**General**

API Gateway

* HTTP API and REST API - Cost is calculated by no of requests and amount of data that is transferred
* WebSocket API – Cost is calculated by messages sent and received and for the time user device is connected to the WebSocket API
* API version management, traffic management, authorization and access control, monitoring
* Acts as a front door to your applns from back-end services/applns on Amazon EC2, Amazon ECS, AWS Elastic Beanstalk, code running on AWS Lambda, or any web applns

1. Metering – set api plans, configure throttling and set quota limits per API key
2. Security – authorize access to your apis
3. Resiliency – help manage traffic with throttling so backend operations can withstand traffic spikes, caching the outoput of API calls to avoid calling your backend every time
4. Operations Monitoring – Metrics dashboard to monitor calls , API calls, latency data and error rates
5. Lifecycle management – lets you operate multiple api version and multiple stages for each version simultaneously so existing appln can continue to call pervious versions after a new api version is published
6. Designed for developers – quickly create apis and assign static content for their responses to reduce corss-team development effort and time to market for your appln
7. Real time two way communication – build two way chat apps streaming dashboards and notifications without running or managing servers

Types of Api Gateway

1. HTTP API – APIs that proxy to AWS lambda functions or http backends – ideal for serverless workloads – no api management functionality
2. Rest Api – API proxy and API management features in a single solution, usage plans, api keys publishing and monetizing apis
3. WebSocket API – maintains a persistent connection between connected clients to enable real-time message communication. Can define backend integrations with AWS lambda, Amazon Kinesis or HTTP endpoints when messages are received from connected clients

HTTP APIs/Rest APIs – Amazon API gateway console, AWS CLI, AWS SDK AWS CloudFormation

HTTP APIs are ideal for:

1. Building proxy APIs for AWS Lambda or any HTTP endpoint
2. Building modern APIs that are equipped with OIDC and OAuth 2 authorization
3. Workloads that are likely to grow very large
4. APIs for latency sensitive workloads

REST APIs are ideal for:

1. Customers looking to pay a single price point for an all-inclusive set of features needed to build, manage, and publish their APIs.

* HTTP APIs come standard with CORS support, OIDC and OAuth2 support for authentication and authorization and automatic deployments on stages
* Yes, you can import an API definition using OpenAPI 3. It will result in the creation of routes, integrations, and API models.
* Export OpenAPI from REST API and import Open API to HTTP API -> import operation – info and warning error fields
* All of the APIs created with Amazon API Gateway expose HTTPS endpoints only.
* JSON, XML, query string parameters, and request headers with transform templates
* Amazon API Gateway
  + AWS lambda functions in your account,
  + AWS step function state machine
  + Call HTTP endpoints hosted on AWS Elastic Beanstalk, Amazon EC2, non AWS hosted HTTP based operations accessed via public internet
* Mapping template to generate static conetne to be returned, helping mock your APIs before the backend is ready
* You can expose an api method in API Gateway that sends data directly to Amazon Kinesis
* Client platforms – api gateway generates SDKs – mobile apps with android and iOS and web app development with JavaScript, SDKs for ruby and java
* Client SDKs are only generated for REST APIs in Amazon API Gatway
* Resource <=> data model
* GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
* Usage plans for third party developers , define throttling, quota limits and association with API keys, you can extract utilization data on an per-API key bases
* Basic, professional and enterprise plans , basic plan with 1000 requests per day and maximum of 5 requests per second RPS
* Rest API – multiple stages dev and production
* Stage is similar to tabs ; defines the path
* Stage variables – key value pairs defined on the stage ; environment variables
* Resource policy - json document that can be attached to the API
* You can roll back a stage to previous deployment
* You can use open source swagger importer tools
* Monetize by publishing APIs as products in AWS marketplace
* API Gateway conforms to Open API specification
* API Gateway supports documentation inheritance
* Apply resource policy to an API to restrict access to a specific amazon VPC or VPC endpoint

**Security and Authorization**

* AWS Signature Version 4 or Lambda authorizers to support your own bearer token auth strategy
* You can retrieve temporary credentials associated with a role in your AWS account using Amazon Cognito.
* You can use AWS credentials - access and secret keys - to sign requests to your service and authorize access like other AWS services.
* When an API is called, API Gateway checks if a Lambda authorizer is configured, API Gateway then calls the Lambda function with the incoming authorization token. You can use Lambda to implement various authorization strategies (e.g. JWT verification, OAuth provider callout) that return IAM policies which are used to authorize the request. If the policy returned by the authorizer is valid, API Gateway will cache the policy associated with the incoming token for up to 1 hour.
* API Gateway can generate API keys and associate them with an usage plan. Calls received from each API key are monitored and included in the Amazon CloudWatch Logs you can enable for each stage. However, we do not recommend you use API keys for authorization. You should use API keys to monitor usage by third-party developers and leverage a stronger mechanism for authorization, such as signed API calls or OAuth
* API Gateway supports throttling settings for each method or route in your APIs. You can set a standard rate limit and a burst rate limit per second for each method in your REST APIs and each route in WebSocket APIs. Further, API Gateway automatically protects your backend systems from distributed denial-of-service (DDoS) attacks, whether attacked with counterfeit requests (Layer 7) or SYN floods (Layer 3)
* Amazon API Gateway can generate a client-side SSL certificate and make the public key of that certificate available to you
* Amazon API Gateway is integrated with [AWS CloudTrail](https://aws.amazon.com/cloudtrail/) to give you a full auditable history of the changes to your REST APIs
* In Amazon API Gateway, you can proxy requests to backend HTTP/HTTPS resources running in your Amazon VPC by setting up [Private Integrations](https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-private-integration.html) using VPC Links.
* You can also create Private APIs in Amazon API Gateway which can only be accessible by resources within your Amazon VPC through Amazon VPC Endpoints.
* You can apply a Resource Policy to an API to restrict access to a specific Amazon VPC or VPC endpoint. You can also give an Amazon VPC or VPC endpoint from a different account access to the Private API using a Resource Policy
* If you’re using REST APIs, you can set up a CloudFront distribution with custom SSL certificate in your account and use it with [Regional APIs](https://docs.aws.amazon.com/apigateway/latest/developerguide/create-regional-api.html) in API Gateway. You can then configure the [Security Policy](https://aws.amazon.com/about-aws/whats-new/2017/09/amazon-cloudfront-now-lets-you-select-a-security-policy-with-minimum-tls-v1_1-1_2-and-security-ciphers-for-viewer-connections/) for the CloudFront distribution with TLS 1.1 or higher based on your security and compliance requirement

