7
8   git_submodules_config {
9     fetch_submodules = true
10  }
11
12   source_version = "refs/heads/master"
13
14   tags = [
15     Environment = "Test"
16   ]
17 }
18
19 resource "aws_s3_bucket" "terraform-codepipeline-s3" {
20   bucket = "terraform-codepipeline-s3"
21 }
22
23 resource "aws_s3_bucket_acl" "terraform_codepipeline-s3" {
24   bucket = aws_s3_bucket.terraform-codepipeline-s3.id
25   acl    = "private"
26 }
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62

```

Step 6 : Create a another file which contains all the configurations related to code deploy. Provide all the required attribute fileds. Next For Code Deploy we need to create a Deployment_group also with deployment_style, make sure providing a correct lambda deployment_style which is blue/green by providing required permissions.

```

Oct 25 13:01
terraform-codedeploy.tf - automate-tf - Visual Studio Code
Help
main.tf  terraform-codedeploy.tf  terraform-codepipeline.tf  terraform-codebuild.tf  appspec.yml  terraform-codecommit.tf  terraform-s3.tf  role.json
1
2
3 resource "aws_codedeploy_app" "terraform_codedeploy" {
4   compute_platform = "Lambda"
5   name            = "terraform-codedeploy"
6 }
7
8 //DEPLOYMENT GROUP....
9 resource "aws_codedeploy_deployment_group" "terraform_deployment_group" {
10   app_name           = aws_codedeploy_app.terraform_codedeploy.name
11   deployment_group_name = "terraform_deployment_group"
12   service_role_arn  = aws_iam_role.servicerole-codebuild.arn
13   deployment_config_name = "CodeDeployDefault.LambdaAllAtOnce"
14
15   deployment_style {
16     deployment_option = "WITH_TRAFFIC_CONTROL"
17     deployment_type   = "BLUE_GREEN"
18   }
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
187
188
189
189
190
191
192
193
194
195
196
197
197
198
199
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
235
236
237
237
238
239
239
240
241
242
243
244
245
245
246
247
247
248
249
249
250
251
252
253
254
255
255
256
257
257
258
259
259
260
261
262
263
264
265
265
266
267
267
268
269
269
270
271
272
273
274
275
275
276
277
277
278
279
279
280
281
282
283
284
285
285
286
287
287
288
289
289
290
291
292
293
294
295
295
296
297
297
298
299
299
300
301
302
303
304
304
305
306
306
307
308
308
309
309
310
311
311
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1
```

```

Oct 25 13:44
terraform-codepipeline.tf - automate-tf - Visual Studio Code

 terraform-codepipeline.tf x  terraform-codedeploy.tf  terraform-codebuild.tf  appspec.yaml  terraform-codecommit.tf  terraform-sns.tf  rule.json  terraform-sns  ...
 terraform_repository > terraform-codepipeline.tf > resource "aws_codepipeline" "terraform-codepipeline" > stage > name
 61 resource "aws_codepipeline" "terraform-codepipeline" {
 62   name      = "terraform-codepipeline"
 63   role_arn  = aws_iam_role.servicerole-codebuild.arn
 64
 65   artifact_store {
 66     location = aws_s3_bucket.terraform-codepipeline-s3.bucket
 67     type      = "S3"
 68   }
 69
 70   stage {
 71     name = "CodeCommit"
 72     action {
 73       name      = "Source"
 74       category  = "Source"
 75       provider  = "CodeCommit"
 76       version   = "1"
 77       owner     = "AWS"
 78       output_artifacts = ["SourceArtifact"]
 79       configuration = {
 80         RepositoryName = aws_codecommit_repository.terraform_repo.repository_name
 81         BranchName    = "master"
 82       }
 83     }
 84   }

```

Adding Build Stage with the project name that we created previous in configuration section.

```

Oct 25 13:44
terraform-codepipeline.tf - automate-tf - Visual Studio Code

 terraform-codepipeline.tf x  terraform-codedeploy.tf  terraform-codebuild.tf  appspec.yaml  terraform-codecommit.tf  terraform-sns.tf  rule.json  terraform-sns  ...
 terraform_repository > terraform-codepipeline.tf > resource "aws_codepipeline" "terraform-codepipeline" > stage > name
 85
 86   stage {
 87     name = "CodeBuild"
 88     action {
 89       name      = "Build"
 90       category  = "Build"
 91       owner     = "AWS"
 92       provider  = "CodeBuild"
 93       input_artifacts = ["SourceArtifact"]
 94       output_artifacts = ["BuildArtifact"]
 95       version   = "1"
 96
 97       configuration = {
 98         ProjectName = aws_codebuild_project.terraform_codebuild.name
 99       }
100   }
101 }

```

Adding the Deploy stage with the Application and deployment_group that we created previous in codedeploy configuration file.

```

Oct 25 13:44
terraform-codepipeline.tf - automate-tf - Visual Studio Code

 terraform-codepipeline.tf x  terraform-codedeploy.tf  terraform-codebuild.tf  appspec.yaml  terraform-codecommit.tf  terraform-sns.tf  rule.json  terraform-sns  ...
 terraform_repository > terraform-codepipeline.tf > resource "aws_codepipeline" "terraform-codepipeline" > stage > name
102
103   stage {
104     name = "CodeDeploy"
105
106     action {
107       name      = "Deploy"
108       category  = "Deploy"
109       owner     = "AWS"
110       provider  = "CodeDeploy"
111       input_artifacts = ["BuildArtifact"]
112       version   = "1"
113
114       configuration = {
115         ApplicationName = aws_codedeploy_app.terraform_codedeploy.name
116         DeploymentGroupName = aws_codedeploy_deployment_group.terraform_deployment_group.deployment_group_name
117         # AppSpecTemplateArtifact = "BuildArtifact"
118         # AppSpecTemplatePath = "appspec.yaml"
119       }
120     }
121   }
122 }

```

Before going to console and start the pipeline we need to create a **buildspec.yaml** file for codebuild. Here is my buildspec.yaml file.

