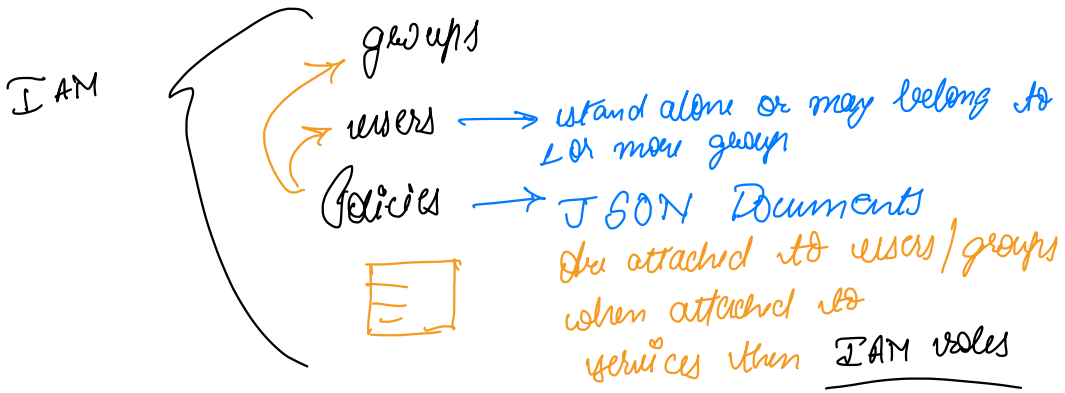
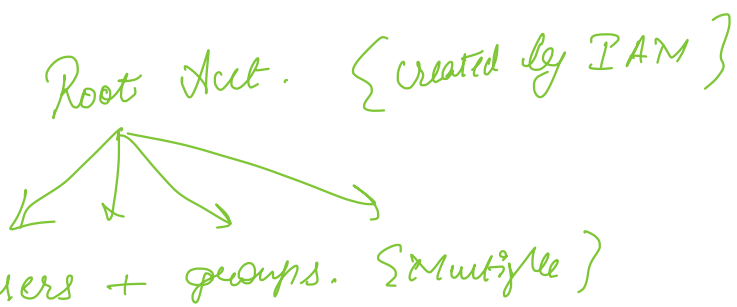
§ IAM { Identity & Access Management }

* Global service



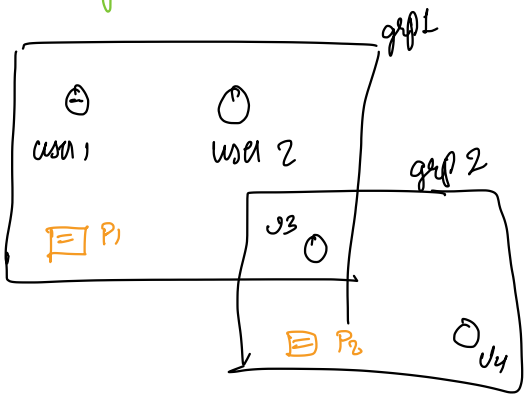
* Least privelleges principle for IAM

* Create users to refrain from using the root account.



* Security of account is user responsibility.

IAM policies are inherited.



user	policy
U1	P1
U2	P1
U3	P1 + P2
U4	P2
U5	P3

Policy structure!

{
- "version"
- "id"
- "Statement": [→ List of document
  {
  - "Sid":
  - "Effect": → Allow / Deny
  - "principal": { → whom does the policy apply to
  }
  - "Action": [→ List of action policy allows
  ]
  - "Resources": [→ List of resources it is applied on
  ]
  }
]

{Condition : when does the policy effects}

IAM roles are policies attached to a service so that they can communicate with other services.

aws configure → to sign in with a user.

* 3 different ways to access AWS

- AWS management console { username + password } + MFA
- AWS CLI { access key protected }
- AWS SDK { access key protected used when we need to integrate AWS into a dev-environment and access using API calls.

* Access Keys { username } secret access key { password } } secrets!

