**Accounts and Organisations**

**AWS Organizations:**

Allow us to **centrally manage** multiple accounts and **apply policies** (permissions) to those accounts.

You can also get one centralised **consolidated bill** across the connected accounts.

Must have a **Management Account** (main account where you create the organization)

Can create **additional accounts** or join existing accounts to the organization.

You can form a hierarchy with **Organizational Units** (OUs), OU can have multiple accounts associated.

You can create accounts **programmatically** through the Organizations API.

You can enable **AWS SSO** (Single Sign On) using an on-prem directory.

Can enable CloudTrail to record all API actions in the accounts in the Organization.

When you create an account with AWS Organizations, a role is created: **OrganizationAccountAccessRole,** this role has full permissions in the account. This role can be assumed by any user with the sts:AssumeRole permissions. In the real world these policies should be locked down to only specific users. You can alternatively modify the role **OrganizationAccountAccessRole.**

**Service Control Policies (SCP):**

**Limit** and **control** the maximum available permissions in the accounts of an AWS organisation. **Does not grant permissions**, simply limits what you can do if you do have the permissions.

For example, IAM gives you the permissions/ privileges to perform an action, and the SCP controls whether you are allowed to in that particular account.

Can also control tagging and API actions available in that account.

Think of it as: SCP is for Member Accounts and OU’s, IAM is for users, groups, and roles.

Users and Roles in the **Management** account in AWS are not restricted by SCP’s, only the **Member** accounts in the organisation. Best practice is to not create all users in management account.

Can be applied to OUs or to the accounts themselves. Accounts can also **inherit** SCP’s i.e. if Prod is a child account of Dev then Prod will have the same policies as Dev plus any of its own restrictions.

Only deny permissions are inherited, allow permissions must be applied at both parent and child levels of an OU or account.

|  |  |
| --- | --- |
| **Deny List Strategy**  **(deny overrides allow)** | **Allow List Strategy**  **(allow overrides deny)** |
| The FullAWSAccess SCP is attached to every OU and account. | The FullAWSAccess SCP is removed from every OU and account. |
| All permissions allowed by default. | Meaning all permissions are denied by default. |
| Must set a deny to alter default permissions. | To allow a permission, SCP’s with allow statements must be added to the OU or account including root. |
| Allows all permissions to flow down from root. | Every SCP in the hierarchy must allow the API’s you wish to use. Can be cumbersome. |
| Can override with a deny in an SCP at levels below management account. |  |
| **Explicit** **deny** always overrides an **allow**. | Explicit allow overrides implicit deny. |
| The default setup. | Either option depends on use case. |

**Explicit Deny** always takes **precedence** (deny at any level overrides an allow)

SCP’s which **deny** are inherited from a parent account.

**Tag Policies** can be used to standardise tags applied to resources.

**AWS Control Tower** – Extension to AWS Organizations that provides additional control.

**Landing Zone –** Creating a configuration in Control Tower is known as creating a **Landing Zone** which is a well architected multiple-account baseline based on best practices. Can set up OU’s and accounts.

**AWS Control Tower** – with a **landing zone**, you can set up an organization (if you don’t have one already), multiple AWS accounts and OU’s (Security OU, Sandbox OU and Production OU).

**Guardrails** – used by Control Tower for **governance** and **compliance**: **Preventative** **guardrails** (based on SCP’s and disallow API actions), **Detective** **guardrails** (implemented using AWS Config rules and Lambda functions/ monitor and govern compliance).

Root user for management account can perform actions that guardrails disallow.

**AWS Security Token Service (STS)** – returns short lived, temporary credentials

**sts:AssumeRole** – API call in the policy to assume role

**Temporary credentials** include: AccessKeyID, Expiration, SecretAccessKey, SessionToken

**Identity Management and Permissions**

**Authenticate:**

Through the **IAM** service:

**Console**, **AWS** **CLI**, **API** (authenticating through code): ways to authenticate and access resources:

CLI and API are **programmatic** (through AWS API not Console) methods of accessing AWS. They use an **Access Key ID** and **Secret Access Key**

**IAM Principals** must be authenticated to send requests.

**Users**, **Roles**, **Federated user** (user from another source: Facebook, Google etc.), **Application**

**Principal** – person or application making a request for an action or operation on an AWS resource

**IAM Policies** – allow or deny a request for an action/ operation on resources

**Inline policy:** 1-1 relationship with the user group or role. These policies are not shared across other roles or other entities. E.g., If you delete the user you delete the policy along with it.

