**Amazon Web services :**

**1) For Static Content:**

**Amazon S3(Amazon Simple Storage Service)** is designed for large-capacity, low-cost file storage in one specific geographical region.\* The storage and bandwidth costs are quite low.

**Amazon CloudFront** is a Content Delivery Network (CDN) which proxies and caches web data at edge locations as close to users as possible.

When end users request an object using this domain name, they are automatically routed to the nearest edge location for high performance delivery of your content. ([Amazon](http://aws.amazon.com/cloudfront/))

The data served by CloudFront may or may not come from S3. Since it is more optimized for delivery speed, the bandwidth costs a little more.

If your user base is localized, you won't see too much difference working with S3 or CloudFront (but you have to choose the right location for your S3 bucket: US, EU, APAC). If your user base is spread globally and speed is important, CloudFront may be a better optio

**What Is Amazon Route 53?**

You can use Amazon Route 53 to help you get a website or web application up and running. Amazon Route 53 performs three main functions:

* **Register domain names** – Your website needs a name, such as example.com. Amazon Route 53 lets you register a name for your website or web application, known as a *domain name*. For an overview, see [How Domain Registration Works](http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/welcome-domain-registration.html).
* **Route internet traffic to the resources for your domain** – When a user opens a web browser and enters your domain name in the address bar, Amazon Route 53 helps the Domain Name System (DNS) connect the browser with your website or web application. For an overview, see [How Internet Traffic Is Routed to Your Website or Web Application](http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/welcome-dns-service.html).
* **Check the health of your resources** – Amazon Route 53 sends automated requests over the internet to a resource, such as a web server, to verify that it's reachable, available, and functional. You also can choose to receive notifications when a resource becomes unavailable and choose to route internet traffic away from unhealthy resources. For an overview, see [How Amazon Route 53 Checks the Health of Your Resources](http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/welcome-health-checks.html).

You can use any combination of these functions. For example, you can use Amazon Route 53 both to register your domain name and to route internet traffic for the domain, or you can use Amazon Route 53 to route internet traffic for a domain that you registered with another domain registrar. If you choose to use Amazon Route 53 for all three functions, you register your domain name, then configure Amazon Route 53 to route internet traffic for your domain, and finally configure Amazon Route 53 to check the health of your resources.

# What Is AWS Elastic Beanstalk?

Amazon Web Services (AWS) comprises dozens of services, each of which exposes an area of functionality. While the variety of services offers flexibility for how you want to manage your AWS infrastructure, it can be challenging to figure out which services to use and how to provision them.

With Elastic Beanstalk, you can quickly deploy and manage applications in the AWS Cloud without worrying about the infrastructure that runs those applications. AWS Elastic Beanstalk reduces management complexity without restricting choice or control. You simply upload your application, and **Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.** Elastic Beanstalk uses highly reliable and scalable services that are available in the [AWS Free Usage Tier](http://aws.amazon.com/free).

Elastic Beanstalk supports applications developed in Java, PHP, .NET, Node.js, Python, and Ruby, as well as different container types for each language. A container defines the infrastructure and software stack to be used for a given environment. When you deploy your application, **Elastic Beanstalk provisions one or more AWS resources, such as Amazon EC2 instance**s. The software stack that runs on your Amazon EC2 instances depends on the container type. For example, Elastic Beanstalk supports two container types for Node.js: a 32-bit Amazon Linux image and a 64-bit Amazon Linux image. Each runs a software stack tailored to hosting a Node.js application. You can interact with Elastic Beanstalk by using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or eb, a high-level CLI designed specifically for Elastic Beanstalk.

To learn more about the AWS Free Usage Tier and how to deploy a sample web application in it using AWS Elastic Beanstalk, go to [Getting Started with AWS: Deploying a Web Application](http://docs.aws.amazon.com/gettingstarted/latest/deploy/welcome.html).

You can also perform most deployment tasks, such as changing the size of your fleet of Amazon EC2 instances or monitoring your application, directly from the Elastic Beanstalk web interface (console).

