iNurture Education Solutions
TOMORROW'S HERE

Web Applications and Security

Topics to be covered

- Amazon Identity and Access Management (IAM),
- Introduction to Elastic Beanstalk,
- Deploying Scalable Application On AWS, Selecting And Launching An Application Environment,
- Provisioning Application Resources with Cloud formation,
- Introduction to Cloud Watch,
- Describe Amazon Cloud Watch metrics and alarms,
- AWS Messaging Services(SNS,SQS,SES).
- Introduction to AWS Security,
- AWS Directory Service,
- AWS Key Management Service, Securing Data at Rest and In Motion.

AWS Identity and Access Management (IAM)

AWS Identity and Access Management (IAM)

- Use **IAM** to manage access to **AWS resources** –
  - A resource is an entity in an AWS account that you can work with
  - Example resources; An Amazon EC2 instance or an Amazon S3 bucket

- *Example* – Control who can terminate Amazon EC2 instances

- Define fine-grained access rights –
  - **Who** can access the resource
  - **Which** resources can be accessed and what can the user do to the resource
  - **How** resources can be accessed

- IAM is a no-cost AWS account feature

AWS Identity and Access Management (IAM)

IAM: Essential components

IAM user

A **person** *or* **application** that can authenticate with an AWS account.

IAM group

A **collection of IAM users** that are granted identical authorization.

IAM policy

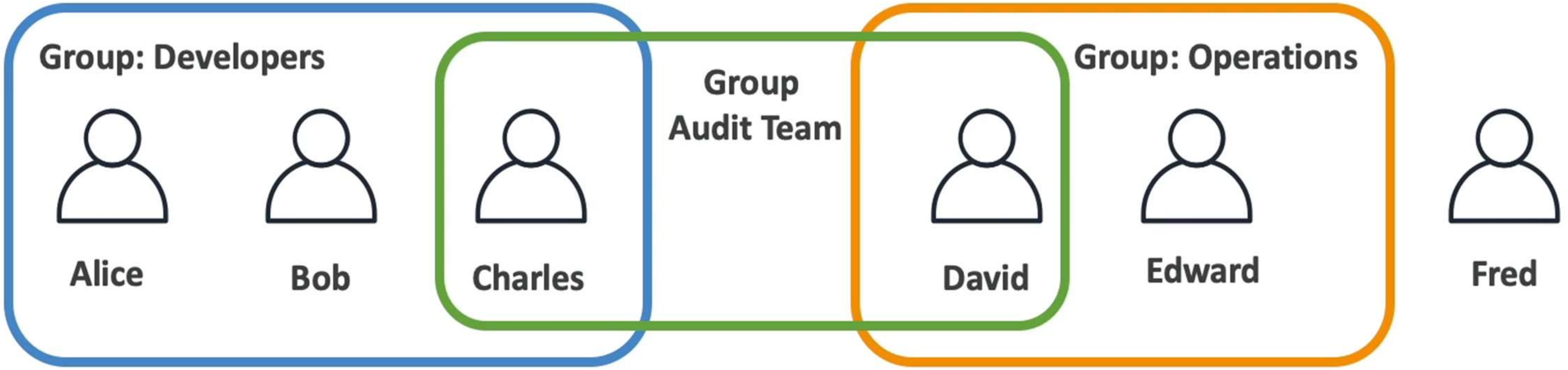
The document that defines **which resources can be accessed** and the **level of access** to each resource.

IAM role

Useful mechanism to grant a set of permissions for making AWS service requests.

