Preliminary note

This document is served as part of this article published on our blog and is intended as a compendium to that. To fully understand the following concepts, please refer to that and the official AWS documentation.

In-depth Concepts

Even if I suggest giving at least a quick read to all the topics proposed here, I'll try to summarize what I think are the most critical concepts you need to know to try this exam. The idea is not to write down everything about every service, which will result in an online course, instead present to the reader some of the aspects that let me understand questions better.

Please, note that hands-on experience is irreplaceable, so what I'm describing here may not be enough for you; in that case, I'll suggest, go over FAQs again for every unclear area you'll need to cover.

Code Build

Fully managed build service: can build from **CodeCommit**, **S3**, **BitBucket**, **GitHub**/**GitHub Enterprise**.

It can manage **Build** & **Test phases;** generally, remember that **unit tests are usually done here**.

CodeBuild Agent is used to simplifying Debugging procedures, as it can debug locally.

CodeBuild uses a **buildspec.yml** file which has the following structure.

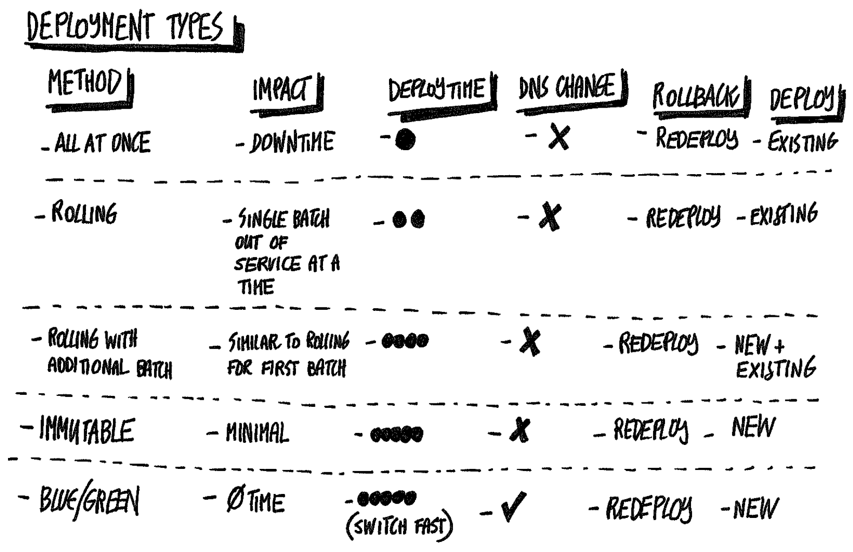
It allows for **local testing** and can deploy **multiple artifacts simultaneously** using the option **runOrder with a fixed number.**

It allows for deploying **On-premise;** it can **roll back** artifacts to the previous version and apply **In place** / **Rolling** / **BlueGreen** deployments.

A **deployable content** is a mix of **source code + appspec.yml.** A couple defines a **Revision**. A **Revision** can be stored in either **S3** or **CodeCommit** and can be deployed on **EC2s**, **Lambdas**, **Containers**.

Deployment Strategies

Deployment strategies are valid in general and do not apply only to CodeDeploy. Here is a quick list that describes different approaches. Check them in detail here. Also, here is a list I made from slides to quickly show how they work.



DEPLOYMENT TYPES

METHOD	IMPACT	DEPLOY TIME	DNS CHANGE	ROLLBACK	DEPLOY
- ALL AT ONCE	- DOWNTIME	- ●	- X	- REDEPLOY	- EXISTING
- ROLLING	- SINGLE BATCH OUT OF SERVICE AT A TIME	- ●●	- X	- REDEPLOY	- EXISTING
- ROLLING WITH ADDITIONAL BATCH	- SIMILAR TO ROLLING FOR FIRST BATCH	- ●●●●	- X	- REDEPLOY	- NEW + EXISTING
- IMMUTABLE	- MINIMAL	- ●●●●●	- X	- REDEPLOY	- NEW
- BLUE/GREEN	- Ø TIME	- ●●●●● (SWITCH FAST)	- ✓	- REDEPLOY	- NEW

AWS CodePipeline

Automate actions from **code** to **deploy.** It's safer, faster, and standardized, so it's the preferred way to deploy with AWS. Typically in exams, a **good 60%** of questions include an AWS pipeline approach to investigate. Using CodeCommit, CodeBuild, and CodeDeploy, it can recover, build, test, and deploy code changes.

