Architecting on AWS version 7

These whiteboard notes were shared with my students for their personal use only. I would appreciate if they were not shared with other people.

It’s a live document, updated the July 6, 2022

When using it, be sure to download the most current version from:

<https://1drv.ms/w/s!AllwbeWFfFjNhOoptovcQV30g7yoRQ?e=kGn6KW>

Subjects in green are no longer in course version 7

Subjects in brown are new to course version 7

These notes are provided in good faith, and while I have done my best to ensure their correctness, I strongly encourage that you refer to the AWS product documentation.

Author: Joël Rouvray

Architecting fundamentals

* AWS services
  + Amazon Web Services (AWS) is the first, world’s most comprehensive and preferred cloud solution for businesses and public organizations.
  + Why customers move to AWS? 🡪 Maximize cloud agility and minimize complexity and risk
  + More than 200 service categories, each of which exposes an area of functionality, with AWS Foundational Services being used on a regular basis.
  + Solutions architects are responsible for managing an organization’s cloud computing architecture 🡪 They *plan*, *research*, and *build*.
  + The only way to interact with AWS services is through an API (application programming interface). You can call an AWS API using:
    - Management Console
    - CLI
    - Automation tools: AWS SDKs, AWS CloudFormation, AWS CDKs, etc.
  + AWS services typically fall into one of two categories:
    - "unmanaged", as EC2, require the user to manage how the service responds to changes in load, errors, and situations where resources become unavailable.
    - "managed”, as RDS, still require the user to configure them; however, they typically require far less configuration.
* AWS Global Infrastructure 🡪 Where the cloud "run”
  + A hierarchical structure connected by a private network
    - AWS Data Centers
      * “Small” isolated buildings (~ 50-80K servers, ~ 25MW-30MW) 🡪 Minimize the “blast radius”
      * When a datacenter fills up, AWS creates a new one, physically separate
      * Separated by <250 µsec à “Local” resources between datacenters
      * You never can choose a data center. In fact, AWS customers do not know or care how many data centers make up an Availability Zone.
      * Data Centers host purpose-build custom equipment 🡪 Not burdened with unnecessary features, high efficiencies.
      * AWS implements controls, builds automated systems, and conducts third-party data center audits to confirm security and compliance.
    - AWS Availability Zones (AZ)
      * Made up of one or more data centers
      * Separated by <2ms, typically, miles apart within a particular geographic location (AWS Region)
      * Data sync. replication between AZ (S3, EFS, DynamoDB, Aurora, etc.)
      * Many AWS services are inherently highly available, such as EFS, Lambda or S3, as they are natively replicated or distributed across at least 3 AZs. 🡪 For them, you cannot select AZs.
      * Other services as EC2 instances, EBS volumes, EMR clusters, NAT gateways, etc. run over a single AZ 🡪 For them, you should:
        + Select the Availability Zone where the resource will be created.
        + Deploy and configure the resource to minimize system downtime if the selected AZ fail à Is a best practice to “build for failures” and support “disposable” infrastructure.
      * One Availability Zone name (for example, us-east-1*b*) may relate to different physical AZs for two AWS customer accounts
    - AWS Regions
      * 26 geographic areas by December 2021, chosen to be closest to where the business traffic demands. Last was Asia Pacific (Jakarta)
      * Hundreds or thousands of miles between them
      * Made up of 3 or more Availability Zones
      * You always choose a Region to deploy resources and store data, except for global services (e.g., IAM, Route 53)
      * Data async. replication possible between Regions (S3 Cross Region Replication, DynamoDB Global Tables, etc.)
      * *You* enable, control and pay async. data replication across regions, using the AWS backbone (*never* enabled by AWS)
    - AWS Local Zones
      * Are "zones" that bring specific AWS resources (e.g., EC2, EBS, ECS, EKS) closer to several that are relatively far from primary data centers in an AWS region. The other AWS services (S3, Aurora, etc.) are still located in a “plain” Region and accessible privately over the AWS Global Network.
      * Provide these customers with those specific resources with *extremely low latency* that can be measured in single-digit milliseconds.
      * They are a logical part of the nearest AWS Region, which is the parent region. You can extend any VPC from the parent region into AWS Local Zones by creating a new subnet and assigning it to the AWS Local Zone.
      * Although they are *currently* *only available in cities in the US*, AWS announced that it will be expanding the local zones internationally across the world. These new AWS Local Zones will be made available starting in 2022 in more than 26 countries, including Brazil, Argentina, Chile, Perú, Colombia and Mexico.

Mapa

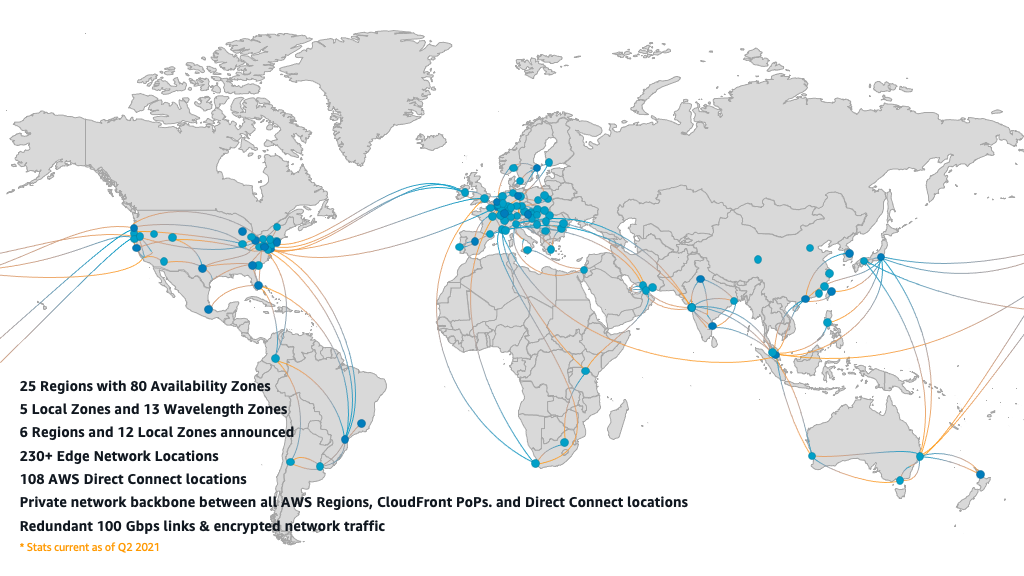
Descripción generada automáticamente

* + - AWS Edge Locations
      * The “edges” of the *AWS Global Network* (AWS backbone)
      * Located in more than 300 major cities around the world, they are the nearest point to a requester of an AWS service.

Mapa

Descripción generada automáticamente

* + - * Can support local AWS services as:
        + CloudFront à CDN (web cache)
        + WAF, Shield, etc. à DDoS security at the edge
        + S3 Transfer Acceleration à Route S3 traffic through the closest Edge Location
        + Global Acceleration 🡪 Route your TCP and UDP traffic through the closest Edge Location
        + Direct Connect à Private wired access to the AWS backbone
        + Route 53 à DNS
        + Lambda@Edge à Runs Node.js or Python code serverless
  + AWS Global Network à The AWS backbone that interconnects every AWS site
    - Private
    - Purpose-built
    - Fully redundant
    - Low latency
    - High bandwidth



* + How do you choose a region?
    - Data sovereignty, compliance, governance
    - Not all AWS services are available in all regions
    - Costs vary according to regions
    - Performance: latency and throughput depend on the "distance" between users and the Region
* The *Well Architected Framework*
  + Can answer these questions:
    - Where can I find guidance, recommendations, and best practices on *preparing* a secure, high-performing, resilient, and efficient application infrastructure?
    - How to know if *we are doing* things right?
    - Where to get help to *evolve* my AWS environment?
  + Provides a consistent set of strategies and best practices for customers and partners to evaluate designs and implement optimized cloud environments.
  + Organized in 6 *pillars*:
    - Security
    - Cost optimization
    - Reliability
    - Performance efficiency
    - Operational excellence
    - Sustainability (introduced December 2, 2021)
  + The AWS *Well-Architected Tool* is a self-service tool designed to help architects and their managers to review the state of your existing workloads and compare them to the latest AWS architectural best practices, without the need for an AWS Solutions Architect.

Account security: how to grant and deny access to AWS resources

* There are many security services in AWS (data center, networking, storage, etc.). In this module we will see *account* security.
* Principals (authentication), policies (authorization) and federation
* What should be the *first step* after creating an account? à Secure the AWS *account root user* because they have complete access to all AWS services and resources in the account.

1. You should enable multi-factor authentication (MFA) for your account root user (and other privileged IAM users).
2. Even if you will be the only person who works on your account, you should create a first *IAM admin user* to delegate AWS resources access management
3. You also should delegate access to the billing console to accountants (or delegate it to admin users as well). By default, admin users cannot view or manage billing <https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial_billing.html> .
4. You should limit the use of the root user to create the first IAM user with administrator access and delegate access to the billing console.
   * + After that, very few tasks require AWS account root user credentials (i.e., close the AWS account or change your AWS support plan).
     + Lock away the AWS account root password.
     + Do not create root access keys. If they were already created, delete them and do not recreate them unless absolutely necessary.
     + Avoid using the account root user for day-to-day interactions with AWS, even the administrative ones.
5. Instead of creating much more IAM users, you can use existing users from another AWS account, or federated users from Active Directory, Google, Facebook, etc.
   * This allows single sign-on (SSO), which simplifies and harden authentication (no more multiple passwords).

* What is IAM? à AWS *Identity and Access Management* (IAM) is the global service to centrally manage:
  + Users (person, AWS service)
    - *Who* can access AWS resources?
    - Must be *authenticated* before they can access it.
  + Permissions
    - *What* action can they do with *which* AWS resource?
    - Must be *authorized* to access it.
* *Authentication* in AWS (are you the one you claim to be?)
  + Who can be authenticated? 🡪 AWS *principals*:
    - IAM users
    - Federated users
    - Roles
    - Assumed-role sessions
    - AWS services (EC2 Instance running applications, Lambda function, CloudWatch, etc.)
    - Anonymous/Everyone users (not recommended)
    - AWS accounts (all IAM users and roles in the account, used for cross-account delegation)
  + Principals must be *authenticated* *before* being *authorized* to access AWS resources (except for Anonymous/Everyone users)
  + *Where* are users authenticated (sign-in)?
    - Users managed and authenticated *inside* AWS (*IAM Users* and the predefined *Account root user*) à *IAM* service
    - Users managed and authenticated *outside* AWS (*Federated users*) à AWS can accept (A.K.A. federate) users authenticated by a trusted *Identity Provider* (IdP)
      * Which Identity Providers are supported to authenticate Federated users?
        + Enterprise identity providers 🡪 Any service that supports *SAML 2.0* (Microsoft AD, Salesforce.com, Shibboleth, etc.).
        + Web identity providers 🡪 Any service that supports *OpenID Connect* (Google, Facebook, Login with Amazon, etc.).
        + Amazon Cognito user pools 🡪 “Your own” user directory.
  + *How* users can be authenticated?
    - To prove that you are the one you claim to be (authentication) you submit *credentials*.
    - Users managed and authenticated *inside* AWS provide *IAM credentials.*
      * IAM *username* and a *password*
        + To sign-in to the AWS Management Console
        + To access AWS Forums, AWS Support Center and AWS Marketplace
      * AWS Access Keys (*access key ID* and *secret access key*)
        + To use the *CLI* or *AWS tools for Windows PowerShell* (you cannot use the *IAM username* and *password*).
        + To sign programmatic requests to *APIs* (directly or through CLI or SDK).
      * Best practices:
        + Never use hardcoded credentials or store Access Key on an EC2 instance.Never!Use roles!
        + Do not create access keys for a user who requires access only through the Management Console
        + Can (and should) add MFA to provide an additional level of IAM user protection.
    - Users managed and authenticated *outside* AWS 🡪 When *federated users* request access to an AWS resource, if they are trusted (to assume the role), the AWS *STS* (Security Token Service) provide them with *temporary security credentials* (Session Key, Session Secret and Session Token).
      * The same mechanism applies when an *IAM user* or *AWS service* assumes a role
      * Extending the previous point, when an EC2 instance assume a role, any application that is deployed on that instance can retrieve from *Instance Metadata* the temporary credentials to access other AWS resources (*AWS service role for an EC2 instance* A.K.A. *EC2 Instance Profile*)
      * How to simplify STS access to mobile applications? à Amazon Cognito identity pools
      * Using *temporary security credentials* (generated by STS) instead of *access keys* (manually created and managed) is a best practice. Why?
        + They are generated dynamically à You can provide access to your AWS resources to external users without having to define a local IAM user for them.
        + They are not stored à You do not have to distribute or embed them with an application.
        + They expire automatically (the default expiration time depends on API and can be configured to last for anywhere from a few minutes to several hours) à You do not have to rotate them, or explicitly revoke them when they are no longer needed.
* *Authorization* in AWS (are you allowed to do what you want to do?)
  + By default, all AWS operations or resources access are denied for all *principals* (users and so on), even without an explicit Deny, except for the *account root user* (who, unless limited by *Organizations SCP*, always has complete access to all AWS services and resources in the account).
  + Remember: you must *authorize* users (grant permissions) *before* they can *access AWS resources* in your account
  + How to *authorize* access to AWS resources?
    - To assign *long-term* permissions (policy) to *IAM users* *within your account*, use *groups*
      * Attach a policy (permissions) to the group and put the IAM users in the group. All the users in the group inherit the permissions assigned to the group.
      * A user will be able to access those resources *until manually removed* from the group.
      * Is a best practice avoid assign policies *directly* to an IAM user (except to deny access).
    - To assign *temporary* permissions to *any principal* (e.g., local account IAM users, IAM users from another account, federated users, AWS services as EC2 instances) use *roles*
      * A role is a profile that, when assumed, allows requests to AWS for a limited time.
      * A role specifies *who* can assume it (trust policy) and *what* permissions it grants (permissions policy).
      * Before an IAM user, application, or service can use a role that you created, you must grant permissions to assume (switch to) the role.
      * Then, the user request temporary security credentials to the AWS Security Token Service (AWS STS) to assume roles that enable access to AWS resources using one of these methods:
        + Management console
        + CLI
        + AssumeRole, AssumeRoleWithSAML, or AssumeRoleWithWebIdentity API operation
        + Role trust policy example: the EC2 service can assume this role and vend short-term credentials to your application:

{

"Effect": "Allow",

"Principal": {

"Service": "ec2.amazonaws.com"}

"Action": "sts:AssumeRole",

}

* + - * When users request access to an AWS resource, if they are trusted (to assume temporary the role), they give up their original permissions and take on the permissions assigned to the role. When the user exits or stops using the role, the original user permissions are restored.
    - The use of roles is not limited to grant access to federated users or to delegate access across AWS accounts. It is good practice:
      * Using them to enforce the principle of least privilege (the user can switch temporarily to a role with more permissions when required by specific tasks)
      * Using them instead of AWS Access Keys (e.g., programmatically access, EC2 instances).
* How to define permissions (policies)?
  + You *authorize* access to AWS resources by *attaching* *policies* to IAM users, groups, or roles (IAM *identities*), or directly to AWS resources.
  + Note: do not confuse *principals* with *identities*.
    - *Principals* refers entities who can be authenticated (as a person), *identities* refer to entities to which policies can be attached (roles, IAM users, and groups)
    - You cannot specify *groups* and *EC2 instance profiles* as *principals* (they cannot sign-in)
  + A policy is a JSON document that, when associated with an *identity* or *resource*, defines their permissions.
  + The policy contains actions (API calls) which are allowed or denied on some resource.
  + An explicit allow overrides this default and an explicit deny overrides any allows (if a single denies the request, AWS stops evaluating policies and denies the entire request).
  + Policies only control access to AWS resources à They have no visibility above the hypervisor.
  + AWS evaluates these policies when the user or role makes the request.
  + Best practice: follow *least privilege principle* à Avoid grant privileges beyond the minimum required for a user or group to fulfill their job usual requirements. When required by a specific task, the user can temporarily switch to a role with more permissions.
  + AWS supports six types of policies:
    - Identity-based à Most used. Attached to IAM identities (IAM users, groups, or roles)
    - Resource-based à Attached to AWS resources.
    - ACLs à Legacy. Like resource-based policies, although they are the only policy type that does not use the JSON policy document structure.
    - Session policies à Advanced feature to create distinctive role session permissions or further restrict session permissions.
    - Permissions boundaries 🡪 To delegate permissions management to trusted employees. They do not grant any permissions, instead, permissions boundaries are filters that set maximum permissions to users or roles
    - Organizations SCPs (Service Control Policies) à Used by AWS Organizations to set permission guardrails across accounts. They do not grant any permissions, instead, SCPs are filters that allow only the specified services and actions to be used in affected accounts.
  + A single identity (role, IAM user, or group) can have multiple policies attached to it. Together, they represent the permissions for the user that apply to the request.
    - The AWS enforcement code evaluates all policies for a layered defense, where:
      * *SCP* and *permissions boundary* act as a filter to limit permissions,
      * *Resource* and *identity*-based policies can grant or deny permissions.
    - Any deviation from the narrowly defined permissions should result in a denial. And of course, explicit deny overrules everything.

Gráfico, Gráfico en cascada

Descripción generada automáticamente

* + Policies categorization
    - To grant permissions:
      * IAM identity-based policies
        + Managed policies

AWS managed policies

Service access policies

Job function policies

Customer managed policies

* + - * + Inline policies
      * IAM resource-based policies
    - To set maximum permissions:
      * IAM permissions boundaries policies
      * Organizations SCPs (Service Control Policies)
  + *Identity-based* policies à You attach the policy to IAM users, groups or roles (A.K.A. identities) and the policy specify what resources they can access
    - An example of an *identity-based* policy to allow read-only access to a static website on a S3 bucket if logged-in with MFA is:

{

"Version": "2012-10-17", 🡪 *Police language version*

"Statement": [

{

"Effect": "Allow", 🡪 *Either Allow or Deny*

"Action": "s3:GetObject", 🡪 *API call (to get objects)*

"Resource": "arn:aws:s3:::staticwebsite/\*" 🡪 *On this bucket*

"Condition": {

"BoolIfExists": {"aws:MultiFactorAuthPresent": true}

🡪 *Option to MFA enforcement*

}

]

}

* + - Identity-based policies can be further categorized in *managed* policies and *inline* policies.
      * *Managed* policies are “standalone” policies that can be attached to multiple IAM identities. This allows reusability.
      * Managedpolicies can be further categorized in *AWS managed* or *customer managed*.
        + *AWS* managed policies à Prewritten policies provided by AWS.

Designed to provide permissions for many common use cases (e.g., *ReadOnlyAccess* for *service access* or *AdministratorAccess* for *job function*)

Assigning the right permissions to IAM identities is easier than if you had to write the policies yourself from scratch.

Created and updated by AWS. You can use them, but you cannot edit them.

Best practice: Whenever possible, you should use AWS-managed policies to ensure that they are correct and receive automatic updates that reflect changes in AWS functionality.

* + - * + *Customer* managed policies à Created and managed by *you*.

Use case: Where an AWS managed policy does not meet your need.

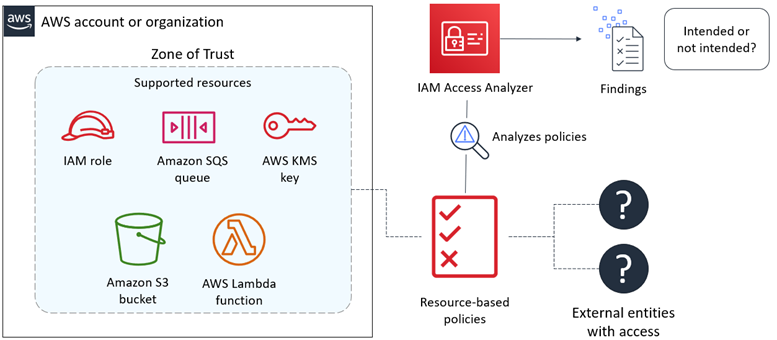
A single *customer managed* policy can be attached to multiple identities, *but only within your own account*.

* + - *Inline* policy is one that is directly *embedded* within a group, user, or role.
      * Maintain a strict one-to-one relationship between a policy and an identity
      * The policy is an inherent part of the associated identity à Deletion of the identity deletes the inline policy as well
      * An inline policy cannot be reused on other identities, but different identities can have its own copy of the policy.
      * Use case: When you want to be sure that the permissions in the policy are not inadvertently embedded to the wrong group, user, or role (there only can be embedded to a single identity).
      * As a best practice, use customer managed policies instead of inline policies.
        + Managed policies are centralized and reusable, so when the policy is updated, the changes are applied to all attached IAM identities.
  + *Resource-based* policies à You attach a policy to a resource and the policy specify which principals can access that resource
    - Supported only by some AWS services.
    - Use cases: when identity-based policies cannot be used. This is the case for:
      * There is no identity to attach it, or Anonymous/Everyone access such as S3 anonymous access
      * When an AWS service does not use a service role, such as an API Gateway.
      * When in a cross-account access, the principal must maintain access to resources in both accounts, such as copying data between accounts.
      * The role trust policy.
    - Unlike an identity-based policy, a resource-based policy must specify who (which principal) can access that resource.
    - Resource-based policies behave like inline policies 🡪 Deletion of the resource deletes the resource-based policy as well.
  + How to delegate permissions management to “power users” or developers? 🡪 Use *permissions boundaries*
    - Does not grant permissions. Instead, they restrict them, acting like filters
    - Limit the permissions that an identity-based policy can grant to users or roles
    - Allows users or roles to perform only the actions that are allowed by *both* its identity-based policies and its permissions boundaries (the permissions intersection).

Diagrama

Descripción generada automáticamente

* + - Use case: delegate permission management to “power users” or developers while limiting the maximum permissions they can grant to the users and roles that they create and manage.
* AWS security services and tools
  + Newly created AWS resources are private by default. The only way AWS resources become public unintentionally is by customer misconfiguration.
  + AWS customers can use multiple mechanisms for controlling access to resources: IAM policies, bucket policies, ACLs, Query String Authentication, CloudFront distributions, Access Points, Object ACLs, and your own custom application code. *This makes it easy to makes mistakes that lead to data leakage*.
  + AWS offers a wide range of security services and tools to help you achieve the level of security that you need to enforce within your environment. *Many services overlap*. As example, detecting when an S3 bucket has been made public can be made by Trusted Advisor, Macie, AWS Config Rules, S3 itself, etc.
  + AWS security services and tools examples:
    - How to identify security vulnerabilities in your resource-based policies? 🡪 IAM Access Analyzer
      * Helps you identify external access to your resources and data (such as S3 buckets or IAM roles), which can be a security risk.



