AWS Serverless : Lambda

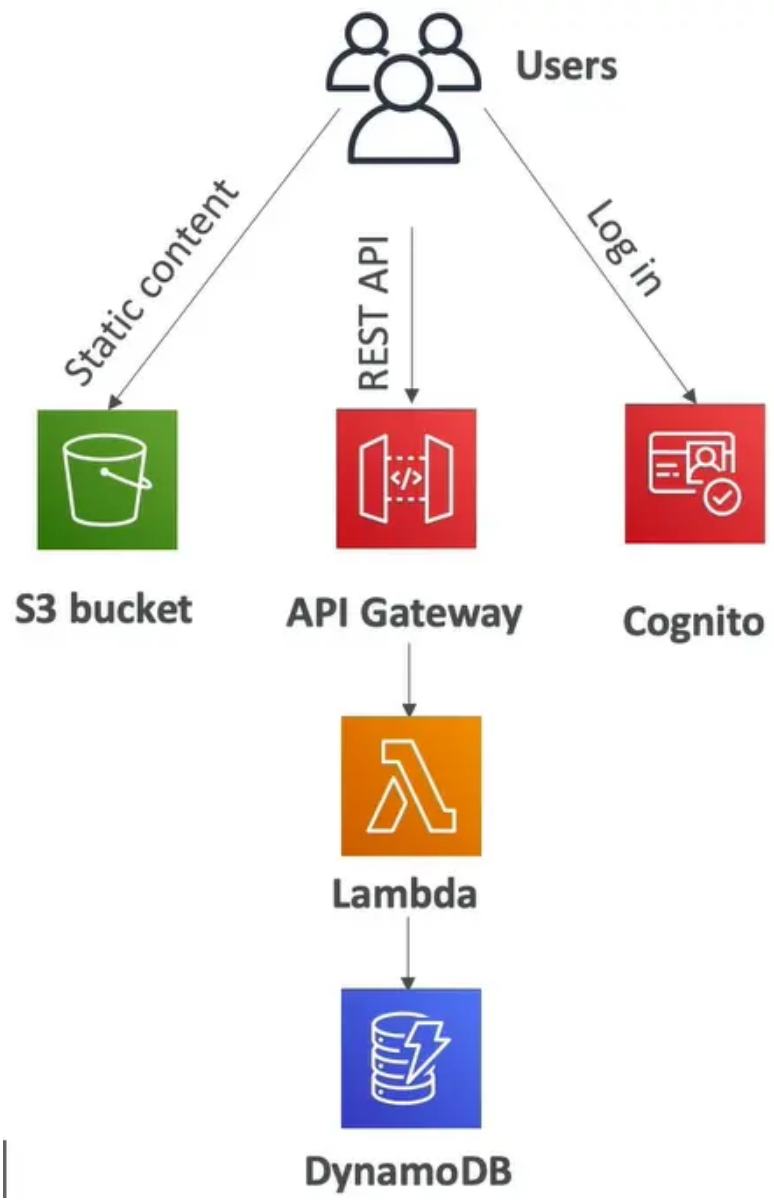
Lambda

what's serverless ?

- serverless is a new paradigm in which the developers don't have to manage servers anymore
- they just deploy code
- they just deploy function '
- initially [最初].. Serverless == FAAS (function as a Service)
- Serverless was poineered [创业] by AWS Lambda but now also includes anything that's managed:"database,messaing,storage,etc..."
- Serverless does not mean there are no servers

it means you just don't manage / provision /see them

Serverless in AWS



- AWS Lambda
- DynamoDB
- AWS Cognito
- AWS API Gateway
- Amazon S3
- AWS SNS & SQS
- AWS Kinesis Data Firehose
- Aurora Serverless
- Step Function
- Fargate

Lambda Overview

Amazon EC2

- virtual servers in the cloud
- limited by RAM and CPU
- continuously running
- scaling means intervention [介入] to add / remove servers

Amazon Lambda

- Virtual functions – no servers to manage !
- limited by time – short executions
- run on–demand
- scaling in automated

Benefits of AWS Lambda

- easy pricing
  - pay per request and compute time
  - free tier of 1,000,000 AWS Lambda requests and 400,000 GBs of compute time

- integrated with the whole AWS suite of services
- integrated with many programming languages
- easy monitoring through AWS CloudWatch

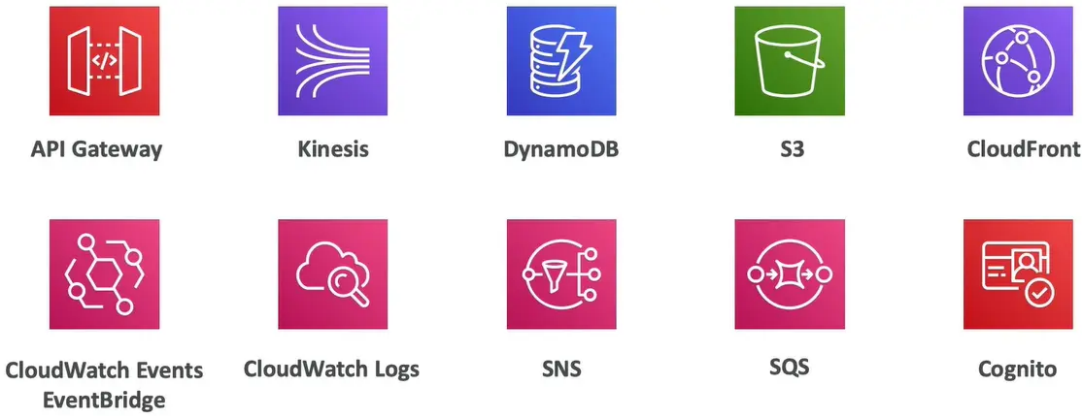
- easy to get more resources per functions (up to 10GB of RAM)
- increasing RAM will also improve CPU and network

AWS Lambda language support

- node.js (javascript)
- python
- java (java 8 compatible)
- C# (.NET Core)
- Golang
- C# / Powershell
- ruby
- custom runtime API (community supported, example Rust)

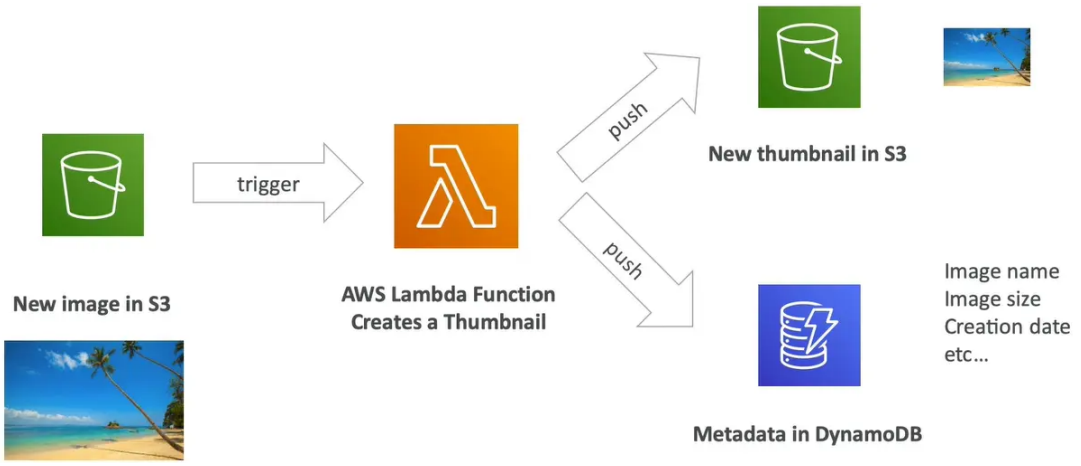
- Lambda Container Image
  - the container image must implement the Lambda Runtime API
  - ECS / Fargate is preferred for running arbitrary [任意的] Docker images

aws Lambda intergrations main ones

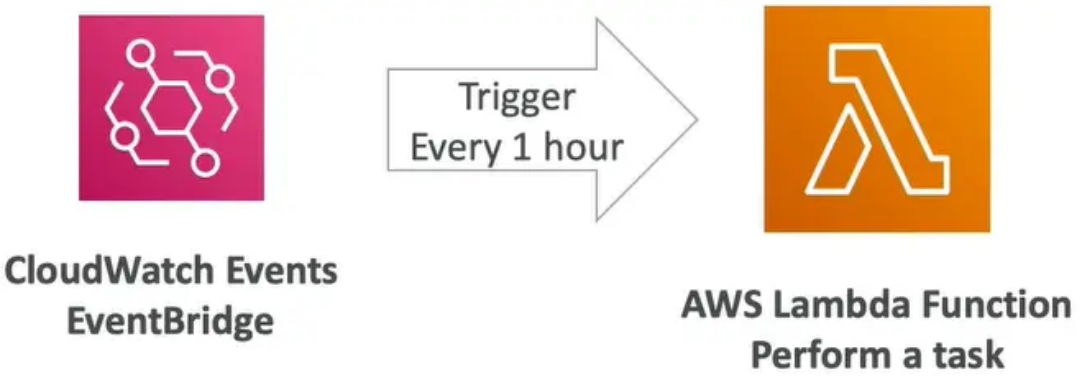


Example:

serverless thumbnail creation



Serverless CRON Job

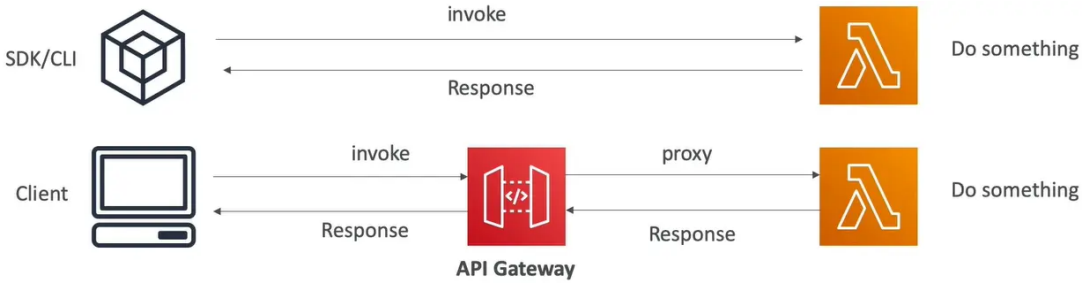


AWS Lambda Pricing: example

- you can find overall pricing information here : https://aws.amazon.com/lambda/pricing/
- pay per calls:
  - first 1,000,000 requests are free
  - $0.20 per 1 million requestss thereafter ($0.0000002 per request)
- pay per duration [持续时间] (in increment of 1ms [以1ms为增量])
  - 400,000GB seconds of compute time per month if FREE
  - ==400,000 seconds if function is 1 GB RAM
  - == 3,200,000 seconds if function is 128 MB RAM
  - after that $1.00 for 600,000 GB-seconds
- it is usually very cheap to run AWS Lambda so it't very popular

Lambda –Synchronous Invocations [同步调用]

- Synchronous : CLI, SDK,API Gateway,Application Load Balancer
  - result is returned right away
  - errror handling must happen client side (retries, exponential backoff, etc..)



