CATEGORY DEEP DIVE

Serverless

## Introduction

Beginner | 10 minutes

The Serverless - Deep Dive introduces fundamental concepts, reference architectures, best practices, and hands-on activities to help you get started building serverless applications. This is the ideal place to get started if you’re new to serverless. For seasoned serverless builders, we have resources and links to more advanced topics.

### **What is serverless?**

Serverless allows you to build and run applications and services without thinking about servers. It eliminates infrastructure management tasks such as server or cluster provisioning, patching, operating system maintenance, and capacity provisioning. You can build them for [nearly any type of application](https://aws.amazon.com/serverless/?p=gsrc&c=dd_s&e=gs2020&p=deepdiveserverless/#Serverless_application_use_cases) or backend service, and everything required to run and scale your application with high availability is handled for you.  
  
Serverless applications are event-driven and loosely coupled via technology-agnostic APIs or messaging. Event-driven code is executed in response to an event, such as a change in state or an endpoint request. Event-driven architectures decouple code from state. Integration between loosely couple components is usually done asynchronously, with messaging.  
  
AWS Lambda is a serverless compute service that is well suited to event-driven architectures. Lambda functions are triggered by events via integrated event sources such as Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS), and Amazon Kinesis that can be used to create asynchronous integrations. Lambda functions consume and produce events that other services can then consume.  
  
Serverless architecture patterns use Lambda with other managed services that are also serverless. In addition to message and streaming services, serverless architectures use managed services such as Amazon API Gateway for API management, Amazon DynamoDB for data stores, and AWS Step Functions for orchestration. The serverless platform also includes a set of developer tools including the Serverless Application Model, or SAM, which help simplify deployment and testing of your Lambda functions and serverless applications.

### **Why use serverless?**

No server management: There is no need to provision or maintain any servers. There is no software or runtime to install, maintain, or administer.  
  
Flexible scaling: Your application can be scaled automatically or by adjusting its capacity through toggling the units of consumption (e.g. throughput, memory) rather than units of individual servers.  
  
Pay for value: Pay for consistent throughput or execution duration rather than by server unit.  
  
Automated high availability: Serverless provides built-in availability and fault tolerance. You don't need to architect for these capabilities since the services running the application provide them by default.

### **Core serverless services**

Serverless applications are generally built using fully managed services as building blocks across the compute, data, messaging and integration, streaming, and user management and identity layers. Services such as AWS Lambda, API Gateway, SQS, SNS, EventBridge or Step Functions are at the core of most applications, supported by services such as DynamoDB, S3 or Kinesis.

|  |  |  |
| --- | --- | --- |
| **Category** | **Service** | **Description** |
| Compute | AWS Lambda | AWS Lambda lets you run stateless serverless applications on a managed platform that supports microservices architectures, deployment, and management of execution at the function layer. |
| API Proxy | API Gateway | [Amazon API Gateway](https://aws.amazon.com/api-gateway/?e=gs2020&p=deepdiveserverless) is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. It offers a comprehensive platform for [API management](https://aws.amazon.com/api-gateway/api-management/?e=gs2020&p=deepdiveserverless). API Gateway allows you to process hundreds of thousands of concurrent API calls and handles traffic management, authorization and access control, monitoring, and API version management. |
| Messaging & Integration | SNS | [Amazon SNS](https://aws.amazon.com/sns/?e=gs2020&p=deepdiveserverless) is a fully managed pub/sub messaging service that makes it easy to decouple and scale microservices, distributed systems, and serverless applications. |
| SQS | [Amazon SQS](https://aws.amazon.com/sqs/?e=gs2020&p=deepdiveserverless) is a fully managed message queuing service that makes it easy to decouple and scale microservices, distributed systems, and serverless applications. |
| EventBridge | [Amazon EventBridge](https://aws.amazon.com/eventbridge/?e=gs2020&p=deepdiveserverless) is a serverless event bus that makes it easy to connect applications together using data from your own applications, integrated Software-as-a-Service (SaaS) applications, and AWS services. |
| Orchestration | Step Functions | [AWS Step Functions](https://aws.amazon.com/step-functions/?e=gs2020&p=deepdiveserverless) makes it easy to coordinate the components of distributed applications and microservices using visual workflows. |

### **Let's build!**

Below are a couple of resources to help introduce you to our core serverless services.

Run a serverless "Hello, World!"