Cross Account Resources

Remember, it is possible to build and deploy resources from an account different from the one you're running your pipeline into. To do so, check this example.

AWS CloudFormation

Allows for **Infrastructure as Code**, **versioning, replicating, updating** templates like code.

IaC becomes a need when we strive for reusability and to make each infrastructure independent.

Runs **automated testing** for CI/CD environments, **integrates** with CI/CD environments and management tools. **Support For Chef and Puppet integrations.**

AWS CloudFormation components

AWS CF template: it's a blueprint for building AWS resources. It can be used to replicate infrastructures exactly across the region. Check all its components in detail, especially **custom resources, depends on** and **wait conditions.**

Nested Stacks

They are created from other stacks to make portions of IaC reusable, using **AWS::CloudFormation::Stack**; a typical approach as infrastructures of projects tend to grow, and we need a way to extract common patterns.

Nested Stack vs. Exporting Values

Nested stacks are usually exploited when we need to **isolate** IaC components with common patterns; on the other hand, **exporting values** is a technique to use when we want to pass information from one stack to another, especially when the entire IaC is not under our complete control.

Manage errors during rollbacks

When we use nested stacks, it is not uncommon to have **UPDATE_ROLLBACK_FAILED** state for stacks starting from the ROOT one, and the idea is to use the **continue-update-rollback command**, passing a list of resources to skip.

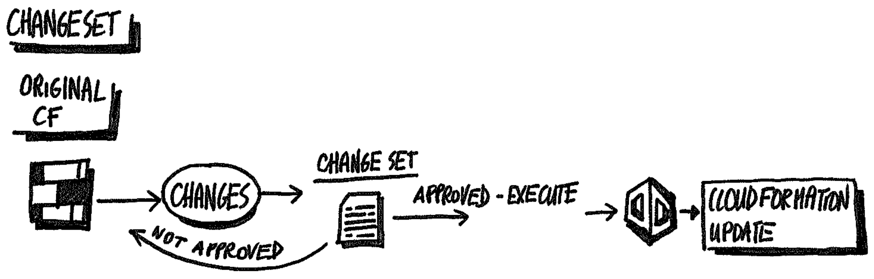
Note: changes to a deletion policy, an updated policy, a condition, or output declaration are **not** considered an update; to manually trigger an update, we need to **mock a change to metadata**. Also, some resources like ASG, RDS, Certificate Manager, REDSHIFT, ES, Cloud Formation Stacks are very long to create, so we need to extend **timeout periods**.

Depends on

“Depends on” is a property to define that **a Resource (even a task) depends on** the **completion of another resource**. A **WaitCondition** resource requires a **WaitConditionHandle**. There are examples of WaitConditions you can use to learn and can be found here.

Change Sets

Presents a summary of the proposed changes when we update a stack set. We need to approve a changeset before doing an update. We can add other changes before approving.



Stack Sets

A group of stacks is managed in a single template described in an ADMIN account; then, with a TRUST RELATIONSHIP policy, we can **create, delete, and update** stacks in target accounts in different regions.

Scripts Helpers for CloudFormation

cfn-init: called once to check metadata in the template.

cfn-hup: monitors for changes in the metadata in the template.

cfn-signal: send COMPLETE or FAILURE signal to WAIT CONDITIONS or CREATE POLICY.

cfn-get-metadata: get the current metadata in the CF stack.

Note: CloudFormation can be deployed using BeanStalk / Opswork / Normal CI/CD.

Elastic Beanstalk

Environments

Has web servers and workers (two different operational tiers) and admits only one app per environment.

It can have multiple environments per app and can make use of ASGs. Runs a single app version for scalability, and an environment can be seen as a version applied to resources.