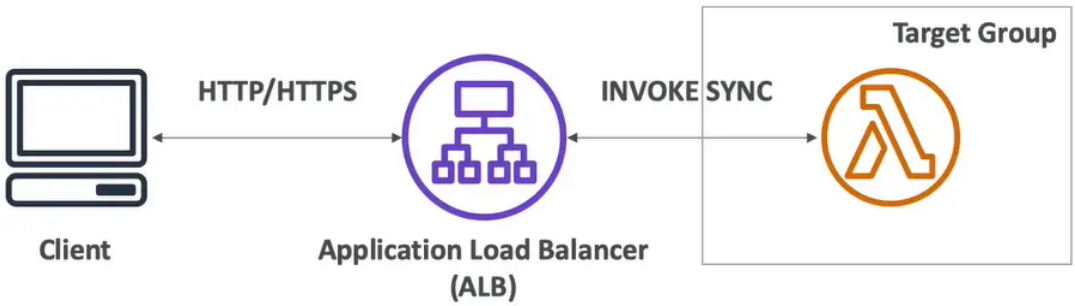
Sychronous Invocations – Service

- User Invoked:
  - Elastic Load Balancing (application load balancer)
  - Amazon API Gateway
  - Amazon CloudFront (Lambda@Edge)
  - Amazon S3 Batch
- Service Invoked
  - Amazon Cognito
  - AWS Step Functions
- Other Services

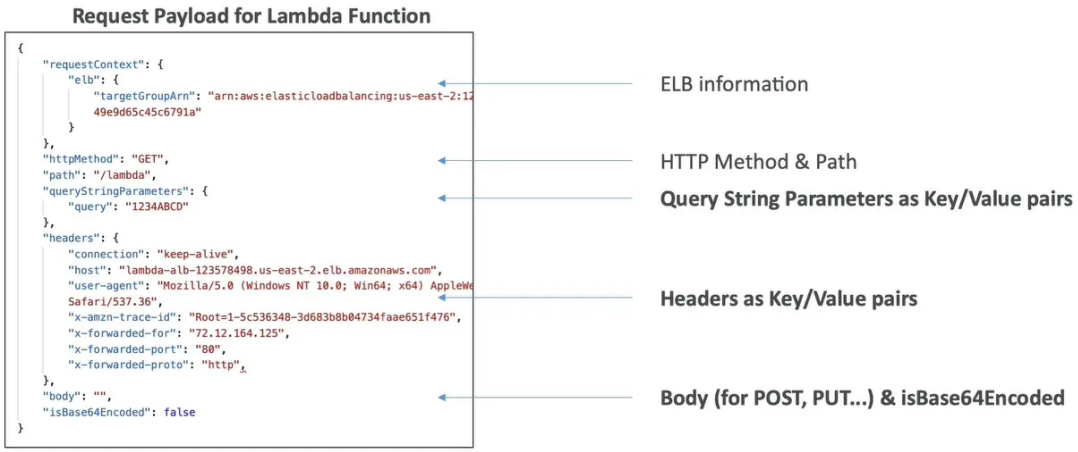
- Amazon Lex
- Amazon Alexa
- Amazon Kinesis Data Firehose

Lambda Integration with ALB

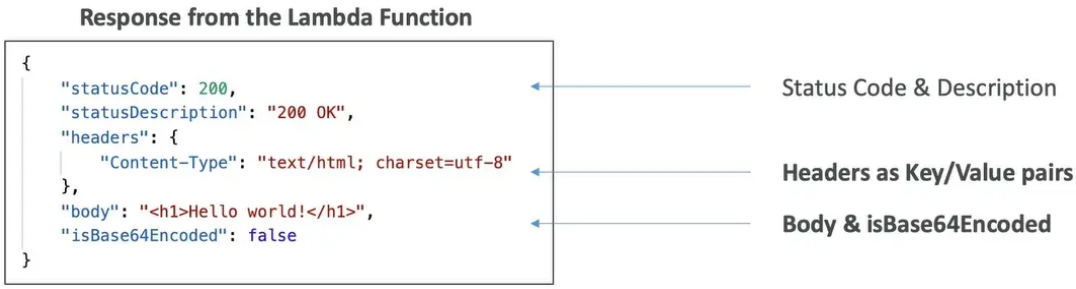
- to expose a Lambda function as an HTTP(s) endpoint
- you can use the application load balancer (or an API Gateway)
- the Lambda function must be registered in a target group



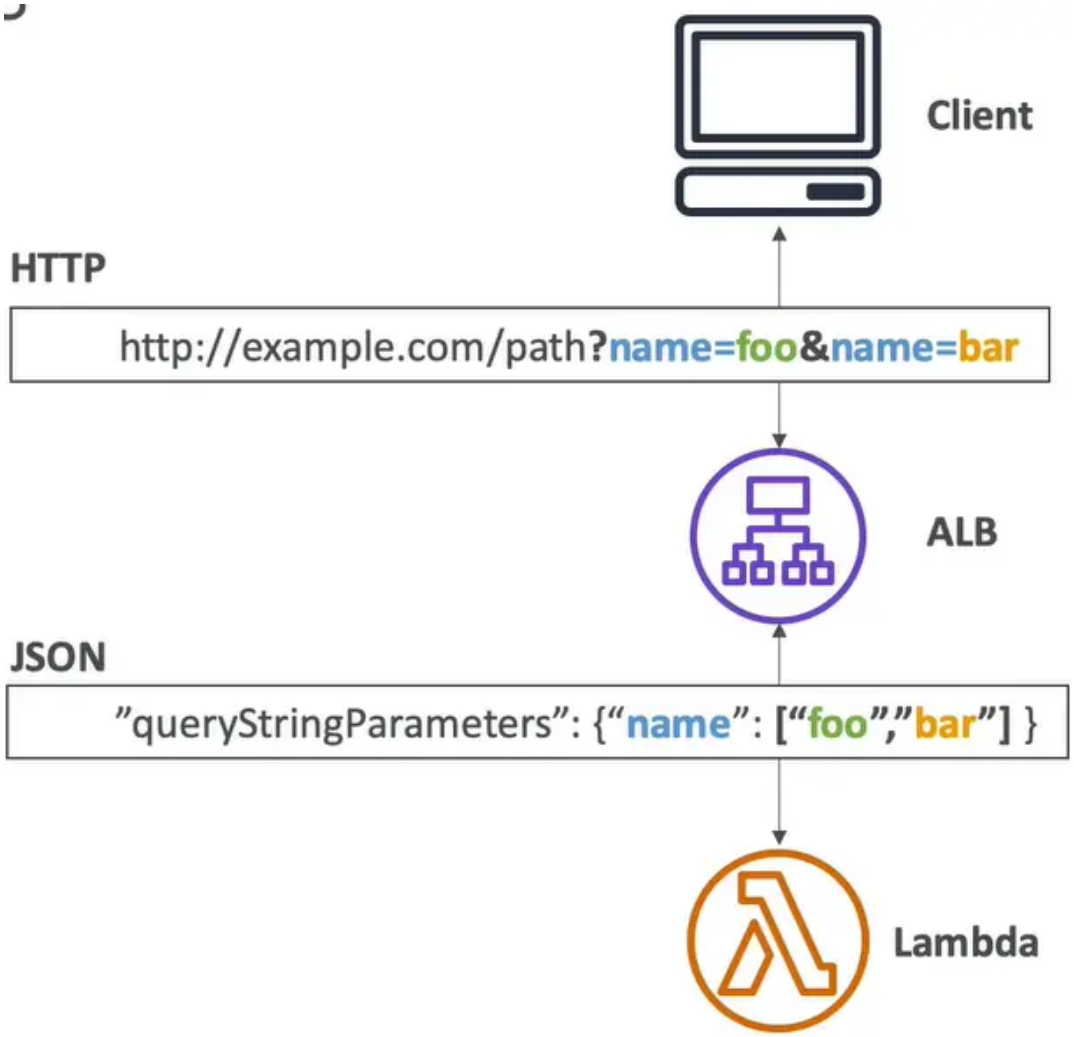
ALB to Lambda: HTTP to JSON



Lambda to ALB conversions : JSON to HTTP

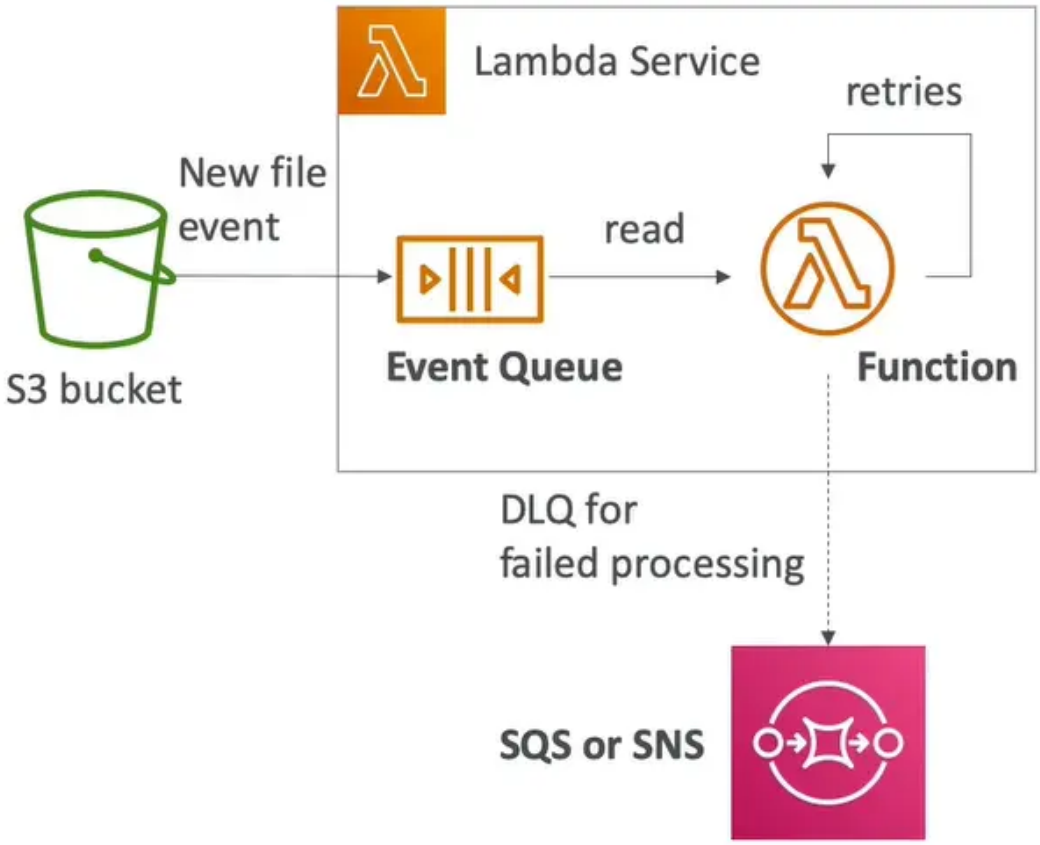


ALB Multi-Header Values



- ALB can support multi header values (ALB setting)
- when you enable multi-value headers,HTTP headers and query string parameters that are sent with multiple values are shown as arrays within the AWS Lambda event and response objects

Asynchronous Invocations [异步调用]



- S3,SNS, CloudWatch Events..
- the events are placed in an **Event Queue**
- Lambda attempts to retry on errors
  - 3 tries total
  - 1 minute wait after 1st, then 2 minutes wait
- make sure the processing is **idempotent**［幂等的］(in case of retries)
- if the function is retried, you will see **duplicate logs entries in CloudWatch Logs**
- can define a DLQ (dead-letter queue) – **SNS or SQS** – for failed processing (need correct IAM permissions)
- asynchronous invocations allow you to speed up the processing if you don't need to wait for the result (ex:you need 1000 files processed)

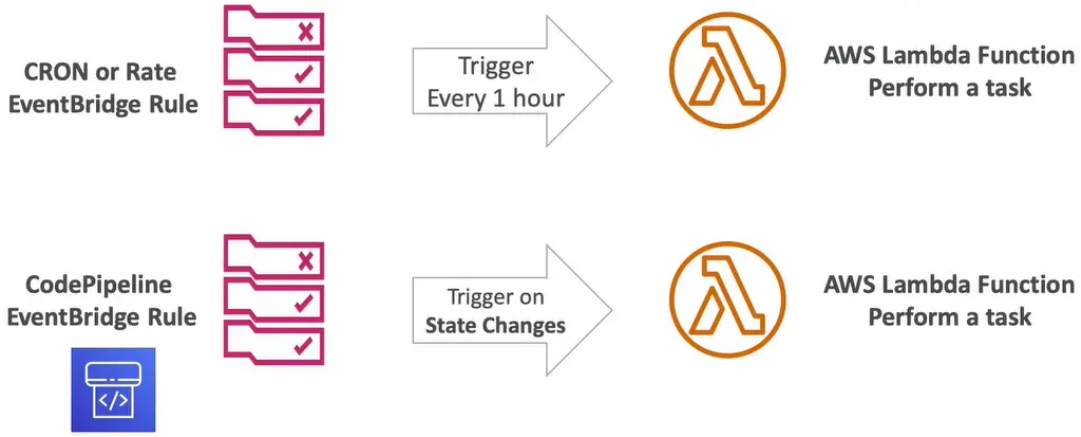
Asynchronous Invocations – Services

- **Amazon Simple Storage Service (S3)**
- **Amazon Simple Notification Service (SNS)**
- **Amazon CloudWatch Events / EventBridge**
- AWS CodeCommit (CodeCommit Trigger : new brance,new tag, new push)
- AWS CodePipeline (invoke a Lambda function during the pipeline, Lambda must callback)