Create a Hello World Lambda function using the AWS Lambda console and learn the basics of running code without provisioning or managing servers.

[Begin tutorial >>](https://aws.amazon.com/getting-started/hands-on/run-serverless-code/?e=gs2020&p=deepdiveserverless&p=gsrc&c=dd_s)

Create thumbnails from uploaded images

Create a Lambda function invoked by Amazon S3 every time an image file is uploaded into an S3 bucket and automatically create a thumbnail of that image.

[Begin tutorial >>](https://docs.aws.amazon.com/lambda/latest/dg/with-s3-tutorial.html?p=gsrc&c=dd_s)

Building Your First Application with AWS Lambda

Create a simple microservice

Use the Lambda console to create a Lambda function and an Amazon API Gateway endpoint to trigger that function.

[Begin tutorial >>](https://docs.aws.amazon.com/lambda/latest/dg/services-apigateway-blueprint.html)

Create a serverless workflow

Learn how to use AWS Step Functions to design and run a serverless workflow that coordinates multiple AWS Lambda functions.

[Begin tutorial >>](https://aws.amazon.com/getting-started/hands-on/create-a-serverless-workflow-step-functions-lambda/?e=gs2020&p=deepdiveserverless&p=gsrc&c=dd_s)

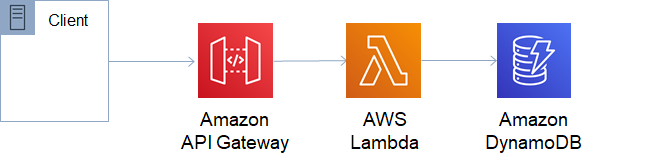
## Fundamentals

Intermediate | 20 minutes

In this section you will learn about event-driven design, the core principle behind scalable serverless applications.

Event-driven design

An event-driven architecture uses events to trigger and communicate between decoupled services. An event is a change in state, or an update, like an item being placed in a shopping cart on an e-commerce website. Events can either carry the state (e.g., the item purchased, its price, and a delivery address) or events can be identifiers (e.g., a notification that an order was shipped).  
  
Event-driven architectures have three key components: event producers, event routers, and event consumers. A producer publishes an event to the router, which filters and pushes the events to consumers. Producer services and consumer services are decoupled, which allows them to be scaled, updated, and deployed independently.  
  
To understand why an event-driven architecture is desirable, let’s look at a synchronous API call.



Customers leverage your microservices by making HTTP API calls. Amazon API Gateway hosts RESTful HTTP requests and responses to customers. AWS Lambda contains the business logic to process incoming API calls and leverage DynamoDB as a persistent storage.  
  
When invoked synchronously, API Gateway expects an immediate response and has a 30 second timeout. With synchronous event sources, if the response from Lambda requires more than 30 seconds, you are responsible for writing any retry and error handling code. Because of this, any errors or scaling issues that occur with any component downstream from the client, such as Read/Write capacity units in DynamoDB, will be pushed back to the client for front-end code to handle. By using asynchronous patterns and decoupling these components, you can build a more robust, highly scalable system.

Send fanout event notifications

Learn to implement a fanout messaging scenario where messages are "pushed" to multiple subscribers, eliminating the need to periodically check or poll for updates and enabling parallel asynchronous processing of the message by the subscribers.

[Begin tutorial >>](https://aws.amazon.com/getting-started/hands-on/send-fanout-event-notifications/?e=gs2020&p=deepdiveserverless)

Integrate Amazon EventBridge into your serverless applications

Learn how to build an event producer and consumer in AWS Lambda, and create a rule to route events.

