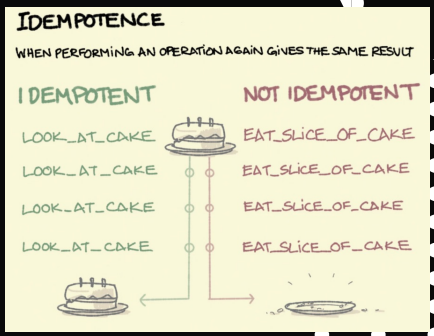


EC2 → Amazon Elastic Computer Cloud

ECS → Amazon Elastic container Service

upstream - does not depend on other service

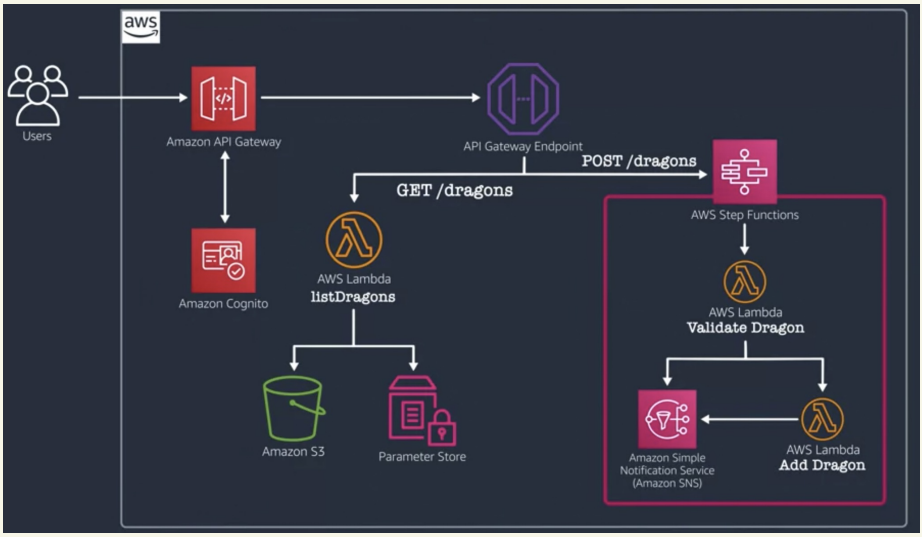
downstream - depends on upstream services

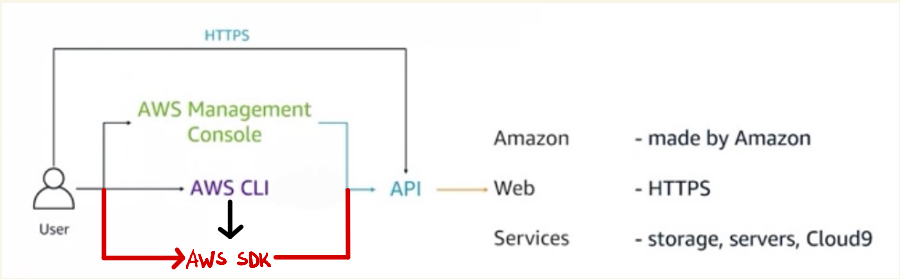
C9

Amazon API gateway HTTP APIs

→ Cloud 9
→ S3 bucket
→ Boto 3
→ Cognito
→ Lambda

→ Step function
→ State Language
→ X-Ray
→ API gateway response caching
→ Edge





aws --region ca-central-1 s3 mb s3://modern-webapp

option — command — sub command — parameter

→ Ways to interact with BOTO3

Client Interface
- → Low level
- → One-to-one map to AWS API
- → Responses come as python dictionaries

Resource Interface
- → High Level
- → Object-oriented
- → Wrapper for client
- → Exposes a subset of AWS API's

Client →

Returns data in the form of dictionaries

```
import boto3
client = boto3.client('s3')
response = client.list_objects(Bucket='aws')
for content in response['Contents']:
    obj_dict = client.get_object(Bucket='aws', Key=content['Key'])
    print(content['Key'], obj_dict['LastModified'])
```

Resource →

Returns data in the form of object

```
import boto3
s3 = boto3.resource('s3')
bucket = s3.Bucket('aws')
for obj in bucket.objects.all():
    print(obj.key, obj.last_modified)
```

access client from resource ←

```
s3_client = boto3.resource('s3').meta.client
```