AWS CLI → open source { cmd starts with aws }

* AWS CLI is built on AWS SDK for python.

* AWS CLI → PC
  AWS cloudshell → browser per shell env.

IAM security tools

1. IAM Credential Report (account level)
   └> List all users and the status of various credentials

(2) IAM Access advisor (user level)

↳ • Service permission granted to user
• could be used revise policies for user based on his usage.
• which service accessed by user & when

IAM access analyzer → Provide security tools to analyze the usage & access of each user.

AWS Budget setup

* Even though u have administrator access does not necessarily mean that u can view the billing, for that

go to root acct. > Billing & Cost Management > accounts (upar se)

* go to Bill
  - select month & year
  - { charges by service, charges by acct. Savings. } } breakdown of cost on AWS

IAM user role access to billing info (deactivated by dy.)

* go to budgets
  - create budgets
    { existing templates { zero spend budget, monthly cost budget }

IAM best practices

1. don't use root account, instead make users
2. 1 physical user → 1 AWS user
3. users to grp, permissions to grp
4. create strong passcod policy
5. MFA enforce
6. Roles for AWS services
7. Never ever share access Keys & credentials.

IAM identity center → enables to manage workforce user across to multiple AWS accounts. { free service }

but

IAM → allows to create user native to just 1 AWS account

AWS organization

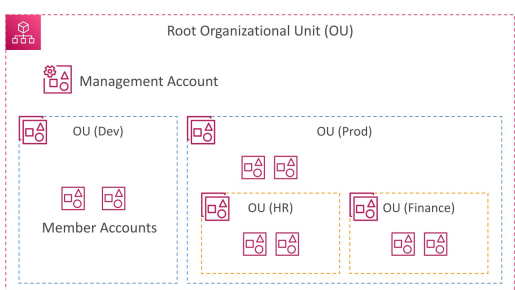
→ Global service

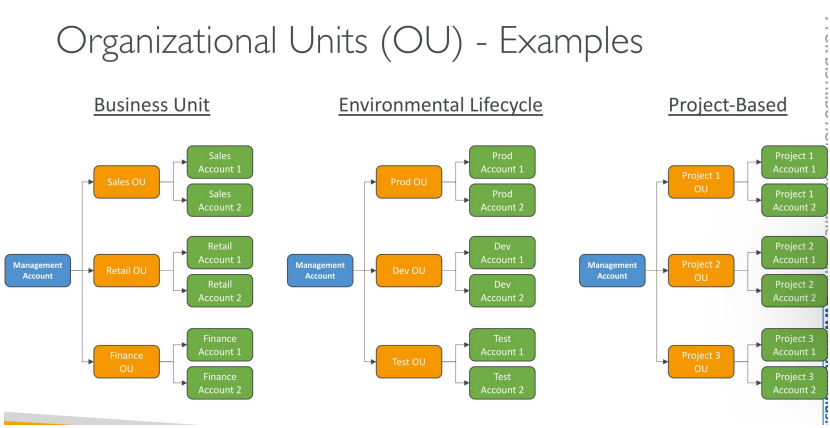
→ Allows to manage multiple AWS account

main acct. (management acct.)

other acct. (member acct.) → can be part of a single organization.

* "consolidated billing"
* pricing benefits from aggregated usage (volume discount) on reserved instances which shared across the entire organization & saving plans
* Account creation API (easy to create accounts)

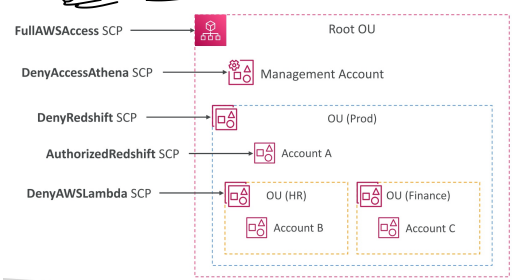




- Advantages
  - Multi Account vs One Account Multi VPC } better security because accounts are more separated than VPC ?
  - Use tagging standards for billing purposes
  - Enable CloudTrail on all accounts, send logs to central S3 account
  - Send CloudWatch Logs to central logging account
  - Establish Cross Account Roles for Admin purposes
- Security: Service Control Policies (SCP)
  - IAM policies applied to OU or Accounts to restrict Users and Roles
  - They do not apply to the management account (full admin power)
  - Must have an explicit allow (does not allow anything by default – like IAM)

can be applied to all accts. except management acct.

