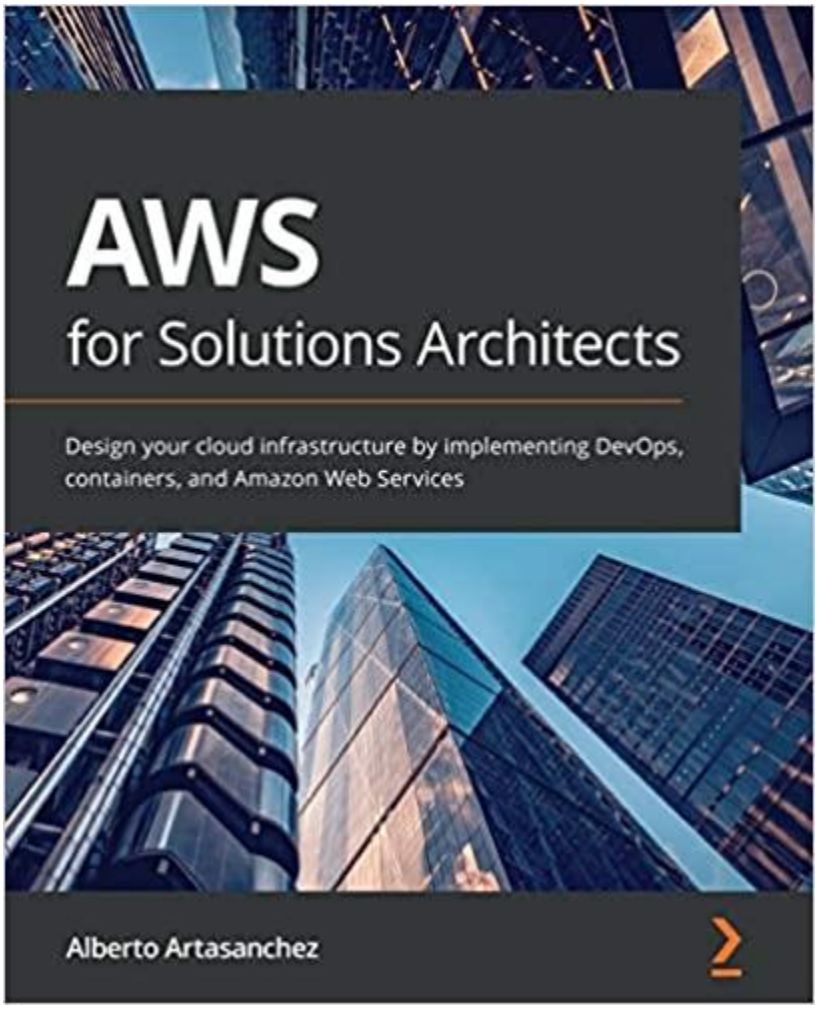
Book review **– AWS for Solution Architects** by **Alberto Artasanchez** reviewed by **Umair Hoodbhoy, Arunabh Sahai**

https://www.amazon.com/gp/product/1789539234/ref=ewc_pr_img_2?smid=ATVPDKIKX0DER&psc=1

Summary: A good book with many use cases, design patterns, AWS services and comparisons.

Good book to read and understand the complex AWS services and know why/when/where/what/how to use them TOGETHER Effectively for your use cases/workloads!

It has new AWS services, e.g. Local Zones for latency sensitive workloads. It includes many good contents, e.g. Kubernetes/containers workloads, Athena, Data Analytics/Lakes, How to get certified, Migration services, Quantum Ledger DB, Microservices, 2 Pizza rule, etc.



Chap 1 – good tips for ALL AWS certifications and resources, e.g. Acloud.guru, Whizlabs, Jon Bonso's Udemy courses. Suggest to add other good course instructors like Stephane Maarek, Neal Davis, Adrian Cantrill, etc.

Chap 2 – Very updated, e.g. new Local Zones (LZs) (introduced in Q4, 2020) can be used for latency sensitive workloads, e.g. gaming, streaming, etc.... I like the Migration tools, e.g. AWS Migration Hub, Application Discovery Service, design pattern library, CloudEndure Migration, Data Migration, etc.…Digital transformation examples, tips and pitfalls are great!

Amazon is continuously enhancing its data centers to provide the latest technology. Amazon's data centers have a high degree of redundancy. Amazon uses highly reliable hardware, but hardware is not foolproof. Occasionally, a failure can happen that interferes with the availability of resources in each data center. If all instances are hosted in only one data center, the possibility exists that a failure can occur with the whole data center and then none of your resources will be available. The following diagram illustrates how Regions, AZs, and LZs are organized:

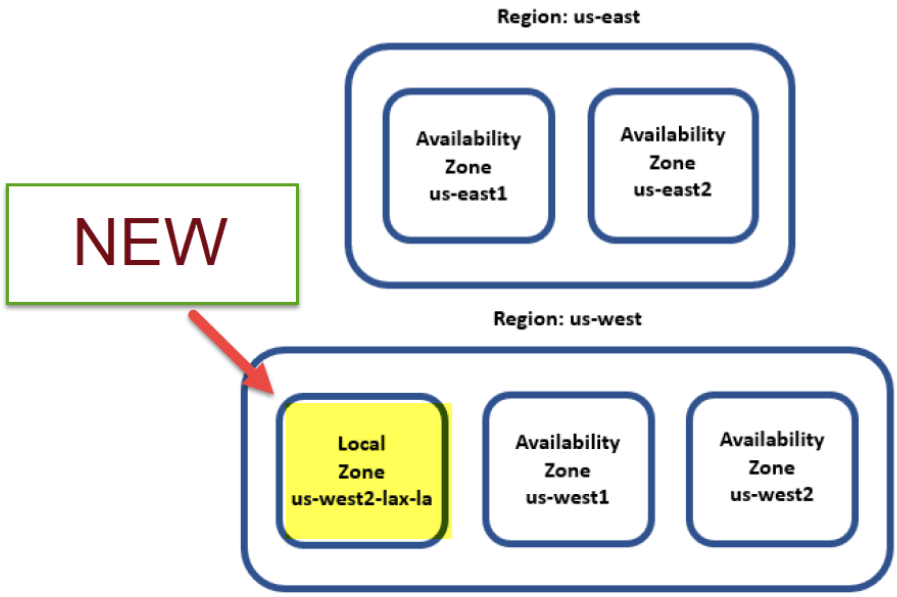


Figure 2.1 – Example of Regions, AZs, and LZs

Notice that Regions can contain both **AZs** as well as **LZs**. It is important to highlight

Local Zones

LZs are new components in the AWS infrastructure family.