```

version: 0.2
phases:
  install:
    commands:
      - curl -sS https://releases.hashicorp.com/terraform/0.13.0/terraform_0.13.0_linux_amd64.zip > terraform.zip
      - unzip terraform.zip -d /usr/bin/
      - chmod +x /usr/bin/terraform
  pre_build:
    commands:
      - echo Entered into the pre_build phase...
  build:
    commands:
      - terraform -version
      - terraform init
      - terraform plan
      - terraform apply --auto-approve
      - echo "COMPLETED"
  artifacts:
    files:
      - appspec.yaml
      - build/*

```

And also need to **appspec.yaml** file codedeploy is required. We will talk about this later, for now just add **2 stages** that are codecommit and codebuild.

So, Now go to the Console And check all the resources are created or not with all the configurations the we specified

Codecommit repository.

Name	Description	Last modified	Clone URL
terraform_repository	-	1 minute ago	HTTPS SSH HTTPS (GRC)
product-expo	-	3 days ago	HTTPS SSH HTTPS (GRC)

Codebuild project

Name	Source provider	Repository	Latest build status	Description	Last Modified
terraform_codebuild	AWS CodePipeline	-	In progress	terraform_codebuild_projet	3 hours ago
product-expo_codebuild	AWS CodeCommit	product-expo	Succeeded	-	9 days ago

CodeDeploy Application,

Note: Don't add Codedeploy in the pipeline as a stage for now, we can configure this later, but we see now it will create a deployment group and application.

The screenshot shows the AWS CodeDeploy Applications page. It lists two applications:

- serverless-automation-app**: Compute platform AWS Lambda, Created 6 days ago.
- terraform-codedeploy**: Compute platform AWS Lambda, Created 23 hours ago.

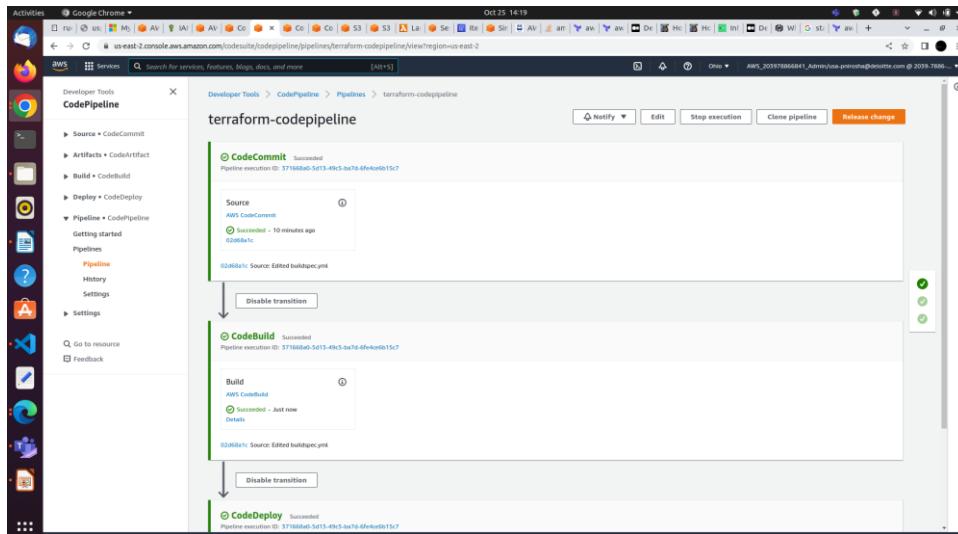
We can also check the IAM roles and policies that will be created by terraform.

The screenshot shows the AWS IAM Roles page. It displays a single role named "servicerole-codedbuild". The role has one trusted entity: "AWS Service: sns, and more".

The permissions section shows 10 managed policies assigned to the role:

- AWSCodeCommitFullAccess
- AmazonSQSFullAccess
- IAMFullAccess
- AmazonS3FullAccess
- CloudWatchFullAccess
- AmazonDynamoDBFullAccess
- AWSCodeDeployFullAccess
- AWSCodeBuildAdminAccess
- AWSCodePipelineFullAccess
- AWSLambdaFullAccess

Now we can start the pipeline. Before that make sure the buildspec.yaml file is there.



Step 8 : Before Creating a lambda functions, create a IAM role and add required policies to the lambda which will allows the lambda function to communicate other resources. So lets create a file, that it contain all the iam configuration for lamda function.

```

resource "aws_iam_role" "lambda_role" {
  name = "terraform_Lambda_Function_Role"
  assume_role_policy = <EOF>
  Version: "2012-10-17",
  Statement: [
    {
      Action: "sts:AssumeRole",
      Principal: "*",
      Service: "lambda.amazonaws.com"
    }
  ]
  Effect: "Allow",
  Sid: ""
}

resource "aws_iam_role_policy_attachment" "multiple_iam_policies_attachment" {
  for_each = toset([
    # Works with AWS Provided policies too!
    "arn:aws:iam::aws-policy:AmazonSFullAccess",
    "arn:aws:iam::aws-policy:AmazonSQSFullAccess",
    "arn:aws:iam::aws-policy:AmazonDynamoDBFullAccess",
    "arn:aws:iam::aws-policy:AmazonCloudWatchLogsFullAccess",
    "arn:aws:iam::aws-policy:AmazonCodeBuildAdminAccess",
    "arn:aws:iam::aws-policy:AWSCodeDeployFullAccess",
    "arn:aws:iam::aws-policy:AWSCodePipelineFullAccess",
    "arn:aws:iam::aws-policy:CloudWatchFullAccess",
    "arn:aws:iam::aws-policy:IAMFullAccess",
    "arn:aws:iam::aws:policy/AWSLambda_FullAccess"
  ])
  role     = aws_iam_role.lambda_role.name
  policy_arn = each.value
}

```

Step 9 : Create a configuration file that contains all the required configuration that can create a lambda function. Before that we need to zip our lambda function , for this i am using python3.8 for function code

Before that, here i have TWO lambda functions one is order_simulator.py and other one order_processor.py

