Lambda execution

AWS LAMBDA SERVERLESS APPLICATIONS



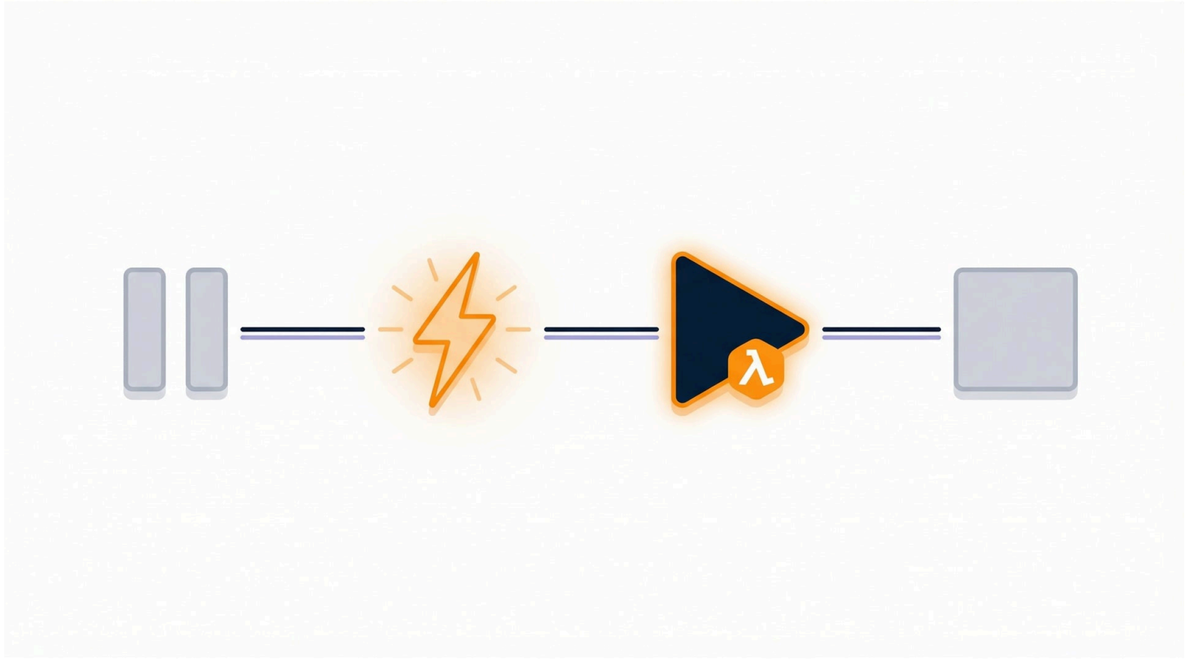
Claudio Canales
Senior DevOps Engineer

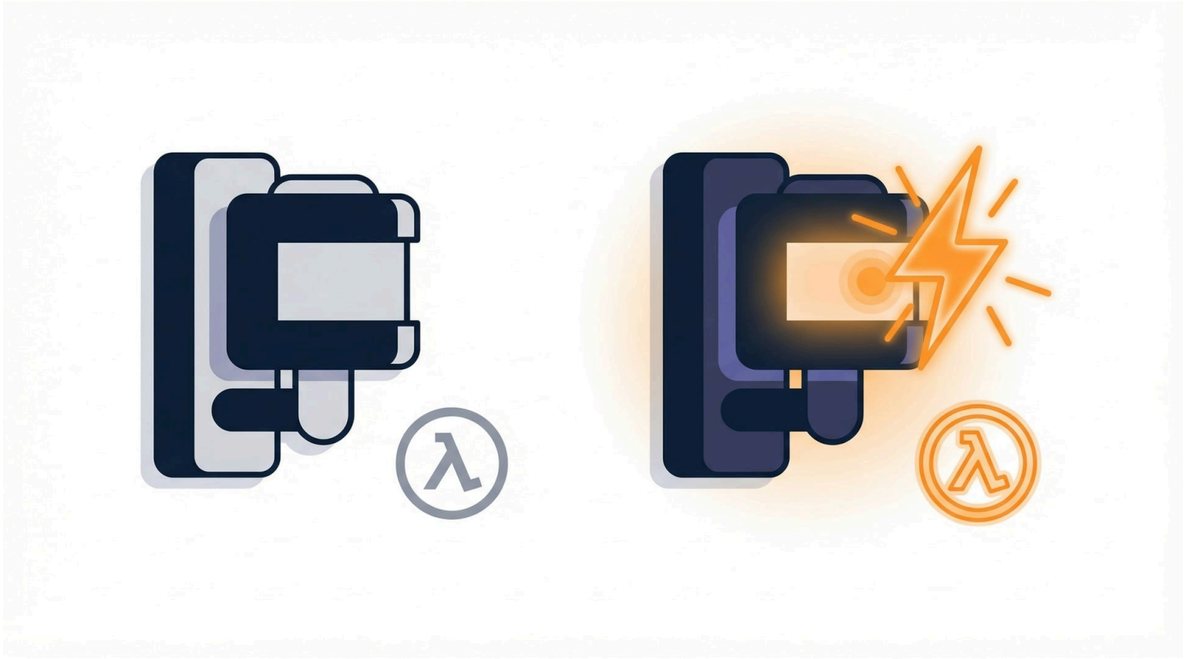
execution



- Lambda
- : S3 -> resize
- ,

idle





- Idle:
- Event: ,

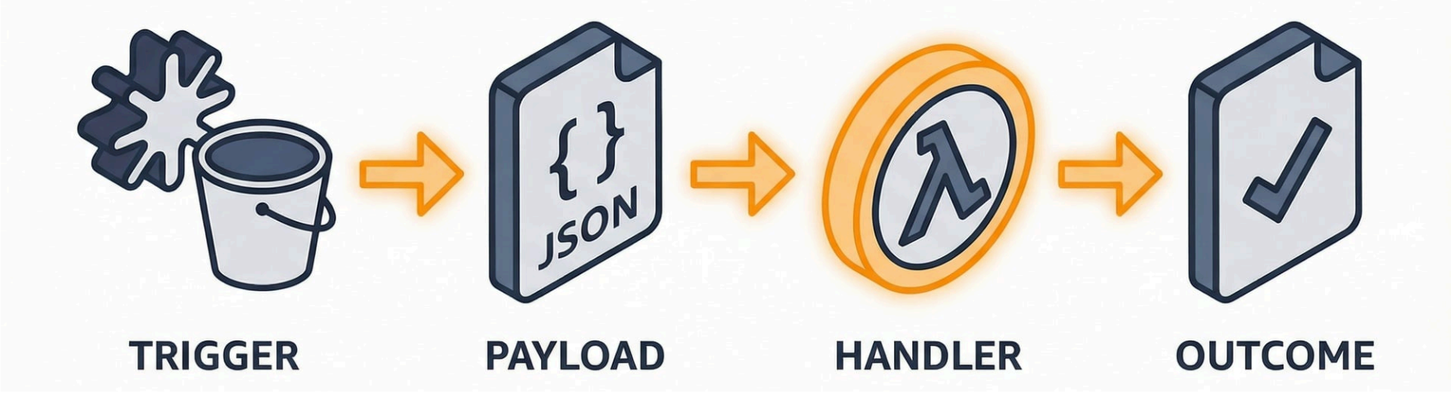
- : idle
- "off" "on"

AWS Lambda ?