SCP Hierarchy



- Management Account
  - Can do anything
  - (no SCP apply)
- Account A
  - Can do anything
  - EXCEPT access Redshift (explicit Deny from OU)
- Account B
  - Can do anything
  - EXCEPT access Redshift (explicit Deny from Prod OU)
  - EXCEPT access Lambda (explicit Deny from HR OU)
- Account C
  - Can do anything
  - EXCEPT access Redshift (explicit Deny from Prod OU)

* SCP at OU level have higher precedence than user level

SCP Examples
Blocklist and Allowlist strategies

allow to all

deny to some

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowsAllActions",
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        },
        {
            "Sid": "DenyDynamoDB",
            "Effect": "Deny",
            "Action": "dynamodb:*",
            "Resource": "*"
        }
    ]
}
```

Blocklist

allow to some

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:*",
                "cloudwatch:*"
            ],
            "Resource": "*"
        }
    ]
}
```

Allowlist

IAM Conditions

restrict who can only company network

aws:SourceIp
restrict the client IP **from** which the API calls are being made

deny everyone everything except from the given ip CIDRs

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": ["192.0.2.0/24", "203.0.113.0/24
                }
            }
        }
    ]
}
```

policy applies when the src ip address is not within the specified IP ranges

aws:RequestedRegion
restrict the region the API calls are made **to**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": ["ec2:*", "rds:*", "dynamodb:*"],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": ["eu-central-1", "eu-west-1"]
        }
      }
    }
  ]
}
```

deny anything on ec2, rds, dynamodb to aws any resource if we are in the regions given

ec2:ResourceTag
restrict based on **tags**

applies to tag on ec2 instance

allow to start & stop instances for the given instances if the ec2 instance resource tag project is "data analytics" and also user must be part of department

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["ec2:startInstances", "ec2:StopInstances"],
      "Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*",
      "Condition": {
        "StringEquals": {
          "ec2:ResourceTag/Project": "DataAnalytics",
          "aws:PrincipalTag/Department": "Data"
        }
      }
    }
  ]
}
```

user tag.

aws:MultiFactorAuthPresent
to force **MFA**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": ["ec2:StopInstances", "ec2:TerminateInstances"],
      "Resource": "*",
      "Condition": {
        "BoolIfExists": {
          "aws:MultiFactorAuthPresent": false
        }
      }
    }
  ]
}
```

allow all action on ec2 instance for all instances

deny start & stop to all instances. if multi factor authen. is not present.

IAM for S3

- s3:ListBucket permission applies to arn:aws:s3:::test
- => bucket level permission

- s3:GetObject, s3:PutObject, s3:DeleteObject applies to arn:awn:s3:::test/*
- => object level permission

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:ListBucket"],
      "Resource": "arn:aws:s3:::test"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject"
      ],
      "Resource": "arn:aws:s3:::test/*"
    }
  ]
}
```

Bucket level

within bucket { all obj in bkt }

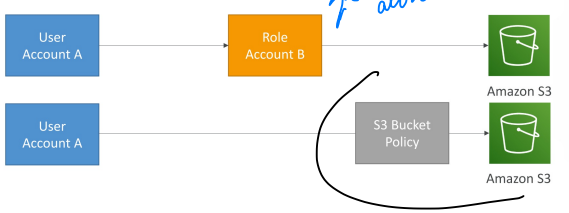
Resource Policies & aws:PrincipalOrgID

- aws:PrincipalOrgID can be used in any resource policies to restrict access to accounts that are member of an AWS Organization


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:PutObject", "s3:GetObject"],
      "Resource": "arn:aws:s3:::2022-financial-data/*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": ["o-yyyyyyyyyy"]
        }
      }
    }
  ]
}
```