**Management, Metrics, and Logging**

* Amazon API Gateway logs API calls, latency, and error rates to Amazon CloudWatch in your AWS account.
* Amazon API Gateway sends logging information and metrics to Amazon CloudWatch. You can utilize the Amazon CloudWatch console to set up custom alarms
* By default, Amazon API Gateway monitors traffic at a REST API level. Optionally, you can enable detailed metrics for each method in your REST API from the deployment configuration APIs or console screen.
* I can determine which version of the API my customers are using. Metric details are specified by REST API and stage. Additionally, you can enable metrics for each method in your REST API
* Amazon API Gateway integrates with Amazon CloudWatch Logs. You can optionally enable logging for each stage in your API. For each method in your REST APIs, you can set the verbosity of the logging, and if full request and response data should be logged
* Logs, alarms, error rates and other metrics are stored in Amazon CloudWatch and are available near real time

## Throttling and Caching

* throttling at multiple levels including global and by service call.
* Throttling limits can be set for standard rates and bursts.  API owners can set a rate limit of 1,000 requests per second for a specific method in their REST APIs, and also configure Amazon API Gateway to handle a burst of 2,000 requests per second for a few seconds. Any requests over the limit will receive a 429 HTTP response.
* with usage plans you can set throttling limits for individual API keys
* Throttling ensures that API traffic is controlled to help your backend services maintain performance and availability.
* Throttling rate limits can be set at the method level. You can edit the throttling limits in your method settings through the Amazon API Gateway APIs or in the Amazon API Gateway console
* API Gateway checks against your AWS account limit. If the traffic is below the set account limit, API Gateway checks the limit you have set on a stage or method. If the traffic is below the stage limit, then API Gateway applies the usage plans limits you set on a per-API key basis
* You can add caching to API calls by provisioning an API Gateway cache and specifying its size in gigabytes.
* If caching is not enabled and throttling limits have not been applied, then all requests will pass through to your backend service until the account level throttling limits are reached. If throttling limits are in place, then Amazon API Gateway will shed the necessary amount of requests and send only the defined limit to your back-end service. If a cache is configured, then Amazon API Gateway will return a cached response for duplicate requests for a customizable time, but only if under configured throttling limits. This balance between the backend and client ensures optimal performance of the APIs for the applications that it supports. Requests that are throttled will be automatically retried by the client-side SDKs generated by Amazon API Gateway. By default, Amazon API Gateway does not set any cache on your API methods.
* Amazon API Gateway acts as a proxy to the backend operations that you have configured. Amazon API Gateway will automatically scale to handle the amount of traffic your API receives.

## Billing

* Amazon API Gateway bills per million API calls, plus the cost of data transfer out, in gigabytes. If you choose to provision a cache for your API, hourly rates apply. For WebSocket APIs, API Gateway bills based on messages sent and received and the number of minutes a client is connected to the API.
* The API owner is charged for the calls to their APIs on API Gateway
* API calls are counted equally for billing purposes whether the response is handled by your backend operations or the Amazon API Gateway caching operation

## WebSocket APIs

* WebSocket routing in Amazon API Gateway is used to correctly route the messages to a specific integration. You specify a routing key and integration backend to invoke when defining your WebSocket API. The routing key is an attribute in the message body. A default integration can also be set for non-matching routing keys.
* When a new client is connected to the WebSocket API, a unique URL, called the callback URL, is created for that client. You can use this callback URL to send messages to the client from the backend service
* With Amazon API Gateway, you can either use IAM roles and policies or AWS Lambda Authorizers to authorize access to your WebSocket APIs
* When a client is connected or disconnected, a message will be sent from the Amazon API Gateway service to your backend AWS Lambda function or your HTTP endpoint using the $connect and $disconnect routes. You can take appropriate actions like adding or removing the client for the list of connected users
* You can use the callback URL GET method on the connection to identify if the client is connected to the WebSocket connection
* you can disconnect the connected client from your backend service using the callback UR
* The maximum supported message size is 128 KB.
* You will be charged based on 2 metrics: Connection minutes and messages.
* Connection minutes: Total number of minutes the clients or devices are connected to the WebSocket connection (rounded to a minute).
* Messages: Total number of messages sent to and received from connected clients. Messages are charged in increments of 32KB.
* No, if messages on the WebSocket connection fail authentication or authorization, they do not count toward your API usage bill