**Architecture 101**

**Access Management**

**Principal :** A person or an application that can make an authenticated(security info is exchanged) or anonymous(no security info is exchanged) request to perform an action on a system.

**Authentication :** The process of authenticating a principal against an identity. This could be via user\_ id/password or API keys. Authentication is the process of proving that you are who you say you are. So you're a principal.

**Identities:** Objects that requires authentication and are authorized to access resources.

You're attempting to prove that you're an identity and the process of doing that is called authentication. When you log into a website be it Netflix, your email, or the Linux Academy platform, or even AWS you are authenticating onto that platform.

**Authorization:** The process of checking and allowing or denying access to a resource or identity.

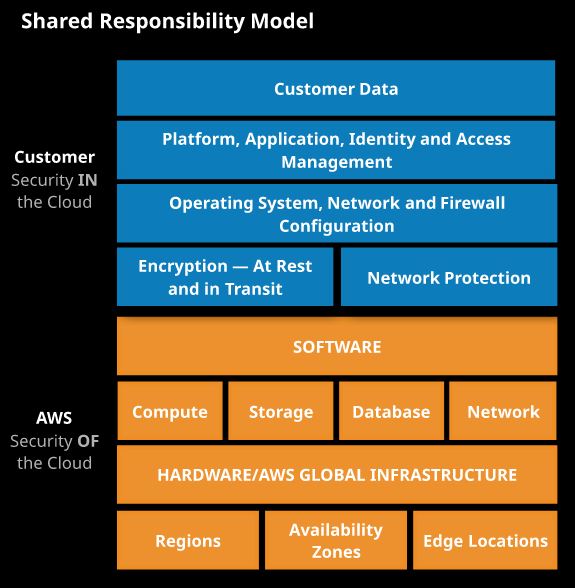
Once you are authenticated , you become the identity the identity is then given permissions either allow permissions or deny permissions to access certain areas of that system.

**Shared Responsibility/Security Model:**

The security of the cloud is AWS's job and the security in the cloud is your job.

**AWS:**  Manages the AWS regions, availability zones, edge locations, which are all AWS physical networking components & the software that’s used by many of the AWS products and services are managed by AWS.

**Customer:** Manages the data you store inside AWS, the authentication & authorization process, operating systems or patches to those operating systems, firewall configuration, encryption are managed by you.

So basically when you deploy stuffs in AWS there is an element of shared responsibility, and the amount of responsibility for you or AWS varies from Product to Product. Some products you're responsible for very little of and other the products you control and are responsible for major components or even the majority of it.

**Service Models:**

Service models define how a product or service is delivered, how you pay and what you receive. They also define which part of the product you manage and accept the risks for and which part the vendor is responsible for.

**IaaS:** Infrastructure as a Service. Example - DigitalOcean, Linode, Rackspace, AWS EC2, Cisco Metapod, Microsoft Azure, Google Compute Engine (GCE)

**PaaS:** Platform as a Service. Example - AWS Elastic Beanstalk, Windows Azure, Heroku, Force.com, Google App Engine, Apache Stratos, OpenShift

**SaaS:** Software as a Service. Example - Gmail, Netflix, Google Apps, Dropbox, Salesforce, Cisco WebEx, Concur, GoToMeeting

**FaaS:** Function as a Service. Example - Lambda.

Components managed by Customer & Cloud Provider within different Service Models:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **IaaS** | **PaaS** | **SaaS** |  |  |
| Data | Data | Data |  | Customer Managed |
| Application | Application | Application |  | Cloud Provider Managed |
| Run Time | Run Time | Run Time |  |  |
| Operating System(OS) | Operating System(OS) | Operating System(OS) |  |  |
| Virtualization | Virtualization | Virtualization |  |  |
| Host/Servers | Host/Servers | Host/Servers |  |  |
| Network & Storage | Network & Storage | Network & Storage |  |  |
| Data Center | Data Center | Data Center |  |  |

**High Availability:**

High Availability is hardware/software configuration allowing a system to recover quickly event of a failure. Highly available System may **minimize outage and in case of a failure**, the system may recover quickly but there may be disruption of service.

**Fault Tolerance:**

Fault Tolerant systems are designed to **operate through a system failure with no user impact.** They are generally more expensive and are complex to achieve.

**RPO(Recovery Point Objective):**

How much loss a business can endure expressed in time. The maximum time between a failure and last successful backup. RPO can be lowered by increasing the frequency of backups.

**RTO(Recovery Time Objective):**

The maximum amount of time a system can be down. How long a solution takes to recover. Lowering the RTO essentially means improving the speed at which you can affect quick and efficient recovery.

