AWS

**CloudFront**

This is Cached the data in edge locations and also separate from region/AZ

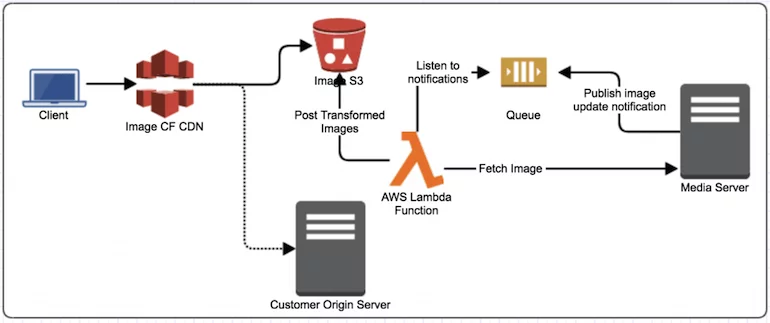
**Caching Content with Amazon Cloud Front with Lambda**

Amazon CloudFront speeds up the delivery of your content through Edge caches. When user accesses your website or application and requests content, the request is routed to the nearest CloudFront edge location. Only for first user there will be latency in retrieving the content and all subsequent users of the same content will be able to retrieve the content quickly as the content will be cached in edge location.

Following is the process that happens on user’s request for content:

* CloudFront checks its cache for the requested object. If the requested object is found in the cache, it’ll be returned.
* If the requested object is not found in the CloudFront cache,
  + The request will be redirected to the configured *Origin Server*. Origin Server could be S3 Bucket, Custom HTTTP Server or any other server.
  + CloudFront caches the object returned from Origin Server in the nearest edge location and returns to the user.

Objects in CloudFront can be cached for a configured TTL (Time-To-Live), and once the TTL is expired the object will no longer be available to serve from the cache as shown in Figure



**S3- Simple Object Storage**:

**Lambda**

**Cloudformation**

**API Gateway**

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the "front door" for applications to access data, business logic, or functionality from your backend services. Using API Gateway, you can create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications. API Gateway supports containerized and serverless workloads, as well as web applications.

API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, CORS support, enable the cache,authorization and access control, throttling, monitoring, and API version management. API Gateway has no minimum fees or startup costs. You pay for the API calls you receive and the amount of data transferred out and, with the API Gateway tiered pricing model, you can reduce your cost as your API usage scales.

## API Types

### **RESTful APIs**

Build RESTful APIs optimized for serverless workloads and HTTP backends using HTTP APIs. [HTTP APIs](https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api.html) are the best choice for building APIs that only require API proxy functionality. If your APIs require API proxy functionality and API management features in a single solution, API Gateway also offers [REST APIs](https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html).

### **WEBSOCKET APIs**

Build real-time two-way communication applications, such as chat apps and streaming dashboards, with [WebSocket APIs](https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-websocket-api-overview.html). API Gateway maintains a persistent connection to handle message transfer between your backend service and your clients.

## Amazon API Gateway: A Single Entry-Point (Micorservices)

Using a single API Gateway in the architecture across multiple web portal applications and Micorservices is an important consideration towards the goal of reusability of components and cost optimization. **Amazon API Gateway** provides a highly scalable solution to create and publish RESTful and WebSocket APIs. It provides flexibility in choosing multiple backend technologies such as[AWS Lambda](https://aws.amazon.com/lambda/) functions, [AWS Step Functions](https://aws.amazon.com/step-functions/) state machines, or call HTTP(s) endpoints hosted on [AWS Elastic Beanstalk](https://aws.amazon.com/elasticbeanstalk/), [Amazon EC2](https://aws.amazon.com/ec2/), and also non-AWS hosted HTTP based services.

