

# It's serverless all the way down!

Using serverless technologies to build, deploy, and serve scalable data pipelines



*Who has used serverless technologies?*





what is serverless



- what is serverless **technology**
- what is serverless **computing**
- what is serverless **architecture**
- what is serverless **framework**
- what is serverless **application**
- what is serverless **in aws**
- what is serverless **computing in azure**
- what is serverless **deployment**
- what is serverless **computing in aws**
- what is serverless **lambda**

*Report inappropriate predictions*

# Why go serverless?

# When to go serverless?

When to **NOT** go serverless?



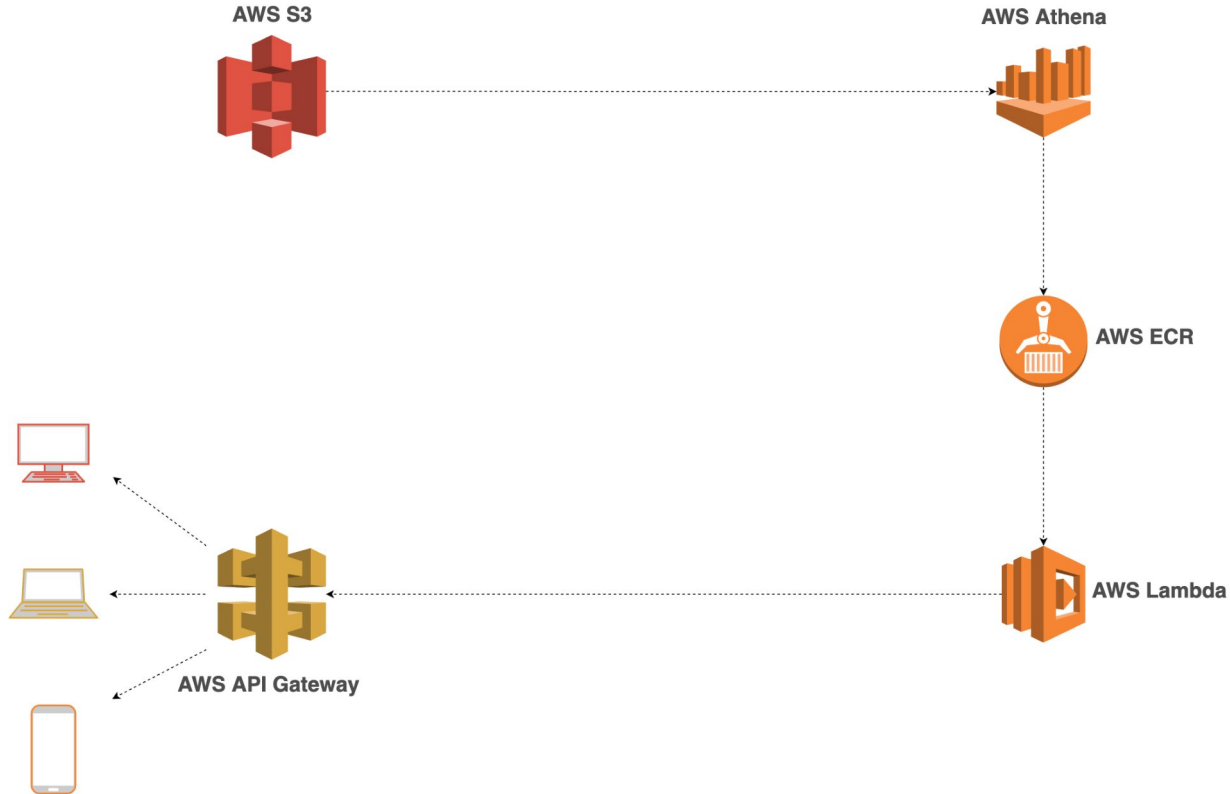
how to build serverless



- how to build serverless **applications**
- how to build serverless **architecture**
- how to **create** serverless **.yaml** file
- how to **create** serverless **website in aws**
- how to **create** serverless **project**
- how to **develop** serverless **application**
- how to **create** serverless **database in azure**
- how to **create** serverless **sql pool in azure**
- how to **create** serverless **architecture**
- how to **create** serverless **api**

*Report inappropriate predictions*

# Before giving an example, let's define some things





# For the sake of time...

- I've already populated an **S3** bucket with data that we'll retrieve here
- I've already created our **Athena** query engine that we'll use to query our data in S3