**Scaling:**

Scaling is the ability of a system to scale with ever increasing demand, and it can be done in two different ways.

**Vertical Scaling:** Vertical Scaling is achieved by adding additional resources in form of CPU or memory to existing machines. By doing so, machine is able to service additional customers or perform compute tasks faster.

(-)Maximum Machine Sizes will constrain the ability to scale-either technically or from a cost perspective.

**Horizontal Scaling:** Horizontal scaling is achieved by adding additional machines into a pool of resources, each of which provides the same service.

(+)Doesn't suffer any size limitations like Vertical Scaling and can scale to near infinite levels.

(+)Since this is a pool of resources the risk is spread, failure of any resource in the pool would effect a small number of resources

(-)Requires application support to scale effectively.

**Tiered Application Design:**

Architecturally an application consists of 3 tiers:

**Presentation Tier:** Interacts with the consumer of the application.

**Logic Tier:** Delivers the application functionality.

**Data Tier:** Controls interaction with a database of a kind.

**Monolithic Application:** If the presentation, logic and the data tier are implemented in the same codebase and are not separated they are called monolithic application.

(-) Hard to scale.

(-) Only Vertical Scaling is possible.

**Tiered Application:** Applications if designed correctly, implement these tiers as isolated components. Architecturally this can be provisioned in different machines or pools of machines. As each tier has different demands for CPU, memory & IO, it allows each tier's performance to be managed independently.

(+) Offers lot more flexibility in terms of scaling the application.

(+) Horizontal Scaling is possible.

**Encryption:**

Encryption is the process of taking plain text and converting into ciphertext, and converting ciphertext into plain text. Plain text & Ciphertext can be text, images and any other data.

Encryption generally uses an algorithm and one or more keys.

Encryption is used in two major situations:

**Encryption at Rest:** An encrypted file that's stored on a hard drive or USB

**Encryption at Transit:** Process of making sure communication between two files are encrypted. Example- https websites.

The process of encryption can be **symmetrical**, where the same key is used for encryption & decryption or **asymmetrical**, different - called public key & private key is used for encryption and decryption.

**Cost efficient or cost effective:** Implementing a solution within AWS using products or product features that provide the required service for as little initial and ongoing cost as possible. Using your funds effectively and knowing if product X is better or worse than product V for a given solution.

**Secure:** In a systems architecture context, implementing a given solution that secures data and operations as much as possible from an internal or external attack.

**Application session state:** Data that represents what a customer is doing, what they have chosen, or what they have configured. Examples include items and quantities in a shopping cart, notes on an X-ray, and 3D position of a real-time heart scan. Session state can be stored on a server (stateful server) or externally to a server (stateless server).

**Undifferentiated heavy lifting:** A part of an application, system, or platform that is not specific to your business. Allowing a vendor (AWS) to handle this part frees your staff to work on adding direct value to your customers.

**AWS Architecture 101**

**AWS Accounts:**

AWS Accounts provide 3 functions:

**Authentication:**

AWS Accounts are isolated. Initially they are created with a root user. This user via it's username, password, API Keys, is the only identity that can use(authenticate) to the account.

If the account credential is leaked, the impact(blast radius) is limited to that account.

**Authorization:**

Authorization is controlled on a per account basis. The root user starts with full control of the account and resources. Additional identities can be created or external identities can be granted access.

Unless defined otherwise no identity apart from the root account has access to the resources.

**Billing:**

Every AWS account has its own isolated billing information. This is initially in the form of an attached credit card, but established accounts can be converted to use traditional, term-based invoicing. By default, you are only billed for resources in your account. Billing or security exploits are limited to a single account.

However we can have "Consolidated Billing" where a master account is charged for all member account usage.

**AWS Region:**

To help improve reliability and allow customers to operate in each country effectively whilst obeying any data sovereignty, or data location laws, AWS has introduced a concept known as an AWS region, which is basically a local grouping of infrastructure.

**Number of AWS Regions:** 24(3 are announced)

**Use Cases:**

(+)Regions are Isolated so failure on one region won't services impact other region.

(+)Region also helps isolate the data to a specific country so that it allows you to control which data sovereignty laws the data comes under.

**AWS Global network:**

The Global AWS infrastructure is connected by a highspeed network called the "AWS Global network".

**Use Cases:**

(+)There are many AWS services that are able to utilize this high speed network to achieve better transfer rates, lower latency, and an improved user experience and they do this by transferring data over this network rather than the public internet and this network has been optimized for performance by AWS for use with AWS products.