LZs place select services close to end users and allow them to create AWS applications that can deliver single-digit millisecond responses.

You can think of them as a subset of an AZ. They offer some but not all of the services an AZ provides. A VPC can be extended in any AWS Region into an LZ if a subnet is created and assigned to the LZ. Subnets in an LZ operate like any other subnets created in AWS.

AWS allows the creation of database instances in an LZ. LZs provides connections to the internet and they can use AWS Direct Connect.

The naming convention for LZs is to use the AWS Region followed by a location identifier, for example: us-west-2-lax-2a.

Section 2: AWS Service Offerings and Use Cases

• Chapter 3, Storage in AWS – Choosing the Right Tool for the Job

Cover details in EBS, EFS, S3, etc....

E.g. Good table below to show the different characteristics of different S3s:

In the following, we have a summary of the profile of each storage class and how they compare to each other:

Features	S3 Standard	S3 Intelligent Tiering	S3 Standard-IA	S3 One Zone-IA	S3 Glacier	S3 Glacier Deep Archive
Durability	11 9's	11 9's	11 9's	11 9's	11 9's	11 9's
Availability	99.9	99.9	99.9	99.5%	99.9	99.9
Availability Zones	>=3	>=3	>=3	1	>=3	>=3
Minimum capacity charge per object	N/A	N/A	128KB	128KB	40KB	40KB
Access Latency	milliseconds	milliseconds	milliseconds	milliseconds	1 min - 12 hours	hours
Data Retrieval Fee	none	per GB data retrieved	per GB data retrieved	per GB data retrieved	per GB data retrieved	per GB data retrieved
Type of Storage	Object	Object	Object	Object	Object	Object
SSL Support	Yes	Yes	Yes	Yes	Yes	Yes
Availability SLA	99.9%	99%	99%	99%	99.9%	99%

Figure 3.1 – Summary of storage class features

Also, good table to show the main 3 storage types and use cases: S3, EBS and EFS:

The following table should also help you to decide what service is best for your use case:

	Performance	Cost	Availability	Storage Limit	File Size Limit
Amazon S3	By default, it supports 100 requests per second and scalable to 300	Average of $0.0235 per GB/month	99.99 % availability	No limit on the number of objects	5TB object limit
Amazon EBS	Provisioned IOPS can deliver 4000 operations per second	Anywhere from $0.025 to $0.100 per GB/month	99.99 % availability	Maximum storage size of 16 TB	File size of up to 16 TB
Amazon EFS	Capable of up to 7000 operations per second	From $0.30 to $0.36 per GB/month	No SLA in force	No limit on system size	File size of up to 52 TB

Figure 3.2 – Choosing the service based on your use case

Hopefully, this section was useful to help you to discern when to use the various storage services that AWS offers. Let's continue learning more about the fundamental service that is Amazon S3.

- Chapter 4, Harnessing the Power of Cloud Computing

Explain use cases for On Premise, IaaS, PaaS and SaaS.

EC2: types, AMI, best practices, backup, snapshots and recovery.

The following image shows a summary of the current EC2 family instance types and their main characteristics:

	General Purpose		Compute Optimized	Memory Purpose		Accelerate the Computing	Storage Optimized		
Type	T3, A1	MS	CS	RS	XI	P3, G3, F1	H1	I3	D2, I3
Description	Burstable Changing workloads	Balanced consistent workloads	High ratio compute to memory	In-memory DBs	In-memory apps	Graphic processing AI/ML	Balanced of compute and memory HDD backed	Balanced of compute and memory HDD backed	Highest disk ratio

Figure 4.2 – Instance type families and their characteristics

It is very likely that Amazon will continue to enhance its EC2 service by offering new instance types and enhancing the current ones.

> **Important note**
>
> To find the most up-to-date information on Amazon EC2 instance types, visit this link: `https://aws.amazon.com/ec2/instance-types/`.

- Chapter 5, Selecting the Right Database Service

Introducing different types of databases and how to choose the right tool for your workload/job.

Classifying databases by content

Even with all the current offerings, most databases fall into one of these categories:

- Relational databases
- Key-value databases
- Document databases
- Wide column storage databases
- Graph databases

Classifying databases by usage

Broadly speaking, two operations can be performed with a database:

- Ingest data (or write data into the database)
- Retrieve data (or read data from the database)

Also, explain Amazon Kendra for Enterprise search, In-memory DB, IMDBs, Neptune, TSDB, Time stream, Ledger DB (LDB) (and also Quantum LDB) for Crypto!

Ledger databases

A **ledger database** (**LDB**) is database that delivers a cryptographically verifiable, immutable, and transparent transaction log orchestrated by a central authority:

- **LDB immutability**: Imagine you deposit $1,000 in your bank. You see on your phone that the deposit was carried out, and it now shows a balance of $1,000. Then imagine you check it again tomorrow and now it says $500. You would not be too pleased, would you? The bank needs to ensure that the transaction was immutable, and no one can change it after the fact. In other words, only inserts are allowed, and updates cannot be performed. This assures that transactions cannot be changed once they are persisted.
- **LDB transparency**: In this context, transparency refers to the ability to be able to track changes to the data over time. The LDB should be able to keep an audit log. This audit log at a minimum should include who changed data, when the data was changed, and what the value of the data was before it was changed.
- **LDB cryptographic verifiability**: How can we ensure that our transaction will be immutable? Even though the database might not support update operations, what's stopping someone from using a backdoor and updating the record? If we use cryptography when the transaction is recorded, the entire transaction data is hashed. In simple terms, the string of data that forms the transaction is whittled down into a smaller string of characters that is unique. Whenever the transaction is hashed, it needs to match that string. In the ledger, the hash comprises the transaction data, and it also appends the hash of the previous transaction. By doing this, we ensure that the entire chain of transactions is valid. If someone tried to enter another transaction in between, it would invalidate the hash and it would be detected that the foreign transaction was added via an unauthorized method.