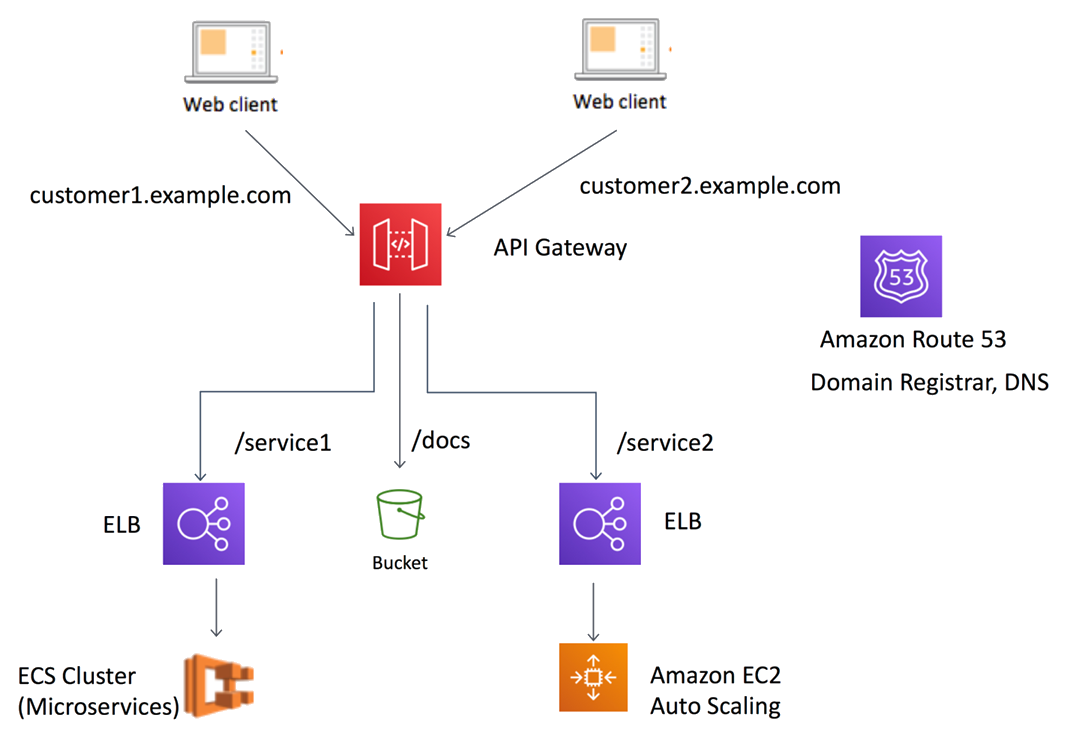
API Gateway allows for handling common API management tasks such as security, caching, throttling, and monitoring. While its primary objective is to provide that abstraction layer on top of your backend APIs and Micorservices25, it can also allow backends to be simple web applications for web portal access or [Amazon S3](https://aws.amazon.com/s3/) buckets for providing access to static web content or documents.