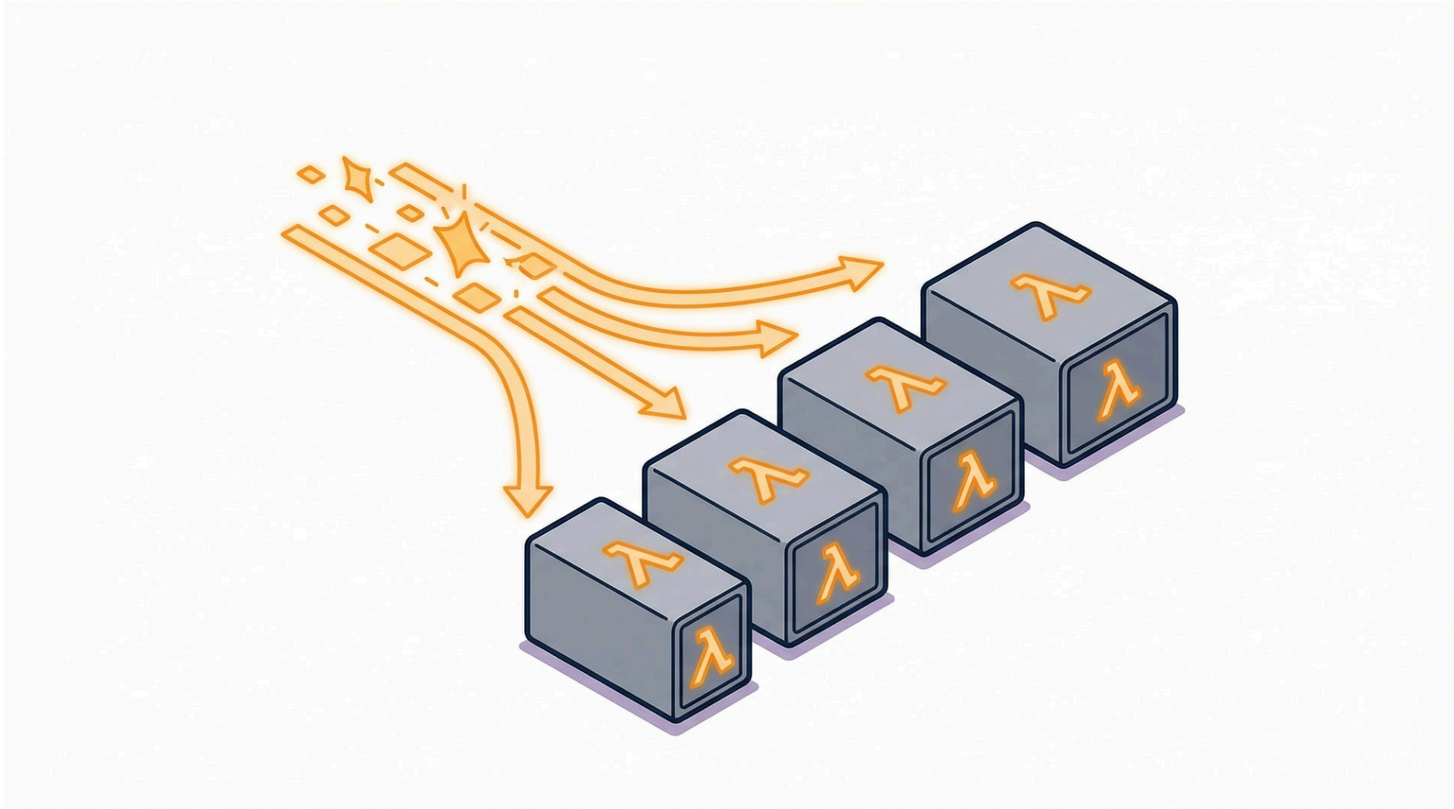


- - compute □□□□□□□
- -
- runtime

Lambda

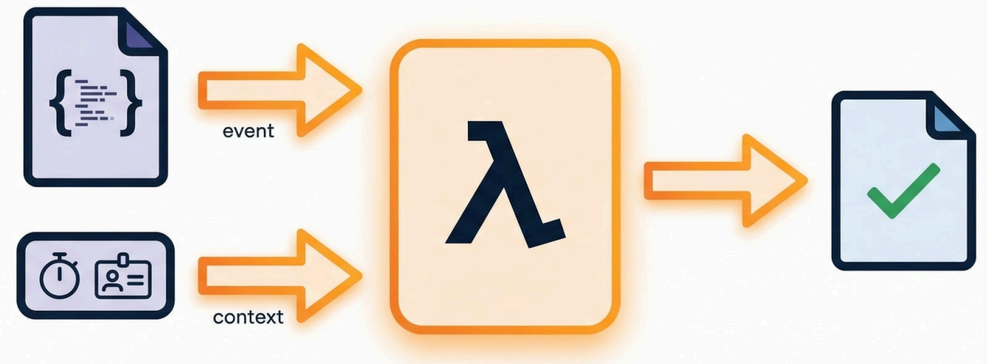


- event source Lambda
- Lambda event payload
- outcome



- events: parallel invocations
- Lambda environments spin-up
- environment cold start

```python
def lambda_handler(event, context):
    # Your code here
    return {
        'statusCode': 200,
        'body': 'Hello from Lambda!'
    }
```



- `event` : payload (JSON)
- `context` : runtime
- return: (success/fail)

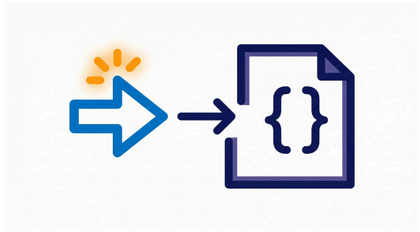
event □□□□□□□□



S3 event: bucket + object key□

```
# event["Records"][0]["s3"]
{
  "bucket": {"name": "my-bucket"},
  "object": {"key": "data.csv"}
}
```

- : ,



API event: method, path, body□

```
# API request event
{
  "httpMethod": "POST",
  "path": "/users",
  "body": "{\"name\": \"John\"}"
}
```

- , JSON □□□□

context □□□□□□□□

```
def lambda_handler(event, context):
    time_left = (
        context.get_remaining_time_in_millis()
    )
    request_id = context.aws_request_id
```

- `get_remaining_time_in_millis`
- `aws_request_id` request ID□

Stateless execution

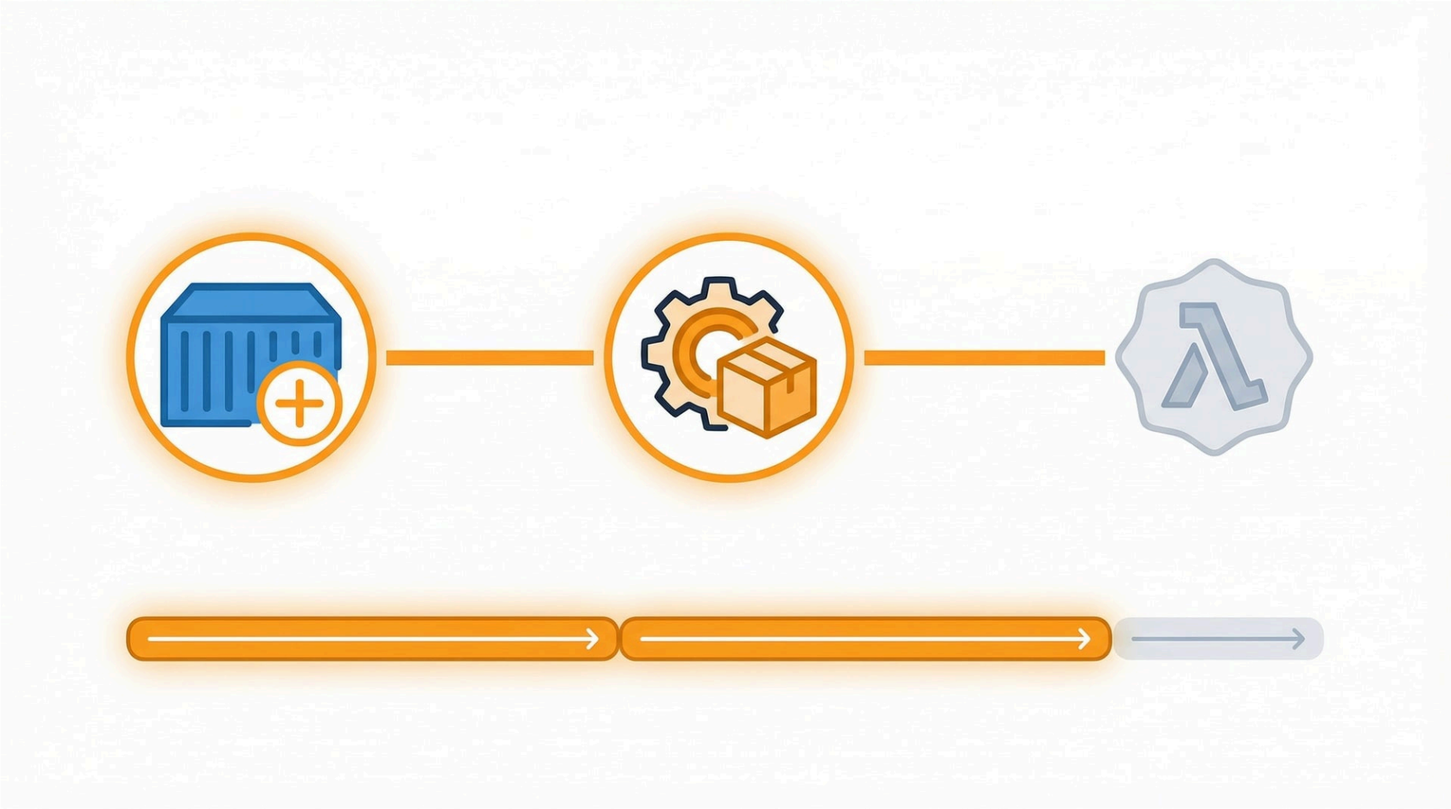


- invocation
- Reuse ,
- Lambda

```
counter = 0

def lambda_handler(event, context):
    global counter
    counter += 1  # may reset
    return counter
```

Cold starts



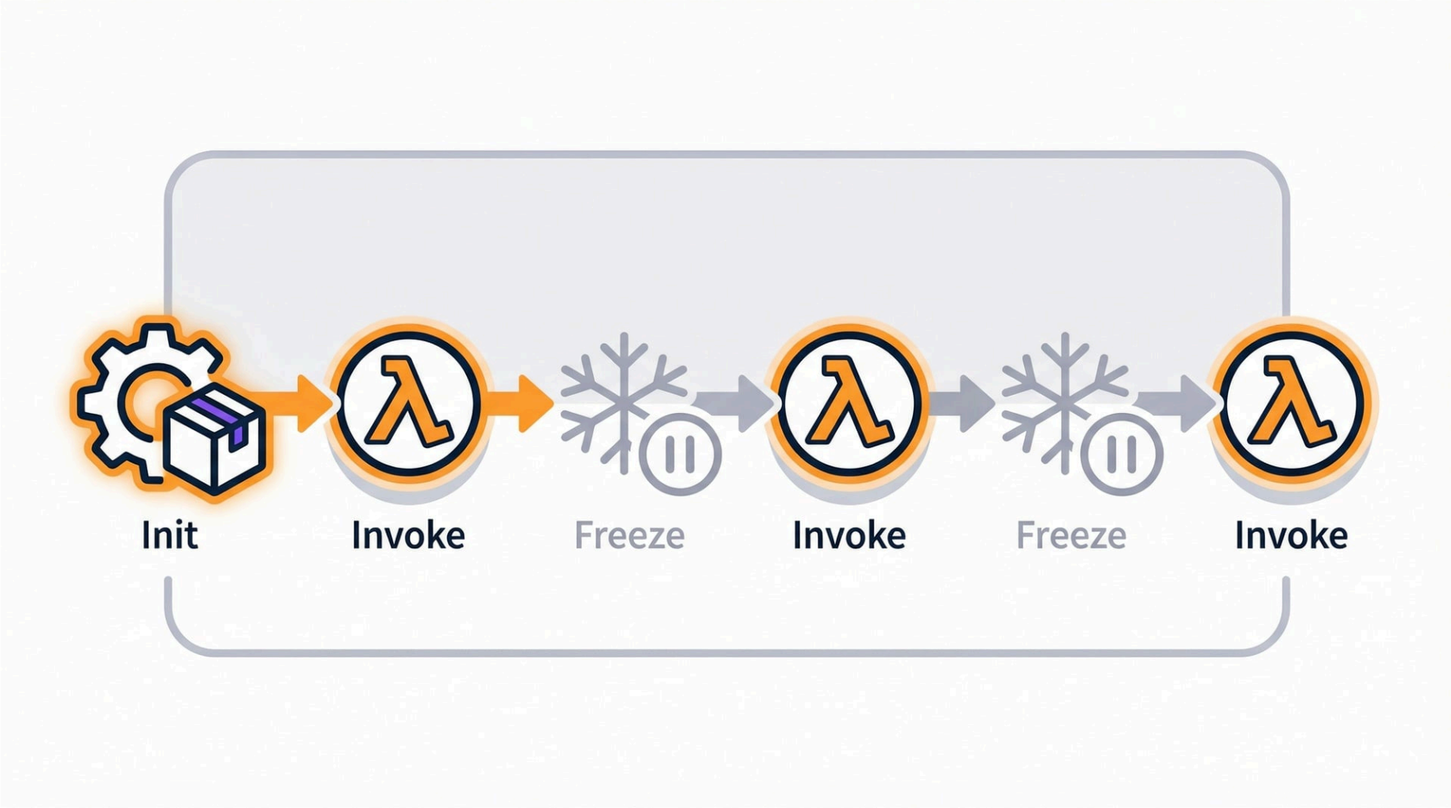
- execution environment
- dependencies
- Initialization,

