Networking & Content Delivery

R53, CloudFront, WAF, API gateway

AWS Certified Solutions Architect– Professional (SAP-C01) -
Study notes - Sep’2019

Route53

R53 - Choosing between Alias and Non-alias record

https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/resource-record-sets-choosing-alias-non-alias.html

Route 53 *alias records* provide a Route 53–specific extension to DNS functionality. Alias records let you route traffic to selected AWS resources, such as CloudFront distributions and Amazon S3 buckets. They also let you route traffic from one record in a hosted zone to another record.

Unlike a CNAME record, you can create an alias record at the top node of a DNS namespace, also known as the *zone apex*. For example, if you register the DNS name example.com, the zone apex is example.com. You can't create a CNAME record for example.com, but you can create an alias record for example.com that routes traffic to www.example.com.

When R53 receives a DNS query for an alias record, Route 53 responds with the applicable value for that resource:

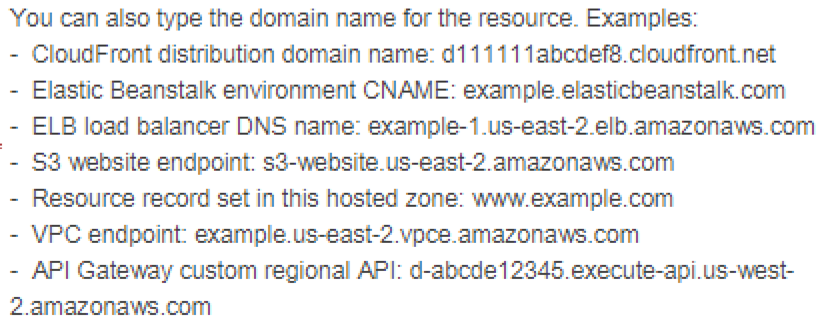
- **An Amazon API Gateway custom regional API or edge-optimized API** – Route 53 responds with one or more IP addresses for your API.
- **An Amazon VPC interface endpoint** – responds with one or more IP addresses for your interface endpoint.
- **A CloudFront distribution** – Route 53 responds with one or more IP addresses for CloudFront edge servers that can serve your content.
- **An Elastic Beanstalk environment** – Route 53 responds with one or more IP addresses for the environment.
- **An ELB load balancer** – Route 53 responds with one or more IP addresses for the load balancer.
- **An Amazon S3 bucket that is configured as a static website** – Route 53 responds with one IP address for the Amazon S3 bucket.
- **Another Route 53 record in the same hosted zone** – Route 53 responds as if the query is for the record that is referenced by the alias record.

When you use an alias record to route traffic to an AWS resource, Route 53 automatically recognizes changes in the resource. For example, suppose an alias record for example.com points to an ELB load balancer at lb1-1234.us-east-2.elb.amazonaws.com. If the IP address of the load balancer changes, Route 53 automatically starts to respond to DNS queries using the new IP address.

R53 - Routing traffic

https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routi

Alias Target →



You can also type the domain name for the resource. Examples:

- CloudFront distribution domain name: d111111abcdef8.cloudfront.net
- Elastic Beanstalk environment CNAME: example.elasticbeanstalk.com
- ELB load balancer DNS name: example-1.us-east-2.elb.amazonaws.com
- S3 website endpoint: s3-website.us-east-2.amazonaws.com
- Resource record set in this hosted zone: www.example.com
- VPC endpoint: example.us-east-2.vpce.amazonaws.com
- API Gateway custom regional API: d-abcde12345.execute-api.us-west-2.amazonaws.com

- To **API Gateway** API → Use R53 to create an alias record. An alias record is a R53 extension to DNS. It's similar to a CNAME record, but you can create an alias record both for the root and for subdomains (You can create CNAME records only for subdomains.)
- To **CloudFront** (Only for public hosted zones) → Create a CF Web distribution. If you want to use your own domain name, use R53 to create an alias record that points to your CF distribution. When Route 53 receives a DNS query that matches the name and type of an alias record, Route 53 responds with the domain name that is associated with your distribution.
- To **ELB/Elastic Beanstalk** → create an alias record that points to your load balancer/EBStalk
- To **S3 bucket** →
  - **If you DO NOT want SSL/TLS** → Point to an S3 bucket configured to host a static web site. The bucket name should be same as the domain name
  - **If you want SSL/TLS** → An S3 bucket that's *not* configured to host a static website, and a CloudFront distribution that's configured to use your S3 bucket as the origin. An S3 bucket that's configured as a website endpoint doesn't support SSL/TLS, so you need to route traffic to the CloudFront distribution and use the S3 bucket as the origin for the distribution.
- To a **VPC interface endpoint** → Use AWS PrivateLink to access selected services. These services include some AWS services, services that are hosted by other AWS customers and partners in their own VPCs, and supported AWS Marketplace partner services. To route domain traffic to an interface endpoint, use Amazon Route 53 to create an alias record.

Route53 - Routing policy

https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-policy.html

- **Simple routing policy** – Use for a single resource that performs a given function for your domain, for example, a web server that serves content for the example.com website.
- **Failover routing policy** – Use when you want to configure **active-passive failover**.
- **Geolocation routing policy** – Use when you want to route traffic based on the location of your users.
- **Geoproximity routing policy** – Use when you want to route traffic based on the location of your resources and, optionally, shift traffic from resources in one location to resources in another.
- **Latency routing policy** – Use when you have resources in multiple AWS Regions and you want to route traffic to the region that provides the best latency.
- **Multivalue answer routing policy** – Use when you want Route 53 to respond to DNS queries with up to eight healthy records selected at random.
- **Weighted routing policy** – Use to route traffic to multiple resources in proportions that you specify.

  For example, if you want to send a portion of your traffic to one resource and the rest to another resource, you might specify weights of 1 and 4. The resource with a weight of 1 gets 1/5th of the traffic (1/1+4), and the other resource gets 4/5ths (4/1+5). You can gradually change the balance by changing the weights. If you want to stop sending traffic to a resource, you can change the weight for that record to 0.

R53 - hosted zones - Public vs Private hosted zones

https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/hosted-zone-public-considerations.html

Considerations When Working with a **Private** Hosted Zone:

Amazon VPC settings - set enableDnsHostnames & enableDnsSupport to true

Route 53 health checks - you can associate Route 53 health checks **only with weighted and failover** records.

Routing policies - Only **Simple**, **Failover**, **Multivalue answer** and **Weighted** are supported

Split-view DNS - You can use same domain name for both private and public zone

Associating a VPC with more than one private hosted zone - Yes its allowed

Public and private hosted zones that have overlapping namespaces - first checks for identical match in private zone. If not found, forwards to public DNS resolver

Custom DNS servers - Configure your DNS server to fwd to amz dns server (VPC IP range +2)

Considerations When Working with a **Public** Hosted Zone:

NS and SOA Records - When a public hosted zone is created, R53 automatically creates a Name server (NS- 4 name servers) and Start of Authority (SOA- base dns info) record. You use the NS record with your registrar to forward all requests to R53 name servers

Multiple Hosted Zones That Have the Same Name - You can create more than 1 public zone with the same domain name but different name records (servers).

Reusable Delegation Sets - is name for set of 4 name servers. You can create delegation set programmatically if needed in bulk.

R53 - hosted zones - private zone with multiple VPCs

https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/hosted-zones-working-with.html

A hosted zone is a container for records, and records contain information about how you want to route traffic for a specific domain, such as example.social, and its subdomains (admin.example.social). A hosted zone and the corresponding domain have the same name. There are two types of hosted zones:

- *Public hosted zones* contain records that specify how you want to route traffic on the internet.
- *Private hosted zones* contain records that specify how you want to route traffic in an Amazon VPC.

Important

When you create a private hosted zone, you must associate a VPC with the hosted zone, and the VPC that you specify must have been created by using the same account that you're using to create the hosted zone. After you create the hosted zone, you can associate additional VPCs with it, including VPCs that you created by using a different AWS account.

Associating an Amazon VPC and a Private Hosted Zone That You Created with Different AWS Accounts

https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/hosted-zone-private-associate-vpcs-different-accounts.html

If you want to associate VPCs that you created by using one account with a private hosted zone that you created by using a different account, you first must authorize the association.

1. Using the account that created the hosted zone, authorize the association of the VPC with the private hosted zone with 1 of foll: **AWS SDK** or **AWS Tools for Windows PowerShell**, **AWS CLI** or **Amazon Route 53 API**. Note:
   - To associate multiple VPCs in different accounts, submit one authorization request for each VPC.
   - You must specify the hosted zone ID, so the private hosted zone must already exist.
   - You can't use the Route 53 console either to authorize this association
2. Using the account that created the VPC, associate the VPC with the hosted zone with above 3 methods
3. *Optional but recommended* – Delete the authorization to associate the VPC with the hosted zone. Deleting the authorization does not affect the association, it just prevents you from reassociating the VPC with the hosted zone in the future. If you want to reassociate the VPC with the hosted zone, you'll need to repeat steps 1 and 2 of this procedure.

Active-Active and Active-Passive Failover

https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/dns-failover-types.html

- You configure Active-Active failover using any routing policy (or combination) **other than failover**,
- You configure Active-Passive failover using the failover routing policy.