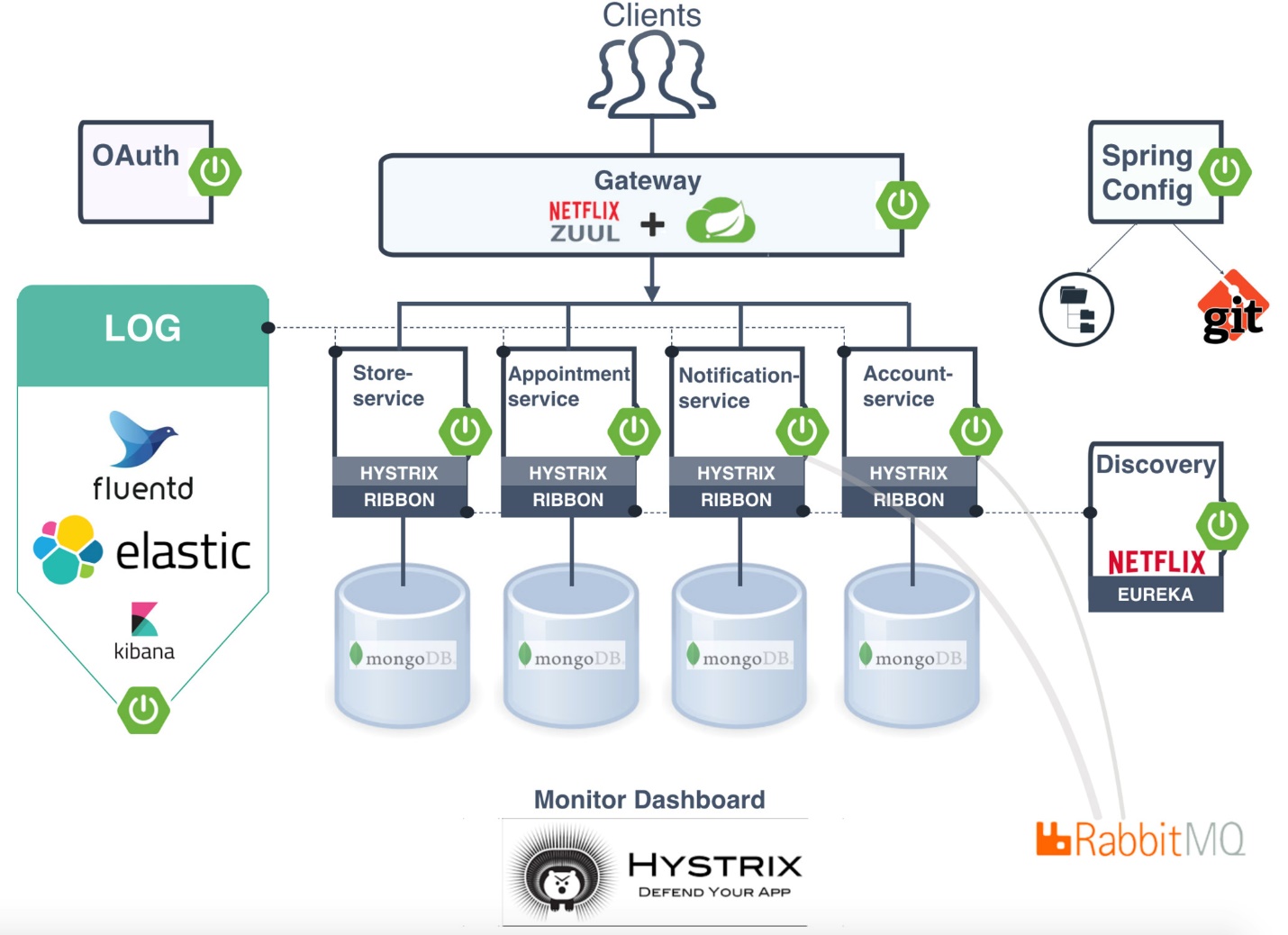
Along with above capabilities, the following key features of API Gateway help to create the architecture described here.

1. **Custom Domain Names support:**  
   When an API is deployed using API Gateway, the default API endpoint domain name is not user friendly as can be seen here:https://api-id.execute-api.region.amazonaws.com/stageapi-id is generated by API Gateway; region is specified by you when creating the API; and stage is specified by you when deploying the API.The default API endpoint can be difficult to recall and not user-friendly. To provide a simpler and more intuitive URL for your API users, it allows you to specify a custom domain name such as customer1.example.com via its integration with AWS Certificate Manager, which allows for SSL certificate-based validation of the sub-domains. API Gateway allows you to map multiple sub-domains to a single API endpoint allowing you to white-label the domains based on an external customer’s requirement.
2. **API request /response transformation:**  
   API Gateway allows you to specify the integration of each path of the API endpoint separately. This allows you to route API requests for each path to a separate backend endpoint and at the same time apply any request/response transformations, such as customer header insertion or modification of existing headers to manage any custom handling of APIs.

