Table of Contents

Compute in AWS

1 Compute in AWS

1.1 What is compute?

In cloud computing (AWS, n.d.; AWS Compute, n.d.), the term “compute” describes concepts and objects related to software computation. It is a generic term used to reference processing power, memory, networking, storage, and other resources required for the computational success of any program.

For example, applications that run machine learning algorithms or 3D graphics rendering functions require many gigs of RAM and multiple CPUs to run successfully. In this case, the CPUs RAM, Graphic Processing Units required will be called compute resources, and the applications would be compute intensive applications.

Let us look at some compute FAQs to understand the term in the context of modern computing.

1.2 What are compute resources?

Compute resources are measurable quantities of compute power that can be requested, allocated, and consumed for computing activities. Some examples of compute resources include:

1.2.1 CPU

The central processing unit (CPU) is the brain of any computer. CPU is measured in units called multicores. Application developers can specify how many allocated CPUs are required for running their application and to process data.

1.2.2 Memory

Memory is measured in bytes. Applications can make memory requests that are needed to run efficiently.

If applications are running on a single physical device, they have limited access to the compute resources of that device. But if applications run on the cloud, they can simultaneously access more processing resources from many physical devices. Let’s take a closer look at this.

AWS Compute Optimizer

1 AWS Compute Optimizer

AWS Compute Optimizer (AWS Compute Optimizer, n.d.) is a service that analyzes the configuration and utilization metrics of your AWS resources. It reports whether your resources are optimal, and generates optimization recommendations to reduce the cost and improve the performance of your workloads. Compute Optimizer also provides graphs showing recent utilization metric history data, as well as projected utilization for recommendations, which you can use to evaluate which recommendation provides the best price performance tradeoff. The analysis and visualization of your usage patterns can help you decide when to move or resize your running resources, and still meet your performance and capacity requirements.

Compute Optimizer provides a console experience, and a set of APIs that allows you to view the findings of the analysis and recommendations for your resources across multiple AWS Regions. You can also view findings and recommendations across multiple accounts, if you opt in the management account of an organization. The findings from the service are also reported in the consoles of the supported services, such as the Amazon EC2 console.

2 Supported resources and requirements

Compute Optimizer generates recommendations for the following resources:

- Amazon Elastic Compute Cloud (Amazon EC2) instances
- Amazon EC2 Auto Scaling groups
- Amazon Elastic Block Store (Amazon EBS) volumes
- AWS Lambda functions
- Amazon Elastic Container Service (Amazon ECS) services on AWS Fargate
- Commercial software licenses

For Compute Optimizer to generate recommendations for these resources, they must meet a specific set of requirements, and must have accumulated sufficient metric data.

3 Opting in

You must opt in to have Compute Optimizer analyze your AWS resources. The service supports standalone AWS accounts, member accounts of an organization, and the management account of an organization.

4 Analyzing metrics

After you opt in, Compute Optimizer begins analyzing the specifications and the utilization metrics of your resources from Amazon Cloud Watch for the last 14 days. For example, for Amazon EC2 instances, Compute Optimizer analyzes the vCPUs, memory, storage, and other specifications. It also analyzes the CPU utilization, network in and out, disk read and write, and other utilization metrics of currently running instances.

5 Enhancing recommendations

After you opt in, you can enhance your recommendations by activating recommendation preferences, such as the enhanced infrastructure metrics paid feature. It extends the metrics analysis lookback period for EC2 instances, including instances in Auto Scaling groups, to three months (compared to the 14day default).

Amazon Elastic Compute Cloud (EC2)

1 Amazon Elastic Compute Cloud (EC2)

Amazon Elastic Compute Cloud (AWS EC2, n.d.)Provides on demand, scalable computing capacity in the Amazon Web Services (AWS) Cloud. Using Amazon EC2 reduces hardware costs so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. You can add capacity (scale up) to handle compute heavy tasks, such as monthly or yearly processes, or spikes in website traffic. When usage decreases, you can reduce capacity (scale down) again.

The following diagram shows a basic architecture of an Amazon EC2 instance deployed within an Amazon Virtual Private Cloud (VPC). In this example, the EC2 instance is within an Availability Zone in the Region. The EC2 instance is secured with a security group, which is a virtual firewall that controls incoming and outgoing traffic. A private key is stored on the local computer and a public key is stored on the instance. Both keys are specified as a key pair to prove the identity of the user. In this scenario, the instance is backed by an Amazon EBS volume. The VPC communicates with the internet using an internet gateway.

Amazon EC2 supports the processing, storage, and transmission of credit card data by a merchant or service provider, and has been validated as being compliant with Payment Card Industry (PCI) Data Security Standard (DSS).

Amazon EC2 Working and Features

1 Amazon EC2 Working and Features

Amazon Elastic Compute Cloud (EC2) is a web service provided by Amazon Web Services (AWS) that enables users to run virtual servers in the cloud.

Here's an overview of how EC2 works and some of its key features:

2 How Amazon EC2 Works:

2.1 Instances:

EC2 instances are virtual servers that run on physical hardware within AWS's data centers. Users can launch instances with different configurations based on their computing needs.

2.2 AMI (Amazon Machine Image):

An AMI is a preconfigured template that includes the necessary information to launch an EC2 instance. It contains the operating system, application server, and applications.

2.3 Regions and Availability Zones:

EC2 instances can be launched in different geographic regions worldwide. Each region consists of multiple isolated locations called Availability Zones, providing redundancy and fault tolerance.

2.4 Elastic Load Balancer (ELB):

ELB distributes incoming application traffic across multiple EC2 instances to ensure that no single instance becomes a bottleneck.

2.5 Amazon EBS (Elastic Block Store):

EBS provides block level storage volumes for EC2 instances. It allows users to create and attach persistent storage volumes to instances.

2.6 Security Groups:

Security groups act as virtual firewalls for instances, controlling inbound and outbound traffic. Users define rules to allow or deny traffic based on port and protocol.

3 Key Features of Amazon EC2:

3.1 Scalability:

EC2 allows users to scale up or down by launching or terminating instances based on demand. Auto Scaling enables automatic adjustments to the number of instances in a group.

3.2 Variety of Instances:

EC2 provides a wide range of instance types optimized for different use cases, including compute optimized, memory optimized, storage optimized, and GPU instances.

3.3 Pay-as-You-Go Pricing:

EC2 follows a pay-as-you-go pricing model, where users pay for the compute capacity they use by the hour or second. Reserved Instances and Savings Plans offer cost savings for committed usage.

3.4 Customizable:

Users can customize instances based on their specific requirements, including the choice of operating system, instance type, storage, and network configurations.

3.5 Integration with Other AWS Services:

EC2 integrates seamlessly with other AWS services such as Amazon S3, Amazon RDS, AWS Lambda, and more, allowing users to build comprehensive and scalable applications.

Amazon EC2 instance types

1 Amazon EC2 instance types

Amazon EC2 provides a wide selection of instance types (AWS EC2 Instances, n.d.) optimized to fit different use cases. Instance types comprise varying combinations of CPU, memory, storage, and networking capacity and give you the flexibility to choose the appropriate mix of resources for your applications. Each instance type includes one or more instance sizes, allowing you to scale your resources to the requirements of your target workload.

1.1 General Purpose Instances:

- Instance Types: `t4g`, `t3`, `t3a`, `t2`, `t3i`
- Description: These instances provide a balance of compute, memory, and networking resources. They are suitable for a variety of diverse workloads.

1.2 Compute Optimized Instances:

- Instance Types: `c7g`, `c6g`, `c5`, `c5a`, `c5n`, `c4`
- Description: These instances are designed for compute intensive applications that require high performance processors. They are ideal for applications such as batch processing, scientific modeling, and high performance web servers.

1.3 Memory Optimized Instances:

- Instance Types: `u4sg`, `u6tb1.metal`, `u9tb1.metal`, `r7`, `r6g`, `r5`, `r5a`, `r5n`, `u24tb1.metal`, `u18tb1.metal`, `u12tb1.metal`, `u6tb1.metal`, `u9tb1.metal`, `x1e`, `x1`, `u12tb1.metal`
- Description: These instances are designed to deliver fast performance for workloads that process large data sets in memory. They are suitable for applications like real time big data analytics, in memory databases, and high performance computing.

1.4 Storage Optimized Instances:

- Instance Types: `i3`, `i3en`, `d2`, `h1`, `i4`, `u24tb1.metal`, `u18tb1.metal`, `u12tb1.metal`, `u6tb1.metal`, `u9tb1.metal`
- Description: These instances are designed for workloads that require high, sequential read and write access to very large data sets on local storage. They are ideal for applications such as NoSQL databases, data warehousing, and distributed file systems.

