Building with AWS Serverless: How to Choose

First published November 22, 2022
Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.
Contents

Abstract ................................................................................................................................. 1
Are you Well-Architected? ................................................................................................. 1
Introduction ......................................................................................................................... 2
Finding the right approach ............................................................................................... 2
Determining your criteria for serverless ........................................................................... 2
  AWS services .................................................................................................................. 3
  AWS Lambda .................................................................................................................. 4
  Amazon ECS .................................................................................................................... 8
Choosing between Amazon ECS, App Runner, Batch, and Amazon Lightsail ............. 10
  AWS App Runner .......................................................................................................... 11
  AWS Batch ..................................................................................................................... 11
  AWS Lightsail ............................................................................................................... 11
  Amazon ECS in the container context ........................................................................... 12
Modern web applications ................................................................................................. 13
  AWS Amplify Hosting .................................................................................................... 13
  Amazon CloudFront and Amazon S3 .......................................................................... 14
  Amazon ECS and static site hosting ............................................................................ 14
Conclusion ......................................................................................................................... 14
Contributors ....................................................................................................................... 15
Abstract

This decision guide is designed for those who know that they want to choose a Serverless strategy for modern app development – or have come to that conclusion after reviewing the “Serverless or Kubernetes on AWS: how to choose” guide. It begins with the assumption that you plan to adopt a Serverless with AWS strategy for your app development.

This guide provides a mental model for architects exploring the “containers or functions” question for organizations that have decided to adopt the AWS Serverless strategy. The paper is meant to inform architectural decisions. It is not a decision tree, and doesn’t present a binary answer, but is instead designed to provide awareness of the inevitable trade-offs.

Are you Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

In the Serverless Application Lens, we focus on best practices for architecting your serverless applications on AWS.

For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the AWS Architecture Center.
Introduction

AWS provides a broad range of options to run cloud native applications such as microservices, APIs, event-driven architectures, and many others. It’s possible to use serverless and/or containers to build, run, and manage these types of applications. It’s important to understand that decisions on how to do this should be guided by a range of factors, including business use cases, strategy, skills, and managed services.

As stated at the outset, we begin with the assumption that you plan to adopt a Serverless with AWS strategy for your app development. To help you do that, we’ll walk through your Serverless options, and discuss specific AWS services, including AWS Lambda, AWS Fargate, and AWS App Runner, as well as the nuances of using Amazon Elastic Container Service (Amazon ECS) on Amazon Elastic Compute Cloud (Amazon EC2) in this context.

Finding the right approach

One of the tenets of modern application development is that developers create an application once – and then be able to update and maintain it. Those same developers (or another team - such as a DevOps or site reliability engineering (SRE) team) may also be responsible for managing, running, and releasing updates to the application. Serverless functions and serverless containers imply different approaches to building, managing, deploying, and releasing applications. Customers that choose the serverless approach benefit by reducing operational risk by pushing that work to AWS.

Often customers seek the benefits of standardization and built-in best practices when extending AWS to meet their organization’s needs. We advocate for choice in these cases, allowing your application architects to select the right level of compute abstraction for their workload’s needs. Building on the strong operational foundation of AWS-managed compute and integration services allows you to focus on the value-add differentiators and patterns particular to your organization. Customers that take this approach benefit from improved speed to market and lower operational burden.

Determining your criteria for serverless

Each workload built has different characteristics and requirements. For example, a document processing workload has very different latency and uptime requirements than a transactional website.
As such, we advise customers to evaluate the most appropriate compute platform on a workload-by-workload basis.

Some workloads lend themselves to one compute service over another. Other workloads may use a mix of options to get the best of those services. Consider the following in selecting the appropriate serverless compute service.

**AWS services**

AWS offers a number of serverless function and serverless container services.

The remainder of this document explores how the aforementioned considerations apply to those services in building modern applications. The following table provides a summary for comparison.