The prototypical use case for LDBs are bank account transactions. If we use an LDB for this use case, the ledger records all credits and debits related to the bank account. It can then be followed from a point in history, allowing us to calculate the current account balance. With immutability and cryptographic verification, we are assured that the ledger cannot be deleted or modified. With other methods, such as RDBMSes, all transactions could potentially be changed or erased.

Quantum Ledger

Amazon **Quantum Ledger** (**QLDB**), like many other Amazon offerings, is a fully managed service. It delivers an immutable, transparent, and cryptographically verifiable ledger managed by a central trusted authority. Amazon QLDB keeps track of application value changes and manages a comprehensive and verifiable log of any change to the database.

Historically, ledgers have been used to maintain a record of financial activities. Ledgers can keep track of the history of transactions that need high availability and reliability. Some examples that need this level of reliability are as follows:

- Financial credits and debits
- Verifying the data lineage of a financial transaction
- Tracking the location history of inventory in a supply chain network

Amazon QLDB offers a variety of blockchain services such as anonymous data sharing and smart contracts while still using a centrally trusted transaction log.

• Chapter 6, Amazon Athena – Combining the Simplicity of Files with the Power of SQL

Athena is a very powerful data analytics tool/platform and good to see a chapter on it with deep details, e.g. Athena Federated Query, workgroups, etc...

In this chapter, once we have introduced Amazon Athena, we will learn more about Amazon Athena workgroups. We will also review Amazon Athena's APIs toward the end of the chapter.

In this chapter, we will cover the followings topics:

- Introduction to Amazon Athena
- Deep diving into Amazon Athena
- Understanding how Amazon Athena works
- Using Amazon Athena Federated Query
- Learning about Amazon Athena workgroups
- Reviewing Amazon Athena's APIs
- Understanding when Amazon Athena is an appropriate solution

Introduction to Amazon Athena

Water, water everywhere, and not a drop to drink… This may be the feeling you get in today's enterprise environments. We are producing data at an exponential rate, but it is sometimes difficult to find a way to analyze this data and gain insights from it. Some of the data that we are generating at a prodigious rate is of the following types:

- Application logging
- Clickstream data
- Surveillance video
- Smart and IoT devices
- Commercial transactions

Often, this data is captured without analysis or is at least not analyzed to the fullest extent. Analyzing this data properly can translate into the following:

- Increased sales
- Cross-selling opportunities
- Lower operational and hardware infrastructure costs
- Avoiding downtime and errors before they occur
- Serving customer bases more efficiently

• Chapter 7, AWS Glue – Extracting, Transforming, and Loading Data the Simple Way

Glue is the ETL data service and a cornerstone service.

- Crawling with AWS Glue crawlers
- AWS Glue best practices

Why is AWS Glue a cornerstone service?

Traditionally, the most important resources that a business has are its human and financial capital. However, in the last few decades, more and more businesses have come to the realization that another resource may be just as important. It's only been in the last few years that they have come to understand that another commodity that is just as valuable, if not more valuable, is its data capital.

Data has taken a special place at the center of some of today's most successful enterprises. For this reason, business leaders have concluded that in order to survive in today's business climate they must collect, process, transform, distill, and safeguard their data like they do their other traditional business capital.

Businesses that have made this realization have been able to transform and reinvent their value propositions. Data-driven businesses will be able to increase their profitability and efficiency, reduce costs, deliver new products and services, better serve their customers, stay in compliance with regulatory requirements, and ultimately, thrive. Unfortunately, as we have seen many examples of in recent years, companies that don't make this transition will not be able to survive.

An important part of being a data-driven enterprise is the ability to ingest, process, transform, and analyze this data. AWS Glue is a foundational service at the heart of the AWS offering and we will dedicate the rest of this chapter to this service.

With the introduction of Apache Spark, enterprises can process petabytes' worth of data daily. Being able to process this amount of data opens the door to making data an enterprise's most valuable asset. Processing this data at this scale allows enterprises to literally create new industries and markets. Some examples of business activities that have benefited greatly from this massive data processing are as follows:

- Personalized marketing
- Drug discovery
- Anomaly detection (such as fraud detection)
- Real-time log and clickstream processing

AWS has created a service that leverages Apache Spark and takes it to the next level. The name of that service is AWS Glue.

To achieve its intended purpose as an **Extract, Transform, and Load** (**ETL**) service, AWS Glue has a series of components. These are as follows:

1. The AWS Glue console
2. The AWS Glue Data Catalog
3. AWS Glue classifiers
4. AWS Glue crawlers
5. AWS Glue code generators

Let's pay a visit to the major ones.

Putting it all together

Now that we have learned about all the major components in AWS Glue, let's look at how all the pieces fit together. The following diagram illustrates this:

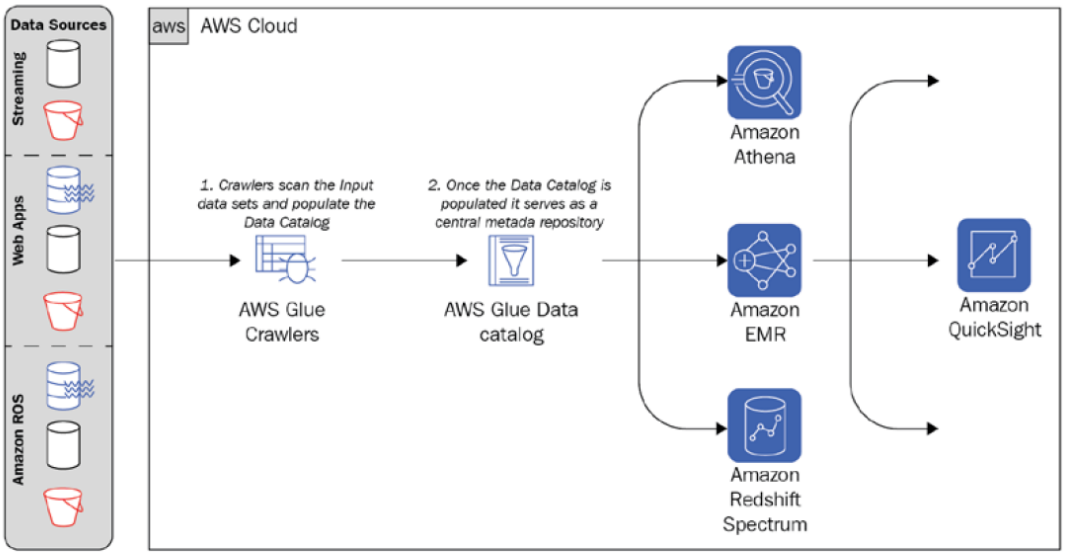


Figure 7.1 – AWS Glue typical workflow steps

- Chapter 8, Best Practices for Application Security, Identity, and Compliance

Good details on IAM, Control Tower, Guard Duty, Shield, WAF, Cloud Directory, Inspector, Macie (AI/ML based detection), Artifact reports, Certificate Manager, Cloud HSM, Directory, KMS, Secret Managers, SSO, Security hub, …etc.