1.5 Accelerated Computing Instances:

- Instance Types: `p4`, `inf1`, `p3`, `p2`, `f1`
- Description: These instances use hardware accelerators or coprocessors to perform functions such as floating point number calculations, graphics processing, or data pattern matching more efficiently. They are suitable for applications like machine learning, gaming, and financial modeling.

2 Amazon EC2 Instance Size

Amazon EC2 Instance Size/Type dictates how much resources will be available for this instance.

This includes vCPU, Memory, Network Bandwidth, Instance Storage .

Instance	vCPU*	CPU Credits / hour	Mem (GiB)	Storage	Network Performance
t2.nano	1	3	0.5	EBS-Only	Low
t2.micro	1	6	1	EBS-Only	Low to Moderate
t2.small	1	12	2	EBS-Only	Low to Moderate
t2.medium	2	24	4	EBS-Only	Low to Moderate
t2.large	2	36	8	EBS-Only	Low to Moderate
t2.xlarge	4	54	16	EBS-Only	Moderate
t2.2xlarge	8	81	32	EBS-Only	Moderate

Amazon EC2 pricing

1 Amazon EC2 pricing

Amazon EC2 (EC2 Pricing, n.d.) provides the following pricing options:

1.1 Free Tier

You can get started with Amazon EC2 for free. To explore the Free Tier options, see AWS Free Tier.

1.2 On Demand Instances

Pay for the instances that you use by the second, with a minimum of 60 seconds, with no long-term commitments or upfront payments.

1.3 Savings Plans

You can reduce your Amazon EC2 costs by making a commitment to a consistent amount of usage, in USD per hour, for a term of 1 or 3 years.

1.4 Reserved Instances

You can reduce your Amazon EC2 costs by making a commitment to a specific instance configuration, including instance type and Region, for a term of 1 or 3 years.

1.5 Spot Instances

Request unused EC2 instances, which can reduce your Amazon EC2 costs significantly.

1.6 Dedicated Hosts

Reduce costs by using a physical EC2 server that is fully dedicated either for your use, On Demand or as part of a Savings Plan. You can use your existing server bound software licenses and get help meeting compliance requirements.

1.7 On Demand Capacity Reservations

Reserve compute capacity for your EC2 instances in a specific Availability Zone for any duration of time.

1.8 Per second billing

Removes the cost of unused minutes and seconds from your bill.

2 Estimates, billing, and cost optimization

To create estimates for your AWS use cases, use the AWS Pricing Calculator.

To see your bill, go to the Billing and Cost Management Dashboard in the AWS Billing and Cost Management console. Your bill contains links to usage reports that provide details about your bill. To learn more about AWS account billing, see AWS Billing and Cost Management User Guide.

To calculate the cost of a sample provisioned environment, see Cloud Economics Center. When calculating the cost of a provisioned environment, remember to include incidental costs such as snapshot storage for EBS volumes.

You can optimize the cost, security, and performance of your AWS environment using AWS Trusted Advisor.

Scaling AMAZON EC2 and Auto scaling

1 Scalability

Scalability, in the context of cloud computing and services like Amazon EC2, refers to the ability of a system to handle an increasing amount of workload or demand by adding resources or adjusting its capacity.

There are two main types of scalability: vertical scalability and horizontal scalability.

1.1 Vertical Scalability (Scaling Up):

- Definition: Vertical scalability involves increasing the capacity of a single resource, such as upgrading to a larger and more powerful server or adding more resources (CPU, RAM) to an existing server.
- Example: Upgrading an Amazon EC2 instance from a smaller instance type to a larger one with more CPU and memory resources.

1.2 Horizontal Scalability (Scaling Out):

- Definition: Horizontal scalability involves adding more instances or nodes to a system to distribute the workload and handle increased demand.
- Example: Adding more Amazon EC2 instances to a web application to distribute incoming traffic and improve overall performance.

2 Benefits of Scalability:

2.1 Flexibility:

Scalability allows systems to adapt to changing workloads and requirements. It provides flexibility in adjusting resources based on demand.

2.2 Performance:

Scalability ensures that performance remains optimal even as the number of users or the complexity of tasks increases. It helps maintain responsiveness and reliability.

2.3 Cost Efficiency:

With horizontal scalability, resources can be added incrementally as needed. This can lead to cost savings by avoiding overprovisioning and only paying for the resources consumed.

2.4 High Availability:

Scalable systems often involve redundancy and distribution of resources, improving overall system availability and reliability. If one component fails, others can continue to handle the workload.

3 What is Amazon EC2 Auto Scaling?

Amazon EC2 Auto Scaling helps you ensure that you have the correct number of Amazon EC2 instances available to handle the load for your application. You create collections of EC2 instances, called *Auto Scaling groups*. You can specify the minimum number of instances in each Auto Scaling group, and Amazon EC2 Auto Scaling ensures that your group never goes below this size. You can specify the maximum number of instances in each Auto Scaling group, and Amazon EC2 Auto Scaling ensures that your group never goes above this size. If you specify the desired capacity, either when you create the group or at any time thereafter, Amazon EC2 Auto Scaling ensures that your group has this many instances. If you specify scaling policies, then Amazon EC2 Auto Scaling can launch or terminate instances as demand on your application increases or decreases.

For example, the following Auto Scaling group has a minimum size of one instance, a desired capacity of two instances, and a maximum size of four instances. The scaling policies that you define adjust the number of instances, within your minimum and maximum number of instances, based on the criteria that you specify.

4 Features of Amazon EC2 Auto Scaling

With Amazon EC2 Auto Scaling, your EC2 instances are organized into Auto Scaling groups so that they can be treated as a logical unit for the purposes of scaling and management. Auto Scaling groups use launch templates (or launch configurations) as configuration templates for their EC2 instances.

The following are key features of Amazon EC2 Auto Scaling:

4.1 Monitoring the health of running instances

Amazon EC2 Auto Scaling automatically monitors the health and availability of your instances using EC2 health checks and replaces terminated or impaired instances to maintain your desired capacity.

4.2 Custom health checks

In addition to the built-in health checks, you can define custom health checks that are specific to your application to verify that it is responding as expected. If an instance fails your custom health check, it's automatically replaced to maintain your desired capacity.

4.3 Balancing capacity across Availability Zones

You can specify multiple Availability Zones for your Auto Scaling group, and Amazon EC2 Auto Scaling balances your instances evenly across the Availability Zones as the group scales. This provides high availability and resiliency by protecting your applications from failures in a single location.

4.4 Multiple instance types and purchase options

Within a single Auto Scaling group, you can launch multiple instance types and purchase options (Spot and On Demand Instances), allowing you to optimize costs through Spot Instance usage. You can also take advantage of Reserved Instance and Savings Plan discounts by using them in conjunction with On Demand Instances in the group.

4.5 Automated replacement of Spot Instances

If your group includes Spot Instances, Amazon EC2 Auto Scaling can automatically request replacement Spot capacity if your Spot Instances are interrupted. Through Capacity Rebalancing, Amazon EC2 Auto Scaling can also monitor and proactively replace your Spot Instances that are at an elevated risk of interruption.

4.6 Load balancing

You can use Elastic Load Balancing or VPC Lattice load balancing and health checks to ensure an even distribution of application traffic to your healthy instances. Whenever instances are launched or terminated, Amazon EC2 Auto Scaling automatically registers and deregisters the instances from the load balancer.

4.7 Scalability

Amazon EC2 Auto Scaling also provides several ways for you to scale your Auto Scaling groups. Using auto scaling allows you to maintain application availability and reduce costs by adding capacity to handle peak loads and removing capacity when demand is lower. You can also manually adjust the size of your Auto Scaling group as needed.

4.8 Instance refresh

The instance refresh feature provides a mechanism to update instances in a rolling fashion when you update your AMI or launch template. You can also use a phased approach, known as a canary deployment, to test a new AMI or launch template on a small set of instances before rolling it out to the whole group.

4.9 Lifecycle hooks

Lifecycle hooks are useful for defining custom actions that are invoked as new instances launch or before instances are terminated. This feature is particularly useful for building event driven architectures, but it also helps you manage instances through their lifecycle.

4.10 Support for stateful workloads

Lifecycle hooks also offer a mechanism for persisting state on shut down. To ensure continuity for stateful applications, you can also use scale in protection or custom termination policies to prevent instances with long running processes from terminating early.

For more information about the benefits of Amazon EC2 Auto Scaling, see (Amazon EC2 Auto Scalling, n.d.)

What is Load Balancing ?