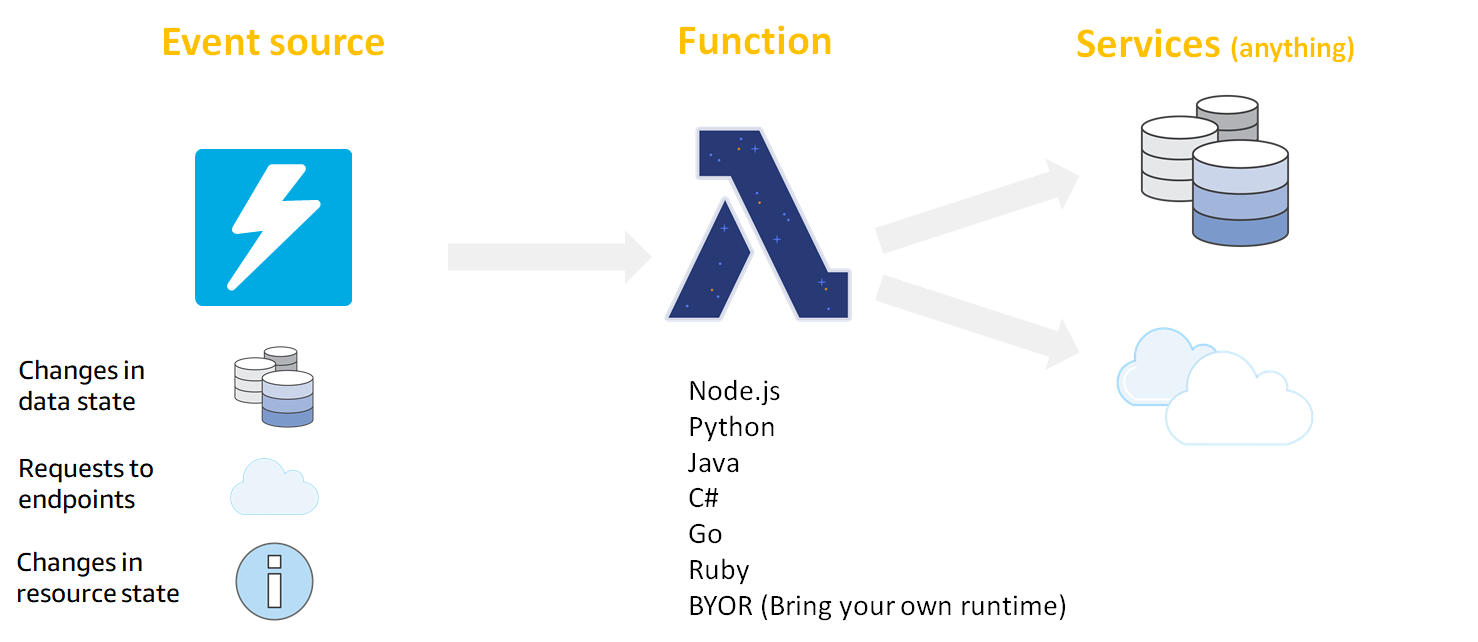
[Begin tutorial >>](https://aws.amazon.com/blogs/compute/integrating-amazon-eventbridge-into-your-serverless-applications/?e=gs2020&p=deepdiveserverless)

Moving to event-driven architectures

Triggers & event sources

Lambda functions are triggered by events. They then execute code in response to the trigger and can also generate their own events. There are a lot of options for triggering a Lambda function, and you have a lot of flexibility to create custom event sources to suit your specific needs.  
  
The main types of event sources are:

* Data stores, such as Amazon S3, Amazon DynamoDB or Amazon Kinesis, can trigger Lambda functions. If it stores data that you want to track changes to, you can potentially use it as an event source.
* Endpoints that emit events can invoke Lambda. For example, when you ask Alexa to do something, she emits an event that triggers a Lambda function.
* Messaging services, such as Amazon SQS or Amazon SNS, can also be event sources. For example, when you push something to an SNS topic, it might trigger a Lambda function.
* When certain actions occur within a repository, such as when committing code to your AWS CodeCommit repo, it can trigger a Lambda function to, for example, start your CI/CD build process.



Invoking AWS Lambda functions

Learn all about invoking AWS Lambda functions with our developer guide.

