API Gateway

If you are studying for AWS Developer Associate Exam, this guide will help you with quick revision before the exam. it can use as study notes for your preparation.

Dashboard Other Certification Notes

API Gateway

Allows us to create REST APIs which can be public and accessible to the clients. It can proxy requests to Lambda functions and other services which expose HTTP end-points.

Overview

- Integration with AWS Lambda: no infrastructure to manage
- Support for WebSockets protocol
- Handles API versioning
- Handles multiple environments (dev, test, prod)
- Handles security (authentication and authorization)
- Ability to create API keys => request throttling
- Supports Swagger/OpenAPI import to quickly define APIs
- Transform and validate requests
- Ability to cache API responses

Integrates with

- Lambda Functions
- HTTP: expose HTTP end-points from the back-end
- Other AWS services, examples:
  - AWS Step Function workflow

API Gateway Endpoint Types

- Edge-Optimized (defaut): for global clients
  - Requests will be routed through the CloudFront Edge locations
  - API Gateway will still live in one region where it was created
- Regional: for clients within the same region. Can be combined with CloudFront for control over caching strategies and distribution
- Private: can only be access within the VPC using VPC endpoint (ENI)

Deployment Stages

- Changes made in API Gateway are not in effect as long as they are not part of a deployment
- Changes are deployed to stages
- Recommended naming for stages: dev, test, prod, v1, v2, etc..
- Each stage can get its own configuration parameters, they can be rolled back
- Common use case for stages: migrate from previous version to the next version (example from v1 to v2)
- Stage variables:
  - Env. variables for API Gateway stages
  - Use case: configure HTTP end-point to which the stage talks to; pass configuration parameters to Lambda functions
  - Stage variables are passed to the “context” object in AWS Lambda
- Common pattern for the API Gateway and stage variables:
  - We create stage variables to indicate the corresponding Lambda alias
  - API Gateway will still live in one region where it was created
- Regional: for clients within the same region. Can be combined with CloudFront for control over caching strategies and distribution
- Private: can only be access within the VPC using VPC endpoint (ENI)

Deployment Stages

- Changes made in API Gateway are not in effect as long as they are not part of a deployment

- Changes made in API Gateway are not effected as long as they are not part of a deployment
- Changes are deployed to stages
- Recommended naming for stages: dev, test, prod, v1, v2, etc..
- Each stage can get its own configuration parameters, they can be rolled back
- Common use case for stages: migrate from previous version to the next version (example from v1 to v2)
- Stage variables:
  - Env. variables for API Gateway stages
  - Use case: configure HTTP end-point to which the stage talks to; pass configuration parameters to Lambda functions
  - Stage variables are passed to the "context" object in AWS Lambda
- Common pattern for the API Gateway and stage variables:
  - We create stage variables to indicate the corresponding Lambda alias
  - AWS_PROXY: Lambda Proxy. The request is passed to the Lambda without being modified
    - No mapping templates, headers and query strings can be modified. They are passed as arguments to the function
  - HTTP_PROXY: same as AWS_PROXY, used for other back-end services

Mapping Templates

- They are only usable for integration with AWS services
- They can be used for modify requests and responses
- Uses Velocity Template Language (VTL)
- We can filter output results
- Concrete example: integrate JSON to XML with SOAP
  - SOAP is XML based, REST is JSON based
  - Client interacts with API Gateway using JSON, API Gateway interacts with the SOAP endpoint using XML
  - API Gateway extracts the data from JSON, builds the SOAP request, calls the SOAP API, transforms the SOAP response to JSON

AWS API Gateway Swagger/OpenAPI Spec

- Common way to define REST APIs using API definition as code
- Existing Swagger/OpenAPI 3.0 specs can be imported into API Gateway
  - Method
  - Method Request
  - Integration Request
  - Method Response
  - AWS extensions for API Gateway every single setup
- APIs can be exported as Swagger/OpenAPI spec
- Swagger can be written in YAML/JSON
- Other application can generate code from the exported APIs