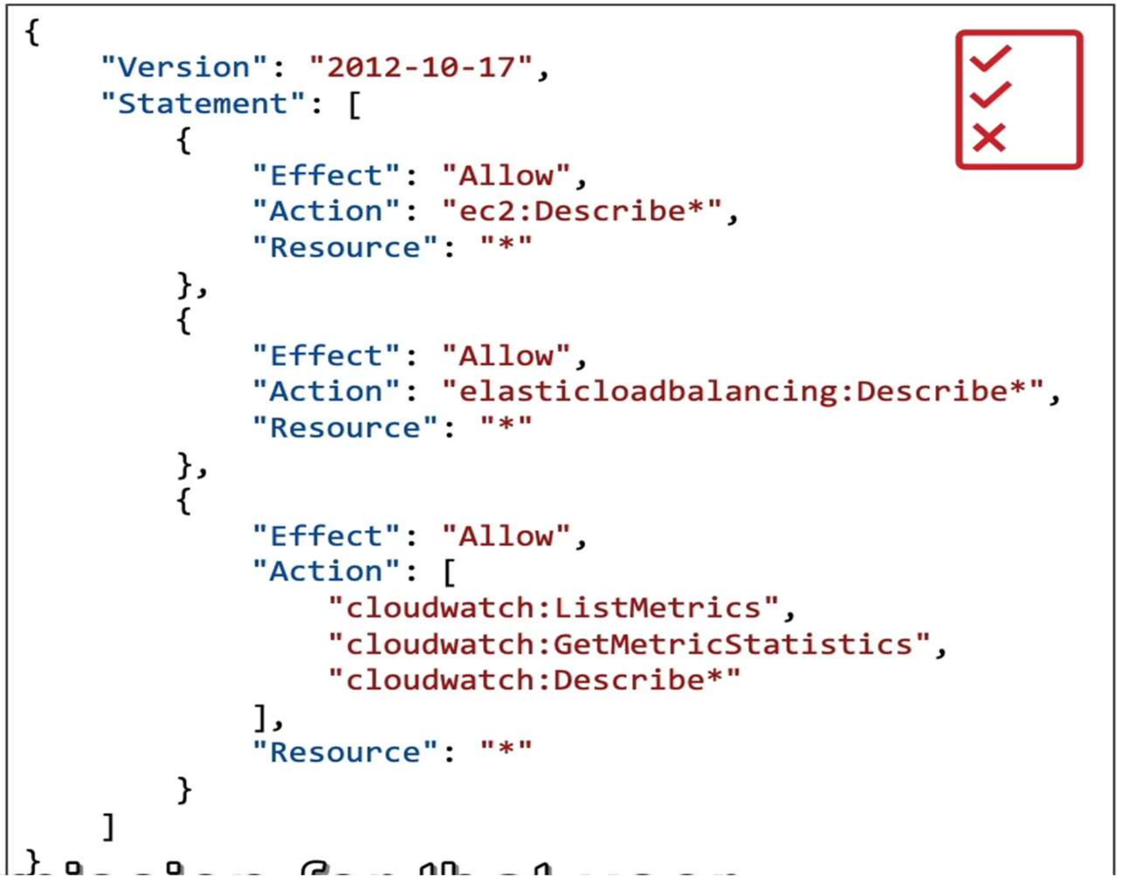
IAM: Users & Groups

- IAM = Identity and Access Management, **Global** service
- **Root account** created by default, shouldn't be used or shared
- **Users** are people within your organization, and can be grouped
- **Groups** only contain users, not other groups
- Users don't have to belong to a group, and user can belong to multiple groups



IAM: Permissions

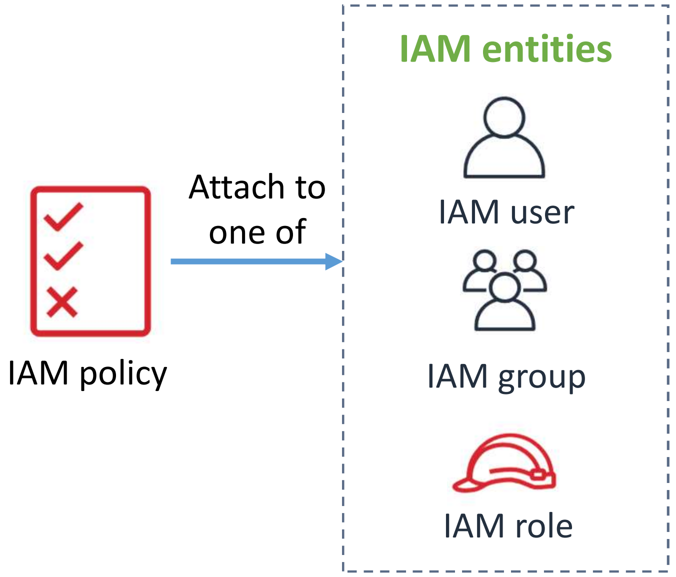
- **Users or Groups** can be assigned JSON documents called policies
- These policies define the **permissions** of the users
- In AWS you apply the **least privilege principle**: don't give more permissions than a user needs



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

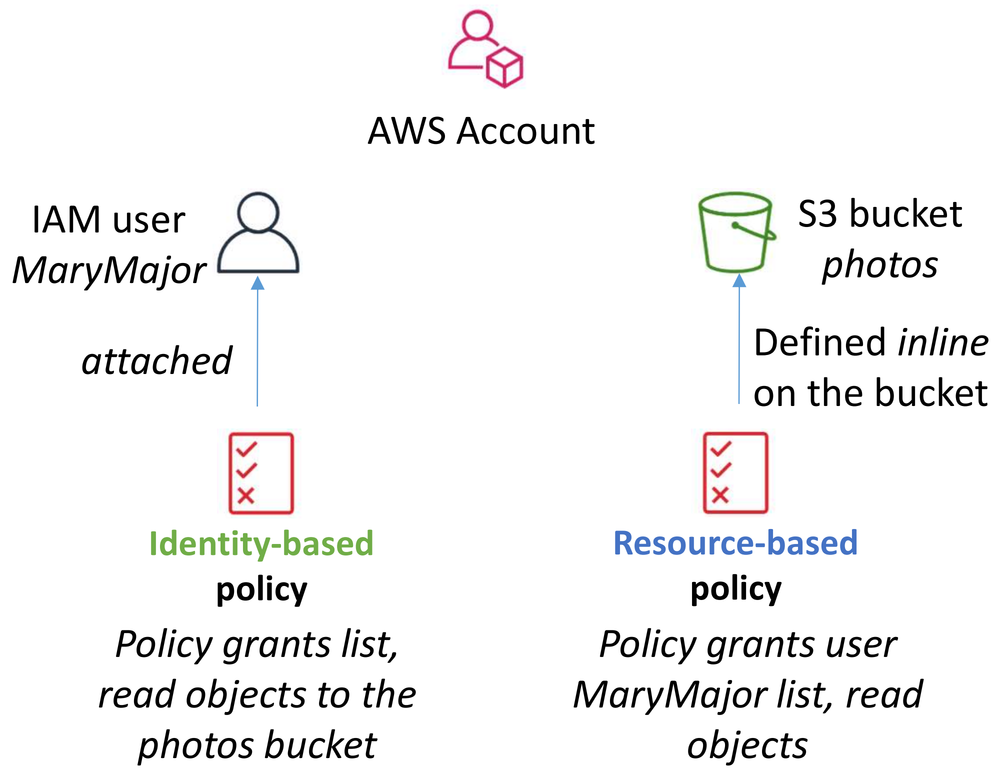
IAM policies

- **An IAM policy is a document that defines permissions**
  - Enables fine-grained access control
- Two types of policies – *identity-based* and *resource-based*
- **Identity-based** policies –
  - Attach a policy to any IAM entity
    - An IAM user, an IAM group, or an IAM role
  - Policies specify:
    - Actions that ***may*** be performed by the entity
    - Actions that ***may not*** be performed by the entity
  - A single *policy* can be attached to multiple *entities*
  - A single *entity* can have multiple *policies* attached to it
- **Resource-based** policies
  - Attached to a resource (such as an S3 bucket)