Warm starts



- environment reuse
- initialization skip
-

Init, invoke, freeze

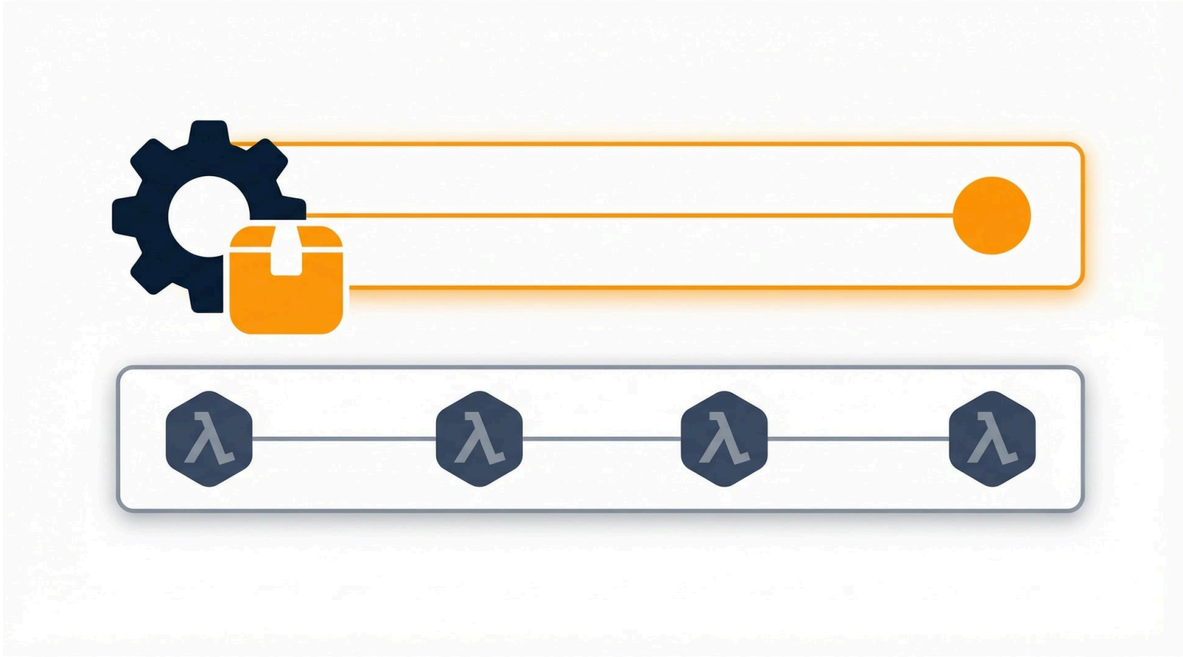


- Init environment
- invocation
- env freeze

Cold starts optimization

```
# init (once per env)
import boto3
s3 = boto3.client('s3')
```

```
# handler (every invocation)
def lambda_handler(event, context):
    return {"statusCode": 200}
```



- imports SDK

AWS LAMBDA

SERVERLESS APPLICATIONS !

Lambda functions

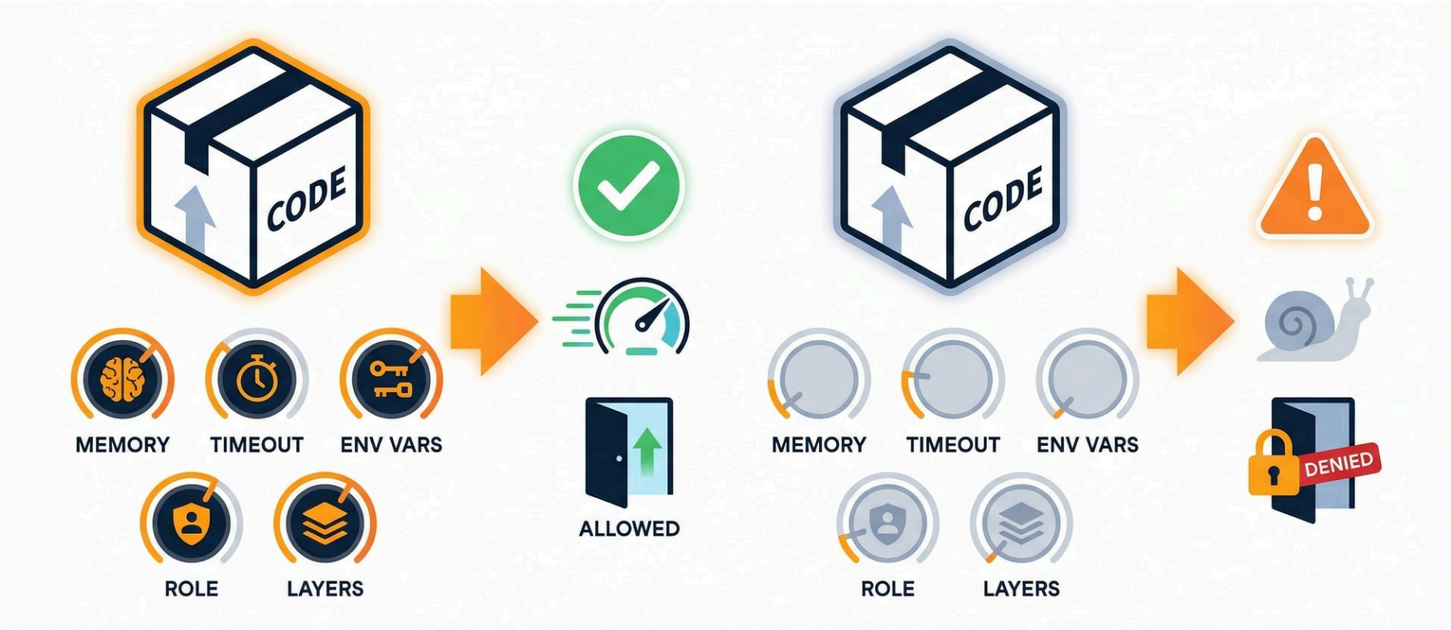
AWS LAMBDA

SERVERLESS APPLICATIONS

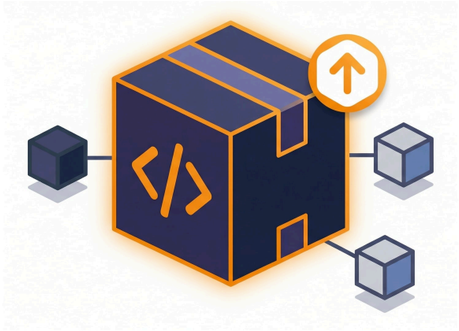


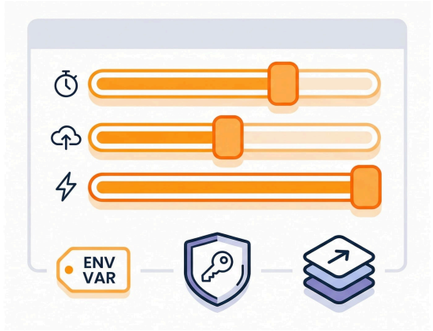
Claudio Canales
Senior DevOps Engineer

-
- , ,
- : dev prod


CODE
MEMORY
TIMEOUT
ENV VARS
ROLE
LAYERS
ALLOWED
CODE
MEMORY
TIMEOUT
ENV VARS
ROLE
LAYERS
DENIED

Code configuration





- **Code package**
- + dependencies.
- deploy

- **Function configuration**
- Memory, timeout, environment variables.
- Execution role + layers.

knobs

- **Runtime + handler:** - Lambda □□□□□□□
- **Memory + timeout:** , ,
- **Environment variables:** hardcoding
- **Execution role:** AWS APIs permissions□
- **Layers:** functions shared dependencies□