Active-Passive Failover

- **Configuring Active-Passive Failover with One Primary and One Secondary Resource**

To create an active-passive failover configuration with one primary record and one secondary record, you just create the records and specify **Failover** for the routing policy.

- **Configuring Active-Passive Failover with Multiple Primary and Secondary Resources**

R53 considers the primary failover record to be healthy as long as at least one of the associated resources is healthy.

1. Create a health check for each resource that you want to route traffic to, such as an EC2 instance or a web server.
2. Create records for your primary resources (X)
3. Create records for your secondary resources (Y)
4. Create two failover alias (A) records, one primary and one secondary:
   a. **Primary record** → Create Alias record pointing to X, Failover Record Type - **Primary**, Routing policy: **Failover**, Evaluate Target Health - **Yes**, Associate with Health Check - **No**.
   b. **Secondary record** → Create Alias record pointing to Y or S3 endpoint, Failover Record Type - **Secondary**, Routing policy: **Failover**, Evaluate Target Health - **Yes**, Associate with Health Check - **No**.

- **Configuring Active-Passive Failover with Weighted Records**

You can also use weighted records for active-passive failover, with caveats. If you specify nonzero weights for some records and zero weights for other records, Route 53 responds to DNS queries using only healthy records that have nonzero weights. If all the records that have a weight greater than 0 are unhealthy, then Route 53 responds to queries using the zero-weighted records.

Types of Route 53 Health Checks

https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/health-checks-types.html

Health checks that monitor an endpoint

Monitor an endpoint that you specify either by IP address or by domain name. At regular intervals that you specify, Route 53 submits automated requests over the internet to your application, server, or other resource to verify that it's reachable, available, and functional.

Health checks that monitor other health checks (calculated health checks)

Monitors whether Route 53 considers other health checks healthy or unhealthy. E.g. You can create a health check for each resource without configuring notification for those health checks. Then you can create a health check that monitors the status of the other health checks

Health checks that monitor CloudWatch alarms

Create CloudWatch alarms that monitor the status of CloudWatch metrics, such as the number of throttled read events for an DynamoDB or the number of ELB hosts that are considered healthy. After you create an alarm, you can create a health check that monitors the same data stream that CloudWatch monitors for the alarm.

- **HTTP and HTTPS health checks** – Route 53 must be able to establish a TCP connection with the endpoint **within four seconds**. In addition, the endpoint must respond with an HTTP status code of 2xx or 3xx within two seconds after connecting.
- **TCP health checks** – Route 53 must be able to establish a TCP connection with the endpoint **within ten seconds**.
- **HTTP and HTTPS health checks with string matching** – As with HTTP and HTTPS health checks, Route 53 must be able to establish a TCP connection with the endpoint **within four seconds**, and the endpoint must respond with an **HTTP status code of 2xx or 3xx within two seconds** after connecting.
  After a Route 53 health checker receives the HTTP status code, it must receive the response body from the endpoint within the next two seconds. Route 53 searches the response body for a string that you specify. **The string must appear entirely in the first 5,120 bytes of the response body** or the endpoint fails the health check. If you're using the Route 53 console, you specify the string in the **Search String** field. If you're using the Route 53 API, you specify the string in the SearchString element when you create the health check.

  For health checks that monitor an endpoint (except TCP health checks), if the response from the endpoint includes any headers, the headers must be in the format that is defined in RFC7230

How Health Checks Work in Simple Amazon Route 53 Configurations

https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/dns-failover-simple-configs.html

Check the health of EC2 instances and other resources (non-alias records)

If you're routing traffic to resources that you can't create alias records for, such as EC2 instances, you create a record and a health check for each resource.

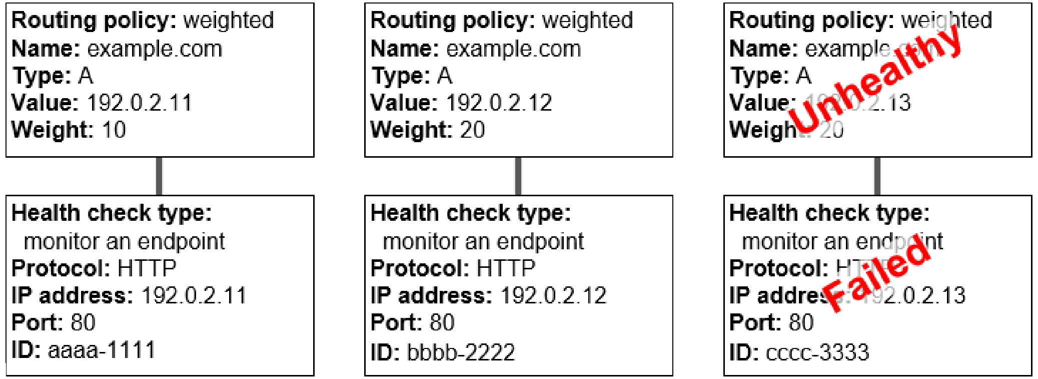
Evaluate the health of an AWS resource (alias records)

If you're using alias records to route traffic to selected AWS resources, such as ELB, you can configure R53 to evaluate the health of the resource (Specify **Yes** for **Evaluate Target Health**) and to route traffic only to resources that are healthy. You don't need to create a health check for the resource.

The following example shows a group of weighted records in which the third record is unhealthy.

- When R53 initially selects from among all three records, it responds to requests using the first record about 20% of the time, 10/(10 + 20 + 20).
- When Route 53 determines that the third record is unhealthy, it responds to requests using the first record about 33% of the time, 10/(10 + 20).

If you omit a health check from one or more records in a group of records, Route 53 has no way to determine the health of the corresponding resource. Route 53 treats those records as healthy.



Routing policy: weighted
Name: example.com
Type: A
Value: 192.0.2.11
Weight: 10

Health check type: monitor an endpoint
Protocol: HTTP
IP address: 192.0.2.11
Port: 80
ID: aaaa-1111

Routing policy: weighted
Name: example.com
Type: A
Value: 192.0.2.12
Weight: 20

Health check type: monitor an endpoint
Protocol: HTTP
IP address: 192.0.2.12
Port: 80
ID: bbbb-2222

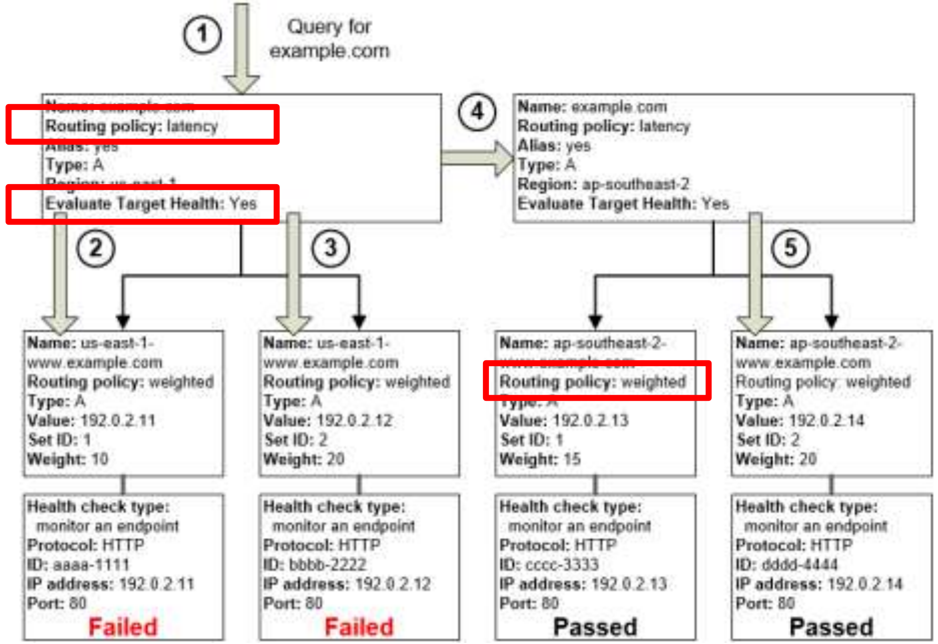
Routing policy: weighted
Name: example.com
Type: A
Value: 192.0.2.13
Weight: 20

Health check type: monitor an endpoint
Protocol: HTTP
IP address: 192.0.2.13
Port: 80
ID: cccc-3333

How Health Checks Work in Complex Amazon Route 53 Configurations

https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/dns-failover-complex-configs.html

In complex configurations, you **use a combination of alias records and non-alias records** to build a decision tree that gives you greater control over how R53 responds to requests.
e.g. you **might use latency alias records** to select a region close to a user and **use weighted records** for two or more resources within each region to protect against the failure of a single endpoint or an AZ.