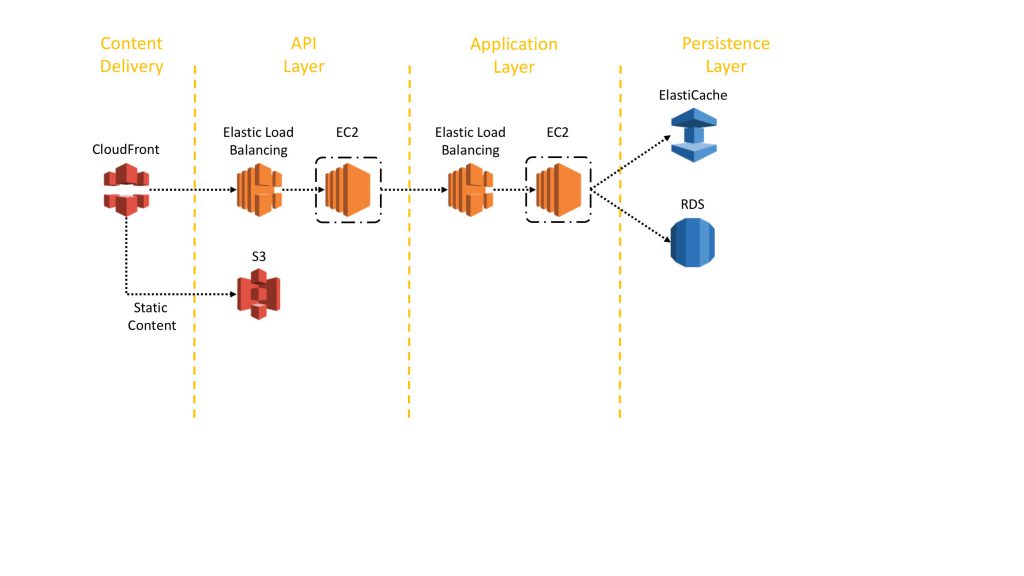
To use Elastic Beanstalk, you create an application, upload an application version in the form of an application source bundle (for example, a Java .war file) to Elastic Beanstalk, and then provide some information about the application. Elastic Beanstalk automatically launches an environment and creates and configures the AWS resources needed to run your code. After your environment is launched, you can then manage your environment and deploy new application versions. The following diagram illustrates the workflow of Elastic Beanstalk.


        AWS Elastic Beanstalk Flow
      

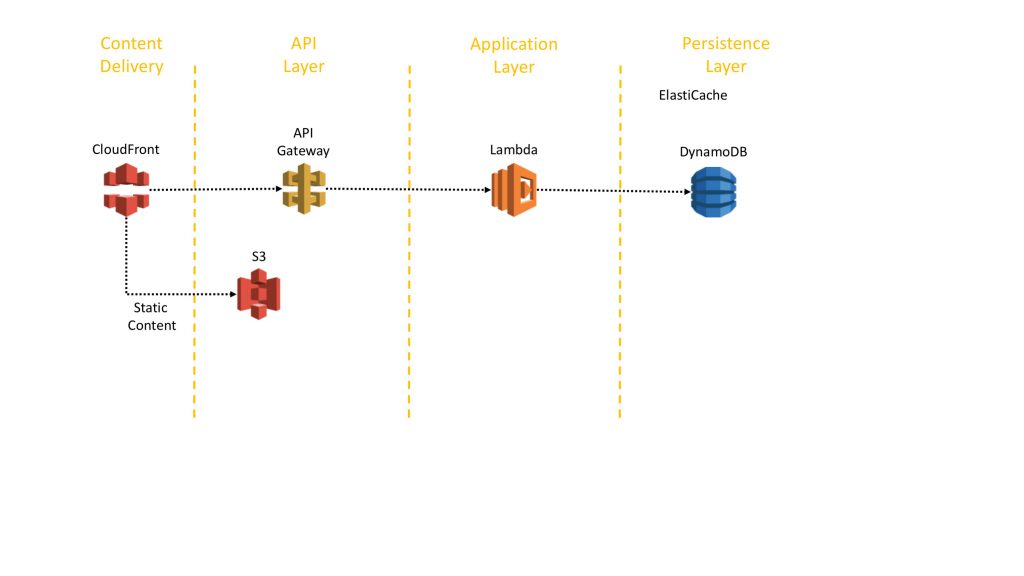
**RDS**

Amazon Relational Database Service (RDS [https://**aws**.**amazon**.com/**rds**/](https://aws.amazon.com/rds/)) will be used to host our database. We will create an RDS database and our web services will be used to connect to the database