# Here's what our data in S3 looks like

```
~/projects/serverless-prez main
```

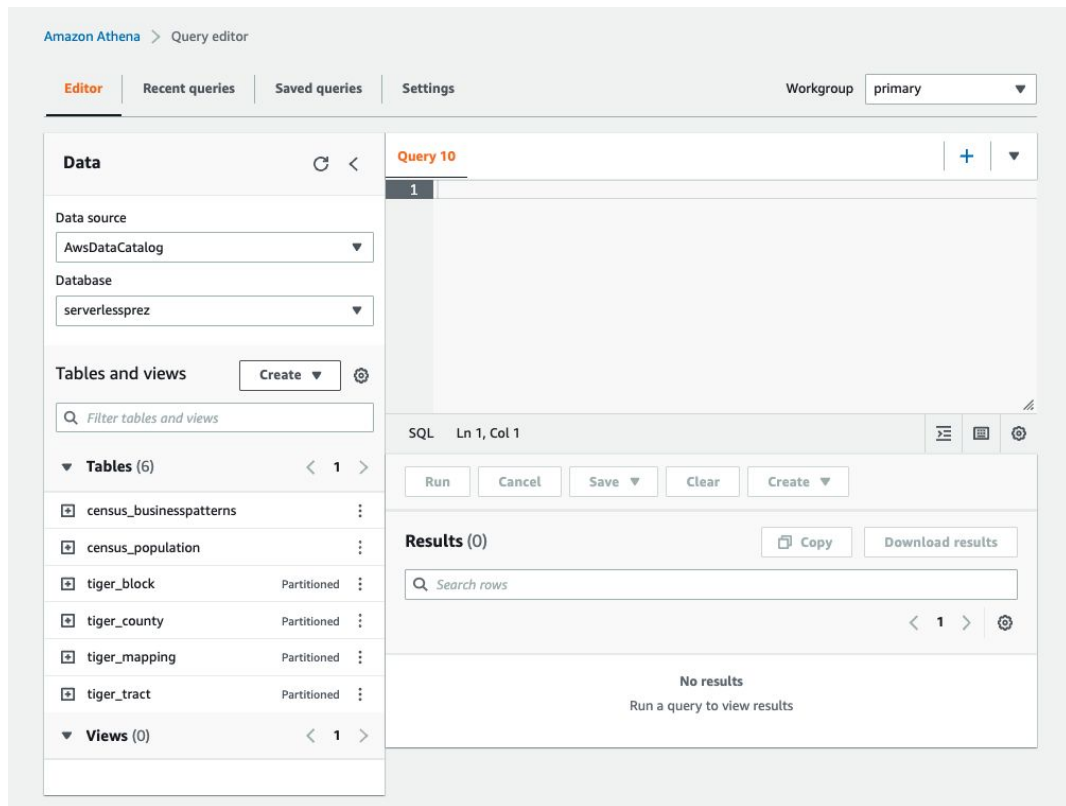
```
> tree -d -L 2 data/processed
```

```
data/processed
```

```
├── census
│   ├── businesspatterns
│   └── population
└── tiger
    ├── block
    ├── county
    ├── mapping
    ├── tract
    └── zcta
```

```
9 directories
```

# Here's what our Athena environment looks like



Now let's build the rest of it!

# We'll start with creating our Lambda function

Lambda > Functions

**Functions** (0)

Last fetched 15 seconds ago



Actions ▼

Create function

🔍 Filter by tags and attributes or search by keyword

< 1 >



Function name



Description



Package type



Runtime



Last modified



There is no data to display.

# Configure and create

Lambda > Functions > Create function

## Create function [Info](#)

Choose one of the following options to create your function.

### Author from scratch



Start with a simple Hello World example.

### Use a blueprint



Build a Lambda application from sample code and configuration presets for common use cases.

### Container image



Select a container image to deploy for your function.

### Browse serverless app repository



Deploy a sample Lambda application from the AWS Serverless Application Repository.

## Basic information

### Function name

Enter a name that describes the purpose of your function.

serverless-test

Use only letters, numbers, hyphens, or underscores with no spaces.

### Runtime [Info](#)

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Python 3.9

### Architecture [Info](#)

Choose the instruction set architecture you want for your function code.

☒ x86\_64

☐ arm64

### Permissions [Info](#)

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

► Change default execution role

► Advanced settings

Cancel

Create function

# Edit function code

Lambda > Functions > serverless-test

## serverless-test

Throttle Copy ARN Actions

▼ Function overview Info

+ Add trigger

serverless-test

Layers (0)

+ Add destination

Description  
-

Last modified  
1 minute ago

Function ARN  
arn:aws:lambda:us-east-1:639381120660:function:serverless-test

Function URL Info  
-

Code Test Monitor Configuration Aliases Versions

### Code source Info

Upload from

File Edit Find View Go Tools Window Test Deploy Changes not deployed

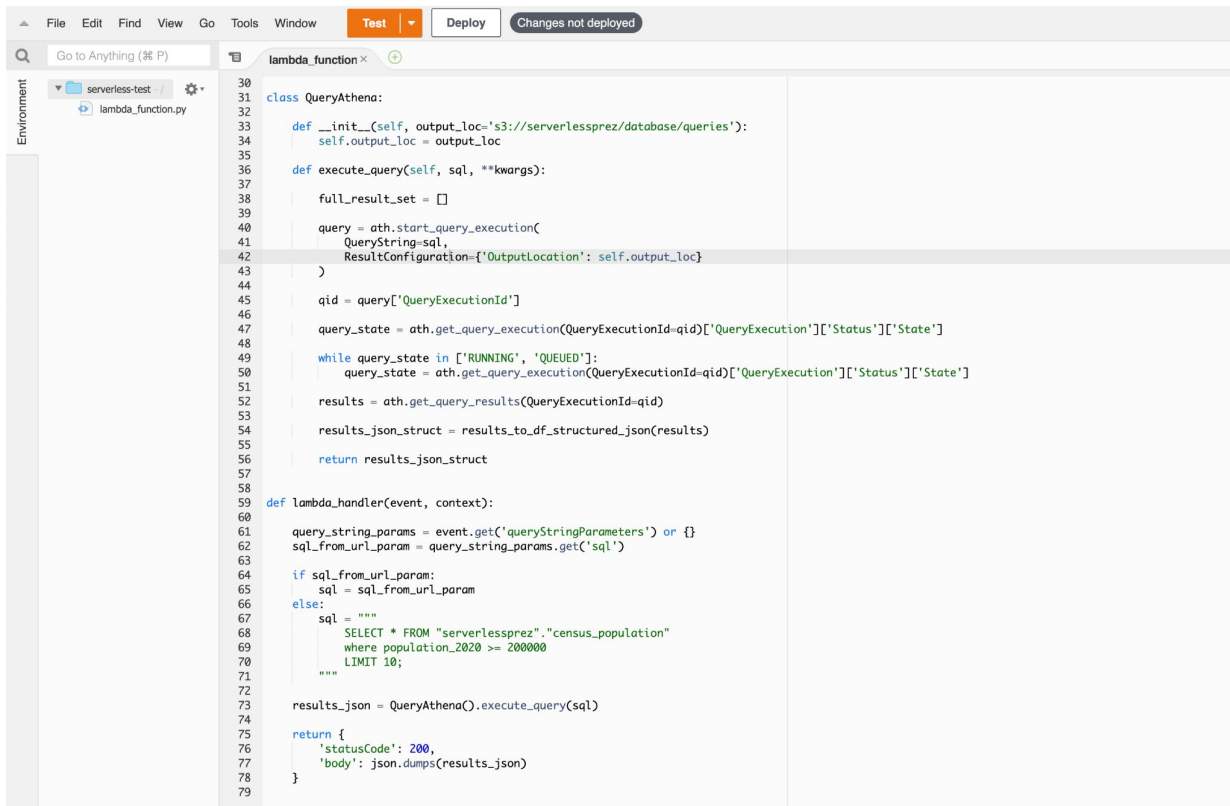
Go to Anything (⌘ P)