Name: example.com
Routing policy: latency
Alias: yes
Type: A
Region: us-east-1
Evaluate Target Health: Yes

Name: example.com
Routing policy: latency
Alias: yes
Type: A
Region: ap-southeast-2
Evaluate Target Health: Yes

Name: us-east-1-www.example.com
Routing policy: weighted
Type: A
Value: 192.0.2.11
Set ID: 1
Weight: 10

Health check type: monitor an endpoint
Protocol: HTTP
ID: aaaa-1111
IP address: 192.0.2.11
Port: 80
Failed

Name: us-east-1-www.example.com
Routing policy: weighted
Type: A
Value: 192.0.2.12
Set ID: 2
Weight: 20

Health check type: monitor an endpoint
Protocol: HTTP
ID: bbbb-2222
IP address: 192.0.2.12
Port: 80
Failed

Name: ap-southeast-2-www.example.com
Routing policy: weighted
Type: A
Value: 192.0.2.13
Set ID: 1
Weight: 15

Health check type: monitor an endpoint
Protocol: HTTP
ID: cccc-3333
IP address: 192.0.2.13
Port: 80
Passed

Name: ap-southeast-2-www.example.com
Routing policy: weighted
Type: A
Value: 192.0.2.14
Set ID: 2
Weight: 20

Health check type: monitor an endpoint
Protocol: HTTP
ID: dddd-4444
IP address: 192.0.2.14
Port: 80
Passed

What Happens When You Associate a Health Check with an Alias Record?

You can associate a health check with an alias record instead of or in addition to setting the value of **Evaluate Target Health** to **Yes**. However, it's generally more useful if Route 53 responds to queries based on the health of the underlying resources—the HTTP servers, database servers, and other resources that your alias records refer to.

What Happens When You Omit Health Checks?

In a complex configuration, if you omit a health check from non-alias records, R53 has no way to determine the health of the corresponding resource. Route 53 treats those records as healthy.

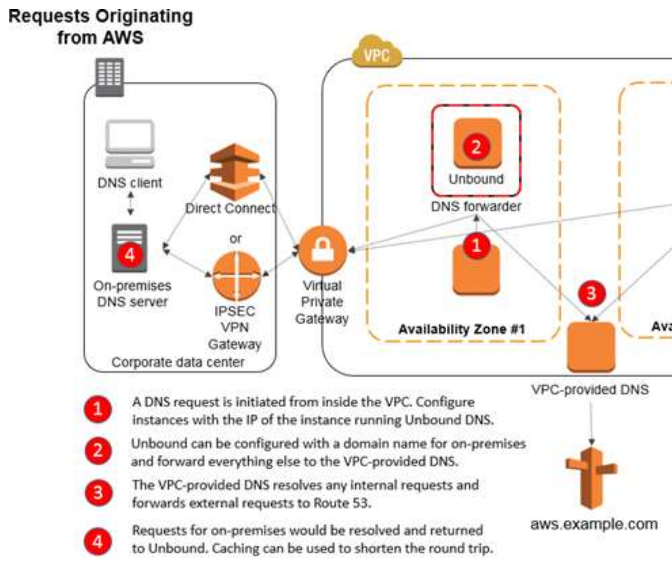
What Happens When You Set Evaluate Target Health to No?

In general, you should set **Evaluate Target Health** to **Yes** for all the alias records in a tree. If you set **Evaluate Target Health** to **No**, Route 53 continues to route traffic to the records that an alias record refers to even if health checks for those records are failing.

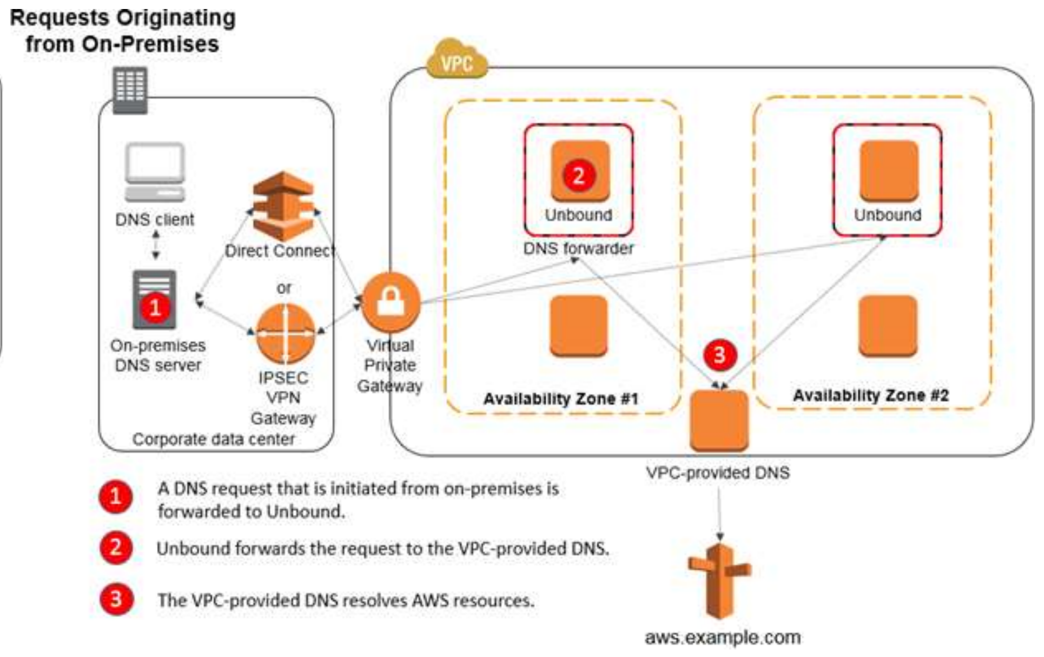
Set Up DNS Resolution Between On-Premises Networks and AWS by Using Unbound

https://aws.amazon.com/blogs/security/how-to-set-up-dns-resolution-between-on-premises-networks-and-aws-by-using-unbound/

Set up DNS resolution between your on-premises DNS with Amazon VPC by using Unbound, an open-source, recursive DNS resolver. This solution is not a managed solution like Microsoft AD and Simple AD, but it does provide the ability to route DNS requests between on-premises environments and an Amazon VPC–provided DNS.



1. A DNS request is initiated from inside the VPC. Configure instances with the IP of the instance running Unbound DNS.
2. Unbound can be configured with a domain name for on-premises and forward everything else to the VPC-provided DNS.
3. The VPC-provided DNS resolves any internal requests and forwards external requests to Route 53.
4. Requests for on-premises would be resolved and returned to Unbound. Caching can be used to shorten the round trip.



1. A DNS request that is initiated from on-premises is forwarded to Unbound.
2. Unbound forwards the request to the VPC-provided DNS.
3. The VPC-provided DNS resolves AWS resources.

API Gateway

AWS API Gateway

https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

Pricing: Post free tier, 3.5 USD for 1m calls for first 333m calls, dedicated cache (optional) 0.038 USD for 1.6 GB

- Support for stateful (WebSocket) and stateless (REST) APIs.
- Powerful, flexible authentication mechanisms, such as AWS IAM policies, Lambda authorizer functions, and Cognito user pools.
- Developer portal for publishing your APIs.
- Canary release deployments for safely rolling out changes.
- CloudTrail logging and monitoring of API usage and API changes.
- CloudWatch access logging and execution logging, including the ability to set alarms.
- Ability to use AWS CloudFormation templates to enable API creation.
- Support for custom domain names.
- Integration with AWS WAF for protecting your APIs against common web exploits.
- Integration with AWS X-Ray for understanding and triaging performance latencies.

REST APIs are made up of resources (/transactions or /activityfeed) and methods (POST /transactions, GET /activityfeed)
https://zmbcgpqtgc.execute-api.us-east-1.amazonaws.com/test/activityfeed

Method requests /responses refer to public interface of REST API method that defines the parameters, body AND status codes, headers & body (*consumers will use this*). Lambda authorizers can be added in the Method request

Integration requests / response refer to internal interface in which you map the body of a route request or the parameters and body of a method request to the formats required by the backend. Integration request support lambda, http, mock, aws service or vpc link. lambda proxy integration will send entire request in event object of handler function. For custom mapping, use mapping template.

Async lambda invocation:

1. In **Integration Request**, add an X-Amz-Invocation-Type header.
2. In **Method Request**, add an InvocationType header and map it to the X-Amz-Invocation-Type header in the **Integration Request** with either a static value of 'Event' or the header mapping expression of method.request.header.InvocationType. For the latter, the client must include the InvocationType:Event header when making a request to the API method.

AWS API Gateway - Proxy lambdas vs Custom integration

https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

Lambda Proxy integration request

```
{
  "resource": "Resource path",
  "path": "Path parameter",
  "httpMethod": "Incoming request's method name"
  "headers": {String containing incoming request headers}
  "multiValueHeaders": {List of strings containing incoming request headers}
  "queryStringParameters": {query string parameters }
  "multiValueQueryStringParameters": {List of query string parameters}
  "pathParameters":  {path parameters}
  "stageVariables": {Applicable stage variables}
  "requestContext": {Request context, including authorizer-returned key-value pairs}
  "body": "A JSON string of the request payload."
  "isBase64Encoded": "A boolean flag to indicate if the applicable request payload is Base64-encode"
}
```

Error response
(statusCode mapped by APIG to http response codes)

```
{
 "isBase64Encoded" : "boolean",
 "statusCode": "number",
 "headers": { ... },
 "body": "JSON string"
}
```

Lambda Proxy integration response

API Gateway requires the backend Lambda function to return output according to the following JSON format

```
{
  "isBase64Encoded": true|false,
  "statusCode": httpStatusCode,
  "headers": { "headerName": "headerValue", ... },
  "multiValueHeaders": { "headerName": ["headerValue", "headerValue2", ...], ...
},
  "body": "..."
}
```

To enable CORS for the Lambda proxy integration, you must add Access-Control-Allow-Origin:*domain-name* to the output headers. *domain-name* can be * for any domain name.