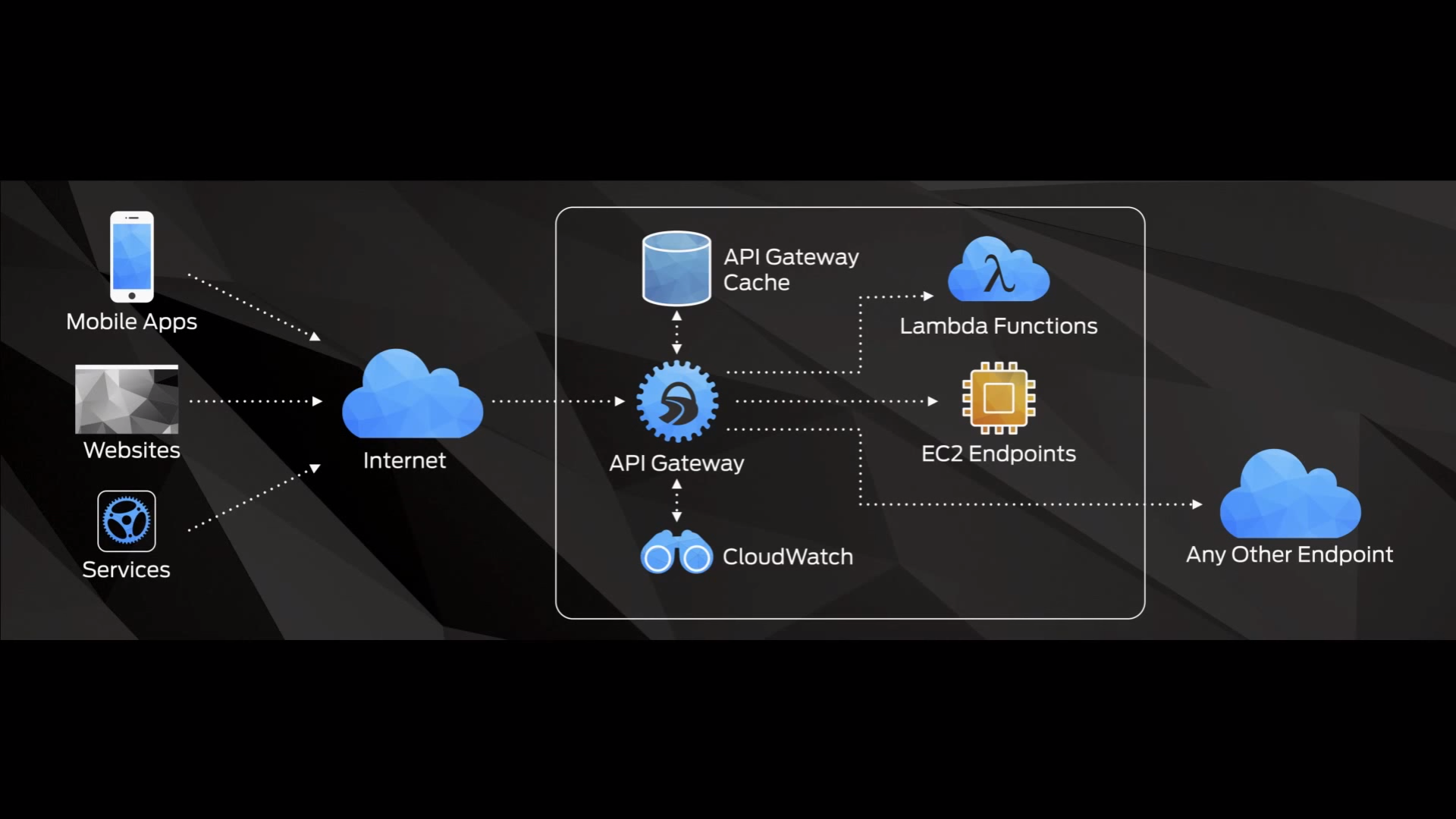
## Simple Microservice architecture on AWS