Session - aws credentials
aws region we are submitting our API call

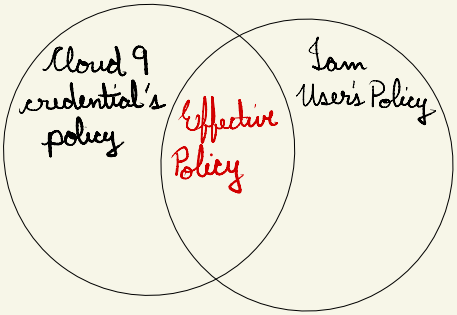
S3 select - SQL query read / filter a subset of data in an object

Data filtering done by S3

aws system parameter manager store: store / retrieve key-value pair

→ Credentials passed into the client
→ Credentials passed into the session
→ Environmental Variable
→ Credentials file
→ IAM role

→ cat ~/.aws/credentials } reset every 5 min



API Gateway :-

→ Amazon API enables you to create, publish, maintain, monitor, & secure your REST, HTTP & websocket API

→ REST API ⇒ fast, stateless, standard, horizontal scalable, & dependable APIs

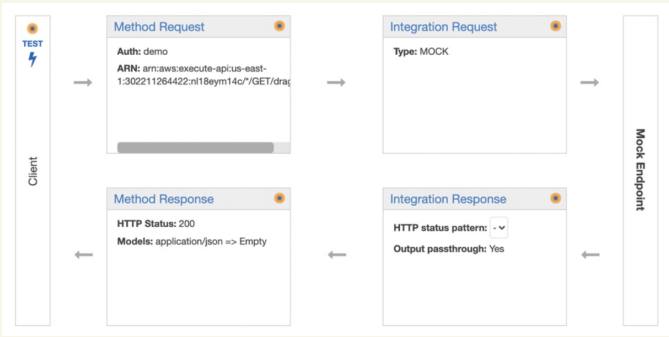
→ features ⇒ Canary deployment, CloudTail, CloudWatch, custom domain support, AWS service

Methods :- client-facing interface by which calls can be made

→ Edge-optimised API endpoints are best for geographical client.

→ Regional API endpoints are intended for clients in the same region.

→ Private API endpoints allows a client to securely access private API resources inside a VPC



API Gateway REST API → Validates requests/responses with model

→ Transforms requests/responses with mapping

Models :— Creating using JSON schema

→ Can be used to do request validation

Mapping :— → Backend expect a diff format than the client is sending. API Gateway handels this

→ exceptional proxy integration

→ Written in VTL → supports XML

→ inject new parameters → conditional statement

→ hard code data → refrence data in run time

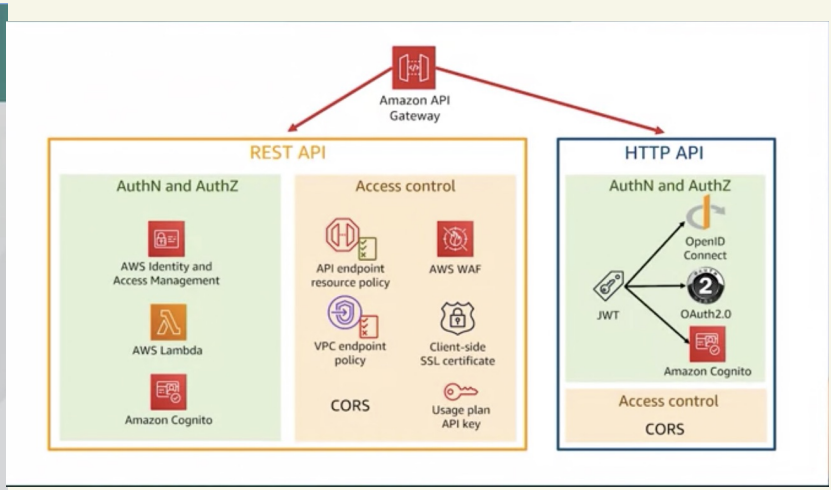
→ map data in complex structures

Mock integration :- Enables your API to return a response for a request directly.