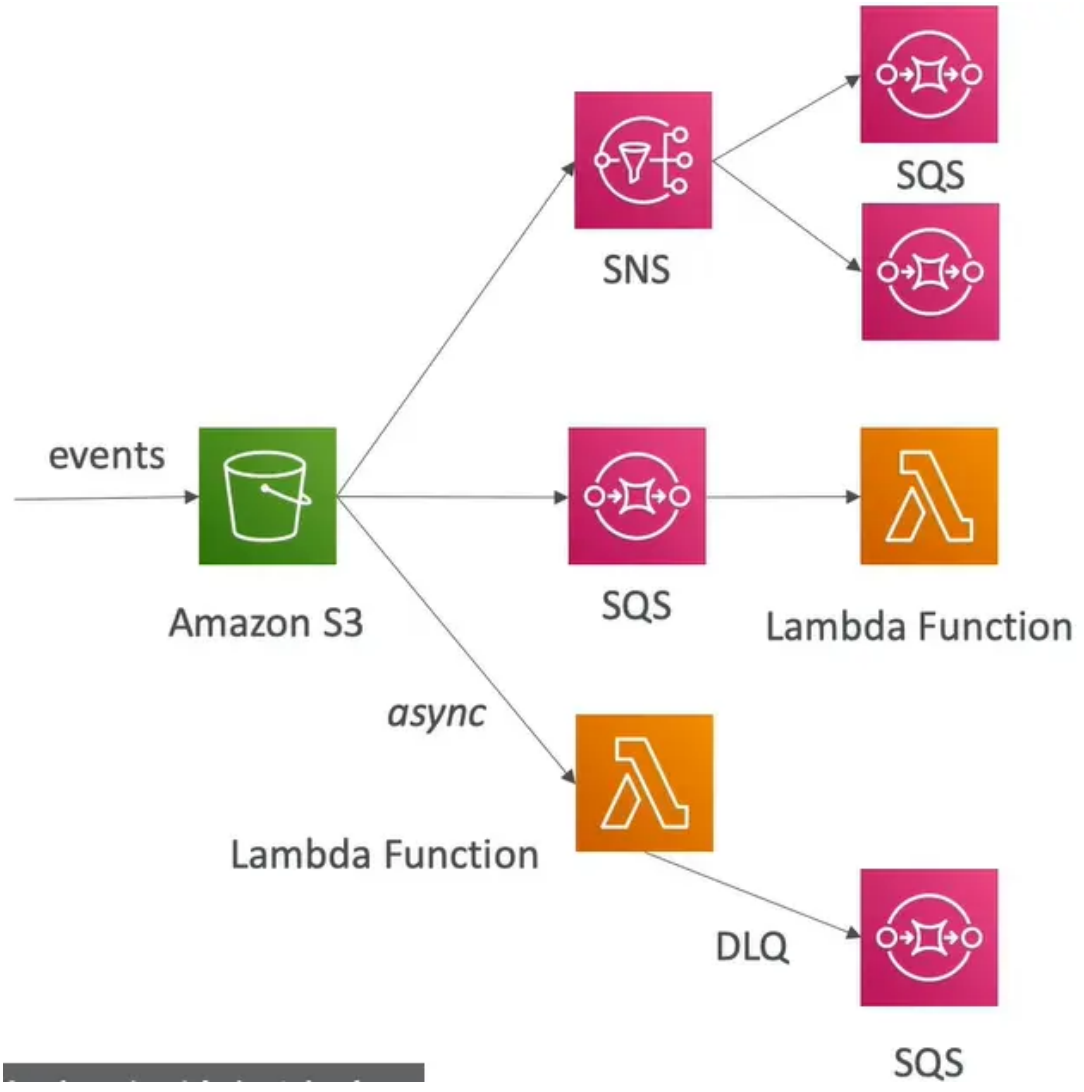
--------other---------

- Amazon CloudWatch Logs (log processing)
- Amazon Simple Email Service
- AWS CloudFormation
- AWS Config
- AWS IoT
- AWS IOT Events

CloudWatch Events / EventBridge



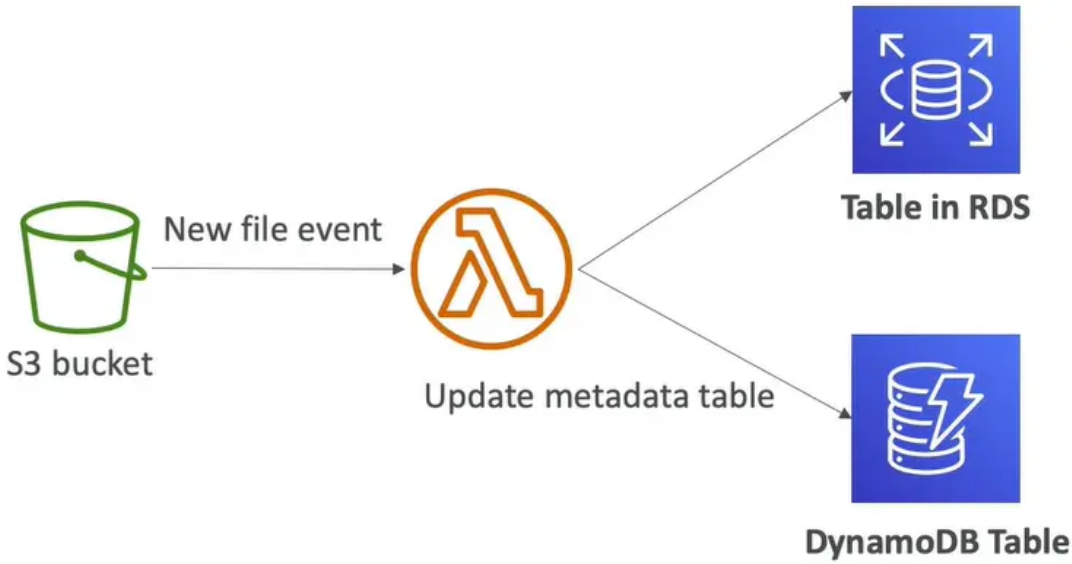
S3 Events Notifications



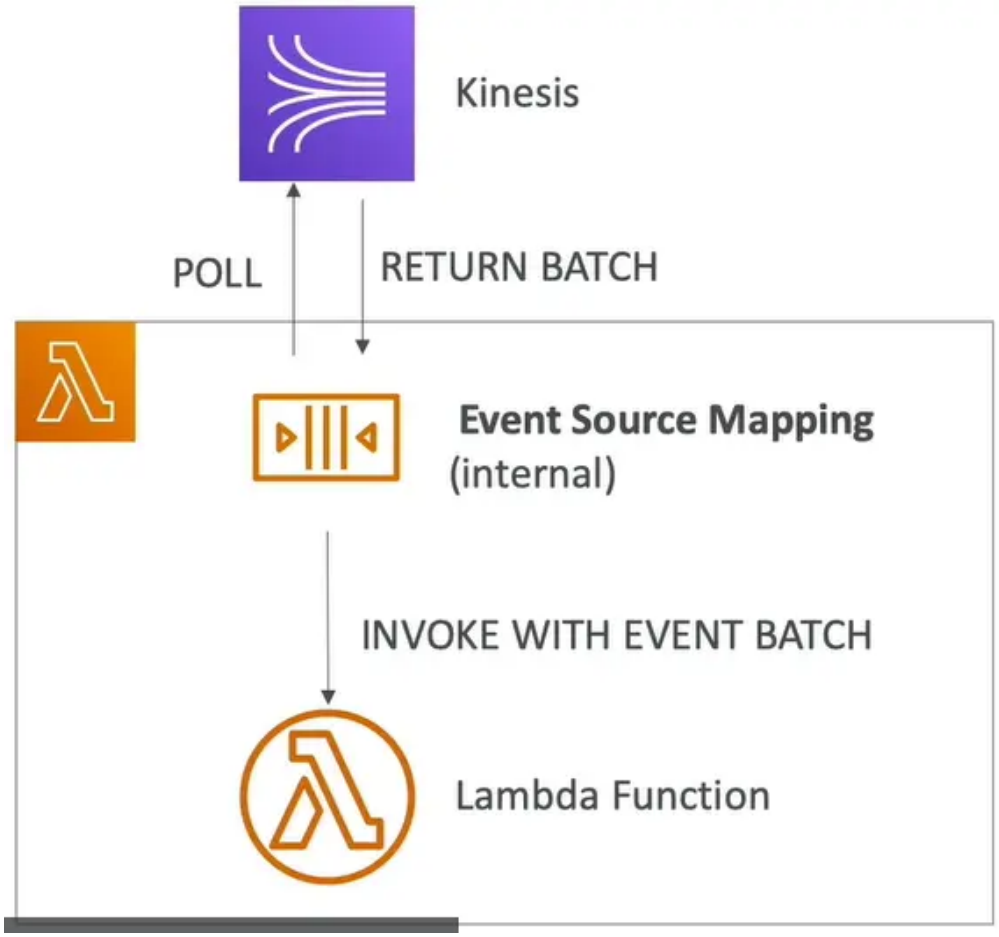
- S3: ObjectCreated, S3: ObjectRemoved,S3:ObjectRestore,S3 :Replication [复制]
- Object name filtering possible (*.jpg)
- use case : generate thumbnails of images uploaded to S3
- S3 event notification typically deliver events in seconds but can sometimes take a minute or longer
- if two writes are made to a single non-versioned object at the same time,it is possible that only a single event notification will be sent

- if you want to ensure that an event notification is sent for every successful write, you can enable versioning on your bucket

simple S3 event pattern – metadata sync



Event Source Mapping

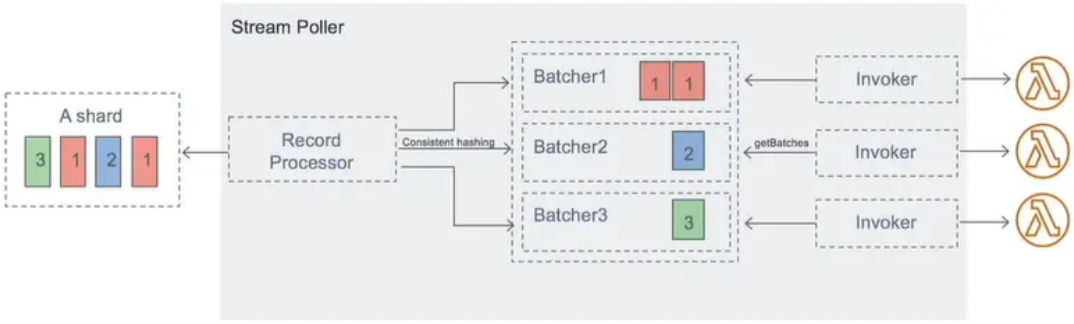


- Kinesis Data Streams
- SQS & SQS FIFO queue
- DynamoDB Streams

- common denominator: records need to be polled from the source
- your lambda function is invoked synchronously

Streams & Lambda (Kinesis & DynamoDB)

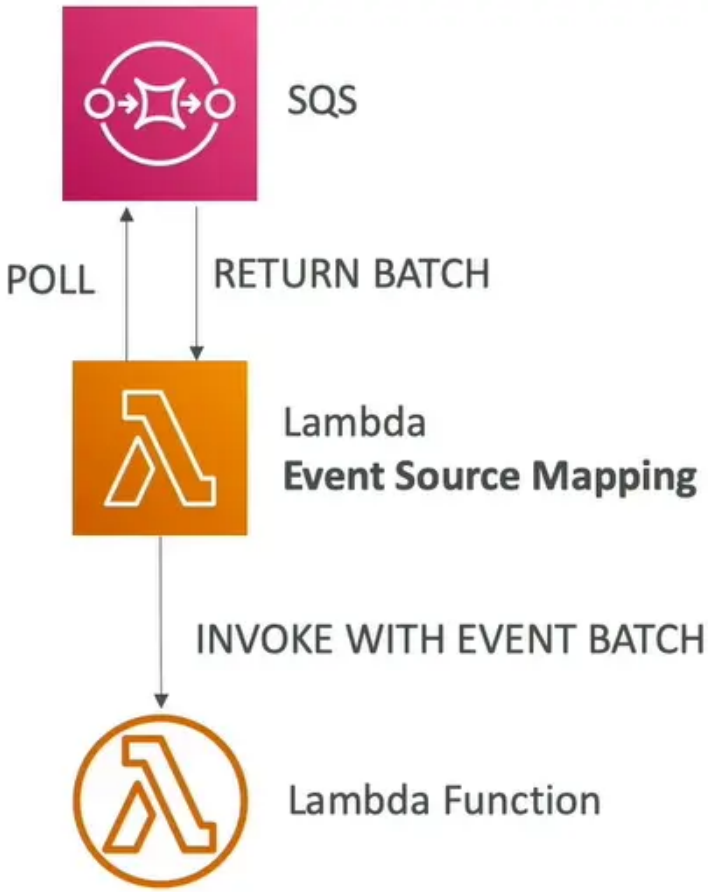
- an event source mapping creates an iterator for each shard, processes items in order
- start with new items ,from the beginning or from timestamp
- processed items aren't removed from the stream (other consumers can read them)
- low traffic: use batch window to accumulate records before processing
- you can process multiple batches in parallel
  - up to 10 batches per shard
  - in-order processing is still guaranteed for each partition key



Streams & Lambda – error handling

- by default, if your function returns an error, the entire batch is reprocessed until the function succeeds, or the items in the batch expire.
- to ensure in-order processing, processing for the affected shard is paused until the error is resolved
- you can configure the event source mapping to :
  - discard [丢弃] old events
  - restrict [限制] the number of retries
  - split the batch on error (to work around lambda timeout issues)
- discard events can go to a Destination

Event Source Mapping SQS & SQS FIFO



- event source mapping will poll [轮询] SQS (Long Polling)
- specify batch size (1–10 messages)
- recommanded : set the queue visibility timeout to 6x the timeout of your Lambda function
- to use a DLQ
  - set-up on the SQS queue,not Lambda (DLQ for Lambda is only for async invocations)
  - or use a Lambda destination for failures

Queues & Lambda

- lambda also supports in-order processing for FIFO (first-in, first-out)queues,scaling up to the number of active message groups
- for standard queues, items aren't necessarily processed in order.
- lambda scales up to process a standard queue as quickly as possible.