Custom integration Error response - sample error mapping from integration response to mathod response

```
"x-amazon-apigateway-integration": {
  "responses": {
    "default": {
     "statusCode": "200",
     "responseParameters": {
     "method.response.header.error_trace_function": "integration.response.body.errorMessage.trace.function",
      "method.response.header.error_status": "integration.response.body.errorMessage.httpStatus",
      "method.response.header.error_type": "integration.response.body.errorMessage.errorType",
      "method.response.header.error_trace": "integration.response.body.errorMessage.trace"
     },
     ...
    }
  }
```

AWS API Gateway - Custom domain name

https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-custom-domains.html

Instead of https://*api-id*.execute-api.*region*.amazonaws.com/*stage*, *use* https://api.example.com/myservice

A custom domain can be associated with a REST API or a Websocket API, but not both.

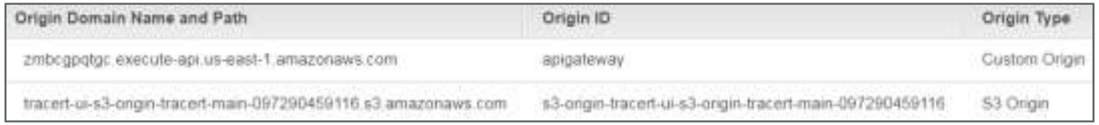
Custom domain names are not supported for private APIs.

Edge-optimized API gateway - uses an internal CloudFront distribution. For default domains, Api gateway automatically creates a DNS record to map the Api domain name to the CloudFront domain. For custom domains, Manually create the DNS record to map the Api domain name to CloudFront. Also attach certificate

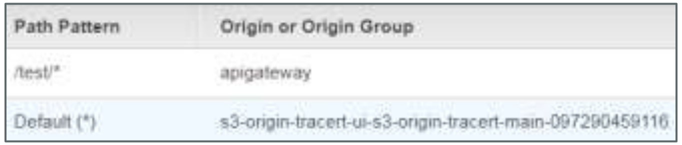
Regional API - For custom domain name, API Gateway creates a regional domain name for the API. You must set up a DNS record to map the custom domain name to the regional domain name through the mapped regional API endpoint. Also attach certificate.

OR Setup R53 → CloudFront → API gateway / S3

★ Register domain in R53 (or 3rd party registrar)
★ Request certificate in ACM (or import 3rd party)
★ API Gateway - Create Regional API, Add Custom domain Name (traceroute.social), Attach above certificate
★ API Gateway - Create APIs (*Messages*), Resources (*/activityfeed*), Methods (*GET /activityfeed*)
★ CloudFront - Create CloudFront distribution, Add Alternate CNAME (traceroute.social), Attach above certificate
★ CloudFront - Add 2 Origins (Api gateway and S3), Add behaviour for routing requests between the two



Origin Domain Name and Path	Origin ID	Origin Type
zmbcgpqtgc.execute-api.us-east-1.amazonaws.com	apigateway	Custom Origin
tracert-ui-s3-origin-tracert-main-097290459116.s3.amazonaws.com	s3-origin-tracert-ui-s3-origin-tracert-main-097290459116	S3 Origin



Path Pattern	Origin or Origin Group
/test/*	apigateway
Default (*)	s3-origin-tracert-ui-s3-origin-tracert-main-097290459116

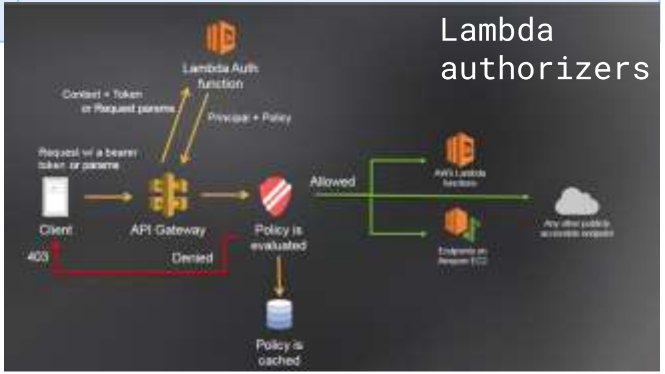
★ Route53 - Create Public Hosted zone, Add ALIAS Record to point to above CloudFront
★ Route53 - On issuance of ACM certificate, you can directly create CNAME records in R53 from ACM for resolving certificate
★ Route53 - NS and SOA records are directly created by R53 on creation

AWS API Gateway - Securing Apis

AWS API Gateway supports following 4 mechanisms for controlling and managing access to an API.
At a high level, we can apply 2 schemes: One, at the API level and Any one of 3 METHOD level schemes

	API level	METHOD level		
			Authorizers	
Scheme	Resource policies	IAM policies	Lambda authorizers	Cognito
Type	JSON Policy documents	JSON Policy documents	Bearer Token (Oauth or SAML) OR Request parameters	JWT Bearer Token
Use case	Restrict API usage based on: 1.IAM users from a specified AWS account 2.specified source IP ranges or CIDR blocks 3.specified VPCs or VPC endpoints (in any account)	Restrict API usage based on: IAM users from same account	Custom Authentication scheme	Restrict API usage based on: Custom scopes of specified access-protected resources
Applicability	Use this as an additional check for source VPC or IP range checks		**Alternative 2:** 1.Use authorizer 2.Alternatively, use request signing with key pair per account 3.Validate signature in lambda authorizer	**Alternative 1:** Use User pools as IdP or Identity pools for 3rd party IdPs, Use Client credentials flow with clients

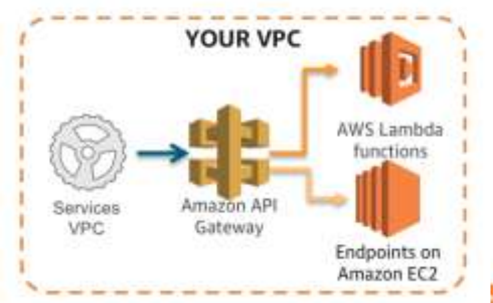
Use **AWS WAF** to protect gateway from common web exploits, such as SQL injection and XSS attacks. e.g you can create rules to allow or block requests from specified IP address ranges or CIDR blocks or originating from a specific country or region, containing malicious SQL code, or containing malicious script. You can also create rules that match a specified string or a regular expression pattern in HTTP headers, method, query string, URI, and the request body. Additionally, create rules to block attacks from specific user-agents, bad bots, and content scrapers.
AWS WAF is your first line of defense against web exploits. When AWS WAF is enabled on an API, AWS WAF rules are evaluated before other features, such as resource policies, IAM policies, Lambda authorizers, and Amazon Cognito authorizers.



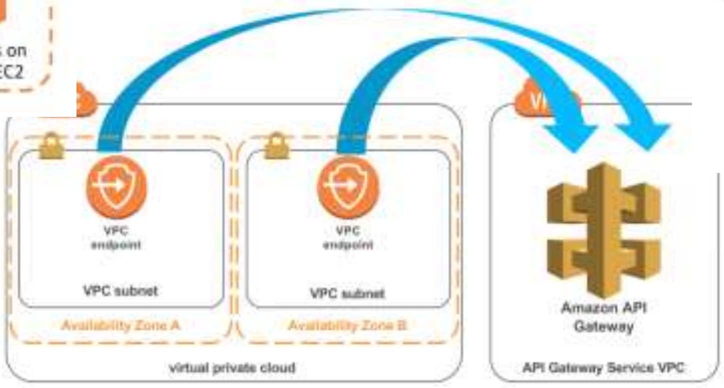
AWS API Gateway - Public vs Private Api endpoints

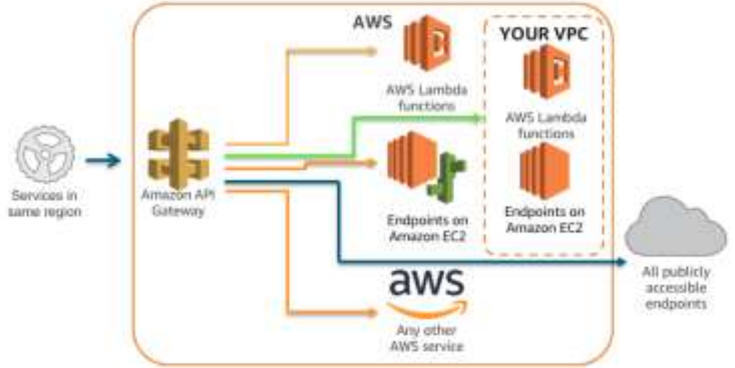
API Gateway private endpoints are made possible via AWS PrivateLink interface VPC endpoints. Interface endpoints work by creating elastic network interfaces in subnets that you define inside your VPC. Those network interfaces then provide access to services running in other VPCs, or to AWS services such as API Gateway. When configuring your interface endpoints, you specify which service traffic should go through them. When using private DNS, all traffic to that service is directed to the interface endpoint instead of through a default route, such as through a NAT gateway or public IP address.