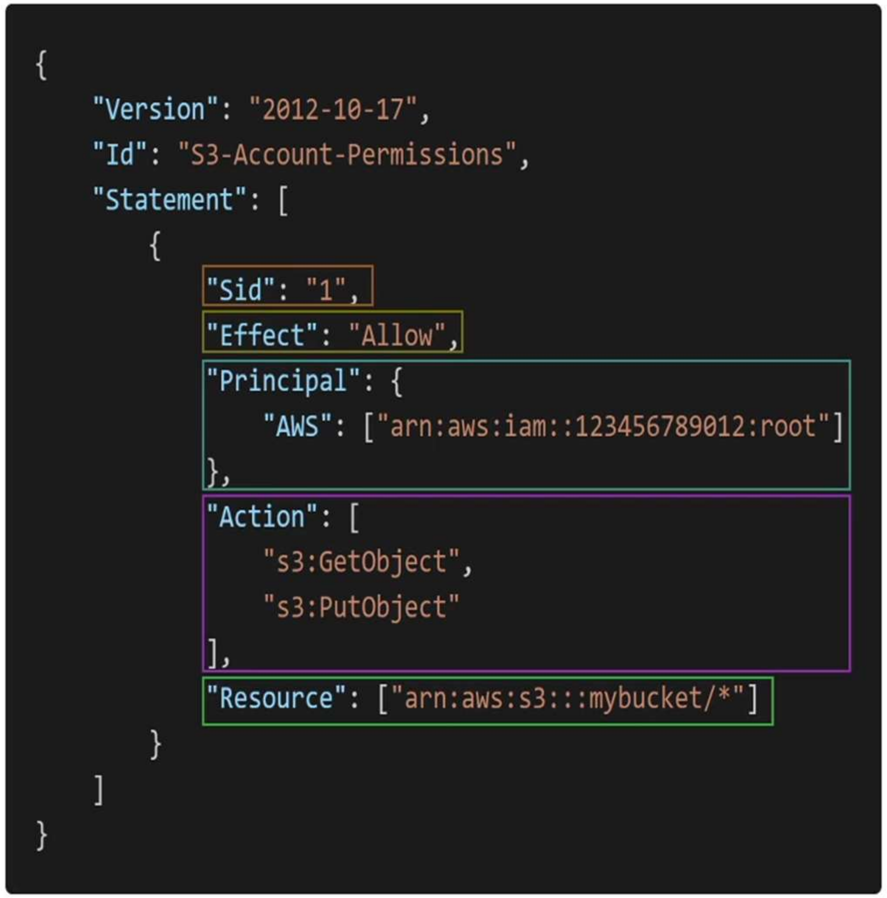
Resource-based policies

- *Identity-based policies* are attached to a user, group, or role
- **Resource-based policies** are attached to a resource (*not* to a user, group or role)
- Characteristics of resource-based policies –
  - Specifies who has access to the resource and what actions they can perform on it
  - The policies are *inline* only, not managed
- Resource-based policies are supported only by some AWS services


AWS Account
IAM user
MaryMajor
attached
S3 bucket
photos
Defined *inline* on the bucket
Identity-based policy
Policy grants list, read objects to the photos bucket
Resource-based policy
Policy grants user MaryMajor list, read objects

IAM Policies Structure

- Consists of
  - **Version:** policy language version, always include "2012-10-17"
  - **Id:** an identifier for the policy (optional)
  - **Statement:** one or more individual statements (required)
- Statements consists of
  - **Sid:** an identifier for the statement (optional)
  - **Effect:** whether the statement allows or denies access (Allow, Deny)
  - **Principal:** account/user/role to which this policy applied to
  - **Action:** list of actions this policy allows or denies
  - **Resource:** list of resources to which the actions applied to
  - **Condition:** conditions for when this policy is in effect (optional)



```
{
    "Version": "2012-10-17",
    "Id": "S3-Account-Permissions",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Allow",
            "Principal": {
                "AWS": ["arn:aws:iam::123456789012:root"]
            },
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": ["arn:aws:s3:::mybucket/*"]
        }
    ]
}
```

Authenticate as an IAM user to gain access

When you define an **IAM user**, you select what ***types of access*** the user is permitted to use.

***Programmatic* access**

- Authenticate using:
  - Access key ID
  - Secret access key
- Provides AWS CLI and AWS SDK access

AWS CLI

AWS Tools and SDKs

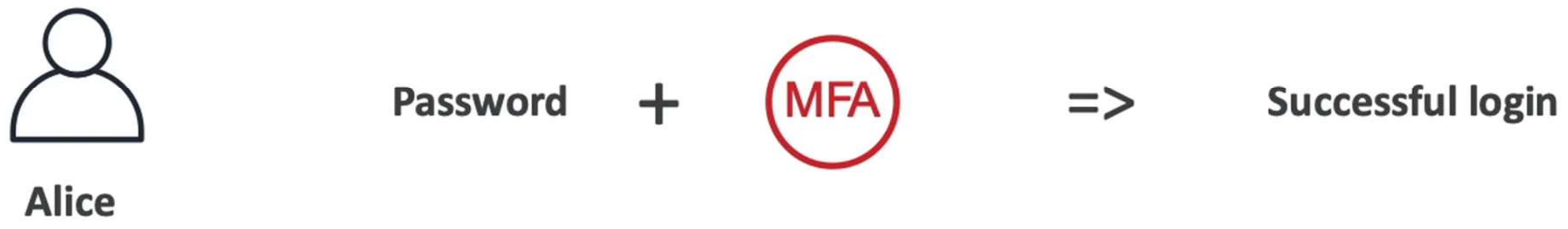
***AWS Management Console* access**

- Authenticate using:
  - 12-digit Account ID *or* alias
  - IAM user name
  - IAM password
- If enabled, **multi-factor authentication (MFA)** prompts for an authentication code.