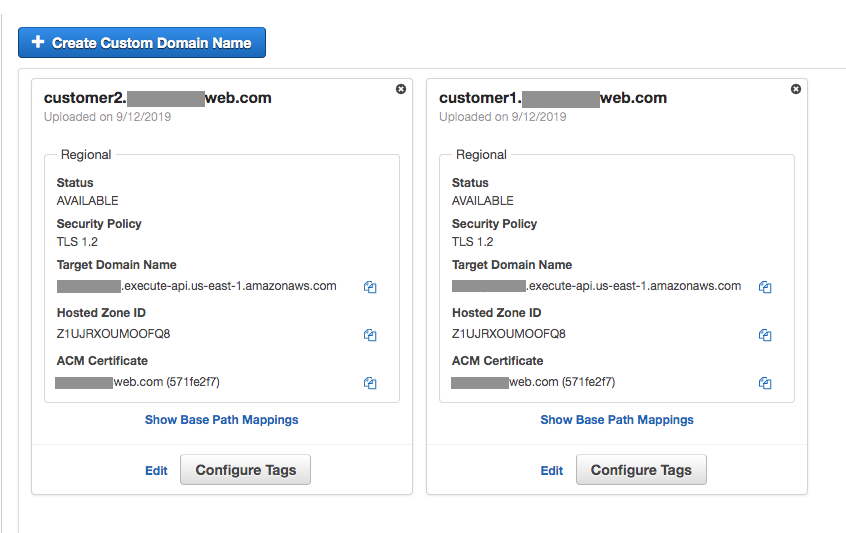
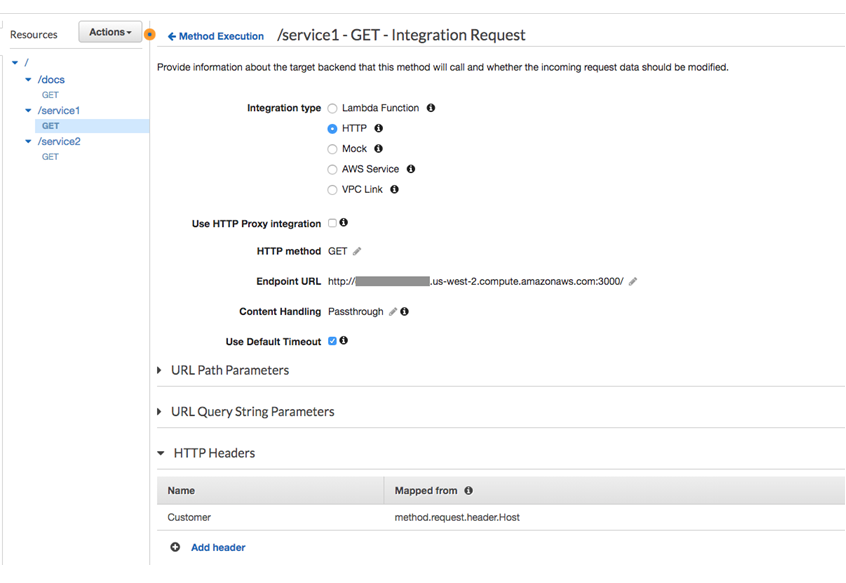
## Architecture and Its Benefits

In the architecture shown in the diagram below, the features explained in this blog are utilized.





**Event driven Micorservices architecture.**Event-driven architecture is a methodology used to produce, handle events and implement applications where events transmit among decoupled software components and services. This architecture is an example of a typical SaaS provider who wants to offer its services to other enterprises and needs to support white-labeling domains for this web and API infrastructure. This is achieved using the following steps:

* 1. A single domain of example.com can be registered with a domain registrar and you can create subdomains by creating CNAME records for example customer1.example.com, customer2.example.com by updating DNS information with the domain registrar. This can be handled by AWS’s own DNS and Registrar service [Amazon Route 53](https://aws.amazon.com/route53/) or can be any third party domain name provider.
  2. Once complete, AWS provides [AWS Certificate Manager](https://aws.amazon.com/certificate-manager/) (ACM) to create a certificate for the following domains: example.com and \*.example.com. This makes sure that the ACM certificate once applied to the API Gateway can allow for multiple subdomains to be served by it.
  3. Using the certificate created in ACM, you can create custom domain for the API endpoint. In this example this API endpoint will serve two subdomains for two different external customers and specifying base path mappings as needed. The following two subdomains are created as custom domains using this capability: customer1.example.com and customer2.example.com.  
     **Note**: Make sure to add CNAME records for customer1 and customer2 at your DNS provider to point to the target domain name created within your API Gateway for each of the two customer sub-domains.  
     
  4. The API Endpoint is then configured with the following API resources:
     1. HTTP integration of /service1 to route traffic to the ELB endpoint of microservice hosted on an ECS cluster
     2. HTTP integration of /service2 to route traffic to the ELB endpoint of web application hosted on an EC2 cluster
     3. /docs API resource is integrated with AWS S3 for any static documents  
        
  5. API Gateway allows you to capture the FQDN of the URL and map it to Custom Headers or Query String Parameters which are then sent to the backend service integrated with the corresponding API resource and the HTTP method. For example we can create a custom header called “Customer” to forward customer1 or customer2 to the backend application for customer-specific business logic. This is done using the Method Request parameters and Integration Request configuration within API Gateway.

<https://aws.amazon.com/blogs/architecture/using-api-gateway-as-a-single-entry-point-for-web-applications-and-api-microservices/>

**AWS SDK/CLI**