Environment

- serverless-test
  - lambda\_function.py

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')
8     }
9
```

# I'll use code I've already written for this



The screenshot shows a code editor interface with a menu bar (File, Edit, Find, View, Go, Tools, Window) and buttons for 'Test' and 'Deploy'. A search bar at the top left says 'Go to Anything (% P)'. The left sidebar shows an 'Environment' panel with a folder 'serverless-test' containing a file 'lambda\_function.py'. The main editor area shows the code for 'lambda\_function.py' with line numbers 30 to 79. The code defines a 'QueryAthena' class and a 'lambda\_handler' function.

```
30
31
32 class QueryAthena:
33     def __init__(self, output_loc='s3://serverlessprez/database/queries'):
34         self.output_loc = output_loc
35
36     def execute_query(self, sql, **kwargs):
37
38         full_result_set = []
39
40         query = ath.start_query_execution(
41             QueryString=sql,
42             ResultConfiguration={'OutputLocation': self.output_loc}
43         )
44
45         qid = query['QueryExecutionId']
46
47         query_state = ath.get_query_execution(QueryExecutionId=qid)['QueryExecution']['Status']['State']
48
49         while query_state in ['RUNNING', 'QUEUED']:
50             query_state = ath.get_query_execution(QueryExecutionId=qid)['QueryExecution']['Status']['State']
51
52         results = ath.get_query_results(QueryExecutionId=qid)
53         results_json_struct = results_to_df_structured_json(results)
54
55         return results_json_struct
56
57
58 def lambda_handler(event, context):
59
60     query_string_params = event.get('queryStringParameters') or {}
61     sql_from_url_param = query_string_params.get('sql')
62
63     if sql_from_url_param:
64         sql = sql_from_url_param
65     else:
66         sql = """
67         SELECT * FROM "serverlessprez"."census_population"
68         where population_2020 >= 200000
69         LIMIT 10;
70         """
71
72     results_json = QueryAthena().execute_query(sql)
73
74     return {
75         'statusCode': 200,
76         'body': json.dumps(results_json)
77     }
78
79
```



# Test our Lambda function before deploying

The screenshot shows the AWS Lambda console interface. In the background, a code editor displays a Python Lambda function named `lambda_function` that uses `json` and `boto3` libraries. The foreground features a 'Configure test event' modal window. This window provides instructions on how to create a test event, which is a JSON object mimicking AWS service requests. It offers two actions: 'Create new event' (selected) and 'Edit saved event'. The 'Event name' field is populated with 'serverless-test-event'. Under 'Event sharing settings', the 'Private' option is selected. A 'Template' dropdown is set to 'hello-world'. At the bottom, the 'Event JSON' section shows a pre-filled JSON object with three key-value pairs, and a 'Format JSON' button is available.

**Configure test event**

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, configure the JSON event, then choose Test.

**Test event action**

- ☒ Create new event
- ☐ Edit saved event

**Event name**

serverless-test-event

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

**Event sharing settings**

- ☒ Private  
This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)
- ☐ Shareable  
This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

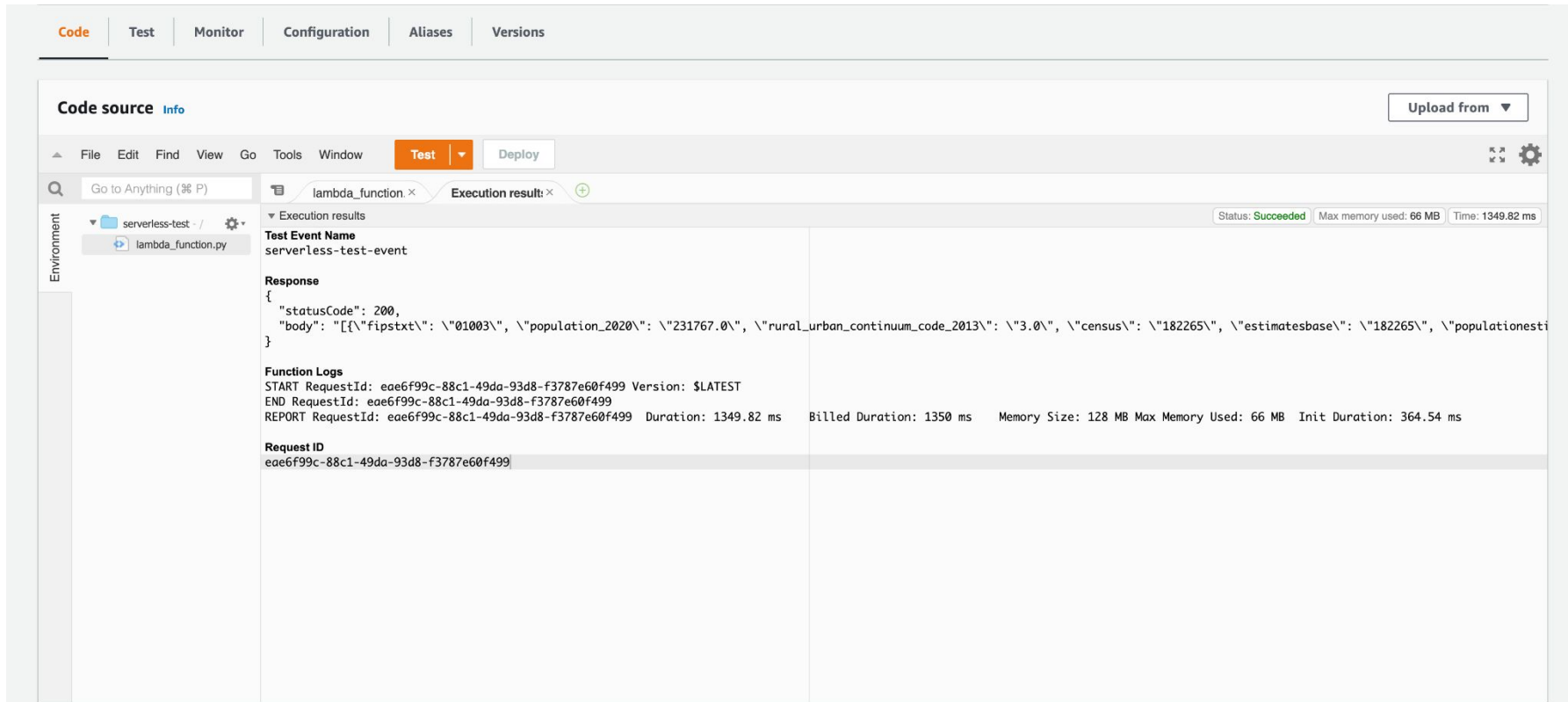
**Template - optional**

hello-world

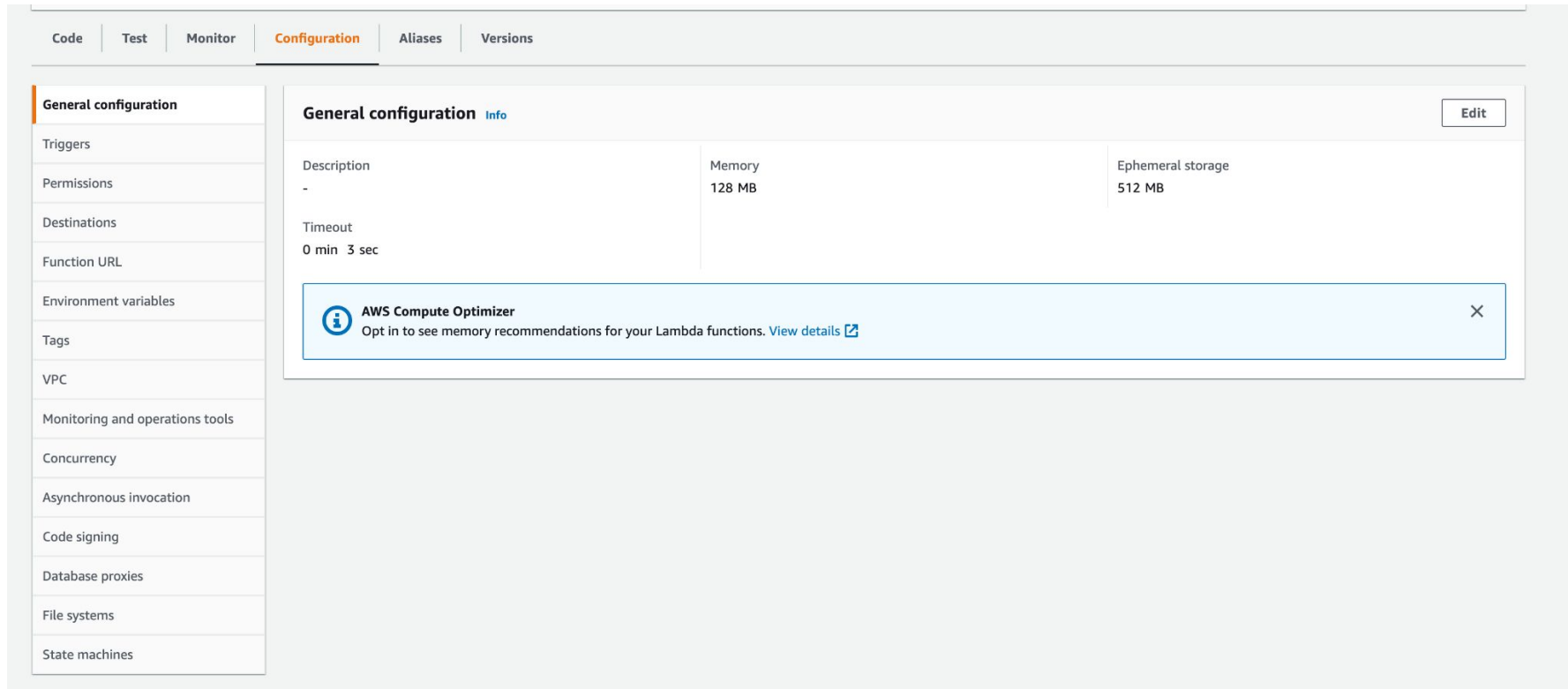
**Event JSON** Format JSON