API Gateway as a fully managed service runs its infrastructure in its own VPCs. When you interface with API Gateway publicly accessible endpoints, it is done through public networks. When they're configured as private, the public networks are not made available to route your API. Instead, your API can only be accessed using the interface endpoints that you have configured.



Private Api endpoints





Public Api endpoints

AWS API Gateway - Monitoring

AWS X-Ray, AWS CloudTrail, and Amazon CloudWatch are tools that Amazon API Gateway developers can use to trace, log, and monitor API execution and management operations.

AWS X-Ray is an AWS service that allows you to trace latency issues with your Amazon API Gateway APIs. X-Ray collects metadata from the API Gateway service and any downstream services that make up your API. X-Ray uses this metadata to generate a detailed service graph that illustrates latency spikes and other issues that impact the performance of your API.

Amazon CloudWatch logs API execution operations, which are calls that an API customer or client application makes against the API Gateway execute-api component. CloudWatch metrics include statistics about caching, latency and detected errors. You can inspect CloudWatch logs to troubleshoot your API implementation or execution using the API dashboard in the API Gateway console or using the CloudWatch console.

AWS CloudTrail logs API Gateway API management operations, which are REST API calls that an API developer or owner makes against the API Gateway apigateway component. You can use CloudTrail logs to troubleshoot API creation, deployment and updates. You can also use Amazon CloudWatch to monitor CloudTrail logs.

Difference b/w CLB, ALB, NLB

https://aws.amazon.com/elasticloadbalancing/features/

Benefits of Migrating from Classic to App. Load Balancer

- Support for path-based routing.
- Support for host-based routing.
- Support for routing based on fields in the request
- Support for routing requests to multiple applications on a single EC2 instance. You can register each instance or IP address with the same target group using multiple ports.
- Support for redirecting requests from one URL to another.
- Support for returning a custom HTTP response.
- Support for registering targets by IP address, including targets outside the VPC for the load balancer.
- Support for registering Lambda functions as targets.
- Support for the load balancer to authenticate users of your applications through their corporate or social identities before routing requests.
- Support for containerized applications. Amazon Elastic Container Service (Amazon ECS) can select an unused port when scheduling a task and register the task with a target group using this port. This enables you to make efficient use of your clusters.
- Support for monitoring the health of each service independently, as health checks are defined at the target group level and many CloudWatch metrics are reported at the target group level. Attaching a target group to an Auto Scaling group enables you to scale each service dynamically based on demand.
- Access logs contain additional information and are stored in compressed format.
- Improved load balancer performance.

Feature	ALB	NLB	CLB
Protocols	HTTP, HTTPS	TCP, UDP, TLS	TCP, SSL/TLS, HTTP, HTTPS
Platforms	VPC	VPC	EC2-Classic, VPC
Health checks	✔	✔	✔
CloudWatch metrics	✔	✔	✔
Logging	✔	✔	✔
Zonal fail-over	✔	✔	✔
Connection draining (deregistration delay)	✔	✔	✔
Load Balancing to multiple ports on the same instance (dynamic host port mapping imp for ECS)	✔	✔	
IP addresses as targets	✔	✔ (TCP, TLS)	
Load balancer deletion protection	✔	✔	
Configurable idle connection timeout	✔		✔
Cross-zone load balancing	✔	✔	✔
Sticky sessions	✔		✔
Static IP		✔	
Elastic IP address		✔	
Preserve Source IP address (Proxy protocol)		✔	
Resource-based IAM Permissions	✔	✔	✔
Tag-based IAM permissions	✔	✔	
Slow start	✔		
Web Application firewall (WAF) integration	✔		
WebSockets	✔	✔	
PrivateLink Support		✔ (TCP, TLS)	
Source IP address CIDR-based routing	✔		
Layer 7			
Path-Based Routing	✔		
Host-Based Routing	✔		
Native HTTP/2	✔		
Redirects	✔		
Fixed response	✔		
Lambda functions as targets	✔		
HTTP header-based routing	✔		
HTTP method-based routing	✔		
Query string parameter-based routing	✔		
Security			
SSL offloading	✔	✔	✔
Server Name Indication (SNI) multiple SSL certs	✔	✔	
Back-end server encryption	✔	✔	✔
User authentication (OIDC, SAML)	✔		

Multiple SSL certificates in a single ALB using SNI

https://aws.amazon.com/blogs/aws/new-application-load-balancer-sni/

You can now host multiple TLS secured applications, each with its own TLS certificate, behind a single load balancer. In order to use SNI, all you need to do is bind multiple certificates to the same secure listener on your load balancer. ALB will automatically choose the optimal TLS certificate for each client

How does SNI work?
Server Name Indication (SNI) is an extension of TLS and is used by the ALB to choose a certificate for a client. Since TLS operates at the transport layer, below HTTP, it doesn't see the hostname requested by a client. SNI works by having the client tell the server "This is the domain I expect to get a certificate for" when it first connects. The server can then choose the correct certificate to respond to the client. All modern web browsers and a large majority of other clients support SNI.

In addition to this, ALB also has SMART CERTIFICATE selection. To establish a TLS connection, a client starts a TLS handshake by sending a "ClientHello" message that outlines the capabilities of the client: the protocol versions, extensions, cipher suites, and compression methods. Based on what an individual client supports, ALB's smart selection algorithm chooses a certificate for the connection and sends it to the client. (either the older RSA or newer ECDSA algorithm)

Classic Load Balancer doesn't support adding multiple certificates. To add multiple certificates for different domains to a load balancer, do one of the following:

- Use a SAN certificate to validate multiple domains, including wildcard domains, with ACM OR
- Use an Application Load Balancer (ALB)
- Pass through TCP and terminate SSL in EC2 instances (not recommended)

CloudFront

CloudFront - Caching behaviour

https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/ConfiguringCaching.html

Improving Cache Hit ratio - Configure your origin to add a *Cache-Control max-age* directive to your objects

Caching Based on Query String Parameters (Only for web distributions, not RTMP):
You can configure CloudFront do one of the following:

- Cache Based Only on Parameters for Which Your Origin Returns Different Versions of an Object
- Always List Parameters in the Same Order
- Always Use the Same Case for Parameter Names and Values
- Don't Use Parameter Names that Conflict with Signed URLs

Caching Based on Cookie Values (Only for web distributions, not RTMP)

- For web distributions, CloudFront by default doesn't consider cookies when caching your objects in edge locations
- Amazon S3 and some HTTP servers do not process cookies. Do not configure CloudFront cache behaviors to forward cookies to an origin that doesn't process cookies, or you'll adversely affect cacheability / performance.
- You can configure each cache behavior in a web distribution to do one of the following:
  - Forward all cookies to your origin - caches based on all cookies, forwards all
  - Forward a whitelist of cookies that you specify - cache based on whitelisted cookies, forwards all
  - Don't forward cookies to your origin - also removes cookies from request and response

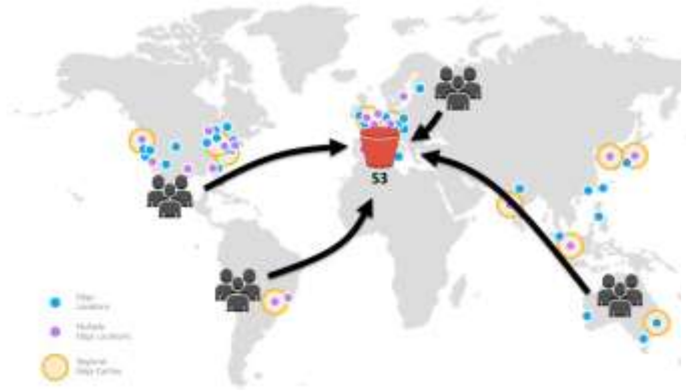
Caching Based on Request Headers (Only for web distributions, not RTMP)

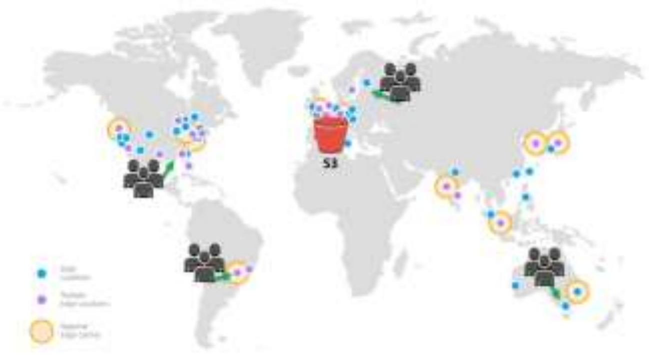
- Forward all headers to your origin - No caching
- Forward a whitelist of cookies that you specify - cache based on whitelisted headers, forwards all
- Forward only the default headers to your origin - No caching
- Caching behavior changes based on S3 or Custom origin
- It's possible to cache based on CORS setting, Device Type, Location, Language, Protocol, Compressed files,

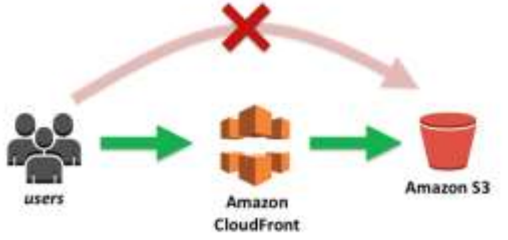
Remove Accept-Encoding Header When Compression is Not Needed
- If Request contains Accept-Encoding: gzip header → CF adds gzip– Accept-Encoding: gzip–to the cache key
- This behaviour ensures that CloudFront serves either an object or a compressed version of the object, based on the value of the Accept-Encoding header.