Runtime handler

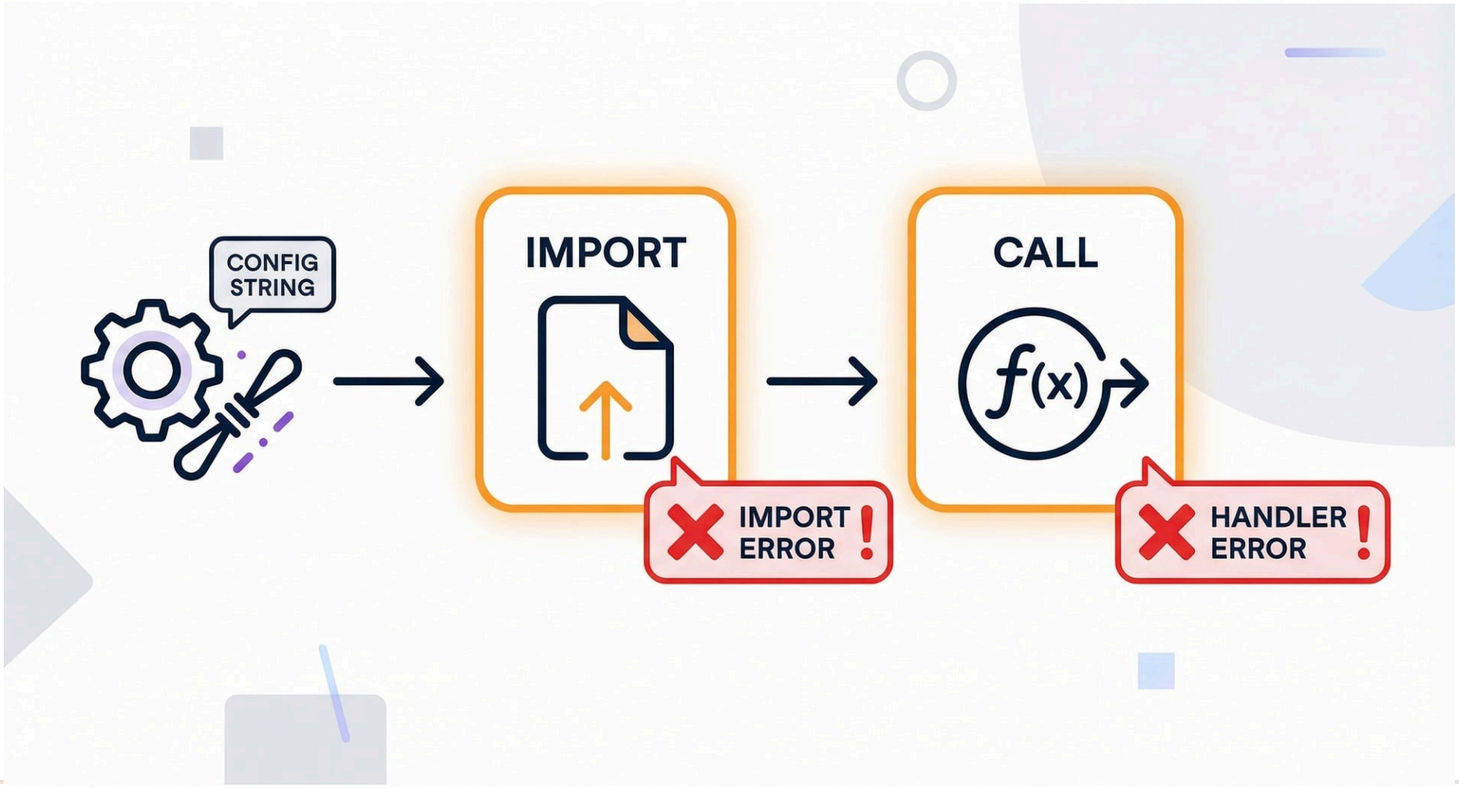
- Runtime = language environment (, Python).
- Handler = entry point Lambda

- Handler : `file.function` .
- : `lambda_function.lambda_handler` .



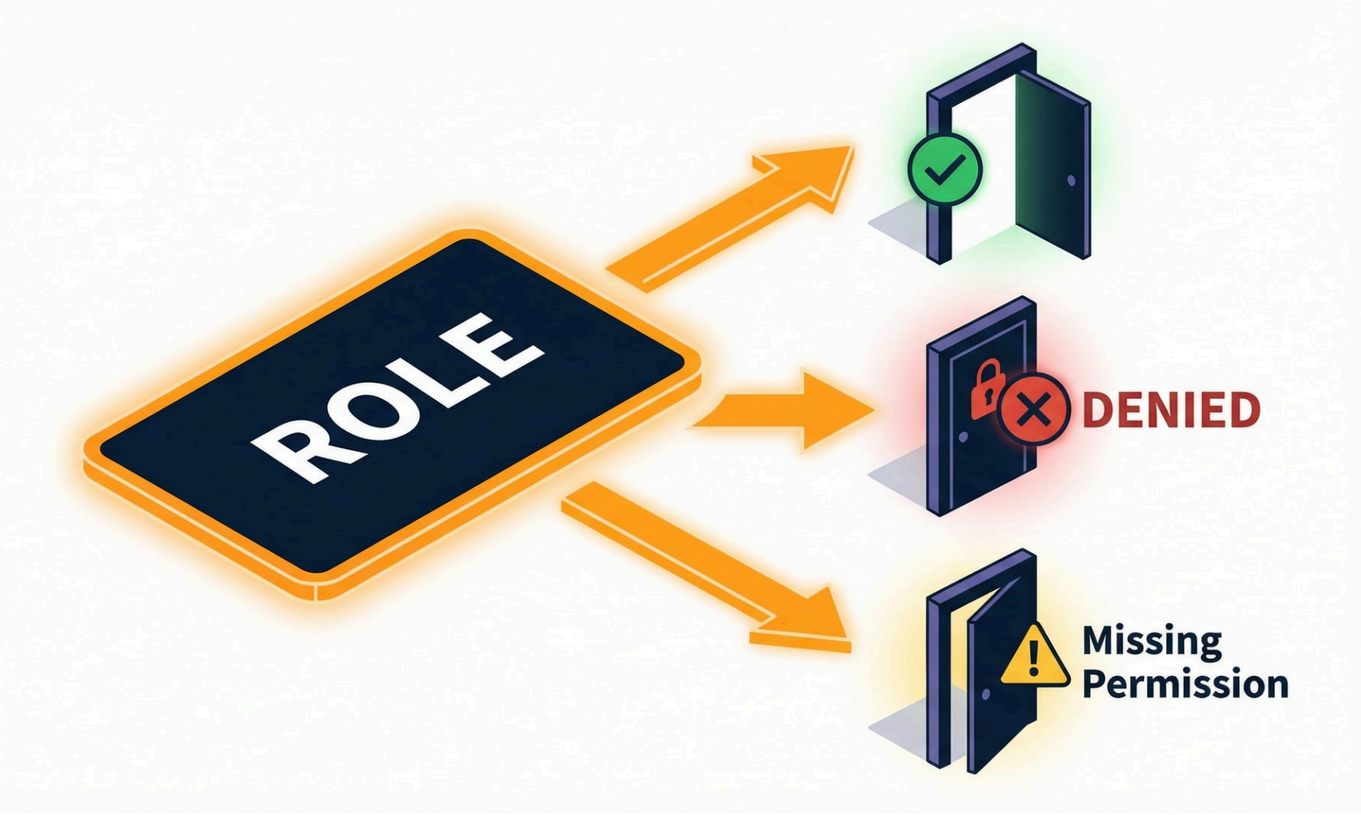
Handler string:

```
# lambda_function.py
def lambda_handler(event, context):
    return {"statusCode": 200}
```



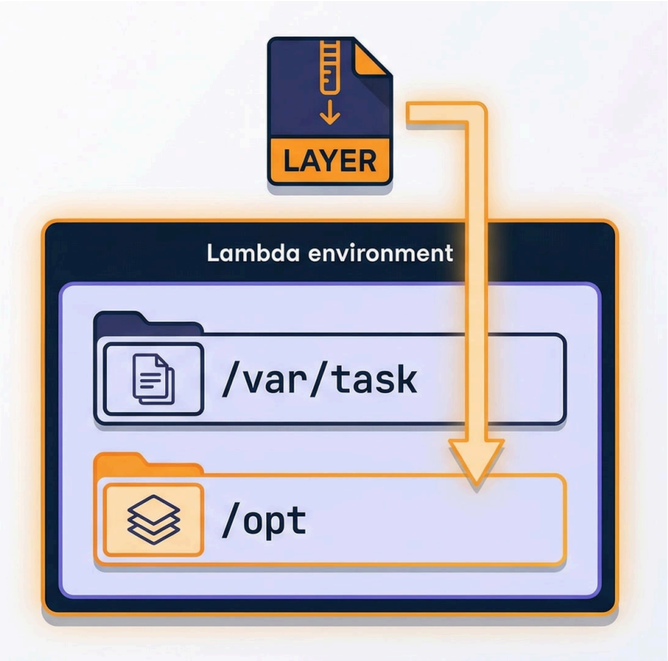
Execution role (IAM)

- IAM = Identity and Access Management (permissions).
- Execution role function
- - AWS APIs



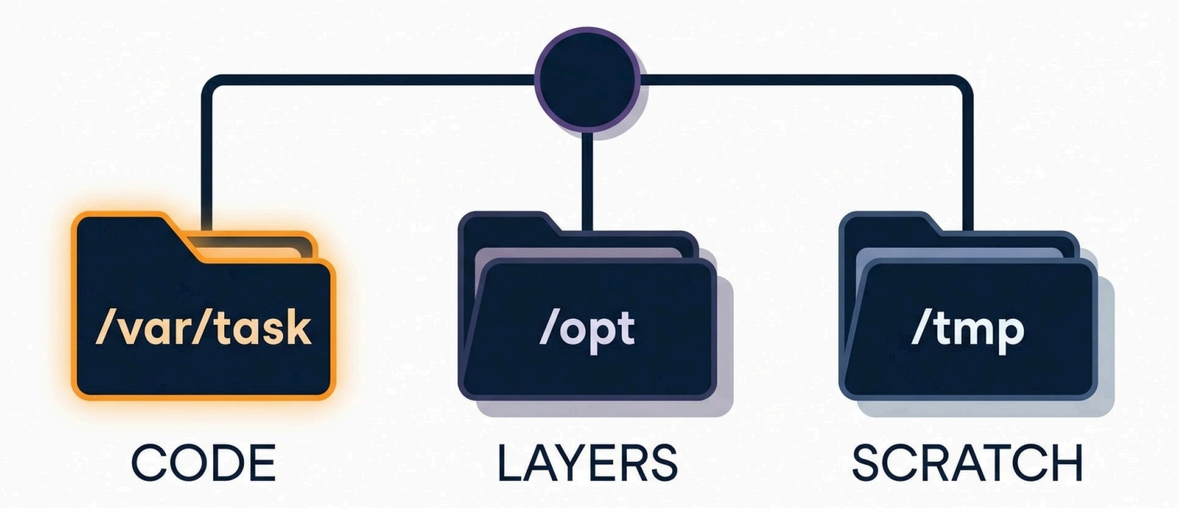
Lambda layer ?