AWS Management Console

Multi Factor Authentication - MFA

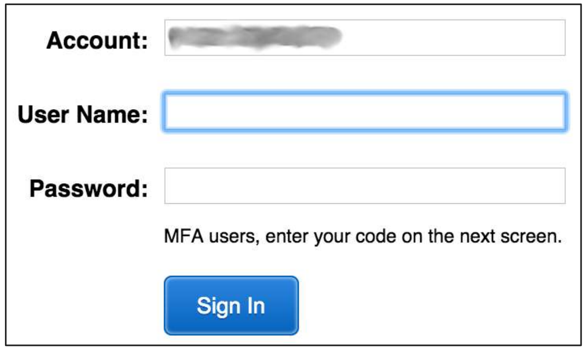
- Users have access to your account and can possibly change configurations or delete resources in your AWS account
- You want to protect your Root Accounts and IAM users
- MFA = password *you know* + security device *you own*



- Main benefit of MFA:
  <u>if a password is stolen or hacked, the account is not compromised</u>

IAM MFA

- MFA provides increased security.
- In addition to **username** and **password**, MFA requires a unique authentication code to access AWS services.

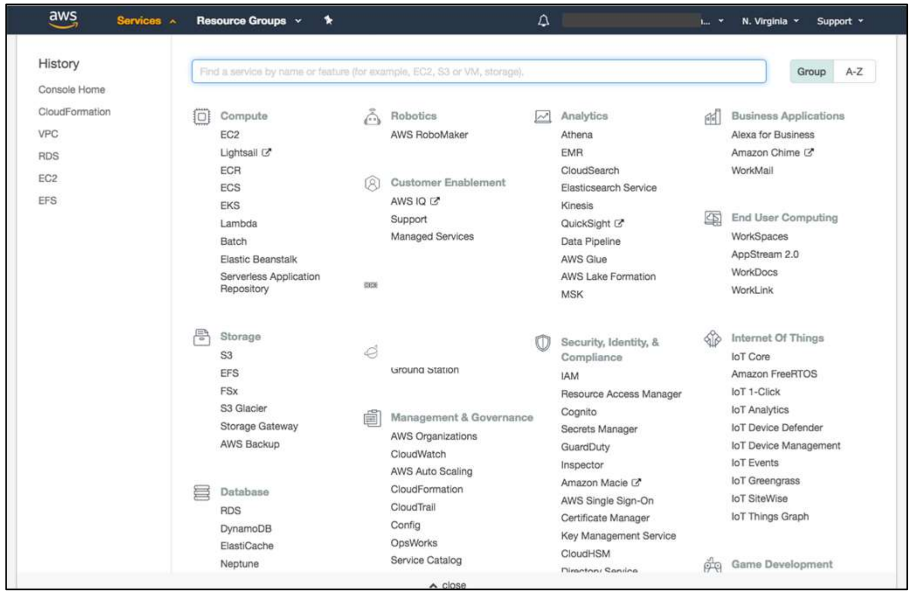


Account:	[redacted]
User Name:	
Password:	

MFA users, enter your code on the next screen.

Username and password

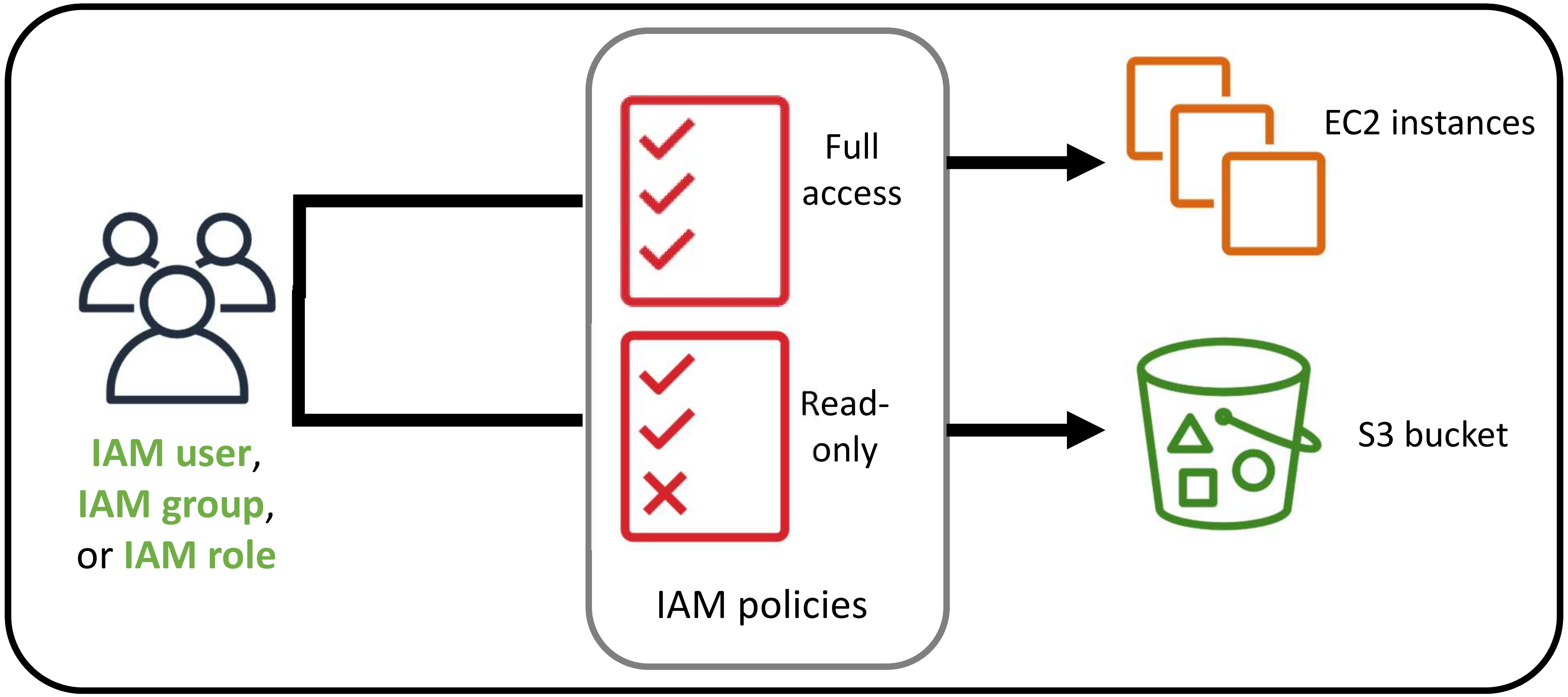
MFA token



AWS Management Console

Authorization: What actions are permitted

After the user or application is connected to the AWS account, what are they allowed to do?