These are the most important and relevant concepts when working with the AWS Organizations service:

- **Organization**: The overarching owner that will control all AWS accounts.
- **Root account**: The owner account for all other AWS accounts. Only one root account can exist across the organization. The root account needs to be created when the organization is created.
- **Organizational unit**: A grouping of underlying AWS accounts and/or other organizational units. An organizational unit can be the parent of other organizational units and so on. This enables the potential creation of a hierarchy of organizational units that resembles a family tree. See the following diagram for more clarity.
- **AWS account**: A traditional AWS account that manages AWS resources and services. AWS accounts reside under an organizational unit or under the root account.
- **Service Control Policy (SCP)**: A service control policy specifies the services and permissions for users and roles. An SCP can be associated with an AWS account or with an organizational unit.

The following figure illustrates how these components interact with each other:

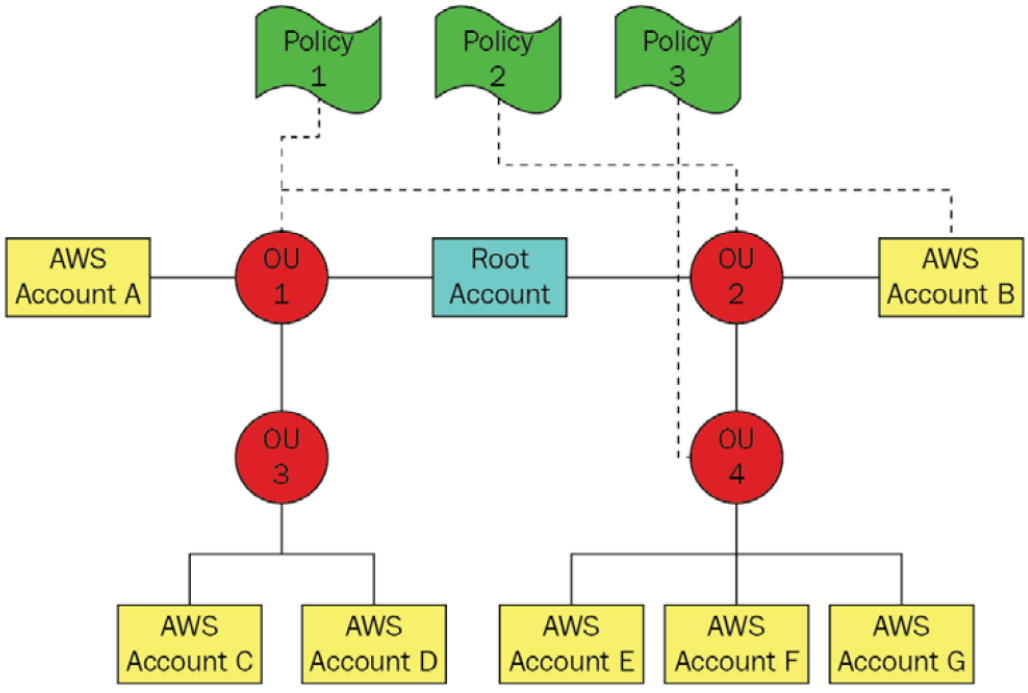


Figure 8.2 – Sample organizational unit hierarchy

Section 3: Applying Architectural Patterns and Reference Architectures

In this section, you'll understand the AWS cloud architectural patterns and how they can be applied to a variety of use cases.

This part of the book comprises the following chapters:

- Chapter 9, Serverless and Container Patterns

9

Serverless and Container Patterns

In this chapter and this section of the book, we will learn about a variety of patterns that are commonly used by many of the top technology companies in the world, including Netflix, Microsoft, Amazon, Uber, eBay, and PayPal. These companies have survived and thrived by adopting cloud technologies and the design patterns that are popular on the cloud. It is hard to imagine how these companies could exist in their present form if the capabilities delivered by the cloud did not exist. In addition, the patterns, services, and tools presented in this chapter make the cloud that much more powerful.

In this chapter, we will first present the concept of containerization. We will then move on to learn about two popular tools – Docker and Kubernetes. Toward the end of the chapter, we will also cover AWS Batch and its components, along with its best practices.

In this chapter, we will cover the following topics:

- Understanding containerization
- Virtual machines and virtualization
- Containers versus VMs
- Learning about Docker
- Learning about Kubernetes

- Learning about AWS Fargate
- Learning about AWS Batch

Let's get started.

PrivateLink support

Amazon EKS supports **PrivateLink** as a method to provide access to Kubernetes masters and the Amazon EKS service. With PrivateLink, the Kubernetes masters and Amazon EKS service API endpoint display as an **Elastic Network Interface** (**ENI**), including a private IP address in the Amazon VPC. This provides access to the Kubernetes masters and the Amazon EKS service from inside the Amazon VPC, without needing to have public IP addresses or have traffic go through the internet.

Automatic version upgrades

Amazon EKS manages patches and version updates for your Kubernetes clusters. Amazon EKS automatically applies Kubernetes patches to your cluster, and you can also granularly control things when and if certain clusters are automatically upgraded to the latest Kubernetes minor version.

Community tools support

Amazon EKS can integrate with many Kubernetes community tools and supports a variety of Kubernetes add-ons. One of these tools is KubeDNS. **KubeDNS** allows users to provision a DNS service for a cluster. Similar to how AWS offers console access and a command-line interface, Kubernetes has a web-based UI and a command-line interface tool called `kubectl`. Both of these tools offer the ability to interface with Kubernetes and provide cluster management.