* + - How does it work?
      1. You select your entire organization or your account (the zone of trust) in the same AWS Region that it's enabled in.
      2. Once enabled, it analyzes the resource-based policies in your zone of trust.
      3. For each instance of a resource that is shared outside of your zone of trust, it generates a finding.
      4. You can review findings to determine whether the access is intended and safe, or the access is unintended and a security risk.
      5. After the first analysis, Access Analyzer analyzes these policies periodically. If a new policy is added, or an existing policy is changed, Access Analyzer analyzes the new or updated policy within about 30 minutes

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

* + How to retrieve information about the last time that a user, user group, role, or policy was used to attempt to access AWS services and Amazon S3, Amazon EC2, IAM, and Lambda actions 🡪 *Access Advisor*

Captura de pantalla de un celular

Descripción generada automáticamente

* Managing multiple Accounts
  + Isolating some of your workload is generally a good practice
  + For most use cases, a single account is not enough
  + Many organizations want to use multiple accounts to create administrative or billing isolation, and to limit the impact of errors.
  + How do you manage multiple accounts?
    - To grant permissions across multiple accounts, you can use:
      * resource-based policies (they do not need an identity to attach the policies to)
      * roles (if resource-based policies are not available)
      * access control lists (legacy)
    - *AWS Organizations* is an account management service that helps you centrally govern and consolidate your multiples accounts, allowing:
      * Cost optimization (e.g., share Reserved Instances)
      * Billing consolidation
      * Security and governance
      * Limit the blast radius from an account security standpoint, enhancing users and workloads isolation
      * Resource grouping
      * Defining business units
    - AWS Organizations enables *SCP* (Service Control Policies) to set maximum permissions for users, roles, and *accounts* in your organization:
      * Like permissions boundaries, does not grant permissions. Instead, they restrict them, acting like filters.
      * Attaching an SCP to an Organizations entity (root, OU, or account) defines a guardrail.
        + Allows users, roles, and accounts to perform only the actions allowed by the intersection of their identity or resource-based policies and their SCP.
      * It is the only way to limit the *AWS root account* permissions
    - *AWS Control Tower* goes further.
      * Help you setup a secure, multi-account AWS environment quickly, based on AWS best practices, and manage it at scale.
      * How does it work?
        + Setup AWS Organizations with best practices blueprints. 🡪 Landing Zone.
        + Enable pre-packaged guardrails to enforce policies or detect violations 🡪 Policy management.
        + Get continuous visibility into how your workloads comply with guardrails 🡪 Dashboard for visibility.

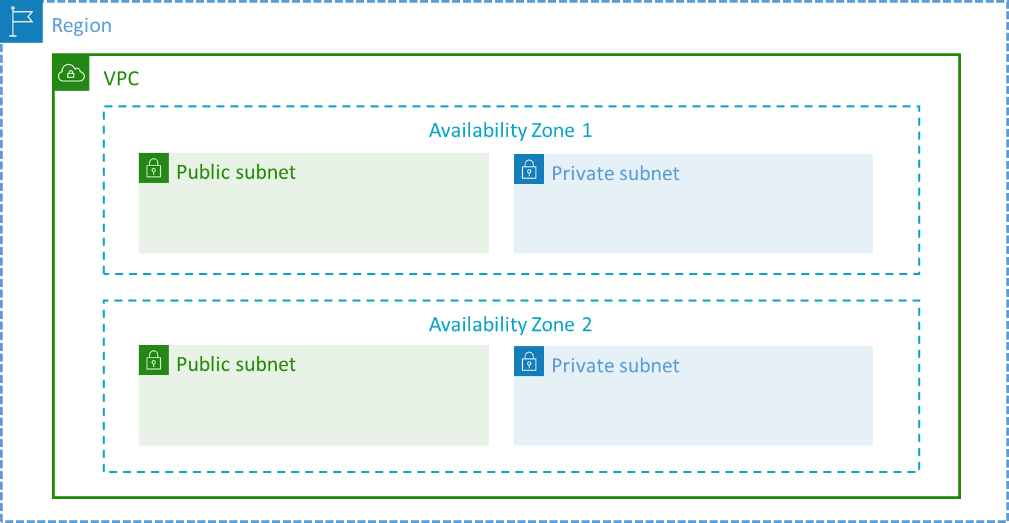
Networking-1: How to configure the network for instances to run securely?

* What is VPC? à A private, logical network on top of a single AWS region where you can launch AWS resources.
* Using VPC you can:
  + Define a range of private IP addresses to be used by your resources deployed into that VPC.
  + Create a redundant architecture using multiple AZs to protect your applications from the failure of a single AZ.
  + Configure several layers of network security for instances.
  + Connect resources external to your VPC without going through the internet (endpoints, etc.: module 10)
* How to define the range of private IP addresses to be used by resources deployed into the VPC?
  + When you create your VPC, you specify its set of private IP addresses using the CIDR (Classless Inter-Domain Routing) notation.
  + A main (default) route table is automatically created enabling traffic within the VPC
    - The main route table cannot be disabled.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

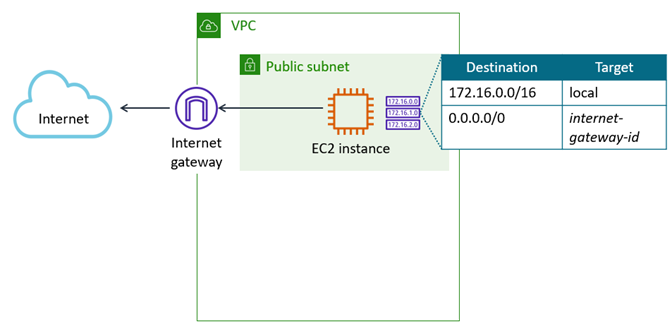
* + Best practices for choosing a VPC CIDR block
    - When you create your VPC, choose CIDR blocks wisely.
    - Plan, anticipating connectivity to other VPCs and on-premises networks
      * Don’t use an IP range that is already in use in your organization.
        + This way, you can later extend the VPC to other VPCs and on-premises networks.
      * RFC1918 range recommended.
        + 10.0.0.0 – 10.255.255.255 (10/8 prefix)
        + 172.16.0.0 – 172.31.255.255 (172.16/12 prefix)
        + 192.168.0.0 – 192.168.255.255 (192.168/16 prefix)
    - Use if possible /16 VPC 🡪 65536 addresses
      * RFC1918 allows 273 /16 CIDR blocks
* How can you protect your applications from the failure of a single AZ?
  + Divide your VPC creating custom *subnets in different AZs*, as each subnet resides entirely within one AZ and cannot span zones.
  + So, you can choose the AZ where to create your resources and implement a redundant architecture.



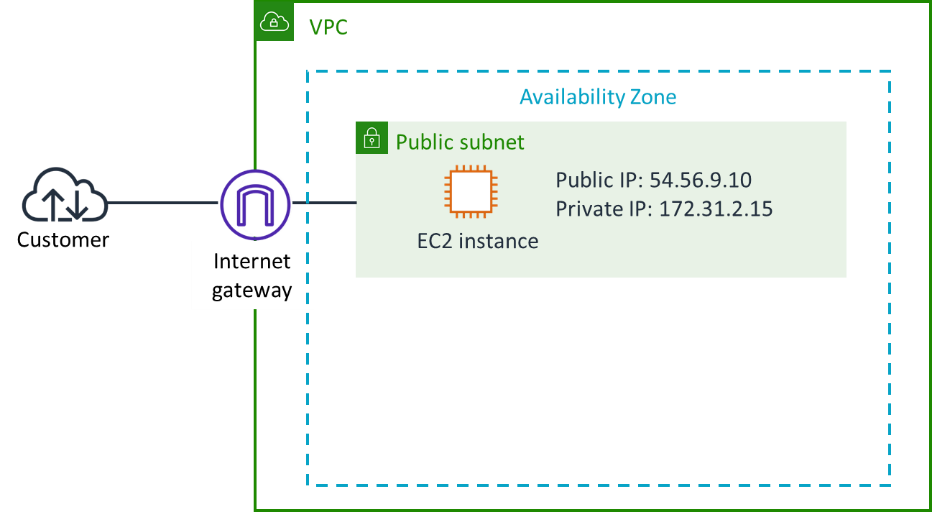
* + Note: AWS will reserve five IP addresses from each subnet
  + Best practices for Availability Zones:
    - Start with two AZs so you can build architectures that tolerate the unavailability of one AZ.
    - Eventually, add as many AZs as there are in the Region to take advantage of Spot Instances price differences between AZs.
  + How to configure several layers of network security?
  + *Defense in depth* is a strategy focused on creating multiple layers of security.
  + How to configure a layer of network security defining *different internet accessibility* for instances? 🡪 Create *public*, *private/app* and optionally, *protected/restricted/data* subnets.
    - Because a subnet can only be associated with one route table at a time, when custom routing tables are created, a route enabling traffic within the VPC is automatically added to the custom routing table to replace the use of the corresponding route in the main route table (and this route cannot be deleted either).
    - Resources in a *public subnet* supports inbound/outbound internet access
      * How to allow internet *outbound* access for instances in a public subnet?
        1. Create an *Internet Gateway* and attach it to the VPC
        2. Create and configure a custom *public* *route table*

Add to the public route table a route to direct internet-bound traffic (0.0.0.0/0 or ::/0 for IPv6) to the Internet Gateway

Associate the public routing table to the public subnet



* + - * How to allow internet *inbound* access for instances in a public subnet?
        + Assign an *Elastic IP Address* (persistent public IP) to instances.
        + Better and best practice 🡪 Setup an external facing Elastic Load Balancer as it serves as a single point of contact for multiple instances in multiple AZs



* + - Resources in a *private subnet* are only allowed to (indirect) outbound internet access (for example, for software updates).
      * How to allow internet *outbound* access for instances in a private subnet but prevent the internet from initiating a connection with those instances?
        1. For IPv4, create a *NAT instance* or a *NAT Gateway* (better, full administrated by AWS) in a public subnet and allocate it an Elastic IP Address (EIA). For *IPv6*, you can use the free *Egress-only internet gateway*

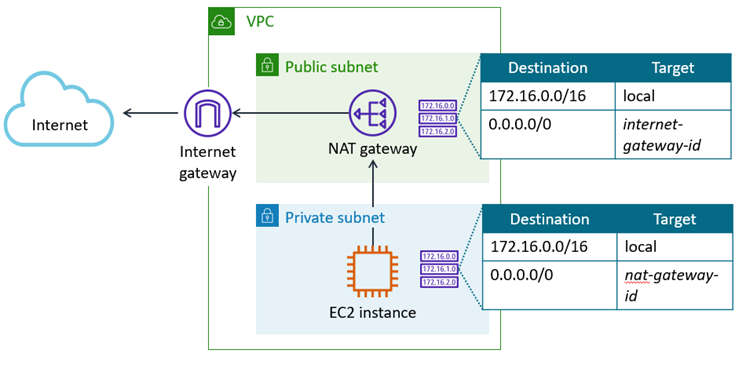
It replaces the source private IP address of the instances with the Elastic IP Address of the NAT gateway.

For High Availability (and to reduce inter-AZs traffic charges), configure two NAT Gateway (or NAT instances), each in a different AZ.

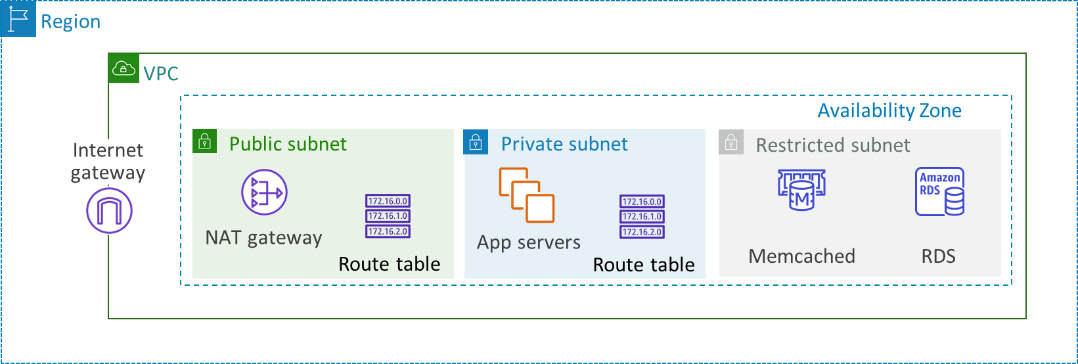
* + - * 1. Create and configure a *private route* *table*

Add to the private route table a route to direct internet-bound traffic (0.0.0.0/0) to the *NAT* device

Associate the private routing table to the private subnet



* + - As the optional *protected/restricted/data subnets* must not talk with the internet, they do not need an explicit route table.



* + - Best practices for CIDR *size* blocks:
      * Allocate substantially more IPs for private subnets than for others. E.g.:
        + VPC /16 à 65536 addresses
        + Public subnet /23 à 497 addresses available
        + Private subnet /19 à 8187 addresses available
        + Restricted subnet /24 à 246 addresses available
      * As you cannot increase (or decrease) the size of an existing CIDR block, consider larger ranges over smaller ones.
        + Use at least /24 subnets 🡪 251 addresses
        + Note: you can add a new CIDR block to an existing VPC and create new subnets using the new address range. But this make VPC management tough.
  + How to configure a layer of network security to *ALLOW traffic (whitelist)* not only by IP addresses (as with subnets and route tables), but also by *protocol* and port 🡪 Edit the *Security Groups* (SG) firewalls
    - Virtual firewalls at the *instance* (and network *port* or *instances group*) level
    - Stateful à Responses to allowed inbound traffic are allowed to flow out regardless of outbound rules, and vice versa.
    - Default values:
      * *Allow* *all* *outbound* traffic
      * *Block* *all* *inbound* traffic à Is *mandatory* explicitly *allow* the required *inbound* traffic
        + Any other inbound traffic is automatically blocked (except for outbound responses)
    - Only “allow” rules
      * Order is irrelevant
      * No “deny” rules
    - Example of a SG for an internet-facing web server, with inbound rules edited to allow HTTP and HTTPS traffic and to respond to ping requests. Outbound rules remain with their default settings.

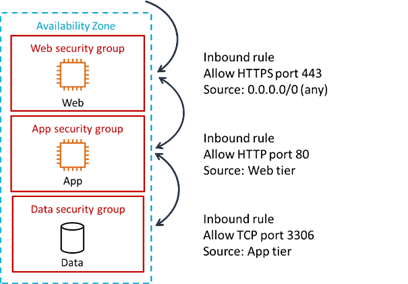
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

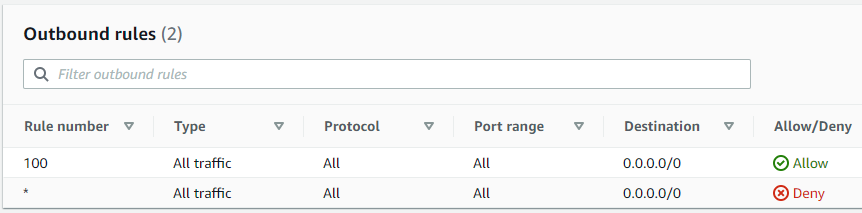
* + - Best practice: modifying the default outbound rule on Security Groups increases complexity and is not recommended unless required for compliance
  + Since the default routing table allows communications between all instances within the VPC, how to override this behavior to *control traffic* *between resources within the VPC*? 🡪 Create a layer of network security using Security Groups chaining to *implement a common three-tier web application model* that separates the architecture into Web/Presentation, Application and DB/Persistence/Data layers.
    - Optional layer
    - Independent of the subnets



* + How to add a layer of network security to *DENY (blacklist) explicit traffic* 🡪 Configure *Network Access Control Lists* (NACLs A.K.A. Network ACLs)
    - Are optional as AWS automatically deny (blocks) all traffic not explicitly allowed by Security Groups
    - Firewalls at the subnet level 🡪 Network ACL rules apply to all instances in the subnet
    - Stateless 🡪 Require explicit ordered rules for both inbound and outbound traffic
    - By default, allow all inbound and outbound traffic

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico, Teams

Descripción generada automáticamente



* + - Use custom NACLs to create exceptions to Security Groups allowed traffic
    - Best practices: 
      * Use custom NACL to *deny* traffic (such as temporarily under a DoS attack), not to allow it
      * Custom NACLs blocked traffic should be a subset of the Security Group allowed traffic range (as other inbound traffic is automatically blocked, except for outbound responses)
      * The result is a range of authorized IPs, protocols, and ports (Security Group ALLOW) minus a subset of exceptions (NACLs DENY)
      * E.g., a NACL edited to deny inbound traffic from North Korea

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico, Teams

Descripción generada automáticamente

* + - Note: *Web Application Firewall* (WAF) and *Shield* are Web ACL services that automatically detect and mitigate DoS and DDoS attacks.
  + **Note**: other advanced strategies are possible in the use of SGs and NACLs, such as:
    - Setting the same rules for SG and NACL but with separation of duties (network administrators manage NACLs and server administrators manage SGs)
    - Use fine-grained rules with NACLs and let SGs handle inter-VPC connectivity
    - “Reducing your attack surface”, e.g., restrict SGs outbound traffic
    - Using NACL as a backup layer of defense
    - Etc.

Compute: EC2 instances and Lambda

* EC2 (Elastic Compute Cloud) Instances
  + Virtual machines available in the Amazon cloud
  + Virtual Machines vs. Physical Servers
    - VMs can do anything a physical server can
    - VMs are completely isolated from each other
    - VMs are resizable
    - VMs are launched within minutes à Can be used as disposable resources
    - VMs are independent from the physical layer à Allow migration between physical servers
  + In addition to Virtual Machines, what are the options to compute in the AWS cloud?
    - Containers (Docker, ECS, EKS, Fargate)
    - Lambda functions
  + Why choose an EC2 instance to compute? à *Complete control* of your computing resources:
    - Can choose the Operating System, tools, applications, etc. used when instance boot à AMI
    - Can choose the server configuration that best suits your needs à Instance Type
    - Can update OS, load applications, and configure them running scripts at launch à Instance User Data
* What is an AMI (Amazon Machine Image)? à VM template including:
  + An image for the volume used to boot the instance (include Operating System, tools, applications, etc.).
  + Launch permissions that allow to share AMIs with other AWS accounts. Options:
    - Public
    - Explicit (shared with specific accounts)
    - Implicit (default, only the owner access)
  + A block device mapping that specifies optional EBS volumes to attach to the instance when it's launched
* Where do you get an AMI?
  + Pre-Built by AWS
  + AWS Marketplace
  + Community AMIs
  + Your own AMIs based on one of the previous three à *EC2 Image Builder* can help you, simplifying the creation, maintenance, validation, sharing, and deployment of Linux or Windows images for use with Amazon EC2 and on premises
* How to choose the EC2 instance configuration that best suits your needs? à Amazon EC2 instance types
  + Family, Generation, Size (vCPU count, RAM size, I/O limits)
  + AMI and Instance family dependent choices
    - CPU architecture à x86 Intel Xeon, x86 AMD Epyc, ARM AWS Graviton
    - Virtualization types à PV (legacy), HVM (Xen based), Nitro Systems (new generation, offload I/O virtualization, KVM based)
    - Best practice: upgrade to the latest generation.
      * Newer generations provide better functionalities and performance, which can result in using fewer or smaller instances.
      * Newer generations have a lower price per hour compared to earlier generations.
  + You can change a running instance type or size, but it requires a stop and restart.
  + *AWS Compute Optimizer* 🡪 Is an online tool to help you optimize your EC2 instances
    - Use machine learning to analyze *historical* (at least 12 hours) utilization metrics
    - You can reconfigure your instances based on new instance size and type recommendations to reduce costs (over-provisioned findings) or improve performance (under-provisioned findings).
  + How to automate OS update, load applications and configure them running scripts at launch? à Instance User Data
    - Script runs after the instance starts but before it becomes accessible on the network
    - By default, user data is executed only at launch.
      * If you stop an instance, modify the user data, and start the instance, the new user data is not executed automatically by default.
    - Can be:
      * Linux script 🡪 executed by *cloud-init*
      * Windows batch or PowerShell scripts 🡪 executed by *EC2Launch service*
    - Executed with root or Administrator privilege
    - Note: When use CloudFormation to deploy EC2 instances, *CloudFormation::Init* is a more powerful option
* Operating EC2 Instances
  + Metadata 🡪 How to use information about your instance itself?
    - How to *retrieve Instance Metadata more easily* than searching in Linux configuration files or in the Windows registry, or retrieve the public IP address (the public IPv4 address is not displayed using ifconfig or ipconfig)? à <http://169.254.169.254/latest/meta-data>
    - How to *assign your own metadata* to identify, filter, search, etc. your instances? à *Custom tags* are metadata labels (each with a customer-defined key and a value) that you should assign to resources to track or organize them into groups, such as by purpose, owner, or environment.
      * One use of tags is when you must manage many resources of the same type.
      * You can also quickly search for and identify a specific resource by the tag you have applied to it.
  + Tenancy 🡪 How to run your EC2 instances on hardware dedicated just for you?
    - By default, your EC2 instances runs on hardware *shared* with other accounts (A.K.A. “tenants”) à Multitenancy (A.K.A. shared tenancy)
    - But you can run your EC2 instances on hardware fully *dedicated* for your use (Dedicated Tenancy)
      * *Dedicated Instances* à Your EC2 instances run on hardware fully dedicated to your account
        + Intended to address the “noisy neighbors” fear
        + Additional fee of $ 1,440 per month per region, regardless of the number of dedicated instances
        + Legacy, as EC2 instances are completely isolated from each other
      * *Dedicated Host* à *One* physical server fully dedicated to your account where run instances (is not a “bare metal”)
        + Use cases

Allow you to *save money on licensing costs* by Bringing Your Own Licenses (BYOL) that are provided on a CPU/core basis (such as Microsoft, Oracle or RHEL licenses) à You are responsible for managing your own licenses, *AWS License Manager* can help you.

As a Dedicated Host is a physical server fully dedicated for your use, it can help address the “noisy neighbors” fear cheaper than Dedicated Instances (you pay for each whole server, as you are not charged for the Dedicated Instance $2 per hour per region fee). Legacy use as well as Dedicated Instances.