1 What is Load Balancing ?

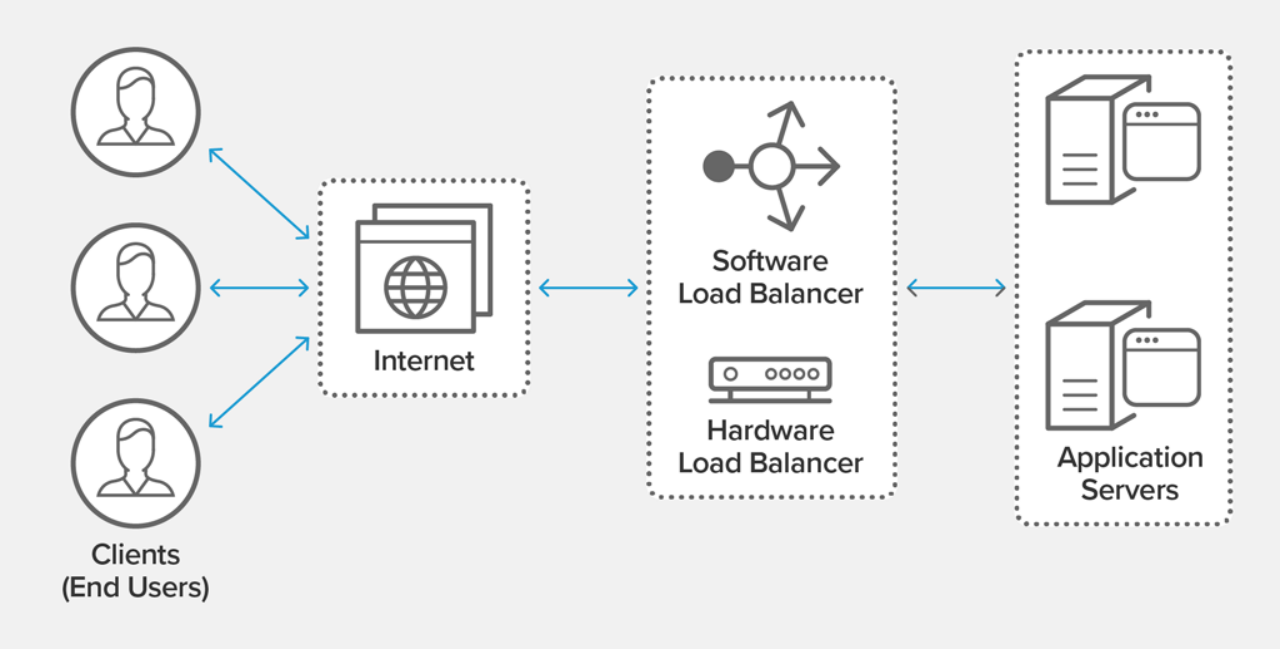
Load balancing (Load Balancing, n.d.) refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a *server farm* or *server pool*.

Modern high traffic websites must serve hundreds of thousands, if not millions, of concurrent requests from users or clients and return the correct text, images, video, or application data, all in a fast and reliable manner. To cost-effectively scale to meet these high volumes, modern computing best practice generally requires adding more servers.

A load balancer acts as the “traffic cop” sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it.

In this manner, a load balancer performs the following functions:

- Distributes client requests or network load efficiently across multiple servers
- Ensures high availability and reliability by sending requests only to servers that are online
- Provides the flexibility to add or subtract servers as demand dictates



2 Load Balancing Algorithms

Different load balancing algorithms provide different benefits; the choice of load balancing method depends on your needs:

2.1 Round Robin

Requests are distributed across the group of servers sequentially.

2.2 Least Connections

A new request is sent to the server with the fewest current connections to clients. The relative computing capacity of each server is factored into determining which one has the least connections.

2.3 Least Time

Sends requests to the server selected by a formula that combines the
fastest response time and fewest active connections. Exclusive to NGINX Plus.

2.4 Hash

Distributes requests based on a key you define, such as the client IP address or the request URL. NGINX Plus can optionally apply a consistent hash to minimize redistribution of loads if the set of upstream servers changes.

2.5 IP Hash

The IP address of the client is used to determine which server receives the request.

2.6 Random with Two Choices

Picks two servers at random and sends the request to the one that is selected by then applying the Least Connections algorithm (or for NGINX Plus the Least Time algorithm, if so configured).

3 Benefits of Load Balancing

- Reduced downtime
- Scalable
- Redundancy
- Flexibility
- Efficiency

Amazon Elastic Load Balancing (ELB)

1 Amazon Elastic Load Balancing (ELB)

Elastic Load Balancing (ELB, n.d.) automatically distributes your incoming traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in one or more Availability Zones. It monitors the health of its registered targets, and routes traffic only to the healthy targets. Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

2 Load balancer benefits

A load balancer distributes workloads across multiple compute resources, such as virtual servers. Using a load balancer increases the availability and fault tolerance of your applications.

You can add and remove compute resources from your load balancer as your needs change, without disrupting the overall flow of requests to your applications.

You can configure health checks, which monitor the health of the compute resources, so that the load balancer sends requests only to the healthy ones. You can also offload the work of encryption and decryption to your load balancer so that your compute resources can focus on their main work.

3 Accessing Elastic Load Balancing

You can create, access, and manage your load balancers using any of the following interfaces:

3.1 AWS Management Console

Provides a web interface that you can use to access Elastic Load Balancing.

3.2 AWS Command Line Interface (AWS CLI)

Provides commands for a broad set of AWS services, including Elastic Load Balancing. The AWS CLI is supported on Windows, macOS, and Linux.

3.3 AWS SDKs

Provide language specific APIs and take care of many of the connection details, such as calculating signatures, handling request retries, and error handling.

3.4 Query API

Provides low-level API actions that you call using HTTPS requests. Using the Query API is the most direct way to access Elastic Load Balancing. However, the Query API requires that your application handle low-level details such as generating the hash to sign the request, and error handling.

Messaging and Queuing

1 Messaging and Queuing

A Message Queue (MS1, n.d.) is a fundamental concept in distributed systems, allowing for asynchronous communication between different components or applications.

Here's a more detailed explanation:

1.1 Definition:

- Queue: A queue is a line of items waiting to be processed in sequential order, with the first item added being the first to be processed (FIFO First In, First Out).
- Message Queue: In the context of distributed systems, a message queue (MS2, n.d.) is a mechanism that facilitates communication between different applications by managing a sequence of messages.

1.2 Components:

- Producer: The producer is an application or component that creates messages and sends them to the message queue. Producers generate messages based on certain events or triggers.
- Consumer: The consumer is an application or component that connects to the message queue to retrieve and process messages. Consumers act on the messages received based on the application's logic.

1.3 Message:

- A message is the data payload transported between the sender (producer) and the receiver (consumer) applications. Messages can contain information, commands, or data that needs to be processed.
- Examples of messages include task instructions, task completion notifications, or simple status updates.

1.4 Basic Architecture:

- Producers create messages and deliver them to the message queue.
- Consumers connect to the queue and retrieve messages to process.
- Messages remain in the queue until a consumer retrieves and processes them.
- This architecture allows for decoupling between producers and consumers, as they do not need to be aware of each other's existence.

1.5 Benefits:

- **Asynchronous Communication:** Message queues enable asynchronous communication between distributed components, allowing systems to be more responsive and loosely coupled.
- **Scalability:** The decoupled nature of message queues makes it easier to scale applications. Producers and consumers can operate independently, and additional consumers can be added as needed.
- **Reliability:** Messages are stored in the queue until they are successfully processed, providing a level of fault tolerance. If a consumer fails or becomes unavailable, messages can be retried or picked up by other consumers.

Monolithic Applications vs Micro services

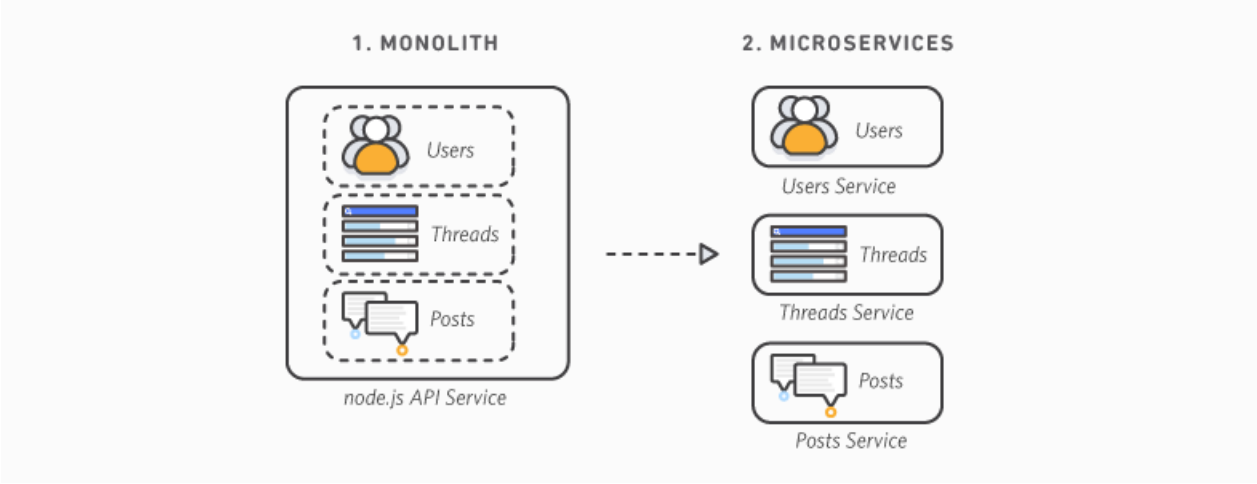
1 Monolithic Applications vs Micro services

A monolithic architecture (MM Differences, n.d.) is a traditional software development model that uses one code base to perform multiple business functions. All the software components in a monolithic system are interdependent due to the data exchange mechanisms within the system. It's restrictive and time consuming to modify monolithic architecture as small changes impact large areas of the code base. In contrast, micro services are an architectural approach that composes software into small independent components or services. Each service performs a single function and communicates with other services through a well-defined interface. Because they run independently, you can update, modify, deploy, or scale each service as required.