```

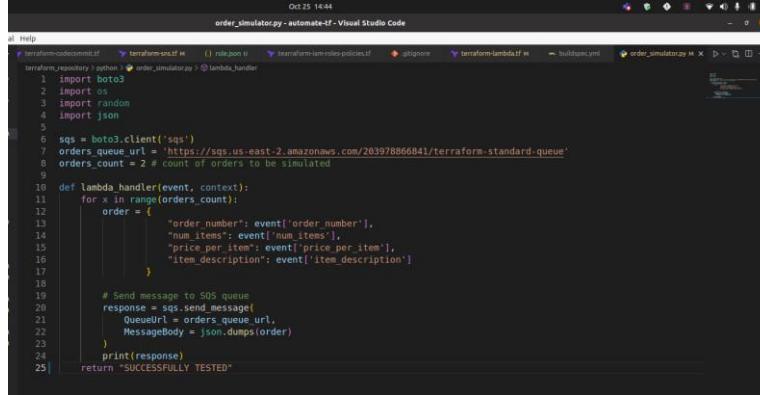
resource "aws_lambda_function" "Terraform_order_processor" {
  filename        = "$path.module/python/order_processor.zip"
  function_name   = "Terraform_order_processor"
  role           = aws_iam_role.lambda_role.arn
  handler        = "order_processor.lambda_handler"
  runtime         = "python3.8"
  timeout        = "10"
  depends_on     = [aws_iam_role_policy_attachment.multiple_iam_policies_attachment]
}

resource "aws_lambda_alias" "Terraform_order_simulator_lambda_alias" {
  name          = "alias"
  function_name = aws_lambda_function.Terraform_order_simulator.arn
  function_version = "1"
}

```

Let's understand what all these function doing.

For order_simulator lambda functon – it will sends all our messages to the SQS with specified order. We need to mention QUEUE_URL...

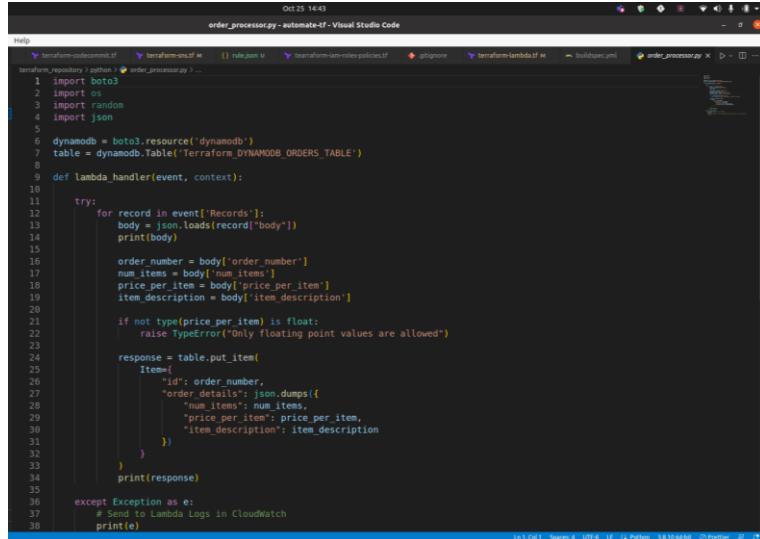


```
Oct 25 14:44
order_simulator.py - automate-tf - Visual Studio Code

Help
terraform-codecommit.tf  terraform-sqs.tf  () rule.json  terraform-lam-moles-policies.tf  .gitignore  terraform-lambda.tf  buildspec.yml  order_simulator.py  D  E  I  M  O  P  S  T  V  X  Z

1 terraform_repository python order_simulator.py lambda_handler
2
3 import boto3
4 import random
5 import json
6
7 sqs = boto3.client('sqs')
8 orders_queue_url = "https://sqs.us-east-2.amazonaws.com/203978866841/terraform-standard-queue"
9 orders_count = 2 # count of orders to be simulated
10
11 def lambda_handler(event, context):
12     for x in range(orders_count):
13         order = {
14             "order_number": event['order_number'],
15             "num_items": event['num_items'],
16             "price_per_item": event['price_per_item'],
17             "item_description": event['item_description']
18         }
19
20         # Send message to SQS queue
21         response = sqs.send_message(
22             QueueUrl = orders_queue_url,
23             MessageBody = json.dumps(order)
24         )
25         print(response)
26
27 return "SUCCESSFULLY TESTED"
```

order_processor lamda function – it will process the messages that are stored in sqs and it sends to all the valid messages to the Dynamodb

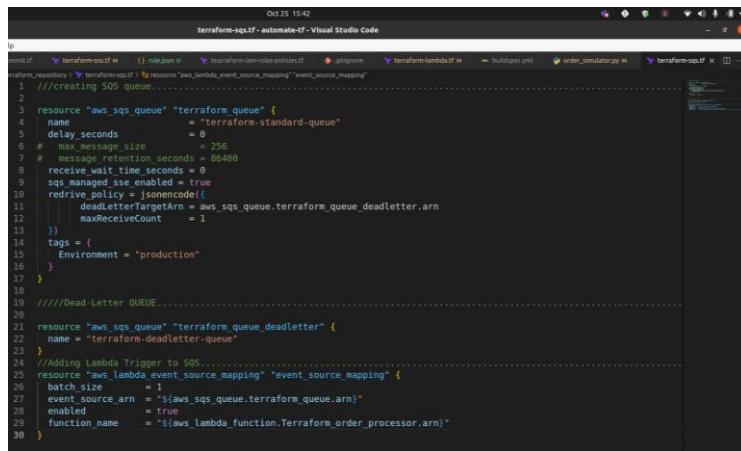