IAM: Authorization

- Assign permissions by creating an IAM policy.
- Permissions determine **which resources and operations** are allowed:
  - All permissions are implicitly denied by default.
  - If something is explicitly denied, it is never allowed.

Best practice: Follow the **principle of least privilege**.

IAM permissions

Note: The scope of IAM service configurations is global. Settings apply across all AWS Regions.

IAM policy example

```
{
  "Version": "2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Action":["DynamoDB:*","s3:*"],
    "Resource":[
      "arn:aws:dynamodb:region:account-number-without-hyphens:table/table-name",
      "arn:aws:s3:::bucket-name",
      "arn:aws:s3:::bucket-name/*"]
    },
    {
    "Effect":"Deny",
    "Action":["dynamodb:*","s3:*"],
    "NotResource":["arn:aws:dynamodb:region:account-number-without-hyphens:table/table-name”,
      "arn:aws:s3:::bucket-name",
      "arn:aws:s3:::bucket-name/*"]
    }
  ]
}
```

Explicit allow gives users access to a specific DynamoDB table and…

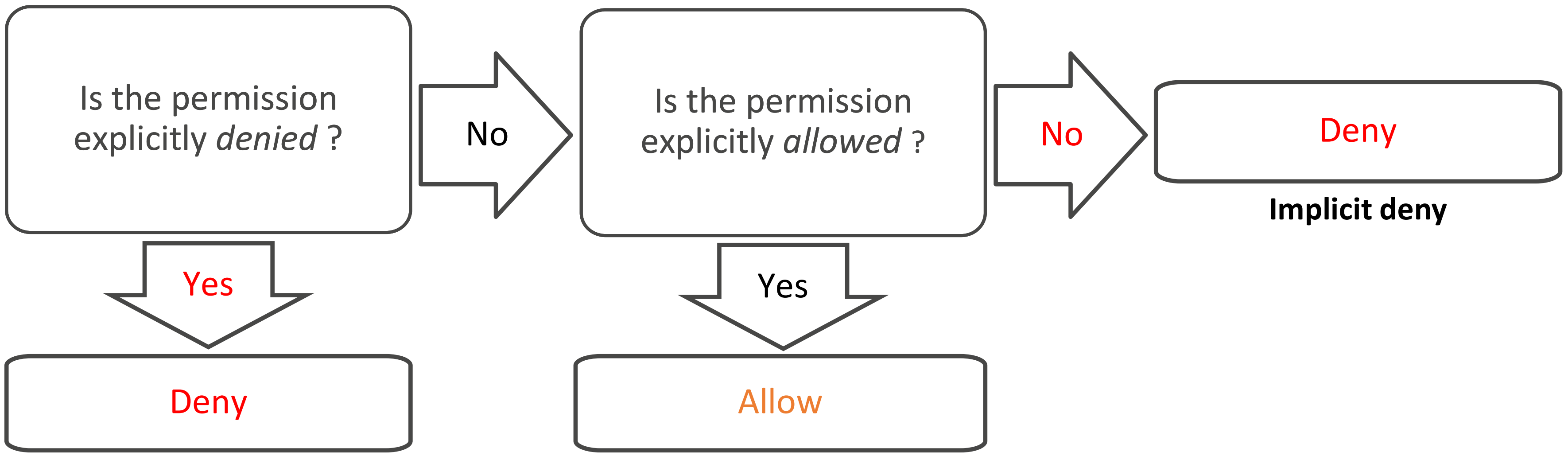
…Amazon S3 buckets.

Explicit deny ensures that the users cannot use any other AWS actions or resources other than that table and those buckets.

An explicit deny statement **takes precedence** over an allow statement.

IAM permissions

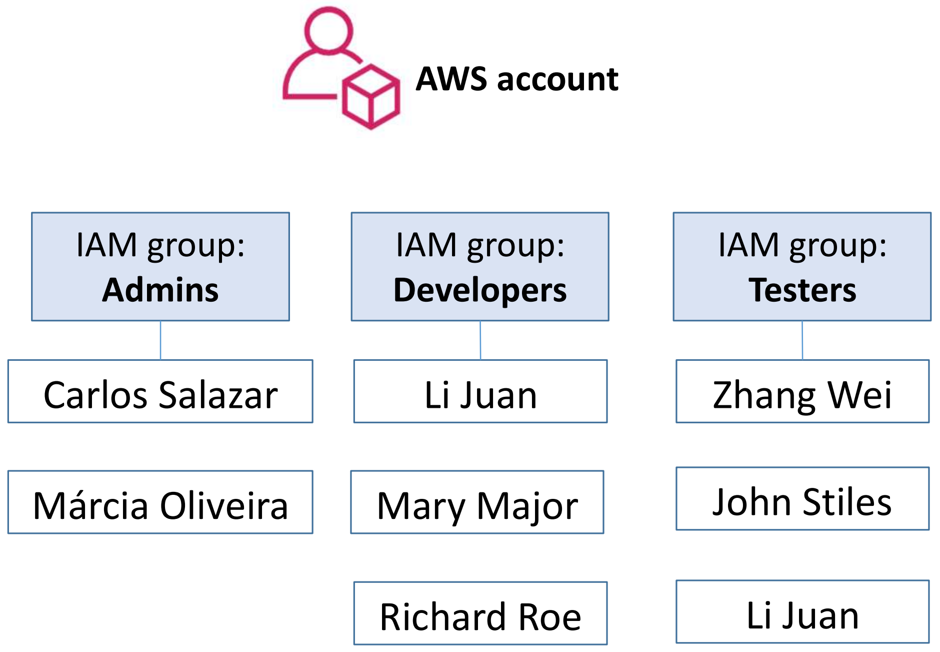
How IAM determines permissions:



Is the permission explicitly *denied* ?
No
Is the permission explicitly *allowed* ?
No
Deny
Implicit deny
Yes
Deny
Yes
Allow

IAM groups

- An **IAM group** is a collection of IAM users
- A group is used to grant the same permissions to multiple users
  - Permissions granted by attaching IAM *policy* or policies to the group
- A user can belong to multiple groups
- There is no default group
- Groups cannot be nested

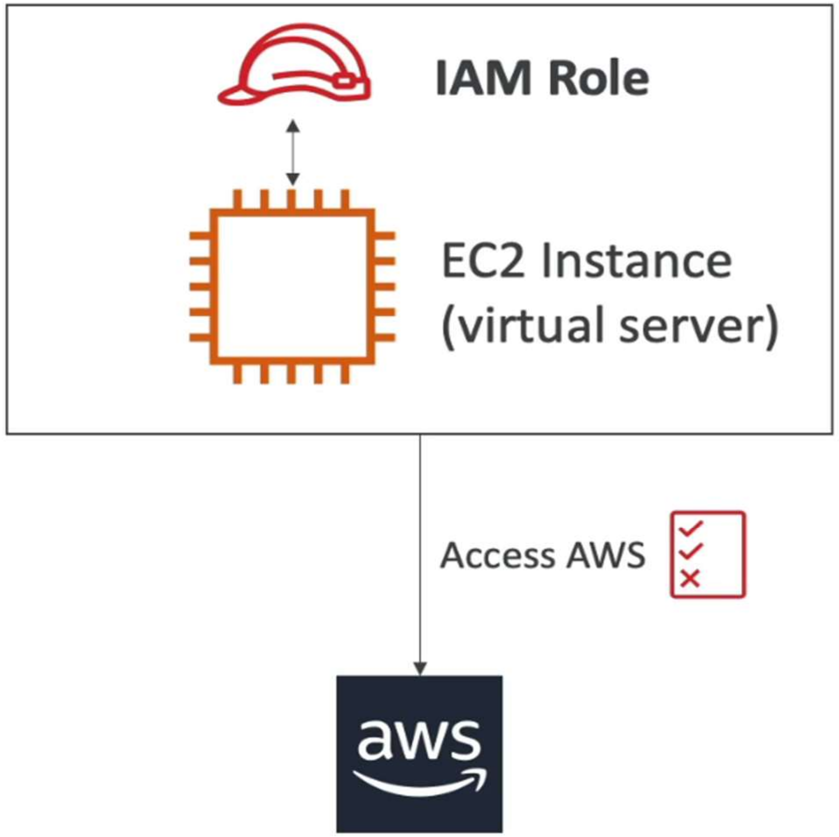


IAM roles

- An **IAM role** is an IAM identity with specific permissions
- Similar to an IAM user
  - Attach permissions policies to it
- Different from an IAM user
  - Not uniquely associated with one person
  - Intended to be *assumable* by a **person**, **application**, or **service**
- Role provides *temporary* security credentials
- Examples of how IAM roles are used to **delegate** access –
  - Used by an IAM user in the same AWS account as the role
  - Used by an AWS service—such as Amazon EC2—in the same account as the role
  - Used by an IAM user in a different AWS account than the role

IAM Roles for Services

- Some AWS service will need to perform actions on your behalf
- To do so, we will assign **permissions** to AWS services with **IAM Roles**
- Common roles:
  - EC2 Instance Roles
  - Lambda Function Roles
  - Roles for CloudFormation