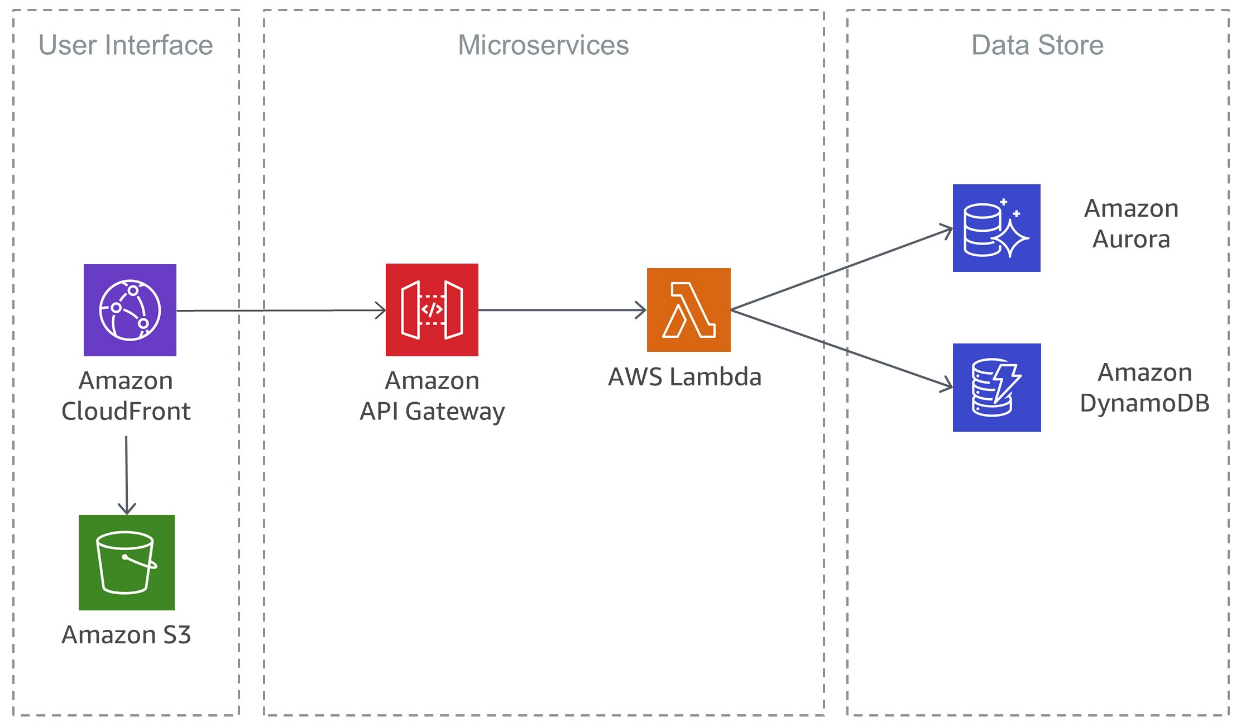
[See the developer guide >>](https://docs.aws.amazon.com/lambda/latest/dg/lambda-invocation.html)

Choosing Events, Queues, Topics, and Streams in Your Serverless Application

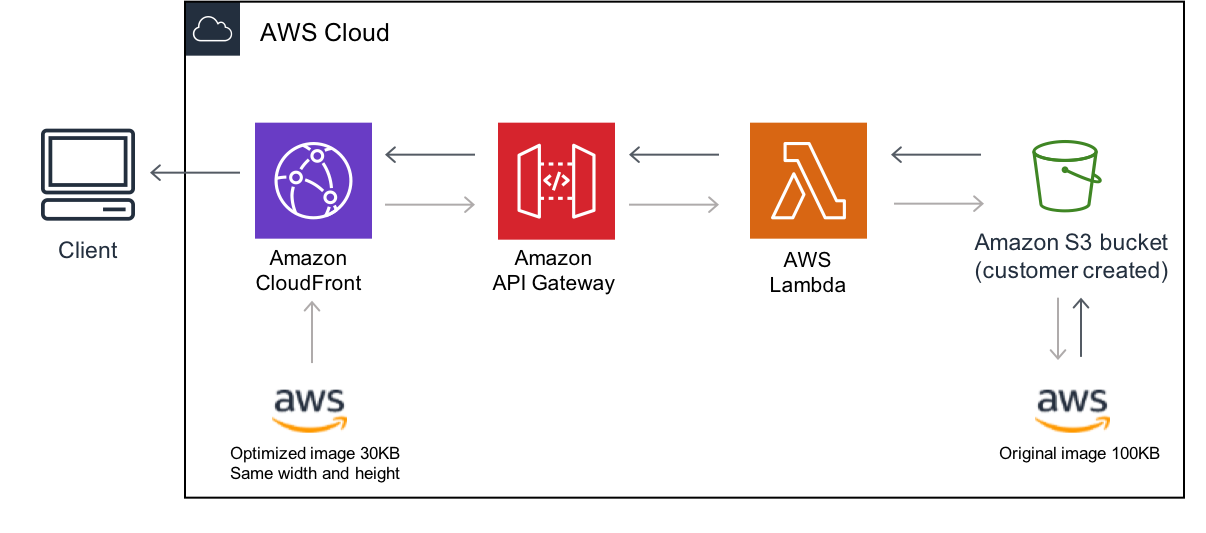
Reference architectures

In this section you will find a suite of reference architectures covering common serverless application use cases.