* + - * + Capacity

Each dedicated host can only support one instance type at a time (example, you cannot mix *R5* with *C5* instances on a single Dedicated Host)

The instance type will define the maximum *number of instances* and *instance sizes* that you can run on the Dedicated Host

Nitro-based instances allow to mix multiple EC2 instance *sizes* of the same type on a single Dedicated Host (previous instances cannot)

You are charged for the entire Dedicated Host, regardless of the number or the size of instances that you launch on it (you pay per host, and are responsible for waste)

* + How to streamline and simplify launching instances? 🡪 Launch templates
    - If you frequently create similar instances, or if you require a high degree of consistency among instances, you can record the parameters necessary to launch a new instance in a *Launch Template*.
    - Then, to launch a new instance, you can run the Launch Template instead of specifying each time the parameters, minimizing the risk of deployment errors, and saving a considerable amount of time.
    - Launch Templates support Auto Scaling, Spot Fleet, CloudFormation, Spot and On-Demand instances.
* EC2 Pricing Options 🡪 How to choose the right EC2 consumption model?
  + Reserved Instances
    - You “rent” an instance for 1- or 3-year term with a discount (purchases of RI are non-refundable)
    - Use cases:
      * Minimum instance capacity for scale-out applications à No more Reserved Instances than necessary at any time (when the RI does nothing, or even if the RI is turned off, you still pay for it)
      * DB instances à Long-term usage
    - Associated with a specific region. You never can change it.
    - RI can be applied to EC2, RDS, Redshift, ElastiCache, or Elasticsearch (now OpenSearch) instances
    - Payment options:
      * All Upfront Payment à Pay the whole contract upfront
      * No Upfront Payment à Make monthly payments
      * Partial Upfront Payment à Pay at least 50% upfront, rest monthly (interesting to burn up budget before the end of a fiscal period)
    - *If you select the Availability Zone*, RI also *reserve capacity* in this AZ.
    - Two purchase options:
      * Convertible RI
        + You can exchange Convertible RI for new Convertible RI with different instance type, operating system, tenancy (shared or dedicated), or payment option, as long as the change results in equal or greater value.
        + You can exchange multiple smaller Convertible RIs for one new bigger Convertible RI, and vice versa (only Linux).
      * Standard RI
        + You can only modify the instance size (within the same instance type) and the Availability Zone (within the same region)
        + If you reserved the wrong Standard RI or no longer need it, you can resell it in the Reserved Instance Marketplace, where you can also buy Standard RIs with shorter terms or lower prices
    - Can be shared between multiple accounts (within a billing family)
    - Discounts
      * Convertible 1-year: ~30%
      * Standard 1-year: ~40%
      * Convertible 3-year: ~50%
      * Standard 3-year: ~60%
      * Reaches 72% for Standard 3-year, all upfront payment
  + Savings Plans
    - Newer alternative to Reserved Instances à Greater flexibility with same savings.
    - It is a discount plan for a commitment to *spending a minimum dollar amount per hour* over a 1- or 3-year term. Any usage beyond that commitment will be charged at the usual On-Demand rates.
    - The main distinction between *RI* and *Savings Plans* is that the first commits *instances*, while the second commits an *expense* (dollars per hour).
    - Purchases of Savings Plans are non-refundable and *cannot* be resold on the AWS Reserved Instances Marketplace.
    - Same payment options as RI.
    - Two purchase options:
      * *Compute* Savings Plans à Just like *Convertible RI*, provide the most flexibility
        + You can modify EC2 instance’s parameters
        + Can change the Region (RI cannot)
        + Help to reduce your costs by up to 66%, like *Convertible* RI
      * *EC2 Instance* Savings Plans à Just like *Standard RI*, apply to a specific instance family
        + You can only modify the AZ (within a region), size, tenancy, and OS (new).
        + Provide the lowest prices, offering savings up to 72%, like *Standard* RI
    - Note: Savings Plans *does not reserve capacity* as Reserved Instances do if you select the Availability Zone. It is only a discount plan.
  + Spot Instances
    - AWS almost always has instances which are not being used, available at up to 90% off the On-Demand price.
    - But they can be interrupted or not available due to high demand.
      * This may sound scary at first, but it should not be a problem 🡪 You should “build for failures” and support “disposable” infrastructure.
      * Over 95% of the Spot Instances were not interrupted in the last 3 months
    - The hourly Spot price of each instance type in each Availability Zone is set by AWS.
      * The price is adjusted gradually, based on supply and demand.
    - Use cases:
      * Non-interactive processes with flexible start and end times that are only feasible at low instance prices
        + Examples: batch process, reporting, system "housekeeping", non-time-critical end-of-cycle processing, media transcoding and rendering, HPC, Machine Learning, Big Data analytics
        + Request Spot Instances by bidding on the maximum price that you are willing to pay per hour per instance

If the spot market price is below your bid price, then you should get the instance.

When the market price exceeds your bid, instance will be reclaimed and you get 2 minutes to wrap up your work à The *hibernate* feature allows you to pause EBS-backed instances, and automatically resume it when capacity is available again.

You will never pay more than your bid, and you can pay less

* + - * + AWS provides some resources to help you choose Spot Instances.:

Spot Instance Advisor 🡪 Helps you determine pools with the least chance of interruption and provides the savings you get over On-Demand rates

Interfaz de usuario gráfica

Descripción generada automáticamente

Spot Instance Pricing History 🡪 E.g., 3 months Spot Instance pricing history, just before black Friday 2021

Aplicación

Descripción generada automáticamente con confianza media

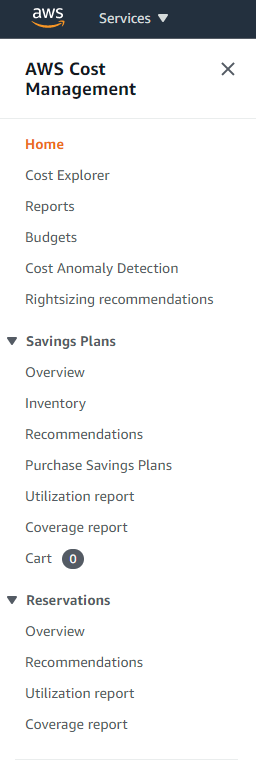
* + - * Scale-out Reserved Instances in *cloud native workloads* à Usually interactive, stateless, fault-tolerant, multi-AZ, loosely coupled
        + Examples: web servers, containers
        + Request Spot Instances by bidding a "virtual offer" (equal to the On-Demand price)

You should get the instance as long as capacity is available

You are unlikely to be interrupted by AWS

You still will be saving money because the spot market price will never be higher (and probably much lower!) than the On-Demand price.

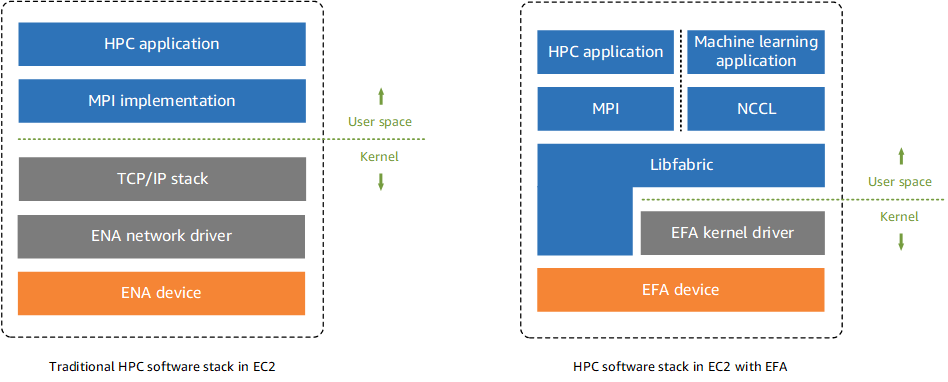
* + - Pay-As-You-Go
      * If AWS interrupt the instance in the first instance hour, you will not be charged for that usage. If AWS interrupt it after the first instance hour, you will be charged for your usage to the nearest second.
      * Otherwise, Spot Instance usages are billed for any time your instances are in a "running" state in the same way as the "on-demand" instances (by hour or second depending on OS, no long-term commitments)
  + On-Demand Instances
    - Simpler but you pay the most
    - Pay-As-You-Go
      * Instance usages are billed for any time your instances are in a "running" state, in fragments of:
        + Amazon Linux, Ubuntu, and beginning June 2021, Windows Server à Seconds, with a minimum of 60 seconds
        + Others à Hours
      * No long-term commitments, and no upfront payments (only monthly payments)
    - Use cases
      * If Spot Instances are interrupted or not available (or when you foresee high demand!)
      * Amazon Linux, Ubuntu or Windows Server short-term applications (more agile than looking for a spot instance for a few minutes of use as you pay them by seconds)
      * Applications that must not be interrupted
      * Applications that must start at a precise time and you do not know their duration (otherwise you could use cheaper *Spot Blocks*)
    - During periods of exceptionally high demand, you might not be able to get an instance à Foresee *Capacity Reservation*.
  + You can easily combine Spot Instances, On-Demand, Savings Plans and RI to further optimize workload cost with elasticity and availability.
  + *Cost Explorer* tool available in the *Billing and Cost Management* service helps you identify cost-saving opportunities
    - *Rightsizing recommendations* optionprovides potential savings by downsizing instances or terminating idle instances
    - *Saving Plans 🡪 Recommendations* option provides customized Savings Plans recommendations based on your past usage
    - *Reservation 🡪 Recommendations* option provides recommendations about the potential savings available through the use of Reserved Instances (RI)
    - Note: The Billing and Cost Management service is not available for Admin members. Access must be explicitly granted by the *root account*



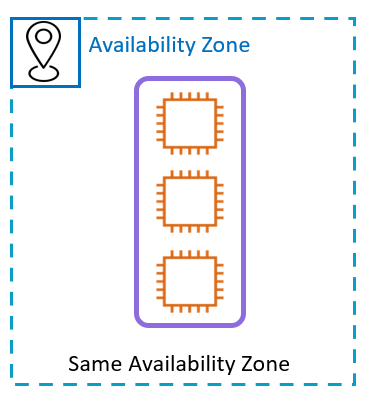
* Block Storage
  + For applications running on EC2 instances that require block access (such as RDS databases or boot volumes)
  + Two options:
    - Instance store 🡪 Directly attached storage (DAS)
      * Disks physically attached to the server.
      * Ephemeral: data survives reboots but does not survive stopping the instance.
      * Only some instance types
      * Disk type (HDD, SDD, NVMe) and capacity depends on instance type
      * Do not support snapshots to S3
      * Use case
        + Temporary data, buffers, and caches.
        + Instances for distributed storage (such as Aerospike or GPFS)
        + Ultra-high storage performance, especially using “i” instance types (up to 8 NVMe SSD)
    - EBS (Elastic Block Storage) 🡪 Network-attached block storage
      * Persistent
      * Thick provisioned (pay for the requested space: used and free space)
      * Up to 16 TB
      * EBS must be in the same Availability Zone as the instance
      * Data is automatically replicated *within the AZ* and any failed components are automatically replaced behind the scenes, transparently
      * Should configure *snapshots* to S3 for protection against AZ failure or data corruption (*Data* *Lifecycle Manager* or *AWS Backup* can help you)
      * In some cases, can be used for clustering, multi-attaching (sharing) the EBS volume between up to 16 instances. Requires specific *Nitro Systems* instances, *Provisioned IOPS* volumes and cluster file systems as *Red Hat Global File System 2* (GFS2)
      * How to choose the right EBS type?
        + Lowest cost per TB, but slower than other options à *Cold* HDD (sc1), cannot be a boot or RDS volume
        + Lowest cost per MB/s (sequential access), but limited to 500 MB/s max. à *Throughput Optimized* HDD (st1), cannot be a boot or RDS volume
        + Lowest cost per IOPS (random access), but limited to 16K IOPS max. à *General Purpose* SSD (gp2 and new gp3)
        + Highest IOPS (up to 64K IOPS), MB/s (up to 1 GB/s), performance consistency, and maximum performance with small capacities, but always more expensive 🡪 *Provisioned IOPS* SSD (io1 and new io2)
        + You can modify the EBS volume type, capacity, and performance online (but it may take a while to complete). After modifying, wait at least 6 hours before modifying again.

When you increase the size of an EBS volume, you must also extend the file system at the OS level.

* + - * Note: *io2 Block Express* is the next generation EBS (beginning July 2021) with up to:
        + 64 TB
        + 256 000 IOPS
        + 4GB/s
        + Sub-millisecond, low-variance I/O latency
* Services and functionalities available for HPC (High-Performance Computing)
  + AWS Batch
    - Fully managed service that plans, schedules, and executes your batch jobs without manual intervention
    - Can dynamically provisions the optimal quantity and type of Instances On-Demand, Spot Instances, or Fargate serverless containers.
    - Only pay for the resources used. When capacity is no longer needed, AWS Batch will remove it.
    - Offers queues where you send the jobs. Each queue could be configured with a certain priority so you can configure which jobs will run first
    - Optimized for applications that scale with the number of jobs running in parallel, as HPC applications
  + AWS ParallelCluster
    - Infrastructure as Code for HPC environment in AWS
    - Built on the open source *CfnCluster* project
  + Amazon FSx for Lustre file systems
    - Fully managed Lustre storage
    - Lustre is an open-source parallel filesystem for HPC environments
    - Massively scalable performance
    - Seamless data swap with S3
  + Enhanced networking 🡪 The goal is to ensure that the network is not the bottleneck in a HPC environment.
    - ENA (Elastic Network Adapter)
      * High-performance networking capabilities on supported instance types
      * Uses SR-IOV (single-root IO virtualization) 🡪 A specification allowing a EC2 instance to bypass the underlying hypervisor and use direct memory access (DMA) instead of interrupting the CPU
      * Up to 100GbE
    - EFA (Elastic Fabric Adapter)
      * Enables customers to run applications requiring high levels of inter-node communications at scale.
      * Is an ENA with the functionality to use the *Libfabric* API to:
        + Bypass the operating system kernel
        + Replace the TCP/IP stack
      * To interface with the Libfabric API:
        + HPC applications uses *Intel MPI* (Message Passing Interface)
        + Machine Learning applications uses *Nvidia NCCL* (Collective Communications Library)
      * Up to 400Gbps



* + How to control where your instances are launched? à Placement Groups
    - By default, *AWS* choose where (server, rack, datacenter) an EC2 instance is launched
    - But *you* can control where your instances are placed using *Placements Groups*
      * How to provide the best node-to-node network performance to a *scale-out cluster* (such High-Performance Computing)? à *Cluster* Placement Groups (only support specific Instance Types) help us to launch a bunch of EC2 instances close to each other.
        + Similar principle to VMware ESXi *Affinity*.



* + - * How to minimize the risk of simultaneous failures in a *small group* of *critical instances* (such as MS Domain Controllers)? 🡪 *Spread* Placement Groups launch them across distinct racks, with each rack having its own network and power source, so that a hardware failure impacts only one node
        + Similar principle to VMware ESXi *Anti-affinity*.
        + You can have a maximum of seven running instances per AZ per Spread Placement Group.
        + Typically legacy. Used to be useful when there were Regions that only had one AZ, but now you probably have more control by distributing those instances across the multiple AZs available in a Region.

Imagen que contiene Forma

Descripción generada automáticamente

* + - * How to distribute evenly a *large number* of *distributed systems* instances (such as Aerospike or IBM GPFS) across distinct partitions (racks) to reduce the risk of simultaneous failures? à *Partition* Placement Group
        + AWS tries to distribute the instances evenly across the number of partitions that you specify (*multiple* instances per rack, up to seven *racks* per AZ, in *multiple AZ* in the same Region)

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* AWS Lambda
  + What is a serverless platform?
    - A pool of prewarmed servers already configured by AWS are waiting for you
      * “Serverless” doesn’t mean “no server” 🡪 The service is still running on servers
      * Don't require you to provision, scale, maintain, or manage any servers
      * Stop thinking about servers!
    - Event-driven
      * You no longer have to build or maintain server-based polling infrastructure to check for changes.
      * Instead of paying for uptime (including idle time), as with EC2 Instances or RDS over EBS, you will pay for:
        + *execution time* as with *Lambda* or *Fargate*
        + *number of requests* as with *SQS* or *DynamoDB On-Demand*
      * Automatically connect, run, and disconnect service.
      * Do not pay for idle time.
    - Do not operate within a VPC infrastructure.
      * Built-in high availability and fault tolerance.
      * VPC-based resources can access serverless services by an *VPC Endpoint*.
    - Some serverless services à Aurora Serverless (relational database), DynamoDB On-Demand (NoSQL database), Fargate (containers), Lambda (stateless code), EFS (NFS storage).
  + What is AWS Lambda? 🡪 The AWS computing service to run *stateless* *code* serverless.
    - Runs on a schedule or in response to events, like other serverless services.
    - Accepts Node.js, Java, Go, C#, Python, Ruby, and PowerShell Core code.
    - You choose the amount of RAM up to 10GB (CPU and network allocated. proportionately)
    - Only pay for the compute time you consume per 1/1000 second (beginning Dec. 2020). There is no charge when your code is not running.
    - The default timeout is 3 seconds but can be set to up to 15 minutes long.
    - Can run at the edge (only support Node.js and Python)
    - Use case: Microservices architectures, respond to events, etc.

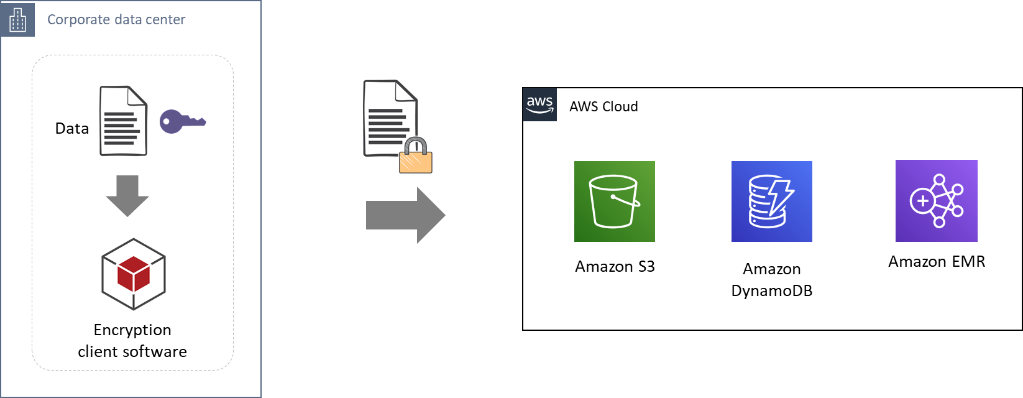
Storage: Object and File Storage

* Object storage
  + Amazon S3 (Simple Storage Service)
    - “New” storage for the internet
    - The “de facto” object storage since 2006
  + What problems does object storage help you to solve?
    - Web Access
      * Accessible from anywhere in the world, via a URL over HTTP or HTTPS
      * Do not use a client (as required with NFS or SMB) or driver (as required with iSCSI or FCP)
    - Access Control
      * Private by default
      * But access can also be shared securely using IAM policies
    - Scalability
      * Virtually unlimited capacity
      * Flat namespace (not a file system)
      * Object located using *object metadata* instead a of directory path 🡪 Object key, object tags, and optionally, a version.
    - Immutability
      * If you want to change a part of a file, you have to make the change and then re-upload the entire modified file.
      * This behavior allows *versioning*, a safeguard against application data corruption, malicious or accidental deletion, or object overwrite
  + S3 advantages
    - *Serverless* storage service: fully managed, trigger events, etc.
    - Secure
      * HTTPS for data in transit
      * AES-256 encryption for data at rest
    - Fast
      * Milliseconds access delay and high throughput
      * Spread load evenly so that no individual application is affected by traffic spikes
    - Highly available
      * Designed for 99.99% availability over a given year
      * Backed with the Amazon S3 SLA
    - Highly durable
      * Data is redundantly stored across a minimum of 3 AZ (except for *S3 One-Zone IA*).
      * Use a combination of content-MD5 checksums and CRCs (Cyclic Redundancy Checks) to self-healing and avoid data corruption.
      * 11 “nines” 🡪 You only risk losing 1 object in 1 billion (1000 millions) objects every 100 years
    - Inexpensive ($23 - $4 @TB @month)
  + S3 is not just a storage bucket. More functionalities and use cases:
    - An S3 bucket can act as an elastic web server hosting static content, with *cross-origin resource sharing* (CORS) support.
    - "Query in Place" à Use S3 as “data lake” without needing to load it into a separate analytics platform or data warehouse. Can directly use S3 Select, Athena, Redshift Spectrum, etc.
  + Object storage organization
    - Object (data piece) à Up to 5TB
    - Bucket (data container) 🡪 Unlimited object number and bucket size
  + How to upload data into S3? à Web REST API (Console, CLI, DSK)
    - Using the network
      * Object size from 0 to 5GB à PUT in a single operation: CLI, web console or SDK
      * Object size from 100MB (best practice) to 5TB à Program *Multipart Upload* to upload files in parts: CLI or SDK (not web console support)
      * Other AWS services (as *Storage Gateway*, *DataSync*) and 3rd party clients can also be used.
      * *S3 Transfer Acceleration* move data faster over long distances routing S3 traffic through the closest *Edge Location*.
    - Using a courier to bypass internet à Snowball devices family
  + S3 costs: what is paid and what is free?
    - You do NOT have to pay for:
      * Transfer IN S3
      * Transfer OUT to another service in the *same region*
    - Pay only for what you use
      * Stored GBs per month (thin provisioning)
      * Data retrieval pricing (free for S3 Standard and S3 Intelligent-Tiering)
      * Data transfer OUT to other regions or the internet
      * Amazon S3 Transfer Acceleration pricing
      * PUT, COPY, POST, LIST, and GET requests
      * Data management (Lifecycle policies transitions and S3 Intelligent-Tiering monitoring)
      * Analytics pricing
      * Amazon S3 Replication pricing
      * The price to process your data with S3 Object Lambda
  + When to use other storage options (block, file, off-line) instead of S3?
    - Shared access with locking requirements (S3: “last write wins”) à EFS, FSx support strong consistency and locking
    - Block storage requirements (boot EC2 instances, RDS databases, etc.) à Use EBS
    - Frequently changing data à Better EBS, EFS, FSx
    - Long-term archival storage à Glacier may be cheaper.
  + How to easily ensure that an S3 bucket is not public by mistake, including those that you upload in the future? 🡪 *S3 Block Public Access*
    - Newly created S3 buckets are private by default. The only way S3 buckets become public unintentionally is by customer misconfiguration. S3 customers can use multiple mechanisms for controlling access to resources: IAM policies, bucket policies, ACLs, Query String Authentication, CloudFront distributions, Access Points, Object ACLs, and your own custom application code. This makes it easy to makes mistakes that lead to data leakage.
    - S3 Block Public Access overrides any mechanism that might allow public access
    - Can be set at the:
      * Account level
      * Individual bucket level
    - Provides four settings:
      * Block public access to buckets and objects granted through new access control lists (ACLs)
      * Block public access to buckets and objects granted through any access control lists (ACLs
      * Block public access to buckets and objects granted through new public bucket or access point policies
      * Block public and cross-account access to buckets and objects through any public bucket or access point policies
    - These settings are independent and can be used together (*Block all public access*) or in any combination. In the latter case, Amazon S3 applies the most restrictive combination.