- when an error occurs, batches are returned to the queue as individual items and might be processed in a different grouping than the original batch
- occasionally, the event source mapping might receive the same item from the queue twice,even if no function error occurred.
- Lambda deletes items from the queue after they're processed successfully
- you can configure the source queue to send items to a dead-letter queue if they can't be processed.

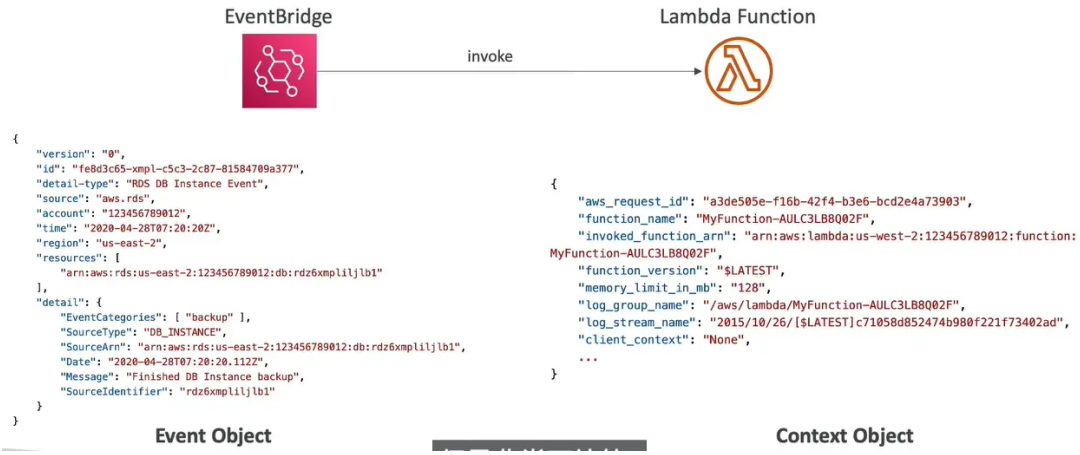
Lambda Event Mapper Scaling

- Kinesis Data Stream & DynamoDB Streams
  - one Lambda invocation per stream shard

- if you use parallelization, up to 10 batches processed per shard simultaneously [同时]

- SQS Standard
  - Lambda adds 60 more instances per minute to scale up
  - up to 1000 batches of messages processed simultaneously
- SQS FIFO
  - message with the same GroupID will be processed in order
  - the Lambda function sacles to the number of active message groups

Event and Context Objects

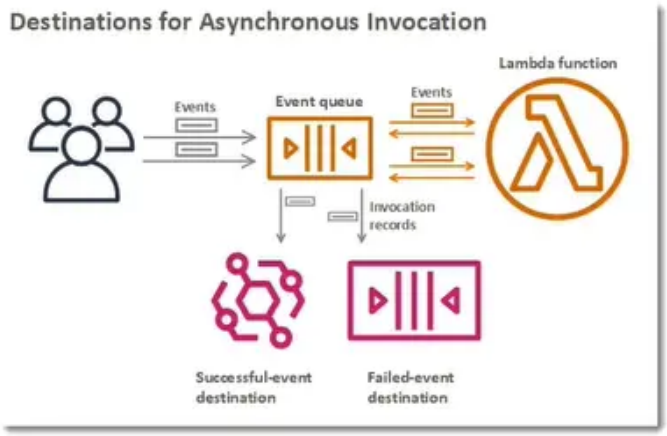


- Event Object
  - JSON-formatted document contains data for the function to process
  - contains information from the invoking service (eg,EventBridge, custom...)
  - lambda runtime converts [转换] the event to an object (eg,dict type in python)
  - example: input arguments, invoking service arguments..
- Context Object
  - provides methods and properties that provide information about the invocation,function, and runtime environment
  - passed to your function by Lambda at runtime
  - example:aws_request_id,function_name, memory_limit_in_mb,...

Access Event & Context Objects using Python

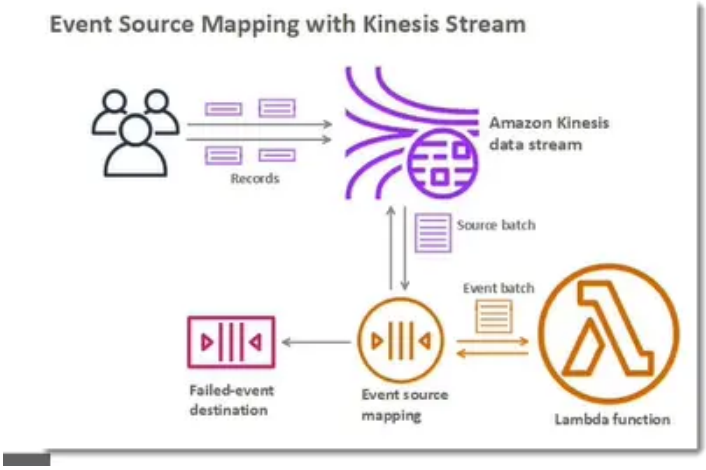
```python
def lambda_handler(event, context):
    print("Event Source:", event.source)
    print("Event Region:", event.region)


    print("Lambda Request ID:", context.aws_request_id)
    print("Lambda function ARN:", context.function_name)
    print("Lambda function ARN:", context.invoked_function_arn)
    print("Lambda function memory limits in MB:", context.memory_limit_in_mb)
    print("CloudWatch log stream name:", context.log_stream_name)
    print("CloudWatch log group name:",  context.log_group_name)
```

Destinations



https://docs.aws.amazon.com/lambda/latest/dg/invocation-async.html

- Nov 2019: can configure to send result to a destination
- Asynchronous invocations – can define destinations for successful and failed event
  - Amazon SQS
  - Amazon SNS
  - AWS Lambda
  - Amazon EventBridge bus
- Note:AWS recommends you use destinations instead of DLQ now (but both can be used at the same time)



- Event Source mapping : for discard event batches
  - Amazon SQS
  - Amazon SNS
- Note : you can send events to a DLQ directly from SQS

Lambda Execution Role (IAM Role)

- grants [授予] the lambda function permissions to AWS services /resources
- sample managed policies for Lambda
  - AWSLambdaBasicExecutionRole – upload logs to CloudWatch
  - AWSLambdaKinesieExecutionRole – Read from Kinesis
  - AWSLambdaDynamoDBExecutionRole – read from DynamoDB Streams
  - AWSLambdaSQSQueueExecutionRole – Read from SQS
  - AWSLambdaVPCAccessExecutionRole – Deploy Lambda function in VPC
  - AWSXRayDaemonWriteAccess – Upload trace data to X-Ray
- when you use an event source mapping to invoke your function, Lambda uses the execution role to read event data.

- best practice : create one Lambda Execution Role per function

Lambda Resource Based Policies

- use resouce-based policies to give other accounts and AWS services permission to use your Lambda resources
- similar to S3 bucket pilicies for S3 bucket
- an IAM principal can access Lambda
  - if the IAM policy attached to the principal authorizes it (eg,user access)
  - or if the resource-based policy authorizes (eg,service access)

- when an AWS service like Amazon S3 calls your Lambda function, the resource-based policy give it access.

Lambda Environment Variables

- environment variable = key/value pair in "string" form
- adjust [调整] the function behavior without updaing code
- the enviornment variables are available to your code
- lambda service ads its own system environment variables as well

- helpful to store secrets (encrypted by KMS)
- secrets can be encrypted by the lambda service key, or your own CMK

Lambda Logging & Monitoring

- CloudWatch Logs
  - AWS Lambda ececution logs are stored in AWS CloudWatch Logs
  - **make sure your AWS Lambda function has an execution role with an IAM policy that authorizes writes to CloudWatch Logs**
- CloudWatch Metrics
  - AWS Lambda metrics are displayed in AWS CloudWatch Metrics
  - Invocations, Durations [持续时间], Concurrent Executions [并发执行]
  - error count, success rates , Throttles [限制]
  - async delivery failures
  - iterator age (kinesis & dynamoDB streams)

Lambda Tracing with X-Ray

- enable in Lambda configuration (**active tracing**)
- runs the X-Ray daemon for you
- use AWS X-Ray SDK in code
- ensure Lambda function has a correct IAM Execution Role
  - the managed policy is called AWSXRayDaemonWriteAccess
- environment variables to communicate with X-Ray
  - **_X_AMZN_TRACE_ID**: contains the tracing header
  - **AWS_XRAY_CONTEXT_MISSING**:by default,LOG_ERROR
  - **AWS_XRAY_DAEMON_ADDRESS**:the X_Ray Daemon IP_ADDRESS:PORT

Customization At the Edge

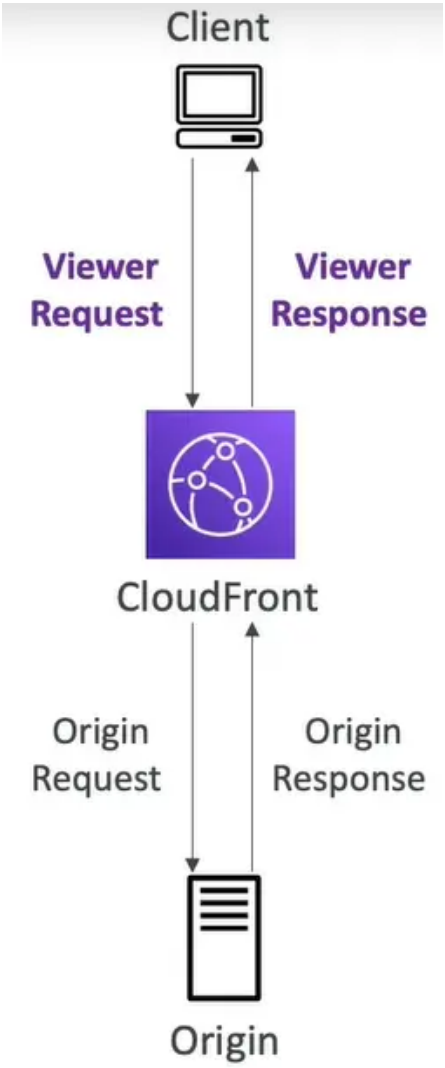
- many modern applications execute some form of the logic at the edge
- Edge Function
  - a code that you write and attach to CloudFront distributions
  - runs close to your users to minimize latency
- CloudFront provides two types:CloudFront Functions & Lambda@Edge
- you don’t have to manage any servers, deployed golbally

- use case: customize the CDN content
- pay only for what you use
- fully serverless