https://{restapi-id}.execute-api.{region}.amazonaws.com/{stageName}

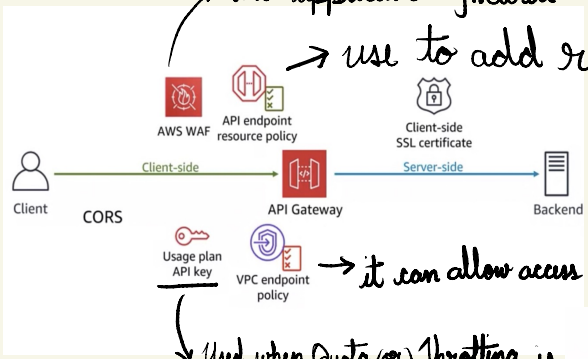
Stages Provide

- Method for deployment
- API versioning
- Performance management
- SDK generation



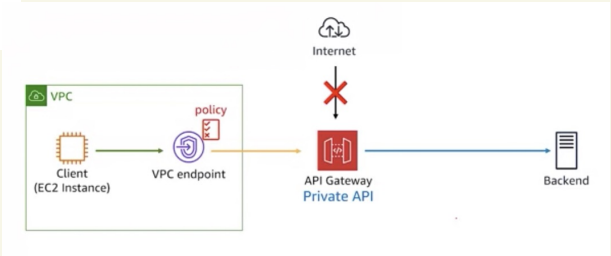
web application firewall

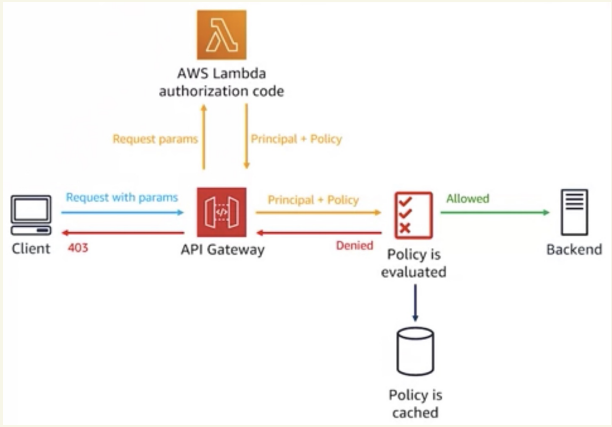
→ use to add restrictions to the resources



→ it can allow access to private API

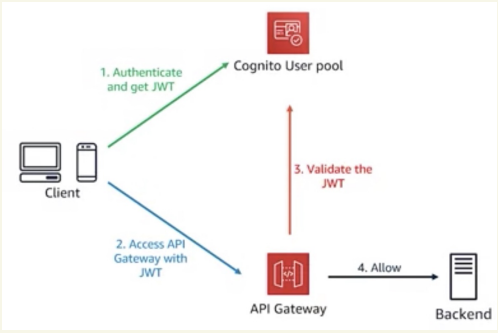
↳ Used when Quota (or) Throttling is to be added





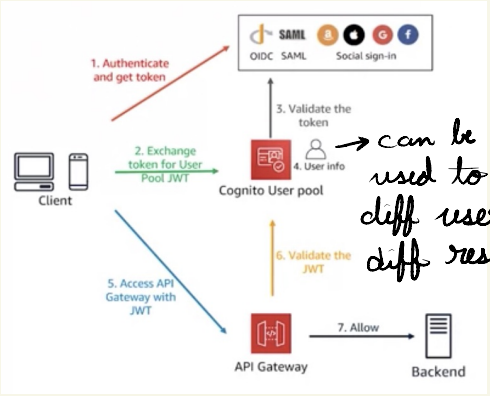
← Lambda
→ can be customised

Cognito

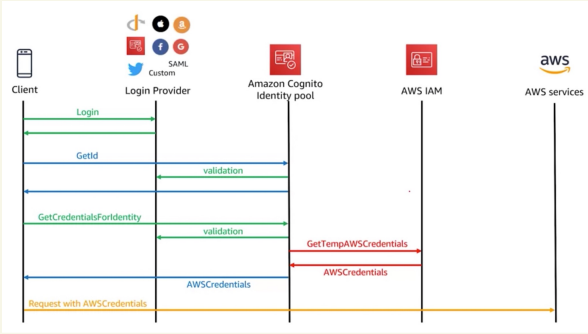


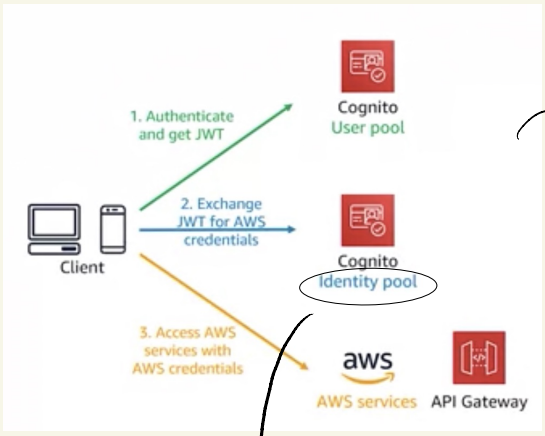
→ custom code can be run once someone signs up.
→ allow backend to use data once sign up

→ When using a 3rd party for authentication



→ can be used to give diff user diff resource





→ Use when you directly want to use other AWS services then use Identity pool

→ Federated identity
Giving basics IAM role to the not authorized user.