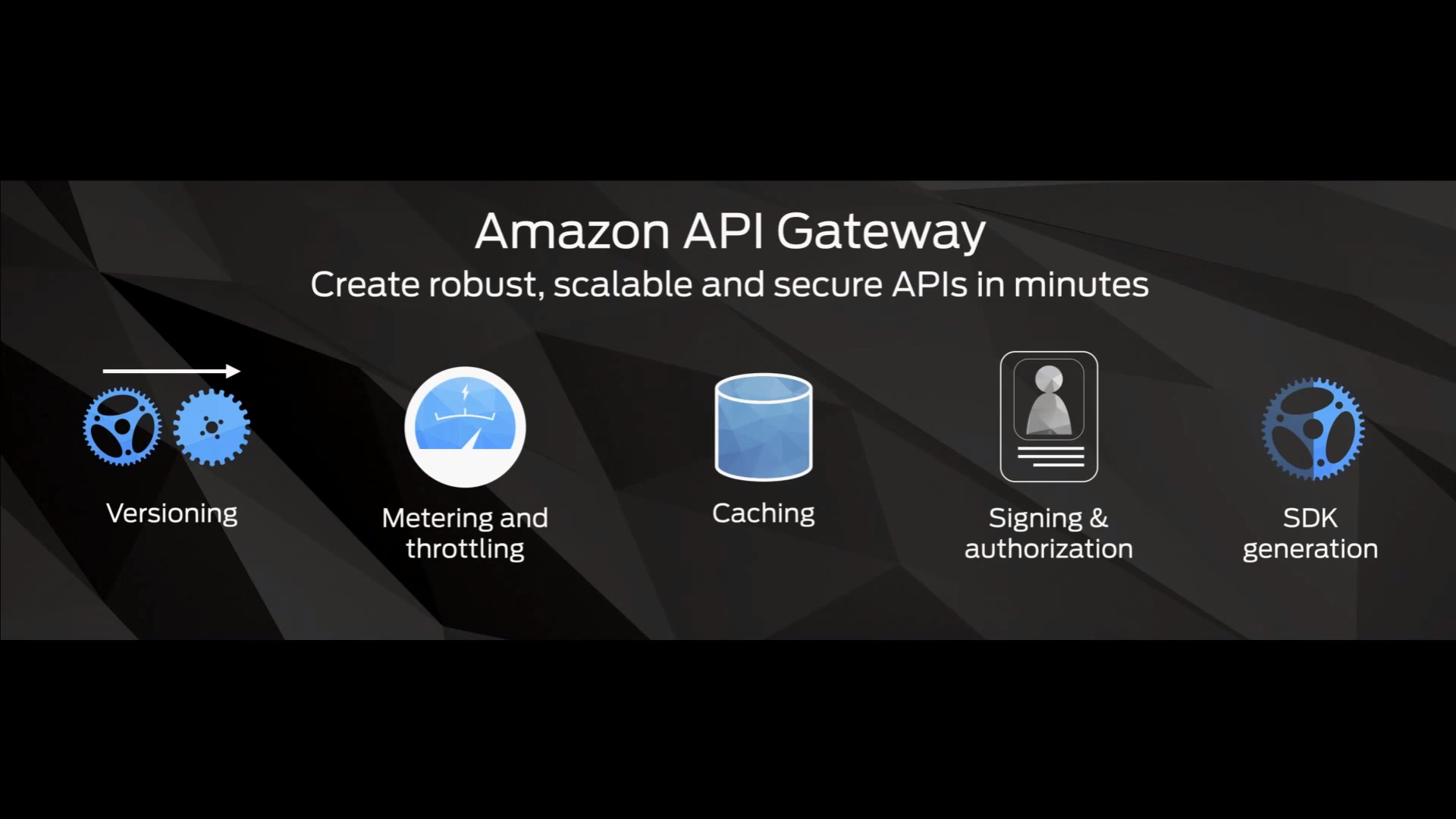


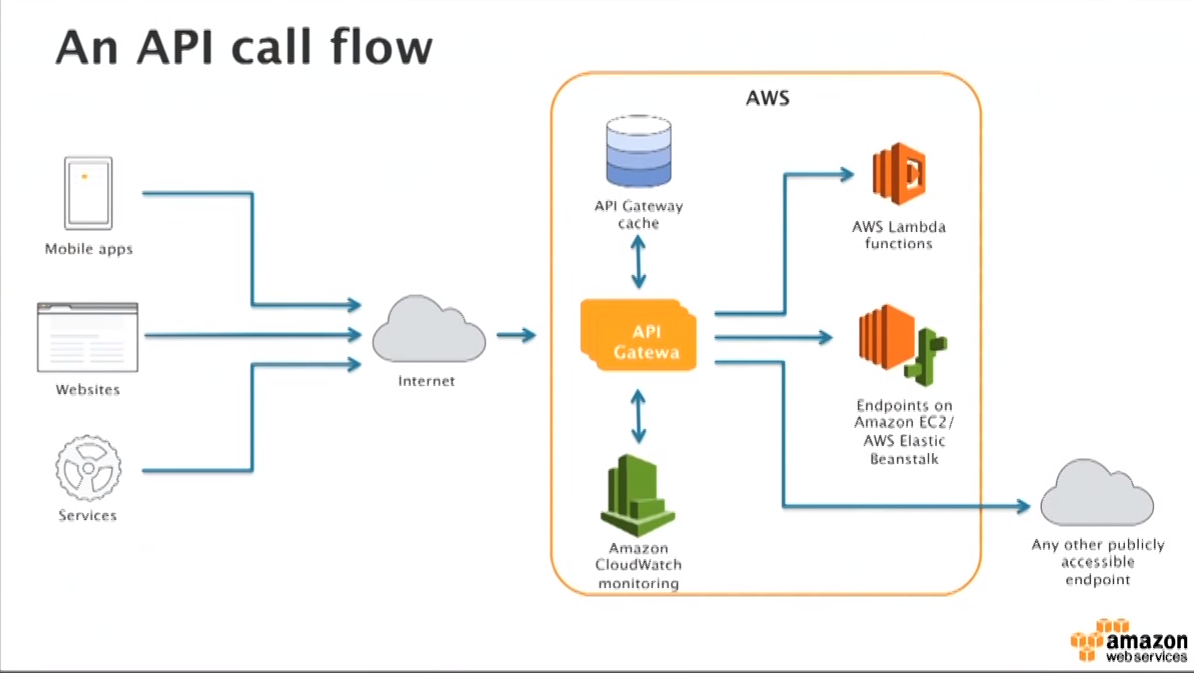
Relational databases are still very popular to store structured data and business objects. AWS offers six database engines (Microsoft SQL Server, Oracle, MySQL, MariaDB, PostgreSQL, and Amazon Aurora) as managed services.



Amazon API Gateway addresses those challenges and reduces the operational complexity of the API Layer. Amazon API Gateway allows customers to create their APIs programmatically, by importing Swagger definitions, or with a few clicks in the AWS Management Console. API Gateway serves as a front door to any web-application running on Amazon EC2, on Amazon ECS, on AWS Lambda, or on any on-premises environment. In a nutshell: it allows running APIs without managing servers. Figure 2 visualizes how API Gateway handles API calls and interacts with other components. Requests from mobile devices, websites, or other backend services are routed to the closest Amazon CloudFront PoP to minimize latency and provide the optimum user experience. API Gateway first checks if the request is in the cache and – if no cached records available – then forwards it to the backend for processing. Once the backend has processed the request, API call metrics are logged in Amazon CloudWatch and content is returned to the client







**What is Lambda?**

“AWS Lambda lets you run code without provisioning or managing servers,” AWS states on the Lambda product page. You can think of Lambda as an event-driven computing platform; Lambda runs when triggered by an event and executes code that’s been loaded into the system.

For example, a simple use case would be that every time an image is uploaded into Amazon Simple Storage Service (S3), a Lambda function could automatically resize the image. The Seattle Times uses this to automatically resize images for mobile, tablet and desktop devices. The event that triggers the Lambda function is the file being uploaded to S3. Lambda then executes the function of resizing the image