```
1 {
2   "key1": "value1",
3   "key2": "value2",
4   "key3": "value3"
5 }
```

# It works!



# Now let's extend the timeout and memory



The screenshot displays the AWS Lambda console's Configuration tab for a specific function. The left sidebar lists various configuration categories, with 'General configuration' selected. The main panel shows the 'General configuration' section with an 'Edit' button. The configuration table includes fields for Description, Memory (128 MB), Ephemeral storage (512 MB), and Timeout (0 min 3 sec). An AWS Compute Optimizer notification is visible at the bottom of the configuration panel.

**Configuration** | Aliases | Versions

**General configuration**

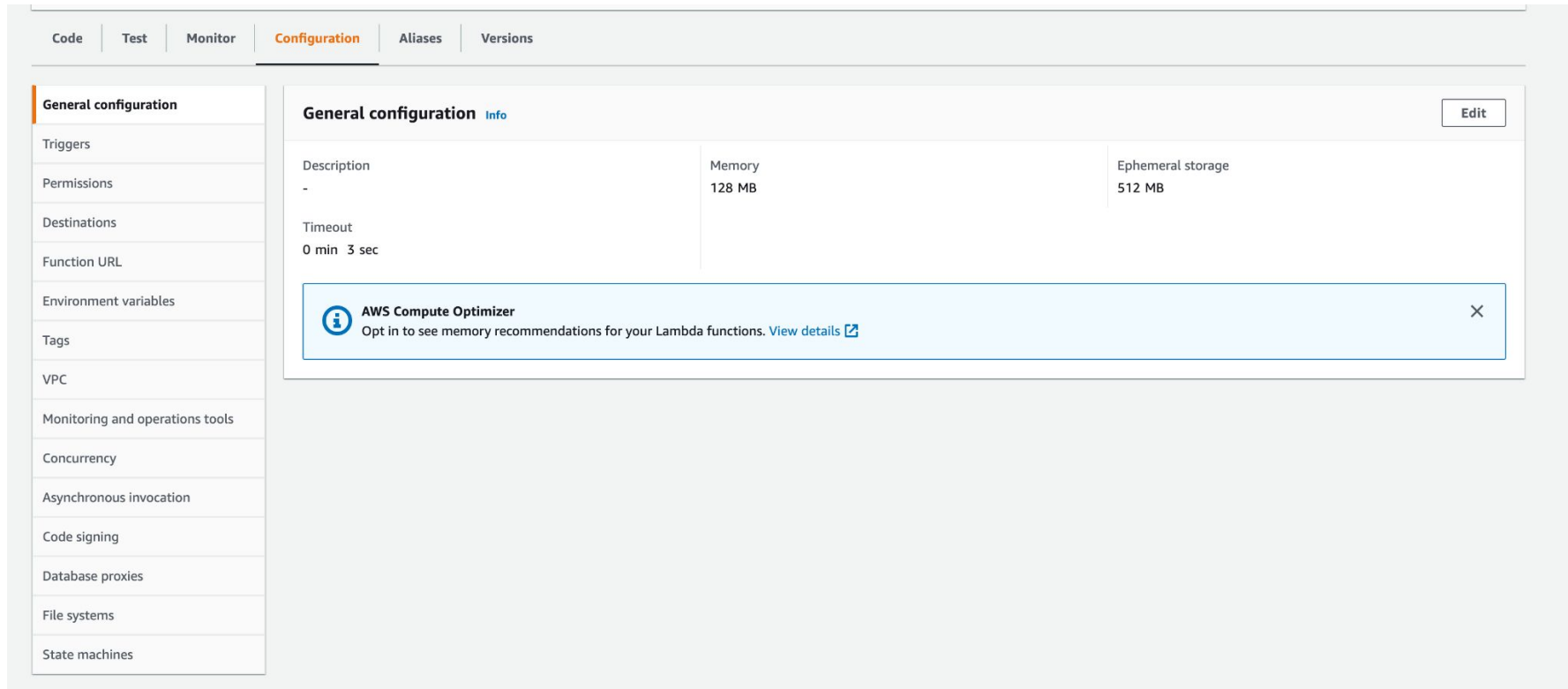
- Triggers
- Permissions
- Destinations
- Function URL
- Environment variables
- Tags
- VPC
- Monitoring and operations tools
- Concurrency
- Asynchronous invocation
- Code signing
- Database proxies
- File systems
- State machines

**General configuration** [Info](#) [Edit](#)

Description	Memory	Ephemeral storage
-	128 MB	512 MB
Timeout		
0 min 3 sec		

**AWS Compute Optimizer**  
Opt in to see memory recommendations for your Lambda functions. [View details](#)

# Now let's extend the timeout and memory



The screenshot displays the AWS Lambda console's Configuration tab for a specific function. The left sidebar lists various configuration categories, with 'General configuration' selected. The main panel shows the 'General configuration' section with an 'Edit' button. The configuration table includes fields for Description, Memory (128 MB), Ephemeral storage (512 MB), and Timeout (0 min 3 sec). An AWS Compute Optimizer notification is visible at the bottom of the configuration panel.

**Configuration** | Aliases | Versions

**General configuration**

- Triggers
- Permissions
- Destinations
- Function URL
- Environment variables
- Tags
- VPC
- Monitoring and operations tools
- Concurrency
- Asynchronous invocation
- Code signing
- Database proxies
- File systems
- State machines

**General configuration** [Info](#) [Edit](#)

Description	Memory	Ephemeral storage
-	128 MB	512 MB
Timeout		
0 min 3 sec		

**AWS Compute Optimizer**  
Opt in to see memory recommendations for your Lambda functions. [View details](#)

# Now let's make our data accessible over HTTPS

Lambda > Functions > serverless-test > Edit basic settings

## Edit basic settings

**Basic settings** [Info](#)

Description - *optional*

**Memory** [Info](#)  
Your function is allocated CPU proportional to the memory configured.

1024

 MB

Set memory to between 128 MB and 10240 MB

**Ephemeral storage** [Info](#)  
You can configure up to 10 GB of ephemeral storage (/tmp) for your function. [View pricing](#)

512

 MB

Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.

**Timeout**

1

 min 

0

 sec


**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☒ Use an existing role

☐ Create a new role from AWS policy templates

**Existing role**  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

serverlessprez



[View the serverlessprez role on the IAM console.](#)

Cancel Save

# Create an API on AWS API Gateway

API Gateway

×

APIs

Custom domain names

VPC links

HTTP API

Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

Works with the following:  
Lambda, HTTP backends

ImportBuild

WebSocket API

Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.

Works with the following:  
Lambda, HTTP, AWS Services

Build

REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:  
Lambda, HTTP, AWS Services

ImportBuild

REST API Private

Create a REST API that is only accessible from within a VPC.

Works with the following:  
Lambda, HTTP, AWS Services

ImportBuild

# Name the API



APIs

Custom Domain Names

VPC Links

## Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

☒ REST ☐ WebSocket

## Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

☒ New API ☐ Clone from existing API ☐ Import from Swagger or Open API 3 ☐ Example API

## Settings

Choose a friendly name and description for your API.

API name\*

serverless-test

Description

Endpoint Type

Regional



\* Required

Create API

# Create a resource on the API

The screenshot displays the Amazon API Gateway console interface. The breadcrumb navigation at the top reads: Amazon API Gateway > APIs > serverless-test (bd98s76pnk) > Resources > / (j62k5u5uve). On the left sidebar, the 'APIs' section is expanded, showing 'Custom Domain Names', 'VPC Links', and 'API: serverless-test'. Under 'API: serverless-test', the 'Resources' link is highlighted. The main content area shows the 'Resources' tab selected, with a dropdown menu open for the resource '/'. The dropdown menu is divided into two sections: 'RESOURCE ACTIONS' and 'API ACTIONS'. Under 'RESOURCE ACTIONS', the options are 'Create Method', 'Create Resource' (which is highlighted), 'Enable CORS', and 'Edit Resource Documentation'. Under 'API ACTIONS', the options are 'Deploy API', 'Import API', 'Edit API Documentation', and 'Delete API'. The main content area on the right displays the message 'No methods defined for the resource.'