2 Key differences: monolithic vs. micro services

Monolithic applications typically consist of a client side UI, a database, and a server side application. Developers build all of these modules on a single code base.

On the other hand, in a distributed architecture, each micro service works to accomplish a single feature or business logic. Instead of exchanging data within the same code base, micro services communicate with an API.



Next, we discuss more differences between the two.

2.1 Development process

Monolithic applications are easier to start with, as not much upfront planning is required. You can get started and keep adding code modules as needed. However, the application can become complex and challenging to update or change over time.

A micro service architecture requires more planning and design before starting. Developers must identify different functions that can work independently and plan consistent APIs. However, the initial coordination makes code maintenance much more efficient. You can make changes and find bugs faster. Code reusability also increases over time.

2.2 Deployment

Deploying monolithic applications is more straightforward than deploying micro services. Developers install the entire application code base and dependencies in a single environment.

In contrast, deploying micro service based applications is more complex, as each micro service is an independently deployable software package. Developers usually containerize micro services before deploying them. Containers package the code and related dependencies of the micro service for platform independence.

2.3 Debugging

Debugging is a software process to identify coding errors that cause the application to behave erratically. When debugging monolith architecture, the developer can trace data movement or examine code behavior within the same programming environment. Meanwhile, identifying coding issues in a micro service architecture requires looking at multiple loosely coupled individual services.

It can be more challenging to debug micro service applications because several developers might be responsible for many micro services. For instance, debugging may require coordinated tests, discussions, and feedback amongst team members, which takes more time and resources.

2.4 Modifications

A small change in one part of a monolithic application affects multiple software functions because of the tightly coupled coding. In addition, when developers introduce new changes to a monolithic application, they must retest and redeploy the entire system on the server.

In contrast, the micro services approach allows flexibility. It’s easier to make changes to the application. Instead of modifying all the services, developers only change specific functions. They can also deploy particular services independently. Such an approach is helpful in the continuous deployment workflow where developers make frequent small changes without affecting the system's stability.

2.5 Scaling

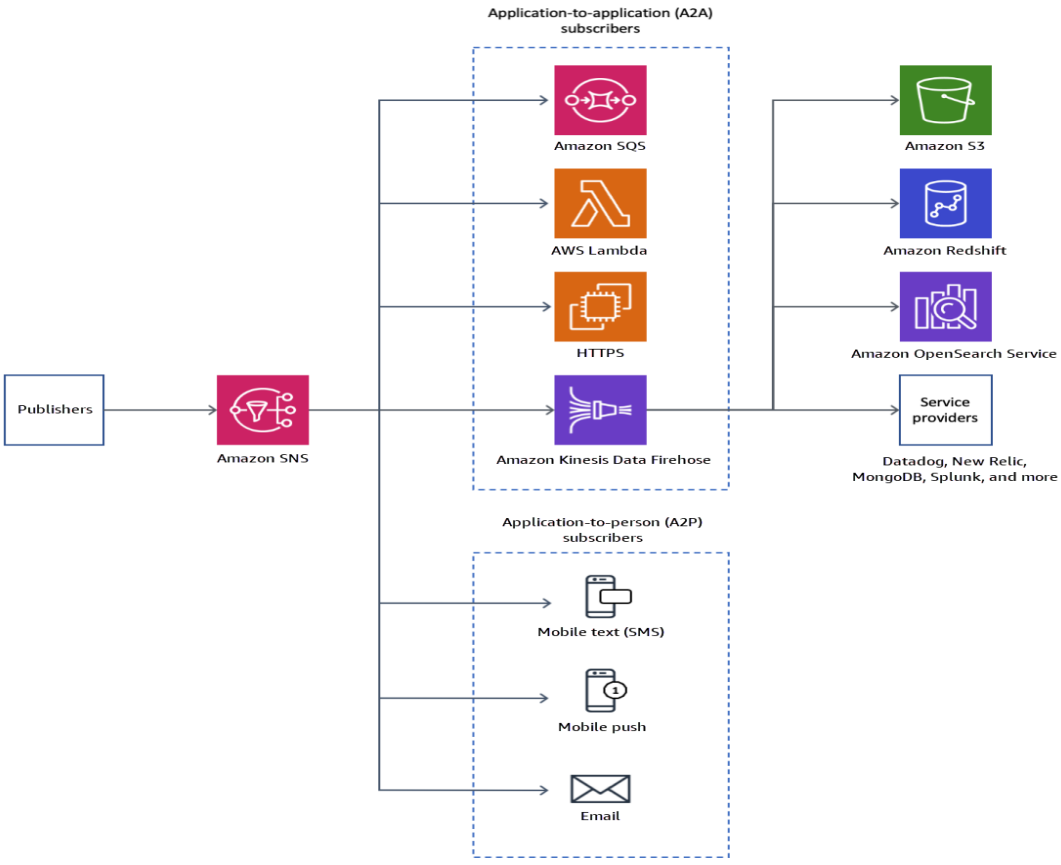
Monolithic applications face several challenges as they scale. The monolithic architecture contains all functionalities within a single code base, so the entire application must be scaled as requirements change. For example, if the application's performance degrades because the communication function experiences a traffic surge, you must increase the compute resources to accommodate the entire monolithic application. This results in resource wastage because not all parts of the application are at peak capacity.

Meanwhile, the micro services architecture supports distributed systems. Each software component receives its own computing resources in a distributed system. These resources can be scaled independently based on current capacities and predicted demands. So, for example, you can allocate more resources to a geographic location service instead of the whole system.

Amazon Simple Notification Service (SNS)

1 Amazon Simple Notification Service (SNS)

Amazon Simple Notification Service (SNS Amazon , n.d.) is a managed service that provides message delivery from publishers to subscribers (also known as *producers* and *consumers*). Publishers communicate asynchronously with subscribers by sending messages to a *topic*, which is a logical access point and communication channel. Clients can subscribe to the SNS topic and receive published messages using a supported endpoint type, such as Amazon Kinesis Data Firehose, Amazon SQS, AWS Lambda, HTTP, email, mobile push notifications, and mobile text messages (SMS).



2 Features and capabilities

Amazon SNS provides the following features and capabilities:

2.1 Application to application messaging

Application to application messaging supports subscribers such as Amazon Kinesis Data Firehose delivery streams, Lambda functions, Amazon SQS queues, HTTP/S endpoints, and AWS Event Fork Pipelines.

2.2 Application to person notifications

Application to person notifications provide user notifications to subscribers such as mobile applications, mobile phone numbers, and email addresses.

2.3 Standard and FIFO topics

Use a FIFO topic to ensure strict message ordering, to define message groups, and to prevent message duplication. You can use both FIFO and standard queues to subscribe to a FIFO topic.

Use a standard topic when message delivery order and possible message duplication are not critical. All of the supported delivery protocols can subscribe to a standard topic.

2.4 Message durability

Amazon SNS uses a number of strategies that work together to provide message durability:

- Published messages are stored across multiple, geographically separated servers and data centers.
- If a subscribed endpoint is not available, Amazon SNS runs a delivery retry policy.
- To preserve any messages that are not delivered before the delivery retry policy ends, you can create a dead letter queue.

2.5 Message archiving, replay, and analytics

You can archive messages with Amazon SNS in multiple ways including subscribing Kinesis Data Firehose delivery streams to SNS topics, which allows you to send notifications to analytics

endpoints such as Amazon Simple Storage Service (Amazon S3) buckets, Amazon Redshift tables, and more. Additionally, Amazon SNS FIFO topics support message archiving and replay as a no code, in place message archive that lets topic owners store (or *archive*) messages within their topic. Topic subscribers can then retrieve (or *replay*) the archived messages back to a subscribed endpoint.

2.6 Message attributes

Message attributes let you provide any arbitrary metadata about the message.