App Versions

App code resides on S3. It can be deployed in different environments. Versioning is supported for enabling rollbacks. Allow rolling updates, and each version points to another S3 Object.

Configurations

It is used to launch new environments quickly for rollbacks. Allow for installing yum packages and supply additional daemon config. Ultimately defines how an environment and its resources behave.

An update through BeanStalk can be done in **2 ways**:

<u>In place</u>: we change the version of the application in place

<u>Environment Swap</u>: we clone and swap the environments

EB Extensions

Inside an application bundle stored in S3, we can have a directory called .ebextensions, where we can save one or more **config.yml** for customizing and adding commands for the installation process.

Docker

Elastic Beanstalk supports Docker and containers in general in the form of containers deployed on EC2 instances. **Typically in questions when we are asked what to use to manage Docker containers without extra operational cost easily, Beanstalk is the right choice**.

Web Environment Tier

It is used to process web requests. Creates an ELB, ASGs, and some EC2 instances which run container-based apps.

The **host manager** is a daemon running on every instance. It helps in Control, Deploy, Update apps and in generating and rotating logs to S3.

Every environment has a CNAME URL that points to an ELB, which is aliased over a Route53 URL.

Worker Environment Tier

It is used to run background jobs for the Web Tier. It has EC2s, ASG, and an IAM role.

It always has an SQS queue created by default. SQS Queue has Dead Letter Queue for managing failed requests and is, in general, used to let Web and Worker tier communicate with each other.

In general

CloudFront can be used to distribute content as well. Support for VPC and is natively integrated with CloudTrail.

RDS can be used, **but** it is best to deploy it **outside of BeanStalk** to destroy it at each deployment. The correct procedure is just to program your application to communicate with an external RDS (update: recently Elastic Beanstalk has introduced a new RDS lifecycle management, but for the questions you'll have to stick with this explanation).

S3 buckets for Elastic Beanstalk **already have Deletion Protection enabled by default** (different from other services).

Elastic Beanstalk Deploy methods

Elastic Beanstalk allows for the same deployment methods **shown before for CodeDeploy**.

AWS Opswork

It is an Application Infrastructure management with the ability to change configuration over time.

Works with:

- **Chef Automate**: a fully managed Chef server with docs & community, dynamic ASG, servers on-prem or on EC2.
- **Puppet Enterprise**: a fully managed puppet primary server, docs & community, an auto-scaling service to register and provide new nodes. Auto healing by checking status against a primary server with autocorrect.
- **Opswork Stack**: the most flexible solution, supporting any server, with AWS docs & community, autoscaling, and auto-healing. Support bash, chef, Powershell, etc.

Opswork Stack

It uses chef recipes: when an update is found or manually triggered, a recipe is run on the instances. A combination of a **recipe + metadata** allows the running of various commands.

What is a stack

It defines some common configurations like for AWS regions. It is a logical container for resources that must be managed together. It can be run in a VPC to isolate resources from direct external user interaction.

An instance can host multiple layers.

Note: Stacks can be organized in layers, and layers are **specialized groups**, for example, serving apps or hosting a DB. A layer depends on chef recipes. All recipes can be stored on a cookbook, and the cookbook can be stored on S3 or Git.

Extra info

Opswork Stack installs an **agent** who is also responsible for auto-healing capabilities; whenever it is not responding, the instance is rebooted.

You can have multiple *apps x stacks x layer.*

An instance in Opswork can be:

- **24/7**: an instance that can be launched and stopped manually
- **Time-based**: which can be run on a schedule
- **Load-based**: start and stopped based on specific metrics.

IAM

IAM allows for managing and enforcing correct security for AWS accounts. We have **Users for permanent access**. We have **Roles for temporary access.**

A Role must be assumed with a **Trust Policy**. When a role is assumed, or a role is given to a User or a Resource, it has an **Access Policy,** which provides access to specific actions to specific resources. Roles **are used for programmatic access.**

Data Protection flow

Some questions usually request to find the correct flow to protect data from client to S3 or from client to an EBS or EFS file system.

