

Serverless Demo

Event Engine Login

1. Use the following link to access the Dashboard to login

```
https://dashboard.eventengine.run/login?hash=4241-1653e974f4-a6
```

2. Enter Event Hash:

```
Event Hash: 4241-1653e974f4-a6
```

3. Click Accept Terms & Login
4. Follow [Login instructions](#) using One Time Passcode process
5. Next Click on **Open Console** button

AWS Console Login

Remember to only use "us-west-2" as your region, unless otherwise directed by the event operator.

Login Link


 Open Console

 Copy Login Link

Credentials / CLI Snippets

Mac / Linux

Windows

Mac or Linux 

```
export AWS_DEFAULT_REGION=  
export AWS_ACCESS_KEY_ID=  
export AWS_SECRET_ACCESS_  
export AWS_SESSION_TOKEN=
```

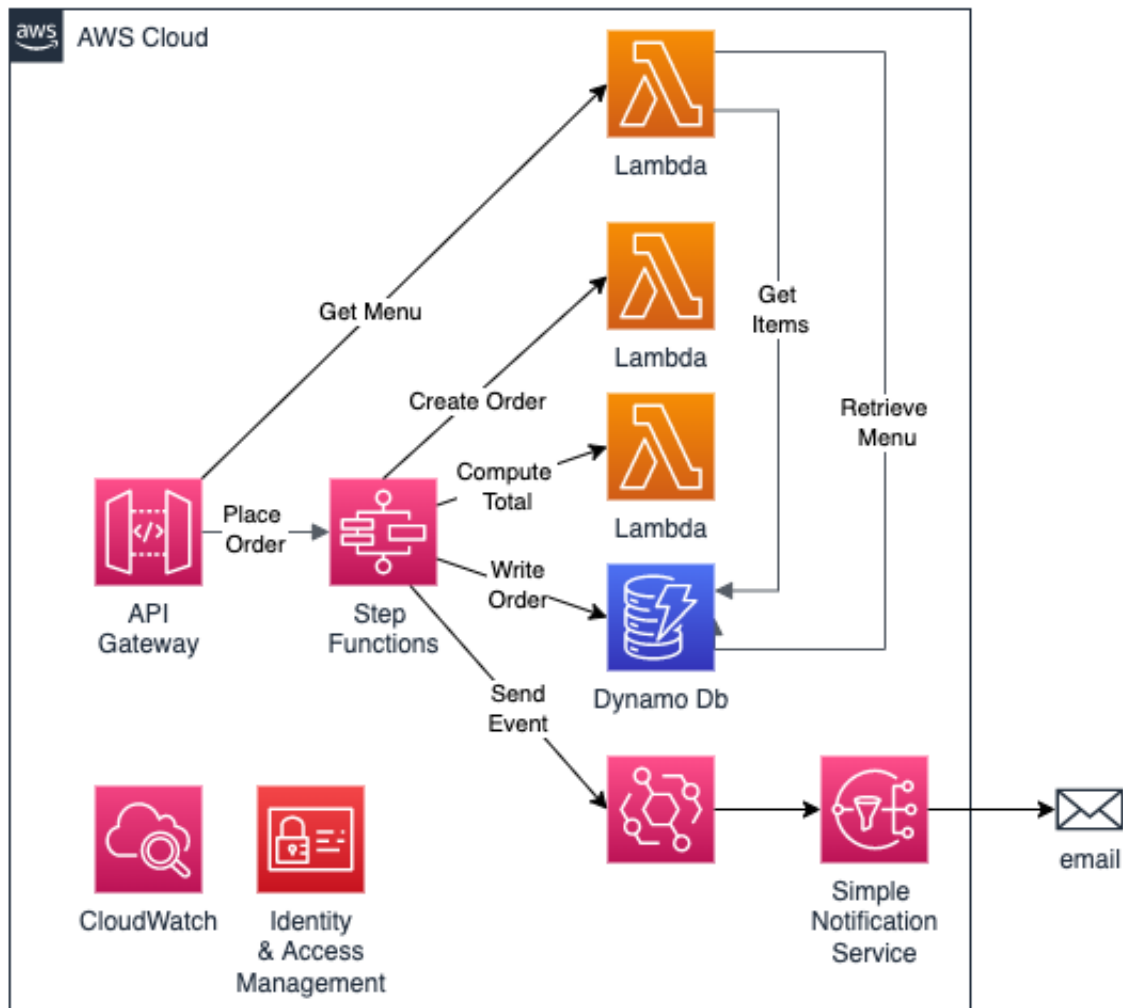
How do I use the AWS CLI?