**AWS Availability Zones:**

Regions are further divided in Availability Zones and they are isolated infrastructure with a region. They are completely separate facilities comprising of one or more data centers connected by high speed network.

**Example:** Each availability zone is denoted by a letter.The North Virginia region, which is us-east-1, we've got different availability zones so us-east-1a, us-east-1b, us-east-1c, and so on.

**Total number of AWS Availability Zones:** 76

**Use Cases:**

(+)If a solution is designed to run from multiple availability zones inside the same region then it's protected against major faults in that region, if one availability zone fails, it won't impact services in another.

(+)High availability can be ensured by using multiple availability zones in a region or multiple regions can be used globally or both. For example Netflix designed their platform to operate globally, utilizing multiple AWS regions and multiple availability zones in each of those regions.

**POPs:**

POPs or Point of Presence are smaller in size. They have local storage and they're connected to the AWS Global network and other AWS regions. Some services can use them for limited compute tasks such as edge based compute. You can even utilize them with some AWS services, but you can't treat them as a fully-fledged region or availability zone. These points of presence are generally used by AWS products such as CloudFront, which is a content delivery network.

**Example:** Australia currently has a single region based in Sydney, which is Asia Pacific 2. Now that region has three availability zones but there are more places that people live in Australia than just Sydney. Perth, for example, which is several time zones away, and even Melbourne, which is a short plane ride. Each of those major population centers has a point of presence mean that users in those areas convention fit from data being stored and delivered from a point closer to them geographically, that this means better speed, lower latency and more consistency in terms of performance. It's just a better user experience.

**Total Number of POPs:** 216 (205 Edge Locations and 11 Regional Edge Caches)

**Use Cases:**

(+)There may be scenarios where a platform need to distribute content or operate one or more applications globally. In such scenarios we could make it a global deployment using multiple AWS regions but in some cases there may be a POP located much near to the user's location we could make use of AWS edge locations, also known as POPs or points of presence.

Exam Tips:

Basically, we start off with AWS regions, which are large pockets of infrastructure and distributed geographically to meet legal and data storage requirements as well as providing global resilience for larger AWS customers such as Netflix.

Moving down from these regions, we've got availability zones and availability zones provide fault tolerance and high availability within that region. They're isolated infrastructure connected in that region with high speed networking.

Each of those availability zones might be one facility or multiple facilities that are close together and then helping to push content even closer to your users are the network of AWS POPs, also known as point of presence, or edge locations, Now I would recommend using this site and exploring the AWS Global Network really becoming comfortable with all of the different components that make up the global AWS cloud computing and networking platform.

**AWS Well-Architected Framework:**

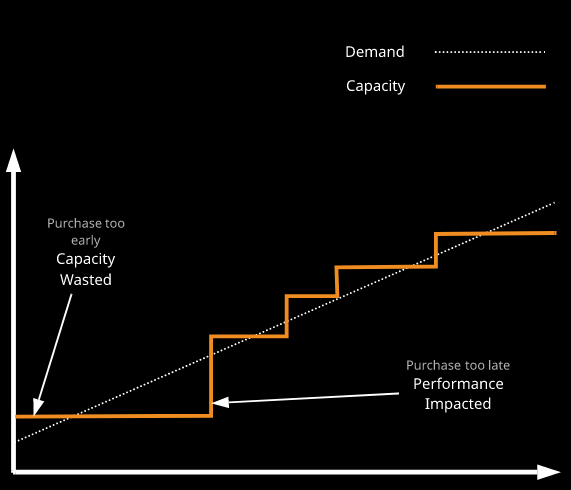
The Well-Architected Framework has been developed to help cloud architects build secure, high-performing, resilient, and efficient infrastructure for their applications.

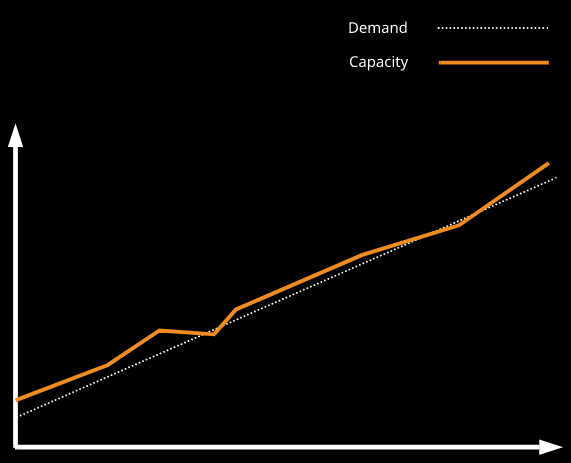
Based on five pillars — operational excellence, security, reliability, performance efficiency, and cost optimization — the Framework provides a consistent approach for customers and partners to evaluate architectures, and implement designs that will scale over time.