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Lambda</th>
<th>ECS / EC2</th>
<th>ECS / Fargate</th>
<th>App Runner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Most Ideal and cost effective for event-driven use cases such as event (messaging, streams) processing, APIs for Web Apps, IT automations, data pipelines.</td>
<td>Many workloads, especially HTTP-based traffic.</td>
<td>Many workloads, especially HTTP-based traffic.</td>
<td>Synchronous web services.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architectures</th>
<th>Lambda</th>
<th>ECS / EC2</th>
<th>ECS / Fargate</th>
<th>App Runner</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Developer Interaction</th>
<th>Lambda</th>
<th>ECS / EC2</th>
<th>ECS / Fargate</th>
<th>App Runner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event-driven architecture, microservices with event sourcing, orchestrations.</td>
<td>Developers eventually need to use container images and create AWS Deployment artifacts such as tasks and services, either through a DevOps pipeline or manually.</td>
<td>Developers eventually need to use container images and create AWS Deployment artifacts such as tasks and services, either through a DevOps pipeline or manually.</td>
<td>Deployment artifacts generated from Git repos or wrapped in a Docker image.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration Automation</th>
<th>Lambda</th>
<th>ECS / EC2</th>
<th>ECS / Fargate</th>
<th>App Runner</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS CloudFormation, Terraform, cloud development kit (CDK), Serverless Framework, Chalice and others.</td>
<td>AWS CloudFormation, Terraform, CDK.</td>
<td>AWS CloudFormation, Terraform, CDK.</td>
<td>CloudFormation, Terraform, CDK.</td>
<td></td>
</tr>
</tbody>
</table>
## DevOps Tools

| Code Pipelines, Code Build integrations, most popular open source tools. |
| Code Pipelines, Code Build integrations, most popular open source tools. |
| Code Pipelines, Code Build Integrations, most popular open source tools. |
| Code Pipelines, Code Build integrations, most popular open source tools. |

## Integrations

| 200+ managed Events Sources and targets for asynchronous patterns. Various integrations with DevOps tooling and observability tools. |
| Various integrations with DevOps tooling and observability tools. |
| Various integrations with DevOps tooling and observability tools. |
| Various integrations with DevOps tooling and observability tools. |

## Managed Service and Operational Overhead

| No servers to manage, functions are deployed and tuned based on event sources. Managed control plane, customer manages worker nodes. Managed control plane and worker nodes. |
| Managed control plane and worker nodes. |
| Managed control plane and worker nodes. |
| Managed control plane and worker nodes. |

## Ecosystem

| Large numbers of AWS partners and integrations, programming language communities. Docker and container ecosystem. AWS integrations and partners. Docke and container ecosystem. AWS integrations and partners. |
| Docker and container ecosystem. AWS integrations and partners. |
| Docker and container ecosystem. AWS integrations and partners. |
| Docker and container ecosystem. AWS integrations and partners. |

## Skills

| Developer and DevOps skills needed. No site reliability engineer (SRE). Developer and DevOps skills needed. Cluster administration needed. Developer and DevOps skills needed. Reduced skillset for managing cluster due to managed worker nodes. No SRE. Developer and DevOps skills needed. No SRE. |
| Developer and DevOps skills needed. Cluster administration needed. |
| Developer and DevOps skills needed. Reduced skillset for managing cluster due to managed worker nodes. No SRE. |
| Developer and DevOps skills needed. No SRE. |

## Performance, Scalability, and Resiliency

| Multi-Availability Zone, managed by AWS. Scaling by event source configuration and incoming demand. Cluster and service level automatic scaling. Service automatic scaling. Automatic scaling of compute resources based on parameters. |
| Cluster and service level automatic scaling. |
| Service automatic scaling. Automatic scaling of clusters. |
| Automatic scaling of compute resources based on parameters. |

## AWS Lambda

**AWS Lambda** is a serverless compute service that lets you run code without provisioning or managing servers. Lambda runs your code on a highly-available compute infrastructure and performs all administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring, and logging. A developer organizes their code into **Lambda functions**. Lambda runs your function only when needed and scales automatically, from a few requests per day to thousands per second.
You can invoke your Lambda functions using the Lambda API, or Lambda can run your functions in response to events from other AWS services. Lambda supports native integrations with more than 200 AWS services and software as a service (SaaS) applications.

Example use Cases

Lambda can be used by a variety of workloads. These are the most common we see among AWS customers:

- **IT automation** — Customers use Lambda for a variety of IT automation tasks. The event-driven nature of the Lambda service makes it an ideal fit to analyze AWS system events such as a new EC2 instance being launched, or running scheduled clean-up jobs. Lambda integrates with services such as AWS Config to customize configuration checks, and Amazon CloudWatch to process log messages.

- **Data processing/pipelines** — Many AWS data storage services integrate with Lambda to simplify data processing use cases. Managed integrations with Amazon DynamoDB Streams, for example, allow developers to write business logic and configure polling properties. Uploads to an Amazon Simple Storage Service (Amazon S3) bucket can trigger an action by Lambda, for example, to create an image thumbnail.