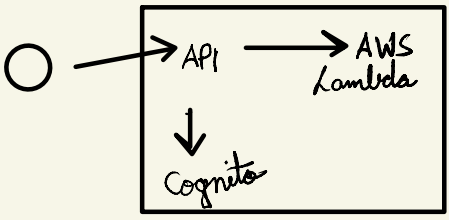
Aws Lambda

Event → JSON-formatted document containing data for a function to process.

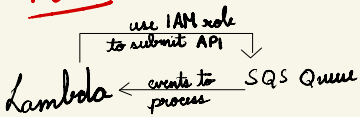
Concurrency → Number of requests a function is serving at any given time

Trigger → resource or configuration that invokes function

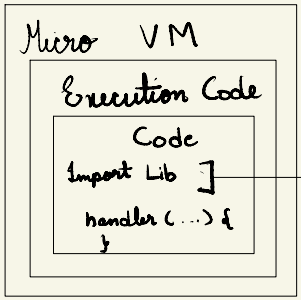
Push



Pull



→ Provisioned Concurrency

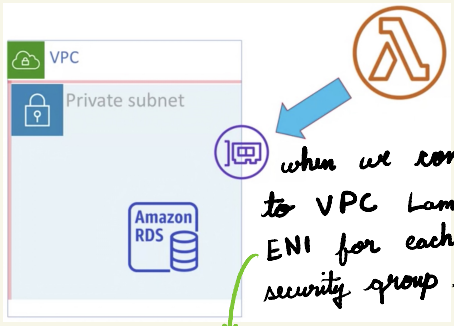


→ Lambda endpoints only support secure connection over HTTPS

→ AWS Lambda encrypts environment variables at rest

→ Key config. allow to use an encryption key managed by AWS key management service.

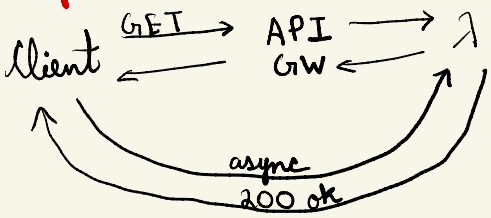
→ Encryption helper enable encryption of environment var. on the client side.



when we connect a function to VPC Lambda create a ENI for each combination of security group & subnet.

Electricity Network Interface

Synchronous & Asynchronous



Asynchronous Config

→ Max. age of event
→ Retry attempts → 1 or 2 only if its idempotent
→ Dead letter queue

(or)

Lambda Destination
- → asynchronous
- → stream → pull model type of invocation with the trigger

can be applied multiple times without changing

Lambda function version

The system creats a new version each time a function is published.

→ code, 1 runtime, enviromental variables, ARN (amazon resource name)

→ we can use AWS management console (or) published version API

→ every lambda version has a unique ARN
→ ARN for both function & version can't be changed once created
→ one for the function & one that points to specific version of the function.

Alias

→ Pointer to a specific AWS Lambda function version accessed by it's ARN.

→ can be used when we have sources such as amazon S3 event, which are used to invoke your lambda function.

→ to create alias we use create alias API.

Boto3 + Lambda

```
import boto3

s3 = boto3.resource('s3','us-east-1').meta.client
ssm = boto3.client('ssm','us-east-1')
bucket_name = ssm.get_parameters('  ')['Parameter']['Value']
file_name =   "          ('  ')   "    "

def listDragons(event, content):
    expression = "select * from s3object s"
    if 'queryStringParameter' in event 4['quer...'] not None:
        { change expression
    result = s3.select_object_content(
        Bucket = bucket_name
        Key = file_name
        ExpressionType='SQL',
        Expression = expression,
        InputSerialization = {'JSON': {'Type': 'Document'}},
        Output "   = {'JSON': {}})
    records = '
    for event in result['Payload']:
        if 'Records' in event:
            records = event['Records']['Payload'].decode('utf-8')
    return {"statusCode": 200,
        "body": json.dumps(records)}
```