**Task: Read the white Paper**

|  |  |
| --- | --- |
| Well-Architected Framework Pillars | |
| Operational Excellence | The operational excellence pillar focuses on running and monitoring systems to deliver business value, and continually improving processes and procedures. Key topics include managing and automating changes, responding to events, and defining standards to successfully manage daily operations. |
| Security | The security pillar focuses on protecting information & systems. Key topics include confidentiality and integrity of data, identifying and managing who can do what with privilege management, protecting systems, and establishing controls to detect security events. |
| Reliability | The reliability pillar focuses on the ability to prevent, and quickly recover from failures to meet business and customer demand. Key topics include foundational elements around setup, cross project requirements, recovery planning, and how we handle change. |
| Performance Efficiency | The performance efficiency pillar focuses on using IT and computing resources efficiently. Key topics include selecting the right resource types and sizes based on workload requirements, monitoring performance, and making informed decisions to maintain efficiency as business needs evolve. |
| Cost Optimization | Cost Optimization focuses on avoiding un-needed costs. Key topics include understanding and controlling where money is being spent, selecting the most appropriate and right number of resource types, analyzing spend over time, and scaling to meet business needs without overspending. |

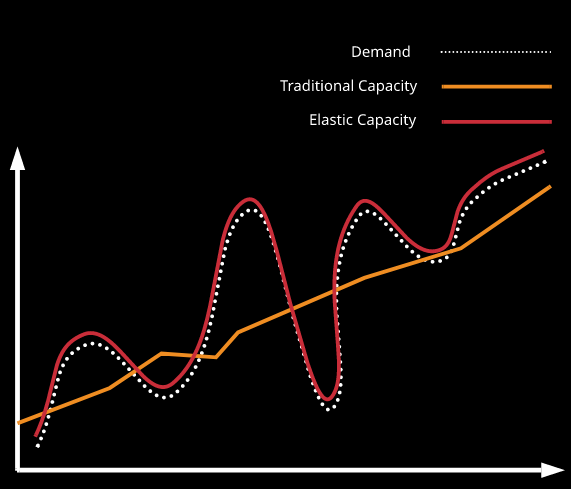
**Elasticity:**

**Vertical Scaling:** Traditional legacy systems use vertical scaling. An attempt is made to forecast demand and purchase servers ideally before the demand passes current capacity. Purchase too early and capacity is wasted. Purchase too late and performance is impacted.

**Horizontal Scaling:** When horizontal scaling is used (more, smaller servers), capacity can be maintained closer to demand. There is less waste because servers are smaller and there's less risk of performance impact as each increase is less expensive, so it generally requires less approval.

**Elasticity:** Elasticity, or elastic scaling, is where automation and horizontal scaling are used in conjunction to match capacity with demand.

Demand is rarely so linear — it can increase or decrease, often in a rapid and sudden way. An efficient platform should scale OUT and IN, matching demands on that system.

Elasticity allows you to achieve maximum cost optimization and also achieve the best performance efficiency.

**S3:**

Simple Storage Service(S3) is a **global** object Storage platform that can be used to store objects in form of text files, photos, audio, movies, large binaries or other types.

**Features:**

Global Service.

Object Storage.

**Bucket:**

Bucket is the basic entity inside S3, which is a container which can store things

Buckets are specific to a region and the data inside a bucket is replicated across a minimum of 3 availability zones in that region.

By default, the bucket is **blocked from all public access**.

Bucket Names are globally unique.

**Objects:**

So a single bucket is capable of storing multiple objects. Bucket can have folders(A bucket is actually a flat structure, it can't contain folders, they are just files that look like folders).

Objects are similar to files stored on a file server. It could be empty or it can vary in size. Objects are different that files in that they're a name value or a key value pair. An object's key is its name. An object's value is its data.

**Size of Objects:** 0 to 5 TB

**Exam Facts for S3:**

* **Bucket names** have to be **globally unique.**  **Minimum of 3** and **maximum of 63 characters**—uppercase or underscores
* Must **start with a lowercase letter** number and can't be formatted as an IP address (1.1.1.1)
* **Default 100 buckets** per account, and hard **1,000-bucket limit via support request**
* **Unlimited objects** in buckets.
* **Unlimited total capacity** for a bucket.
* An object's **key** is its **name**. An object's **value** is its **data**.
* An object's size is from **0 to 5 TB**

**Use Cases of S3:**

* (+) If you need data to be safe and secure then S3 is perfect, it's not vulnerable to a single AZ failure impacting it.
* (+)In scenarios where you need scalability, performance & fault tolerance. S3 can tolerate large amounts of failure within a region and still work correctly. It also scales quite well. S3 largely handles any performance or scaling.
* (+)Share data inside AWS. So if you have multiple servers which need to read and write, objects may be shared media for a web application or maybe patient's scans or other medical imaging for a medical application then again, S3 is perfect.
* (+)If you need to offer this data to the public internet at scale then S3 is brilliant.