**Managed policies:** **AWS** managed, or **Customer** managed. Can attach to multiple identities unlike inline policy.

|  |  |
| --- | --- |
| **Types of policy** | |
| **Identity based policy** | **Resource-based policy** |
| Applies to users, groups and roles | Applies to an actual AWS resource e.g., S3 bucket |
| JSON permissions policy documents that control what actions an entity can perform, on what resources and under what conditions | JSON policy documents attached to a resource i.e., S3 Bucket |
| Support Inline policies |  |

If your request is allowed, the API action you are running will be executed against the resource. E.g., get bucket, create user, run instances.

New user accounts have no permissions by default.

Two names for an account:

**Friendly name**: i.e., Alex

**Amazon Resource Name (ARN):** arn:aws:iam::625148252389:user/Alex

The number portion is the account ID (unique identifier).

ARN’s are unique not just in your account but across accounts. Due to its unique identifier.

**IAM Role** – IAM identity that has specific permissions assigned to it. Roles are assumed by users, applications, and services. Entities such as users or other services can assume the role and act as if they are the role (use the roles policies to gain permissions etc.). Access when assuming a role is short term.

All permissions are implicitly denied by default.

IAM roles will have two policies:

**Trust Policy**: who is allowed to assume the role.

**Permissions Policy:** What can the user do with the resource/service

**Signing Certificate** – through IAM used for accessing certain services e.g., SOAP and CLI interfaces of Amazon EC2 (x.509 certificate authentication).

**SSH Key/ HTTPS Git Credentials** – through IAM access services such as AWS Code Commit

**Keyspace** Credentials – through IAM for access to Amazon Keyspaces

**Access control methods**:

Role-Based Access Control **RBAC**: the act of minimising the permissions assigned to users/groups so that they have the AWS permissions to complete their job role and no other. i.e., Dev group only having the permissions they need and Ops group only having permissions they need.

Attribute-Based Access Control **ABAC:** tag based. Permissions are granted to resources when the tag matches the defined value.

**Permissions Boundaries**

**Control** the **maximum permissions** that an identity-based policy can grant an IAM entity (user or role).

e.g., Regardless of IAM permissions in a policy assigned… if the permissions boundary does not have it explicitly allowed the permissions will still be restricted.

Permissions boundaries **DO NOT GRANT** permissions. Simply limit/controls permissions you can use.

**SCP** is similar to permissions boundary, but SCP is applied at the account level.

Policy Evaluation Logic:

If there is an deny at any stage, then permission will be DENIED.

Check if account has **SCP** > **Resource-based Policy** > **Permissions Boundary** > **Session Policy** > **Identity-based policy**

**Session Policy** – temporary credentials using AWS STS to access AWS resources. Allows users to assume a role and perform actions for a temporary period. A session policy is an inline policy.

Resource-based policies will always have a **“Principal”** element defined.

**Instance Profile**

A container for an IAM Role to be used with EC2 instances.

Attach a role to an instance via Instance Profile.

How your EC2 instance will get permissions to access certain services (i.e., S3)

Created for you if using the AWS Console. When selecting an IAM Role when launching an instance, AWS automatically creates the Instance Profile. If doing programmatically (i.e., via CloudFormation) you must create the Instance Profile.

**IAM Best Practices:**

* Lock away your AWS account root user access keys
* Do not share access keys
* Rotate credentials
* Remove any unnecessary credentials
* Configure a strong password policy for users
* Use MFA
* Create individual IAM users
* Use groups to assign permissions to IAM users
* Use policy conditions for extra security (i.e., add a source IP condition)
* Grant least privilege
* Get started using permissions with AWS managed policies
* Use customer managed policies instead of inline policies
* Use access levels to review IAM permissions
* Use roles to delegate permissions
* Use roles for applications that run on EC2 instances, instead of imbedding any authentication credentials or access keys in your code
* Monitor activity in your AWS account

**AWS Directory Services and Federation**

Services under AWS directory service products:

**AWS Managed Microsoft AD –** managed implementations of Microsoft Active Directory running on Windows Server 2012, can connect onsite AD to AWS managed AD via VPN. Two-way trust relationship – identities in both directory services can authenticate and access each other’s resources. You can also connect to Azure AD and Office365 using **ADSync** and **ADFS** which allows you to synchronize users and federate identities with Azure/O365. Supports group policy, SSO and MFA with Radius.