calling select object API

→ sudo pip install --target ./t1 boto3

→ cd t1

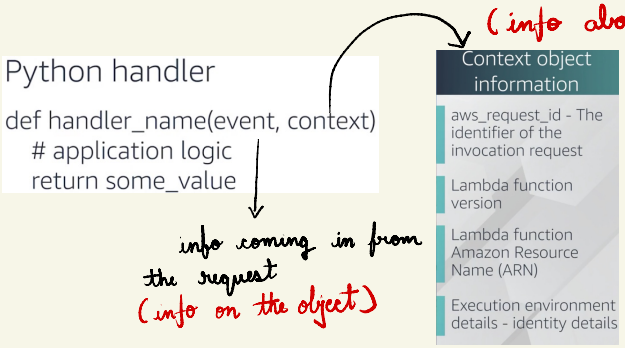
→ zip -r9 ${OLDPWD}/ fil.zip

→ zip -g file.zip 1.py

→ aws lambda create-function --function name <>
--runtime <> --role <> -- handler <> -- publish
-- zip file <>

→ aws lambda invoke -- function -- function name <>
output.txt

→ aws lambda update-function-code --function name <>
-- zip file <>



synchronous → JSON
asynchronous → the returned value will be discarded

→ Boto 3 package is present in lambda but it must be present in zip form as it will update in future and can affect the perf.

Aws Step Function

→ Helps to create server workflow
→ Workflow in step function is called state machine
→ State machine are composed of state
→ Keep tracks of state machine & its execution.

Advantages

→ No server → Auto scaling → Parallel execution

It is built using amazon states language.
JSON-based, structured language used to define state machines & their workflow

State are individual ⇒ input → actions → output

Tasks – one unit of work done

Blue Point of Step function

```
{
  "Comment": "",
  "StartAt": "S1",
  "States": {
    "S1": {
      "Type": "Pass",
      "Result": "S1",
      "Next": "S2"
    },
    "S2": {
      "Type": "Pass",
      "Result": "S2",
      "End": true
    }
  }
}
```

```
"Retry": [
  {
    "ErrorEquals": [""],
    "IntervalSeconds":
    "MaxAttempts":
    "BackoffRate":
  }
]
```

2nd retry = IS + BOR

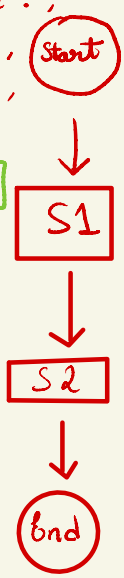
[Task, Choice]

```
{
  "Variable":
  "BooleanEquals":
  "Next":
}
```

Task

```
"Resource"
"Catch": [
  {
    "ErrorEquals"
    "Next"
    "ResultPath"
  },
]
"Parameters": {
  "Message"
"PhoneNumber.$"
},
```

→ Path = string starting with $



Types of state

→ Task – do some real work [call other AWS service (integration)]
→ Choice – Takes decision based on inputs
→ Parallel – when one state is not dependent on another for inputs ["Branches"]
Loop – more than one set of data [Array] → combination of choices

→ Map = we want to pass set of input concurrently ["Max Concurrency":]

→ Pass = takes input & pass it as output [testing]

→ if payload is not working at starting set it in middle to recieve an output

→ we can use it while building state machine & the we can replace with appropriate type

→ Wait → "Seconds"

→ Succed → ends the execution with a success

→ Fail → " " " " failuer [can specify the cause & error]

can be used in catch statement afterwards

AWS Step Function Service Integrations

use to identify the invoke lambda

API action for

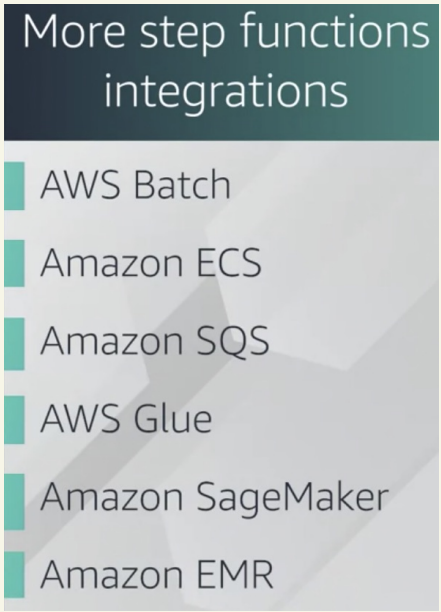
```
"Invoke Lambda function": {
        "Type": "Task",
        "Resource": "arn:aws:states:::lambda:invoke",
        "Parameters": {
            "FunctionName":
            "arn:aws:lambda:<REGION>:<ACCTNUMBER>:function:<FUNCTIONNAME>",
            "Payload": {
                "Input.$": "$"
            }
        },
        "Next": "NEXT_STATE"
    }
```