```
Oct 25 14:49
order_processor.py - automate-tf - Visual Studio Code

Help
terraform-codecommit.tf  terraform-sqs.tf  () rule.json  terraform-lam-moles-policies.tf  .gitignore  terraform-lambda.tf  buildspec.yml  order_processor.py  D  E  I  M  O  P  S  T  V  X  Z

1 import boto3
2 import os
3 import random
4 import json
5
6 dynamodb = boto3.resource('dynamodb')
7 table = dynamodb.Table('Terraform.DYNAMODB_ORDERS_TABLE')
8
9 def lambda_handler(event, context):
10     try:
11         for record in event['Records']:
12             body = json.loads(record["body"])
13             print(body)
14
15             order_number = body['order_number']
16             num_items = body['num_items']
17             price_per_item = body['price_per_item']
18             item_description = body['item_description']
19
20             if not type(price_per_item) is float:
21                 raise TypeError("Only floating point values are allowed")
22
23             response = table.put_item(
24                 Item={
25                     "id": order_number,
26                     "order_details": json.dumps({
27                         "num_items": num_items,
28                         "price_per_item": price_per_item,
29                         "item_description": item_description
30                     })
31                 }
32             )
33             print(response)
34
35     except Exception as e:
36         # Send to Lambda Logs in CloudWatch
37         print(e)
```

Step 10 : So Lets create an SQS queue and dead_letter_queue for storing invalid or bad data. For that create a configuration file which contians the sqs resource configuration.



```
Oct 25 15:42
terraform-sqs - automate-tf - Visual Studio Code

Help
terraform-codecommit.tf  terraform-sqs.tf  () rule.json  terraform-lam-moles-policies.tf  .gitignore  terraform-lambda.tf  buildspec.yml  order_simulator.py  terraform-sqs.tf  D  E  I  M  O  P  S  T  V  X  Z

1 //Creating SQS Queue
2
3 resource "aws_sqs_queue" "terraform_queue" {
4     name                  = "terraform-standard-queue"
5     delay_seconds         = 0
6     max_message_size     = 256
7     message_retention_seconds = 86400
8     receive_timeout_seconds = 0
9     sqs_managed_sse_enabled = true
10    redrive_policy = jsonencode({
11        deadletterTargetArn = aws_sqs_queue.terraform_queue.deadletter.arn
12        maxReceiveCount    = 1
13    })
14    tags = {
15        Environment = "production"
16    }
17 }
18
19 //Dead-Letter Queue...
20
21 resource "aws_sqs_queue" "terraform_queue_deadletter" {
22     name = "terraform-deadletter-queue"
23 }
24
25 //Adding Lambda Trigger to SQS...
26 resource "aws_lambda_event_source_mapping" "event_source_mapping" {
27     batch_size      = 1
28     event_source_arn = "${aws_sqs_queue.terraform_queue.arn}"
29     enabled         = true
30     function_name   = "${aws_lambda_function.Terraform_order_processor.arn}"
31 }
```

Now first go to IAM console, Click on roles and our terraform lambda role is created by terraform

Now click on the `terraform_lambda_function_role`, we can see all the policies are attached to the role which we specified in the configuration file.

Now go to lambda console , terraform created two lambda function with s pecified configuration.

Now, Click the simulator function we can see our code.