Data protection in transit

From customer site to AWS Cloud, we would like to ensure: TLS in-transit encryption, IPSEC VPN for encapsulating on-prem traffic to the cloud, AWS certificate manager to generate public certificates.

Data protection in flight

You can use SSL Termination at **EC2 layer** or SSL Termination at **ELB** with certificates stored in IAM (multiple certificates per ELB) or even Amazon CloudFront Terminating SSL.

Data protection at rest

You can use Amazon S3 SSE, Amazon EBS Volumes, or server-side encryption. Amazon Glacier is encrypted by default.

Amazon Inspector

It is used to check compliance with **AWS Best Practices.** It’s an automated assessment that helps in improving the security and compliance of Applications. It's agent-based and can detect vulnerabilities and generate reports.

AWS System Manager

It is a collection of services to automate procedures and common commands to manage fleets of EC2s and on-prem machines. For access, logging, patch maintenance, collecting software inventory, configuring OS like Windows and Linux, and applying OS patches.

Some of the most requested (and essential) components in questions are:

Automation Document: to create AMIs with required versions using JENKINS or CodeBuild.

Run command: run predefined commands or create custom ones. Choose instances based on tags, define a schedule or control them directly, finally send orders.

Patches: monitor and audit for necessity, create a patch baseline, then schedule an apply command, finally apply the patch.

Parameter Store: uses KMS to protect a uniquely named parameter along with its value.

Notes:

Parameter Store from System Manager: This is generic and also used by **CodeBuild**.

AWS Secret Manager: specific for credentials, has KMS encryption, allows for automatic credential rotation.

AWS Service Catalog

Allows managing in a centralized way many different IT products, which can be an AWS service, a virtual machine, or **an entire infrastructure**.

Allows for the versioning of **products** while a **portfolio** is a collection of products.

IAM is used to give specific access to portfolios to launch particular products in it. **Constraints** are rules that can limit and in general, manage the usage of particular products. They can be used, for example, to limit launching resources based on cost usages.

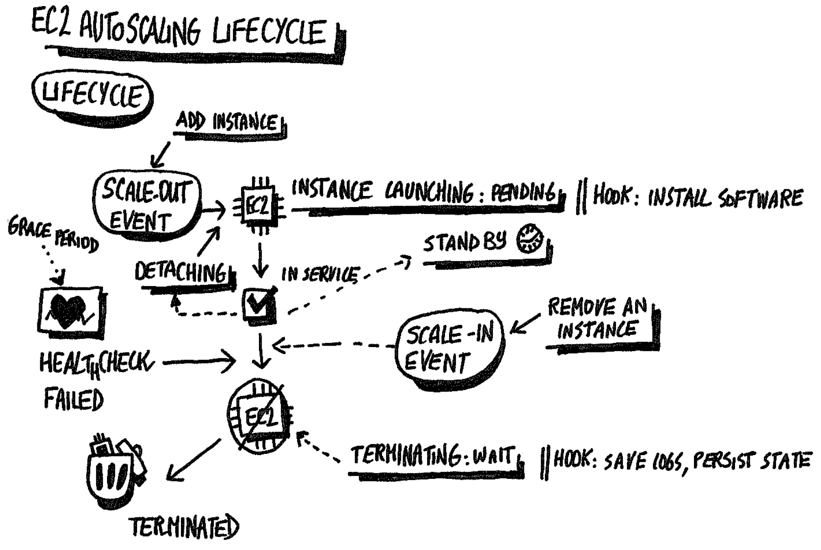
Every resource that is launched is done with an AWS CloudFormation Stack

There are also service actions that can govern what kind of possibilities are offered to end-users to manage some apps, do troubleshooting, and so on.

Finally, it can create **Golden AMIs** with pre-hardened standards.

EC2 Autoscaling and its lifecycle

When an instance is put under an autoscaling group, its life and death are managed by an exact lifecycle. Here is a scheme depicting all the phases that an instance can pass from creation to decommission.