Checkout the AWS CLI documentation here: <https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html>

OK

Build Serverless Application

AWS Serverless Architecture - Workshop



Select Region N. Virginia [us-east-1]

The workshop is setup for 'us-east-1' N. Virginia region. Confirm console region is set accordingly.

1. In the Upper right corner of the console, click the Region drop down.
2. Choose **N. Virginia [us-east-1]**
3. Confirm Region name changes to **N. Virginia**
4. Continue to Open CloudShell

Open [CloudShell](#)

1. Open Console

2. Navigate to CloudShell Service

Use CloudShell to perform the next steps.



Create S3 Bucket

```
export BUCKET_NAME=\
serverless-demo-xtz-`od -An -N2 -i /dev/random | sed 's/^ */g'`

aws s3 mb s3://$BUCKET_NAME --region us-east-1
```

Upload CloudFormation Template to S3 Bucket

```
export CF_REPO=https://raw.githubusercontent.com/robjonesjr
export CF_STACK=$CF_REPO/serverless-workshop/main/ServerlessDemoCF.yaml

curl $CF_STACK | aws s3 cp - s3://$BUCKET_NAME/ServerlessDemoCF.yaml \
--region us-east-1
```

Upload Lambda Layer for AWS-SDK

```
export LAMBDA_LAYER=$CF_REPO/serverless-workshop/main/layer.zip

curl $LAMBDA_LAYER | aws s3 cp - s3://$BUCKET_NAME/layer.zip \
--region us-east-1
```

Upload Menu JSON to S3 Bucket

```
export DYNAMO_MENU=$CF_REPO/serverless-workshop/main/dynamodb_menu.json

curl $DYNAMO_MENU | aws s3 cp - s3://$BUCKET_NAME/dynamodb_menu.json \
--region us-east-1
```

Run Cloud Formation Template (via CLI)

Specify your email address to receive notifications via email

```
export EMAIL_ADDR=youremail@email.com
```

Run Command to create cloud formation stack

```
aws cloudformation create-stack \
--stack-name "Serverless-Demo-Stack" \
--template-url https://$BUCKET_NAME.s3.amazonaws.com/ServerlessDemoCF.yaml
--parameters ParameterKey=MyEmail,ParameterValue=$EMAIL_ADDR \
ParameterKey=MyBucket,ParameterValue=$BUCKET_NAME \
--capabilities CAPABILITY_NAMED_IAM
```

View Cloud Formation Template

This will use Cloud Automation to build out Serverless resources for workshop including:

```
* API Gateway (REST)
* Step Function - State Machine
* Lambda Functions
* Event Bridge Bus and Rule
* SNS Topic
* SNS Subscription
* Security Roles and Policies
```

1. Navigate to **Cloud Formation** service (via search or menu)
2. Click on Stacks

3. Click on: **Serverless-Demo-Stack**
4. Wait for Cloud Formation template to **CREATE_COMPLETE**. You can monitor progress by clicking on **Events** tab, and periodically refreshing. Or, clicking Stack Info tab to see overall progress.
5. Click on the **Outputs** tab. You may need to click the **Refresh** button to see the Outputs.

While the template is running, an email will be received with the subject **AWS Notification - Subscription Confirmation**. Click the *Confirm Subscription* link. This enables the SNS Service to send the notification event.

Test the API Gateway endpoints

1. Navigate to CloudShell Service
2. In the Terminal window, run the following commands:

> Verify CloudFormation Create completed

```
aws cloudformation describe-stacks \  
--stack-name Serverless-Demo-Stack \  
--query 'Stacks[0].StackStatus' \  
--output text
```

> Set the ENDPOINT environment variable

Set the API Gateway Endpoint value from the CloudFormation Outputs

```
export ENDPOINT=`aws cloudformation describe-stacks \  
--stack-name Serverless-Demo-Stack \  
--query "Stacks[0].Outputs[?OutputKey=='DevEndpoint'].OutputValue" \  
--output text`
```

> Test the GET /menu endpoint

```
curl $ENDPOINT/menu | python -m json.tool
```

> Test the POST /order endpoint

```
curl -X POST -H 'Content-Type: application/json' $ENDPOINT/order \
--data-raw '{
  "customer": "Rob Jones",
  "email": "yummy@mamamiapizza.com",
  "menu_id": "e8884d14-a7d2-11ed-afa1-0242ac120002",
  "items": [
    {
      "sku": "MMP-003",
      "quantity": 1
    },
    {
      "sku": "MMP-004",
      "quantity": 1
    }
  ]
}' | python -m json.tool
```

You can modify the Order by adjusting the sku and quantity properties available in the menu.