IAM Roles vs Resource Based Policies

- Cross account:
  - attaching a resource-based policy to a resource (example: S3 bucket policy)
  - OR using a role as a proxy

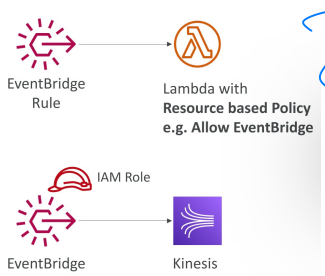


IAM Roles vs Resource-Based Policies

- When you assume a role (user, application or service), you give up your original permissions and take the permissions assigned to the role
- When using a resource-based policy, the principal doesn't have to give up his permissions
- Example: User in account A needs to scan a DynamoDB table in Account A and dump it in an S3 bucket in Account B.
- Supported by: Amazon S3 buckets, SNS topics, SQS queues, etc…

Amazon EventBridge – Security

- When a rule runs, it needs permissions on the target
- Resource-based policy: Lambda, SNS, SQS, S3 buckets, API Gateway…
- IAM role: Kinesis stream, Systems Manager Run Command, ECS task…



Resource based policy : (1) cross account access to resources
(2) granular access control {detailed permissions}

IAM Role : here temporary credentials is used for communication
hence the client knows its permission & assumes
what is specified by the role.
(1) service to service communication
(2) delegate permissions.

best security approach → a resource based policy might specify
which accounts or services can access an
event bus, {who can access}
IAM role may define "what action to perform" by that
service or user.

IAM Permission Boundaries

- IAM Permission Boundaries are supported for users and roles (not groups)
- Advanced feature to use a managed policy to set the maximum permissions an IAM entity can get.

Example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "cloudwatch:*",
                "ec2:*"
            ],
            "Resource": "*"
        }
    ]
}
```

IAM Permission Boundary

+

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:CreateUser",
    "Resource": "*"
  }
}
```

IAM Permissions Through IAM Policy

=

No Permissions

* policies will work only if they are within the permission boundary.

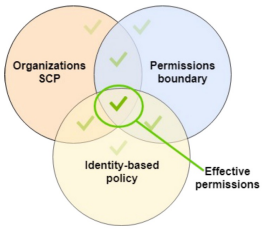
Permission boundaries >> Policies.

IAM user permission for only works on S3, CW, EC2 boundary

IAM policy (does not affect bc not within the permission boundary)

IAM Permission Boundaries

- Can be used in combinations of AWS Organizations SCP

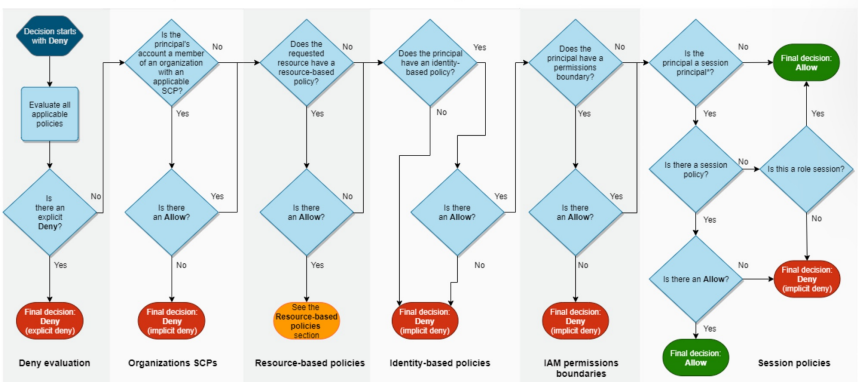


https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_boundaries.html

Use cases

- Delegate responsibilities to non administrators within their permission boundaries, for example create new IAM users
- Allow developers to self-assign policies and manage their own permissions, while making sure they can't "escalate" their privileges (= make themselves admin)
- Useful to restrict one specific user (instead of a whole account using Organizations & SCP)

IAM Policy Evaluation Logic



*A session principal is either a role session or an IAM federated user session.

https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html

≈ as soon as there is an explicit deny the decision is denied.