```

1 import bets
2 import random
3
4 def lambda_handler(event, context):
5     sqs = boto3.client('sqs')
6     queue_url = "https://sqs.us-east-2.amazonaws.com/20397886641/terraform-standard-queue"
7     orders_count = 2 * event['count'] if event['count'] > 0 else 1
8
9     for i in range(orders_count):
10         order_number = event['order_number'] if 'order_number' in event else random.randint(1, 1000)
11         order_qty = random.randint(1, 10)
12         order_per_item = random.randint(1, 100)
13         order_desc = random.choice(['apple', 'banana'])
14
15         # Create a message to SQS
16         response = sqs.send_message(
17             QueueUrl=queue_url,
18             MessageBody=json.dumps(order),
19             MessageAttributes={}
20         )
21
22         print(response)
23
24     return "Successfully TESTED"
25
26

```

Now click alias , we can see our alias function also.

The screenshot shows the AWS Lambda console for the 'Terraform_order_simulator' function. The 'Aliases' tab is active, showing a single alias named 'alias' with a weight of 100%. Other tabs include 'Code', 'Test', 'Monitor', 'Configuration', and 'Versions'.

Now clicking our Terraform_order_processor function we can see that it added the sqs as trigger.

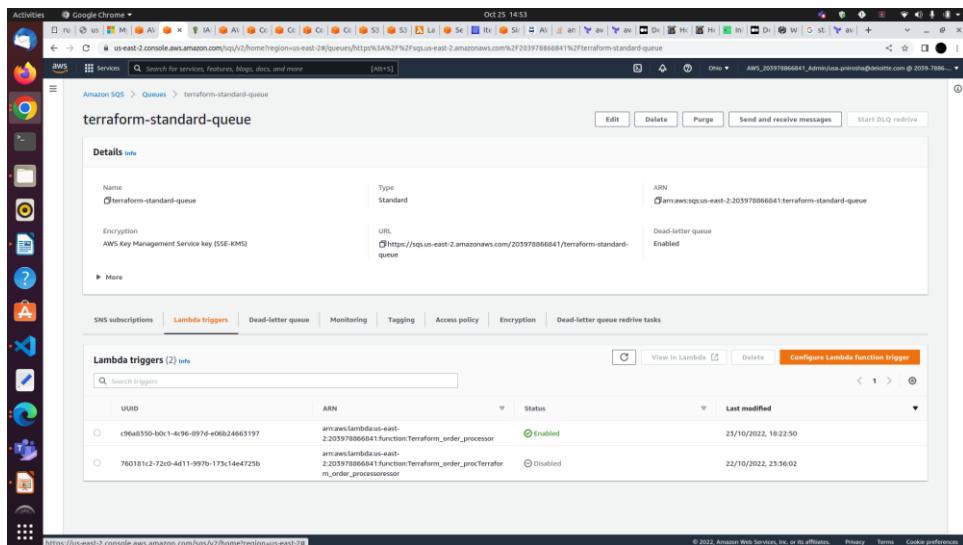
The screenshot shows the AWS Lambda console for the 'Terraform_order_processor' function. The 'Code source' tab is active, displaying the function code in JSON format. The code imports 'batch', 'random', and 'aws_lambda_layer_version' modules, and defines a lambda function named 'order_processor'. The function processes SQS messages and logs them to CloudWatch.

Now, Go to the sqs console, terraform is created two queues for us one for valid data.,

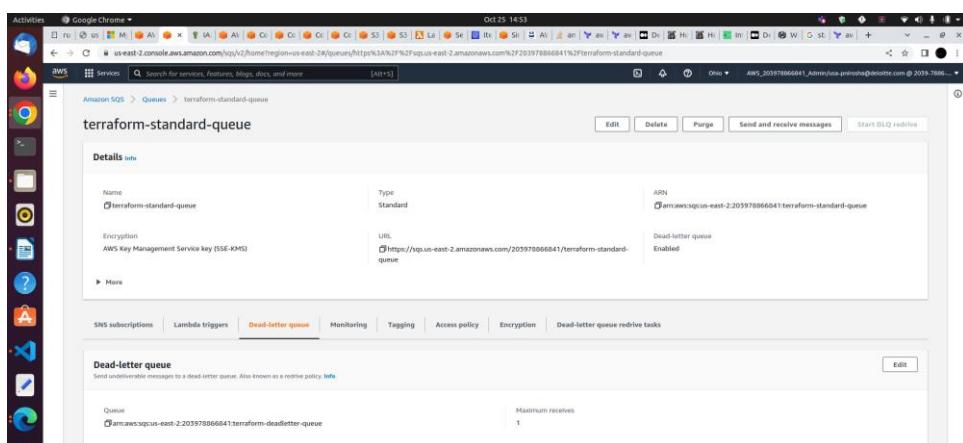
The screenshot shows the Amazon SQS console with four queues listed:

Name	Type	Created	Messages available	Messages in flight	Encryption	Content-based deduplication
SQS-product-expo	Standard	21/10/2022, 12:34:46 GMT+5:30	0	0	AWS Key Management Service key (SSE-KMS)	-
dead-letter-SQS-product-expo	Standard	21/10/2022, 12:52:32 GMT+5:30	17	0	AWS Key Management Service key (SSE-KMS)	-
terraform-deadletter-queue	Standard	23/10/2022, 17:29:33 GMT+5:30	0	0	AWS Key Management Service key (SSE-KMS)	-
terraform-standard-queue	Standard	23/10/2022, 18:22:10 GMT+5:30	0	0	AWS Key Management Service key (SSE-KMS)	-

Now open the terraform_standard_queue, By clicking Lambda triggers we can see that terraform added the our order_processor lambda function



By clicking Dead letter queues, we can see that terraform added the our dead_lead_letter_queue



Step 11 : Now create a file for DynamoDB configuration and add the required attributes such as name attributes rangekey and hashkey..

```
Oct 25 15:57
terraform-dynamodb - automate-tf - Visual Studio Code

Help
File [M]  Editor [M]  Home [M]  terraform-lambda-roles.policies.tf  +  apigateway  terraform-lambda.tfl  terraform-dynamodb_table.yml  order_simulator.py  terraform-ssr.tf  terraform-dynamodb.tf  terraform-dynamodb.x