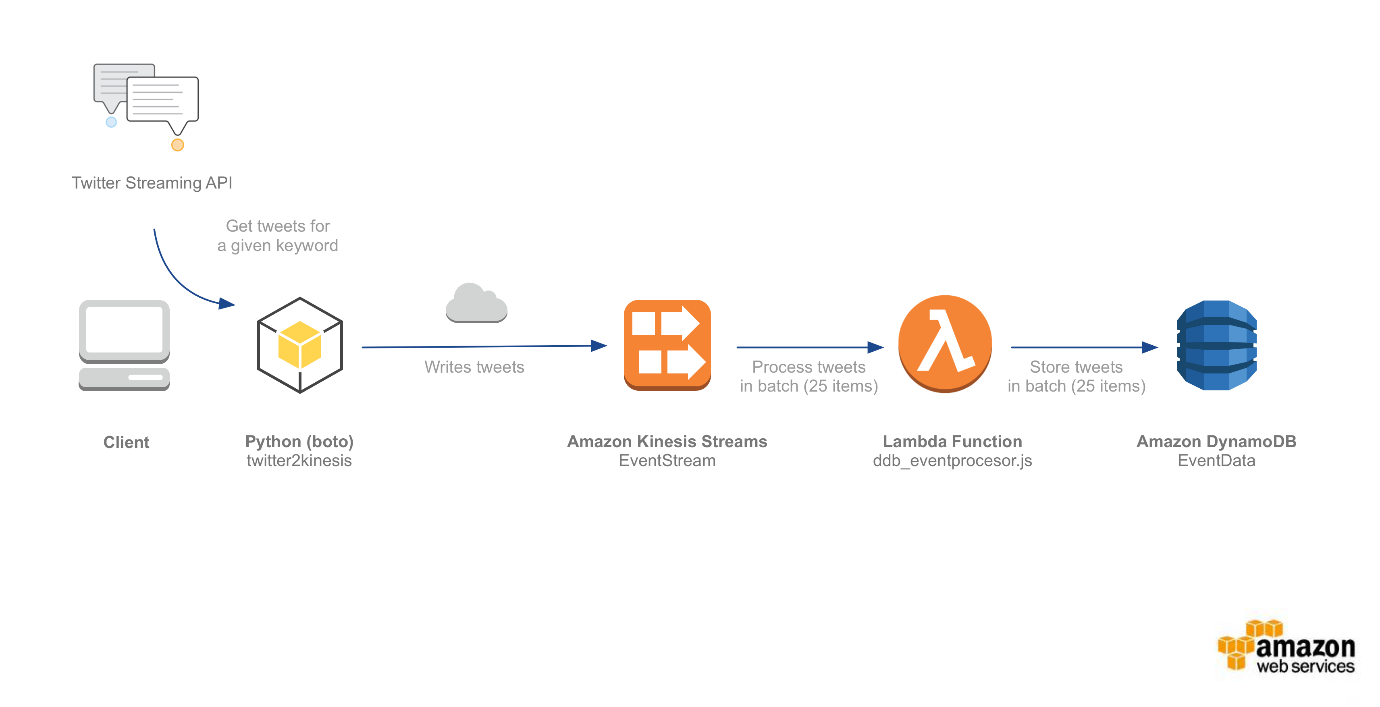
* RESTful microservices



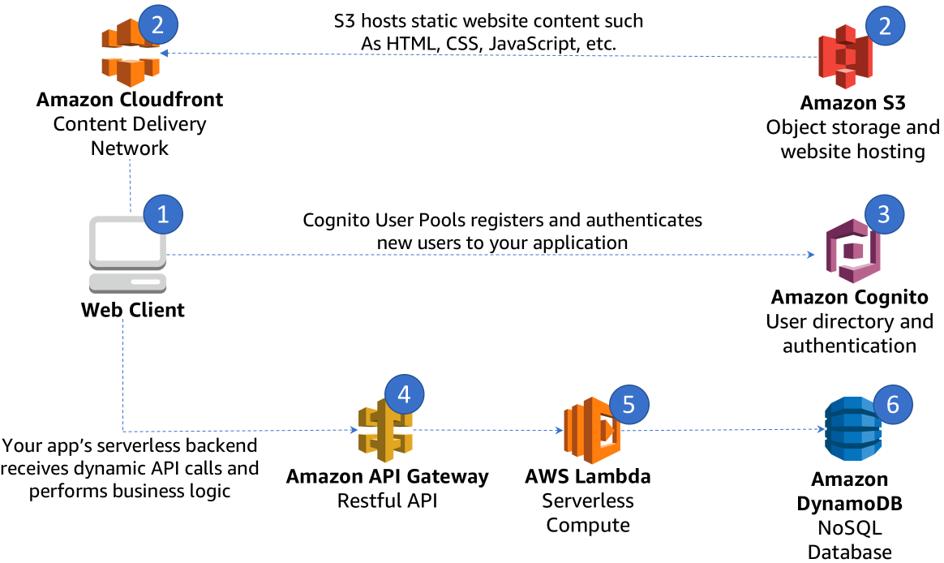
* Imaging processing



* Stream processing



* Web application



Deployment

A best practice for deployments in a microservice architecture is to ensure that a change does not break the service contract of the consumer. If the API owner makes a change that breaks the service contract and the consumer is not prepared for it, failures can occur.  
  
To understand the impact of deployment changes, you need to know which consumers are using your API. You can collect metadata on usage by using API keys, and these can also act as a form of contract if a breaking change is made to an API.  
  
When customers want to soften the impact of breaking changes to an API, they can clone the API and route customers to a different subdomain (for example, v2.my-service.com) to ensure that existing consumers aren’t impacted. This approach allows customers to only deploy new changes to the new API service contract, but comes with trade-offs. Customers taking this approach need to maintain two versions of the API, and will deal with infrastructure management and provisioning overhead.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Deployment** | **Customer Impact** | **Rollback** | **Event Model Factors** | **Deployment Speed** |
| All-at-once | All at once | Redeploy older version | Any event model at low concurrency rate | Immediate |
| Blue/Green | All at once with some level of production environment testing beforehand | Revert traffic to previous environment | Better for async and sync event models at medium concurrency workloads | Minutes to hours of validation and then immediate to customers |
| Canary/Linear | 1–10% typical initial traffic shift, then phased increases or all at once | Revert 100% of traffic to previous deployment | Better for high concurrency workloads | Minutes to hours |

* All-at-once deployments

All-at-once deployments involve making changes on top of the existing configuration. An advantage to this style of deployment is that backend changes to data stores, such as a relational database, require a much smaller level of effort to reconcile transactions during the change cycle. While this type of deployment style is low-effort and can be made with little impact in low-concurrency models, it adds risk when it comes to rollback and usually causes downtime. An example scenario to use this deployment model is for development environments where the user impact is minimal.