≈ if there is no explicit allow or deny then by default it is deny.

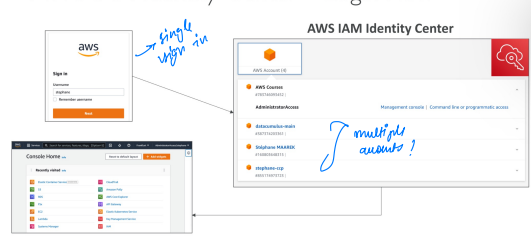
AWS IAM Identity Center (successor to AWS Single Sign-On)

- One login (single sign-on) for all your
  - AWS accounts in AWS Organizations
  - Business cloud applications (e.g., Salesforce, Box, Microsoft 365, …)
  - SAML2.0-enabled applications
  - EC2 Windows Instances

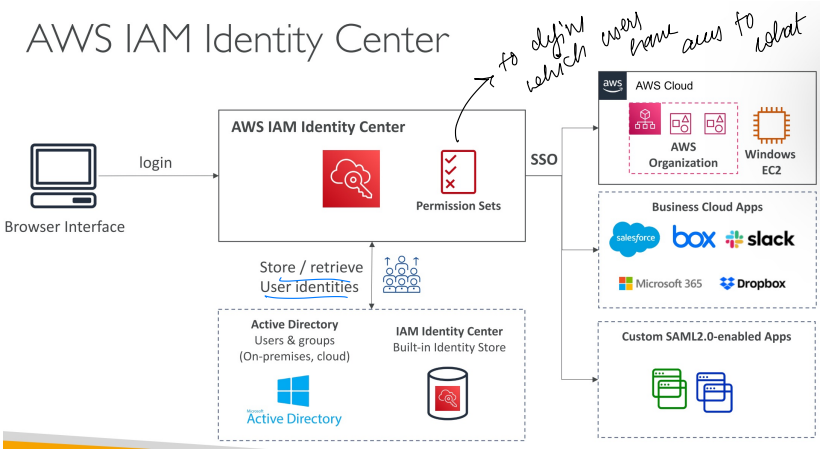
→ as long as this enabled, any app.

- Identity providers
  - Built-in identity store in IAM Identity Center
  - 3rd party: Active Directory (AD), OneLogin, Okta…

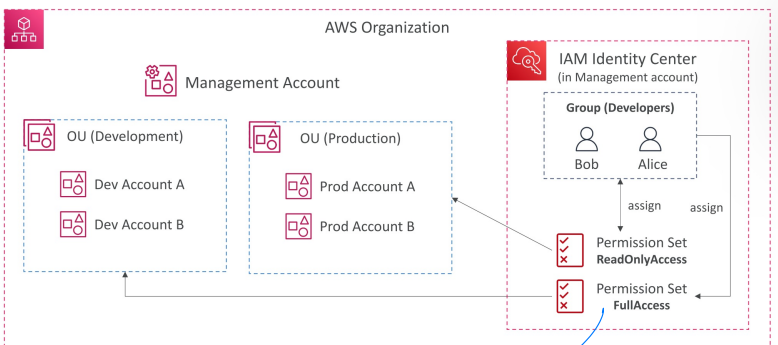
AWS IAM Identity Center – Login Flow



AWS IAM Identity Center

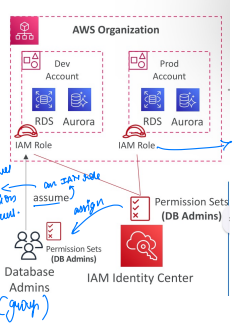


IAM Identity Center



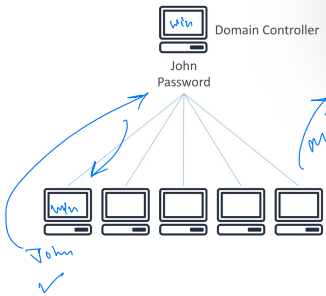
AWS IAM Identity Center
Fine-grained Permissions and Assignments