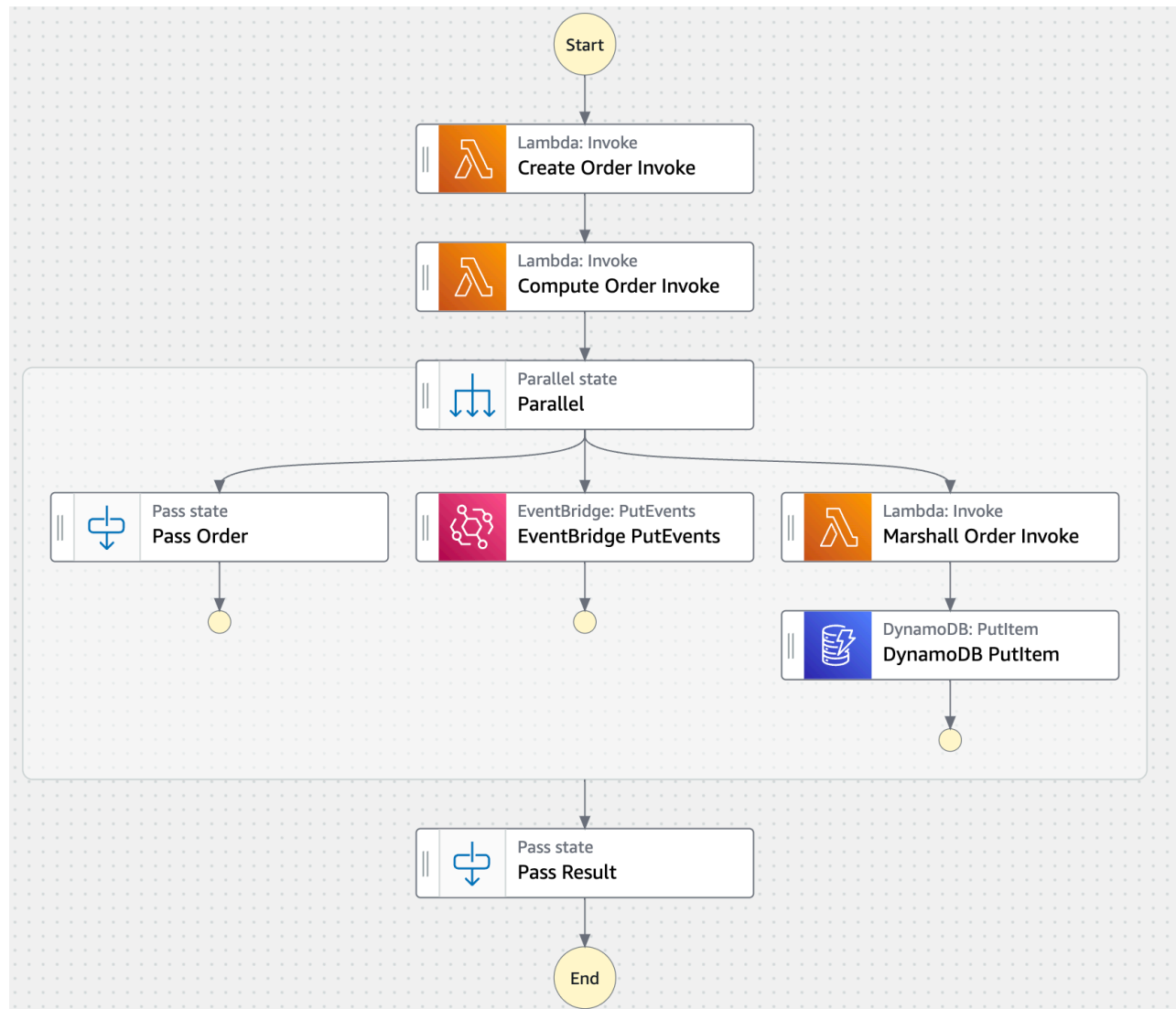
- MMP-001 = Puget Pounder
- MMP-002 = Tree Hugger
- MMP-003 = Cheese
- MMP-004 = Hurricane
- MMP-005 = Thai One On
- MMP-006 = Texas Leaguer
- MMP-007 = Super Margarita

This will result in different order results, including items and grand total calculations.

Explore Serverless Application

Step Functions

1. Navigate to Step Functions Service
2. Click on **ServerlessStateMachine**
3. Scroll down to Executions
4. Click on one of the items



Notice the **Parallel** flow below:

- **Create Order Invoke** => calls nsCreateOrder Lambda func.
- **Compute Order Invoice** => calls pyComputeOrder Lambda func.
- **Marshall Order Invoice** => calls nsMarshallOrder Lambda func, and then passes result to DynamoDb PutItem operation.
- **EventBridge Put Events** => Composes CreateOrder event, and sends to Order event bus. The event bus rule targets SNS topic to send email.