- **Layer** ZIP function
- Lambda runtime layer `/opt`
- Shared libraries common utilities layers
- Layer ; functions reuse



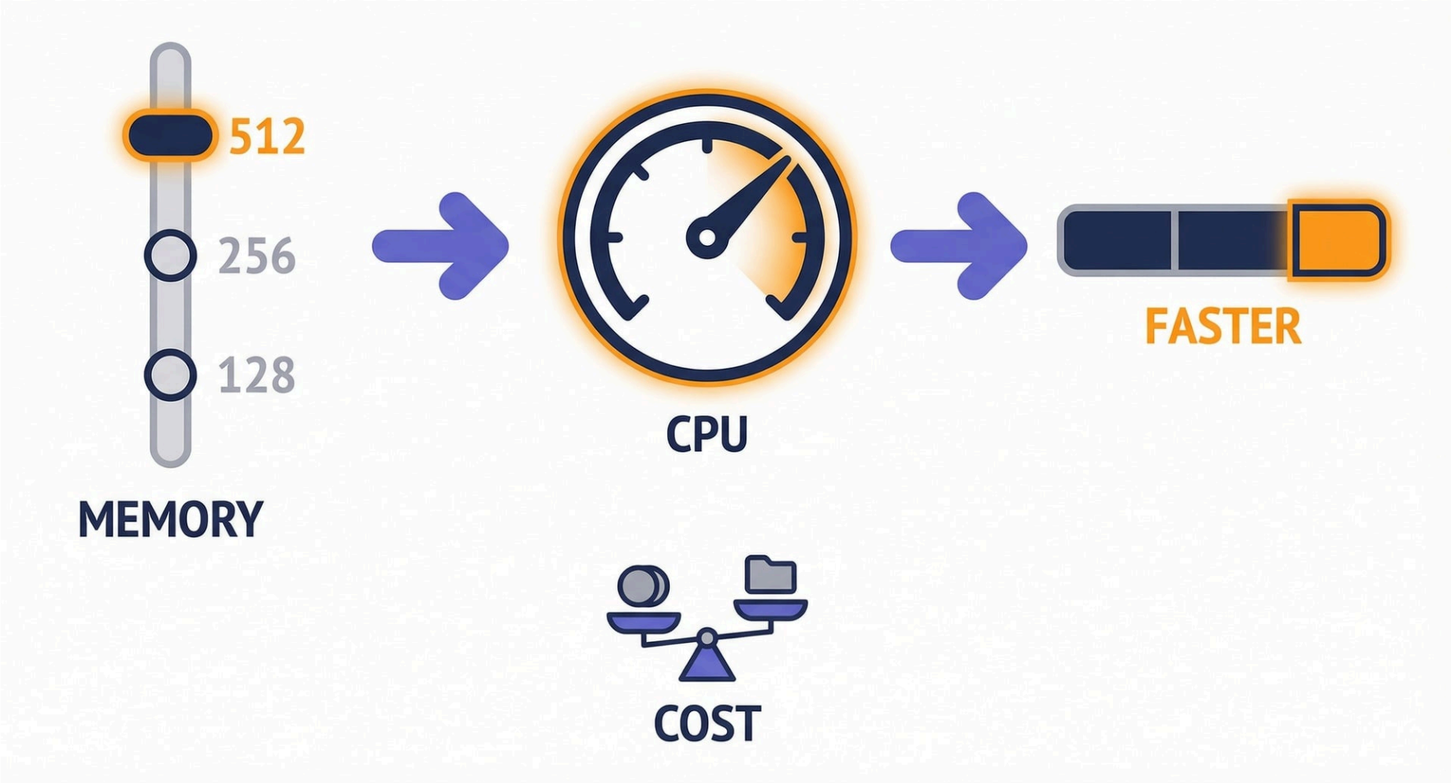
Runtime layers

- function `/var/task` mounted
- layers `/opt` mounted
- Lambda `/tmp`



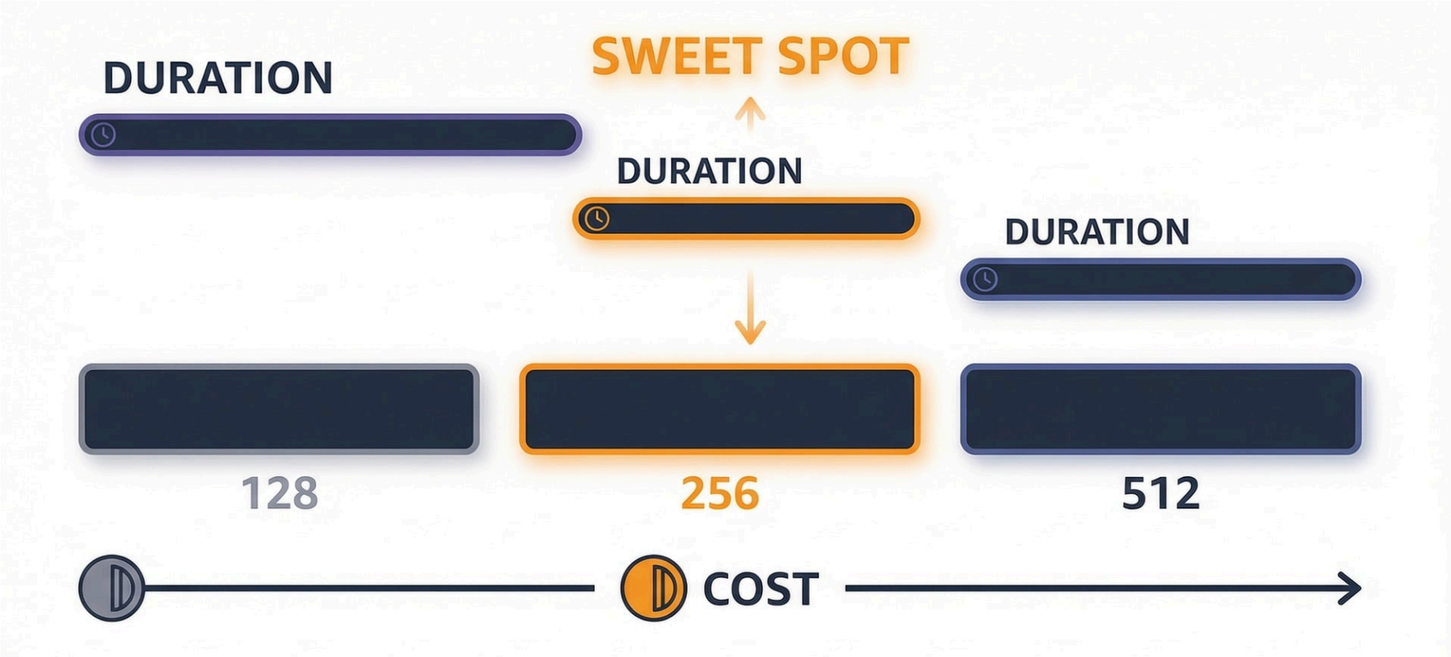
Memory CPU

- memory -> CPU
- CPU-bound tasks runtime
- duration memory



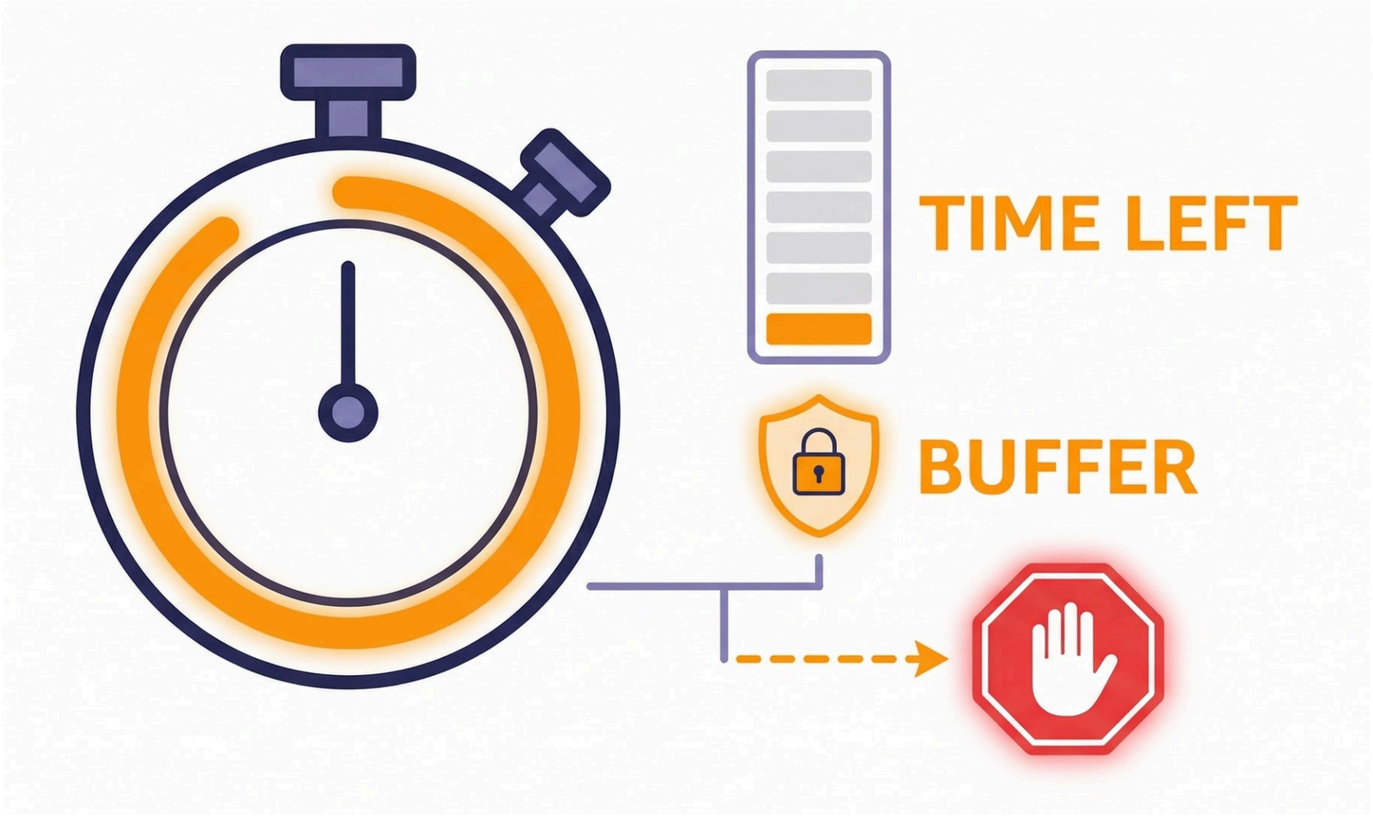
memory

- default (, 256 MB).
- events
- memory duration cost



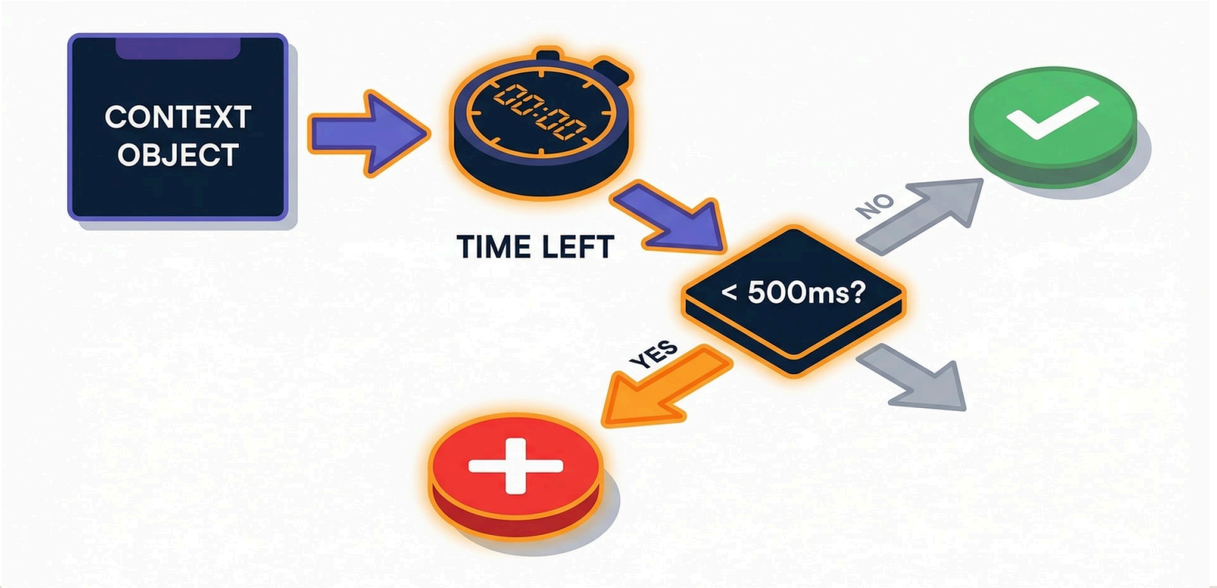
Timeouts execution

- Timeout executions
-
- , fail retry □□□□□



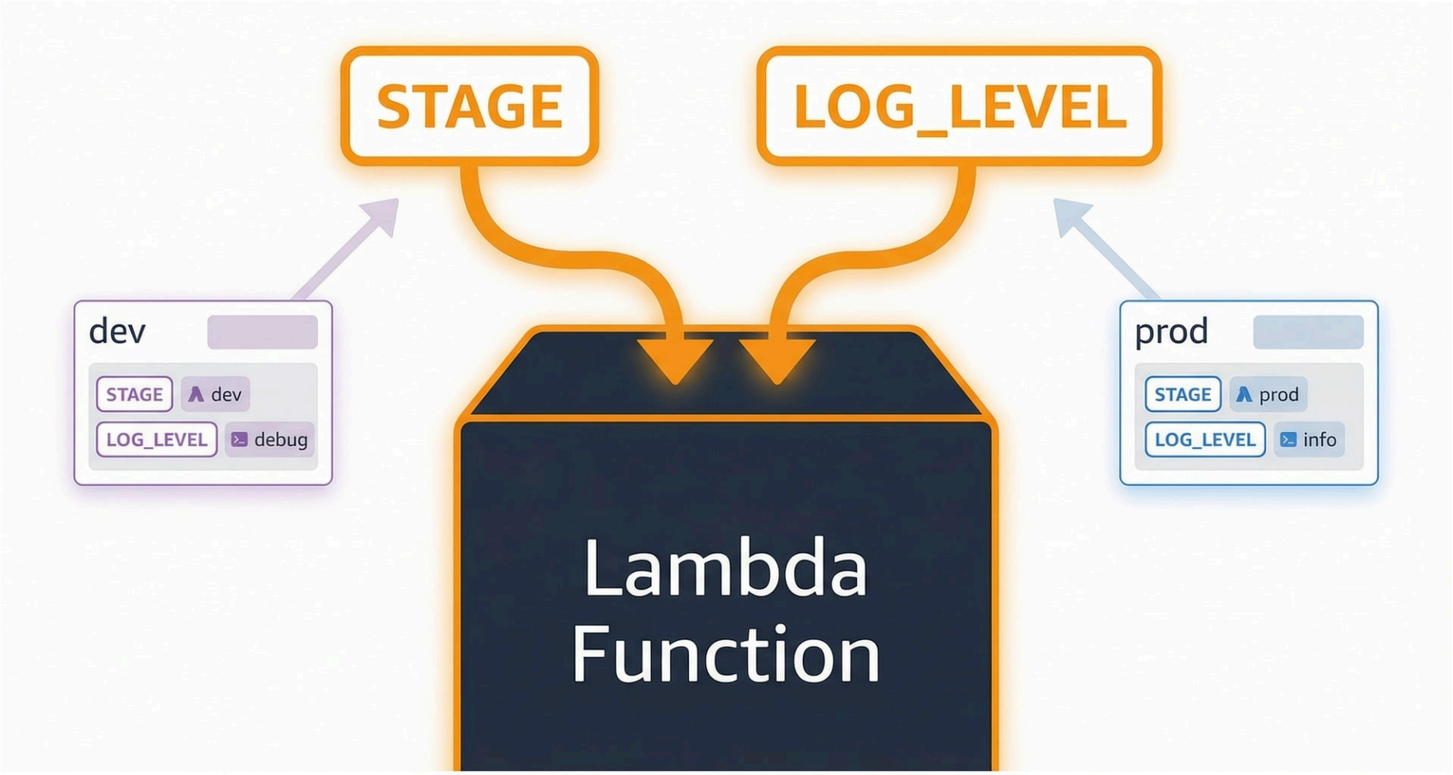
Walkthrough:

```
def lambda_handler(event, context):
    time_left = context.get_remaining_time_in_millis()