Some notes:

- Spot instances can be terminated **before** lifecycle events, so be aware!
- On every lifecycle event, CloudWatch event launches a Lambda if we configure it.

Tricks to Suspend an instance:

- **Disable AZ**: instances are still registered, but ELB will not send traffic
- **Configure Slow Start**
- **Deregister/Delete a Target Group**
- Delete a Load Balancer if termination protection is **not** enabled.

Multi AZ, Multi Region, HA

Some notions about high availability in AWS, considering the most used services in the exams.

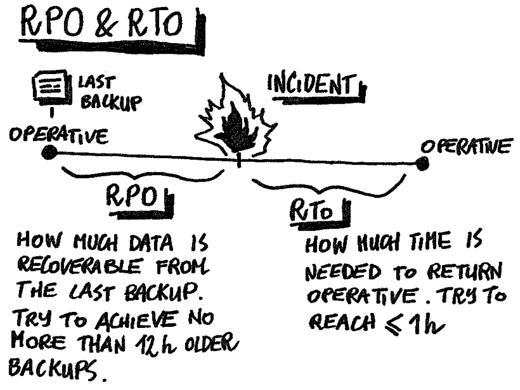
In Route53, we can provide **health checks** for checking the status of an application, then we can use **DNS failover to switch** to a different application or service page for **DR**.

- **For RDS,** we have Cross-Region read replicas and Cross-Region snapshots.
- **For EBS,** we have Cross-Region snapshots.
- **For DynamoDB,** we can use Global Table and DynamoStream.
- **For S3, we have a Cross-Region** Replica.

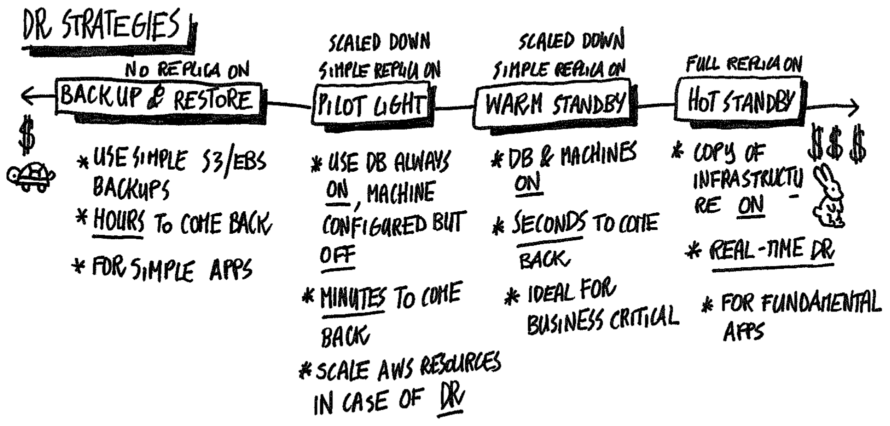
- **For EC2 AMI,** we can use a Cross-Region copy.
- **For Lambda@Edge,** we can use Lambda at Edge Location of CloudFront (not entirely true, and in general extremely costly) in case of requests and responses.

RPO/RTO

These values are highly requested in questions, meaning that you have to understand the architecture and make it fall under one of the suggested strategies.


RPO & RTO
LAST BACKUP
INCIDENT
OPERATIVE
OPERATIVE
RPO
RTO
HOW MUCH DATA IS RECOVERABLE FROM THE LAST BACKUP. TRY TO ACHIEVE NO MORE THAN 12h OLDER BACKUPS.
HOW MUCH TIME IS NEEDED TO RETURN OPERATIVE. TRY TO REACH ≤ 1h

The following schema illustrates all the most critical strategies and how they work:



Route53 deployment Techniques

Weighted Distribution:

- Allows for moving traffic from % of original app to newer versions.
- Allows for canary deployments where a small % of traffic is directed to the latest version.
- Allows reading **elastic IP**, **public IP**, **Elastic Beanstalk tier**, etc.
- It can also be set to 0% - 100% for standard B/G.