provided to the state

```
"Invoke Lambda State": {
      "Type": "Task",
      "Resource":"arn:aws:lambda:<REGION>:<ACCTNUMBER>:function:<FUNCTIONNAME>",
      "Next": "NEXT_STATE"
}
```

→ Amazon Dynamo DB [NoSQL]

```
"Put item into DynamoDB": {
          "Type": "Task",
          "Resource": "arn:aws:states:::dynamodb:putItem",
          "Parameters": {
                "TableName": "MyDynamoDBTable",
                "Item": {
                      "DragonName": {
                            "S": "Atlas"
                      },
                      "Family" : {
                            "S": "Red"
                      }
                }
          },
          "Next": "NEXT_STATE"
    }
```



→ Amazon SNS

```
"Amazon SNS: Publish a message": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
            "Message": "Hello from Step Functions!",
            "PhoneNumber": "+15555555555"
      },
      "Next": "NEXT_STATE"
}
```

Step function
AWS Lambda
External resources
] Write Code

Integration Pattern

You can call a service, and let step functions progress to the next state immediately after it gets an HTTP response. You can call a service and have step functions wait for job to complete or you can call a service with a task token and have step functions wait until that token is returned with a payload

⇒ API Gateway → Lambda function → Step function → step machine

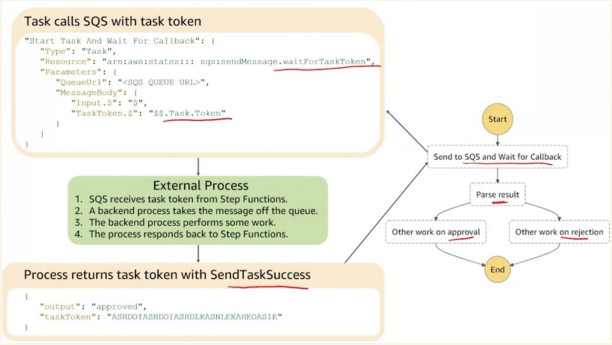
Mapping uses VTL template.

Supported Service Integrations

Service	Request Response	Run a Job (.sync)	Wait for Callback (.waitForTaskToken)
Lambda	✓		✓
AWS Batch	✓	✓	
DynamoDB	✓		
Amazon ECS/AWS Fargate	✓	✓	✓
Amazon SNS	✓		✓
Amazon SQS	✓		✓
AWS Glue	✓	✓	
Amazon SageMaker	✓	✓	
Amazon EMR	✓	✓	
AWS CodeBuild	✓	✓	
AWS Step Functions	✓	✓	✓

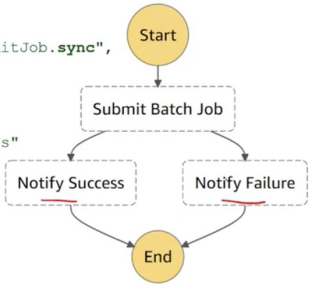
→ Step function
task token
SQS
Backend (process returns task token with response SendTaskSuccess)
Step function

Step function
SQS (Step function wether the inputs are accepted by SQS)
Step function proceeds to other task



Run a job

```
"Submit Batch Job": {
   "Type": "Task",
   "Resource": "arn:aws:states:::batch:submitJob.sync",
   "Parameters": {
      "JobDefinition": "preprocessing",
      "JobName": "preprocessingBatchJob",
      "JobQueue": "primaryQueue",
      "Parameters.$": "$.batchjob.parameters"
   }
}
```