CloudFront and S3

https://aws.amazon.com/blogs/networking-and-content-delivery/amazon-s3-amazon-cloudfront-a-match-made-in-the-cloud/







Low latency & Performance

You can see that there are no longer requests traversing the globe to get to our content. Instead, **requests are routed to the “least latent” Edge Location**; that is, the closest in terms of delivery speed. CloudFront then serves cached content quickly and directly to the requesting user nearby, as shown with the green arrows. If the content is not yet cached with an edge server, CloudFront retrieves it from the S3 bucket origin. And because the content traverses the AWS private network instead of the public internet and CloudFront optimizes the TCP handshake, the request and content return is still much faster than access across the public internet.

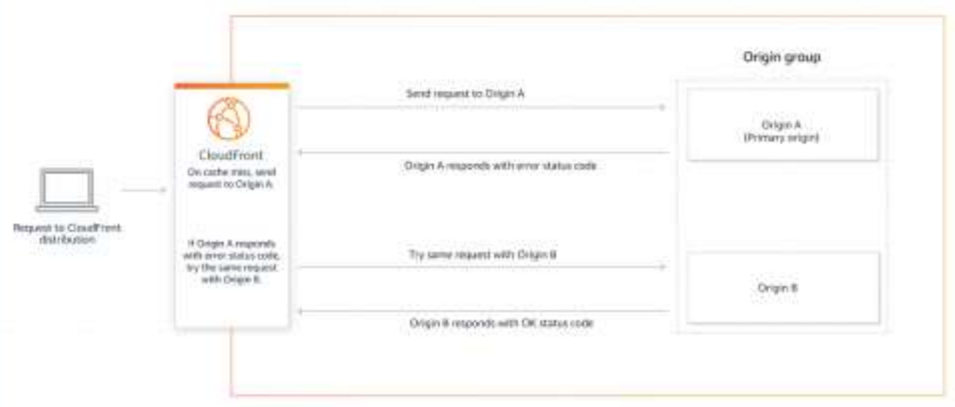
Security

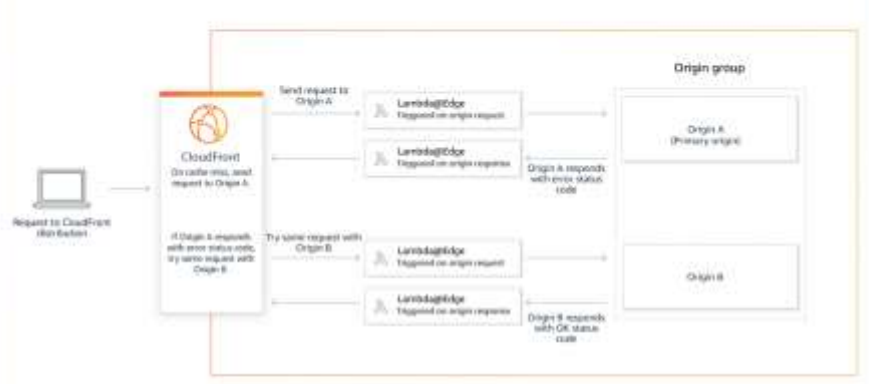
By using CloudFront, we can set up additional access restrictions like **geo-restrictions**, **signed URLs**, and **signed cookies**, to further constrain access to the content following different criteria.

Another security feature of CloudFront is **Origin Access Identity (OAI)**, which restricts access to an S3 bucket and its content to only CloudFront and operations it performs. CF also integrates with WAF and AWS Shield

Optimizing High Availability with CloudFront Origin Failover

https://docs.aws.amazon.com/en_pv/AmazonCloudFront/latest/DeveloperGuide/high_availability_origin_failover.html


Origin group
CloudFront
On cache miss, send request to Origin A
If Origin A responds with error status code, try the same request with Origin B
Request to CloudFront distribution
Send request to Origin A
Origin A responds with error status code
Try same request with Origin B
Origin B responds with OK status code
Origin A (Primary origin)
Origin B
Figure 1 : Origin failover on cache miss.

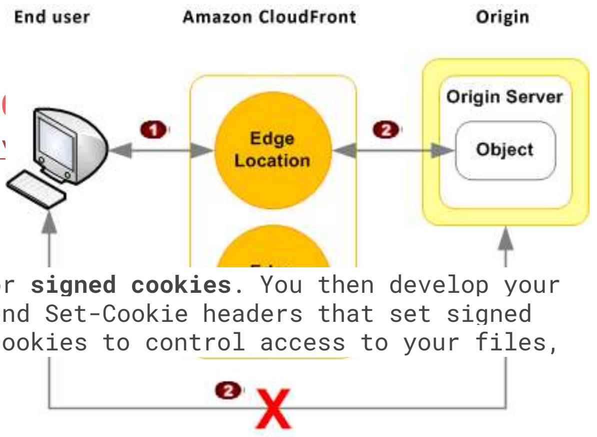

Origin group
CloudFront
On cache miss, send request to Origin A
If Origin A responds with error status code, try same request with Origin B
Request to CloudFront distribution
Send request to Origin A
Try same request with Origin B
Lambda@Edge
Triggered on origin request
Lambda@Edge
Triggered on origin response
Lambda@Edge
Triggered on origin request
Lambda@Edge
Triggered on origin response
Origin A responds with error status code
Origin B responds with OK status code
Origin A (Primary origin)
Origin B
Figure 2 : Origin failover with Lambda@Edge functions triggered on origin request and response events.

The two origins in the origin group can be any combination of the following: S3 buckets or EC2 instances, or custom origins, like your own HTTP web server. After you configure origin failover for a cache behavior, CloudFront does the following for viewer requests:

- When there's a cache hit, CloudFront returns the requested file.
- When there's a cache miss, CloudFront routes the request to the primary origin that you identified in the origin group.
- When a status code that has not been configured for failover is returned, such as an HTTP 2xx or HTTP 3xx status code, CloudFront serves the requested content.
- When the primary origin returns an HTTP status code that you've configured for failover, or after a timeout, CloudFront routes the request to the backup origin in the origin group.

Serving Private Content with Signed URLs and Signed C

https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Pri



1.Restricting Access to Files in CloudFront Edge Caches

Configure CloudFront to require that users access your files using either **signed URLs** or **signed cookies**. You then develop your application either to create and distribute signed URLs to authenticated users or to send Set-Cookie headers that set signed cookies on the viewers for authenticated users. When you create signed URLs or signed cookies to control access to your files, you can specify the following restrictions:

- An ending date and time, after which the URL is no longer valid.
- (Optional) The date and time that the URL becomes valid.
- (Optional) The IP address or range of addresses of the computers that can be used to access your content.

2.Restricting Access to Files in S3 Buckets

Secure the content in your S3 bucket so that users can access it through CloudFront but cannot access it directly by using Amazon S3 URLs. you do the following tasks:

- Create a special CloudFront user called an **origin access identity**.
- Give the origin access identity permission to read the files in your bucket.
- Remove permission for anyone else to use Amazon S3 URLs to read the files.

3.Restricting Access to Files on Custom Origins

If you use a custom origin, you can optionally set up custom headers to restrict access. For CloudFront to get your files from a custom origin, the files must be publicly accessible. But by using custom headers, you can restrict access to your content so that users can access it only through CloudFront, not directly. This step isn't required to use signed URLs, but we recommend it.

Using Signed URLs

https://docs.aws.amazon.com/en_pv/AmazonCloudFront/latest/DeveloperGuide/private-content-signed-urls.html

1.In your CF distribution, specify one or more trusted signers, which are the AWS accounts that you want to have permission to create signed URLs.

2.You develop your application to determine whether a user should have access to your content and to create signed URLs for the files or parts of your application that you want to restrict access to.

3.A user requests a file for which you want to require signed URLs.

4.Your application verifies that the user is entitled to access the file: they've signed in, they've paid for access to the content, or they've met some other requirement for access.

5.Your application creates and returns a signed URL to the user.

6.The signed URL allows the user to download or stream the content.
This step is automatic; the user usually doesn't have to do anything additional to access the content.