2.7 Message filtering

By default, each subscriber receives every message published to the topic. To receive a subset of the messages, a subscriber must assign a filter policy to the topic subscription. A subscriber can also define the filter policy scope to enable payload based or attribute based filtering. The default value for the filter policy scope is `Message Attributes`. When the incoming message attributes match the filter policy attributes, the message is delivered to the subscribed endpoint. Otherwise, the message is filtered out. When the filter policy scope is `Message Body`, filter policy attributes are matched against the payload.

2.8 Message security

Server side encryption protects the contents of messages that are stored in Amazon SNS topics, using encryption keys provided by AWS KMS.

Amazon Simple Queue Service (SQS)

1 Amazon Simple Queue Service (SQS)

Amazon Simple Queue Service (SQS Amazon, n.d.) offers a secure, durable, and available hosted queue that lets you integrate and decouple distributed software systems and components. Amazon SQS offers common constructs such as dead letter queues and cost allocation tags. It provides a generic web services API that you can access using any programming language that the AWS SDK supports.

2 Benefits of using Amazon SQS

2.1 Security

You control who can send messages to and receive messages from an Amazon SQS queue. You can choose to transmit sensitive data by protecting the contents of messages in queues by using default Amazon SQS managed server side encryption (SSE), or by using custom SSE keys managed in AWS Key Management Service (AWS KMS).

2.2 Durability

For the safety of your messages, Amazon SQS stores them on multiple servers. Standard queues support at least once message delivery, and FIFO queues support exactly once message processing and high throughput mode.

2.3 Availability

Amazon SQS uses redundant infrastructure to provide highly concurrent access to messages and high availability for producing and consuming messages.

2.4 Scalability

Amazon SQS can process each buffered request independently, scaling transparently to handle any load increases or spikes without any provisioning instructions.

2.5 Reliability

Amazon SQS locks your messages during processing, so that multiple producers can send and multiple consumers can receive messages at the same time.

2.6 Customization

Your queues don't have to be exactly alike—for example, you can set a default delay on a queue. You can store the contents of messages larger than 256 KB using Amazon Simple Storage Service (Amazon S3) or Amazon DynamoDB, with Amazon SQS holding a pointer to the Amazon S3 object, or you can split a large message into smaller messages.

Server less computing

1 Server less computing

Server less computing (Serverless Computing, n.d.) has been dominating the headlines, tech shows, and industry conferences. Even though the idea of 'server less' has been around since 2006, it is a relatively new concept. Many people seem to believe that server less is the natural evolution of computing model. In this article, we will explore key concepts of server less computing and introduce a solution from Alibaba Cloud called Function Compute, an event driven and fully managed compute service.

2 Evolution in Computing Models

Over the last few decades, Enterprise IT has evolved at a rapid pace, from physical servers, to virtual machines and to containerization technology.

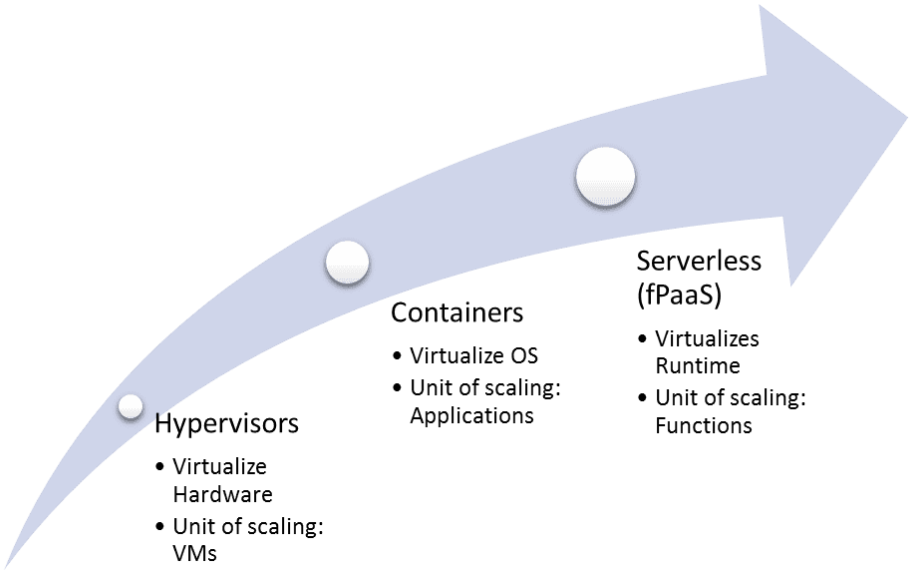
The first generation of computing was based on Von Neumann architecture and employed fixed programs. These programs were usually tightly coupled with the hardware they ran on. As we transitioned to disk based storage, the operating system became less dependent on the hardware it ran on. However, software applications came to resemble like a "monolith" in which the user interface and data access code were combined into a single program running on one platform. These monoliths were self-contained and any changes needed to the functionality required a huge undertaking. While the operating systems became less dependent on the hardware they ran on, the applications themselves became more dependent on the specific operating system. In early 2000s, if we had to launch a new application, we probably had to plan purchase of a new server first. So in summary, the physical server were the building block of an application.

Then in 2001, virtualization technology started to gain traction as the hardware became commoditized (especially x86 based architecture) and made virtual machines very

popular. Hypervisor software, made popular by companies such as VMWare, made it easy to create virtual machines. By running many virtual machines on one physical device, the need to buy physical servers became less and the architectural building block shifted from a physical server to a virtual machine.

While virtualization made it easy to operate machines, virtual disk images were rather large and clumsy, and were not easy to scale as required by the volatile computing workload. Container technology enabled operating system level virtualization and proved to be a much more lightweight alternative instead. Along the way, we saw several other major developments. We ae going to mention some of the key ones.

- In 2009: He Roku, popularized Platform as a Service, enabling a containerized 12factor application via a build pack
- In 2010: OpenStack brought together an extraordinary diverse group of vendors to create an open source infrastructure as a service
- In 2011: Cloud Foundry, an open source PaaS, along with similar tools, helped accelerate DevOps movement by supporting continuous delivery over the entire application development lifecycle. Cloud Foundry's container based architecture ran apps in any programming language over a variety of cloud service providers
- And in 2013: Docker (combined LCX, Union file system and cgroups) created a containerization standard which became an instant hit among the developer community



In summary, the containerization technology made it easy to build and deploy micro services based architecture, enabling further loose coupling and much greater scalability.

3 Server less Key Challenges

Many challenges remain, even with the use of container technology. Some of these include:

Capacity planning is not trivial. Infrastructure engineers still need to plan ahead for computing resource utilization, such as how many or what size or machine types would need to be configured (CPU, memory and networking or I/O bandwidth requirements). Overestimation can lead to increased costs, and underestimation could hurt the business.

Machine level auto scaling is cumbersome. In some instances, it may take several minutes to scale out. Management of real time auto scaling and load balancing takes a lot of planning effort.

And similar challenges exist around fault tolerance, operations and maintenance.

Some of these challenges can be addressed by many open source and proprietary solutions, however, they still require resource commitment and are not trivial to execute, especially by small and medium sized businesses.

AWS Lambda

1 AWS Lambda

AWS Lambda (Lambda, n.d.) is a compute service that lets you run code without provisioning or managing servers.

Lambda runs your code on a high availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, and logging. With Lambda, all you need to do is supply your code in one of the language runtimes that Lambda supports.

You organize your code into Lambda functions. The Lambda service runs your function only when needed and scales automatically. You only pay for the compute time that you consume there is no charge when your code is not running.

2 When to use Lambda

Lambda is an ideal compute service for application scenarios that need to scale up rapidly, and scale down to zero when not in demand. For example, you can use Lambda for:

2.1 File processing:

Use Amazon Simple Storage Service (Amazon S3) to trigger Lambda data processing in real time after an upload.

2.2 Stream processing:

Use Lambda and Amazon Kinesis to process real-time streaming data for application activity tracking, transaction order processing, clickstream analysis, data cleansing, log filtering, indexing, social media analysis, Internet of Things (IoT) device data telemetry, and metering.

2.3 Web applications:

Combine Lambda with other AWS services to build powerful web applications that automatically scale up and down and run in a highly available configuration across multiple data centers.

2.4 IoT backends:

Build server less back ends using Lambda to handle web, mobile, IoT, and third party API requests.

2.5 Mobile backends:

Build backends using Lambda and Amazon API Gateway to authenticate and process API requests. Use AWS Amplify to easily integrate with your iOS, Android, Web, and React Native frontends.

When using Lambda, you are responsible only for your code. Lambda manages the compute fleet that offers a balance of memory, CPU, network, and other resources to run your code. Because Lambda manages these resources, you cannot log in to compute instances or customize the operating system on provided runtimes.