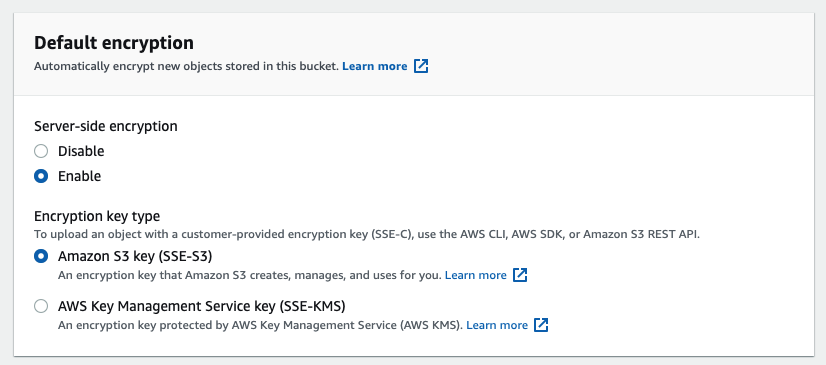
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

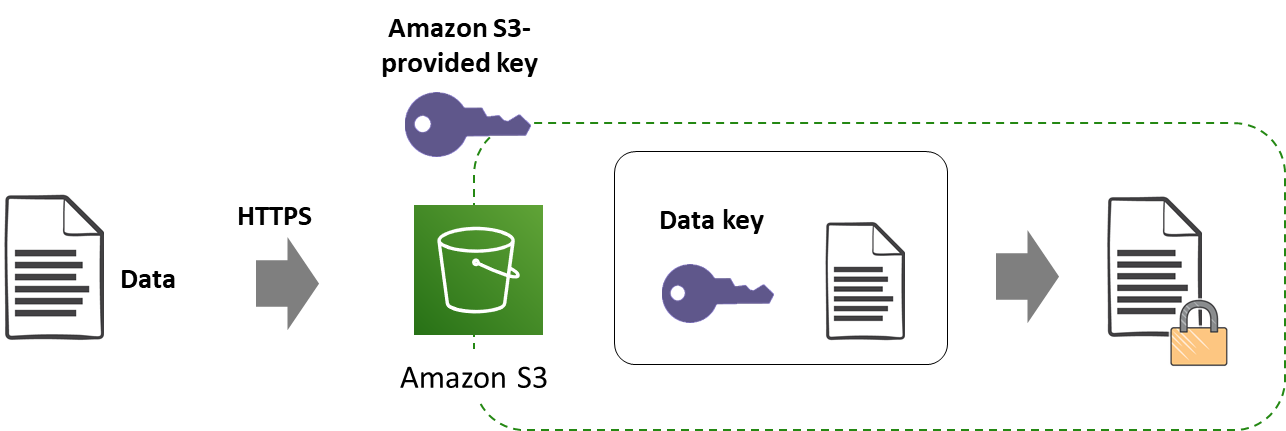
* + - Due to the many ways to make S3 buckets and objects public, AWS continues to iterate on services and features to aid customers discover public buckets, such as:
      * IAM Access Analyzer
      * Trusted Advisor
      * GuardDuty
      * Amazon Inspector
      * AWS Config
      * Etc.
* S3 Glacier à Off-line (A.K.A. archive, long-term, cold) data storage
  + Glacier storage characteristics
    - Off-line
      * To access an archive, you must download it first (like FTP or tape)
      * Retrieval latency from minutes to up two days, depending on retrieval speed options and tier
      * The longer you can wait, the lower the retrieval costs
    - Long-term
      * S3 Glacier Flexible Retrieval (formerly S3 Glacier “plain”) à Bill min. 90 days
      * S3 Glacier Deep Archive à Bill min. 180 days
  + Store archives in an extremely durable (11 nines), highly available (99,99%), and infinitely scalable way
  + Very low-cost storage cost (as low as $0.99 @TB @Month) but expensive retrieval (up to $30 @TB)
  + Glacier storage organization
    - Archive (data piece) à is not an object (not metadata, up to 40TB vs. 5TB, no share, no versioning)
    - Vault (data container) à is not a bucket (no share, no website)
  + How to interact with Glacier?
    - How to add data into Glacier?
      * Usually, you’ll use *S3 Lifecycle Policies* or *S3 Intelligent-Tiering* to transition S3 “cool” objects into Glacier
      * You cannot add data to Glacier directly using the AWS management console.
      * To directly add files up to 4 GB use:
        + CLI command upload-archive
        + SDK and the S3 PUT API
      * To directly add files up to 40 TB use
        + CLI commands initiate-multipart-upload and then complete-multipart-upload
        + SDK and the multipart upload API
      * Can also use other AWS services (as DataSync) and 3rd party clients (as Commvault).
    - How to retrieve data from Glacier?
      * Using the AWS web console or S3 API, indirectly from Glacier to S3 Infrequent Access
      * Retrieving directly an archive from Glacier is an asynchronous operation in which you first initiate a job, and then download the output after the job completes.
        + CLI commands initiate-job and then get-job-output
        + SDK and the Glacier's REST API
      * Other AWS services (as DataSync) and 3rd party clients (as Commvault) can also be used.
* What options do I have for encrypting data stored on Amazon S3?
  + *Client-side* encryption (CSE) 🡪 *You* encrypt your data *before* sending it to AWS, and decrypt data after receiving it from AWS.
    - *You* create, store, and manage the encryption keys and encrypt your data with algorithms known only to you. AWS receives your encrypted data; it does not play a role in encrypting or decrypting it.
    - In addition to being protected *at-rest*, data is also protected while *in-transit* (while traveling to and from AWS)
    - Can use the *Amazon S3 Encryption Client,* the *AWS Encryption SDK* or an encryption library of your choice.
    - Can use a key stored in AWS *Key Management Service* (AWS KMS), in a *on-premises key management infrastructure* (KMI), or a key that you store within your application.
    - You must protect your encryption keys: if you lose them, you won't be able to decrypt your data.
    - Use case: When you prefer full end-to-end control of the encryption and decryption of your data or need to encrypt it before sending it to AWS for storage.



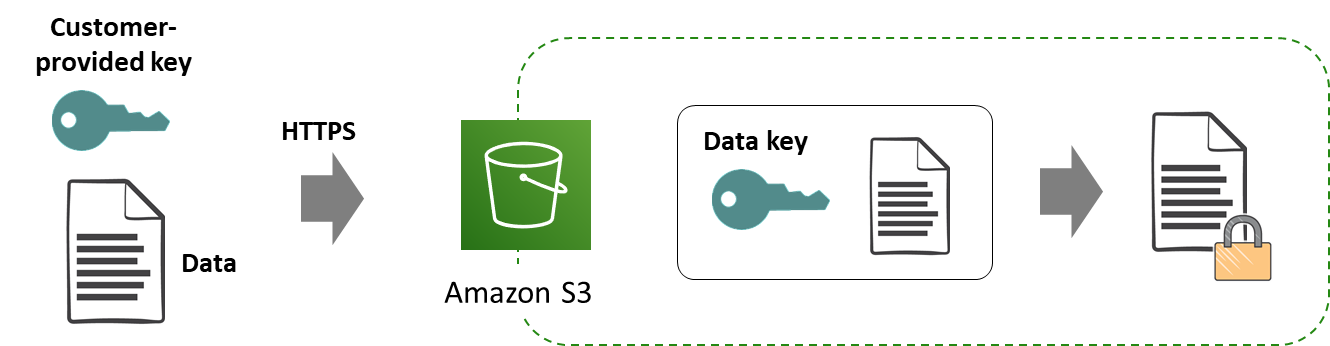
* + *Server-side* encryption (SSE) 🡪 *AWS* encrypts data *after* it has been received (when it writes it to disks in its data centers and decrypts it when you access it).
    - Uses 256-bit Advanced Encryption Standard (AES-256) to encrypt your data
    - You can encrypt
      * An S3 *object* on upload simply by adding an additional request header when writing the object
      * An S3 *bucket,* so *all new* objects are encrypted when stored in the bucket (not available for SSE-C).



* + - Note that metadata, which you can include with your object, is not encrypted. Therefore, AWS recommends that customers do not place sensitive information in S3 metadata
    - SSE options are:
      * SSE-S3
      * SSE-C
      * SSE-KMS
    - SSE with Amazon *S3-managed* keys (SSE-S3)
      * AWS handles key management, rotation, and protection
      * AWS store the encrypted data and encryption keys in different hosts
      * You cannot view, use, track, or audit AWS owned keys.
      * Each object is encrypted with a unique key. As an additional safeguard, it encrypts the key itself with a primary key that it regularly rotates.
      * SSE-S3 is automatically used with Glacier.
      * There are no additional fees for using SSE-S3.
      * Use case: when you prefer that Amazon take full care of the encryption.



* + - SSE with *customer-provided* keys (SSE-C)
      * *You* manage the encryption keys that you provide to AWS as part of your request and *Amazon* perform the encryption and decryption of data you store in AWS.
      * Therefore, you don't need to maintain any code to perform data encryption and decryption. The only thing you do is manage the encryption keys you provide.
      * AWS does not store the encryption key you provide. That means if you lose the encryption key, you lose the object.
      * There are no additional fees for using SSE-C.
      * Optional to S3, not available to Glacier.
      * Use case: when you want to maintain your own encryption keys, but don’t want to implement or leverage a client-side encryption library.

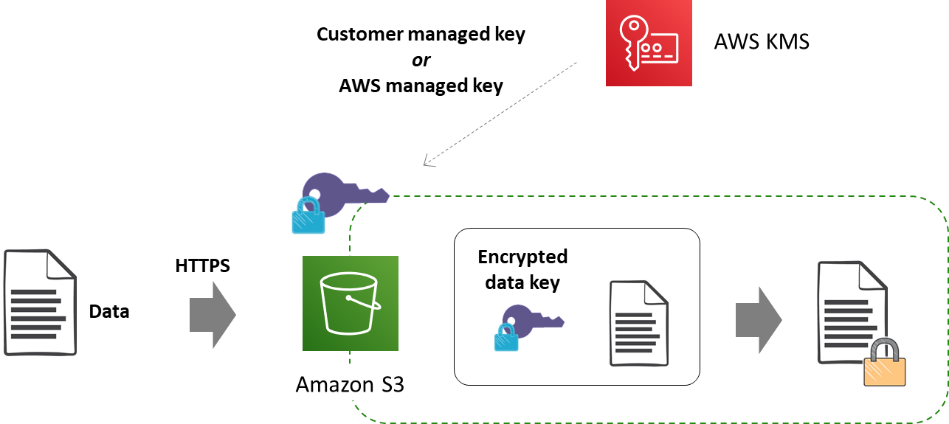


* + - SSE with KMS *keys stored in AWS KMS* (SSE-KMS)
      * SSE-KMS is similar to SSE-S3, but using AWS Key Management Service (AWS KMS) to manage your keys provides additional benefits (and KMS charges)
      * AWS KMS is a managed service that enables you to create or import, securely store, and manage the keys used to *encrypt your data*.
      * Can use the default *AWS managed key* or a *customer managed key*.
      * KMS gives you more flexibility and control:
        + You can disable customer managed keys.

If you disable access to the encryption key, it is technically impossible for AWS to decrypt the data.

Allows you to stop ongoing exfiltration of your data.

* + - * + There are separate permissions for the use of the master key, providing an additional layer of control as well as protection against unauthorized access to your stored data.
        + Provides an audit trail so you can see who used your key to access which object and when, as well as view failed attempts to access data from users without permission to decrypt the data.
      * *AWS managed keys* are automatically rotated by KMS every 3 years, you can configure KMS to automatically rotate *customer managed* keys every year.
      * SSE-KMS is also supported by multiple services as EBS, RDS, Redshift, DynamoDB, EFS, etc.
      * Note: AWS KMS is replacing the term *customer master key* (CMK) with KMS key. The concept has not changed.



* S3 and Glacier costs:
  + Storage and access costs vary depending on *storage class* (tier)
    - S3 Standard *always works*, but it *may be* the most expensive option
    - Other storage classes *may be* cheaper à Cheaper the GB, more expensive the access
  + Many storage classes (cost and associated retrieval speed) within a same S3 bucket for each *object*
    - S3 Standard à “Hot” object
    - S3 Infrequent Access à “Cool” object
    - S3 One-Zone IA à *Reduced Redundancy* Storage “cool” object
    - S3 Glacier Instant Retrieval 🡪 “Cold” object with *instant* retrieval
    - S3 Glacier Flexible Retrieval (previously Glacier “plain”) à “Cold” archive with *expedited*, *standard,* or *bulk* archive retrieval options
    - S3 Glacier Deep Archive à “Frozen” archive with *standard* or bulk *archive* retrieval options
  + How to optimize your costs?
    - Choosing *manually* a storage class (tier)
    - *Automatically* transition data across storage classes or delete “old” objects (*Lifecycle policies* A.K.A. tiering)
    - Using S3 *Intelligent Tiering*
  + Choose *manually* the storage class (tier) when creating the object, based on:
    - Access frequency
      * S3 Standard
        + No minimum duration charges
        + Free access
      * All the others
        + Duration charges à Minimum 1-, 3- or 6-months depending on the class

After the minimum, duration costs are calculated based on the average daily bytes stored, and are billed monthly

* + - * + Retrieval cost @ MB (except for S3 Glacier Flexible Retrieval whose recovery in Bulk mode is now free)
    - Access delay
      * Online à Milliseconds
      * Offline 🡪 From minutes up to 48 hours.
        + Choose the speed (and cost) for each retrieval.
    - AZ redundancy à S3 *One Zone* IA exception
      * Data is stored in a single availability zone. If the AZ is destroyed, you lose your *S3 One Zone IA* bucket.
      * For data that can be easily recreated *and* infrequently accessed.
      * *S3 One-Zone IA* is cheaper than S3 “Standard” IA ($10.00 @TB vs. $12.50 @TB monthly storage cost).
  + **Note**: Beginning Nov 30, 2021, a new storage class, *Amazon S3 Glacier Instant Retrieval* (Glacier IR) is available. The name can be misleading, because unlike the other Glacier classes that have an offline access, this one has *online* access (milliseconds Time For First Byte), just like the other "standard" S3 classes.

Tabla

Descripción generada automáticamente con confianza baja

* + *Automatically* transition data across storage classes or delete “old” data 🡪 Use *Lifecycle policies* (Tiering)
    - Allows you automatically:
      * *Transition* data to a “colder” storage class tiers (one-way) based on *age* (x days after creation or after becoming noncurrent).
      * *Expire* data based on *age*
      * Beginning November 2021, you also can:
        + Limit the *number* of old (noncurrent) versions of an object

E.g., deleting old (noncurrent) versions of an object after 5 days and when there are at least 2 newer versions of the object.

* + - * + Transition objects to other storage classes based on *size*

E.g., move only large media files to S3 Glacier.

* + - * Note: before you are transitioning objects to S3 Standard-IA or S3 One Zone-IA, you must store them at least 30 days in the S3 Standard storage class. But shorter lifecycle policies to Glacier (include zero-day) are allowed.
    - Use case:
      * For data with a defined lifecycle
      * *You* decide when
      * Much control, but you also pay for storage class *transition* and *access*
    - Tiering costs:
      * Lifecycle transition requests to (per million objects transitioned):
        + Infrequent Access à $10
        + Glacier IR à $20
        + Glacier FR à $30
        + Glacier Deep Archive à $50
  + How to use tiering if the data has unknown or changing access patterns? How do I avoid bad surprises with access costs? à Use *S3 Intelligent-Tiering* storage class
    - S3 Intelligent-Tiering storage class transition objects between S3 *Standard,* S3 *Standard IA* and *Glacier IR* tiers based on *access*
    - Objects are transitioned to *S3 IA* if has not been accessed for 30 consecutive days, and then to *Glacier IR* after additional 60 days
      * If you access objects transitioned to S3 IA or Glacier IR, these objects are first moved back to S3 Standard,
        + so they will be accessed free of charge,
        + but will pay the *S3* *Standard storage* monthly fee again.
      * Beginning Nov. 2020, can also configure S3 Intelligent-Tiering to archive objects to Glacier FR and Glacier Deep Archive
    - Use case:
      * For data with unknown or changing access patterns
      * Simplest
    - Predictable maximum cost
      * When stored in S3 Standard à $23 per TB @ month
      * When stored in S3 IA à $12.50 per TB @ month
        + And so on for Glacier storage classes.
      * No transitioning fees
      * No retrieval fee, except $30 @ TB for expedited retrieval (but S3 Std. tier storage cost apply again for this object for at least 30 days)
      * No minimum duration fee
      * *But* *must add* $1 per million objects stored
  + *S3-Intelligent Tiering* storage class can be combined with *lifecycle policies*, so you could even configure rules to keep objects in Intelligent Tiering for a few months and finally delete them.
* File Storage
  + Shared folders with file locking 🡪 File system avoid conflicts when many users or applications try to update a file simultaneously
  + Fully managed services
  + Thin provisioned (pay only for the space you use)
  + File systems are distributed across an unconstrained number of storage servers across AZs:
    - Highly available 🡪 4 “nines” SLA
    - Durable 🡪 11 “nines”
    - Scalable 🡪 PB of data can be shared with up to tens of thousands of concurrent clients
    - Fast 🡪 massively parallel access with consistent performance, low latency, and multiple GB/s and hundreds of thousands of IOPS per file system
  + Support:
    - Lambda
    - Containers
    - EC2 instances
    - On-premises servers when connected with *AWS Direct Connect*, *AWS site-to-site VPN* or *Amazon File Gateway*
  + Options:
    - NFSv4 for Linux à EFS (only supports Linux)
    - SMB à FSx for Windows File Server (is accessible from Windows, Linux using the *cifs-utils* tool, and MacOS)
    - Lustre (parallel file system for HPC) à FSx for Lustre (some differences with EFS and FSx for Windows)
    - New file systems are available for FSx:
      * FSx for NetApp ONTAP (Sep 2, 2021)
      * FSx for Open ZFS (Nov 30, 2021)
* Data Migration Tools
  + Offline 🡪 AWS *Snow* devices family
    - Is a service that provides secure, rugged devices, so you can bring AWS computing and storage capabilities to your edge environments, and transfer data into and out of AWS.
    - Only available within a Region
    - Include on-board computing capabilities as well as storage.
      * *Snowcone* 🡪 Small size, up to 14TB usable
      * *Snowball Edge* 🡪 Medium size, up to 80TB usable (can be clustered up to 15 devices for up to 1.2PB capacity).
      * *Snowmobile* 🡪 45-foot container pulled by a semi-trailer truck, up to exabyte
    - Use case: when you need to transfer data in large batches or as a one-time transfer and you anticipate that the data transfer will take a week or more to complete through the Internet or DX connection
  + Online
    - *DataSync* is an online data transfer service that simplifies, automates, and accelerates copying large amounts of data
      * between:
        + on-premises storage systems and AWS Storage services,
        + AWS Storage services.
    - The *DataSync software agent* allow read files or full and incremental writes
      * From *on-premises storage* as:
        + Network File System (NFS)
        + Server Message Block (SMB) file servers
        + Hadoop Distributed File Systems (HDFS)
        + Self-managed object storage
        + AWS Snowcone
      * To:
        + EFS
        + FSx for Windows File Server
        + S3
        + Glacier (directly)

Diagrama

Descripción generada automáticamente

* + - You can use AWS DataSync console, AWS CLI, or AWS SDK to transfer files or objects between AWS services without deploying a *DataSync agent*



* + - * By default, data is encrypted in transit using Transport Layer Security (TLS) 1.2.
      * Automatically handles scripting copy jobs, scheduling and monitoring transfers, validating data, and optimizing network usage.
    - AWS Transfer Family
      * Transfer files into and out of S3 with SFTP protocol
    - AWS Storage Gateway
      * Is a hybrid cloud storage service that gives you on-premises applications access to virtually unlimited cloud storage with low-latency:
        + Maintains a *cache* of recently written or read data so your on-premises applications can have low-latency access to data that is stored durably in AWS
        + Seamlessly *connect* on-premises applications with AWS cloud storage using SMB, NFS, iSCSI or Virtual Tapes (as directly access a S3 bucket from a NFS Linux mount)
        + You deploy it on-premises, using:

a *virtual machine* containing the Storage Gateway software on VMware ESXi, Microsoft Hyper-V, or Linux KVM

an AWS Storage Gateway *Hardware Appliance* pre-loaded with Storage Gateway software

* + - * Offers four different types of gateways
        + *S3 File* Gateway

Present *S3* storage as local SMB 2 or SMB 3 shares, and NFS v3 or NFS v4.1 exports

Access those *files* stored in AWS from your data center or directly in AWS

Access those files as *objects* directly in Amazon S3

* + - * + *FSx File* Gateway

Present *FSx for Windows* storage as local SMB 2 or SMB 3 shares

Access those files stored in FSx from your data center or directly in AWS

* + - * + *Volume* Gateway

Present *S3* storage as local iSCSI LUNs (networked block volume) to your on-premises hosts

You can take snapshots of your cloud-backed block volumes so EC2 instances can access those EBS snapshots directly

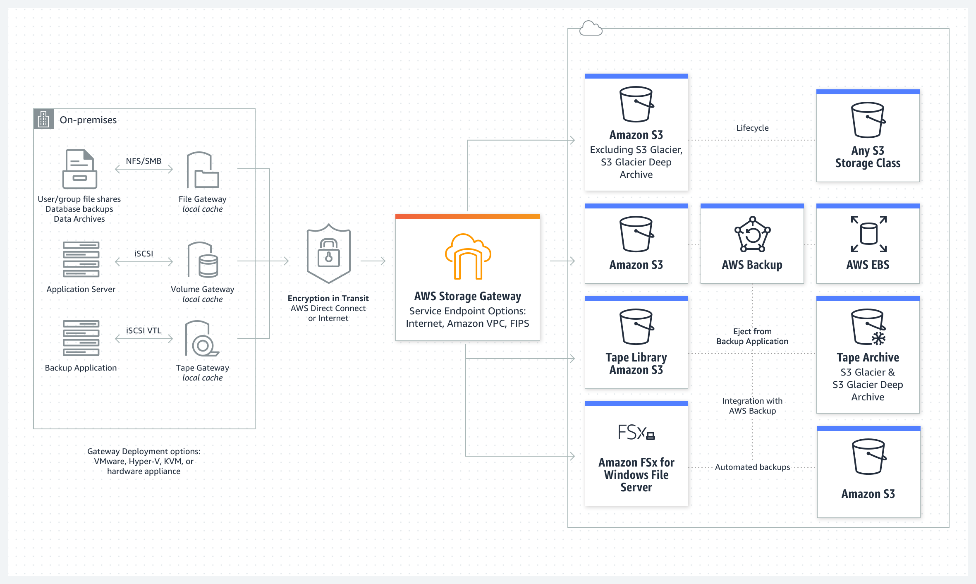
As other Storage Gateways, it can retain your frequently accessed data locally in the cache for low latency access (*cache* mode), but also can store locally your *entire dataset* to be available for low latency access (like a local SAN LUN) while also asynchronously getting backed up to S3 (*stored* mode).