7.CloudFront uses the public key to validate the signature and confirm that the URL hasn't been tampered with. If the signature is invalid, the request is rejected.
If the signature is valid, CloudFront looks at the policy statement in the URL (or constructs one if you're using a canned policy) to confirm that the request is still valid. For example, if you specified a beginning and ending date and time for the URL, CloudFront confirms that the user is trying to access your content during the time period that you want to allow access.
If the request meets the requirements in the policy statement, CloudFront does the standard operations:

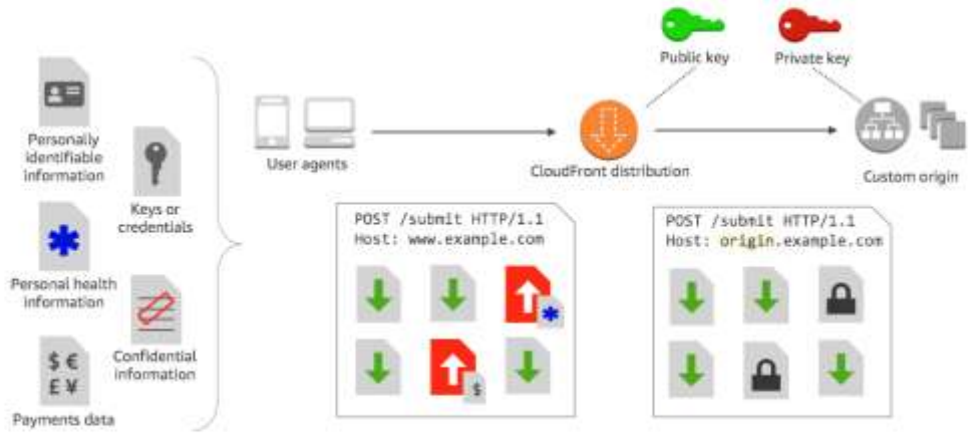
- determines whether the file is already in the edge cache,
- forwards the request to the origin if necessary,
- and returns the file to the user.

CloudFront - Using Field level encryption

https://docs.aws.amazon.com/AmazonCloudFront/latest/Develc

Field-level encryption allows you to securely upload user-submitted sensitive information to your web servers. The sensitive information provided by your clients is encrypted at the edge closer to the user and remains encrypted throughout your entire application stack, ensuring that only applications that need the data—and have the credentials to decrypt it—are able to do so.

To use field-level encryption, you configure your CloudFront distribution to specify the set of fields in POST requests that you want to be encrypted, and the public key to use to encrypt them. You can encrypt up to 10 data fields in a request. (You can't encrypt all of the data in a request with field-level encryption; you must specify individual fields to encrypt.)



1. **Get a public key-private key pair.** You must obtain and add the public key before you start setting up field-level encryption in CloudFront.
2. **Create a field-level encryption profile.** Field-level encryption profiles, which you create in CloudFront, define the fields that you want to be encrypted.
3. **Create a field-level encryption configuration.** A configuration specifies the profiles to use—based on the content type of the request or a query argument—for encrypting specific data fields. You can also choose the request-forwarding behavior options that you want for different scenarios; for example, when the profile name specified by the query argument in a request URL doesn't exist in CloudFront.
4. **Link to a cache behavior.** Link the configuration to a cache behavior for a distribution, to specify when CloudFront should encrypt data.

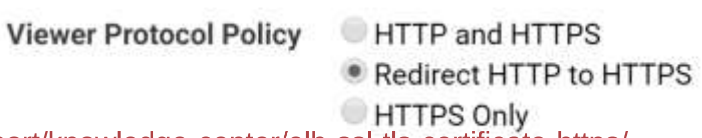
Decrypting Data Fields at Your Origin

CloudFront encrypts data fields by using the AWS Encryption SDK. The data remains encrypted throughout your application stack and can be accessed only by applications that have the credentials to decrypt it.

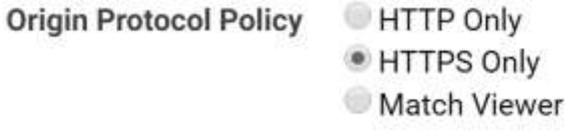
After encryption, the cipher text is base64 encoded. When your applications decrypt the text at the origin, they must first decode the cipher text, and then use the AWS Encryption SDK to decrypt the data.

CloudFront - Enabling SSL

https://aws.amazon.com/premiumsupport/knowledge-center/elb-ssl-tls-certificate-https/


Viewer Protocol Policy

HTTP and HTTPS [✓ Redirect HTTP to HTTPS] HTTPS Only


Origin Protocol Policy

HTTP Only [✓ HTTPS Only] Match Viewer

If you don't have a certificate issued for your domain name, you can

1. Request a public certificate using ACM OR
2. To use a third-party certificate with a load balancer, you can either import the certificate into ACM OR
3. upload a certificate to IAM certificate store

Note: ACM certificates can be used only with following services integrated with ACM:

- ELB
- CloudFront
- API Gateway
- ElasticBeanstalk

To use the ACM certificates with Amazon CloudFront, the certificates must be imported or requested in the US East (N. Virginia) Region.

Algorithm	ACM (Preferred)	IAM
1024-bit RSA (RSA_1024)	Yes	Yes
2048-bit RSA (RSA_2048)	Yes	Yes
RSA (up to 16384 bits)		Yes
Elliptic Curve (ECDSA)		Yes

Although ACM allows you to import certificates with a key algorithm of 4096-bit RSA and EC, these certificates can't be associated with load balancers through integration with ACM.

Classic Load Balancer doesn't support adding multiple certificates. To add multiple certificates for different domains to a load balancer, do one of the following:

- Use a SAN certificate to validate multiple domains behind the load balancer, including wildcard domains, with ACM
- Use an ALB, which supports multiple SSL certificates and smart certificate selection using Server Name Indication (SNI).

ACM Certificate Characteristics:

Domain Validation: does not provide extended validation (EV) certificates or organization validation (OV) certificates.
Validity period-13 months
Managed Renewal and Deployment
Browser and Application Trust
Multiple Domain Names
Wildcard Names
Algorithms

- 2048-bit RSA (RSA_2048)
- 4096-bit RSA (RSA_4096)
- Elliptic Prime Curve 256 bit (EC_prime256v1)
- Elliptic Prime Curve 384 bit (EC_secp384r1)

You cannot use ACM certificates for email encryption. You cannot download the private key for an ACM Certificate.

Why to use CloudFront for dynamic content?

https://blog.shikisoft.com/serving-dynamic-website-with-amazon-cloudfront/

- Your clients will connect to your applications through an AWS edge location closest to them. Then, this connection will utilize AWS network infrastructure which is expected to be more stable and provide faster access to your servers on an AWS region, even if the content was not cached before.
- Although we will decrease the time to live (TTL) or cache expiration to 0 as I will describe below; this will not mean that CloudFront will not cache your content. Actually, it will. The only difference will be that it will send the last caching time of the content alongside with the request to your origin. This will allow your origin to respond faster if the content was not changed and edge location will serve the content from its cache. When you compare the small packet size in this process with retrieving the full content from your origin, this will decrease the load on your servers and delivery time of your content to your clients.
- The most obvious benefit will be the decrease in SSL handshaking time and time to first byte metrics. SSL handshaking is done on edge locations and time to first byte improvement is a result of the enhanced network speed.
- Amazon CloudFront only accepts well-formed connections and reduces the number of requests and TCP connections back to your web application. These will help you to prevent many of the common DDoS attacks such as SYN floods and UDP reflection attacks; because they will not reach to your origin. Also geographically distributed architecture will help you to isolate these attacks in a close location to them, allowing you to continue to serve your application without any impact on other locations.
- You can also enhance your web application's security by integrating AWS Web Application Firewall (WAF) with your CloudFront distribution and make use of its features. By creating AWS WAF Web Access Control Lists (ACL), you can filter and block requests based on request signatures, URI, query string matches; as well as IP addresses when the number of their requests matching a rule exceeds a treshold you define. Of course, you will need a well-architected monitoring mechanism such as Amazon CloudWatch in this process.

Customizing Content at the Edge with Lambda@Edge

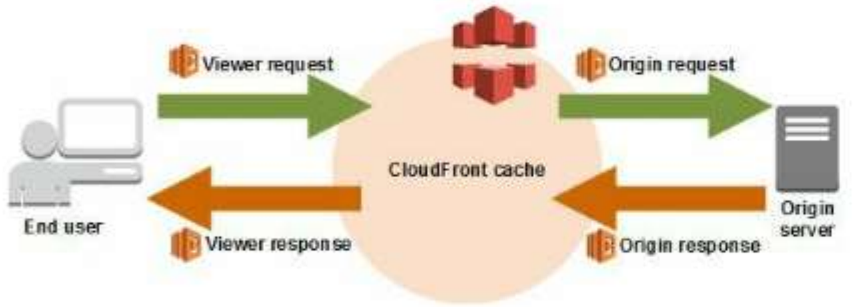
Lambda@Edge lets you execute functions that customize the content that CloudFront delivers. You can author functions in one Region, US-East-1 (N. Virginia), and then execute them in AWS locations globally that are closer to the viewer, without provisioning or managing servers. Lambda@Edge scales automatically. Processing requests at AWS locations closer to the viewer instead of on origin servers significantly reduces latency and improves the user experience.

There are many uses for Lambda@Edge processing. For example:

1. A/B testing - A Lambda can inspect cookies and rewrite URLs so that users see different versions of a site
2. CF can return different images based on the device they're using by checking the User-Agent header or check Referer header and cause CF to return the images to bots that have the lowest available resolution.
3. Check cookies for other criteria.
4. Generate HTTP responses (e.g. a gzipped static page) when CF viewer request or origin request events occur.
5. A function can inspect headers or authorization tokens, and insert a header to control access to your content before CloudFront forwards the request to your origin OR redirect unauth. users to login page
6. Personalize Content by Country or Device Type.
    1. Example: Redirecting Viewers to a Country-Specific URL
7. Content-Based Dynamic Origin Selection. Examples:
    1. Using an Origin-Request Trigger to Change From a Custom Origin to an S3 Origin or vice versa
    2. OR Using an Origin-Request Trigger to Change the S3 Origin Region
    3. OR Using an Origin-Request Trigger to Gradually Transfer Traffic From One S3 Bucket to Another
    4. OR Using an Origin-Request Trigger to Change the Origin Domain Name Based on the Country Header
8. A function can make network calls to ext. resources to confirm user credentials, or fetch additional content to customize a response.