    if time_left < 500:
        raise TimeoutError("Not enough time left")
    return {"statusCode": 200}
```



Environment variables

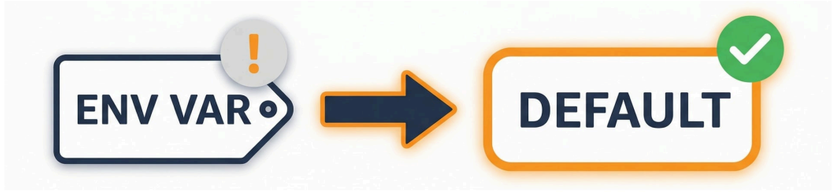
-
- dev/test/prod
- Secrets Manager



Walkthrough: environment variables

```
import os

STAGE = os.getenv("STAGE", "dev")
LOG_LEVEL = os.getenv(
    "LOG_LEVEL", "INFO"
)
```



- Environment variables `os`
- `os.getenv` default

pitfalls

- Stage hard-code □□□□□
- Timeout IAM permissions □□□□□
- Secrets plain environment variables



AWS LAMBDA

SERVERLESS APPLICATIONS !

Lambda

AWS LAMBDA SERVERLESS APPLICATIONS



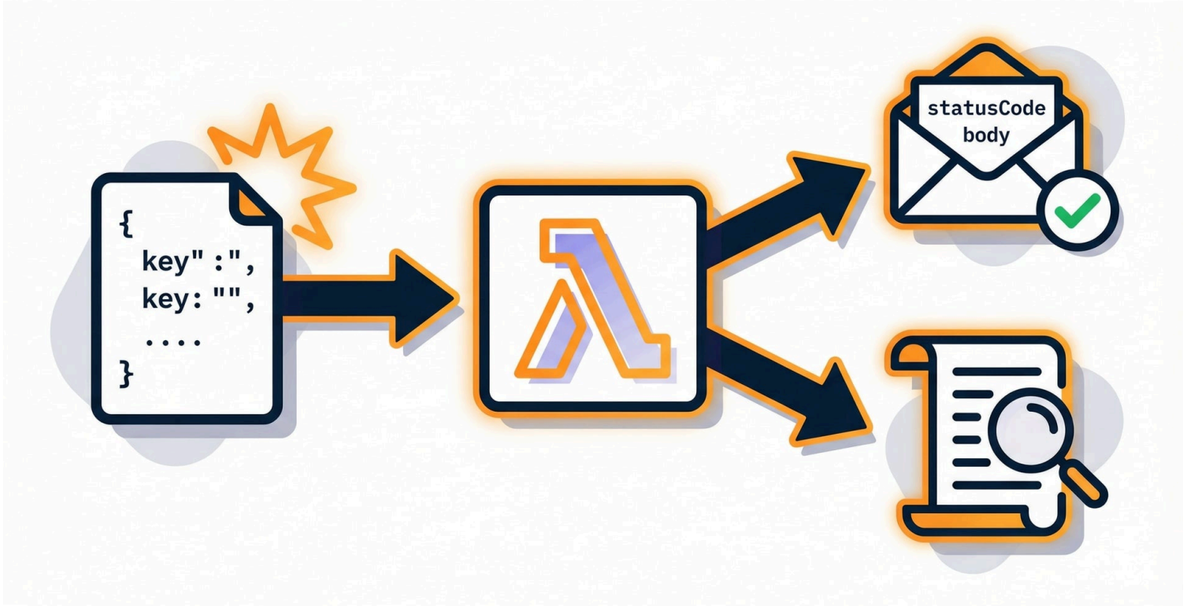
Claudio Canales
Senior DevOps Engineer

-
-
- " "