- **Event-driven use cases:** — Lambda is itself event-driven, your function only runs when invoked. Event-driven architectures enable greater flexibility, scalability, and reliability. System components are decoupled with events tying them together. Lambda fits naturally in a microservices architecture based on an event-driven pattern.

- **Web APIs and microservices** — Backends for web applications are often built with Lambda. The function resource model allows developers to clearly define microservice boundaries and independently deploy updates. Microservices implemented as asynchronous patterns have the added benefit of increased reliability. Services with sporadic utilization also benefit from Lambda's pay-for-use pricing structure.

Lambda can support many other use cases and workloads beyond those listed here. None of the workloads previously listed prevent similar workloads being implemented using containers. We’ve found that some workloads are better suited to Lambda due to its event-driven nature and deep integrations with other AWS services. This means that developers are more productive in building these solutions.
Workloads with low utilization and/or unpredictable scaling can offer compute cost savings on Lambda beyond always-on compute solutions. These savings are in addition to the lower total cost of ownership as Lambda’s operational model pushes ongoing maintenance to AWS. Asynchronous processing can create a lower latency experience for end users and is easy to implement using built-in integrations and well-understood patterns.

Customers also find value in building on Lambda when testing a new idea or capability. Lambda allows developers to quickly get started by creating and deploying a function without the need to invest in platform building. Consider best practices when building Lambda functions.

**Challenging workloads for Lambda**

We have already discussed workloads that benefit from the higher-level abstractions and built-in integrations available from AWS Lambda. Other workloads aren’t optimized for this.

Lambda is generally not a strong fit for modernizing an existing application. Serverless containers are preferred. Many existing applications rely on architectural patterns significantly different from Lambda or are designed for an always-on infrastructure. For example, legacy Java EE applications may rely on an HTTP session to keep state or use technologies such as dependency injection and reflection that were not intended to be initialized on every request. We often find that containers are a better initial landing spot for modernization initiatives. Customers will often utilize approaches such as the [strangler fig pattern](#) to evolve to a microservices-based architecture that relies on Lambda for growing (or replacement) functionality.

The previous workloads could also be implemented with containers but benefit from a function-based approach. The following workloads could also be implemented with Lambda. However, the workarounds are generally a less than ideal technical fit, and are better left to other options. Examples can include:

- **Legacy workloads/migrations** — The Lambda programming model requires adherence to an event-driven approach. Traditional applications, such as always-on web applications and long-running jobs, generally require re-architecture to run on Lambda without technical trade-offs. Consider the latency requirements and utilization pattern of the application to determine if these trade-offs will be acceptable.
- **Processing that exceeds fifteen minutes** — A single Lambda function can run for at most fifteen minutes. Various options to chain Lambda functions are available (such as direct invoke or Lambda destinations). However, we prefer an orchestration-based approach to split the function into multiple parts. [AWS Step Functions](https://aws.amazon.com/stepfunctions) provides the ability to readily coordinate data flow across Lambda functions and a multitude of other AWS services. Operations that need to be implemented as a single unit of work may be a fit for a service such as [AWS Batch](https://aws.amazon.com/batch), which can schedule jobs to run on containers.

- **Extreme high traffic workloads** — Lambda offers a pay-for-use pricing model in which you are only charged when your function runs. As noted previously, this can be beneficial for spiky or infrequent workloads. Lambda can also help lower total cost of ownership (TCO). In cases of extremely high traffic and utilization, the billing advantage may be to use an always-on solution. We find that this generally does not happen on day one, but only after achieving significant scale. Building portable application logic allows a transition from Lambda to serverless containers if needed.

- **Multi-threading** — Lambda allows multiple threads to run within the same function; however, you will generally not experience the same benefits as on other compute services. Each invocation is handled by a separate Lambda environment, which is then frozen while idle. If you do choose to use threads, be cautious about closing threads. The danger is that the environment is not kept active due to a continuously running thread.

- **Stateful processing** — Stateful applications are generally not great candidates for Lambda, and can be challenging for containers as well. If your application needs to cache state in memory or rely on caching services for heavily accessed data, Lambda may need to retrieve that data on each invocation. Although patterns exist to mitigate latency caused by retrieval (such as [Lambda provisioned concurrency](https://aws.amazon.com/lambda/concurrency/)), there may be other trade-offs in cost or complexity.