Logging

Amazon EKS is integrated with Amazon CloudWatch Logs and AWS CloudTrail to support reliable visibility and auditability of the cluster and traffic activity. CloudWatch Logs can be used to view logs from the Kubernetes masters, and CloudTrail can be used to inspect API activity logs to the Amazon EKS service endpoints.

This section completes our coverage of Kubernetes. We now move on to a service offered by AWS that can also be used to manage massive workloads. AWS Batch is geared more toward workloads that can run in batches.

Amazon ECS versus Amazon EKS

So, at this point, a question that you are probably asking yourself is when should you choose Amazon ECS and when is Amazon EKS the right solution.

- Chapter 10, Microservices and Event-Driven Architectures

Many good tips, use cases and best practices hints.

In this chapter, we will do a deep dive into the ins and outs of this cornerstone pattern. Specifically, we will cover the following topics:

- Understanding microservices
- Microservice architecture patterns
- Benefits of an event-driven architecture
- Disadvantages of an event-driven architecture
- Learning about the microkernel architecture
- Reviewing microservices best practices

Let's get started.

Learning about microkernel architecture

The **microkernel architecture** pattern (also sometimes called a plugin architecture pattern) is another useful pattern to implement microservices. It is important to note that it is also used to build other software and not just microservices. The resulting application created using microkernel architecture can be considered a product. A **product** is a fully standalone application that can be distributed and implemented in a simple manner without much configuration or extra scripting. The most famous example of an application or product using this pattern is the Eclipse IDE. However, many enterprises also use this pattern to develop external and internal applications.

Another important feature of the microkernel architecture and the reason it is named as such is that the kernel can be extended by installing plugins in the kernel. The kernel provides the core functionality and the plugins allow users to extend the behavior in the manner they desire (assuming a plugin exists with the functionality they want to include). These plugins provide extensibility while implementing separation of concerns and ensuring that bugs in the kernel stay in the kernel and bugs in the plugins stay in the plugins.

The basic kernel provides the core functionality while enabling extensibility, flexibility, and the isolation of features. The plugins provide additional functionality and close the extensibility loop.

Now that we have learned about some of the most common microservice architectures, let's learn about best practices and when we implement them.

Microservices best practices

As with any technology, the devil is in the details. It is certainly possible to create bad microservices. Let's delve into how some common pitfalls can be avoided and some recommended best practices.

Best practice #1: Decide whether microservices are the right tool

The world's leading technology companies, such as eBay, Facebook, Amazon, Microsoft, Twitter, and PayPal are all heavy users of microservices architecture and rely on it for much of their development. However, it's not a panacea. As technologists, once we get a hammer, everything looks like a nail. Make sure that your particular use case is best suited for this architecture. If it's hard to break down your application into functional domains, a microservice architecture might not be the best choice.

To learn more about these and other monitoring services in AWS, you can visit:

`https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring_ec2.html`

Best practice #16: Two pizzas should be enough to feed your team

This is a rule popularized by Jeff Bezos. He famously only invites enough people to meetings so that two large pizzas can feed the attendees. Bezos popularized the *two pizza* rule for meetings and project teams to encourage a decentralized, creative working environment and to keep the start-up spirit alive and well.

This rule's goal is to avoid groupthink. *Groupthink* is a phenomenon that occurs when you have large groups and people start going with the consensus instead of feeling comfortable pushing back at what they think are bad ideas. In some ways, it is human nature to be more hesitant to disagree in large groups.

It is not uncommon for members of the group that are lower in the corporate hierarchy to be intimidated by authority figures such as their boss and people with a bigger title. By keeping groups small and encouraging dialog, some of this hesitancy may be overcome and better ideas may be generated.

Bezos' idea to keep meetings and teams small to foster collaboration and productivity can be backed up by science. During his 50 years of studies and research of teams, J. Richard Hackman concluded that four to six is the optimal number of team members for many projects and that teams should never be larger than 10.

According to Hackman, communication issues *"grow exponentially as team size increases."* Perhaps counterintuitively, the larger a team is the more time will be used to communicate, reducing the time that can be used productively to achieve goals.

In the context of microservice development, the *two pizza rule* is also applicable. You don't want your microservice development and deployment teams to be much bigger than a dozen people or so. If you need more staff, you are probably better off splitting the microservice domains so that you can have two teams creating two microservices rather than one huge team creating an incredibly big and complex microservice.

Obviously, there is no hard rule about exactly how many people is too many people, and in some cases, 10 people may be a good number, while in other cases 14 may be more suitable. But at some point, the number becomes too big and unmanageable. For example, having a 100-person monolithic team with no hierarchy or natural division in it most likely would be too unmanageable.

Microservices in AWS

Now that we understand what DDD and microservices are and how important they are in achieving a well-architected framework, let's see how we can use the plethora of AWS services to implement domain-driven designs and microservices. First, let's get this out of the way. Amazon offers a variety of ways to run container services and enable microservices architectures. AWS also offers an array of services that can complement these containers in the creation of microservices. The orchestration services offered are listed in the following screenshot:

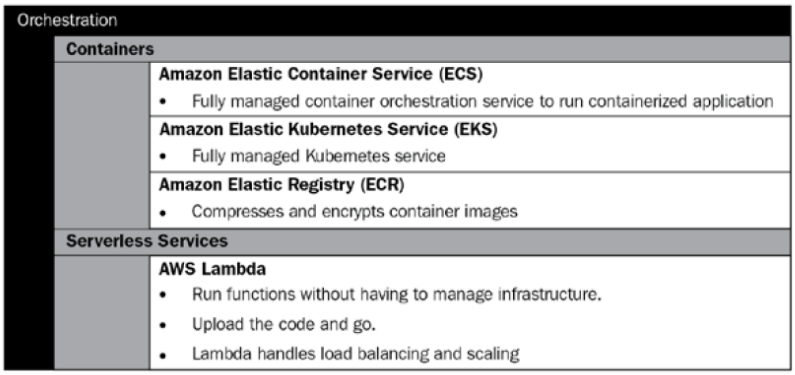


Figure 11.2 – Microservice container services in AWS

The compute services that can used to create microservices are as follows:

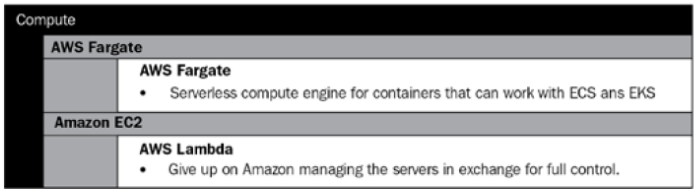


Figure 11.3 – Microservice compute services in AWS

The storage services that can be leveraged to create microservices are listed as follows:

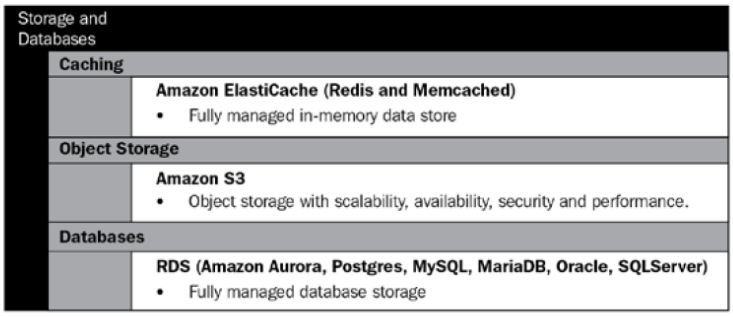


Figure 11.4 – Microservice storage and database services in AWS

The networking services that are normally used to create microservices are as follows:

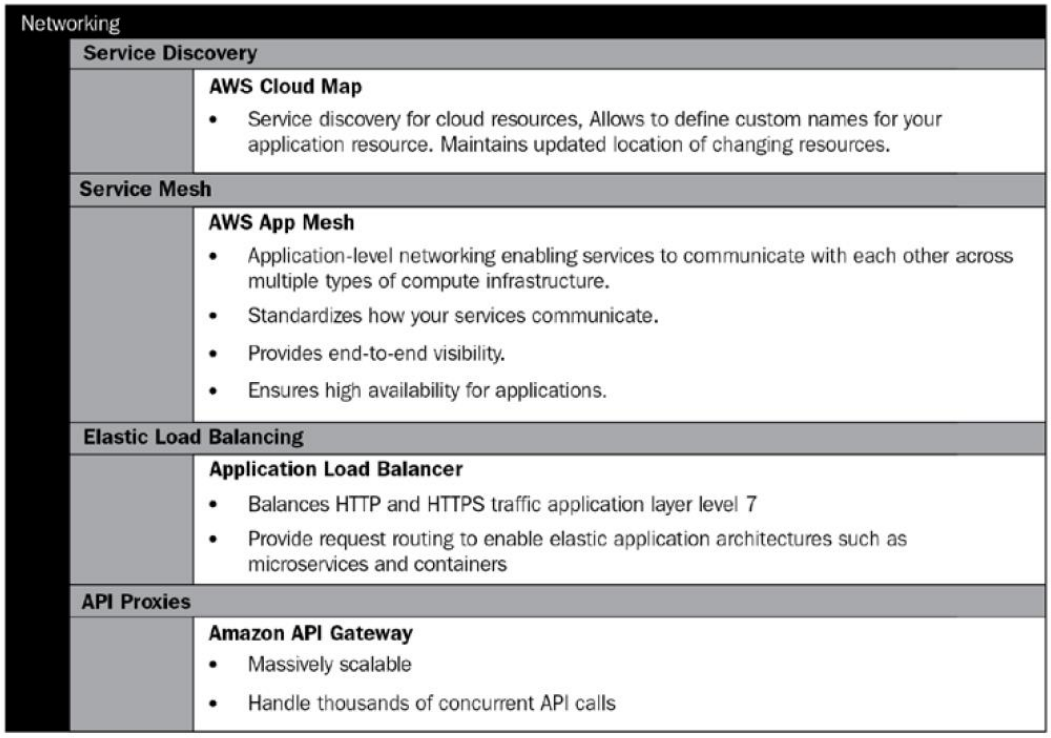


Figure 11.5 – Microservice networking services in AWS

The messaging services that AWS offers to create microservices are as follows:

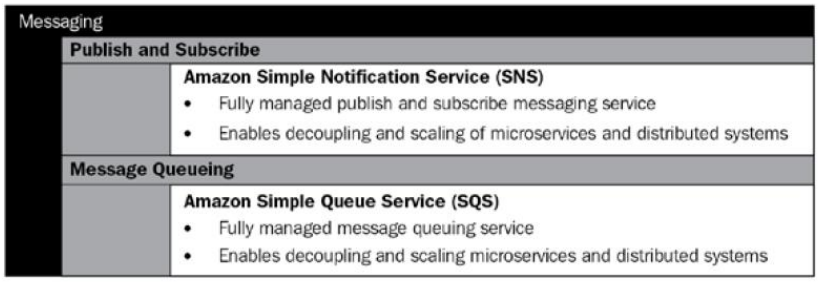


Figure 11.6 – Microservice messaging services in AWS

The networking services that are normally used to create microservices are as follows:

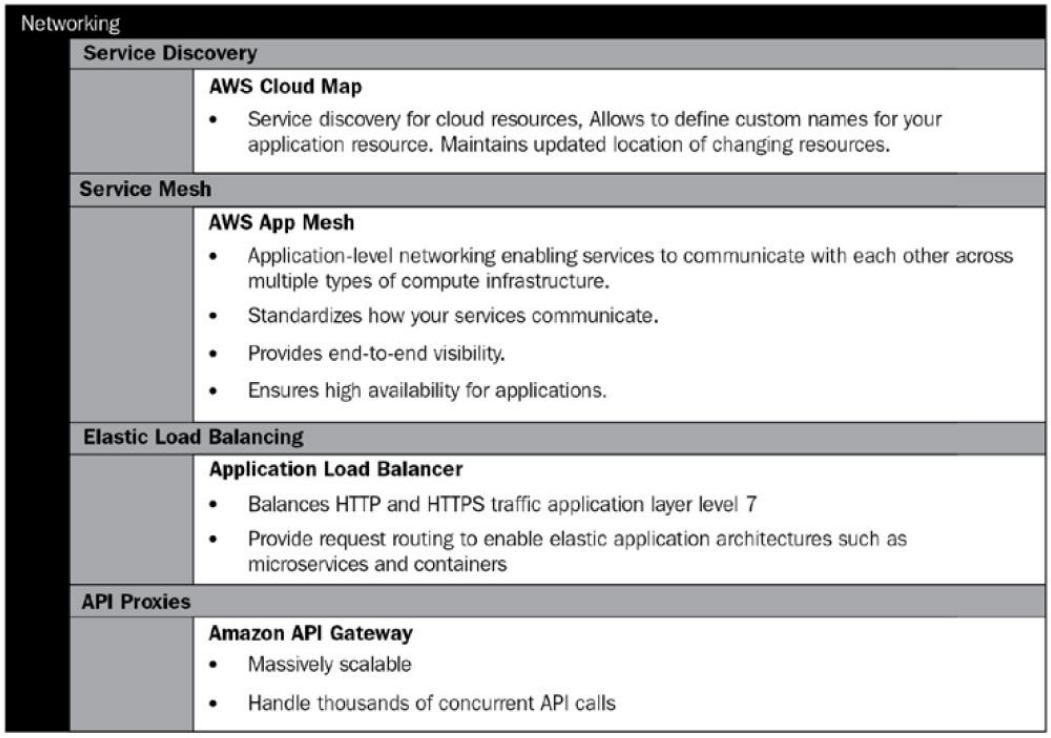


Figure 11.5 – Microservice networking services in AWS

The messaging services that AWS offers to create microservices are as follows:

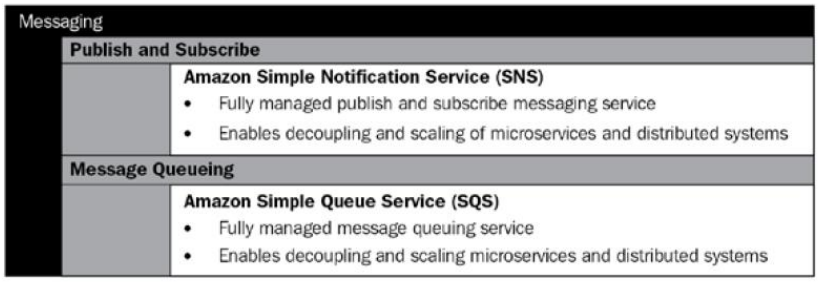


Figure 11.6 – Microservice messaging services in AWS

Perhaps not surprisingly, there are also economic benefits to using managed services such as AWS Fargate. Even though a Fargate instance is normally going to be more expensive than a vanilla EC2 instance, the **Total Cost of Ownership** (**TCO**) is often going to be lower for most use cases. When building a business case to present to your leadership to allow them to choose a solution, make sure that you include all costs associated with a nonelastic solution or an on-premises solution. Costs like in-house staff can easily be overlooked.

AWS Fargate can potentially reduce costs in the following ways:

- You only get charged for the time in which application traffic is flowing and not for the time that the underlying virtual instances are up and running.
- It can automatically provision the correct number of containers that need to run in order to handle a workload.

It is not uncommon for AWS Fargate deployments to reduce costs by more than 10 percent when compared to EC2/EKS/ECS deployments, and when using TCO metrics, a total saving of 25 percent or higher over a typical container deployment is not uncommon.

Amazon EKS versus Amazon ECS

A question that is often asked is, what service is better? Amazon EKS or Amazon ECS? The answer is going to depend on your particular use case. Let's review the strengths and weaknesses of each while comparing them head to head and you can use this information to apply it to your particular problem. The following diagram gives you a general idea of which is better based on their characteristics:

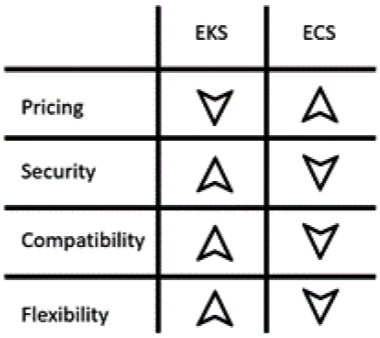


Figure 11.9 – Amazon EKS versus Amazon ECS comparison

- Chapter 12, Data Lake Patterns – Integrating Your Data across the Enterprise

The **data lake pattern** is an incredibly useful pattern in today's enterprise to overcome this challenge. In this chapter, we will do our best to set you up for success in your data lake journey. In this chapter, we will cover the following:

- The definition of a data lake
- The purpose of a data lake
- Data lake components
- Characteristics of a data lake
- The difficulty of making a data lake work like Google
- Data lake best practices
- Key metrics of a data lake

Definition of a data lake

It's a good time to be alive. We have a tremendous amount of information available at just a few keystrokes (thank you, Google) or a simple voice command (thank you, Alexa).

The data that companies are generating is richer than ever before. The amount they are generating is growing at an exponential rate. Fortunately, the processing power needed to harness this deluge of data is ever increasing and becoming cheaper. Cloud technologies such as AWS allow us to scale data almost instantaneously and in a massive fashion:

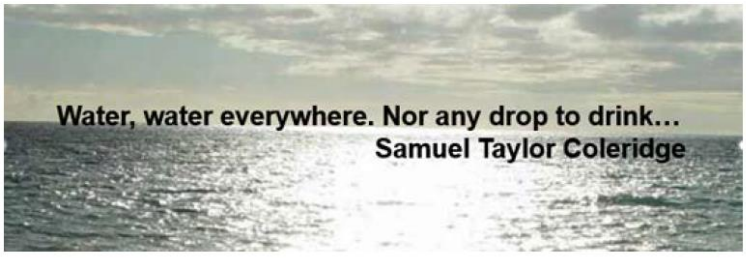


Figure 12.1 – The problem with the abundance of information

Do you remember, before the internet, that we thought that the cause of collective stupidity was a lack of information?

Well, it wasn't…

Data is everywhere today. It was always there, but it was too expensive to keep it. With the massive drops in storage costs, enterprises are keeping much of what they were throwing away before. And this is the problem.