* + - * + *Tape* Gateway

Presents an iSCSI Virtual Tape Library to a on-premises server running backup software (as Veritas NBU, Dell/EMC Networker, Commvault, etc.)

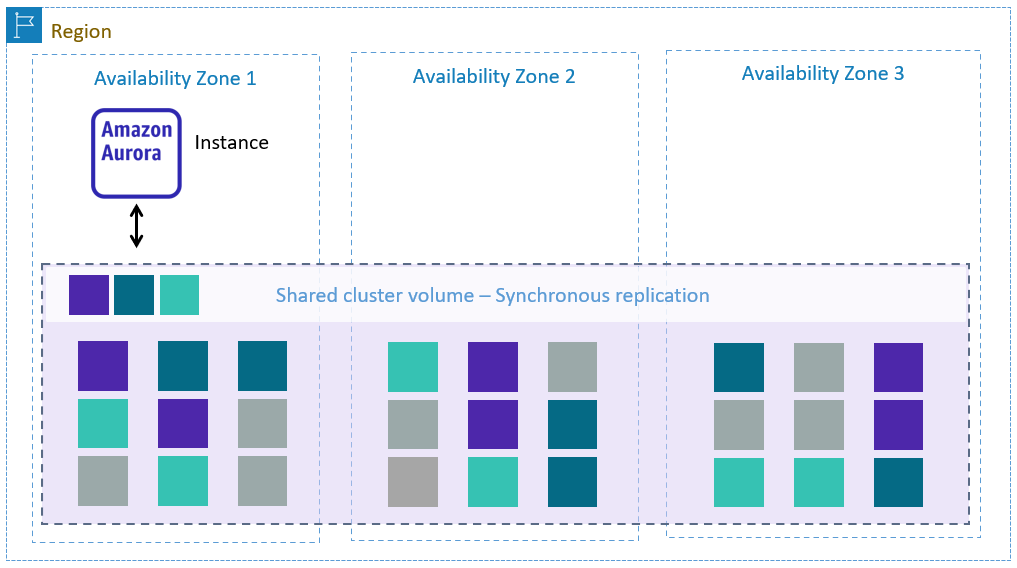
Backup is done by writing virtual tapes to *S3*

Only allow restore virtual tape from S3 to on-premises

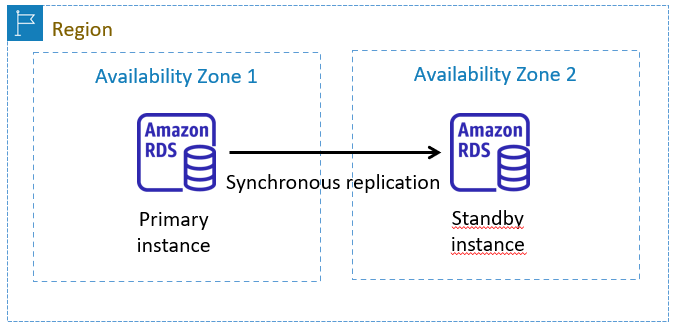


Database Services: What is the right database for your needs?

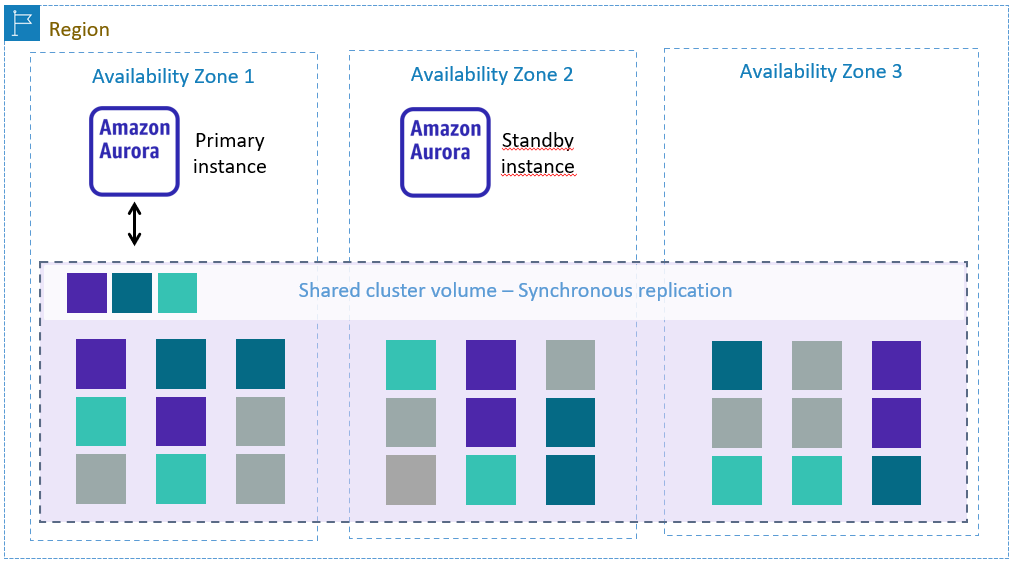
* Who manages which parts? à AWS Shared Responsibility Model
  + AWS infrastructure services (1st level) 🡪 Database hosted in EC2 as IBM DB2 or Couch DB
    - Customer responsibility à same as EC2 instance
  + AWS container services (2nd level) 🡪 E.g., RDS over EBS
    - Customer responsibility à client-side and server-side encryption, scaling, HA, data access permissions
  + AWS abstract services (3rd level), as S3, EFS, etc. 🡪 E.g., DynamoDB On-Demand
    - Customer responsibility à client-side encryption and data access permissions
* How to choose a database?
  + What code, applications, tools, licenses, and knowhow you already use today? Can you change the database?
  + Data requirements: size, type
  + Relational database or NoSQL database?
  + No one size fits all
* Relational databases or NoSQL databases: What should you consider to make the right choice?
  + Scale-Up (vertical) vs. Scale-Out (horizontal) 🡪 Performance and capacity
  + Normalized schema vs. schema-less
  + NoSQL is not a replacement of relational databases
* Amazon RDS (Relational Database Service)
  + Is a managed service that makes easier to set up, operate, maintain, backup, and scale relational databases for OLTP (Online Transaction Processing)
  + Two storage options:
    - SSD based EBS à Oracle, MS SQL Server, PostgreSQL, MySQL, MariaDB
    - Aurora à Drop-in compatibility with PostgreSQL and MySQL
      * Shared storage, designed for databases, SSD based, built-in cache, sync. replication between three Availability Zones with two copies each (tolerates AZ+1 failures).
      * Aurora instances are independent from the Aurora shared storage



* + Capacity:
    - Aurora up to 128 TB
    - RDS over EBS up to 64TB except 16TB on MS SQL Server
  + Backups
    - You should implement backups, especially when using EBS volumes, to protect the DBs from the loss of the AZ where the volumes are defined, and from eventual data corruption.
    - RDS *automated backups*
      * “Hot” backup all your DB instance data: all the databases and all the tables
      * Daily full snapshot of your data (during your preferred backup window)
      * Copy transaction logs every 5 minutes (as updates are made to your DB instance)
      * *RDS over EBS* retention:
        + Default is 7 days
        + Can be set to up to 35 days.
        + Can be turned off
      * *Aurora* retention
        + 1 day for free
        + Can be set to up to 35 days.
        + Cannot be turned off
      * When you initiate a point-in-time recovery, transaction logs are applied to the most appropriate daily backup in order to restore your DB instance to the second you requested, up to the last five minutes (RPO=5 min).
    - RDS *manual DB snapshots*
      * Enable you to take a snapshot of all your DB instance data:
        + in a known state (“hot” backup)
        + as frequently as you wish and
        + are kept until you explicitly delete them.
    - RDS *manual DB snapshot* and RDS *automated backups* are stored in S3
      * No direct S3 access to backups; managed by AWS
      * They are available to you only for *copying* or for *restore* functionality
  + Multi-AZ deployments
    - RDS over EBS Multi-AZ (optional)
      * Synchronously replicate (RPO = 0) EBS volumes in a different AZ in the *same Region*
      * Active passive (RTO < 2min)
        + Automated instance failover
        + Standby DB instance cannot serve read requests



* + - Aurora Multi-AZ
      * *Built-in* synchronous replication over 3 AZs (RPO = 0)
      * *Optional* DB instance redundancy:
        + Active passive (RTO < 1 min), automated instance failover 🡪 standby DB instance cannot serve read requests



* + - * + 2-node active-active cluster à Aurora multi-master

Only support MySQL

Both instances support read and write requests

RTO=0

* + Vertically scaling *RDS over EBS* performance (up or down)
    - Storage layer
      * EBS storage (IOPS and type) à online (but it may take a while to complete)
    - Compute layer (instance size)
      * Manually choose a new instance size
      * Scale offline
  + Vertically scaling *Aurora* performance (up or down)
    - Storage layer à storage always “no limits” (no need to scale up or down)
    - Compute layer (instance size)
      * Aurora “provisioned”
        + Manually choose a new instance size
        + Scale offline
      * Aurora serverless
        + Automatically connect prewarmed instances, run, and disconnect.
        + Auto-scale instance size automatically
        + Set limits or “no limit”
        + Scale online
  + Horizontally scaling RDS performance
    - Only scale-up
    - Write-heavy workloads à Sharding
    - Read-heavy workloads à Read Replicas
      * Primary DB instance 🡪 Supports read and write operations as usual
      * Replica DB instances
        + Only supports read operations
        + Distribute the read workload
        + Max. Replicas DB instances

15 for Aurora

Support Auto Scaling to dynamically adjusts the number of Aurora Replicas provisioned

5 for RDS over EBS

* + - * Data replication
        + RDS over EBS

Asynchronous (can be across AZs or Regions)

RPO ~ 1 sec

Replication from the Primary DB to all Replicas is handled by the database engine



* + - * + Aurora

Built-in synchronous replication over 3 AZs

RPO = 0

Imagen que contiene Diagrama

Descripción generada automáticamente

* + - * Read Replicas can also be used as a *Disaster Recovery* solution if the primary DB instance fails (at AZ or Region level)
        + Increase availability by locating replicas in separate AZs or Regions
        + You *manually* promote a Replica DB instance as a new Primary 🡪 RTO = minutes
    - Aurora Read Replicas across *Regions*
      * A new Aurora storage and DB instance are created in the target region
      * Asynchronous replication between regions
      * Let you scale read operations into an AWS Region that is closer to your users
      * Can also improve your disaster recovery capabilities
      * Primary 🡪 Read-Write
      * Secondary 🡪 Read-Only
      * Two options:
        + Aurora *Cross Region Read Replica*

Replication from the Primary to Secondary is handled by the database engine (just like for RDS over EBS Read Replicas)

* + - * + Aurora *Global Database*

Replication from the primary DB to Secondary is handled by the Aurora storage layer rather than by the database engine, so lag time for replicating changes is even less than for Aurora Cross Region Read Replica

Some limitations apply

* Redshift à First Data Warehouse/BI (Business Intelligence)/OLAP (On-Line Analytical Processing) built for the cloud
  + Managed service
  + Improves *SQL query* performance significantly for complex analytical queries against massive data sets
  + How Redshift improves query performance?
    - Organizes the data in a *columnar* fashion
      * When you extract data from the source database and load it into Redshift, the data in each column is stored sequentially, rather than writing entire rows one after the other, as OLTP databases do.
      * Columnar storage reduces I/O by:
        + Reading only the columns that you include in your query
        + Enabling more effective compression (the data in a column is of the same type as CHAR, DECIMAL, DATE, etc.)
        + As Redshift holds the minimum and maximum value of each column data block, it avoids scanning blocks which are not relevant for a range query (pruning blocks A.K.A. Zone Maps)
    - Provides the ability to distribute, scan, and *process* *queries in parallel across nodes* in the cluster
      * Evolved from the on-premises massive parallel processing (MPP) ParAccel technology
      * Use PostgreSQL 8.0.2 as the underlying database engine.
* NoSQL databases
  + Amazon DynamoDB à Managed key-value and document database
    - Fast and consistent
      * Scale-out DB servers (nearly unlimited throughput, millions of requests per second)
      * SSD storage (consistent single-digit millisecond response times).
    - Built-in redundancy
      * Data is distributed across an unconstrained number of “storage nodes” across at least 3 AZs
      * SLA of 5 nines availability
      * Can enable *point-in-time recovery* (PITR) to backup your table data automatically with per-second granularity so that you can restore to any given second in the preceding 35 days
    - Consistency options for each read
      * Eventually consistent à default
      * Strongly consistent à Twice read cost
    - Scaling DynamoDB
      * Provisioned
        + Default
        + You can specify how much units are required for reads and writes and DynamoDB allocates the necessary machine resources to meet your throughput needs while ensuring consistent, low-latency performance.
        + Resource allocation based on throughput *units*

A *unit of* *read* capacity represents one strongly consistent read per second (or two eventually consistent reads per second) for items as large as 4 KB.

A *unit of* *write* capacity represents one write per second for items as large as 1 KB.

* + - * + Can use *auto scaling* to adjust your table’s provisioned capacity automatically in response to traffic changes🡪 set minimum and maximum limits (and define how much money you are willing to spend).
      * On-Demand
        + Serverless 🡪 automatically scales
        + “No limits”
        + Pay-per-request
        + Use case: for workloads that are less predictable for which you are unsure that you will have high utilization
    - Redundancyacross *Regions* à DynamoDB *Global Tables*
      * Optional
      * Async. replica between regions (RPO ~ 1 sec)
      * Active/Active 🡪 both regions Read-Write, RTO=0
      * “Last writer wins”
      * Only eventually consistent reads across regions
  + Amazon ElastiCache à Managed *in-memory* database, compatible with *Redis* and *Memcached*
  + Amazon Neptune à Managed *graph* database
  + Amazon DocumentDB à Managed *document* database compatible with *MongoDB*
  + Amazon Quantum Ledger Database (Amazon QLDB) à Managed *ledger* database, like a blockchain, but QLDB has a centralized design.
  + Amazon Keyspaces à Managed *wide column* database compatible with *Apache Cassandra*
  + Amazon Timestream à Managed *time-series* database
* Database caching
  + Aurora already has a built-in cache
  + RDS over EBS caching
    - Configure an in-memory NoSQL database as *side cache*
      * A side cache means you must rewrite the application for cache lookups, population, and invalidation

Diagrama

Descripción generada automáticamente

* + - * Most common design patterns for side-cache (but not the only ones):
        + Write through

Writes 🡪 Application writes to RDS *and* cache.

Reads:

The application will first check to see if that item is in cache. If it is, the cache returns the value with response times in microseconds.

If the item is not in cache, the application fetches the item from RDS.

Data in the cache is always up to date

* + - * + Lazy loading

Writes 🡪 Updates are made to RDS without updating the cache

Reads:

The application will first check to see if that item is in cache.

If it is, the cache returns the value with response times in microseconds.

If the item is not in cache, the application fetches the item from RDS and caches the result for subsequent reads

Add a time to live (TTL) value to each write to the cache (seconds or milliseconds) to decrease stale data

* + - Can use AWS Elasticache as fully managed in-memory NoSQL database for side cache. Options:
      * Elasticache *Memcached*
      * Elasticache *Redis*
  + *Dynamo Accelerator* (DAX) is the *in-line cache* for DynamoDB
    - Improve the read latency of your DynamoDB tables by up to 10 times—taking the time required for reads from milliseconds to microseconds
    - Being an in-line cache, there is no need to rewrite the application 🡪 Just point the existing application at a DAX endpoint, and DAX handles the rest.
    - Fully managed

Diagrama

Descripción generada automáticamente

* + - Combine two cache design patterns:
      * Write through for writes
        + DAX first writes the value to DynamoDB, cache the value in DAX, and then returns success to the application.
      * Lazy loading for reads
        + When an application issues a read to DAX, DAX will first check to see if that item is in cache.
        + If it is, DAX returns the value with response times in microseconds.
        + If the item is not in cache, DAX automatically fetches the item from DynamoDB, caches the result for subsequent reads, and returns the value to the application.
  + More on database caching: <https://aws.amazon.com/fr/blogs/database/amazon-dynamodb-accelerator-dax-a-read-throughwrite-through-cache-for-dynamodb/>
* What services can help me to migrate databases into AWS?
  + AWS *Database Migration Service* (AWS DMS)
    - Replicate databases online
      * Supports homogenous (same DB engine) and heterogeneous (different DB engines) migrations
      * Automatically handles formatting of the source data for consumption by the target database. But it does not perform schema or code conversion.
      * Automatically replicate data changes that occur in the source database during the migration process.
      * One-time migration or on-going replication, as for continuous data replication for DR, Dev/Test, etc.
      * Reliable, secure, simple to use and low cost (even free for six months when migrating databases to Aurora, Redshift, DynamoDB or DocumentDB).
      * Note: either the target or the source database must be located in AWS (cannot migrate between two on-premises databases)
    - To perform schema or code conversion in heterogeneous migrations, the AWS *Schema Conversion Tool* (AWS SCT) integrates tightly with AWS DMS.

Monitoring and Scaling

* How do I know what is happening to be able to react? à Monitoring and log
  + AWS *Billing and Cost Management* à Where you are spending money? (Cost reports and forecasting, include Cost Explorer)
  + *CloudWatch* is a suite of different services.
    - What CloudWatch *“plain”* A.K.A. *“metrics and alarms”* do?
      * On selected AWS resources, and on-premises servers or applications when use the *Unified CloudWatch Agent*:
        + Monitor *operational* *health*, giving you the ability to quickly troubleshoot issues, reducing MTTR.
        + Collect and display *metrics* to monitor unusual activity (prolonged spikes, security violation, DDoS, etc.)
        + Metrics can be captured and available every 5 minutes, 1 minute, or up to 1 second, depending on the metric
        + Acts as a metrics repository for long-term analysis

Metrics are stored *internally,* in each Region

Metrics cannot be deleted, but they automatically expire after 15 months

As time goes by, the resolution decreases

Measures with a period of less than 60 seconds are available for 3 hours

1-minute measures are available for 15 days

5 minutes measures are available for 63 days

1-hour measures are available for 15 months

There is no in-built capability to export CloudWatch metrics data (as can be done with CloudWatch Log to S3)

* + - * You can also set an "*alarm*" to trigger an action (as scale-out or scale-in resources or replace failed instances) when a threshold is sustainedly breached
        + An "alarm" is not necessarily an emergency condition.
        + It is a state that indicates that a threshold was exceeded, so an action can be invoked automatically, as:

Stop, terminate, reboot, or recover EC2 instances

Auto Scaling use CloudWatch alarms to replace failed instances and to trigger scale-out or scale-in resources

Display the “alarm”

Send a notification to an SNS topic

Publishes to an SQS queue

* + - CloudWatch *Logs* allows you to:
      * Collect logs from chosen resources
        + AWS services, customs applications, EC2 instances and on-premises servers (requires the Unified CloudWatch Agent)
        + Monitor collected logs in near real time using *metric filters*, counting occurrences, and sending results to CloudWatch Metrics to graph values or trigger alarms.
      * Store them internally
        + Upload automatically a server’s log files so is not lost when an instance is eventually terminated
        + As long as you want
        + Without worrying about system drives running out of space
        + Logs can be exported to S3, and then reviewed with AWS tools like Athena, or by third party tools like ELK
      * Access them without having to log in each individual server
        + CloudWatch Logs console à Basic way to view and filter log data
        + CloudWatch Logs Insight à Advanced tool to search, filter, query, extract, and view log data (available November 2018).
    - CloudWatch *Events* responds to those business challenges:
      * How to trigger a process on a schedule?
      * How to detect AWS resource operational change state and, if necessary, automatically respond to it?
      * CloudWatch Events is now part of *EventBridge*, the preferred way to manage your events, with more features, since it not only accepts events from AWS services, but also from your own and third-party SaaS apps
  + *CloudTrail* is an audit service to log activity in an AWS account for *later* inspection
    - Responds to these questions:
      * *Who* executed *what* action?
      * *When* and from *where*?
    - Two tiers:
      * CloudTrail “Event history”
        + Always log the management (“control plane”) of operations: resources creation, modification, or deletion (API calls) and sign-in attempts (non-API events)
        + Do not log operations performed on or within S3 (“data plane”).
        + Events are recorded with a delay of up to 15 minutes since the API call.
        + Those records are internally available for 90 days.
        + Free of charge, cannot be disabled nor deleted
      * You can setup CloudTrail “trails”
        + To deliver copies of CloudTrail events to S3
        + By default, trail log files are stored indefinitely. You can use S3 object lifecycle management rules to define your own retention policy. For example, you may want to delete old log files or archive them to Amazon Glacier.
        + Also can log data events. Can select logging read-only, write-only, or don’t log management events. By default, trails log all management events.
        + “Trails” are mandatory to:

Log data events performed on or within S3 (often high-volume activities)

Centralize CloudTrail logs from multiple regions (best practice)

Interact with other AWS services, as Athena

* + - * + All log files delivered to S3 are compressed (.gz) JSON files, encrypted by default using S3 SSE, and optionally with AWS KMS.
        + Are best practices:

Enable log file integrity validation using SHA-256 for hashing and SHA-256 with RSA for digital signing.

Enable MFA delete protection for CloudTrail buckets.

* + - * What are the differences between CloudWatch and CloudTrail?
        + CloudWatch Logs

Is “selective”: collects and store logs from *chosen* resources.