Update ASG Launch Configuration Technique

ASG can **only have one launch configuration** at a time. So every new instance in a **scale-out** event will have the **new** launch configuration, while old ones will still have the old configuration. Because **standard** termination policy requires to delete old configuration instances first, by **doubling the instance count** then **reducing it to half** again, you are saying to add a complete set of new instances in the same number as the old ones, then remove all the old ones.

Rollback: do the same using the old configuration instead of the new; the old configuration will replace the new one using the doubling and reducing technique.

AWS CloudFormation best practices

Organize Stacks by lifecycle and ownership: try to group resources in stacks that have a common lifecycle or ownership. Ownership has the advantage of helping to configure resources so that they don't interfere with others.

Cross-Stack references: they are helpful to pass resources from one stack to another. We use the "export" output field in one stack and use *fn::ImportValue* in another stack to import the value. Exported names must be unique per region; importing can happen only in the same region.

Changesets are not enforced by design, so, at least in production, it is better to create them before updating resources.

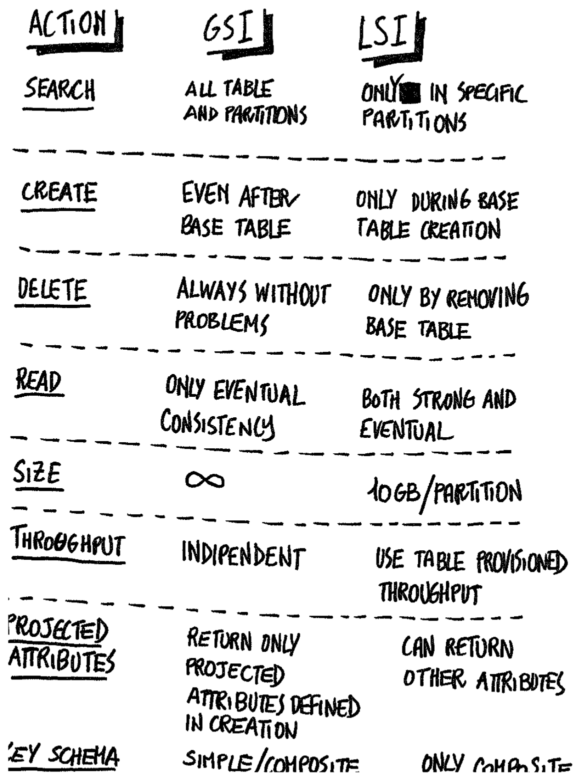
DynamoDB best practices

Secondary indexes: are used to access data with property other than the primary key efficiently. A secondary index is composed of at **least one index key and a sort key.** It is associated with one table and can be composed of attributes from the table.

Global secondary index: can have partitions and sort keys utterly different from the base table. Updates or deletes are made **async**. They are considered globals because they span on all table items and partitions.

It provides 1 RCU with 2 eventually consistent reads for data ≤ 4Kb. It provides 1 WCU with 1 Kb size. We need: 1 WCU for new, update or delete in the table with projected items; 2 WCU for updates on **key attributes** because under the hood is a delete and a recreation.

Local Secondary Index: must have the same partition key as the base table.



ACTION	GSI	LSI
SEARCH	ALL TABLE AND PARTITIONS	ONLY IN SPECIFIC PARTITIONS
CREATE	EVEN AFTER BASE TABLE	ONLY DURING BASE TABLE CREATION
DELETE	ALWAYS WITHOUT PROBLEMS	ONLY BY REMOVING BASE TABLE
READ	ONLY EVENTUAL CONSISTENCY	BOTH STRONG AND EVENTUAL
SIZE	∞	10GB/PARTITION
THROUGHPUT	INDIPENDENT	USE TABLE PROVISIONED THROUGHPUT
PROJECTED ATTRIBUTES	RETURN ONLY PROJECTED ATTRIBUTES DEFINED IN CREATION	CAN RETURN OTHER ATTRIBUTES
EY SCHEMA	SIMPLE/COMPOSITE	ONLY COMPOSITE

Global Tables: allow for multi-master cross-region replication of a table; they do not support multi-account replication (only in one account). They need Dynamo Stream activated and can help with DR and reduce latency for local region reads.