- **Ultra-low latency performance requirements** — Applications that require extremely low latency, even at p99 intervals (1% of requests or less under the latency requirement), may be challenging for Lambda. Lambda uses a request per running environment approach in which your function may “cold start” or initialize on any given call. This means that some proportion of calls may have higher latency. Various optimizations exist to mitigate this concern (such as provisioned concurrency, runtime selection and package minimization). Latency is also less of a concern for asynchronous processing.
• **Specialized hardware** — Applications that require hardware, such as graphics processing units (GPUs), more than 10GB of memory, or more than 10GB of local hard disk space, would require a different compute choice or re-architecture.

It’s important to remember that Lambda is a managed service and, as with all AWS services, will evolve over time. AWS continues to add new features and functionality to broaden the workloads Lambda can support. Review the current state of the service to confirm your understanding of the service.

Most often, Lambda is used in concert with other AWS services or event triggers. These include API Gateway, Step Functions, Event Bridge, Kafka, SQS, SNS, and many others. The deep integration between Lambda and these event sources allows customers to build rich, complex applications that meet a variety of business needs. In contrast to some other compute options, Lambda is optimized to manage integrations and provide smart defaults.

Developers can focus on business logic instead of managing a poller, for example. Lambda in almost all non-trivial cases will be used with some other service.

If your workload fits Lambda, it may also be a better fit for experienced developer teams and could help them be more productive. The lower total cost of ownership is worth the investment in finding or training your staff to support this model for building modern applications.

**Amazon ECS**

Containers are a popular solution to the problem of how to get software to run reliably when moving from one computing environment to another. Whether this is from a developer’s laptop to a test environment, a staging environment into production, or perhaps from a physical machine in a data center to a virtual machine (VM) in the cloud, containers allow for portability. Containers require fewer system resources than traditional or hardware VM environments, while also allowing more rapid deployment, patching, and scaling than VMs.

Containers offer a convenient packaging mechanism for application code and dependencies that can run on a variety of services and support a wide swath of applications. Containers are a good target for modernization, as they allow many traditional applications to continue to run as is. (As opposed to Lambda, which we have already established is not optimized for migration.)
Containers are a great fit for agile and DevOps efforts that accelerate development, test, and production cycles. Containers provide more consistent operations, because teams know applications deployed in containers will run and scale the same way regardless of where they are deployed. We’ve seen an explosion of container adoption over the past few years. With that growth in popularity, the ecosystem of applications, tooling, and communities has also grown and evolved.

Non-trivial container workloads require some degree of orchestration to support scaling and management of deployed containers. The following sections explore use cases and considerations for several of the available serverless container options available. These services simplify the technical choices and considerations in running container workloads on AWS. You can move faster when you are faced with fewer decisions and less of a need to build a containers platform team. Abstractions built on top of Amazon ECS, such as AWS App Runner and AWS Batch, offer fully managed purpose-built services for those types of workloads.

If your organization has already selected Kubernetes to build modern applications, we recommend reading “Building on Kubernetes with AWS: How to Choose.”

Example use cases

Container applications can run a variety of workloads. Many of the same workloads discussed earlier as being good fits for Lambda can be built on containers as well, though with trade-offs that architects must understand and explore. Some of the more common container workloads our customers build include:

- **Web applications and microservices** — Containers are a common platform for long-running tasks such as web application and microservices. These workloads include migrations from legacy platforms to gain at least some of the inherent cost and management benefits of containers.

- **Modernized middleware** — Customers often choose containers to take existing applications as a first step toward building microservices. Many legacy middleware applications can be modified to run on containers with minimal effort. Examples include Java Middleware and .NET. AWS and other vendors provide tools that can help ease these migrations. AWS App2Container, for example, enables customer to migrate Java and .NET web applications to containers for deployment on AWS.
• **Portable applications** — Customers with applications or application components that must live in multiple physical locations and/or clouds find the consistent deployment model of containers beneficial. If an application needs to be deployed in a production facility to meet latency needs, for example, containers allow that application to be built and tested elsewhere before deploying to local hosts.

## Choosing between Amazon ECS, App Runner, Batch, and Amazon Lightsail

AWS offers a number of options to run container-based applications. This range of options is designed to address the different needs of our customers. They vary in simplicity, flexibility and operational overhead. There is no one container-based solution that can serve all customers.