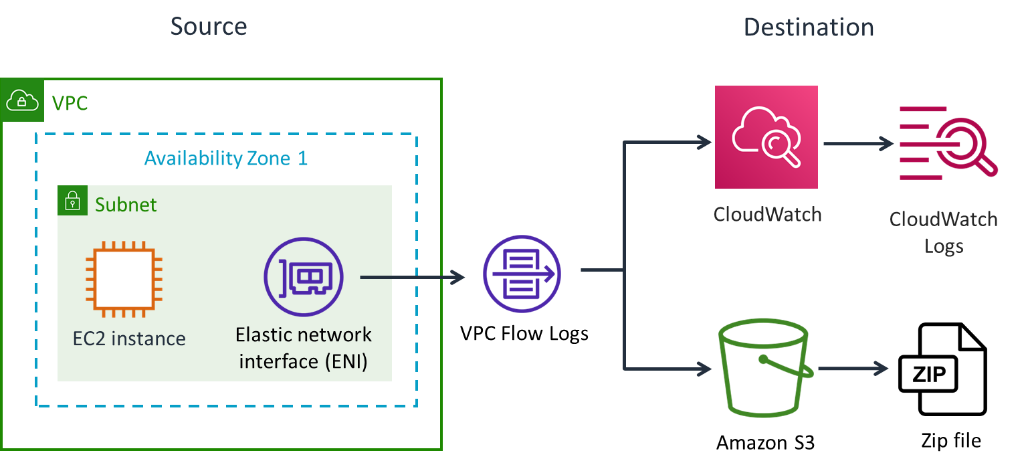
Allows monitoring logs in *near real time* using metric filters, sending values to CloudWatch “plain” to graph values or trigger alarms.

* + - * + CloudTrail

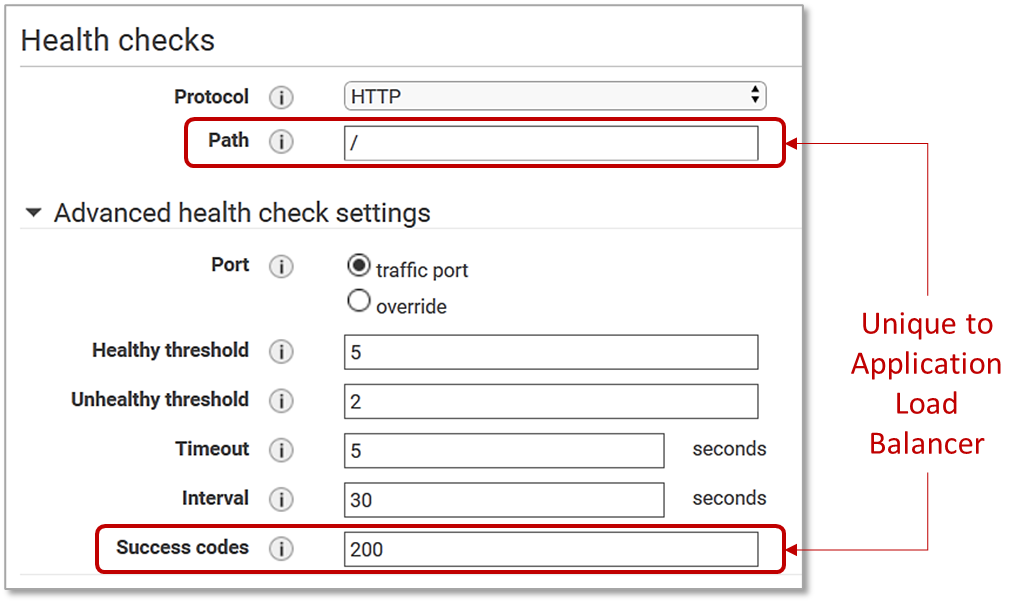
Is “inclusive”: provides a centralized history of *all* the management activity

Focuses on record activity for *later* inspection (up to 15 minutes to record an event)

* + - * + Services as CloudTrail Insights, AWS Config, AWS Config Rules or GuardDuty are usually easier to use than CloudTrail and can also automate record analysis and remediation, but are “selective”
  + *Flow Logs* is an easy way to capture *information* about the network IP traffic going in and out of network interfaces in your VPC (do not capture IP traffic)
    - Agentless
    - Can be enabled on a *VPC*, a *subnet*, or a *network interface*.
      * If you create a flow log for a subnet or VPC, each network interface in that subnet or VPC is monitored
    - Flow Log data is published to *CloudWatch Logs* or *S3*
      * Do not capture real-time log streams. Publishing takes several minutes
    - Flow Logs will not include any of the following traffic:
      * Traffic outside VPC, as API Gateway, Lambda and S3.
      * Traffic to Amazon DNS servers, including queries for private hosted zones.
      * Windows license activation traffic for licenses provided by Amazon.
      * Requests for instance metadata.
      * DHCP requests or responses.
    - Use case:
      * Diagnose overly restrictive security group rules
      * Monitor the traffic that is reaching your instance
      * Determine the direction of the traffic to and from the network interfaces



* + - Available June 2019, *VPC Traffic Mirroring* provides deeper insight into network traffic than VPC Flow Logs by allowing you to analyze actual traffic *content*, including payload.
      * Network traffic is *copied* from the Elastic Network Interface to a target for analysis.
* Fault tolerance and high availability
  + *Fault tolerance* and *high availability* are often used interchangeably but are not the same thing.
  + For AWS:
    - *Fault-tolerance* is a system’s ability to remain in operation even if some of the resources used to build the systems fails, through the use of redundant components.
    - *High availability* is not about preventing system failure, but the ability of the system to recover quickly from it.
  + Many AWS services are inherently reliable, fault tolerant, and highly available, such as Route 53 or S3
  + As an AWS solutions architect, you should configure other services, such as EC2 Instances, to remain in operation (or minimize system downtime) if they fail (or are reclaimed as can occur with Spot instances), while maintaining an agreed response time to users’ requests.
* ELB service
  + How do you create a high available environment to handle *EC2* *instances* and *AZ* failures? à Use ELB (Elastic Load Balancing)
    - Serves as a *single point of contact* for clients for multiple instances in multiple AZs. When integrated with EC2 Auto Scaling, it allows dynamically changing the number of instances
    - *Health checks* feature periodically monitor targets (such as EC2 instances), and when detects failures, stop sending traffic to them.



* + - *Load balancing* feature spread the incoming traffic across the operational targets.
    - Other ELB functionalities
      * Security
        + Allow put web-tier instances in a private subnet, behind the ELB
        + TLS termination
        + SSL decryption offload
      * Sessions
        + Connection draining
        + Sticky sessions
      * Etc.
  + 4 types of load balancers: *Application Load Balancer* (ALB), *Network Load Balancer* (NLB), *Gateway Load Balancer* (GWLB), *Classic Load Balancer* (CLB).
    - ALB (Application Load Balancer) à Functions at the application level (OSI layer 7)
      * Use case à When you need to route requests to different *applications* based on the *HTTP/HTTPS content* (e.g., URL paths)

Diagrama, Escala de tiempo

Descripción generada automáticamente

* + - * Load balancing à *If multiple targets* within a target group, a target is select based on those algorithms:
        + Round Robin (default) à Simple, but targets and request should be similar
        + Least Outstanding Requests à Useful for dissimilar targets or long-standing requests
        + Weighted Target Groups à Helpful for blue/green and hybrid deployments
      * Supports Lambda in addition to EC2 instances, containers and IP addresses used by servers on an extended on-premises network.
    - NLB (Network Load Balancer) à Functions at the connection level (OSI layer 4)
      * Use case à When you need to distribute TCP, UDP or TLS long-lived connections across targets for extreme performance
      * Load balancing à Select a target using a flow-hash algorithm based on connection data
        + Each individual client connection is bound to a single target for the life of the connection.
      * No header modification and preserve client IP to targets
      * Do not support Lambda as target

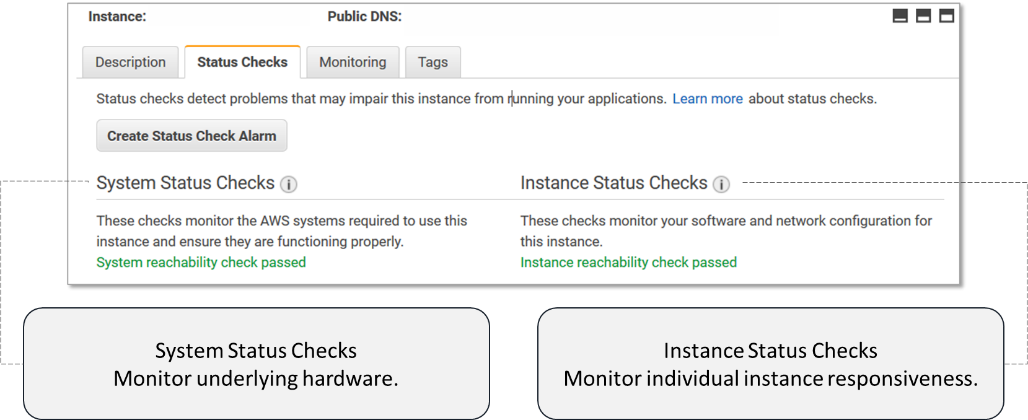


* + - GWLB (Gateway Load Balancer) à A new type of load balancer that includes Layer 3 Gateway and Layer 4 Load Balancer capabilities.
      * Provides a single point of contact and distributes traffic to third-party virtual appliances located in a provider VPC.

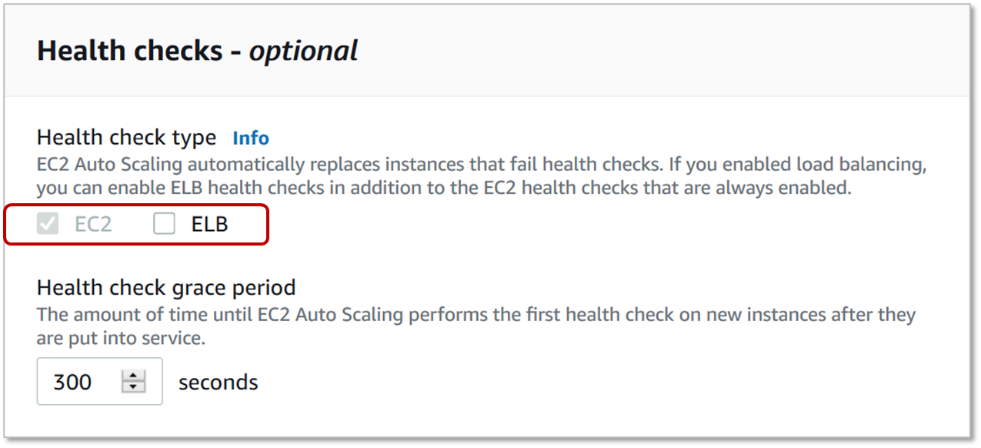
Diagrama

Descripción generada automáticamente

* + - CLB (Classic Load Balancers) à Is the previous generation.
      * Use only with a legacy EC2-Classic network, which became unavailable as of October 30, 2021
    - Which load balancer should I pick?
      * For legacy EC2-Classic network, use Classic Load Balancer
      * For third-party virtual appliances, use Gateway Load Balancer (GWLB)
      * For TCP, UDP or TLS, use Network Load Balancer (NLB)
      * For all other cases, use Application Load Balancer (ALB)
* EC2 Auto Scaling
  + Elasticity is the ability to follow the demand curve, increasing or decreasing the compute capacity of your application
  + What service allows me to react to significant changes in performance needs, giving me elasticity? à Auto Scaling
  + What EC2 Auto Scaling can do?
    - Provide availability
      * Automatically detect an unhealthy instance, terminate it, and launch an instance to replace it



* + - * Can be integrated with *ELB health checks* (for example, to ensure the responsiveness of an application)



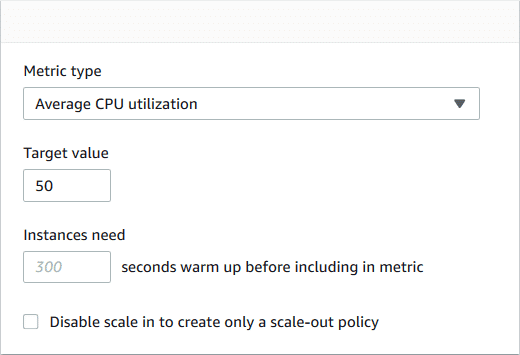
* + - * Can also configure Auto Scaling to automatically re-balance instances across AZs.
    - Providing elasticity by scaling instances *horizontally*
      * Scale-out (launch Instances) to meet demand (also serves to avoid timeouts)
      * Scale-in (terminate instances) to reduce costs
      * Note: EC2 Auto Scaling *cannot* scale instances *vertically* (scale-up and scale-down)
  + What are the steps to configure EC2 Auto Scaling?
    - Create a template that specifies *what* instance configuration to use at launch 🡪 *Launch Template*
    - Choose *how many* instances (the group’s minimum and maximum size), and *where* (subnet, etc.) do you need instances 🡪 Auto Scaling Group
    - Configure *when* scale-out and scale-in 🡪 Auto Scaling Policy
  + Launch Template
    - A configuration template that records the parameters necessary to launch a new instance.
    - Included are the instance type, EBS volume, ID of the AMI, key pair, security groups, and the other parameters that Auto Scaling use to launch EC2 instances.
    - *Launch Template* is preferred over *Launch Configuration* (legacy) as it provide versioning and more advanced Amazon EC2 configuration options, and can also be used to manually provision EC2 instances in a consistent basis.
  + How to choose the group’s minimum and maximum size? 🡪 Auto Scaling Group
    - Minimum size 🡪 Auto Scaling always keep this minimum number of instances running
      * If you are processing *batches* that run periodically, you might want to set the minimum to zero 🡪 You do not pay for instances while the batch does not run.
      * For *interactive applications*, set the minimum to 1 🡪 The application must remain available even if no one uses it
      * For *High Availability*, the minimum capacity size should be set to 2 🡪 The application must tolerate the unavailability of one Availability Zone (or just an instance) with no downtime
    - Maximum size 🡪 Auto Scaling never start more than maximum number of instances
      * Selecting a reasonable maximum capacity size depends on your application, your EC2 quota, and probably a budget.
      * You pay for what you use; therefore, your organization may limit you from running more than a certain number of instances
    - Auto Scaling Group also include the “where”: VPC, subnets and ELB.
  + When scale-out or scale-in? 🡪 Auto Scaling Policy options
    - Time-based scaling à Proactive
      * Predictive scaling à Simplest, AWS *Machine Learning automatically* schedule scaling looking at historic traffic patterns
      * Scheduled scaling à *You* manually schedule scaling
    - Dynamic scaling à Reactive (based on load)
      * Target Tracking
        + Simplest
        + You choose a metric value

ASGAverageCPUUtilization 🡪 Average CPU utilization of the Auto Scaling group.

ASGAverageNetworkIn 🡪 Average number of bytes received on all network interfaces by the Auto Scaling group.

ASGAverageNetworkOut 🡪 Average number of bytes sent out on all network interfaces by the Auto Scaling group.

ALBRequestCountPerTarget 🡪 Number of requests completed per target in an Application Load Balancer target group.



* + - * + Instances are launched or terminated by Target Tracking to maintain the metric value 🡪 Acts like a home thermostat
      * Step Scaling
        + You manually configure CloudWatch alarms that trigger the scaling process, and define a set of *step adjustments*, which vary based on the size of the alarm breach
        + Support multiple CloudWatch metrics simultaneously.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

* + - * + In response to CloudWatch alarm breaches, Step Scaling increase or decrease the number of instances by the amount indicated in step adjustments
      * Simple Scaling à Legacy
    - Multiple scaling options can be used simultaneously (e.g., Predictive Scaling in conjunction with Target Tracking)
  + Purchasing options best practices for EC2 Auto Scaling
    - For *batch* process:
      * If the batch process has *flexible* *start and end times* 🡪 can use *Spot Instances*
      * If the batch process *requires* running on a *recurring start time and duration* on a daily, weekly, or monthly basis 🡪 can use *Scheduled Reserved Instances* in a specified AZ (option recently discontinued).
      * If the batch process run in *a defined amount of time* (1 to 6 hours), *should not be interrupted,* and has *flexible* *start and end times* 🡪 can use *Spot Blocks*
    - For *interactive* applications:
      * Use *Reserved Instances* or *Savings Plans* for *minimum* capacity
      * Use *Spot Instances* for *scale-out* capacity
      * Use *On-Demand Instances* if Spot Instances are interrupted or not available (**or when you foresee high demand and the likelihood that AWS will reclaim your Spot Instances!**)
      * You can automate the buying process using the *Spot Fleet* service

Use automation to improve our infrastructure

* Why should you automate?
  + Manual à Error prone, costly ($, people, effort), long, no control version, lack of audit trails, inconsistent data management, does not scale, etc.
  + Solution? 🡪 Automate
* Choosing the right solution to automate
  + Do not want to worry about the infrastructure that runs your web application? à Elastic BeansTalk (Platform-as-a-Service)
    - Automatically deploy and scale web applications
    - Accepts Java, .NET, PHP, Node.js, Python, Ruby, Go, or Docker code.
    - Deploys on Apache, Nginx, Passenger, and IIS web servers.
    - Can provision S3, RDS and DynamoDB automatically.
    - Can use Amazon Linux AMI or Windows Server 2019.
    - You decide whether you want to manage some (or all) of your Elastic Beanstalk infrastructure elements or enable *Managed Platform Updates* to let AWS automatically manage your platform updates.
  + Focus on build infrastructure? à CloudFormation (Infrastructure-as-Code)
    - Use JSON/YAML template to design, document, build and update stacks (infrastructure deployed by CloudFormation based on template)
    - Did someone make a manual change?
      * Drift detection
      * Roll back the service to the last good state
    - Need prebuild reference architectures built by Solutions Architects?
      * AWS Solutions Implementations
      * AWS Quick Starts
  + Focus on build infrastructure using familiar programming languages? (Embracing a DevOps culture) 🡪 AWS CDK (AWS Cloud Development Kit)
    - Open-source software development framework for infrastructure-as-code (IaC)
    - Uses CloudFormation as a foundation
    - Has all the advantages of CloudFormation while enables you to model application infrastructure using TypeScript, Python, Java, .NET, and Go.
    - Can use pre-configurated resources with proven defaults
  + Focus on maintain your fleet after stack creation? 🡪 Systems Manager (Operations-as-Code)
    - Systems Manager is a toolbox, with multiple tools and services to gain operational insights and automate action on AWS and on-premises resources
    - Some of these require an agent, which can also be installed on on-premises computers.
  + Focus on deploy the application layer? à OpsWorks (Configuration-as-Code)
    - Do you want a fully managed Chef Automate server? à OpsWorks for Chef Automate
    - Do you want a fully managed Puppet Master server? à OpsWorks for Puppet Enterprise
    - Do you want to deploy Chef recipes for free? à OpsWorks Stacks
  + Can combine OpsWorks Stacks with CloudFormation

Running microservices using Containers

* What are microservices? à A design approach to build a decoupled application as a set of *small stateless functions that communicate over well-defined APIs*.

Imagen que contiene Patrón de fondo

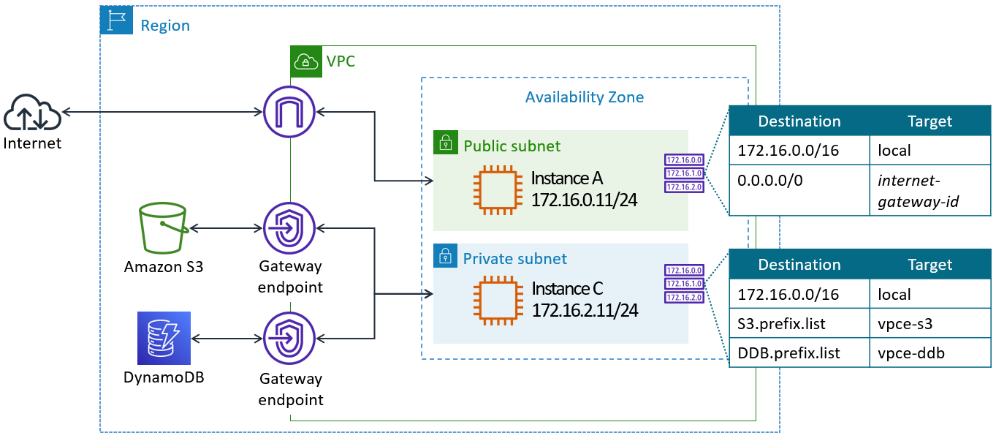
Descripción generada automáticamente

|  |  |  |  |
| --- | --- | --- | --- |
| **Architecture pattern** | **Monolitic** | **Service-oriented (SOA)** | **Microservices** |
| **Program size** | Single large application | Interdependent business task | Stateless small function, API |
| **Persistence** | Shared database | Shared database | Different data stores |
| **Communication** | Shared data store | Enterprise service bus (ESB) | Queues and topics, API |
| **Execution time** | Years | From minutes | From 1/1000 second |
| **Execution platform** | Mainframes, VMs | VMs, Containers | Containers, Lambda |

* Where to run microservices?
  + Containers
  + AWS Lambda (run stateless code serverless)
* How to analyze, trace, and debug distributed applications built using microservices and serverless architectures? 🡪 X-Ray
  + Provides a complete view of requests as they travel through your distributed application
  + Benefits
    - Analyze and debug performance
    - View latency and pinpoint bottlenecks
    - Identify specific user impact
    - Works with AWS and non-AWS services
* Containers and Docker
  + What is a container? à A method for *operating system* virtualization (unlike hypervisors as ESXi, HyperV, Xen, or KVM that virtualize the *physical server*)
    - Lightweight (RAM).
    - Small (storage).
    - No noticeable performance overhead (CPU).
    - Boot in just a couple of seconds.
  + What is Docker? à Popular virtualization engine for running Windows and Linux containers
  + How to deploy, schedule, scale, and manage a container *fleet*? à Orchestration tool
    - ECS (Elastic Container Service) à Free, fully managed AWS container orchestration service
    - EKS (Elastic Container Service for Kubernetes) à Fully managed Kubernetes (A.K.A. k8s or "kube"), a popular open-source container orchestrator
  + How to centrally store container images? à Amazon ECR (Elastic Container Registry)
    - Free, fully managed, Amazon *container image registry*
    - Makes it easy to deploy, version, manage and store Docker images in S3, with integration with Amazon ECS (Elastic Container Service) and EKS (Elastic Container Service for Kubernetes).
  + Where to run containers?
    - EC2 instances
      * Choose instance type, AMI and pricing model
      * Provision and manage EC2 instances (or let the orchestrator do it)
      * Pay per second when use Amazon Linux AMI or since June 2021, Windows Server. For others, pay per hour.
    - Fargate à A serverless service for running containers
      * Run containers directly, without deploying EC2 instances
      * You choose the amount of vCPU (up to 4) and RAM (up to 30GB), and pay for them per second (minimum 60 seconds)
      * Support ECS (Elastic Container Service) or EKS (Elastic Container Service for Kubernetes) orchestrators.
      * Since October 2021, it supports *ECS Windows containers* in addition to *EKS* and *ECS Linux containers*

Networking-2: How to connect resources external to your VPC without going through the public internet?