Lambda performs operational and administrative activities on your behalf, including managing capacity, monitoring, and logging your Lambda functions.

If you do need to manage your compute resources, AWS has other compute services to consider, such as:

- AWS App Runner builds and deploys containerized web applications automatically, load balances traffic with encryption, scales to meet your traffic needs, and allows for the configuration of how services are accessed and communicate with other AWS applications in a private Amazon VPC.
- AWS Fargate with Amazon ECS runs containers without having to provision, configure, or scale clusters of virtual machines.

- Amazon EC2 lets you customize operating system, network and security settings, and the entire software stack. You are responsible for provisioning capacity, monitoring fleet health and performance, and using Availability Zones for fault tolerance.

3 Key features

The following key features help you develop Lambda applications that are scalable, secure, and easily extensible:

3.1 Configuring function options

Configure your Lambda function using the console or AWS CLI.

3.2 Environment variables

Use environment variables to adjust your function's behavior without updating code.

3.3 Versions

Manage the deployment of your functions with versions, so that, for example, a new function can be used for beta testing without affecting users of the stable production version.

3.4 Container images

Create a container image for a Lambda function by using an AWS provided base image or an alternative base image so that you can reuse your existing container tooling or deploy larger workloads that rely on sizable dependencies, such as machine learning.

3.5 Layers

Package libraries and other dependencies to reduce the size of deployment archives and makes it faster to deploy your code.

3.6 Lambda extensions

Augment your Lambda functions with tools for monitoring, observability, security, and governance.

3.7 Function URLs

Add a dedicated HTTP(S) endpoint to your Lambda function.

3.8 Response streaming

Configure your Lambda function URLs to stream response payloads back to clients from Node.js functions, to improve time to first byte (TTFB) performance or to return larger payloads.

3.9 Concurrency and scaling controls

Apply fine-grained control over the scaling and responsiveness of your production applications.

3.10 Code signing

Verify that only approved developers publish unaltered, trusted code in your Lambda functions

3.11 Private networking

Create a private network for resources such as databases, cache instances, or internal services.

3.12 File system access

Configure a function to mount an Amazon Elastic File System (Amazon EFS) to a local directory, so that your function code can access and modify shared resources safely and at high concurrency.

3.13 Lambda Snap Start for Java

Improve startup performance for Java runtimes by up to 10x at no extra cost, typically with no changes to your function code.

Amazon Elastic Container Service (ECS)

1 Amazon Elastic Container Service (ECS)

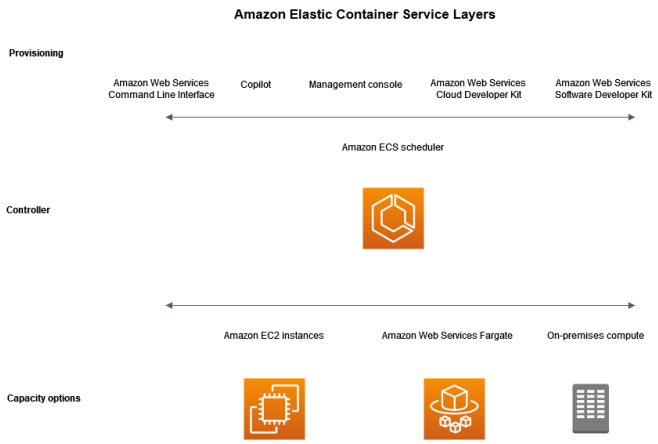
Amazon Elastic Container Service (Amazon ECS) is a fully managed container orchestration service that helps you easily deploy, manage, and scale containerized applications. As a fully managed service, Amazon ECS (ECS, n.d.) Comes with AWS configuration and operational best practices built-in. It is integrated with both AWS and third party tools, such as Amazon Elastic Container Registry and Docker. This integration makes it easier for teams to focus on building the applications, not the environment. You can run and scale your container workloads across AWS Regions in the cloud, and on premises, without the complexity of managing a control plane.

2 Amazon ECS terminology and components

There are three layers in Amazon ECS:

- Capacity The infrastructure where your containers run
- Controller Deploy and manage your applications that run on the containers
- Provisioning The tools that you can use to interface with the scheduler to deploy and manage your applications and containers

The following diagram shows the Amazon ECS layers.



3 Amazon ECS capacity

Amazon ECS capacity is the infrastructure where your containers run. The following is an overview of the capacity options:

3.1 Amazon EC2 instances in the AWS cloud

You choose the instance type, the number of instances, and manage the capacity.

3.2 Server less (AWS Fargate (Fargate)) in the AWS cloud

Fargate is a server less, payasyougo compute engine. With Fargate you don't need to manage servers, handle capacity planning, or isolate container workloads for security.

3.3 Onpremises virtual machines (VM) or servers

Amazon ECS Anywhere provides support for registering an external instance such as an on premises server or virtual machine (VM), to your Amazon ECS cluster.

The capacity can be located in any of the following AWS resources:

- Availability Zones
- Local Zones
- Wavelength Zones
- AWS Regions
- AWS Outposts
- Amazon ECS controller

The Amazon ECS scheduler is the software that manages your applications.

4 Amazon ECS provisioning

There are multiple options for provisioning Amazon ECS:

4.1 AWS Management Console

Provides a web interface that you can use to access your Amazon ECS resources.

4.2 AWS Command Line Interface (AWS CLI)

Provides commands for a broad set of AWS services, including Amazon ECS. It's supported on Windows, Mac, and Linux.

4.3 AWS SDKs

Provides language specific APIs and takes care of many of the connection details. These include calculating signatures, handling request retries, and error handling.

4.4 Copilot

Provides an open source tool for developers to build, release, and operate production ready containerized applications on Amazon ECS.

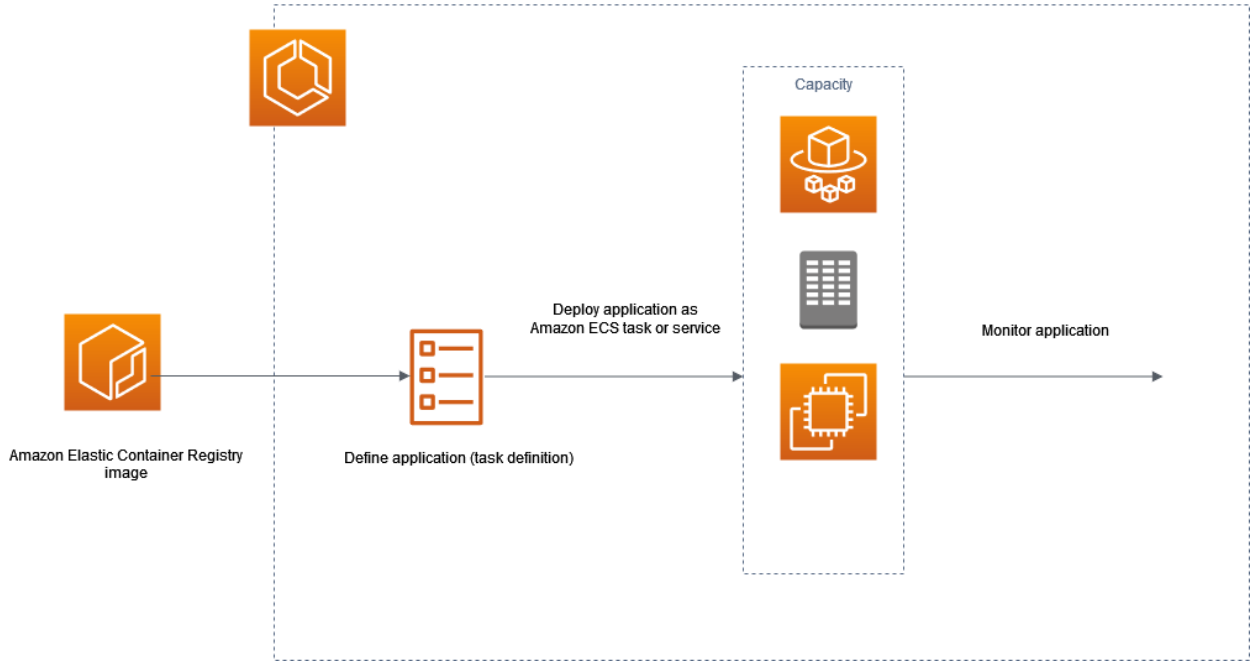
4.5 AWS CDK

Provides an open source software development framework that you can use to model and provision your cloud application resources using familiar programming languages. The AWS CDK provisions your resources in a safe, repeatable manner through AWS CloudFormation.

5 Application lifecycle

The following diagram shows the application lifecycle and how it works with the Amazon ECS components.