**AD Connector –** connect onsite AD to AD Connector via VPN or Direct Connect with your AD credentials and you can authenticate to services in AWS i.e., Amazon Workspaces, WorkDocks and WorkMail. Can also access the AWS Management Console. Map AD identities to IAM Roles. You can also join Windows EC2 instances to an on prem AD domain.

**Identity Federation** – a system of trust when a system has multiple identity sources so that authentication and authorisation to resources is enabled.

**Identity provider (IdP)** – sources of the identity information e.g., user accounts are stored in an identity provider

**Service provider** – application or service being offered. Authenticate and authorise access for users to the service providers application or service.

**Identity Federation Services:**

Different methods for accessing AWS accounts, resources, and cloud applications.

* **AWS SSO** – most likely use case, permissions assigned based on group membership, works with many Idps such as Active Directory, you can attach multiple AWS accounts.
* **AWS Identity and Access Management** – legacy, IdP is configured in AWS IAM
* **Amazon Cognito** – federation support for **mobile** and **web** **apps**, **sign in and sign up**, supports **social networks** apple, Facebook,3 google, amazon etc.

**With SSO:** Connect to **Azure AD**, on **prem AD** via AWS Directory Service, **business applications** (Github, Slack, G Suite etc.) and **AWS Organisations** and accounts in organisations.

**Amazon Cognito:**

**User Pool** – directory for managing sign-in and sign-up for mobile apps. Contains the identities themselves. Alternative to signing in through social IdP’s.

**Identity Pool** – temporary, limited privilege credentials for AWS Services (talks to sts). Identities could come from Cognito user pool or social identity providers. Way you get temporary credentials to AWS services.

**Advanced VPC**

Defining VPC CIDR Blocks

For AWS CIDR block can be between **/16 and /28**

Cannot overlap with any existing CIDR block

Cannot increase or decrease the size of an existing CIDR block

First 4 IP and last IP address reserved

**VLSM** – use a portion of an octet for network portion and some for the host portion.

AWS uses RFC 1918 ranges:

10.0.0.0 – 10.255.255.255

172.16.0.0 – 172.31.255.255

192.168.0.0 – 192.168.255.255

**Region > VPC > Subnet & Availability Zone**

**Internet gateway** (router) assigned to public subnets.

**Main Route Table** – automatically created with VPC. Implicitly associated to all subnets that are not explicitly associated with a route table.

Each subnet can only be associated with one route table at a time.

**Private Route Table** – a route table that you manually create to assign to subnets explicitly.

Static routes > propagated routes in a route table.

**VPC SECURITY:**

**NACL** – Subnet level, rules evaluated in order low to high, allow or explicitly deny traffic.

**Security group** – instance level, allow rules only – all traffic is denied by default. Can have more than one SG per EC2 instance

**Nat Gateway** – allows an instance in a private subnet to connect to the internet. Has a private and public IP address. Created in a public subnet – associated to the main route table.

Use a private route table to point instances in your private subnet to the NAT gateway and the NAT gateway will forward it out to the internet/internet gateway. New way to do things. Manged by AWS.

**Nat Instance** – is an instance (with a special AMI) that can be used to allow an instance in a private subnet to connect to the internet via the NAT instance ID. Add the Nat Instance ID to the private subnets route table. You have to manage the OS for your instance. Old way of doing things.

**VPC peering:** Connect two or more VPC’s together. Doesn’t support transitive peering. CIDR blocks cannot overlap. VPC A creates peering request. VPC B accepts request. Update route tables.

**Transit Gateway:** acts as a hub to connect VPC’s and on prem networks. Can attach to TGW’S in other accounts, VPN’s. Direct Connect gateways and 3rd party appliances.

**Transit Peering**: Like VPC peering but you connect all VPC’s to one Transit gateway and do not have to worry about creating numerous VPC peering connections.

Can connect VPC’s to on prem networks. All traffic is encrypted. Build applications spanning multiple VPC’s. Deploy applications without updating large route tables.

Better for a large number of VPC’s, UP TO 5,000.

**VPC Endpoints**: Keep communications within the AWS network.

**Gateway Endpoint:** VPC level – Used as a gateway for EC2 instances to connect to AWS services without being exposed to the internet. Only S3, DynamoDB.