Amazon API Gateway

APIs > serverless-test (bd98s76pnk) > Resources > / (j62k5u5uve)

APIs

Custom Domain Names

VPC Links

API: **serverless-test**

**Resources**

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Dashboard

Settings

Usage Plans

API Keys

Client Certificates

Settings

Resources

Actions

/ Methods

RESOURCE ACTIONS

- Create Method
- Create Resource
- Enable CORS
- Edit Resource Documentation

API ACTIONS

- Deploy API
- Import API
- Edit API Documentation
- Delete API

No methods defined for the resource.



# Create a resource on the API

Amazon API Gateway

APIs > serverless-test (bd98s76pnk) > Resources > / (j62k5u5uve) > Create

Show all hints ?

APIs

Custom Domain Names

VPC Links

API: **serverless-test**

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Dashboard

Settings

Usage Plans

API Keys

Client Certificates

Settings

Resources

Actions

New Child Resource

Use this page to create a new child resource for your resource.

Configure as ☒ proxy resource

Resource Name\*

serverless-test

Resource Path\*

/ serverless-test

You can add path parameters using brackets. For example, the resource path **{username}** represents a path parameter called 'username'. Configuring **/[proxy+]** as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.

Enable API Gateway CORS

☒

\* Required

Cancel Create Resource

# Create a method on the resource

The screenshot displays the Amazon API Gateway console interface. The breadcrumb navigation at the top reads: **Amazon API Gateway** > **APIs** > **serverless-test (bd98s76pnk)** > **Resources** > **/serverless-test (gj00x0)** > **OPTIONS**.

On the left sidebar, the **API: serverless-test** is selected, and the **Resources** section is expanded. The resource **/serverless-test - OPTIONS** is highlighted.

A context menu is open over the resource, showing the following actions:

- METHOD ACTIONS**
  - Edit Method Documentation
  - Delete Method
- RESOURCE ACTIONS**
  - Create Method
  - Create Resource
  - Enable CORS
  - Edit Resource Documentation
  - Delete Resource
- API ACTIONS**
  - Deploy API
  - Import API
  - Edit API Documentation
  - Delete API

Two panels on the right show the configuration for the selected resource:

- Method Request**
  - Auth:** NONE
  - ARN:** arn:aws:execute-api:us-east-1:639381120660:bd98s76pnk/\*/OPTIONS/serverless-test
- Method Response**
  - HTTP Status:** 200
  - Models:** application/json => Empty

Arrows indicate the flow of configuration from the resource to the Method Request and Method Response panels.

# Create a method on the resource

The screenshot displays the Amazon API Gateway console interface. The breadcrumb navigation at the top reads: Amazon API Gateway > APIs > serverless-test (bd98s76pnk) > Resources > /serverless-test (gj00x0) > OPTIONS.


On the left sidebar, the 'APIs' section is expanded, showing 'Custom Domain Names', 'VPC Links', and 'API: serverless-test'. Under 'API: serverless-test', the 'Resources' link is selected.


The main content area shows the 'Resources' tab for the API. A tree view on the left lists the resource hierarchy: '/' (expanded) > '/serverless-test' (expanded) > 'OPTIONS' (selected). A context menu is open over the 'OPTIONS' resource, listing available HTTP methods: ANY, DELETE, GET (highlighted), HEAD, PATCH, POST, and PUT.


To the right of the resource tree, the 'Method Execution' section is visible. It contains two panels: 'Method Request' and 'Method Response'. The 'Method Request' panel shows 'Auth: NONE' and 'ARN: arn:aws:execute-api:us-east-1:639381120660:bd98s76pnk:/'OPTIONS/serverless-test'. The 'Method Response' panel shows 'HTTP Status: 200' and 'Models: application/json => Empty'. A 'Client' box with a 'TEST' button is positioned between the two panels, with arrows indicating the flow of the request and response.

# Use the Lambda function as the integration point

APIs > serverless-test (bd98s76pnk) > Resources > /serverless-test (gj00x0) > GET Show all hints

Resources **Actions**  /serverless-test - GET - Setup

 /

 /serverless-test

GET

OPTIONS

Choose the integration point for your new method.

**Integration type** ☒ Lambda Function ⓘ

☐ HTTP ⓘ

☐ Mock ⓘ

☐ AWS Service ⓘ

☐ VPC Link ⓘ

**Use Lambda Proxy integration** ☒ ⓘ

**Lambda Region**

**Lambda Function**  ⓘ

**Use Default Timeout** ☒ ⓘ

# Test it out

APIs > serverless-test (bd98s76pnk) > Resources > /serverless-test (gj00x0) > GET

Show all hints ?

Resources Actions

Method Execution /serverless-test - GET - Method Test

Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method

**Path**

No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.

**Query Strings**

**{serverless-test}**

**Headers**

**{serverless-test}**

Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg. Accept:application/json.

**Stage Variables**

No [stage variables](#) exist for this method.

**Client Certificate**

No client certificates have been generated.

**Request Body**

Request Body is not supported for GET methods.

**Request: /serverless-test**

**Status:** 200  
**Latency:** 1799 ms

**Response Body**