Amazon ECS Application Lifecycle



To deploy applications on Amazon ECS, your application components must be configured to run in containers. A container is a standardized unit of software development that holds everything that your software application requires to run. This includes relevant code, runtime, system tools, and system libraries. Containers are created from a read-only template that has called an image. Images are typically built from a Docker file. A Docker file is a plaintext file that specifies all of the components that are included in the container. After they're built, these images are stored in a registry such as Amazon ECR where they can be downloaded from.

After you create and store your image, you create an Amazon ECS task definition. A task definition is a blueprint for your application. It is a text file in JSON format that describes the parameters and one or more containers that form your application. For example, you can use it to specify the image and parameters for the operating system, which containers to use, which ports to open for your application, and what data volumes to use with the containers in the task. The specific parameters available for your task definition depend on the needs of your specific application.

After you define your task definition, you deploy it as either a service or a task on your cluster. A cluster is a logical grouping of tasks or services that runs on the capacity infrastructure that is registered to a cluster.

A task is the instantiation of a task definition within a cluster. You can run a standalone task, or you can run a task as part of a service. You can use an Amazon ECS service to run and maintain your desired number of tasks simultaneously in an Amazon ECS cluster. How it works is that, if any of your tasks fail or stop for any reason, the Amazon ECS service scheduler launches another instance based on your task definition. It does this to replace it and thereby maintain your desired number of tasks in the service.

The container agent runs on each container instance within an Amazon ECS cluster. The agent sends information about the current running tasks and resource utilization of your containers to Amazon ECS. It starts and stops tasks whenever it receives a request from Amazon ECS.

After you deploy the task or service, you can use any of the following tools to monitor your deployment and application:

- Cloud Watch

Amazon Elastic Kubernetes Service (Amazon EKS)

1 Amazon Elastic Kubernetes Service (Amazon EKS)

Amazon Elastic Kubernetes Service (EKS, n.d.) is a managed service that eliminates the need to install, operate, and maintain your own Kubernetes control plane on Amazon Web Services (AWS). Kubernetes is an open source system that automates the management, scaling, and deployment of containerized applications.

2 Features of Amazon EKS

The following are key features of Amazon EKS:

2.1 Secure networking and authentication

Amazon EKS integrates your Kubernetes workloads with AWS networking and security services. It also integrates with AWS Identity and Access Management (IAM) to provide authentication for your Kubernetes clusters.

2.2 Easy cluster scaling

Amazon EKS enables you to scale your Kubernetes clusters up and down easily based on the demand of your workloads. Amazon EKS supports horizontal Pod auto scaling based on CPU or custom metrics, and cluster auto scaling based on the demand of the entire workload.

2.3 Managed Kubernetes experience

You can make changes to your Kubernetes clusters using eksctl, AWS Management Console, AWS Command Line Interface (AWS CLI), the API, kubectl, and Terraform.

2.4 High availability

Amazon EKS provides high availability for your control plane across multiple Availability Zones.

2.5 Integration with AWS services

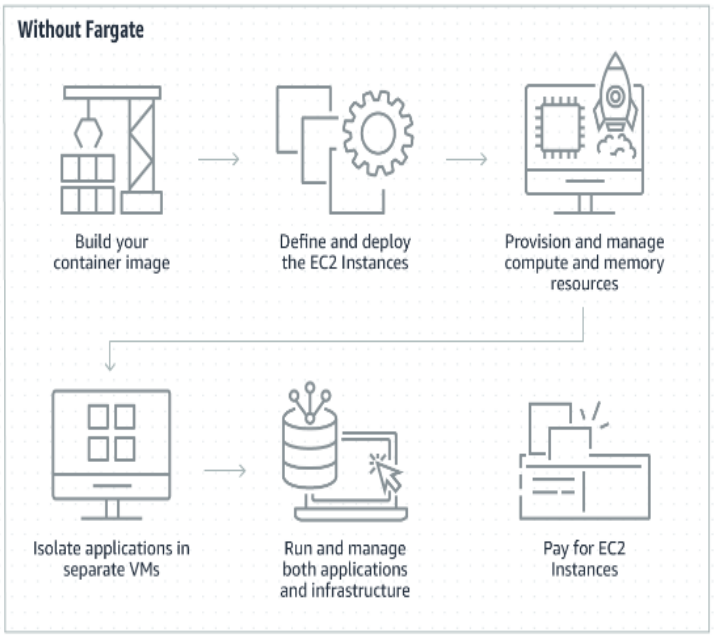
Amazon EKS integrates with other AWS services, providing a comprehensive platform for deploying and managing your containerized applications. You can also more easily troubleshoot your Kubernetes workloads with various observability tools.

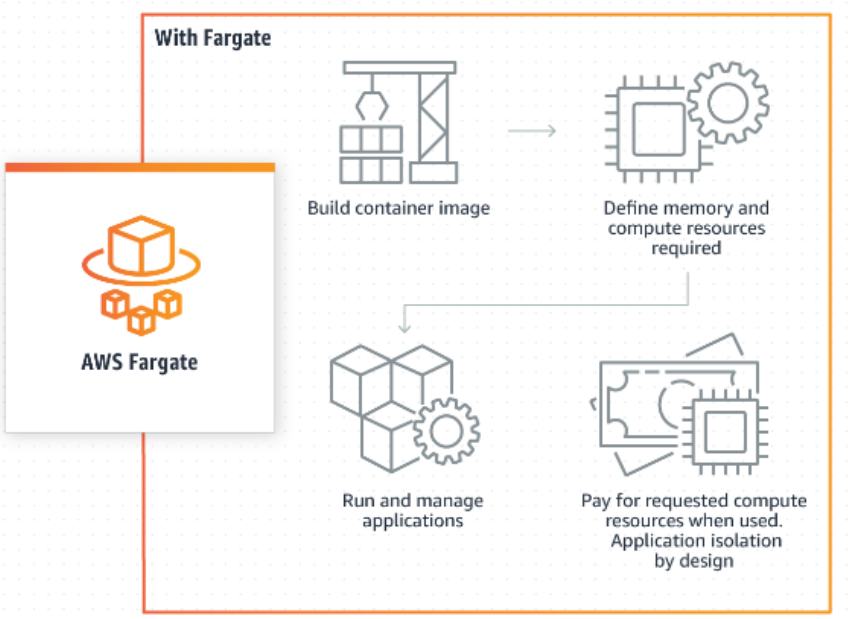
AWS Fargate

1 AWS Fargate

AWS Fargate (Fargate, n.d.) is a technology that you can use with Amazon ECS to run containers without having to manage servers or clusters of Amazon EC2 instances. With Fargate, you no longer have to provision, configure, or scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing.

When you run your Amazon ECS tasks and services with the Fargate launch type or a Fargate capacity provider, you package your application in containers, specify the Operating System, CPU and memory requirements, define networking and IAM policies, and launch the application. Each Fargate task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.





2 Components

2.1 Clusters

An Amazon ECS cluster is a logical grouping of tasks or services. You can use clusters to isolate your applications. When your tasks are run on Fargate, your cluster resources are also managed by Fargate.

2.2 Task definitions

A task definition is a text file that describes one or more containers that form your application. It's in JSON format. You can use it to describe up to a maximum of ten containers. The task definition functions as a blueprint for your application. It specifies the various parameters for your application. For example, you can use it to specify parameters for the operating system, which containers to use, which ports to open for your application, and what data volumes to use with the containers in the task. The specific parameters available for your task definition depend on the needs of your specific application.

Your entire application stack doesn't need to be on a single task definition. In fact, we recommend spanning your application across multiple task definitions. You can do this by

combining related containers into their own task definitions, each representing a single component.

2.3 Tasks

A task is the instantiation of a task definition within a cluster. After you create a task definition for your application within Amazon ECS, you can specify the number of tasks to run on your cluster. You can run a standalone task, or you can run a task as part of a service.

2.4 Services

You can use an Amazon ECS service to run and maintain your desired number of tasks simultaneously in an Amazon ECS cluster. How it works is that, if any of your tasks fail or stop for any reason, the Amazon ECS service scheduler launches another instance based on your task definition. It does this to replace it and thereby maintain your desired number of tasks in the service.

AWS Elastic Container Registry (ECR)

1 AWS Elastic Container Registry (ECR)

Amazon Elastic Container Registry (Amazon (ECR, n.d.)) is an AWS managed container image registry service that is secure, scalable, and reliable. Amazon ECR supports private repositories with resource based permissions using AWS IAM. This is so that specified users or Amazon EC2 instances can access your container repositories and images. You can use your preferred CLI to push, pull, and manage Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts.

2 Components of Amazon ECR

Amazon ECR contains the following components:

2.1 Registry

An Amazon ECR private registry is provided to each AWS account; you can create one or more repositories in your registry and store Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts in them. For more information, see Amazon ECR private registry.

2.2 Authorization token

Your client must authenticate to an Amazon ECR private registry as an AWS user before it can push and pull images. For more information, see Private registry authentication.