DynamoDb Stream: maintain in memory a volatile copy of recent updates operations. They accept no duplicates. It can be used for multi-region replication but cannot be used with a VPC endpoint.

DynamoDb Trigger: can launch custom lambda on per-item updates. Can be attached to a DynamoDb stream

S3 permission strategies

S3 permissions are classified into **Resource-based** and **User policies**. During cross-account upload, the object owner is the one uploading a file, **but** the bucket owner can still deny access or delete the file.

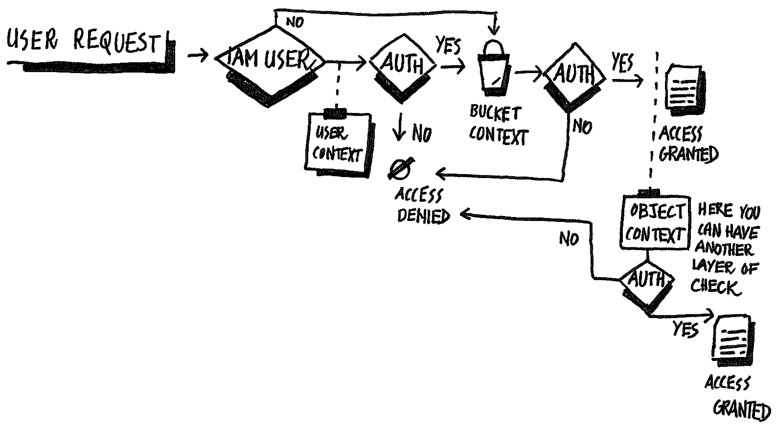
User policies or bucket policies through IAM can control access of users internal to an AWS account. User policies can be attached to a user, a group, or a role.

Resource Policies and ACL:

Bucket policies can be used to grant cross-account access to other AWS accounts to users. Bucket policy can be associated only by account owning bucket.

ACLs: are used whenever we give access to resources that can't have a role attached and can't be used with IAM users! ACLs usually have a sense when talking about accounts. Can be used to grant access to resources to other AWS Accounts; permission can be given by email or canonical ID.

Bucket ACLs are usually needed only to grant permission for writing access logs; it is given to log delivery groups. Object ACL: is the only way someone can manage an object not owned by him—for example, a cross-account bucket.



Web Identity Federation

For services like Amazon, Twitter, G+, Facebook which exchange their auth method and grant responsibility upon user identity. We use the IdP ARN as the principal in the trust policy plus a condition for something unique like an app id.

We use **assumeRoleWithWebIdentity**, and after that, STS gives a unique user id to say that the operation was successful.

Assume role with SAML

When on IDP like Google, Okta, OneLogin, Azure, we can use SAML to authenticate. We can use **AssumeRoleWithSaml** with STS. SAML is not only for having temporary credentials but also for logging into the console.

Custom Identity broker

Auth the user locally using a custom identity broke, usually for logging into on-prem particular identity repo. Call STS assume the role on the user's behalf, getting temporary credentials.

Call a Federation Endpoint, passing the temporary credentials, to obtain a valid, 15min long, signed url for the console.

AWS Config

Provides a detailed view of the configuration of resources on AWS, keeping an inventory. It gives point-in-time or historical values for changes in resources. AWS Config accumulates quick changes in a single cumulative change.

AWS Config gives an API to account for resources not covered directly from the service.

- **AWS Resources**: entities discovered and managed by config.
- **AWS Config Rules**: help define a specific configuration for the resources and invoke lambda for custom logs.
- **Resource Relationship**: AWS Config automatically discovers relationships between resources.
- **Configuration Item**: a point-in-time view of the supported AWs resources.
- **Configuration Snapshots**: a collection of configuration items for a particular point in time.
- **Configuration History**: a collection of items in any time period.
- **Configuration Stream**: is a stream recording all changes.
- **Config Record**: is used to transform elements in a stream into configuration items.