FIRE DRILL
OUTCOME

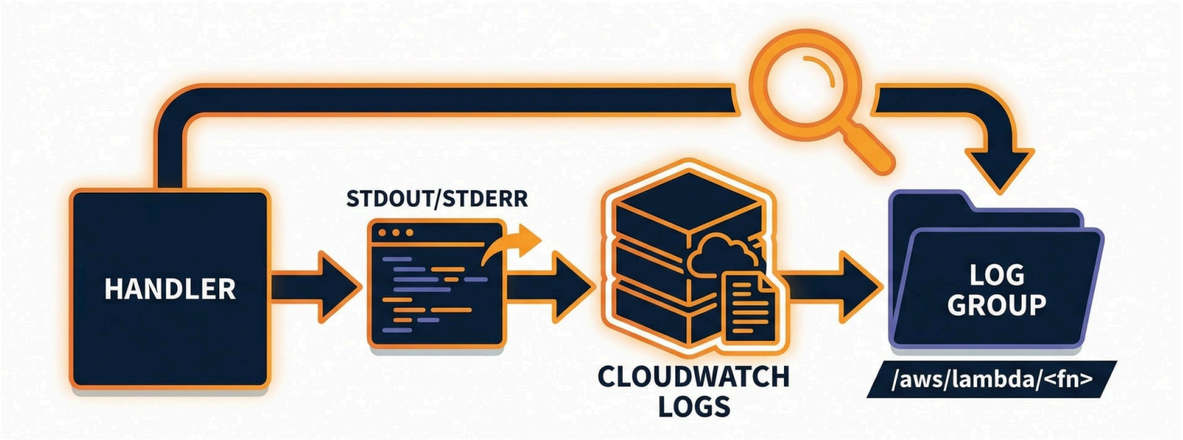


- Event in -> handler runs -> response out.
- +



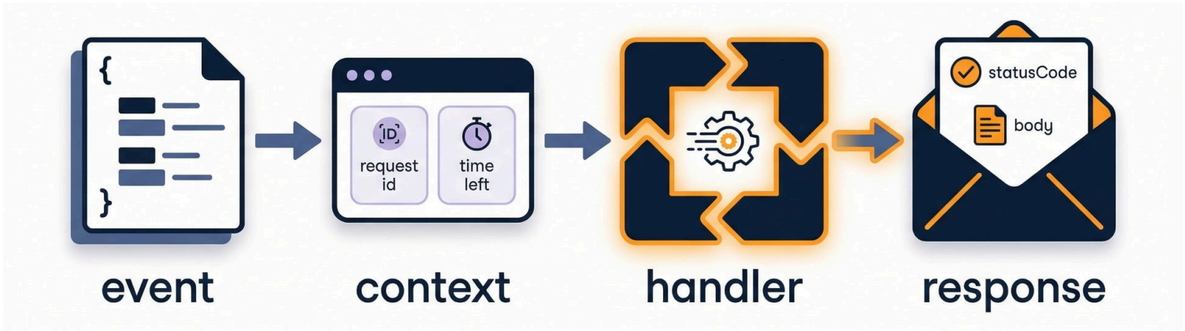
: CloudWatch Logs

- Lambda `print()`
- CloudWatch Logs, AWS
- :
  `/aws/lambda/<function-name>`



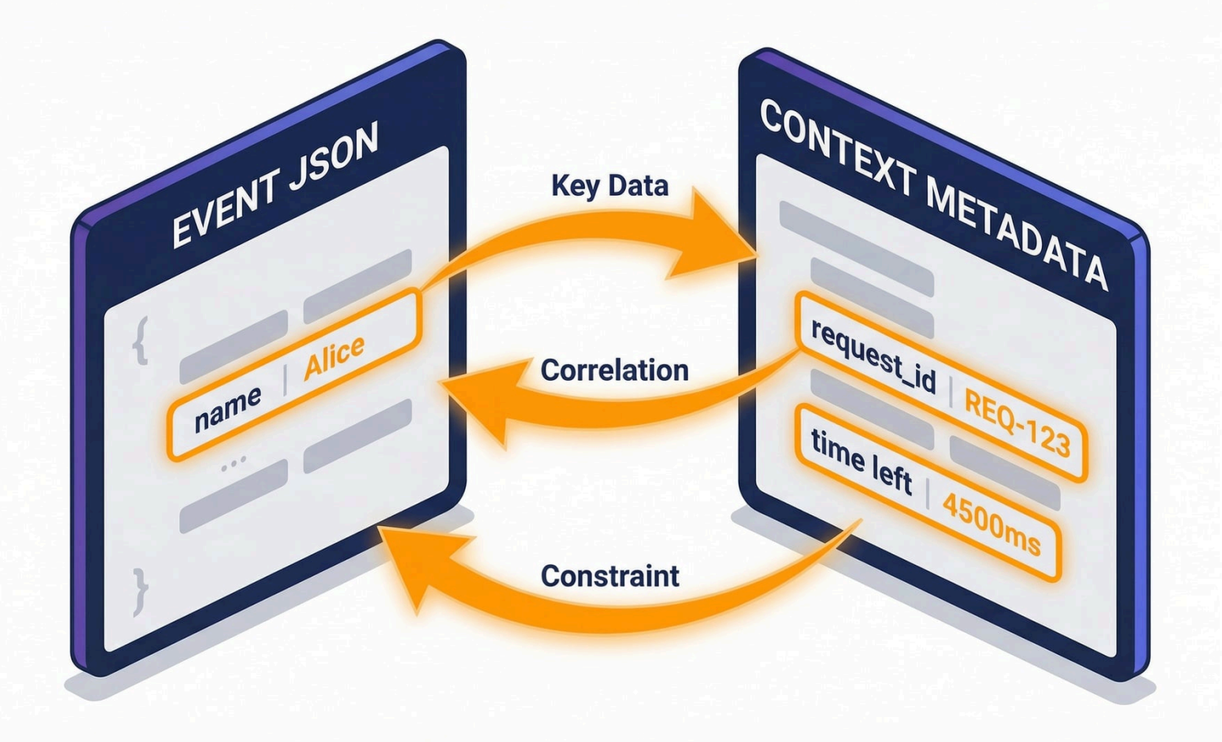
- `event`
- `context`
-

response



: event context

```
def lambda_handler(event, context):
    request_id = context.aws_request_id
    time_left = context.get_remaining_time_in_millis()
    return {"statusCode": 200}
```