2.3 Repository

An Amazon ECR repository contains your Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts.

2.4 Repository policy

You can control access to your repositories and the contents within them with repository policies.

2.5 Image

You can push and pull container images to your repositories. You can use these images locally on your development system, or you can use them in Amazon ECS task definitions and Amazon EKS pod specifications.

3 Features of Amazon ECR

Amazon ECR provides the following features:

- Lifecycle policies help with managing the lifecycle of the images in your repositories. You define rules that result in the cleaning up of unused images. You can test rules before applying them to your repository.
- Image scanning helps in identifying software vulnerabilities in your container images. Each repository can be configured to scan on push. This ensures that each new image pushed to the repository is scanned. You can then retrieve the results of the image scan.
- Cross-Region and cross account replication makes it easier for you to have your images where you need them. This is configured as a registry setting and is on a per Region basis.
- Pull through cache rules provide a way to cache repositories in an upstream registry in your private Amazon ECR registry. Using a pull through cache rule, Amazon ECR will periodically reach out to the upstream registry to ensure the cached image in your Amazon ECR private registry is up to date.

AWS Batch

1 AWS Batch

AWS Batch (batch, n.d.) helps you to run batch computing workloads on the AWS Cloud. Batch computing is a common way for developers, scientists, and engineers to access large amounts of compute resources. AWS Batch removes the undifferentiated heavy lifting of configuring and managing the required infrastructure, similar to traditional batch computing software. This service can efficiently provision resources in response to jobs submitted in order to eliminate capacity constraints, reduce compute costs, and deliver results quickly.

As a fully managed service, AWS Batch helps you to run batch computing workloads of any scale. AWS Batch automatically provisions compute resources and optimizes the workload distribution based on the quantity and scale of the workloads. With AWS Batch, there's no need to install or manage batch computing software, so you can focus your time on analyzing results and solving problems.

2 Components of AWS Batch

AWS Batch simplifies running batch jobs across multiple Availability Zones within a Region. You can create AWS Batch compute environments within a new or existing VPC. After a compute environment is up and associated with a job queue, you can define job definitions that specify which Docker container images to run your jobs. Container images are stored in and pulled from container registries, which may exist within or outside of your AWS infrastructure.

2.1 Jobs

A unit of work (such as a shell script, a Linux executable, or a Docker container image) that you submit to AWS Batch. It has a name, and runs as a containerized application on AWS Fargate or Amazon EC2 resources in your compute environment, using parameters that you specify in a job definition. Jobs can reference other jobs by name or by ID, and can be dependent on the successful completion of other jobs.

2.2 Job Definitions

A job definition specifies how jobs are to be run. You can think of a job definition as a blueprint for the resources in your job. You can supply your job with an IAM role to provide access to other AWS resources. You also specify both memory and CPU requirements. The job definition can also control container properties, environment variables, and mount points for persistent storage. Many of the specifications in a job definition can be overridden by specifying new values when submitting individual Jobs.

2.3 Job Queues

When you submit an AWS Batch job, you submit it to a particular job queue, where the job resides until it's scheduled onto a compute environment. You associate one or more compute environments with a job queue. You can also assign priority values for these compute environments and even across job queues themselves. For example, you can have a high priority queue that you submit time sensitive jobs to, and a low priority queue for jobs that can run anytime when compute resources are cheaper.

2.4 Compute Environment

A compute environment is a set of managed or unmanaged compute resources that are used to run jobs. With managed compute environments, you can specify desired compute type (Fargate or EC2) at several levels of detail. You can set up compute environments that use a particular type of EC2 instance, a particular model such as c5.2xlarge or m5.10xlarge. Or, you can choose only to specify that you want to use the newest instance types. You can also specify the minimum, desired, and maximum number of vCPUs for the environment, along with the amount that you are willing to pay for a Spot Instance as a percentage of the On Demand Instance price and a target set of VPC subnets. AWS Batch efficiently launches, manages, and terminates compute types as needed. You can also manage your own compute environments. As such, you're responsible for setting up and scaling the instances in an Amazon ECS cluster that AWS Batch creates for you.

Amazon Lightsail

1 Amazon Lightsail

Amazon Lightsail is a simplified and user friendly service provided by Amazon Web Services (AWS) that allows users to easily deploy and manage cloud resources for their web applications or websites.

Here is a detailed overview of Light sail's features and benefits:

1.1 Ease of Use:

Description: Light sail is designed to be user-friendly, providing an easy and quick way to set up cloud resources. It is well-suited for users who may not have extensive experience with cloud services.

1.2 Comprehensive Services:

Description: Light sail offers a range of services to cater to various application needs. These services include instances (virtual servers), containers, databases, storage, and more.

1.3 Blueprints for Quick Deployment:

Description: Light sail simplifies the deployment process by offering preconfigured blueprints. These blueprints are templates for common applications like WordPress, Prestashop, or LAMP (Linux, Apache, MySQL, PHP/Python/Perl), making it easy to set up a functioning website or application quickly.

1.4 Use Cases:

Description: Lightsail supports a variety of use cases, such as hosting static content, connecting content to a global audience using a content delivery network (CDN), or setting up Windows Business servers.

1.5 Configuration Guidance:

Description: The Lightsail console guides users through the configuration process, providing systematic instructions. In many cases, components are preconfigured, reducing the complexity of the setup.

1.6 Key Features:

- **Instance Management:** Lightsail allows users to easily create, manage, and scale virtual private servers (instances) without dealing with the complexities of traditional cloud services.
- **Integrated Databases:** Lightsail provides managed database services that can be easily integrated into applications.
- **Scalability:** Users can easily scale their applications vertically by adjusting instance sizes or horizontally by adding more instances.
- **Snapshots and Backups:** Lightsail offers features for creating snapshots and automated backups for data protection.

1.7 Cost Predictability:

Description: Lightsail provides predictable and transparent pricing, making it easier for users to understand and control their costs.

1.8 Global Reach:

Description: Users can leverage Lightsail to deploy applications globally, reaching a wider audience through the use of AWS's global infrastructure.

In summary, Amazon Lightsail is a straightforward and accessible solution for users who want to quickly deploy web applications or websites in the cloud without the need for in depth cloud expertise. The service offers a range of features, preconfigured templates, and a guided console to simplify the entire process.

➢ References

- *Amazon EC2 Auto Scalling*. (n.d.). Retrieved from aws.amazon.com: https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html
- *AWS*. (n.d.). Retrieved from aws.amazon.com: https://aws.amazon.com/what-is/compute/
- *AWS Compute Optimizer*. (n.d.). Retrieved from aws.amazon.com: https://docs.aws.amazon.com/compute-optimizer/latest/ug/what-is-compute-optimizer.html
- *AWS Compute*. (n.d.). Retrieved from aws.Amazon.com: https://aws.amazon.com/what-is/compute/
- *AWS EC2*. (n.d.). Retrieved from aws.amazon.com: https://aws.amazon.com/pm/ec2/
- *AWS EC2 Instances*. (n.d.). Retrieved from aws.amazon.com: https://aws.amazon.com/ec2/instance-types/
- *batch*. (n.d.). Retrieved from aws.amazon.com: https://docs.aws.amazon.com/batch/latest/userguide/what-is-batch.html
- *EC2 Pricing*. (n.d.). Retrieved from aws.amazon.com: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html#ec2-pricing
- *ECR*. (n.d.). Retrieved from aws.amazon.com: https://docs.aws.amazon.com/AmazonECR/latest/userguide/what-is-ecr.html
- *ECS*. (n.d.). Retrieved from aws.amazon.com: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html

- *EKS*. (n.d.). Retrieved from aws.amazon.com: https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html
- *ELB*. (n.d.). Retrieved from aws.amazon.com: https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/what-is-load-balancing.html
- *Fargate*. (n.d.). Retrieved from aws.amazon.com: https://docs.aws.amazon.com/AmazonECS/latest/userguide/what-is-fargate.html
- *Lambda*. (n.d.). Retrieved from aws.amazon.com: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
- *Load Balancing*. (n.d.). Retrieved from www.nginx.com: https://www.nginx.com/resources/glossary/load-balancing/
- *MM Differences*. (n.d.). Retrieved from aws.amazon.com: https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/
- *MS1*. (n.d.). Retrieved from aws.amazon.com: https://aws.amazon.com/sqs/
- *MS2*. (n.d.). Retrieved from www.rabbitmq.com: https://www.rabbitmq.com/documentation.html
- *Serverless Computing*. (n.d.). Retrieved from www.alibabacloud.com: https://www.alibabacloud.com/knowledge/what-is-serverless
- *SNS Amazon* . (n.d.). Retrieved from aws.amazon.com: https://docs.aws.amazon.com/sns/latest/dg/welcome.html
- *SQS Amazon*. (n.d.). Retrieved from aws.amazon.com: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html