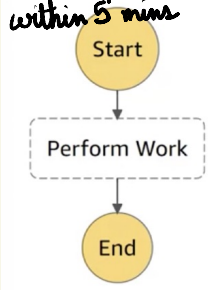
Step Function Activities

→ Any HTTPS protocol can be back end.

```
"Perform Work":{
   "Type": "Task",
   "Resource": "arn:aws:states:us-east-1:123456789012:activity:get-work",
   "HeartbeatSeconds": 60,
   "TimeoutSeconds": 300,
   "End": true
}
```

completed within 5 mins

worker needs to send response to step function every 60 sec that it's still performing task



get Activity Task (will run ∞)

{"ActivityArn": }

{"input:", "taskToken:"}

Send Task Heartbeat

{"taskToken"}

Sen Task Success (or) Failure

{"error:"(or) "output:"

taskToken:"

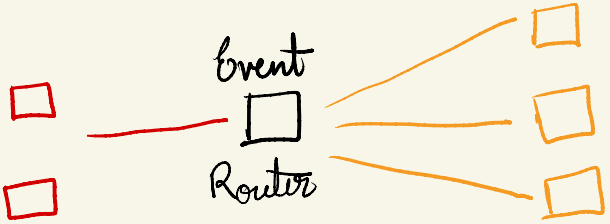
	Standard workflow	Express workflow
Execution start rate	Over 2,000 per second	Over 100,000 per second
State transition rate	Over 4,000 per second per account	Nearly unlimited
State transition persistence	Persisted to disk	In memory only
Execution semantics	Exactly-once workflow execution	At-least-once workflow execution
Pricing	Priced per state transition	Priced by the number of executions you run, their duration, and memory consumption
Maximum duration	365 days	5 minutes
Execution history	Stored in Step Functions and can be sent to Amazon CloudWatch Logs	Sent to Amazon CloudWatch Logs
Service integrations	Supports all service integrations and patterns	Supports all service integrations. Does not support Job-run (.sync) or Callback (.waitForTaskToken) patterns.

Event Driven Architecture

↳ uses event to trigger & communicate between decoupled services.

event → change in a state

Producer —HTTPs Protocol→ Consumer



→ Producer & consumer services are decoupled

⇒ Less **orchestration** than step function
↳ the planning of the elements of a situation to produce a desired effect

→ SQS → Simple Queue Services
↳ we can send, store & receive messages
14 days
↳ Standard SQS [Best effort ordering]
→ idempotent
↳ SQS FIFO

→ SNS → Producer can send message via the HTTPs protocol, fan out message to large number of consumer end point for 11 prom.

→ high throughput
→ lower latency event

Event Bridge

↳ A service bus that makes it easy to connect producers & consumers together using data from the application
→ Setup routing rules
→ Schema Registry → It stores, event structure or schema in a shared central location that consumer can query.

AWS X-RAY

Segment → Compute resources running the application logic
→ resource's name, details about the request, work done
→ can breakdown the data about the work done into subsegments
↳ timing information & downstream calls
→ In Amazon DB, X-Ray uses subsegments to generate inferred segments & downstream nodes on service map.

Service Graph → data sent from application (JSON)

Trace – tracking of the path of a request through the system collecting the segments generated by the request.

→ X-Ray provides a user-centric model.

- → API Gateway – X-Ray
- → AWS Lambda – X-Ray Daemon
- → AWS CloudTrail – X-Ray
- → AWS CloudWatch
  - → EC2
  - → Elastic Beanstalk
  - → SNS → SQS

Tracing → passive [has been enabled on upstream]
→ active [auto based on sampling algo]

X-Ray : Python

X-Ray SDK for python → creating & sending trace data to X-Ray daemon.
(run on UDP port 2000)

Trade data

- Incoming HTTP requests
- Downstream calls made using the AWS SDK
- HTTP clients
- SQL database connectors

Annotations allow you to associate key-value pairs with a trace that can then be filtered in the X-ray console.

Metadata information can be used to store data that can provide further information

Subsegments

Allows you to provide more granular detail about a segment, including custom information

Boto3 + Lambda [X-Ray]

allow us to record all the downstream calls to other AWS services

```
import boto3
from aws_xray_sdk.core import patch_all
patch_all()
s3 = boto3.resource('s3', 'us-east-1').meta.client
ssm = boto3.client('ssm', 'us-east-1')
bucket_name = ssm.get_parameters('    ')['Parameter']['Value']
file_name =  "                ('  ')   "    "

def listDragons(event, content):
    expression = "select * from s3object s"
    if 'queryStringParameter' in event & ['quer...'] not None:
        { change expression
    result = s3.select_object_content(
        Bucket = bucket_name
        Key = file_name
        ExpressionType='SQL',
        Expression = expression,
        InputSerialization = {'JSON': {'Type': 'Document'}},
        Output "  = {'JSON': {}})
    records = ''
    for event in result['Payload']:
        if 'Records' in event:
            records = event['Records']['Payload'].decode('utf-8')
    return {"statusCode": 200,
        "body": json.dumps(records)}
```