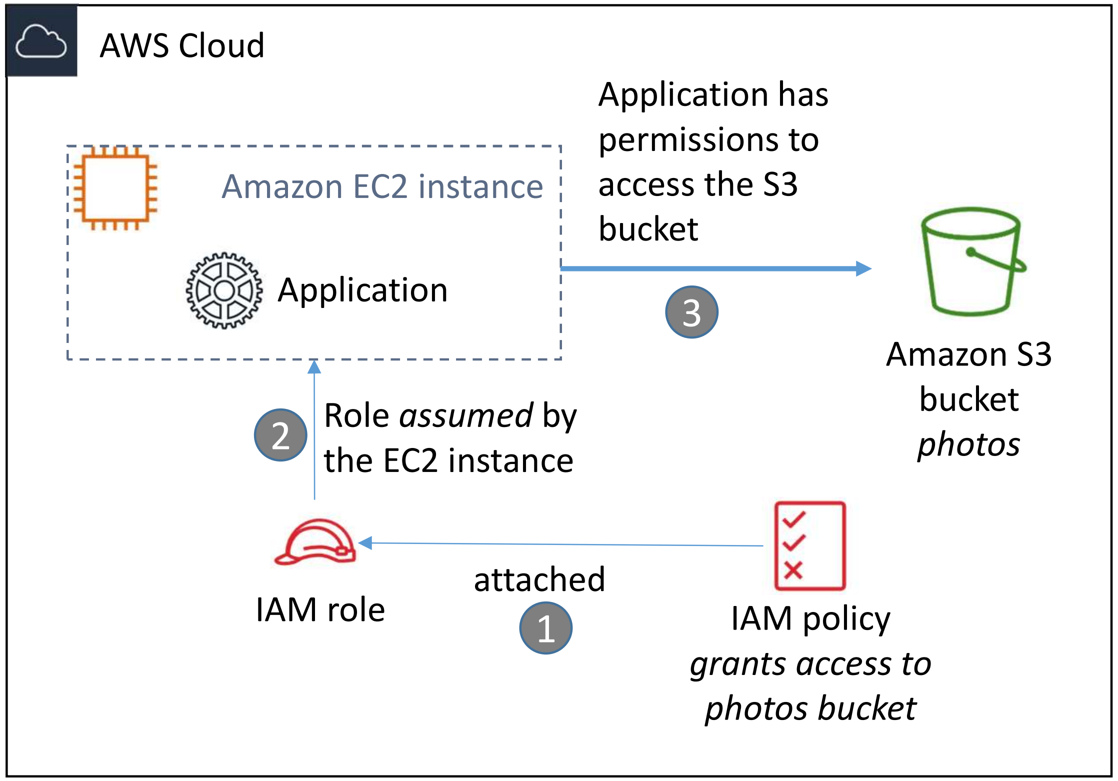
Example use of an IAM role

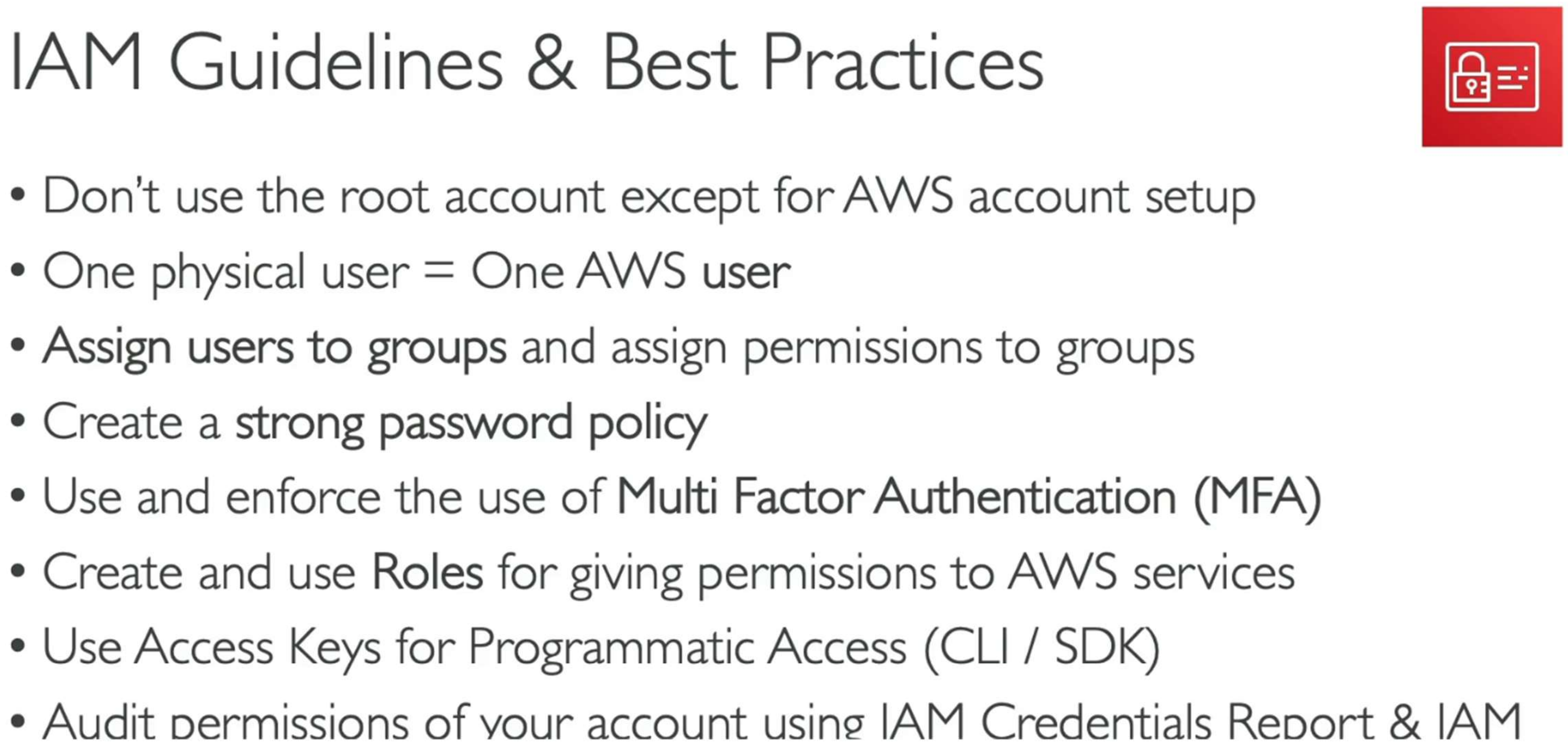
Scenario:

- An application that runs on an EC2 instance needs access to an S3 bucket

Solution:

- Define an IAM policy that grants access to the S3 bucket.
- Attach the policy to a role
- Allow the EC2 instance to assume the role


AWS Cloud
Amazon EC2 instance
Application
Application has permissions to access the S3 bucket
3
Amazon S3 bucket *photos*
2
Role *assumed* by the EC2 instance
IAM role
attached
1
IAM policy *grants access to photos bucket*