```
{
  {
    "fipstxt": "01003",
    "population_2020": "231767.0",
    "rural_urban_continuum_code_2013": "3.0",
    "census": "182265",
    "estimatesbase": "182265",
    "populationestimate_2010": "183112",
    "populationestimate_2011": "186558",
    "populationestimate_2012": "190145",
    "populationestimate_2013": "194885",
    "populationestimate_2014": "199183",
    "populationestimate_2015": "202939",
    "populationestimate_2016": "207601",
    "populationestimate_2017": "212521",
    "populationestimate_2018": "217855",
    "populationestimate_2019": "223234",
    "st": "01",
    "cty": "003",
    "ctyname": "Baldwin County, Alabama"
  },
  {
    "fipstxt": "01073",
    "population_2020": "674721.0",
    "rural_urban_continuum_code_2013": "1.0",
    "census": "658466",
    "estimatesbase": "658567",
    "populationestimate_2010": "658215",
    "populationestimate_2011": "658109",
    "populationestimate_2012": "658061",
    "populationestimate_2013": "659265",
    "populationestimate_2014": "659972",
    "populationestimate_2015": "664455"
  }
}
```

Test

# Deploy the API (so we can access it)

The screenshot displays the Amazon API Gateway console interface. The breadcrumb navigation at the top reads: **Amazon API Gateway** > **APIs** > **serverless-test (bd98s76pnk)** > **Resources** > **/serverless-test (gj00x0)** > **GET**.

On the left sidebar, the navigation menu includes: **APIs**, **Custom Domain Names**, **VPC Links**, **API: serverless-test** (selected), **Resources** (sub-selected), **Stages**, **Authorizers**, **Gateway Responses**, **Models**, **Resource Policy**, **Documentation**, **Dashboard**, **Settings**, **Usage Plans**, **API Keys**, **Client Certificates**, and **Settings**.

The main content area shows the **Resources** tab for the selected API. A context menu is open over the **GET** resource, listing the following actions:

- METHOD ACTIONS**
  - Edit Method Documentation
  - Delete Method
- RESOURCE ACTIONS**
  - Create Method
  - Create Resource
  - Enable CORS
  - Edit Resource Documentation
  - Delete Resource
- API ACTIONS**
  - Deploy API
  - Import API
  - Edit API Documentation
  - Delete API

The **Deploy API** action is highlighted. To the right of the menu, the **Method Request** and **Method Response** configuration panels are visible:

- Method Request**
  - Auth:** NONE
  - ARN:** arn:aws:execute-api:us-east-1:639381120660:bd98s76pnk/\*/GET/serverless-test
- Method Response**
  - HTTP Status:** Proxy
  - Models:** application/json => Empty

Arrows indicate the flow from the **Deploy API** action to the **Method Request** and **Method Response** panels.

# Name the deployment

The screenshot shows the AWS API Gateway console interface. A modal dialog titled "Deploy API" is open in the center. The background shows the "Resources" tab for an API named "serverless-test (bd98s76pnk)". The "Resources" list includes a "GET" method under the path "/serverless-test". The "Actions" menu is open, and the "TEST" option is selected. The "Deploy API" dialog contains the following fields:

- Deployment stage:** A dropdown menu currently showing "[New Stage]".
- Stage name\*:** A text input field containing the text "dev".
- Stage description:** An empty text input field.
- Deployment description:** An empty text input field.

At the bottom of the dialog are two buttons: "Cancel" and "Deploy". The background interface also shows a "Method Response" section for the selected GET method, indicating "HTTP Status: Proxy" and "Models: application/json => Empty". To the right, there is an "Integration Request" section with "Type: LAMBDA\_PROXY" and an "Integration Response" section with the text "Proxy integrations cannot be configured to transform responses."

# Now let's call it from Jupyter