**Interface Endpoint:** Used as a gateway for EC2 instances to connect to AWS services without being exposed to the internet. All other services.

**AWS Client VPN** – connect your client PC to an AWS VPC via VPN connection, Must use external VPN provider.

**AWS Site-to-site VPN** – allows an on prem location to AWS and have a private network established. Managed IPSec VPN.

Uses a **Virtual Private Gateway (VGW)** deployed on the AWS side and a **customer gateway** on the on prem side.

**Direct Connect** – Physically Connect to AWS using a private network link. Better for bandwidth and latency. Very expensive.

Connect on prem site physically to local AWS Direct Connect Location (via customer cage). Customer cage is connected to AWS cage and AWS cage is connected to AWS via physical fibre connection – 1GB/s – 10GB/s.

**Virtual NIC’s:**

ENI – basic adapter to assign IP addresses to an instance. Use when there are no high-performance requirements. Use with all instance types.

ENA – For enhanced network performance, higher bandwidth and latency. Only supported instance types.

EFA (Elastic Fabric Adapter) – for high performance computing (AI & ML). All instance types.

ALB – supports instances, IP addresses, containers, and Lambda targets. Uses HTTP protocols

NLB – Routes based on IP, high performance and low latency. TCP and UDP protocols

CLB – older load balancer.

Gateway Load Balancer – used for virtual network appliances

**CloudFront** – data in a CloudFront origin (S3 or EC2) will be pushed (cached) out to an Edge Location. Edge locations are distributed all over the world.

Allows for better latency and higher bandwidth from you service to the end user. Good for web distributions.

**Lambda@Edge**

Run Node.js and Python Lambda functions to customise the content CloudFront delivers. Basically, run Lambda functions closer to your viewers.

**Lambda**

Serverless functions. Upload code without managing servers. Good for short running tasks.

Developer uploads code to Lambda to execute, a trigger such as an event from the Console, CLI, API or SDK causes the Lambda function to execute. Pay for execution time (15 mins max).

Example, uploading an object to an S3 bucket can trigger a Lambda function that grabs meta-data for the S3 object. You can use a blueprint function, or write one from scratch.

7 LANGUAGES: Python, Ruby, Java, Powershell, Go, Node.js , C#

**SQS – Simple Queue Service** (queuing system for messages sent between microservices and serverless applications)

Standard Queue – best effort ordering for messages, message is sent at least once, sometimes more than once

FIFO Queue – First in first out, messages come once, no duplicates

**Amazon EventBridge (CloudWatch events)**

When an AWS service performs an event, an EventBridge rule is triggered (based on configured rules) and the information is forwarded on to complete an action

E.g., EC2 termination event > EventBridge rule > target is SNS to send a notification

**API Gateway (serverless)**

Deploy and manage API’s with AWS. API’s act as a ‘front door’ for applications to access data and functionality from backend services. RESTful API’s are used with HTTP calls.

Cache calls to improve latency.

**ECS**

Run docker containers and microservices on either serverless (Fargate) or on EC2 instances. Scale and maintain the microservices independently.

ECS runs in tasks – which are running docker containers. Tasks are defined in a task definition.

Has its own container registry for docker images: **Amazon Elastic Container Registry** ECR

**EKS**

Implementation of Kubernetes on AWS. Can run on EC2 or Fargarte.

**AWS Elastic Beanstalk:** Deploy web servers

Upload your code and Beanstalk will handle the deployment, load balancing and autoscaling.

You retain full control over the resources powering your application (NOT SERVERLESS).

**Migration Services:**

**AWS Server Migration Service –** Migrate on prem servers to EC2 instances

**AWS Database Migration Service –** Migrate on prem database to RDS

**AWS DataSync –** Migrate on prem file servers to EFS

**AWS Migration Hub –** one unified console to observe migrations and the above migration tools.

**AWS KMS**

Create and manage encryption keys.

Customer Master Keys are key that can be used by the user to encrypt data in AWS. (Created in AWS or imported into AWS)

AWS Managed CMK are used by AWS to encrypt data. AWS manage the keys.

**AWS Config**

Evaluated configuration of your AWS account and resources against desired configurations.

**AWS WAF**

Web application firewall, create rules to filter web traffic based on IP addresses. Can attach to a load balancer which is attached to EC2 instances, or API gateway etc…