When you deploy containers, there are three layers of services that you can consider:

- **Capacity** (compute resource provider)
- **Orchestration** (coordination of deployment/resources)
- **Abstractions** (purpose-built access to orchestrator)

Depending on your organizational and workload needs, you can take advantage of one or more of these services. The following guidance uses the highest level of abstraction appropriate to your workload. Higher order abstractions simplify your interactions with the service and minimize the operational burden, enabling your teams to focus on differentiated capabilities. The choices you make will determine the amount of platform building, automation, maintenance, and management you will need to do.

Higher abstraction services, such as AWS App Runner and AWS Batch, are purpose-built for specific types of workloads. These services run on top of Amazon ECS, a general-purpose orchestration container service, and are designed to simplify the running of these applications. However, because of this simplification you will have less flexibility.

We recommend that you select your compute service on a workload-by-workload basis. Doing so allows you to maximize the advantage of using purpose-built services: cost, operations, performance, and so on.
AWS App Runner

AWS App Runner is an abstraction service built on Amazon ECS that is purpose-built to run request/response web service applications built with containers. App Runner is a fast, simple, and secure approach to deploying web application on AWS without you needing to manage infrastructure.

To run an application on App Runner you just need to provide source code or a container image. Then App Runner will build and deploy the application with the associated supporting resources, including load balancers, networking, and scaling policies. App Runner is designed to scale up and down based on demand and health of the application.

Customers have access to configuration options on task resources, including CPU and memory resources, scaling and health. In short, AWS App Runner is designed to be a fast, simple, and cost-effective way to deploy from source code or a container image directly to a scalable and secure web application in the AWS Cloud.

You don’t need to learn new technologies, decide which compute service to use, or know how to provision and configure AWS resources. The simplicity of the App Runner service can mean a faster time to market and the ability to focus on differentiated functionality. But it may not be your first choice if you want a lot of flexibility – and you need control of some of the elements that App Runner handles automatically for you (such as infrastructure management).

AWS Batch

AWS Batch is an abstraction service built on Amazon ECS that is purpose-built for batch workloads. Batch relies on ECS as a core orchestrator for managing and abstracting underlying infrastructure. The AWS Batch service provides a sophisticated batch scheduler that can be configured to support a variety of use cases. You pay for the underlying compute capacity utilized by your jobs (Amazon EC2 or Fargate), the fully managed batch jobs scheduler is included at no additional cost.

We recommend that you consider using Batch when building applications that perform batch, or asynchronous work that can be managed in a job queue. Batch manages the orchestration and processing of those jobs by the container application you provide.

AWS Lightsail

Amazon Lightsail provides a streamlined, simplified experience for customers early in their containers journey.
Lightsail is a fully-managed experience and provides support for web service and back-end worker applications packaged as containers. The service offers horizontal scaling of multi-container configuration, DNS configuration options, and basic logging and performance monitoring. The Lightsail all-in-one tiered pricing is easy to understand and estimate.

Lightsail is designed to help you get going quickly. You can use it to spin up websites or applications using pre-configured blueprints including WordPress, Prestashop, or LAMP. You can use Lightsail features to simply host static content, connect your content to a global audience, or get your Windows business server up and running. The Lightsail console is provided to guide you through the configuration process, and in many cases, has components already configured.

Amazon ECS in the container context

Amazon ECS is a fully-managed orchestration engine for running containers at scale. ECS is a highly scalable, fast container management service that makes it easy to run, stop, and manage Docker containers. Although Amazon ECS requires more configuration and decision-making than the previously mentioned services, there are fewer decisions around compute, network, and security than other general-purpose orchestrators without sacrificing scale or features.

We recommend using Amazon ECS if none of the previous scenarios and services fit your workload requirements.

If your strategy includes AWS as a primary cloud provider, we recommend starting with Amazon ECS when running container workloads. Because Amazon ECS is fully integrated into AWS, it simplifies networking, security, and other technical decisions. Also, the control plane is operated as a managed-service with access controlled by AWS Identity and Access Management (AWS IAM).

When using Amazon ECS, the primary decision you need to make is to choose between Amazon EC2 or AWS Fargate to provide capacity.