Sandboxes

Sandboxes are an integral part of the data lake because they allow data scientists, analysts, and other users to manipulate, twist, and turn data to suit their individual use cases. Sandboxes are a play area where analysts can make data changes without affecting other users:

- Authorized users can transfer data from other zones to their own personal sandbox zone.
- Data in a personal private zone can be transformed, morphed, and filtered for private use without affecting the original data source:

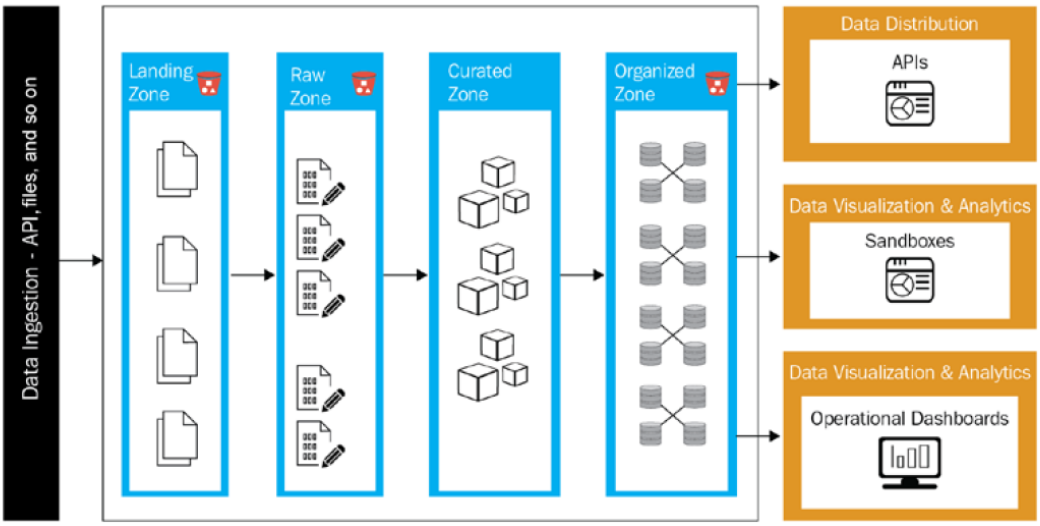


Figure 12.3 – The different components of a data lake

Metrics to gauge the success of your data lake

A list of metrics that can be used to gauge the success of your data lake follows. It is not meant to be a comprehensive list but rather a starting point to generate the metrics applicable to your implementation:

- **Size**: You may want to track two measurements: **total lake size** and **trusted zone size**. The total lake size itself might not be significant or provide any value. The lake could be full of useless data or valuable data. However, this number has a direct effect on your billing costs. One way to keep this number in check and reduce your costs is to set up an archival or purge policy. Your documents could be moved to long-term storage such as Amazon S3 Glacier, or they could be permanently deleted. Amazon S3 provides a convenient way to purge files by using life cycle policies.

  For the trusted zone size, the bigger the number, the better. It is a measure of how much *clean data* exists in the lake. You can dump enormous amounts of data into the raw data zone. If it's never transformed, cleaned, and governed, it is useless.

- **Governability**: This might be a difficult characteristic to measure but it's an important one. Not all data must be governed. The critical data needs to be identified and a governance layer should be added on top of it. There are many opportunities to track governability. The criticality of data is key to establishing an efficient data governance program. Data on the annual financial report for the company is just a little more important than data on the times that the ping pong club meets every week. Data that is deemed critical to track is dubbed a **Critical Data Element** (**CDE**). One way to ensure effective governability is to designate CDEs and relate them at the dataset level to the data in the lake. You can then track what percentage of CDEs are matched and resolved at the column level. Another way is to track the number of approved CDEs against the total CDEs. One final method is to track the number of modifications made to CDEs after the CDEs have already been approved.

- **Quality**: Data quality does not need to be perfect. It just needs to be good enough for the domain.

  For example, if you are using a dataset to generate this quarter's financial report, the numbers being used better be accurate.

  If the use case is trying to determine who should receive a marketing email, the data still must be fairly clean. However, if some of the emails are invalid, it's not going to be a huge issue.

In this chapter, we will cover the following topics:

- Availability in cloud computing
- Reliability in cloud computing
- Scalability in cloud computing
- Architectures to provide high availability, reliability, and scalability
- **Recovery Point Objective (RPO) and Recovery Time Objective (RTO)**
- Chaos engineering
- Scaling in and scaling out versus scaling up and scaling down
- Availability in AWS
- **Elastic Load Balancing (ELB)**

Chaos engineering

Chaos engineering is a methodology devoted to building resilient systems by purposely trying to break them and expose their weaknesses. It is much better to deal with a problem when we are expecting it to happen. A well-thought-out plan needs to be in place to manage failure that can occur in any system. This plan should allow the recovery of the system in a timely manner so that our customers and our leadership can continue to have confidence in our production systems.

A common refrain is that "*we learn more from failure than we learn from success*". Chaos engineering takes this refrain and applies it to computing infrastructure. However, instead of waiting for failure to occur, chaos engineering creates these failure conditions in a controlled manner in order to test the resiliency of our systems.

To achieve this, we will cover the following topics:

- Reviewing the list of available AWS programming languages
- Understanding the serverless AWS microservice architecture
- Setting up services
- Learning about frontend user interfaces (UX)
- Authenticating, authorizing, and managing users
- Grasping **Caps for acronym introduction** (**CDN**) concepts
- Understanding the asynchronous communication service
- Learning about file uploads
- Covering the AWS API Gateway
- Reviewing business APIs
- Understanding state-machine services
- Exploring backend persistence service
- Learning about asynchronous activities

JavaScript

Depending on who you ask, **JavaScript** may be the most popular programming language today. It was born as a client-side scripting language, but it has moved over to the server as well, and many highly trafficked sites run using JavaScript as the server-side scripting language.

One place where we can check out the popularity of languages is the pervasive versioning site, GitHub. According to GitHub, JavaScript has consistently been the most popular language since at least 2014.

Here are the most recent available statistics (as of the first quarter of 2020). As you can see, JavaScript (**18.703%** share) has a slight edge over Python (**16.238%** share) and a bigger lead over Java (**10.938%** share):

Figure 14.1 – Pull requests on GitHub by language. Source: Github.com

With this popularity in development comes a large development community, which translates into many available packages and libraries, as well as a huge user base of available talent.

One of the most popular JavaScript libraries for microservice development is **Feathers**. Feathers is a lightweight JavaScript framework specifically designed for the creation of REST APIs.