* + You can connect services external to your VPC (as S3, Dynamo, AWS Key Management Service, etc.) without going through the public internet (using an Internet Gateway).
    - Most AWS services do not actually sit within a VPC
    - Communication between resources based in a VPC and resources outside that VPC traverses the public internet by default
    - Instances in your VPC do not require public IP addresses to communicate with the service.
    - Traffic between your VPC and the service does not leave the AWS network.
* VPC Endpoints
  + - Permits private connections between your VPC and supported Partners and AWS services without requiring an Internet Gateway, NAT, VPN, or Direct Connect connection.
    - Currently do not support cross-Region requests. You cannot create an endpoint between a VPC and a service in a different Region.
    - Are fully managed, horizontally scaled, redundant, and highly available. Include:
    - Gateway endpoints
    - Interface endpoints
    - Gateway Load Balancer endpoints
    - Gateway endpoints
      * Only to connect *S3* and *DynamoDB* using a *URL*
    - Add to the (private) route table a route to direct the bucket or DB traffic to the S3 or DynamoDB endpoint
      * + The destination in your route is the *URL* of the bucket or DB
        + Gateway endpoint connections are only available for resources within the VPC. Resources on the other side of a VPC peering connection can’t use the gateway endpoint to access the associated bucket or DB.
        + There is no charge for using gateway endpoints



* + Interface endpoints
    - Include:
      * *AWS services* as CloudWatch, SNS, etc. (more than 100 supported services, including S3, but not yet DynamoDB, which currently only supports Gateway Endpoint)
      * *PrivateLink* as services hosted by other AWS customers or AWS Partners in their own VPCs, or AWS Marketplace services.
    - Automatically assign to the Interface Endpoint an *Elastic Network Interface* (ENI) with a private IP address from the CIDR range of your subnet, so you do not need to update route tables.
    - This ENI IP address serves as an entry point for traffic destined to the service.
    - Interface Endpoints can be accessed through VCP peering connections, and by on-premises resources through secure connections like AWS Direct Connect or AWS VPN.
    - Pricing per interface endpoint per AZ: $7.2 @ month + $10 @ TB traffic

Escala de tiempo

Descripción generada automáticamente

* + - Gateway Load Balancer endpoints
    - To connect to a *Gateway Load Balancer* (GWLB) in a 3rd part provider VPC
    - Send traffic to the *Gateway endpoint* (and from there to the GWLB) by updating your VPC route tables
    - Pricing per GWLB endpoint per AZ: $7.20 @ month + $3.50 @ TB traffic

Diagrama, Escala de tiempo

Descripción generada automáticamente con confianza media

* + - More on Gateway Load Balancers: <https://aws.amazon.com/blogs/networking-and-content-delivery/introducing-aws-gateway-load-balancer-supported-architecture-patterns/>
  + VPC Peering - Isolating some of your workload is generally a good practice
    - For most use cases, a single VPC is not enough
    - Many organizations use multiple accounts to create administrative or billing isolation, and to limit the impact of errors.
    - Multiple VPC à Multi-VPC Pattern or Multi-Account Pattern?
    - But you might need resources in one VPC to communicate with resources in another VPC.
    - How to interconnect two VPCs without going through the internet? à VPC peering
      * CIDR blocks cannot overlap
      * Use private IP addresses 🡪 To permit the flow of traffic between the peer VPCs using private IP addresses, add a route to one or more of your VPC’s route tables that points to the IP address range of the peer VPC (or the private or restricted subnet in the peer VPC)
      * Is a one-to-one relationship between two VPCs 🡪 Transitive peering relationships are not allowed
      * Intra and inter-Region support
      * Always stay on the global AWS backbone.
        + Use highly available connections—no single point of failure.
        + Avoid bandwidth bottlenecks.
        + Data transferred across peering connections is charged at the standard data transfer rates.
        + Peering relationship is free
      * Bypass the internet gateway or virtual gateway.
        + Instances can communicate across a peering connection as if they were in the same network.
      * Cross-account support
        + The owner of the requester VPC (or local VPC) sends a request to the owner of the peer VPC
        + The owner of the peer VPC must accept the VPC peering connection request to activate the VPC peering connection.
      * You might also need to update the security group or NACL rules to ensure that traffic to and from the peer VPC is not restricted
      * Limits
        + Default 50 active peering connections per VPC, maximum 125.
        + Maximum 100 static routes per route table
        + A full mesh network design using VPC peering to connect each VPC to every other VPC in the organization quickly reach the limit (15 VPC = 105 full mesh peering) 🡪 In this case, you can use other topologies, such as partial mesh, star, etc.

Diagrama

Descripción generada automáticamente

* + How to manage traffic flow and security across hundreds of VPCs? à Transit Gateway
    - Network transit hub to interconnect
      * VPCs
      * AWS Site-to-Site VPN
      * Direct Connect
      * Other Transit Gateways for inter-Region peering
    - Are fully managed, horizontally scaled, redundant, and highly available
    - Up to 5000 50 Gbps “attachments”
    - Each attachment: $36 @ month
    - Traffic: $14 @ TB
* Hybrid networking 🡪 You can *securely* interact with AWS resources within a VPC as if they were within your on-premises network, and vice versa.
  + Resources within the VPC does not use Internet Gateway nor NAT
  + How to securely interconnect on-premises datacenters or branch office site to a VPC? Two stages:
    - Configure a gateway and attach it to the VPC to provide edge routing and extend the AWS VPC to the on-premises network. Two gateway options:
      * Virtual Private Gateway (VGW)
        + Is a logical, fully redundant distributed edge routing function that sits at the edge of your VPC.
      * Transit Gateway
    - Secure the connection. Two options:
      * Using internet à AWS *Site-to-Site VPN*
        + An IPsec VPN
        + On the “customer side” of the VPN connection configure a *customer gateway*:

Hardware IPsec VPN (Cisco ASA, etc.)

Software IPsec VPN (pfSense, etc.)

Connect on the AWS side to a *Virtual Private Gateway* (VGW) or a *Transit Gateway*

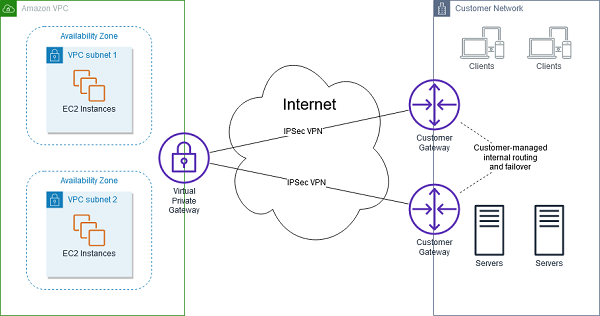
* + - * + On the AWS side of the VPN connection configure an *AWS Site-To-Site VPN*

Fully managed by AWS

Provides two 1.25 Gbps VPN endpoints (tunnels) between a *VGW* or a *Transit Gateway*, which you can simultaneously use for high availability

Connect to the public internet and the customer gateway via a VGW or a Transit Gateway

$36@Month + $90@TB OUT



* + - * + Optionally, on the AWS side of the VPN connection you can use another VPN (pfSense, etc.) if you do not want to use the AWS Site-To-Site VPN

Run on EC2 instance

Connect to the public internet via the Internet Gateway

* + - * Using *private wired* connection à AWS *Direct Connect* (DX)
        + Provided by:

AWS in some AWS Edge Locations

AWS Direct Connect Partners

* + - * + One end of the fiber is connected to your router, the other to a Direct Connect router
        + The Direct Connect router connect to the VPC via a *Virtual Private Gateway* (VGW) or a *Transit Gateway* over private lines.
        + Dedicated Connection

Dedicated wire

1Gbps, 10Gbps or 100Gbps

* + - * + Hosted Connection

Shared wire

Starting with 50Mbps

* + - * + Costs (All DX except Japan):

Dedicated Connections

1Gbps: $216 @ month

10Gbps: $1620 @ month

100Gbps: $16200 @ month

Hosted Connections

50Mbps: $21 @ Month

Traffic

IN: free

OUT: from $20 @ TB USA to $150 @ TB Brazil

* + - * + Use cases:

Advantages

Network performance predictability

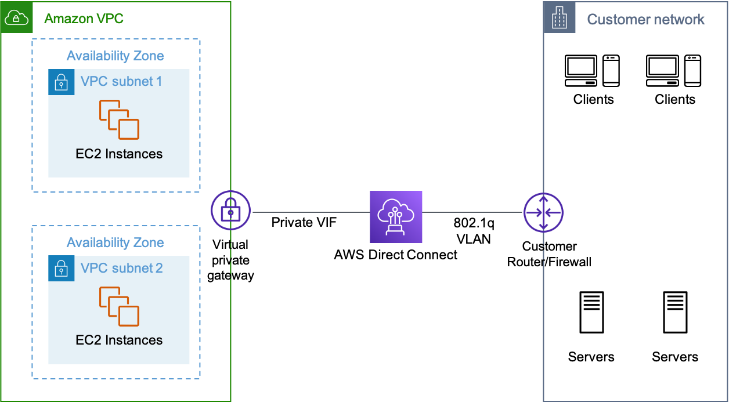
Security and compliance

Increase throughput

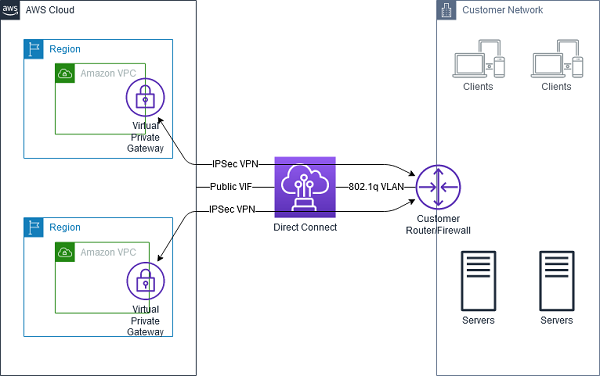
Reduce transfer costs

Data Migrations (pay only for the hours required for data transfer)

Hybrid cloud architectures (permanent)



* + AWS Direct Connect + Site-to-Site VPN
    - Can configure Site-to-Site VPN as a Direct Connect *backup link* to the same VPC, but only for failover. The Direct Connect path will always be preferred.
      * <https://aws.amazon.com/directconnect/resiliency-recommendation/>
    - Can configure Site-to-Site VPN to *encrypt Direct Connect* 🡪 Combines the benefits of the end-to-end secure IPSec connection with better performance of the AWS Direct Connect to provide a more consistent network experience than internet-based VPN connections.



* + How to provide high availability at the region level? 🡪 Use Route53 (AWS DNS service)
    1. Replicate data, and duplicate network and compute in another region
    2. Route53 *health checks* detect region failure



* + 1. Route53 *failover* route traffic to a healthy region

Decoupling your infrastructure asynchronously

* How to call Lambda functions from applications running outside the VPC? 🡪 API Gateway
  + Serves as a “front door” for your API based applications
  + Can be internet facing or internal only
  + It also acts as an additional layer of security to our applications because it can integrate with security services like WAF
  + Provides you the ability to authenticate and authorize all your requests to a backend.
  + Integrates with Amazon CloudWatch by sending log messages and detailed metrics to it.
* Decoupling your infrastructure asynchronously 🡪 Problem: traditional monolithic architectures (Client-Server) are “Tightly Coupled”
  + Chains of tightly integrated servers, each with a specific purpose
  + When one of those components goes down, however, the disruption to the system can ultimately be fatal
  + If you scale one layer, every server on every connecting layer must be connected appropriately also.
* Best Practice: design architectures with components working as independently as possible (decoupled or “Loosely Coupled”), so failures and scaling does not affect other components

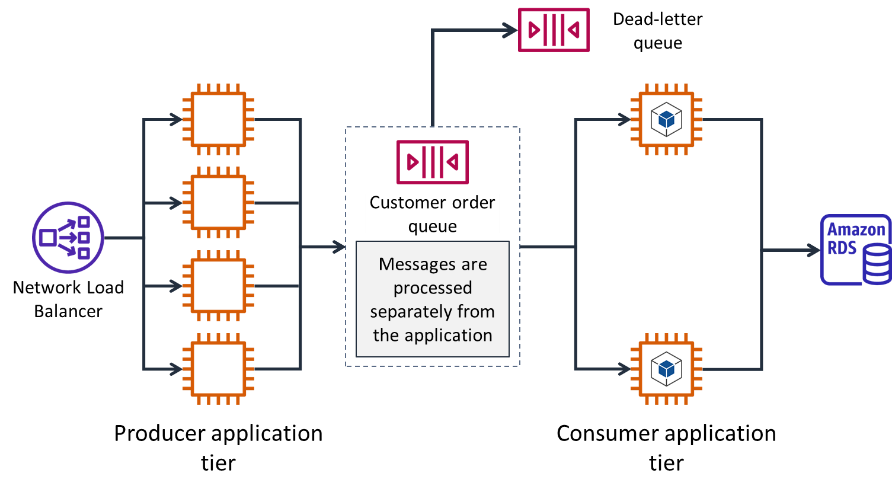
Interfaz de usuario gráfica, Diagrama, Aplicación

Descripción generada automáticamente

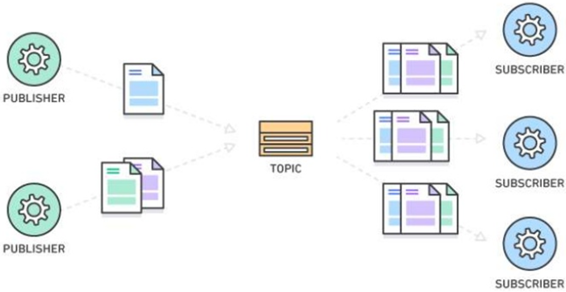
* Solution: use intermediaries between layers of your system so systems can scale and evolve independently and flexibly.
  + Synchronous decoupling à Internal facing load balancer may be a solution, but vulnerabilities may arise:
    - How about failed steps? à Logs, but complex …
    - How about handling sudden traffic spikes? à Over-provisioning compute, but costly …
    - How about out-of-order delivery? à Timestamp, but complicated…
  + Asynchronous decoupling à *Messaging* can be used to easily and reliably communicate between components. Options:
    - Message Queues
    - Message Topics
    - Amazon MQ
* What is a message queue service? à A service like a *buffer* between processing layers.
  + Amazon SQS (Simple Queue Service) à Simple, serverless *message queue* service.
    - Store messages (data) in an orderly manner, like a pipeline.
    - A component called a *producer* put a message into the queue.
    - The message is stored on the queue until another component called a *consumer,* pull (retrieve) the message and process it.
    - To prevent another consumer get the same message, the *visibility timeout* hides the message 🡪 This behavior allows that only one consumer processes any given message (1:1).
      * The default value is 30 seconds.
      * Must be larger than the maximum time it takes to process and delete the message
      * Can be set between 0 seconds to 12 hours.



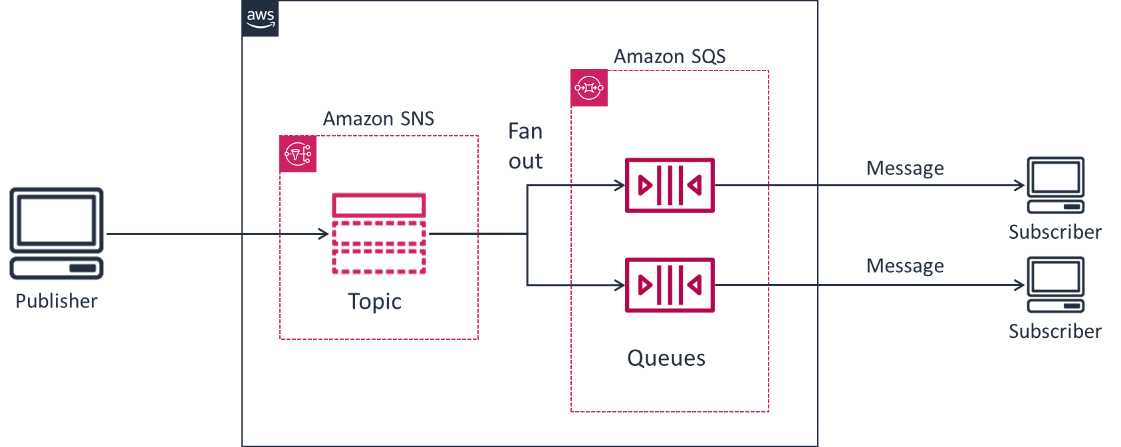
* + How about failed steps?
    - The messages are not automatically deleted. Therefore, the consumer must explicitly delete the message from the queue after processing it 🡪 This behavior allows easy recovery from failed steps.
    - What will happen when the consumer that pooled the message died or the processing fails? à After a while, the visibility timeout will expire, and the message will be available in the queue to be processed again.
    - What to do with the message if subsequent retries (set between 1 and 1000) fails? à The message is redirected to a "dead-letter queue" to analyze why it is stuck, or to process it in another way.



* + How about handling sudden traffic spikes? à Buffering prevents consumers from being flooded.
  + How about order processing? (Queue types)
    - FIFO queues.
      * Strict ordering.
      * Exactly once processing (no-duplicates guarantee).
      * Limited to 300 messages per second.
      * To guarantee order delivering, FIFO queues don't serve messages to more than one consumer at a time.
    - Standard queues (default)
      * Nearly unlimited scaling.
      * Best-effort ordering (messages might be delivered in an order different from which they were sent).
      * At-least-once delivery (duplicates are possible).
      * Supports multiple consumers: meanwhile a consumer processes a message, another consumer can pull and process the next message in the queue à This behavior allows scaling and load balancing.
  + How to do to not receive an empty response for which we will also have to pay? à Set long polling
    - When the queue is empty, SQS wait a time for a message to become available in the queue before sending a response.
    - Polling wait time can be set between 1 and 20 seconds.
    - SQS sends an empty response only if the polling wait time expires.
    - Recommended practice when there is a possibility that the queue is emptied, but not by default
  + Message size: payloads can contain up to 256KB of text. To send messages as large as 2 GB, you can use the *Amazon SQS Extended Client Library for Java* to store messages in S3 or DynamoDB and send references to them.
  + Message queues use cases:
    - Asynchronous and reliable communication between processing layers
      * Producers and consumers not available at the same time (network failures, batch process, backlogs) or may not all process the same amount of work simultaneously (traffic bursts).
        + By default, a message is stored in the queue up to 4 days, and this value can be extended to 14 days.
        + If after that period the message was not processed, it is automatically deleted.
    - Load balancing à Use multiple consumers running concurrently (only standard queues)
    - Scaling trigger  Use queue length as criteria for scaling
* Amazon SNS (Simple Notification Service) à Simple, serverless *publish-subscribe message* service
  + What is a *Pub-Sub message* service? (AKA *message topic* service) à A service like a broadcasting station, where different radio stations play different types of music and listeners tune in to the stations that play the music they like
    - A *publisher* pushes a message to the message topic service
    - The message topic service fan-out immediately the message to anyone interested in these messages (the topic *subscribers*).
    - Once successfully delivered, there is no way to recall it. If no consumers available, then the message is lost.
  + This behavior allows:
    - Subscribers receive all messages published to the topics to which they subscribe, and all subscribers to a topic receive the same messages.
    - Each subscriber is only allowed to consume a message once (consumed copies disappear).
    - Each subscriber can do something different with the message in parallel.



* + Characteristics of Amazon SNS
    - SNS support different formats/transports to deliver notifications
      * HTTP, HTTPS
      * Email, Email-JSON
      * Standard SQS queues (not FIFO SQS queues)
      * SMS
      * Kinesis Data Firehose
      * Lambda functions
      * Platform application endpoint
    - Highly scalable à Soft limited to 10 million subscriptions per topic, and 100,000 topics per account.
    - Message size: SNS messages can contain up to 256 KB of text data, except for SMS messages limited to 1600 Bytes.
    - Message filtering empowers the subscriber to create a filter policy, so that it only gets the notifications it is interested in, as opposed to receiving every single message posted to the topic.
    - Like SQS, beginning Oct. 2020, SNS support:
      * *Standard* Topics
        + Best-effort ordering and at-least-once delivery
      * *FIFO* Topics
        + Strictly preserved message ordering
        + Exactly once message delivery (can also enable a 5-minute content-based deduplication window)
        + Up to 300 publishes/second
        + Only support FIFO SQS as subscription protocol
  + SNS use cases:
    - Parallelly perform different operations on the same data à e.g., deliver the same data to production environment and audit
    - Push messages in different formats à e.g., mail and SMS
    - Time sensitive messages à e.g., notify an event instantly
* SNS and SQS are often used together



* Do you need to migrate applications to AWS or integrate hybrid environments that use *Apache MQ*? à *Amazon MQ* is a managed message broker service for Apache *ActiveMQ* and *RabbitMQ*, combining queues and topics.
* Kinesis 🡪 Is a suite of different services to collect, process, and analyze data *streams*.
  + Kinesis Data *Streams* (A.K.A. “Streams”)
    - Ingest streaming data and send it to applications running on EC2 Instances or Lambda, that will handle the transformation and processing of data in *real-time*.
    - Must code the processing application running on “consumers”
    - Must provision “shards” to read streaming data and write records to be processed by consumers.
      * Each shard can read data up to 1 MB per second and write at a rate of 2 MB per second.
      * Each consumer reads from a particular shard.
      * Must define the quantity of ‘shards’ required by the stream throughput
        + Can manually add or remove shards
        + Can set up auto-scaling (for shards and for EC2 Instances if you use them)
    - Use case: When you need to perform *real-time processing*.

Interfaz de usuario gráfica, Aplicación

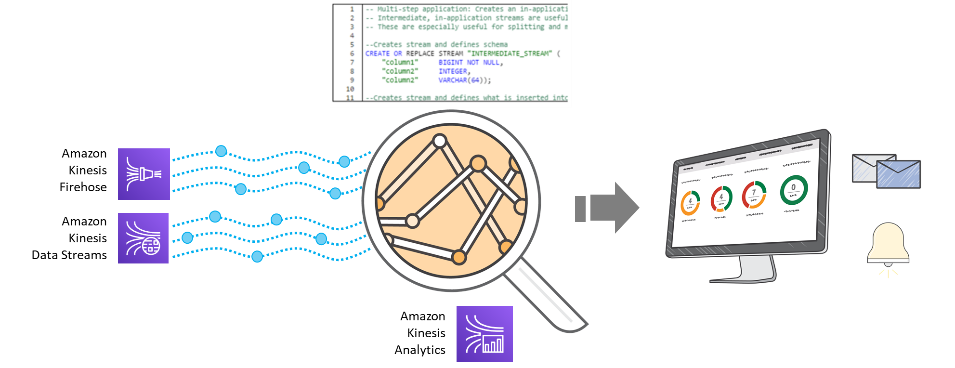
Descripción generada automáticamente

* + Kinesis Data *Firehose* (A.K.A. “Firehose”)
    - Is the easiest way to ingest streaming data and sink data in *near* real-time, in batches, into another service as Kinesis Data Analytics, S3, Redshift, or Elasticsearch (now OpenSearch).
    - Use case: when you need to batch streaming data, optionally transform and/or compress it, and place it into another service or storage, on a simple, serverless platform.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* + Kinesis Data *Analytics* (A.K.A. “Analytics”)
    - Is the easiest way to analyze streaming data in real time
    - Use case: when you want to perform basic windowed analytics on Streams or Firehose, e.g., for real-time or near real-time alerting, with SQL on a simple, serverless platform.