- **Multi-Account Permissions**
  - Manage access across AWS accounts in your AWS Organization
  - Permission Sets – a collection of one or more IAM Policies assigned to users and groups to define AWS access
- **Application Assignments**
  - SSO access to many SAML 2.0 business applications (Salesforce, Box, Microsoft 365, …)
  - Provide required URLs, certificates, and metadata
- Attribute-Based Access Control (ABAC)
  - Fine-grained permissions based on users' attributes stored in IAM Identity Center Identity Store
  - Example: cost center, title, locale, …
  - Use case: Define permissions once, then modify AWS access by changing the attributes



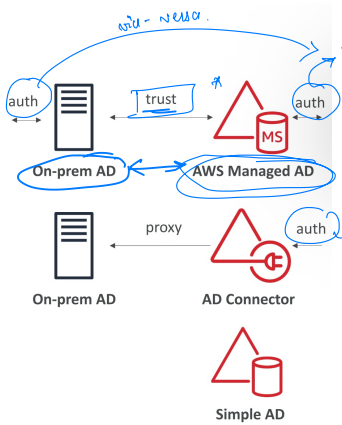
What is Microsoft Active Directory (AD)?

- Found on any Windows Server with AD Domain Services
- Database of **objects**: User Accounts, Computers, Printers, File Shares, Security Groups
- Centralized security management, create account, assign permissions
- Objects are organized in **trees**
- A group of trees is a **forest**



AWS Directory Services

- **AWS Managed Microsoft AD**
  - Create your own AD in AWS, manage users locally, supports MFA
  - Establish "trust" connections with your on-premises AD
- **AD Connector** { more latency }
  - Directory Gateway (proxy) to redirect to on-premises AD, supports MFA
  - Users are managed on the on-premises AD
- **Simple AD**
  - AD-compatible managed directory on AWS
  - Cannot be joined with on-premises AD

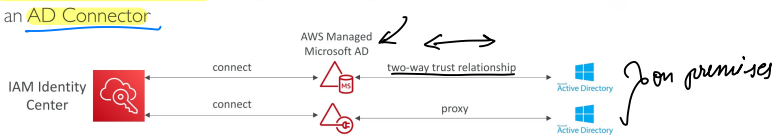


→ with active directory we can create ec2 instances that are going to be running windows & these windows instances can join the domain controller for the network & share all the logins & credentials. & so on.

IAM Identity Center – Active Directory Setup

- Connect to an AWS Managed Microsoft AD (Directory Service)
  - Integration is out of the box { ezz } single sign on

- Connect to a Self-Managed Directory
  1. Create Two-way Trust Relationship using AWS Managed Microsoft AD
  2. Create an AD Connector



AWS Control Towers

→ set up, govern a secure & compliant multi-account AWS environment with best practices leveraging AWS Organizations for account creation

→ automate the set up.

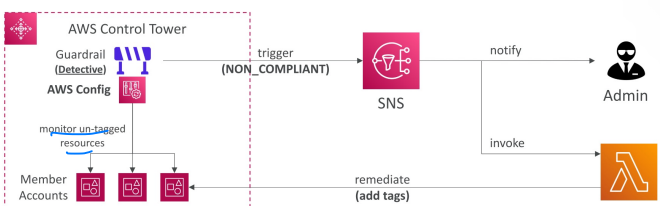
{ Detect policy violations
{ Monitor compliance
} with interactive dashboard.

AWS Control Tower – Guardrails

- Provides ongoing governance for your Control Tower environment (AWS Accounts)
- Preventive Guardrail – using SCPs (e.g., Restrict Regions across all your accounts)
- Detective Guardrail – using AWS Config (e.g., identify untagged resources)

prevent account from doing something { more restrictive }

detect non compliance (and remediation)



EX