terraform.repository  terraform-dynamodb.tf  terraform-dynamodb_table.yml  terraform-DYNAMODB_ORDERS_TABLE
1  ///////////////////////////////////////////////////////////////////
2  ///////////////////////////////////////////////////////////////////
3  resource "aws_dynamodb_table" "Terraform_DYNAMODB_ORDERS_TABLE" {
4      name = "Terraform_DYNAMODB_ORDERS_TABLE"
5      billing_mode = "PROVISIONED"
6      read_capacity = "30"
7      write_capacity = "30"
8      range_key       = "order_details"
9      hash_key        = "id"
10
11     attribute {
12         name = "id"
13         type = "N"
14     }
15
16     attribute {
17         name = "order_details"
18         type = "S"
19     }
20 }
21
```

After Going to DynamoDB console, Click Tables, Now we can see our table

The screenshot shows the AWS DynamoDB console interface. A table named 'Terraform_DYNAMODB_ORDERS_TABLE' is displayed. The table has a partition key 'id (Number)' and a sort key 'order_details (String)'. The 'Capacity mode' is set to 'Provisioned' with 100 bytes of capacity. The 'Table status' is 'Active' and there are 'No active alarms'. The 'Overview' tab is selected.

Step 12 : Now Test our lambda Function With valid and invalid input.

For this, Go to **Lambda Console**, -> click **Fucntions** ->

Open **Terraform_order_simulator** -> click **Test** -> Configure test event -> create a new event -> give a name and pass the valid json data -> click **Save** -> Click **Test**

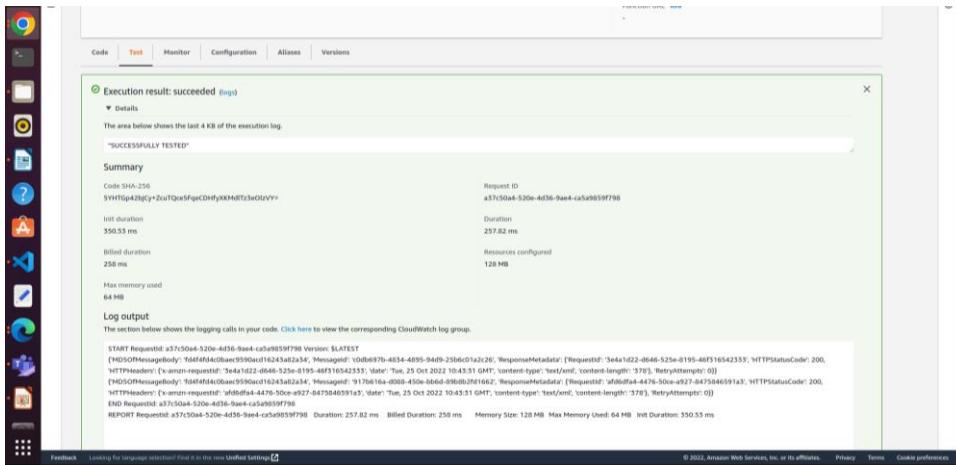
The screenshot shows the AWS Lambda console interface. A test event is configured with the name 'valid-test'. The event JSON is:

```

1: {
2:   "order_number": 1,
3:   "num_items": 2,
4:   "total_weight": 200.15,
5:   "item_description": "Waterbottle: our most popular bottle, available in variety of colours to help brighten up anybody's gear."
6: }