CloudFront Function & Lambda@Edge Use Cases

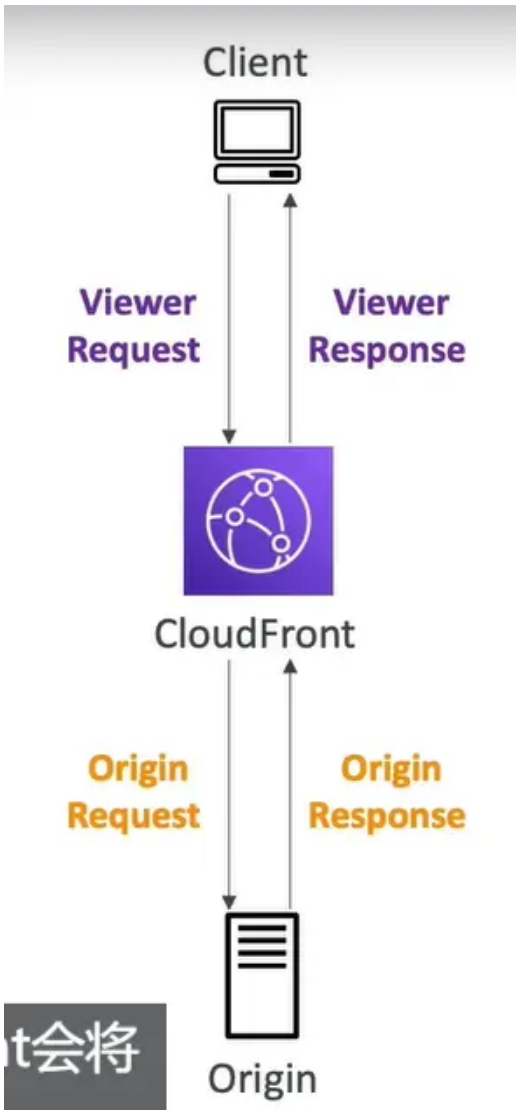
- website security and privacy
- dynamic web application at the edge
- search engine Optimization [优化] (SEO)
- intelligently route across origins and data centers
- bot mitigation at the edge
- real-time image transformation
- A/B testing
- user authentication and authorization
- user priortization
- user tracking and analytics

CloudFront Functions



- lightweight [轻量级] functions written in javascript
- for high-scale, latency-sensitive CDN customizations
- sub-ms startup times, millions of requests / second
- used to change viewer requests and responses
  - Viewer Request: after CloudFront receives a request from a viewer
  - Viewer Response: before CloudFront forwards the response to the viewer
- native feature of CloudFront (manage code entirely within CloudFront)

Lambda@Edge



- Lambda functions written in NodeJS or Python
- Scales to 1000s of requests / second
- used to change CloudFront requests and responses:
  - View Request – after CloudFront receives a request from a viewer
  - Origin Request – before CloudFront forwards the request to the origin
  - Origin Response – after CloudFront receives the response from the origin
  - Viewer Response – before CloudFront forwards the response to the viewer
- Author your function in one AWS Region (us-east-1),then CloudFront replicate [复制] to its locations

cloudFront Functions vs. Lambda@Edge

	CloudFront Functions	Lambda@Edge
Runtime Support	JavaScript	Node.js, Python
# of Requests	**Millions** of requests per second	**Thousands** of requests per second
CloudFront Triggers	- Viewer Request/Response	- Viewer Request/Response - Origin Request/Response
Max. Execution Time	< 1 ms	5 – 10 seconds
Max. Memory	2 MB	128 MB up to 10 GB
Total Package Size	10 KB	1 MB – 50 MB
Network Access, File System Access	No	Yes
Access to the Request Body	No	Yes
Pricing	Free tier available, 1/6th price of @Edge	No free tier, charged per request & duration

CloudFront Functions vs Lambda@Edge – Use Cases

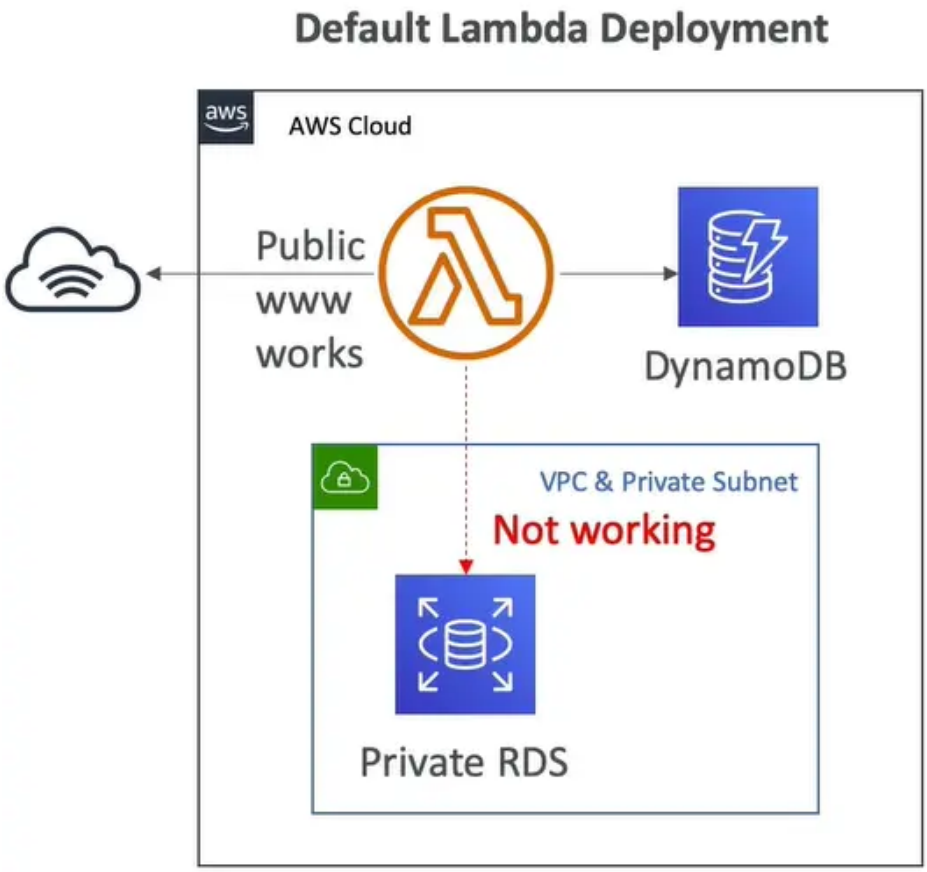
CloudFront Functions

- cache key normalization
  - transform request attributes (headers, cookies, query strings,URL) to create an optimal cache key
- header manipulation
  - insert/modify/delete HTTP headers in the request or response
- URL rewrites or redirects
- request authentication [验证] & authorization [授权]
  - create and validate user-generated token(eg,JWT) to allow/deny requests

Lambda@Edge

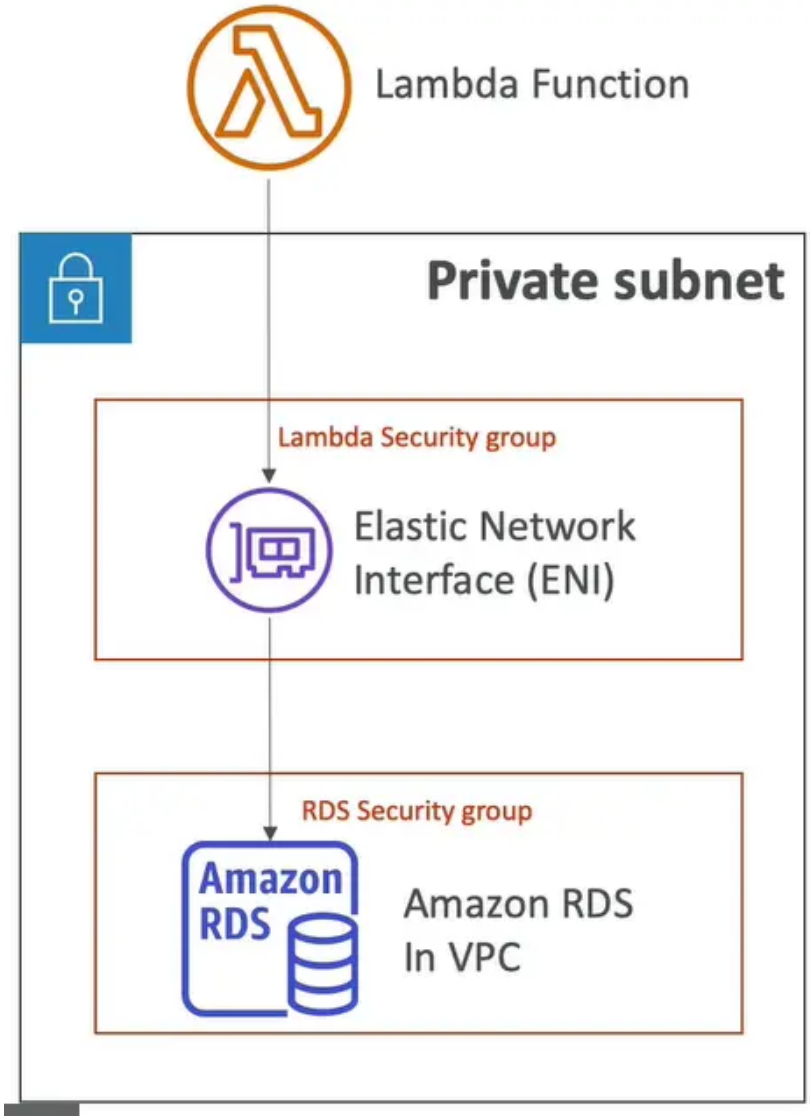
- longer execution time (several ms)
- adjustable [调节] CPU or memory
- your code depends on a 3rd libraries (eg,AWS SDK to access other AWS services)
- network access to use external services for processing
- file system access or access to the body of HTTP requests

Lambda by default



- by default, your Lambda function is launched outside your own VPC (in an AWS-owned VPC)
- therefore it cannot access resources in your VPC(RDS,ElastiCache,internal ELB..)

Lambda in VPC



- you must define the VPC ID, the subnet and the security groups
- lambda will create an ENI (Elastic Network Interface) in your subnets
- AWSLambdaVPCAccessExecutionRole

Lambda Function Configuration

- RAM
  - from 128MB to 10GB in 1MB increments [增量]
  - the more RAM you add, the more vCPU credit you get
  - at 1792 MB,a function has the equivalent of one full vCPU
  - after 1792 MB, you get more than one CPU, and need to use multi-threading [多线程] in your code to benefit from it
- if your application is CPU-bound (computation heavy), increase RAM
- timeout: default 3 seconds,maximum is 900 seconds (15minutes)

Lambda Execution Context

- the execution context is a temporary runtime environment that initializes any external dependencies of your lambda code
- great fo database connections,HTTP clients,SDK clients..
- the execution context is maintained for some time in anticipation of another Lambda function invocation

- the next function invocation can ”re-use” the context to execution time and save time in initializing connnections objects
- the execution context includes the */tmp* directory

initialize outside the handle

BAD!

```
import os

def get_user_handler(event, context):

    DB_URL = os.getenv("DB_URL")
    db_client = db.connect(DB_URL)
    user = db_client.get(user_id = event["user_id"])

    return user
```

The DB connection is established
At every function invocation

GOOD!

```
import os

DB_URL = os.getenv("DB_URL")
db_client = db.connect(DB_URL)

def get_user_handler(event, context):

    user = db_client.get(user_id = event["user_id"])

    return user
```

The DB connection is established once
And re-used across invocations

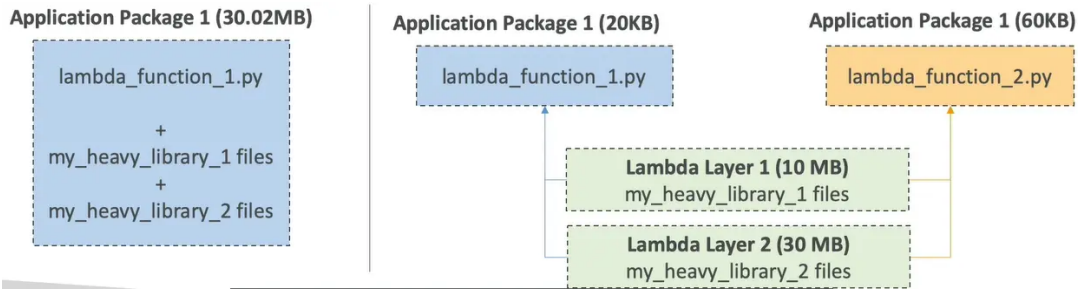
Lambda Functions /tmp space

- if your Lambda function needs to download a big file to work
- if your lambda function needs disk space to perform operation..
- you can use the /tmp directory
- max size is 10GB