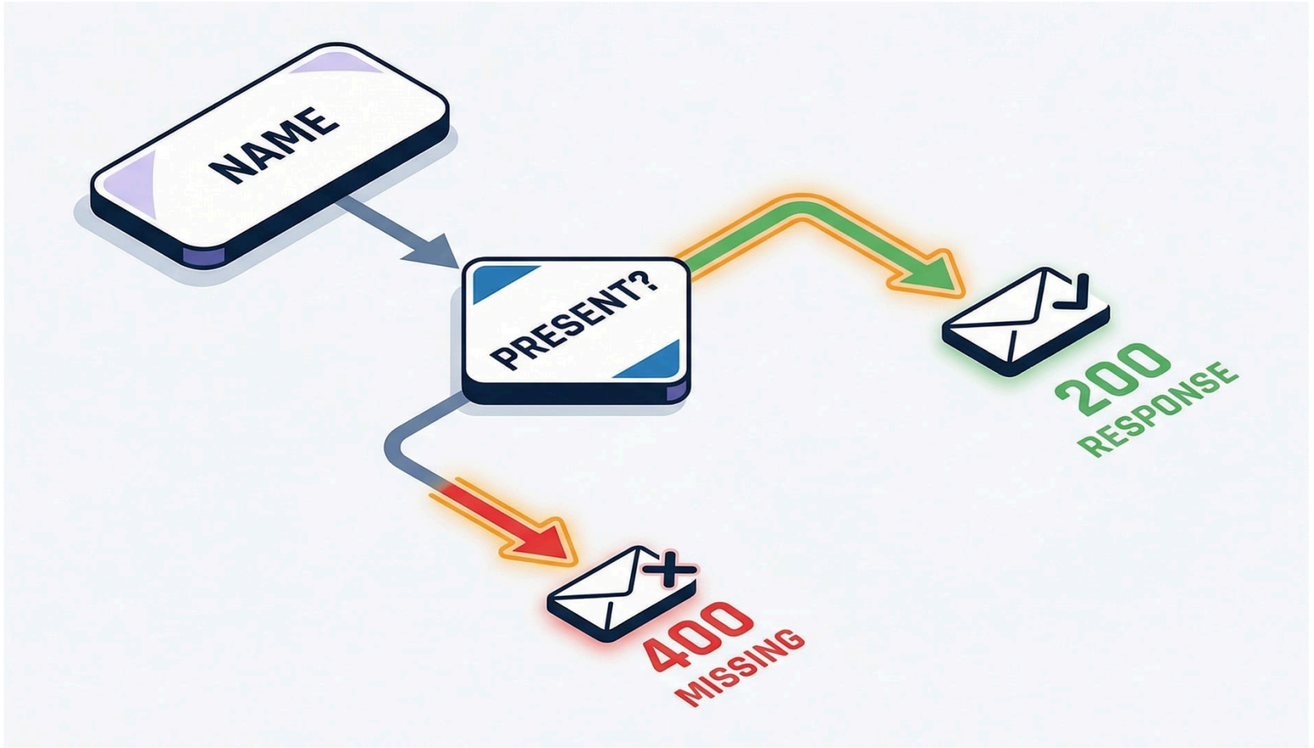


- `event.get()`
-
-

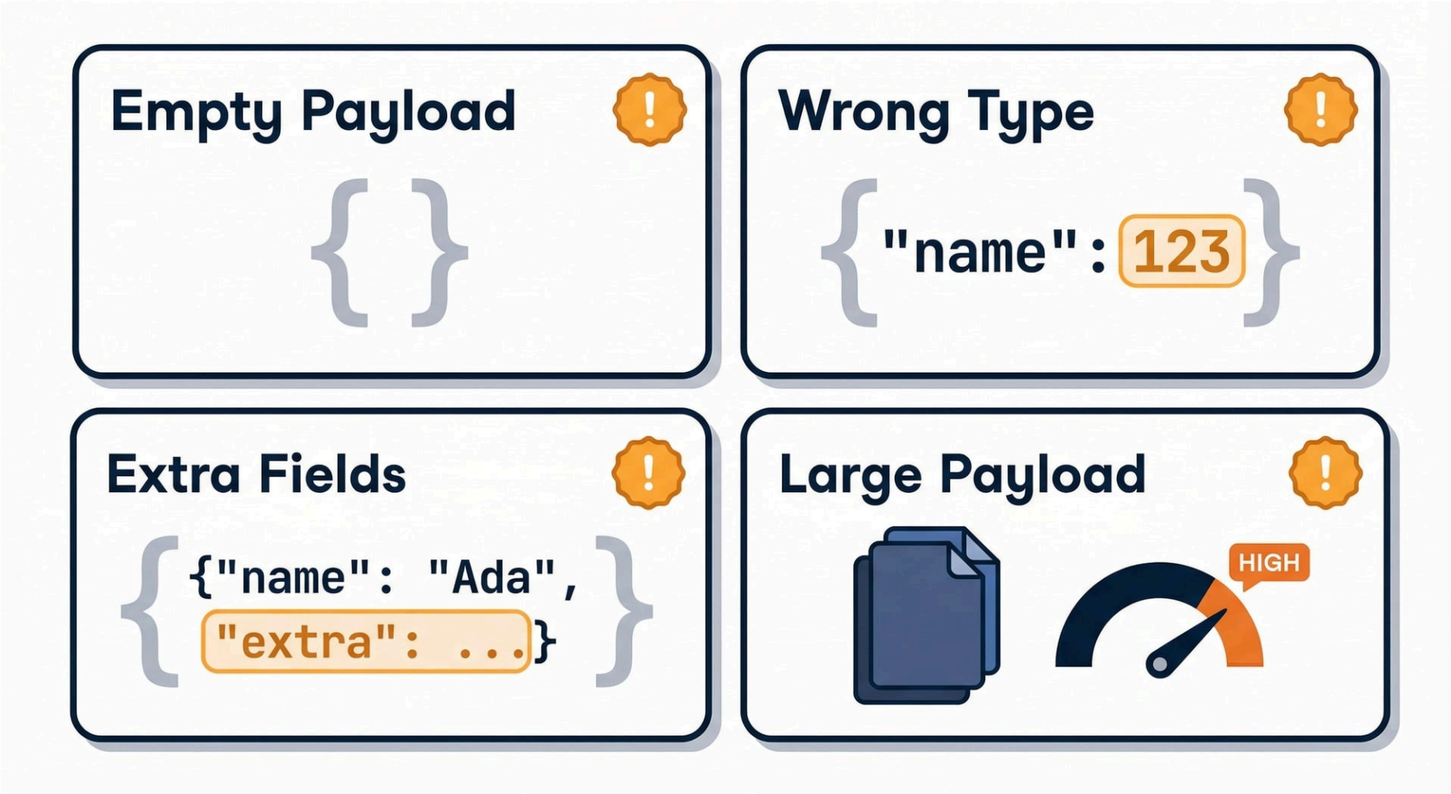


:

```
def lambda_handler(event, context):
    if not event.get("name"):
        return {"statusCode": 400, "body": "Missing: name"}
    return {"statusCode": 200, "body": f"Hello, {event.get('name')}!"}
```



- (JSON)
- (name)
-


Empty Payload
{ }
Wrong Type
{"name": 123}
Extra Fields
{"name": "Ada",
"extra": ...}
Large Payload
HIGH

response □□□□□□

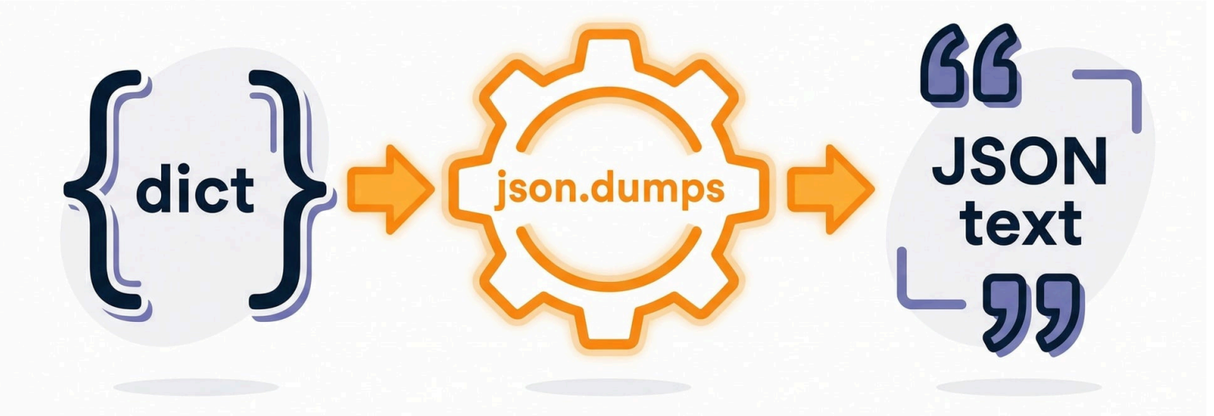
- response
- `statusCode`
- response body JSON-



: JSON □□□□□□

```
import json

def lambda_handler(event, context):
    body = {"message": "ok"}
    return {"statusCode": 200, "body": json.dumps(body)}
```



- (
  JSON)
- `context.aws_request_id`
-
- Lambda
  CloudWatch Logs



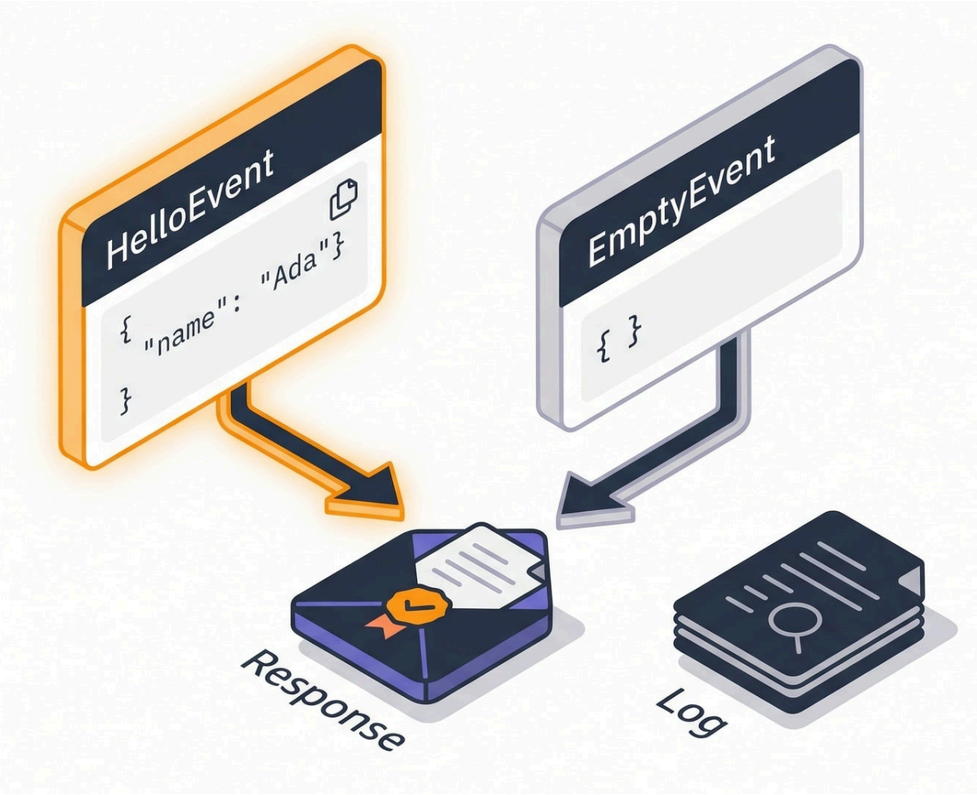
:

```
import json

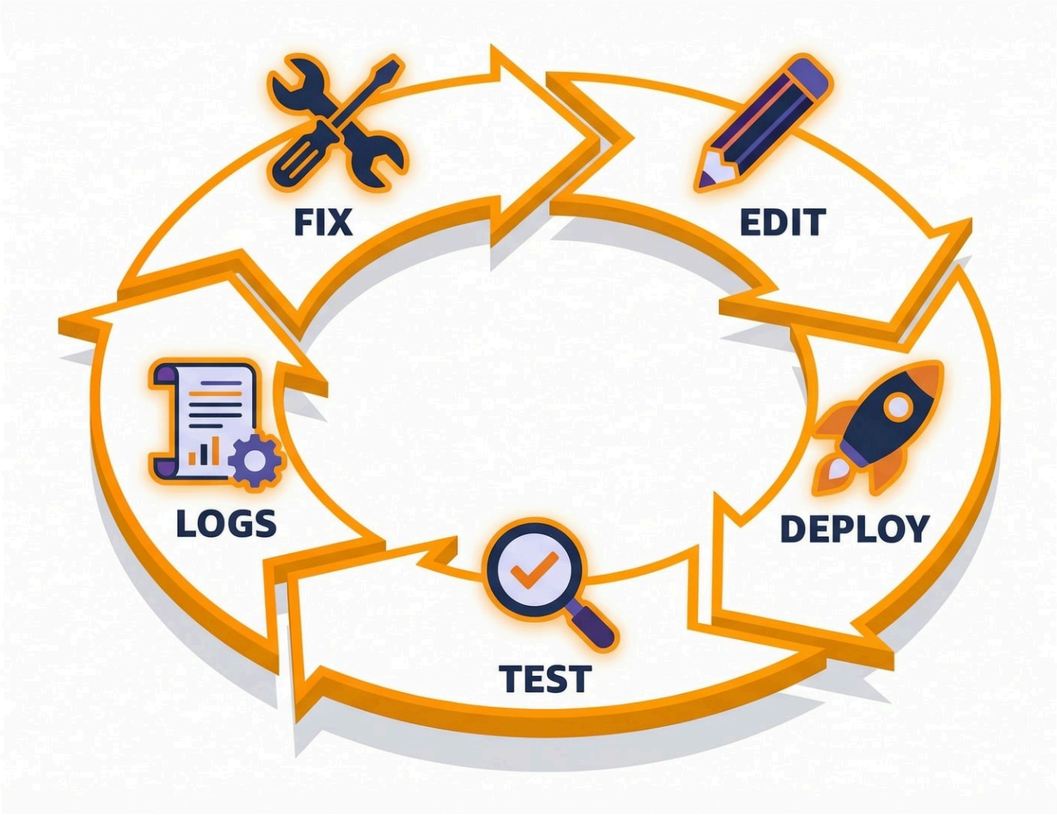
def lambda_handler(event, context):
    print(json.dumps({"request_id": context.aws_request_id, "event": event}))
    return {"statusCode": 200}
```



-
-
- response +



-
- ,
- response CloudWatch Logs □□□□□□



AWS LAMBDA

SERVERLESS APPLICATIONS !