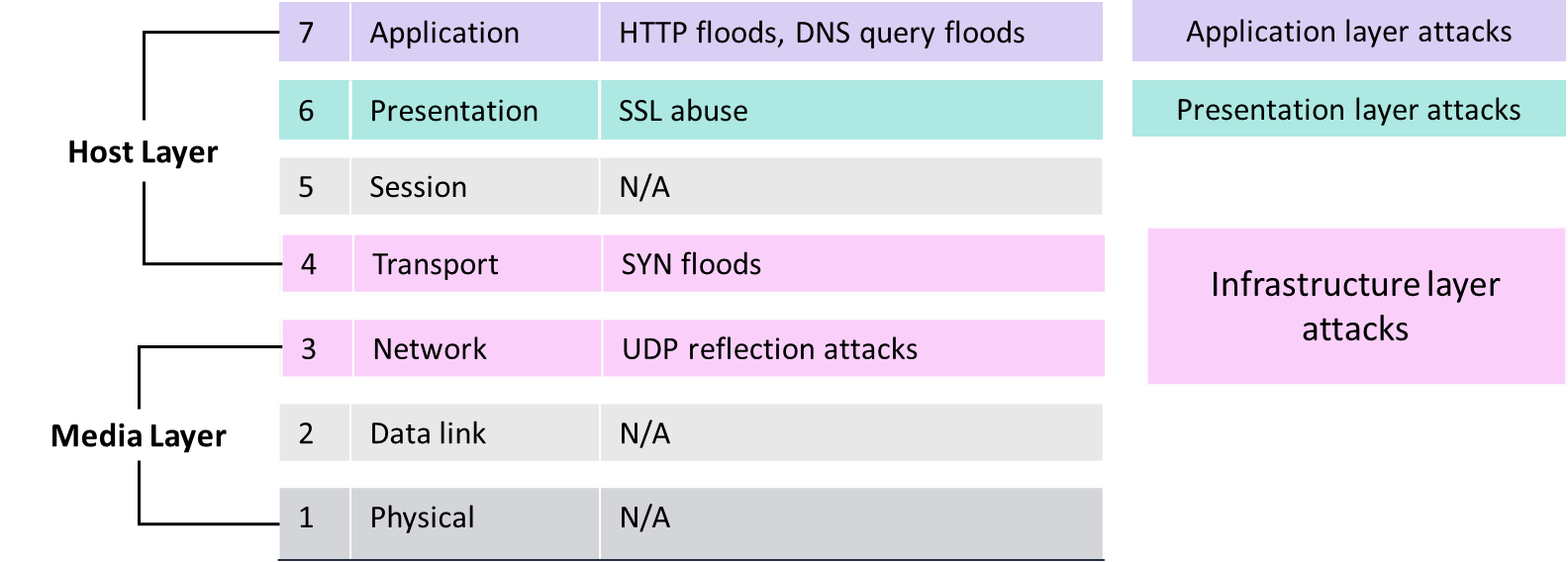
* + Kinesis Data *Video Streams*
    - Securely stream media from connected devices to AWS for storage, analytics, machine learning (ML), playback, and other processing



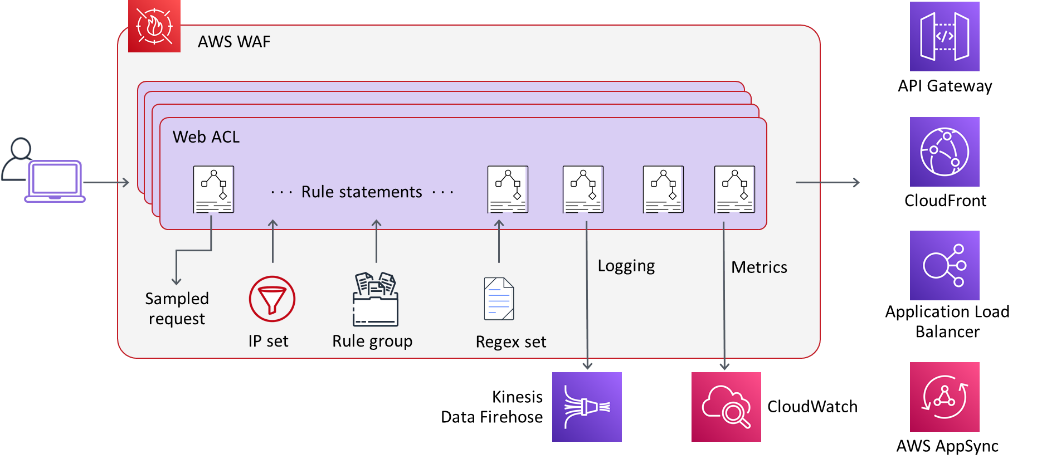
* + Other AWS services are available for ingesting, transforming, storing, and analyzing *streaming* data as:
    - Managed Streaming for Apache Kafka (MSK)
    - Spark Streaming with Elastic MapReduce (EMR)
    - Glue Streaming ETL
    - IoT Analytics within the IoT Core
* How to orchestrate Lambda functions calls using visual workflows? 🡪 Step Functions

Edge Services

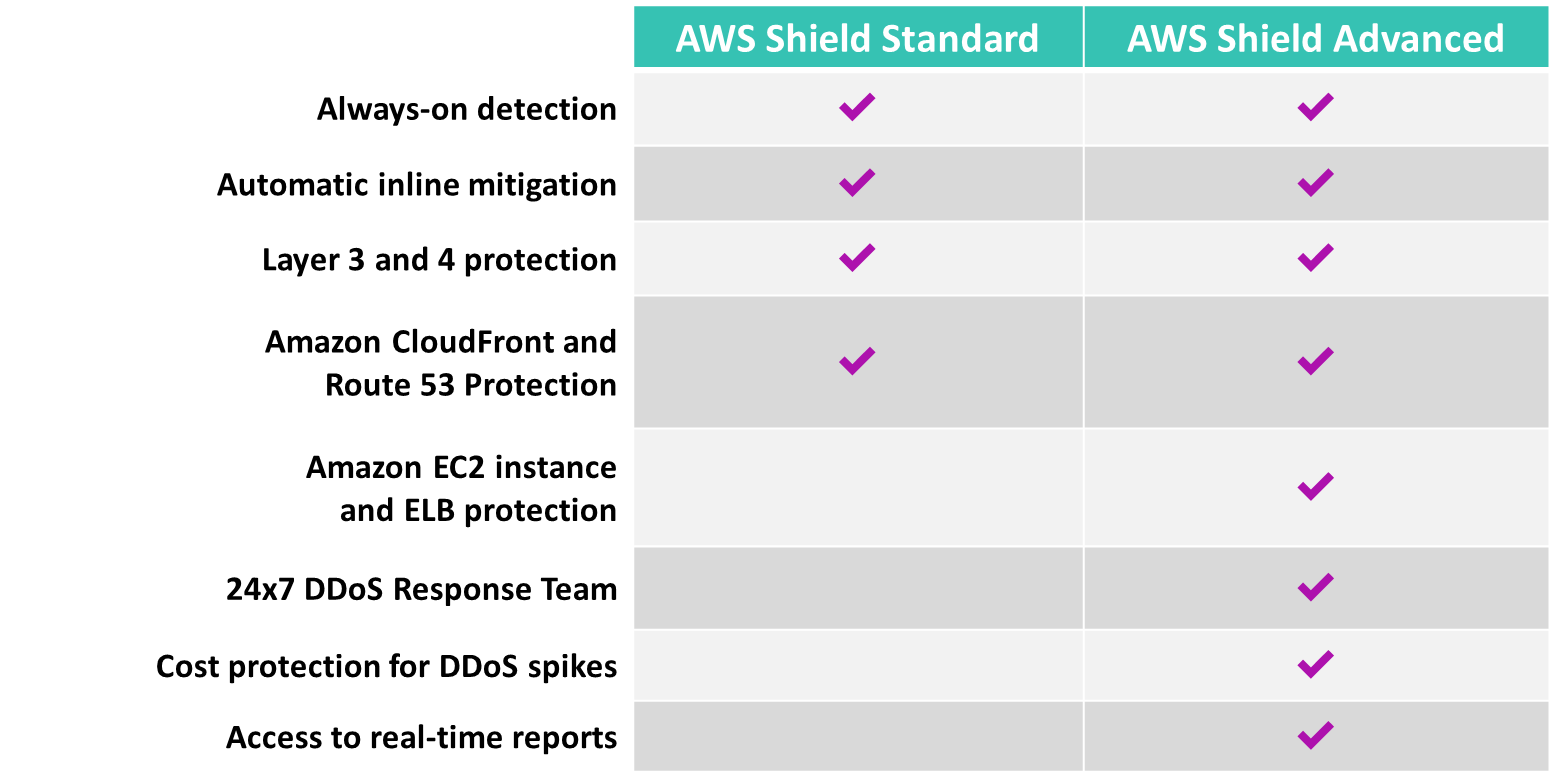
* Caching
  + What should I cache?
    - Data that requires a slow and expensive query to acquire
    - Relatively static and frequently accessed data
    - Information that can be stale for some time
  + Benefits of caching
    - It can improve latency and throughput
      * Proximity to the user.
      * In-memory data.
    - It can increase backend performance à Lower the load on servers, DBs, and storage, so they can finish faster what they are processing.
    - It can reduce cost
      * Because they have less load, smaller and cheaper resources can be used.
      * Pay to read the data only once.
  + Best practice 🡪 Implement caching at multiple layers of an architecture.
    - Edge 🡪 CloudFront
    - Session 🡪 ALB Sticky Sessions, fast database
    - Database 🡪 DAX for inline DynamoDB cache, Elasticache for side cache for RDS over EBS
  + Web tier/session caching à Session management (each web server will remember the authentication, access control and state data).
    - ALB sticky sessions à ALB route a request to the specific server managing the user’s session.
      * However, when scaling, it’s possible traffic might be unequally spread across servers as active sessions prevent routing traffic to the increased capacity.
    - For better scalability, instead of ALB Sticky Sessions, store and share between all web servers the session state in a very fast database (e.g., DynamoDB, DAX, Elasticache)
* Web caching à CloudFront
  + CloudFront is a global Content Delivery Network (CDN) service that serve *web content* (HTTP, HTTPS) close (then faster) to your customers
  + Provides network-layer optimizations for latency and throughput
    - Route each request to the edge location that provides the lowest latency
    - Use the highly available and congestion-free AWS global network to dramatically reduce the number of networks hops and latency for your users' requests.
  + Cache content accelerates delivery of S3 objects, websites, APIs, video content, or other web assets
    - Use *Lazy Loading* cache strategy
  + How to expire stale contents?
    - Time to Live (TTL) à Easiest if reading stale data is tolerable
    - Rename object à More effort but is immediate
    - Invalidate object à Last resort, when you can’t rename the object: the system must forcibly interact with all edge locations.
  + Supports WebSocket protocol 🡪 Persistent connections permits clients and servers to send real-time data to one another without the overhead of repeatedly opening connections
  + Integrated with AWS WAF and Shield to prevent or mitigate DDoS attacks.
* How to improve access to an application by users from all over the world? 🡪 Global Accelerator
  + Avoid internet congestion to keep packet loss, jitter, and latency consistently low
    - route TCP and UDP traffic to the closest AWS Edge Location
    - then moving it off the internet and onto AWS Global Network, optimizing the network path to the region where runs the application
  + Improve resiliency and availability 🡪 If Global Accelerator detects a failure of your application endpoint it instantly triggers traffic re-routing to the next available, closest endpoint in another AZ or AWS Region.
  + Simplify global traffic management by providing 2 static anycast IPv4 addresses that only need to be configured by users once.
    - Can associate these addresses to regional AWS resources or endpoints, such as Application Load Balancers, Network Load Balancers, EC2 instances, and Elastic IP addresses.
  + Protect your applications
    - Mask your application behind two static entry points.
    - Integrate with AWS Shield to prevent or mitigate DDoS attacks.
* S3 Transfer Acceleration
  + As see in the module *Storage: Object and File Storage*, S3 Transfer Acceleration also use the closest *Edge Location* to optimize the network path to the S3 bucket and move S3 data faster over long distances.
* DDoS mitigation
  + Attack overview
    - *Denial of service* (DoS) attacks are characterized by an explicit attempt by attackers to prevent legitimate use of a service.
    - Typically accomplished by flooding a resource or system with needless requests. Systems are overloaded and preventing some or all legitimate requests from being fulfilled.
    - A *Distributed Denial of Service* (DDoS) is a DoS attack where the perpetrator uses more than one unique IP address, often thousands of them. Because the incoming traffic flooding the victim originates from many different sources, it is impossible to stop the attack simply by using ingress filtering.
    - DDoS attacks can be isolated by which layer of the OSI model they attack. Most common are:
      * Infrastructure layer attacks (at layers 3 and 4)
        + Aim to overload the capacity of the network or the application servers
        + Have clear signatures and are easier to detect
        + User Datagram Protocol (UDP) reflection attacks: An attacker can spoof the source of a request and use UDP to elicit a large response from the server. The extra network traffic directed towards the spoofed, attacked IP address can slow the targeted server and prevent legitimate users from accessing needed resources.
        + SYN (synchronized) flood: The intent of an SYN flood attack is to exhaust the available resources of a system by leaving connections in a half-open state. When a user connects to a TCP service, like a web server, the client sends a SYN packet. The server returns an acknowledgment, and the client returns its own acknowledgement, completing the three-way handshake. In an SYN flood, the third acknowledgment is never returned, and the server is left waiting for a response. This can prevent other users from connecting to the server.
      * Application layer attacks (at layer 7)
        + Attempts to overload specific functions of an application
        + DNS query flood: In a DNS query flood, an attacker uses multiple DNS queries to exhaust the resources of a DNS server.
        + HTTP flood/cache-busting attacks: With an HTTP flood, including GET and POST floods, an attacker sends multiple HTTP requests that appear to be from a real user of the web application. Cache-busting attacks are a type of HTTP flood that uses variations in the HTTP request's query string that prevent use of edge-located cached content and forces the content to be served from the origin web server, causing additional and potentially damaging strain on the origin web server.



* + *Edge Locations* are DDoS-resilient by design.
    - AWS Global Infrastructure is highly scaled
    - More than 230 Edge Locations can significantly increase your ability to absorb DDoS attacks and isolate faults while minimizing availability impact.
  + AWS *Web Application Firewall* (WAF)
    - Covers layer 7 (Application) in the OSI model
    - Web ACLs *rules* can allow or block:
      * IP addresses that request originate from
      * Country that requests originate from
      * Values in request headers
      * Strings that appear in requests, either specific strings or strings that match regular expression (regex) patterns
      * Length of requests
      * Presence of SQL code that is likely to be malicious (known as SQL injection)
      * Presence of a script that is likely to be malicious (known as cross-site scripting)
      * Etc.
    - Rules can be:
      * *Managed Rules* provided by:
        + AWS
        + Marketplace Sellers
      * Your own *Custom Rules*
    - Can *monitor* WAF using *CloudWatch Metrics* and record them for two weeks
    - Can activate *logging* to get detailed information about traffic that is analyzed by your web ACL, and send them to *Kinesis Data Firehose* to S3 for storage or analysis using Redshift, Amazon Opensearch (Elastisearch), etc.
    - Can protect:
      * CloudFront
      * API Gateway
      * Application Load Balancers
      * AppSync GraphQL API



* + AWS *Shield*
    - Covers layers 3 and 4 (Network and Transport) in the OSI model
    - Cloud-native
    - Two tiers:
      * AWS Shield *Standard*
        + Always-on DDoS protection
        + At no additional charge
      * AWS Shield *Advanced*
        + Add functionalities to Shield Standard, including since December 2021 automatic layer 7 (application) DDoS mitigation
        + USD 3000 monthly fee + $25-$50 @TB traffic

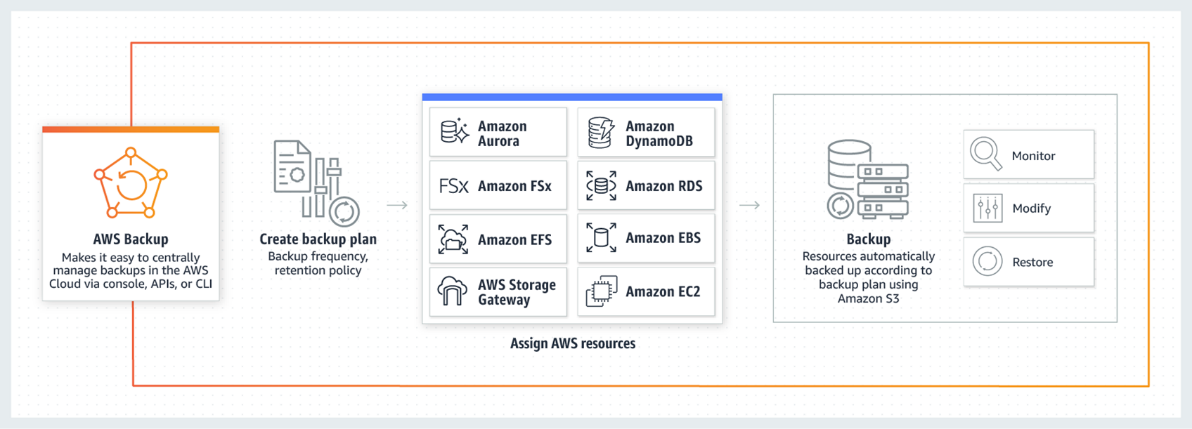


* AWS Outpost
  + Is an extension of your AWS Region to your on-premise location, enabling *low latency* and *local data* processing and storage, for a truly consistent hybrid experience.
  + Is a fully managed service that *run on-premises* a factory configured 42U rack, or 1U or 2U servers, with AWS-designed hardware.
  + Support local EC2 for compute, ECS and EKS for containers, ALB for distribute traffic, RDS and Elasticache for databases, EBS and S3 for storage, EMR for data analytics (and some other AWS services).
  + After installation, you can create a subnet in your existing regional VPC and associate it with Outpost in your location, just as you associate subnets with an AZ in an AWS Region.
  + AWS Outposts are not designed for environments with limited to no connectivity.
  + How can I establish network connectivity between my Outposts and the parent AWS Region? 🡪 Direct Connect, VPN, or the public Internet.
  + By April 2022, available in Americas for Brazil, Argentina, Chile, Peru, Ecuador, Colombia, Costa Rica, Puerto Rico, Mexico, USA and Canada.



Disaster Recovery Planning

* Fault tolerance and high availability
  + Fault tolerance and high availability *are often used interchangeably but are not the same thing*.
  + For AWS:
    - *Fault-tolerance* is a system’s ability to *remain in operation* even if some of the resources used to build the systems fails, through the use of redundant components.
    - *High availability* is not about preventing system failure, but the ability of the system to *recover quickly* from it.
* How to get your application running again within an appropriate amount of time and at an appropriate level of cost?  *Using Amazon Web Services for Disaster Recovery* <https://1drv.ms/b/s!AllwbeWFfFjNg5ggU-EaBZcAEOASFQ?e=22U2l5>
* AWS advantage for Disaster Recovery
  + Do not need to restore a tape backup (except for Storage Tape Gateway)
  + Can restore EBS snapshots in seconds
  + Replicas can be accessed directly from AWS
* *Before* a Disaster Recovery failover, data must be duplicated to another AWS region.
  + AWS tools and services for storage replication to another Region
    - S3 à Cross Region Replication (CRR)
    - EFS, FSx à AWS DataSync
    - EBS à Copy snapshot to another region
    - RDS à Cross Region Read Replica and Aurora Global Database
    - DynamoDB à DynamoDB Global Tables
* AWS Backup
  + For years, AWS has offered tools to backup data stored in AWS storage services. For example, you can use Data Lifecycle Manager to automate EBS snapshots,
  + However, in 2019 Amazon took its backup offering one step further, launching a dedicated service named AWS Backup, which lets you *automate* and *centrally* *manage* backups across AWS services.
    - It lets you backup data from Amazon EFS, DynamoDB, RDS, EBS, etc. and also from on-premise resources via the AWS Storage Gateway.
    - It doesn’t replace the previous backup technologies offered by AWS. It is a central management interface that brings those technologies together, letting you orchestrate and schedule backups easily in one place.



* + How Does it work? 🡪 create a backup plan, will consist of one or more backup rules to apply to resources assigned to the plan:
    - Backup rules 🡪 define a backup, specifying a schedule, frequency, and backup window. A backup rule can be applied to specific resources, or to groups of resources defined by tags.
    - Lifecycle rules 🡪 define what happens with old backups. For example, you can use lifecycle rules to move backups to cheaper storage tiers, or delete them completely, after a specified period of time.
    - Vaults 🡪 backups are grouped into vaults, and each vault is encrypted by a Key Management Service (KMS) key.
* Disaster Recovery strategies to failover an on-premises datacenter over AWS
  + In all three strategies, resources required to support data replication are always on
    - On-premises data can be replicated to AWS using:
      * AWS DataSync
      * AWS Storage Gateway
      * AWS Database Migration Service (AWS DMS)
      * 3rd party tools and services
  + Pilot light à Active/Passive.
    - Disaster Recovery VPC are created
    - Compute resources are created in the Disaster Recovery VPC and switched off
    - In a disaster, switched off resources are restarted using a script (or if they cannot be switched off as for NAT Gateways, they are created using CloudFormation).
    - Finally, Route 53 route traffic to the DR VPC.
  + Fully Working Low-Capacity Standby à Warm standby.
    - A scaled-down version of a fully functional environment is always running in a Disaster Recovery VPC.
    - It can be used for non-production work, such as Quality Assurance, Data Warehouse, etc., releasing resources from the production site
    - In a disaster, the system is scaled-up automatically using *auto-scale* to handle full production load.
    - Finally, Route 53 route traffic to the Disaster Recovery VPC.
  + Multi-Site Active-Active
    - A fully functional system running in AWS at the same time as the on-premises systems.
    - Active/Active configuration, with a proportion of traffic will go to your infrastructure in AWS, and the remainder will go to your on-site infrastructure.
    - In a disaster, the system is scaled-up automatically using *auto-scale* to complete full production load.