- **Amazon ECS on Fargate** is designed for customers who don’t want to worry about managing servers, handling capacity planning, or figuring out how to isolate container workloads. Amazon ECS on Fargate offers lower operational effort, there is no need for you to patch hosts or manage scaling. Amazon ECS on Fargate pricing is pay-for-use based on the CPU and memory resources allocated to each hourly running tasks.
• **Amazon ECS on EC2** offers the widest choice of instance types, including processor, storage, and networking. Amazon ECS on EC2 is ideal for customers who want to manage or customize the underlying compute environment and/or host operating system. The added flexibility allows you to use daemon sets, privileged containers, larger task sizes, and specialized hardware (such as GPU). Amazon ECS on EC2 does introduce additional operational work, including scaling and patching hosts. Amazon ECS costs are based on running EC2 instance hosts; the ECS control plane is free.

We recommend that you select the compute service with the highest level of abstraction, starting with Amazon ECS on Fargate. If you require a feature that is not available from Amazon ECS on Fargate, then we recommend that you move to Amazon ECS on EC2. We also recommend that you review our documentation for the most up-to-date features available, because these are subject to change as we evolve our services.

Both Fargate and EC2 options offer access to [Compute Savings Plans](#) and [Spot Instances](#).

### Modern web applications

We have seen a dramatic shift in the way web applications are built, deployed, and assembled over the past few years. Modern web applications are now often built using browser (or client-side) technologies such as React, Vue, Angular, or Svelte instead of server-rendered templates.

We can apply the same framework we have described throughout this paper to frontend applications as well, though with a caveat. Rich, JavaScript-driven frontends push compute to the client, relying on the end user's browser to assemble, render, and aggregate data. Many of the previously discussed options do not apply to modern web applications.

We recommend that you host modern web application frontends on a static site hosting service. These services are easy to manage and low cost to operate as they do not require active compute to serve content.

### AWS Amplify Hosting

[AWS Amplify Hosting](#) provides fast, reliable hosting of static content. Amplify Hosting is built on top of Amazon CloudFront, our content delivery network. Amplify also offers continuous build and deployment of web applications and a variety of other features.
You would choose Amplify Hosting if you need a fully managed CI/CD and hosting service for fast, secure, and reliable static and server-side rendered apps that is designed to scale with your business. It is aimed at not only getting you going quickly, but also providing a [one-stop-shop for building a static web site](#) on CloudFront.

**Amazon CloudFront and Amazon S3**

From a content service perspective, the pairing of [Amazon CloudFront](#) and [Amazon S3](#) offers similar hosting capabilities to Amplify, but with more granular control and more flexibility. As with many other choices, it’s a question of how much you want to manage and customize yourself. Like the container scenarios discussed previously, Amplify Hosting is purpose-built for running modern web applications. You will find the experience simpler than managing on a combination of Amazon S3 and CloudFront.

**Amazon ECS and static site hosting**

The more conventional approach for static-site hosting is to deploy your application in a container environment running NGINX, Apache, or another web server. This option is costlier, because you will need to maintain active compute to serve pages. It is also arguably the most complex to set up and maintain. If you require an internal or private web application, accessible only within your network, [Amazon ECS](#) may be the best option as other options are available on the internet. You can use [AWS WAF](#) to limit access, if needed, but some organizations may need an option that’s available exclusively on their internal network.

**Conclusion**

AWS offers a variety of options to deploy your workloads, such as microservices, legacy applications, or data processing. We recommend that you consider compute selection on a workload-by-workload basis instead of an organizational mandate. Purpose-built, higher-level abstractions enable teams to innovate and move faster while lowering the operational burden. While this does require architects in your organization to be familiar with a wider variety of services, we find this flexibility increases productivity and developer satisfaction.
Contributors

Authors:

- Roland Barcia, WWSO Director SA Specialists, Serverless
- Josh Kahn, Amazon Web Services, Tech Leader — Serverless, Principal Solutions Architect

Contributors and reviewers:

- Aaron Schwam, Senior Manager, Product Marketing, AWS Product Marketing
- Massimo Re Ferre, Senior Principal Technologist, AWS
- Jurgen Hofkens, Compute SA Leader, EMEA, AWS WWSO Compute
- Steve Follis, Principal Container Specialist, AWS WWSO Compute
- Sandhya Job, Senior Containers Specialist, AWS WWSO Compute
- Luca Mezzalira, Principal SA - Serverless, AWS WWSO Compute
- Uri Segev, Principal ISRAEL Serverless SSA, AWS WWSO Compute
- Roman Boiko, Senior DACH Serverless SA, AWS WWSO Compute
- Kevin Coleman, WW Head of GTM, Kubernetes, AWS WWSO Compute
- Daisy Daivasagaya, Senior Container Specialist, AWS WWSO Compute