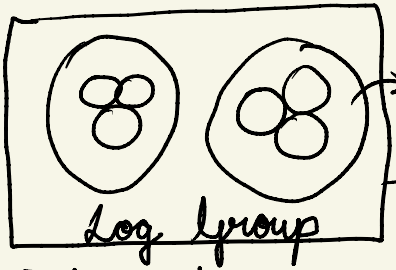
calling select object API

→ sudo pip install aws-xray-sdk

→ aws lambda update-function-configuration
--function-name list-dragons --tracing-config
Mode=Active

Log event - This is a record of some activity recorded by the application.

→ Timestamp of the event ↑ in the raw event message



→ Log stream [log events that share the same sources]

→ Retention

Log Group

[share the same retention monitoring an access control settings]

Logs are sent through HTTPs request

API Gateway
Step function
Lambda

} → Logs → Cloudwatch Logs

API Gateway → API Login → execution logging → log data includes errors, [request & resp. payloads, data used by lambda authorizers, wether API keys were required, " usage plans are enabled]

→ access logging → REST API → who has accessed & how he has accessed

give permissions → AWS identity & Access Management Services [IM role with IM policy]

Express workflow – default

standard workflow – need to enable manually

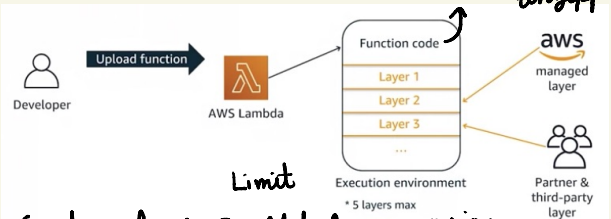
Edge → [less physical space]

↳ CloudFront [caching service]

Invocation types → Lambda [128 - 3008 MB]

- Viewer request to Amazon CloudFront
- Viewer request to origin
- Origin response to Amazon CloudFront
- Origin response to viewer

can't to exceed 250MB unzipped



Create a layer [publish layer version command]

→ name

→ description

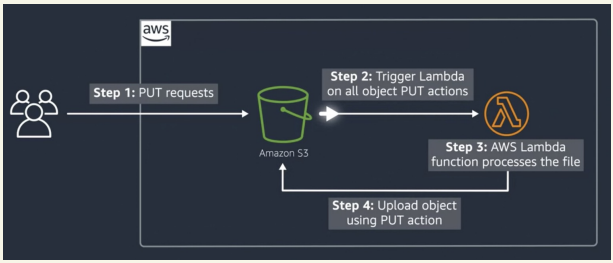
→ zip archive

→ Lambda Best Practices

→ Min. package size [75GB]

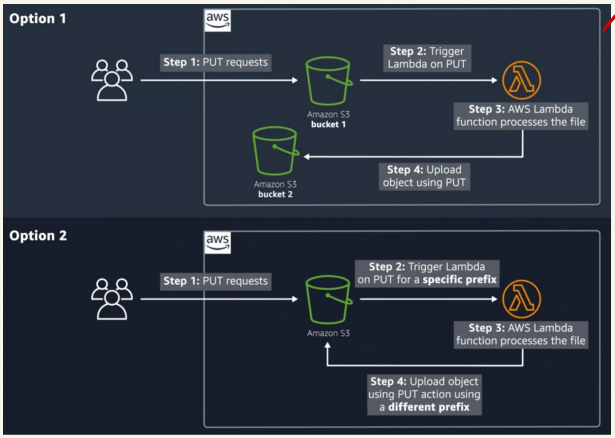
→ Min. the complexity of your dependencies

→ Avoid using recursive call



∞ loop

→ Sol.



Amazon API Gateway HTTP APIs

- Lambda proxy
- HTTP endpoint proxy
- OpenID Connect
- OAuth 2.0 for authorization
- CORS support
- Private integration support

→ set concurrent execution limit to 0 & update the code

→ Take adv. of execution content reuse

don't store user data (or) sensitive data in execution content

→ Load test your lambda to determine optimal timeout value.