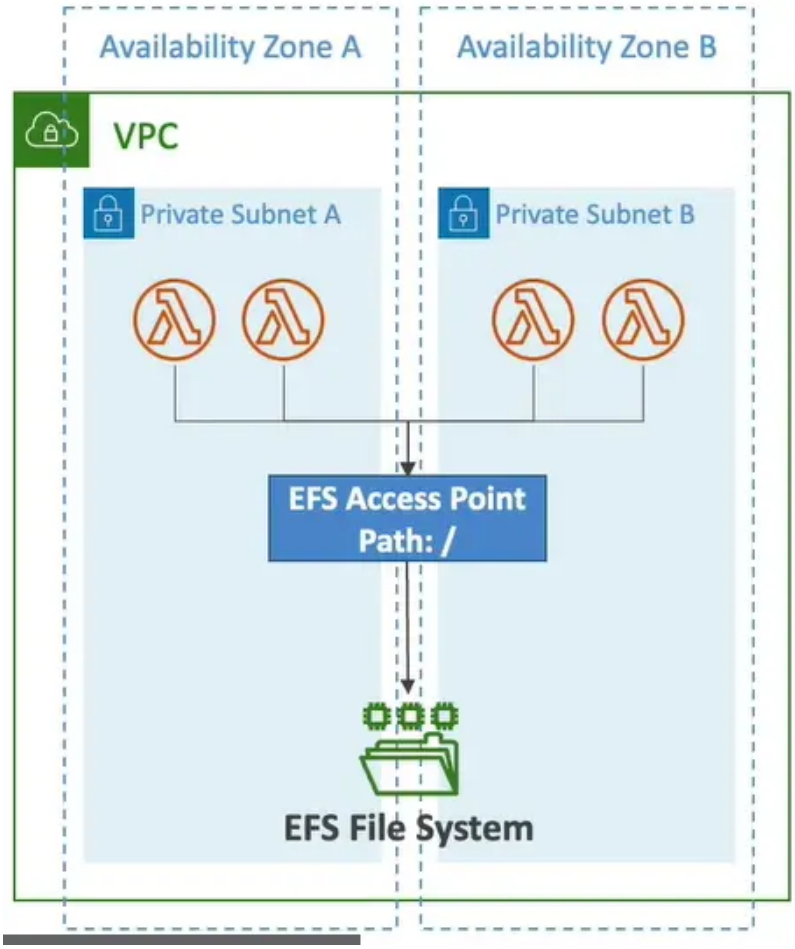
- the directory content remains when the execution context is frozen, providing transient [短暂的] cache that can be used for multiple invocations (helpful to checkpoint your work)
- for permanent persistence [永久持久化]of object (non temporary),use S3
- to encrypt content on /tmp, you must generate KMS Data Keys

Lambda Layers

- custom Runtimes
  - ex: C++
  - ex: Rust
- externalize dependencies to re-use them



Lambda – file systems mounting



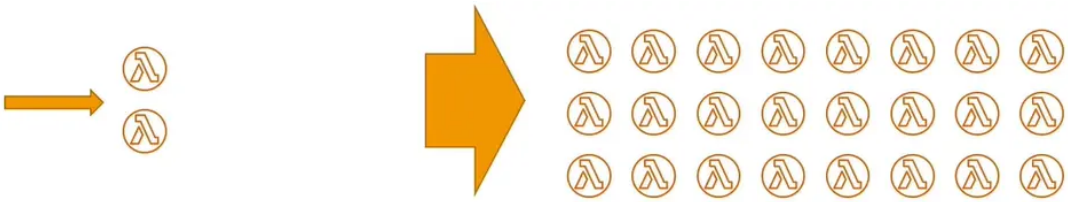
- lambda functions can access EFS file systems if they are running in a VPC
- configure Lambda to mount EFS file systems to local directory during initialization
- must leverage EFS Access points
- limitations: watch out for the EFS connection limits (one function instance = one connection) and connection burst limits

Storage Option

	Ephemeral Storage /tmp	Lambda Layers	Amazon S3	Amazon EFS
Max. Size	10,240 MB	5 layers per function up to 250MB total	Elastic	Elastic
Persistence	Ephemeral	Durable	Durable	Durable
Content	Dynamic	Static	Dynamic	Dynamic
Storage Type	File System	Archive	Object	File System
Operations supported	any File System operation	Immutable	Atomic with Versioning	any File System operation
Pricing	Included in Lambda	Included in Lambda	Storage + Requests + Data Transfer	Storage + Data Transfer + Throughput
Sharing/Permissions	Function Only	IAM	IAM	IAM + NFS
Relative Data Access Speed from Lambda	Fastest	Fastest	Fast	Very Fast
Shared Across All Invocations	No	Yes	Yes	Yes

Lambda Concurrency [并发] and Throttling[节流/限制]

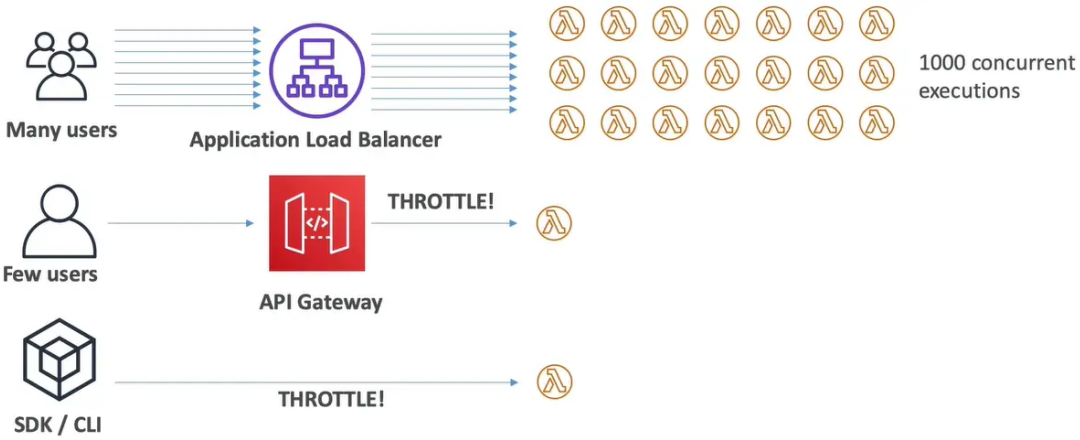
- concurrency limit: up to 1000 concurrent executions



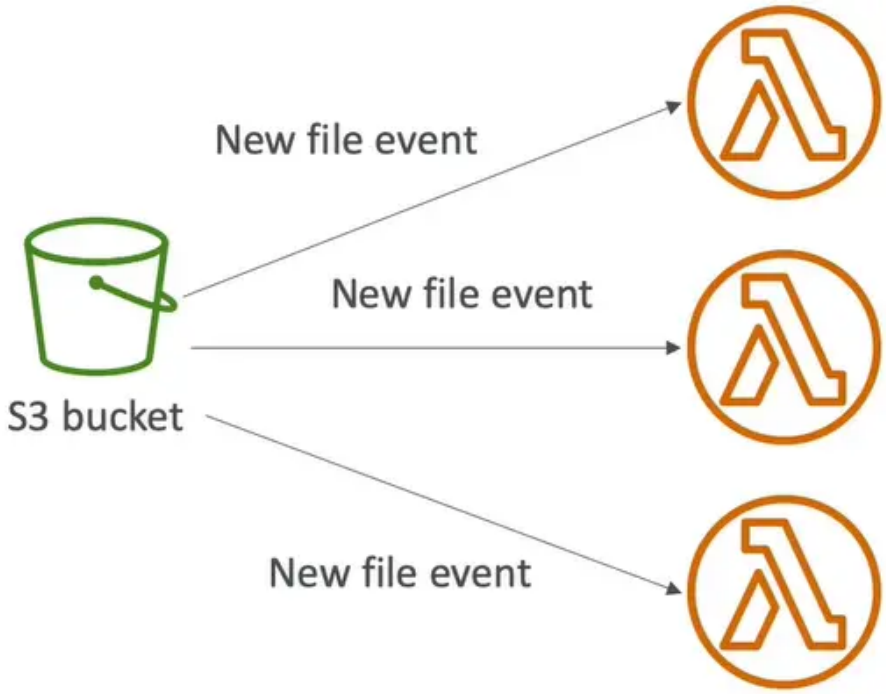
- can set a "reserved [预留] concurrency" at the function level (=limit)
- each invocation over the concurrency limit will trigger a "Throttle"
- Throttle behavior
  - if synchronous invocation => return ThrottleError – 429
  - if asynchronous invocation => retry automaticaly and then go to DLQ
- if you nedd a higher limit, open a support ticket

Lambda Concurrency Issue

- if you don't reserve(=limit) concurrency, the following can happen



Concurrency and Asynchronous Invocations

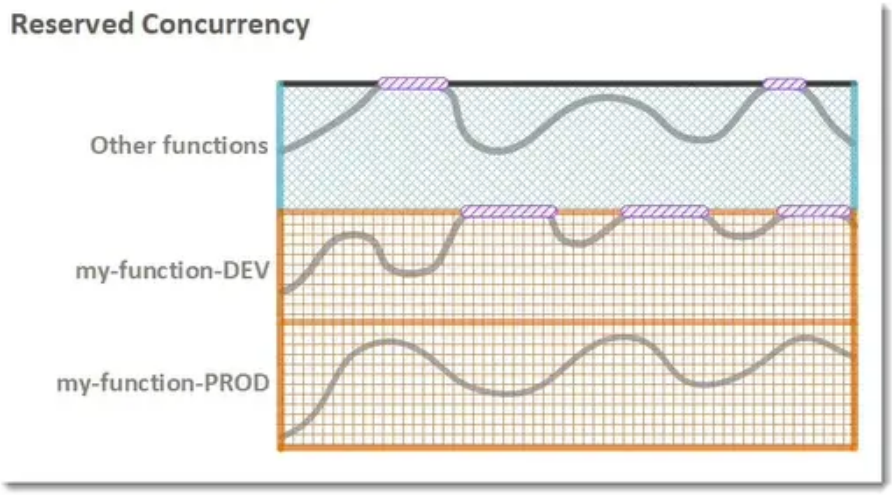


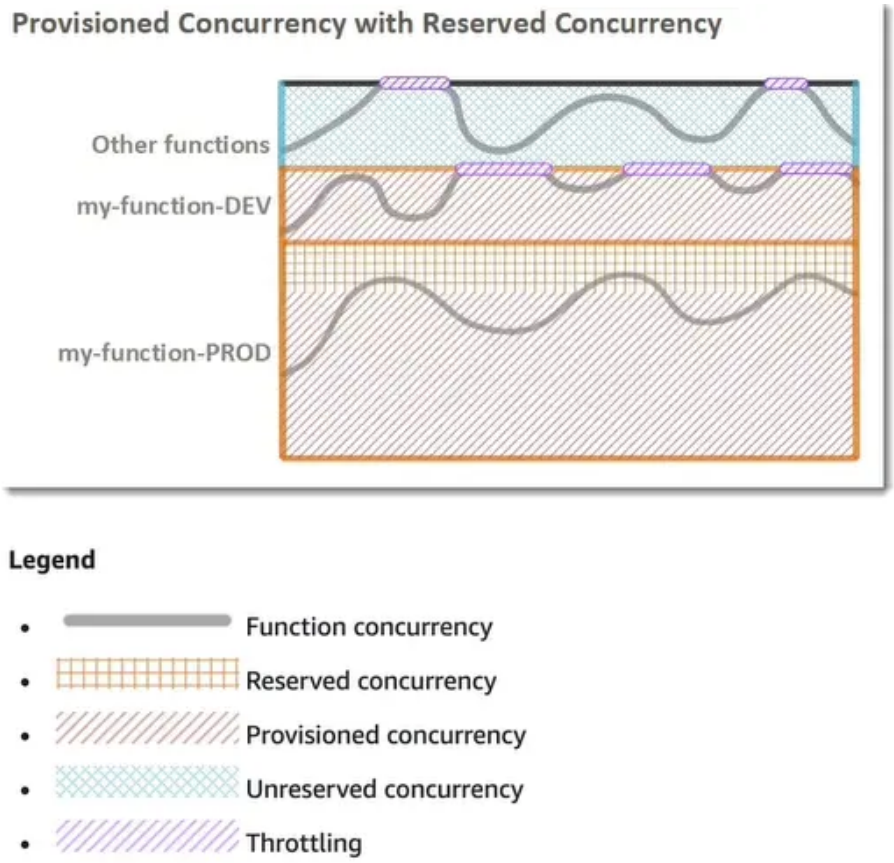
- if the function doesn't have enough concurrency available to process all events, additional requests are throttled.
- for throttling errors(429) and system errors(500–series),Lambda returns the event to the queue and attempts to run the function again for up to 6 hours
- the retry interval [间隔] increase exponentially [指数地] from 1 second after the first attempt to a maximum of 5 minutes