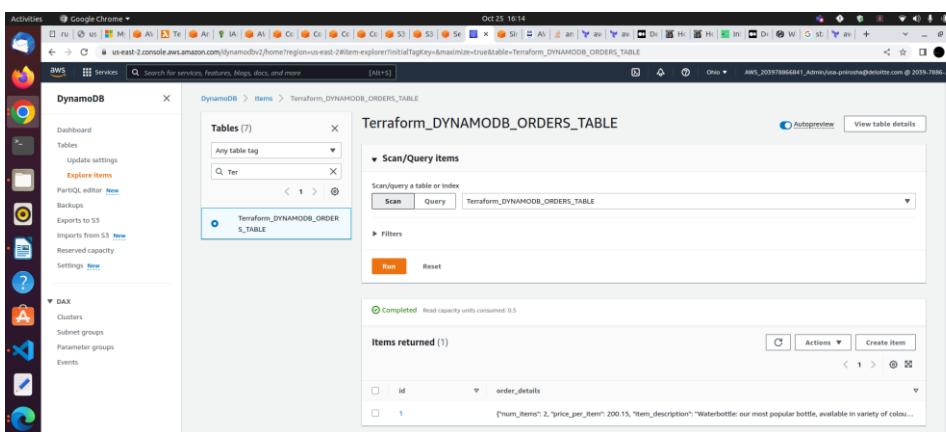
```

After Click on **Test**



If any Failures occurs while testing the function, make sure fuction has required iam policies to excute sqs and Dynamodb, make sure we are passing the correct SQS url and correct Dynamodb Table name in the function python code.

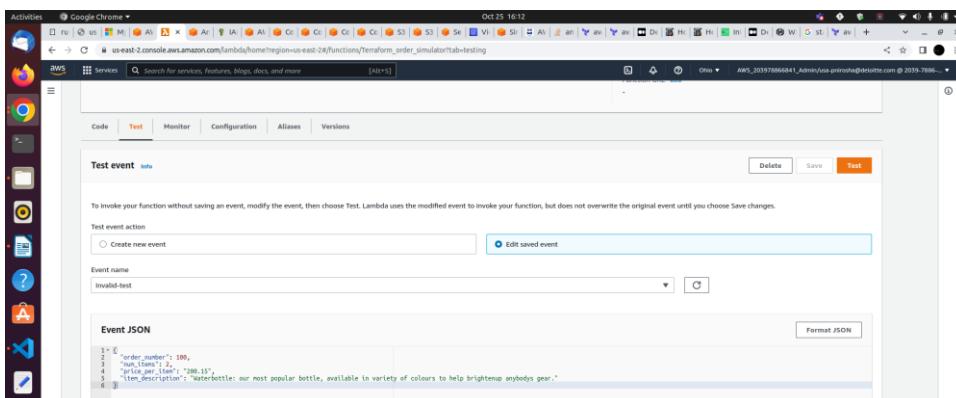
Now Go to **DynamoDB console** -> Click on **Tables** -> click on our table which is **Terraform_DYNAMODB_ORDERS_TABLE**



We can see the our data is stored in dynamodb which is processed by our order_processor lambda function.

Now try to pass the duplicate or invalid data.

Here i am passing the **price as a string format**, which not allowed because we specified price as float value. So let's try



Click on save and Test.

After Executing successfully,

Go to **SQS console** -> click on **Queues** -> click on our dead_letter_queue which is **Terraform-dead-letter-queue** -> Click on **Send and receive messages** -> Now at the bottom we can see poll for messages Click on **poll messages**

If any failures occurs, like not appearing the messages to the dead letter queue, try to increase the latency of standard queue. latency for Dead letter queue is minimum 10. maximum receives parameter had 10 value, So it was processing 10 times.. 11 time if not processed it puts to DLQ. Change the latency to one , then you can see immediate invalid data in dead-letter queue.

The screenshot shows the AWS SQS 'Send and receive messages' interface for the 'terraform-dead-letter-queue'. In the 'Receive messages' section, there are two messages listed:

ID	Sent	Size	Receive count
1f46e367-7b5f-4ff1-a5e9-689f3657eea0	25/10/2022, 16:15:04 GMT+5:30	192 bytes	5
704627fb-c67e-4fe8-97b3-7ebc908a42e1	25/10/2022, 16:15:04 GMT+5:30	192 bytes	2

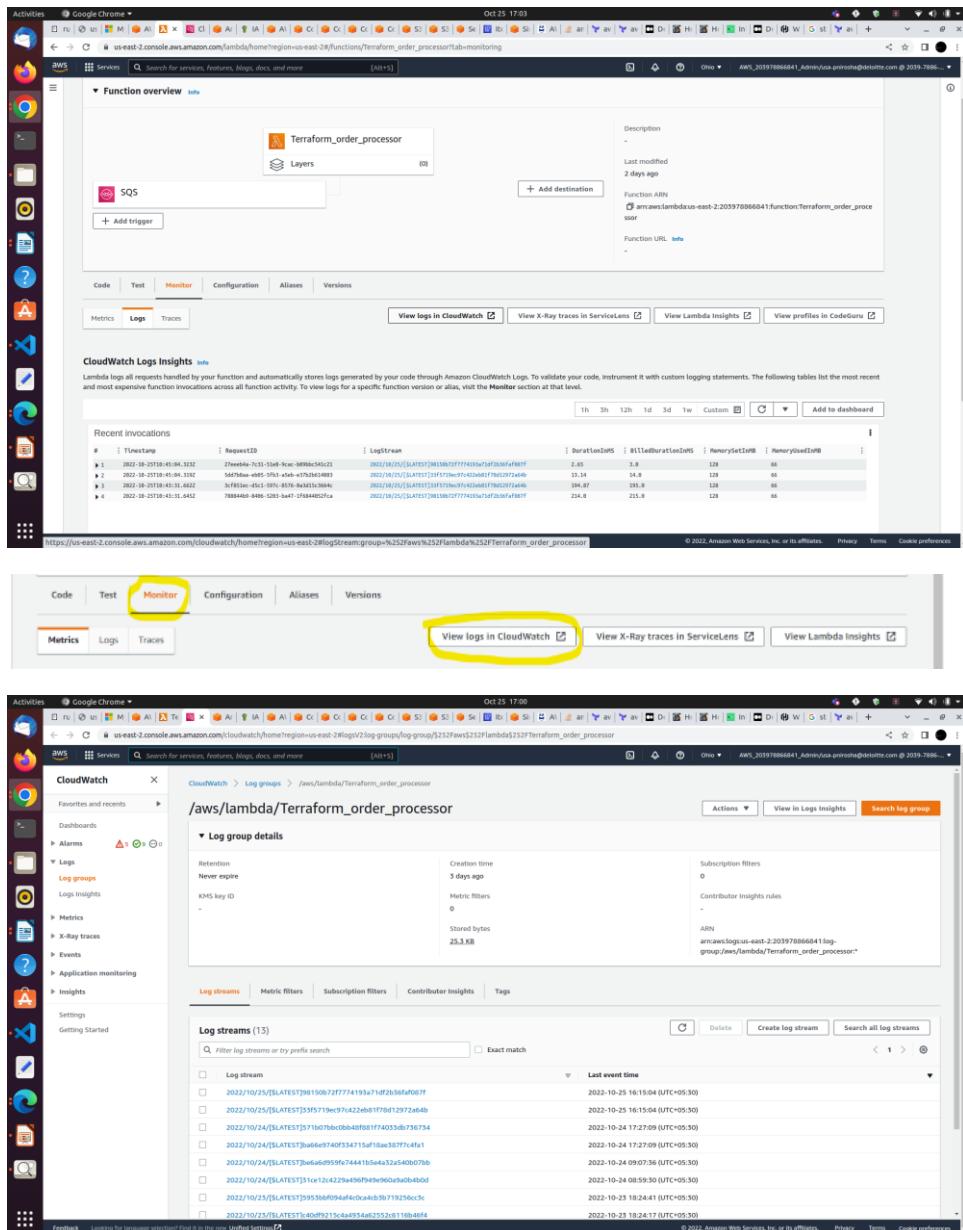
We can see our invalid values are stored. Now click on one messages to see our values. After click on Done.

The screenshot shows the AWS SQS 'Send and receive messages' interface with a message detail modal open. The message ID is 1f46e367-7b5f-4ff1-a5e9-689f3657eea0. The 'Body' tab of the modal shows the following JSON payload:

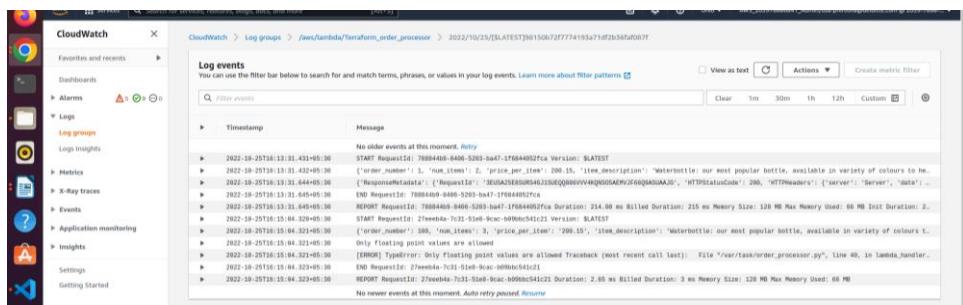
```
{"order_number": 100, "num_items": 3, "price_per_item": "200.15", "item_descriptions": "Waterbottle: our most popular bottle, available in variety of colours to help brighten up anybody's gear"}
```

Also, we could check Cloudwatch logs for the information being logged there:

We can check by directly going to cloudwatch console or By Going back to the lambda console and click on the '**Monitor**' tab. Then click on the '**View logs in CloudWatch**' button. This will open a new tab and take you to CloudWatch.