Caching in API Gateway

- Caching can reduce the number of calls made to the back-end
- Default TTL is 3000 seconds (min: 0, max: 3600s)
- Caching is defined per stage (one cache per stage)
- Possible to override cache setting per method
- Cache can be encrypted
- Cache capacity can be between 0.5GB up to 237GB
- Cache is expensive, use it only in production
- Cache can be invalidated by:
  - Flush the entire cache immediately
  - Clients can invalidate cache with the header `Cache-Control: max-age=0` (needs IAM auth)
  - In case of lack of *InvalidateCache* policy, any client can invalidate the cache

API Gateway Usage Plans & API Keys

- If we want to make an API available as an offering ($) to customers
- Usage plans:
  - Who can access one or more API stages
  - How many request can they perform in a given period of time
  - API key can be used to identify clients and meter their access
- API keys:
  - Alphanumeric strings
  - Clients can securely use the API and we can authenticate requests
  - We can control access
  - Quota limits is the overall number of requests
- Order:
  1. Create one or more APIs and deploy them into stages
  2. Generate/import API keys which can be distributed to application developers/customers
  3. Create usage plan with desired throttle and quota limits
  4. Associate API stages and API keys with the usage plan
- Callers must supply an API key using the **x-api-key** header

Monitoring, Logging/Tracing

- CloudWatch Logs:
  - Can be enabled at stage level
  - Can override settings on a per API basis
  - Log contains information about the request/response body
- X-Ray:
  - Enable tracing to get extra information
  - X-Ray + API Gateway + AWS Lambda gives full picture about the workings of an serverless application
- CloudWatch Metrics:
  - Metrics are per stage with the possibility to enable detailed metrics
  - Useful metrics:
    - **CacheHitCount** and **CacheMissCount** give information about the efficiency of the cache
    - **Count** - the total number of API requests
    - **IntegrationLatency** - the time between the request are relayed the the back-end and response is received by the API Gateway
    - **Latency** - total time between request and response. Max amount of time a request can perform is **29 seconds**
    - **4XXError** - client side errors
    - **5XXError** - server side errors
- Throttling
  - Global account limit by default is 10K request per second across all APIs
  - Soft limit, can be increased upon request
  - In case of throttling => 429 Too Many Requests
  - We can set stage limit & method limit to not use every request in case of a single API
  - We can create usage plans to throttle customer
- Errors:
  - 4xx client errors
    - 400: Bad Request
    - 403: Access Denied, WAF filtered
    - 428: Quota exceeded, Too Many Requests -5xx server errors:
    - 502: Bad Gateway - incompatible output from the back-end or out-of-order invocation due to heavy loads
    - 503: Service Unavailable
    - 504: Integration Failure - Back-end request timeout exceeded

CORS

- Must be enabled to support requests from other domains
- The OPTIONS preflight request must contain the following headers:
  - Access-Control-Allow-Methods
  - Access-Control-Allow-Headers
  - Access-Control-Allow-Origin

Security and IAM Permissions

-IAM Permissions: - We should create an IAM policy if other AWS services access the API Gateway from the same account - We can leverage “Sig v4” capability where IAM credentials are in headers - Authentication = IAM | Authorization = IAM Policy - Resource policies: - Set policy on the API Gateway and set who can access it - Allows for cross account access (combined with IAM Security) - Allow for specific IP source

- Cognito User Pools:
  - Cognito fully manages user lifecycle, token will expire automatically
  - API Gateway verifies the identity automatically from Cognito
  - No custom implementation required
  - | Authentication = Cognito User Pools | Authorization = API Gateway Methods |
- Lambda Authorizer (Custom Authorizer):
  - Token-based authorizer (bearer token)
  - A request parameter-based Lambda authorizer
  - Lambda must return an IAM policy for the user, result policy is cached
  - | Authentication = External | Authorization = Lambda function |

HTTP API vs REST API

- HTTP APIs:
  - Low latency, cost effective AWS lambda proxy, HTTP proxy and private integration (no data mapping)
  - Supported authorization: OIDC, OAuth2 CORS
  - No usage plans and API keys
- REST APIs:
  - No native OpenID Connect / OAuth 2.0 support

- WebSocket APIs:
  - Two-way interactive communication between the user and the server
  - Server can push information to the client (Stateful application)
  - Used for real-time applications such as chat, collaboration platforms, multiplayer games, trading platforms