Blue/Green Deployments:

* With the blue / green deployment pattern, customers shift a section of traffic to the new live environment (green) while keeping the old environment (blue) warm, in case the the system needs to be rolled back. When using this pattern, it's best to keep changes small so rollbacks can be done quickly and easily. Blue / Green deployments are designed to reduce downtime and many customers are using them for deploying to production. API Gateway allows you to easily define which percentage of traffic is shifted to the the new green environment, making it an effective tool for this deployment pattern.  
    
  This style of deployment is particularly suitable for serverless architectures, which are both stateless, and decoupled from the underlying infrastructure.  
    
  You need the right indicators in place to know if a rollback is required. As a best practice, we recommend customers using CloudWatch high-resolution metrics, which can monitor in 1-second intervals, and quickly capture downward trends. Used with CloudWatch alarms, you can enable an expedited rollback to occur. CloudWatch metrics can be captured on API Gateway, Step Functions, Lambda (including custom metrics), and DynamoDB.
* Canary deployments

Canary Deployments:

Canary deployments are an ever-increasing way for you to leverage the new release of a software in a controlled environment and enabling rapid deployment cycles. Canary deployments involve deploying a small number of requests to the new change to analyze impact to a small number of your users.  
  
With canary deployments in API Gateway, you can deploy a change to your backend endpoint (for example, Lambda) while still maintaining the same API Gateway HTTP endpoint for consumers. In addition, you can also control what percentage of traffic is routed to new deployment and for a controlled traffic cut-over. A practical scenario for a canary deployment might be a new website. You can monitor the click-through rates on a small number of end users before shifting all traffic to the new deployment.