Test Invoke Step Function

1. Navigate back to the top level **ServerlessStateMachine**
2. Click on **Start Execution**
3. Select Type: **Synchronous**
4. Copy the JSON (below) into the **Input** text box

5. Click **Start Execution**
6. Wait for results

```
{
  "customer": "Vimala Pydi",
  "email": "eat@mamamiapizza.com",
  "menu_id": "e8884d14-a7d2-11ed-afa1-0242ac120002",
  "items": [
    {
      "sku": "MMP-001",
      "quantity": 1
    },
    {
      "sku": "MMP-002",
      "quantity": 1
    }
  ]
}
```

Lambda Functions

1. Navigate to Lambda Service
2. Click on **Functions** link

Notice the functions:

- nsCreateOrder - This function looks up item information and composes Order document.
- pyComputeOrder - This function computes the subtotal, tax and total amounts.
- nsMarshallOrder - This function composes the DynamoDb JSON.
- nsGetMenu - This function retrieves the Menu from DynamoDb menu table.

Click through the functions to see the code. All of these functions are written for nodejs runtime, but could also be intermixed with functions written in Java, .NET and more.

AWS Lambda natively supports Java, Go, PowerShell, Node.js, C#, Python, and Ruby code, and provides a Runtime API which allows you to use any additional programming languages to author your functions.

Open the **pyComputeOrder** function and modify the TAX_RATE

1. Click on the **Configuration** Tab
2. Click on the **Environment Variables** option
3. Click the **Edit** button

4. Change the Value to 0.15%
5. Click the **Save** button
6. Run the POST /order script in CloudShell to see the new **tax** value

DynamoDb Tables

1. Navigate to DynamoDb Service
2. Click on **Tables** link

Notice the tables:

- menu
- order

For each table, click on "**table**" link, then click on the **Explore table items** button. Navigate to the items below, and then click on the item links to view the content.

In the **orders** table, each document represents an order.

You can toggle between Form / JSON view, and toggle the DynamoDB JSON switch to view alternate formats.

Challenge: Edit the menu, change the name, or add a new item, then save. Run the GET /menu script in CloudShell to see the new menu. Create a new order (POST /order) to see the change.

For example, add this new menu item to the DynamoDb *menu* table:

```
{
  "M": {
    "description": {
      "S": "tomato sauce, mozzarella, 1.5x the pepperoni, provolone"
    },
    "name": {
      "S": "Ultimate Pepperoni"
    },
    "price": {
      "N": "39"
    },
    "sku": {
      "S": "MMP-008"
    }
  }
}
```

EventBridge Order Bus

The PutEvents call made within the Step Function asynchronously decoupled the CreateOrder event from the Synchronous workflow. Now, any interested party can leverage this Event to trigger operations.

1. Navigate to EventBridge Service
2. Click on **Rules** link
3. Select **Orders** EventBus
4. Click on **SendNotification** Rule
5. Notice the Event Pattern used for matching
6. Click on **Targets** to see the configured SNS target for the rule
7. Click on the **Event Pattern**, then modify it to only send a notification when the total amount is greater than \$50. For more information check out [Numeric matching](#) docs.

```
{
  "detail-type": ["Transaction"],
  "source": ["CreateOrder"],
  "detail": {
    "grand_total": {
      "total": [{
        "numeric": [ ">", 50 ]
      }]
    }
  }
}
```

Challenge:

Modify the order body to create an order with different amounts to test the rule. Create a new order (POST /order). For example, increase the quantity for each pizza to 2 (see sample below).

```
curl -X POST -H 'Content-Type: application/json' $ENDPOINT/order \
--data-raw '{
  "customer": "Rob Jones",
  "email": "yummy@mamamiapizza.com",
  "menu_id": "e8884d14-a7d2-11ed-afa1-0242ac120002",
  "items": [
    {
      "sku": "MMP-005",
      "quantity": 2
    },
    {
      "sku": "MMP-006",
      "quantity": 2
    }
  ]
}' | python -m json.tool
```

Note: A Rule can have multiple Targets.

Amazon Simple Notification Service (SNS)

1. Navigate to SNS Service
2. Click on **Topics** link
3. Then, click on **orders-topic** topic link

Notice the subscriptions:

The SNS topic can have zero to many subscriptions. In this case you will see one subscription. This is the email address you added to the CloudFormation template. And if you confirmed the e-mail earlier in the process, the status will indicate **confirmed**.