Customizing Content at the Edge with Lambda@Edge

Lambda@Edge lets you use following CloudFront triggers to invoke a Lambda function.



How to Decide Which CloudFront Event to Use to Trigger a Lambda Function:

- If you want to add, remove, or change headers for objects that are returned by the origin and you want CloudFront to cache the result, use the origin response event.
- If you want the function to execute for every request that CloudFront receives for the distribution, use the viewer request or viewer response events.
- If you want the function to change a value that you're using as a basis for caching, use the viewer request event.
  - URL in the viewer request – ***http://example.com/en/index.html***
  - URL when the request comes from an IP address in Germany – ***http://example.com/de/index.html***
  - Also use the viewer request event if you're caching based on cookies or request headers.
- If you want the function to change the request in a way that affects the response from the origin, use the origin request event.

Customizing Content at the Edge with Lambda@Edge

Other use cases for using lambda@edge

Generating HTTP Responses in Request Triggers:

- When CF receives a request, you can use a Lambda to generate an HTTP response that CF *returns directly to the viewer without forwarding the response* to the origin. Some common scenarios:
  - Returning a small web page to the viewer
  - Returning an HTTP 301 or 302 status code to redirect the user to another web page
  - Returning an HTTP 401 status code to the viewer when the user hasn't authenticated

Updating HTTP Responses in Origin-Response Triggers:

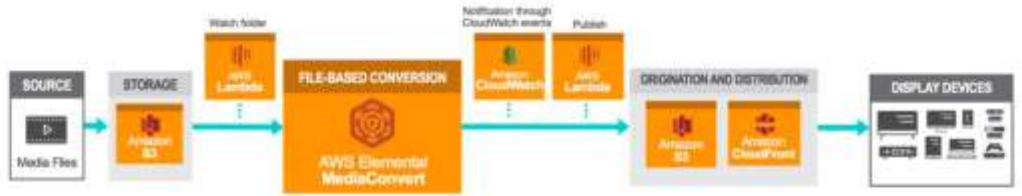
- When CF receives an HTTP response from the origin server, if there is an origin-response trigger associated with the cache behavior, you can modify the HTTP response to override what was returned from the origin. Some common scenarios:
  - Changing the status to set an HTTP 200 status code and creating static body content to return to the viewer when an origin returns an error status code (4xx or 5xx).
  - Changing the status to set an HTTP 301 or HTTP 302 status code, to redirect the user to another web site when an origin returns an error status code (4xx or 5xx).

Accessing the Request Body by Choosing the Include Body Option:

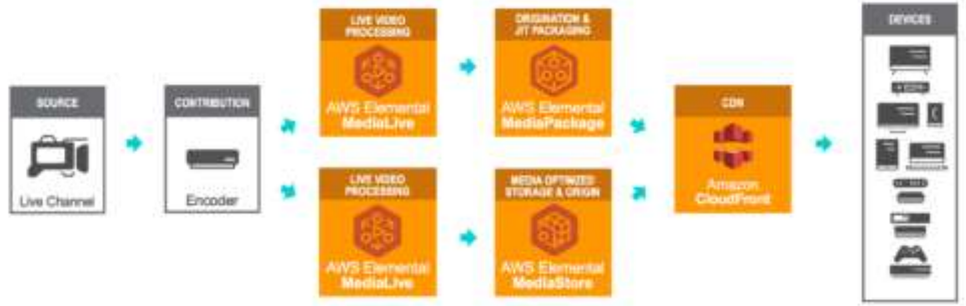
- You can opt to have Lambda@Edge expose the body in a request for writable HTTP methods (POST, PUT, DELETE, and so on), so that you can access it in your Lambda function. You can choose read-only access, or you can specify that you'll replace the body. To enable this option, choose **Include Body** when you create a CF trigger for a viewer request or origin request event. Scenarios:
  - Processing web forms, like "contact us" forms, without sending customer input data back to origin servers.
  - Gathering web beacon data that's sent by viewer browsers and processing it at the edge.

On-demand & Live streaming

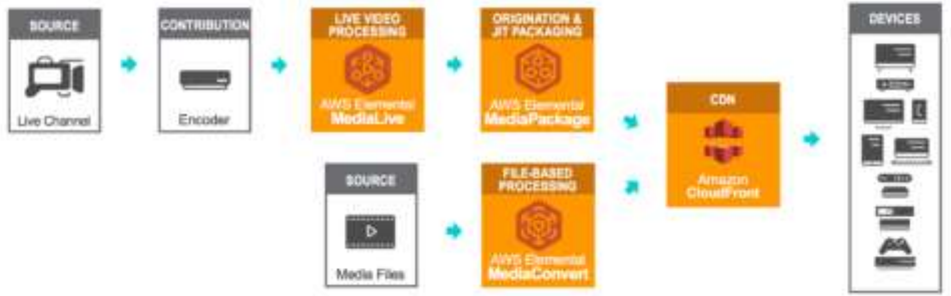
with AWS Elemental Media Convert, Media



Workflow Example: Content Distribution for VOD Services



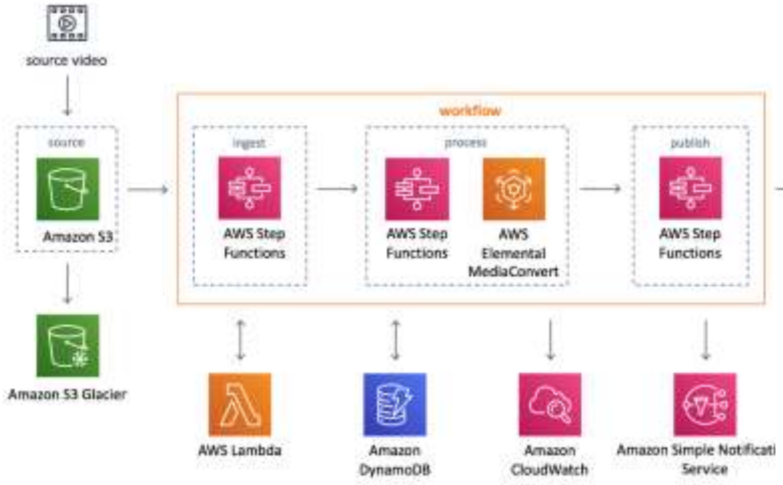
Workflow Example: Large-scale Live Events

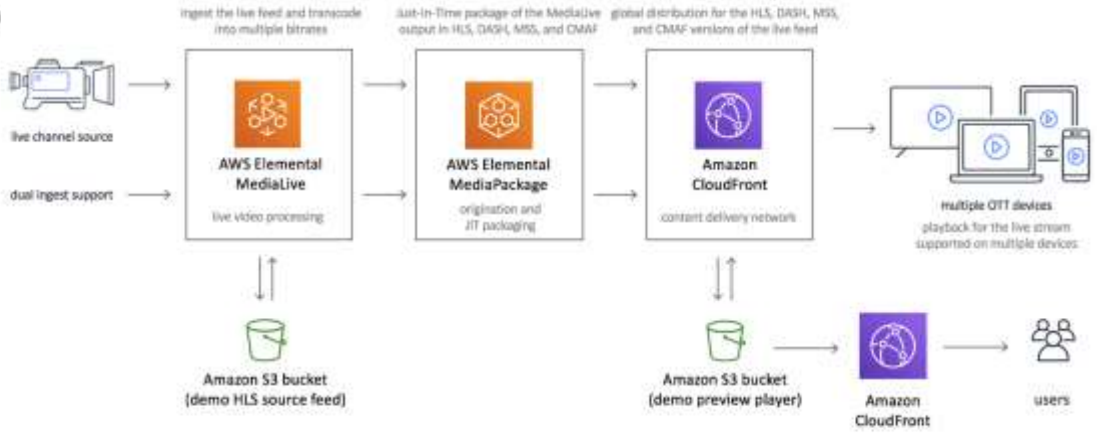


Workflow Example: 24x7 Live Channel Delivery

AWS Solutions - On-demand & Live streaming

https://aws.amazon.com/solutions/video-on-demand-on-aws/
https://aws.amazon.com/solutions/live-streaming-on-aws/




ingest the live feed and transcode into multiple bitrates
Just-In-Time package of the MediaLive output in HLS, DASH, MSS, and CMAF
global distribution for the HLS, DASH, MSS, and CMAF versions of the live feed
live channel source
dual ingest support
AWS Elemental MediaLive
live video processing
AWS Elemental MediaPackage
origination and JIT packaging
Amazon CloudFront
content delivery network
multiple OTT devices
playback for the live stream supported on multiple devices
Amazon S3 bucket (demo HLS source feed)
Amazon S3 bucket (demo preview player)
Amazon CloudFront
users