Cold Starts & Provisioned Concurrency

- cold start
  - new instance => code is loaded and code outside the handler run (init)
  - if the init is large (code, dependencies, SDK..) this process can take some time
  - first request served by new instances has higher latency than the rest
- Provisioned concurrency:
  - concurrency is allocated [分配] before the function is invoked (in advance [提前])
  - so the cold start never happens and all invocations have low latency
  - application auto scaling can manage concurrency (schedule or target utilization)
- Note:
  - note:cold start in VPC have been dramatically reduced in Oct & Nov 2019

reserved and provisioned concurrency





Lambda function dependencies

- if your lambda function depends on external libraries: for example AWS X-Ray SDK,Database Clients,etc..
- you need to install the packages alongside your code and zip it together
  - for node.js,use npm & "node_modules" directory
  - for python,use pip --target options

- for java, include the relevant .jar files
- upload the zip straight to Lambda if less than 50 MB, else to S3 first
- Native libraries work:they need to be complied on Amazon Linux
- AWS SDK comes by default with every Lambda function

Lambda and CloudFormation – inline

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Lambda function inline
Resources:
  primer:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: python3.x
      Role: arn:aws:iam::123456789012:role/lambda-role
      Handler: index.handler
      Code:
        ZipFile: |
          import os

          DB_URL = os.getenv("DB_URL")
          db_client = db.connect(DB_URL)
          def handler(event, context):
              user = db_client.get(user_id = event["user_id"])
              return user
```

- inline functions are very simple
- use the Code.ZipFile property
- you cannot include function dependencies with inline functions

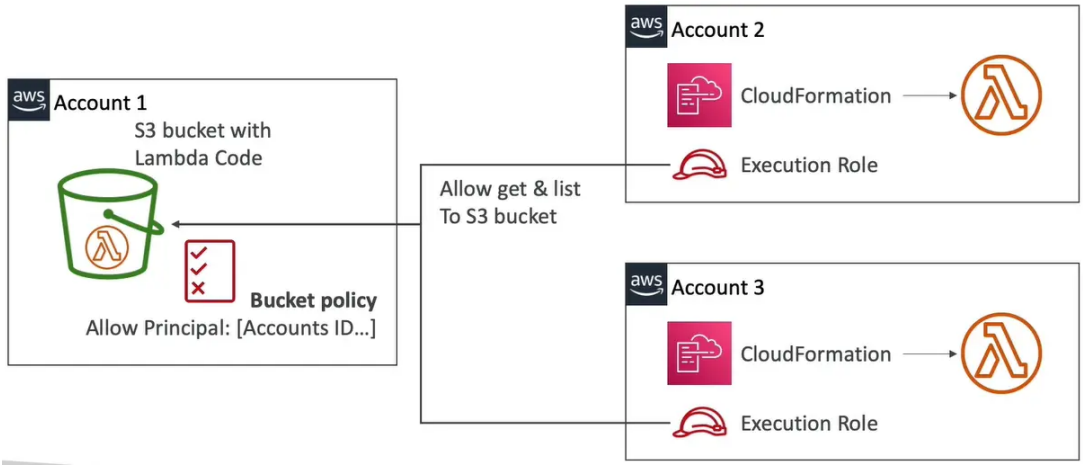
Lambda and CloudFormation – through S3

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Lambda from S3
Resources:
  Function:
    Type: AWS::Lambda::Function
    Properties:
      Handler: index.handler
      Role: arn:aws:iam::123456789012:role/lambda-role
      Code:
        S3Bucket: my-bucket
        S3Key: function.zip
        S3ObjectVersion: String
      Runtime: nodejs12.x
```

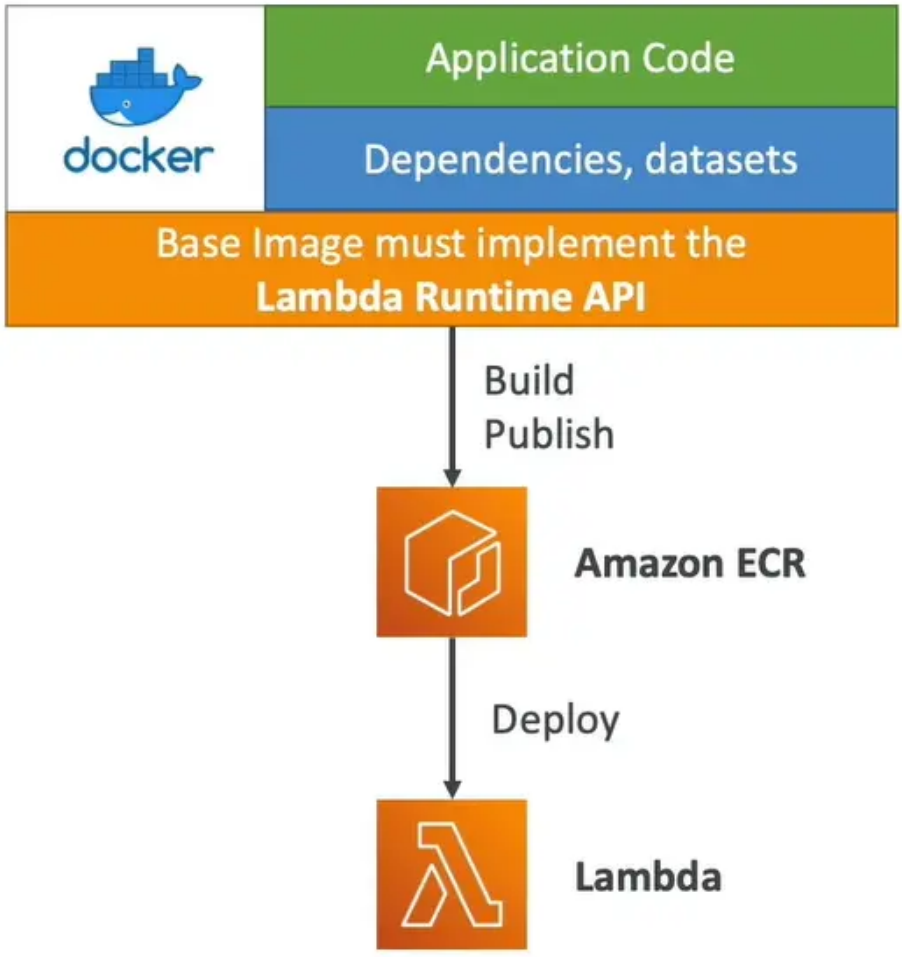
- you must store the Lambda zip in S3
- you must refer [引用] the S3 zip location in the CloudFormation code
  - S3bucket
  - S3key:full path to zip
  - S3ObjectVersion: if versioned bucket

- if you update the code in S3, but don't update S3Bucket,S3key or S3ObjectVersion,CloudFormation won't update your function

lambda and cloudformation – through S3 multiple accounts



Lambda Container Images



- deploy Lambda function as container images of up to 10GB from ECR
- pack complex dependencies,large dependencies in a container
- base images are available for python,node.js,java,.net,go, ruby
- can create your own image as long as it implements the Lambda Runtime API
- test the containers locally using the Lambda Runtime Interface Emulator
- unified [统一] workflow to build apps

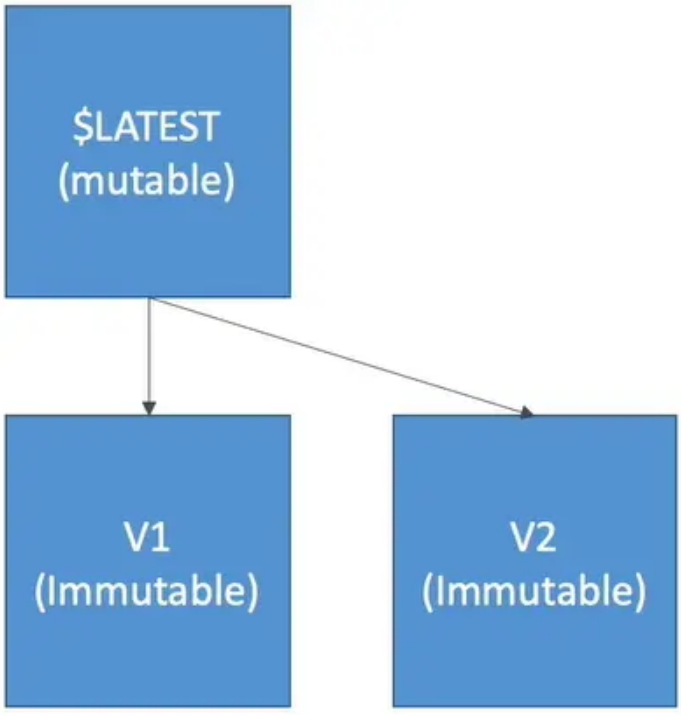
```
# Use an image that implements the Lambda Runtime API
FROM amazon/aws-lambda-nodejs:12

# Copy your applciation code and files
COPY app.js package*.json ./

# Install the dependencies in the container
RUN npm install

# Function to run when the Lambda function is invoked
CMD [ "app.lambdaHandler" ]
```

- example: build from the base images provided by AWS

Lambda container images – best practices

- strategies for optimizing container images:
  - use AWS-provided Base Images
    - stable,built on Amazon Linux 2x, cached by lambda service
  - use multi-stage builds
    - build your code in larger preliminary [初步] images， copy only the artifacts you need in your final container image,discard the preliminary steps
  - build from stable to requently changing
    - make your most frequently occurring changes as late in your Dockerfile as possible
  - use a single repository for functions with large layers
    - ECR compares each layer of a container image when it is pushed to avoid uploading and storing duplicates
- use them to upload large lambda functions (up to 10GB)

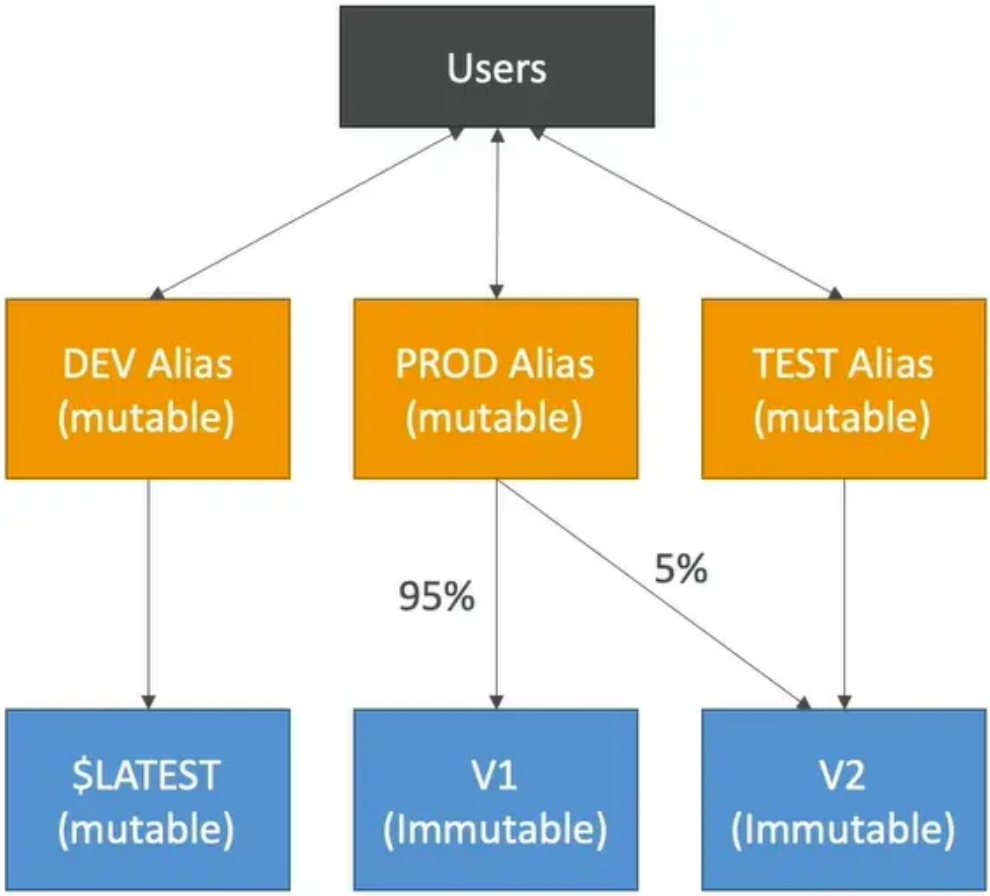
AWS Lambda Versions



- when you work on a Lambda function, we work on $LATEST
- when we're ready to publish a Lambda function,we create a version