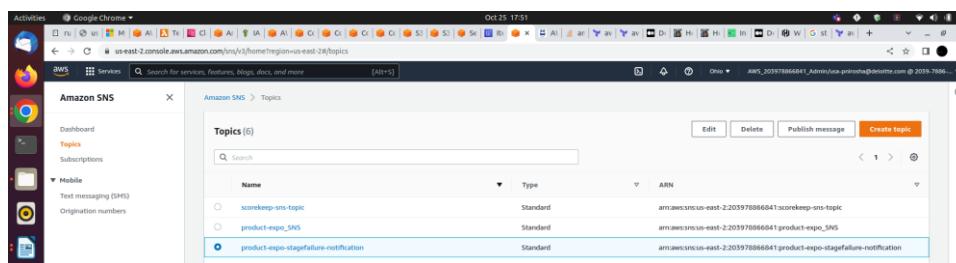
By clicking latest event logs we can see all the things that are proccesed by a order_procesor lambda function.



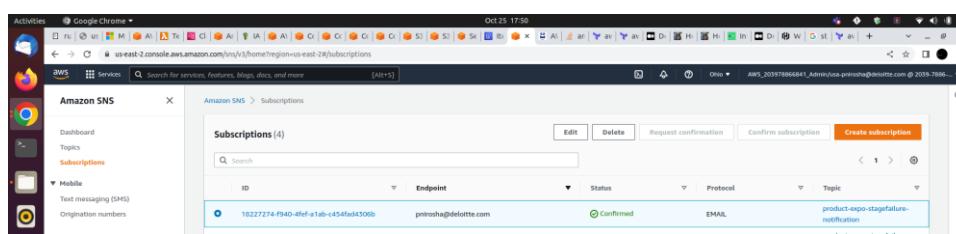
Step 13 : Create a configuration file which will create an SNS Topic and Subscription to the pipeline.

Now Go to **Simple Notification Service(SNS) console** -> Click on Topics

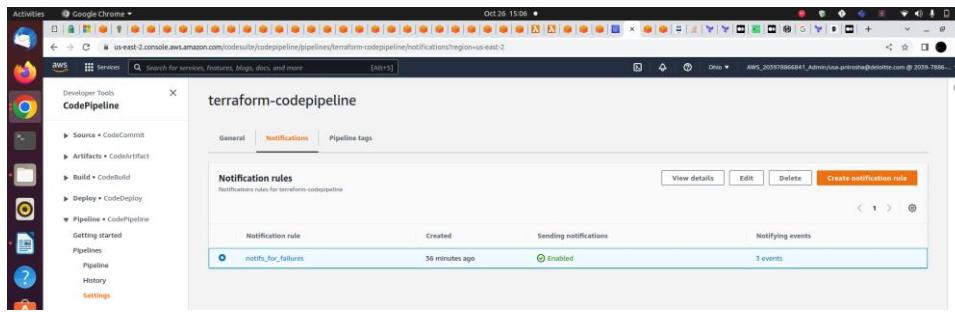
We can see that terraform created an sns topic for us.



Now For subscription, it will send a subscription mail to us, by clicking subscribe in the mail, it will add to the subscription status as confirmed.



Now By visiting our codepipeline, Click on **Notify** -> click on **manage notification rule** -> click on **Notifications** we can a rule that is created. It will send a notifications based on the events that we configure in the terraform file, for this it will sends a notifications to our mail whenever the pipeline execution fails.



Developers can be notified via email if their build fails. It will be useful whenever we are working on teams.

So Finally, We completed the all the things that are in the below serverless architecture.

So what are the usecases of it,

Use cases of Decoupled serverless architecture

Use case 1:

If a team has been given a mandate to recommend an upgrade for the backend architecture to prepare for the upcoming Thanksgiving sale. Historically, our company has seen a 5X traffic spike for e-commerce for the duration of this sale. In the past, there have been cases where the system went down and customer orders were lost, leading to significant revenue loss for the company. We need you to devise a decoupled serverless backend architecture that is able to handle the spike in traffic for the duration of the sale.

This solution, we will leverage an SQS standard queue to send, store, and receive order messages and then configure the SQS Event Source to trigger the Lambda function which in turn processes the orders into a DynamoDB table.

use case 2:

The Engineering team at a company has raised an issue. Every time a developer commits their code to the repository, they have to manually trigger a single EC2 instance-based Jenkins pipeline to build and test the code. Often, the Jenkins server slows down and stops responding, resulting in a lot of time wastage. This approach is a time-consuming task for the developers. They would just like to focus on pushing their application code to the Git repository, and the rest should be taken care of automatically by a more stable pipeline.

By creating an automated pipeline using AWS services that can achieve this. In addition, developers should be notified via email if their build fails.