```
In [1]: %%time
import urllib.parse
import pandas as pd
import requests

sql = urllib.parse.quote_plus(
    """SELECT
        county.statefp,
        county.countyfp,
        pop.population_2020,
        pop.rural_urban_continuum_code_2013,
        pop.ctyname,
        ST_GeomFromLegacyBinary(county.geometry) as geometry
    FROM serverlessprez.census_population pop
    JOIN serverlessprez.tiger_county county
    ON county.statefp = pop.st
    AND county.countyfp = pop.cty
    WHERE pop.st = '47';"""
)

url = f"https://bd98s76pnk.execute-api.us-east-1.amazonaws.com/dev/serverless-test?sql={sql}"
resp = requests.get(url)
df = pd.DataFrame(resp.json())
```

CPU times: user 948 ms, sys: 269 ms, total: 1.22 s  
Wall time: 4.83 s

```
In [2]: df.head()
```

Out[2]:

	statefp	countyfp	population_2020	rural_urban_continuum_code_2013	ctyname	geometry
0	47	039	11435.0	9.0	Decatur County, Tennessee	POLYGON ((-87.976434 35.506732, -87.976406 35....
1	47	043	54315.0	1.0	Dickson County, Tennessee	POLYGON ((0 -87.178306, -87.178306 36.228262, ...
2	47	049	18489.0	9.0	Fentress County, Tennessee	POLYGON ((-84.852506 36.292433, -84.852417 36....
3	47	057	23527.0	2.0	Grainger County, Tennessee	POLYGON ((-83.381502 36.265431, -83.381297 36....
4	47	059	70152.0	4.0	Greene County, Tennessee	POLYGON ((-82.785041 35.987096, -82.785074 35....



# Doing more stuff with that data in Jupyter

```
In [17]: import geopandas as gpd
import folium

df['geometry'] = gpd.GeoSeries.from_wkt(df['geometry'])
gdf = gpd.GeoDataFrame(df, geometry="geometry").reset_index()
gdf['population_2020'] = gdf['population_2020'].astype(float)
gdf = gdf.set_crs(epsg = 4326)

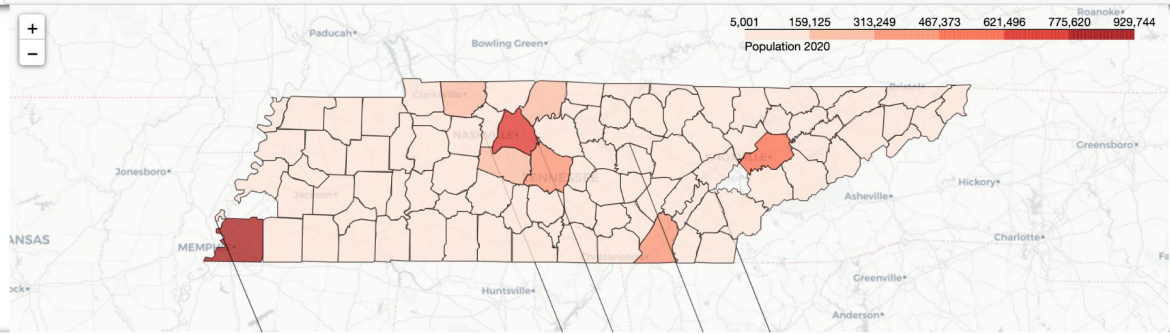
m = folium.Map(location=[35.59, -86.94], zoom_start=7, tiles='CartoDB positron')

choropleth = folium.Choropleth(
    geo_data=gdf.to_json(),
    name='choropleth',
    data=gdf,
    columns=["ctyname", "population_2020"],
    fill_color='Reds',
    fill_opacity=0.75,
    line_opacity=.6,
    key_on="feature.properties.ctyname",
    legend_name='Population 2020',
    highlight=True,
    reset=True
).add_to(m)

choropleth.geojson.add_child(folium.GeoJsonTooltip(['ctyname', 'population_2020'], labels=False))

m
```

Out[17]:



# Need help with a project or have questions?



## **Timothy Dobbins**

Data Scientist | Data Engineer | Consultant

**Github:** [github.com/tmthyjames](https://github.com/tmthyjames)

**Linkedin:** [linkedin.com/in/tim-dobbins](https://linkedin.com/in/tim-dobbins)

**Email:** [tmthyjames@gmail.com](mailto:tmthyjames@gmail.com)

Or scan this →



**Presentation Github link:** [github.com/tmthyjames/serverless-prez](https://github.com/tmthyjames/serverless-prez)