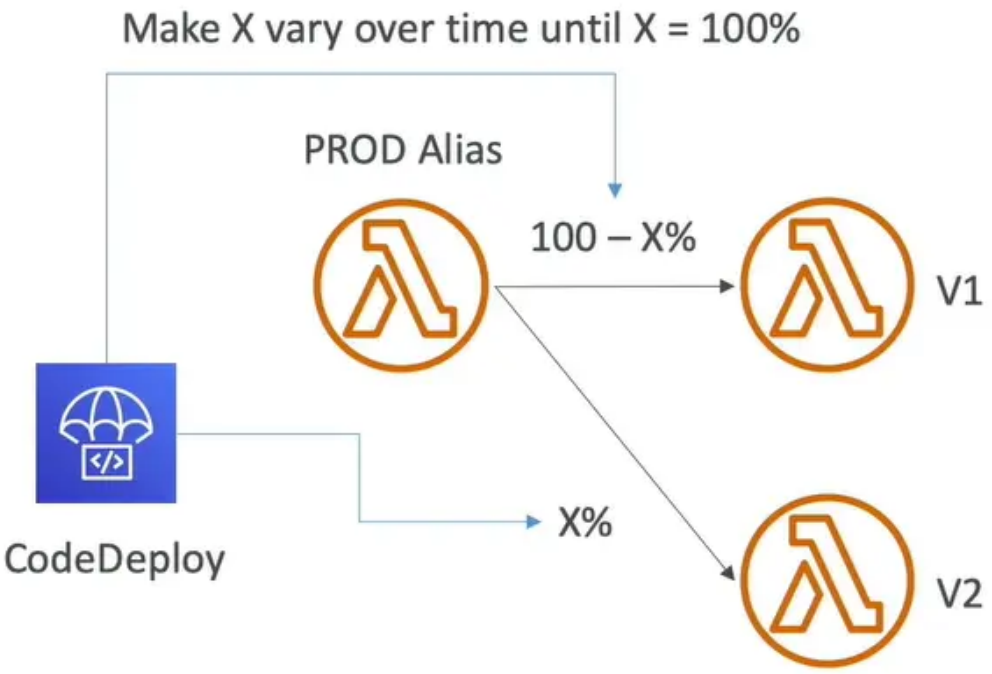
- versions are immutable [不可变的]
- versions have increasing version numbers
- versions get their own ARN (Amazon Resource Name)
- version = code + configuration(nothing can be changed –immutable)
- each version of the lambda function can be accessed

AWS Lambda Aliases [别名]



- Aliases are ”pointers” to Lambda function versions
- we can define a ”dev”,”test”,”prod” aliases and have them point at different lambda versions
- aliases are mutable [可变的]
- Aliases enable Canary deployment by assigning [分配] weights to lambda functions
- aliases enable stable configuration of our event triggers /destinations
- aliases have their own ARNs
- aliases cannot reference aliases

Lambda & CodeDeploy



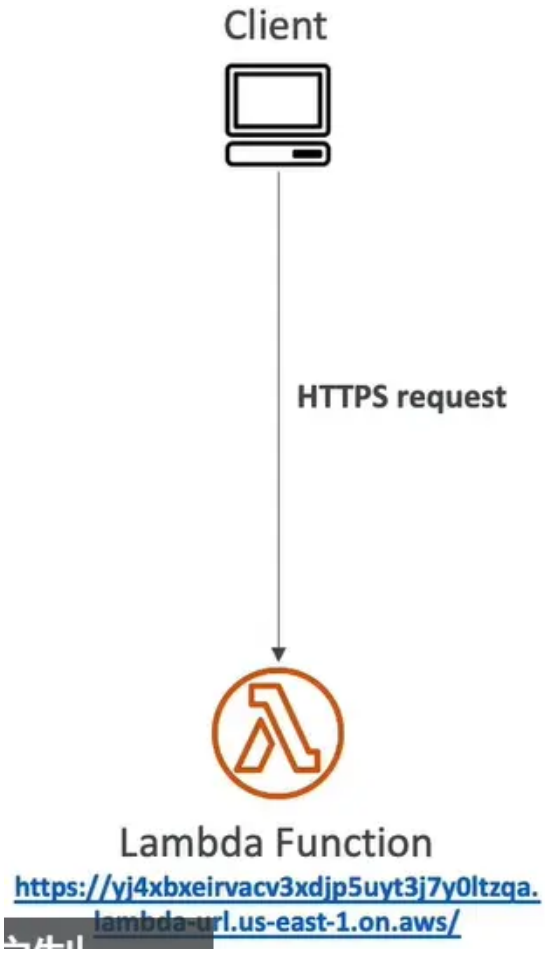
- codeDeploy can helo you automate trffic shift for Lambda aliases
- feature is integrated within the SAM framework
- Linear : grow traffic every N minutes until 100%
  - Linear10percentEvery3Minutes
  - Linear10PercentEvery10Minutes
- Canary: try Xpercent then 100%
  - canary10Percent5Minutes
  - canary10percent30Minutes
- AllAtOnce: immediate
- can create pre & post traffic hooks to check the health of the Lambda function

Lambda & CodeDeploy – AppSpec.yml

```yaml
version: 0.0

Resources:
  - myLambdaFunction:
      Type: AWS::Lambda::Function
      Properties:
        Name: myLambdaFunction
        Alias: myLambdaFunctionAlias
        CurrentVersion: 1
        TargetVersion: 2
```

- Name(required) – the name of the Lambda function to deploy
- Alias(required) – the name of the alias to the lambda function
- CurrentVersion (required) – the version of the Lambda function traffic currently points to
- TargetVersion(required) – the verison of Lambda function traffic is shifted to

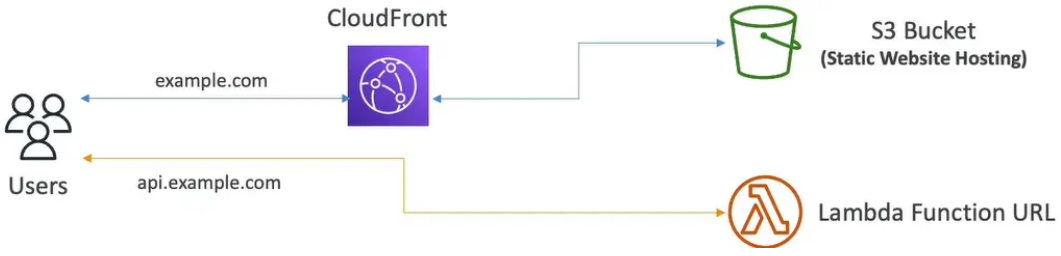
Lambda – Function URL



- dedicated [专用] HTTP(s) endpoint for your Lambda function
- a unique URL endpoint is generated for you (never changes)
  - https://<url-id>.lambda-url.<region>.on.aws(dual-stack IPv4 & IPv6)
- invoke via [通过] a web browser, curl, postman, or any HTTP client
- access your function URL through the public Internet only
  - doesn't support privatelink (lambda functions do support)
- supports resource-based policies & CORS configurations
- can be applied to any function alias or to $LATEST (can't be applied to other function versions)
- create and configure using AWS Console or AWS API
- Throttle your function by using Reserved Concurrency

Lambda –function URL Security

- Resource-based Policy
  - Authorize other accounts / specific CIDR /IAM principals
- Corss-Origin Resource Sharing (CORS)
  - if you call your Lambda function URL form a different domain

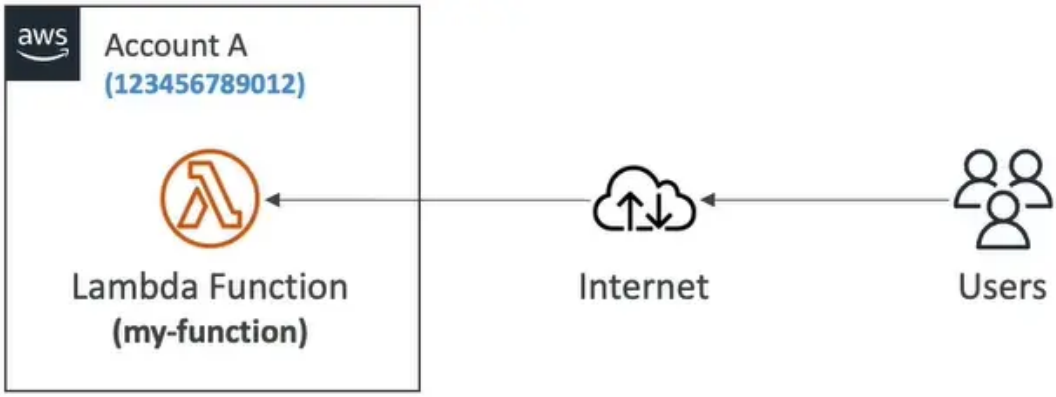


- AuthType NONE – allow public and unauthenticated access

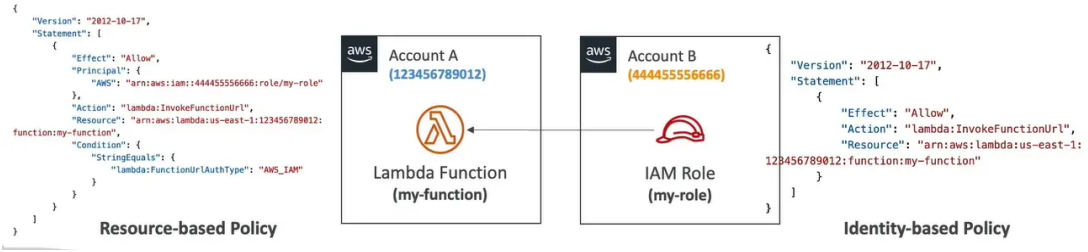
- Resource-based Policy is always in effect (must grant [授予] public access)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "lambda:InvokeFunctionUrl",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:
function:my-function",
            "Condition": {
                "StringEquals": {
                    "lambda:FunctionUrlAuthType": "NONE"
                }
            }
        }
    ]
}
```

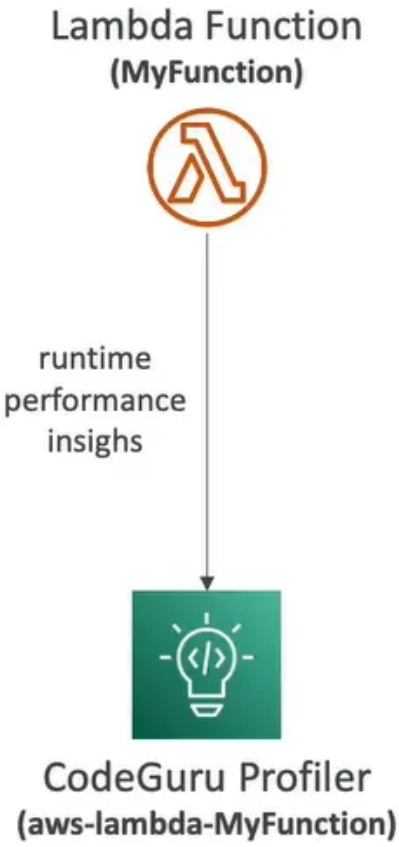
Resource-based Policy



- AuthType AWS_IAM – IAM is used to authenticate and authorize requests
  - both principal's identity-based policy & resource-based policy are evaluated [评估]
  - principal must have lambda:InvokeFunctionUrl permissions
  - Same account – Identity-based Policy OR Resource-based Policy as ALLOW
  - Cross account – Identity-based Policy AND Reource Based Policy as ALLOW



Lambda and CodeGuru Profiling



- gain insights into [深入了解] runtime performance [运行时性能] of your lambda funtions using CodeGuru Profiler
- CodeGuru creates a Profiler Group for your lambda function
- supported for java and python runtimes
- activate from AWS lambda console
- when actived, lambda adds:
  - CodeGuru Profiler layer to your function
  - environment variables to your function
  - AmazonCodeGuruProfilerAgentAccess policy to your function

AWS Lambda Limits to Know – per region

- Execution:
  - memory allocation :128MB –10GB (1MB increments)
  - maximum execution time : 900 seconds (15minutes)
  - environment variables (4KB)
  - disk capacity in the ”function container”(in /tmp):512MB to 10 GB
  - concurrency executions 1000 (can be increased)
- Deployment:
  - lambda function deployment size (compressed [压缩] .zip): 50MB
  - size of uncompressed deployment (code +dependencies) : 250MB
  - can use the /tmp directory to load other files at startup
  - size of environment variable:4KB

AWS Lambda Best Practices

- perform heavy–duty work outside of your function handler
  - connect to database outside of your function handler

    - initialize the AWS SDK outside of your function handler
    - pull in dependencies or datasets outside of you function handler
- use environment variables for:
    - database connection strings ,S3 bucket,etc,..don’t put these values in your code
    - passwords, sensitive values ..they can be encrypted using KMS
- minimize your deployment package size to its runtime necessities
    - break down the function if need be
    - remember the AWS Lambda limits
    - use layers where necessary
- Avoid using recursive [递归] code, never have a lambda function call itself