**Cases where S3 is not to be used:**

* (-) S3 can't be easily mounted into a server as a network drive letter or mount point. Though there are hacks which allow this, but for the exam and probably for production, just don't.
* (-) S3 buckets can't be attached to a server as a disk. That's something that's called block storage and S3 is object storage.

**Replication:**

Replication enables automatic, asynchronous copying of objects across Amazon S3 buckets. Buckets that are configured for object replication can be owned by the same AWS account or by different accounts. You can copy objects between different AWS Regions(Cross Region Replication) or within the same Region(Same Region Replication).

To enable object replication, you add a replication configuration to your source bucket. The minimum configuration must provide the following:

The destination bucket where you want Amazon S3 to replicate objects

An AWS Identity and Access Management (IAM) role that Amazon S3 can assume to replicate objects on your behalf

**Same-Region Replication (SRR):**

Amazon S3 now supports automatic and asynchronous replication of newly uploaded S3 objects to a destination bucket in the same AWS Region. It’s adds anew replication option to S3 building on top of CRR.

**Cross-Region Replication (CRR):**

Cross-Region Replication (CRR) replicates data across different AWS Regions.

Replicated objects can be owned by the same AWS account as the original copy or by different accounts, to protect from accidental deletion.

**Cloud Formation:**

CloudFormation is an Infrastructure as Code (laC) product—you can create, manage, and remove infrastructure using JSON or YAML.

**Template:** A CFN template is JSON or YAML. It contains logical resources and configuration.

**Stack:** Stacks are created and modified based on templates, which can be changed and used to update a stack. Each stack has a unique name for the region for the account.

**Physical Resources:** Stacks take logical resources from a template and create, update, or delete the physical resources in AWS.

Any resource that’s created by cloudformation where we don’t specify a physical resource id is generated a physical resource id by cloudformation. The structure of this id is **<stack name >-<name of the logical resource>-<random string>.** In cloud formation it’s best practice not specify resource name so that resource id(s) are automatically generated by cloudwatch so that same templates can be reused to create resources. For example we need S3 bucket names to be unique across all the accounts gobally.

**Use Cases:**

CloudFormation is effective if you **frequently deploy the same infrastructure** or you require guaranteed consistent configuration.

**Important Tags inside a Cloud Formation Template:**

* **Resources:** This is the only **mandatory section** of a cloud formation template. The resources section of the template the resources section of the template is where we define our resources.
* **AWSTemplateFormatVersion:** Template version, which is AWS's way of future proofing the format. So the current version is "2010-09-09". CloudFormation does make an assumption if this is not present but if you do want to be specific, you can state this template format version.
* **Description:** Description, allows us to give a free text description to the template. \*\*If you are putting a description in, it does need to **immediately follow this AWS template format version**.
* **Metadata:**
* **Parameters:** Parameters is a way that a template can **ask the input from a user**. Parameters is a great way for the template to **make adjustments based on input**.

For example if a template is creating a WordPress blog well the parameters section is a way the template can ask for the size of the blog. So what amount of CPU and memory is the instance going to have that's going to run the blog.

* **Mappings:**
* **Outputs:** The outputs part the template is the way in which CloudFormation can **generate some outputs** to be delivered back to the user or the automated process that is using CloudFormation, an example of this might be if you're making a WordPress blog. This could contain the URL or web address to access that blog.

**Facts:**

* A CloudFormation (CFN/cfn) template is used to initially create a CFN stack. A stack **creates, updates, and deletes** physical AWS resources based on its **logical resources**, which are based on the contents of a **template**.
* A CFN template is written in **JSON or YAML**.
* A template can create up to **200 resources**.
* If a **stack** is **deleted**, then, by default, any **resources** it has created are also **deleted**.
* A stack can be **updated** by uploading a **new version** of a template.
* **New logical resources** cause **new physical resources**.
* **Removed** logical resources cause the stack to **delete** physical resources.
* **Changed** logical resources **update** with **some disruption** or **replace** physical resources. (<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html>)