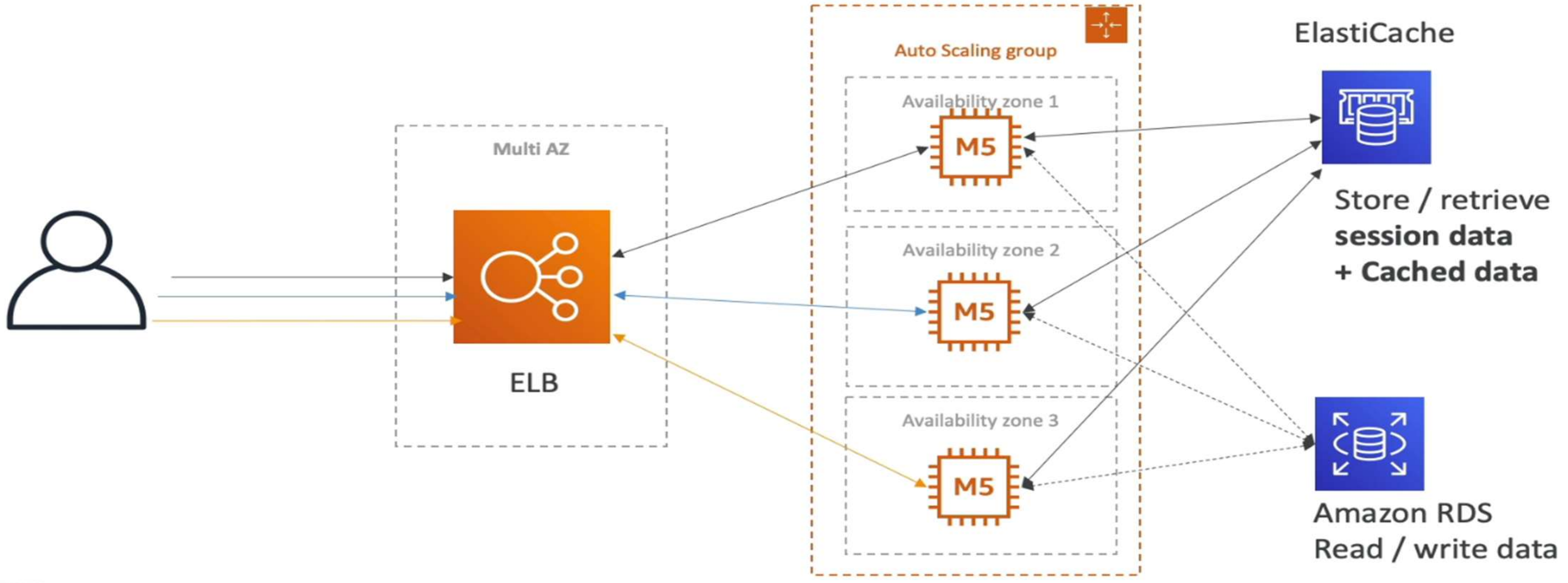


IAM Guidelines & Best Practices

- Don't use the root account except for AWS account setup
- One physical user = One AWS **user**
- **Assign users to groups** and assign permissions to groups
- Create a **strong password policy**
- Use and enforce the use of **Multi Factor Authentication (MFA)**
- Create and use **Roles** for giving permissions to AWS services
- Use Access Keys for Programmatic Access (CLI / SDK)
- Audit permissions of your account using IAM Credentials Report & IAM

Introduction to AWS Elastic Beanstalk

Typical architecture: Web App 3-tier



Developer problems on AWS

- Managing infrastructure
- Deploying Code
- Configuring all the databases, load balancers, etc
- Scaling concerns

- Most web apps have the same architecture (ALB + ASG)
- All the developers want is for their code to run!
- Possibly, consistently across different applications and environments

Introduction to AWS Elastic Beanstalk

- AWS Elastic Beanstalk is an AWS managed service for web applications.
- Elastic beanstalk is a pre-configured EC2 server that can directly take up your application code and environment configurations and use it to automatically provision and deploy the required resources within AWS to run the web application.
- Unlike EC-2 which is Infrastructure as a service, Elastic beanstalk is a Platform As A service (PAAS) as it allows users to directly us a pre-configured server for their application.
- Elastic beanstalk abstracts the underlying configuration work and allows you as a user to focus on more pressing matters.
- With AWS Elastic Beanstalk, you can quickly deploy and manage applications in the AWS Cloud without worrying about the infrastructure that runs those applications.
- AWS Elastic Beanstalk reduces management complexity without restricting choice or control.
- You simply upload your application, and AWS Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

Elastic Beanstalk

- Managed service
  - Instance configuration / OS is handled by Beanstalk
  - Deployment strategy is configurable but performed by Elastic Beanstalk
  - Capacity provisioning
  - Load balancing & auto-scaling
  - Application health-monitoring & responsiveness

- Just the application code is the responsibility of the developer

Elastic Beanstalk

- Support for many platforms:
  - Go
  - Java SE
  - Java with Tomcat
  - .NET on Windows Server with IIS
  - Node.js
  - PHP
  - Python
  - Ruby
  - Packer Builder
  - Single Container Docker
  - Multi-Container Docker
  - Preconfigured Docker
- If not supported, you can write your custom platform (advanced)

AWS Elastic Beanstalk Features:

- **Application**: Elastic Beanstalk directly takes in out project code. So Elastic Beanstalk application is named the same as your project home directory.
- **Application Environments**: Users may want their application to run on different environments like DEV, PROD. You can create and configure different environments to run application on different stages.
- **Environment Health**: One of the most lucrative features about running application on AWS or most of the other cloud platforms is the automated health checks. AWS runs automatic health checks on all EC-2 deployments (Elastic Beanstalk is a managed EC-2 service) which can be monitored from AWS console. For example, in case of web applications AWS will regularly, as scheduled by the developers, ping the application to check if the response is status code 200 and the application is running as expected. Health check responses:
- Red: Application failed all health tests.
- Yellow: Application failed some of the health tests.
- Grey: Application is updating.
- Green: Application passed health check successfully.

AWS Elastic Beanstalk Features:

- **Isolated**: All environments within a single application are isolated from each other (independent of each others' running states). Needless to say two different applications are also isolated.
- **Scalability**: Using Auto-Scaling within Elastic beanstalk makes the application dynamically scalable.
- **Elastic Load Balancing**: All the web requests to the application are not directly relayed to application instances. They first hit the Elastic Load Balancer (ELB), which, as the name suggests, balances the load across all the application instances.
- **Language support**: Elastic Beanstalk supports the applications developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS.
- **Pricing**: There is no extra charge for using Elastic Beanstalk. Users are only required to pay for the services and resources provisioned by Elastic Beanstalk Service.
- **Automatic Provisioning**: Elastic Beanstalk takes away the burden of choosing the right services and configuring their security groups to work together.
- **Impossible to Outgrow**: AWS claims that since Elastic Beanstalk uses Auto Scaling feature it can, in theory, handle any amount of internet traffic.

AWS Key Management Service (AWS KMS)

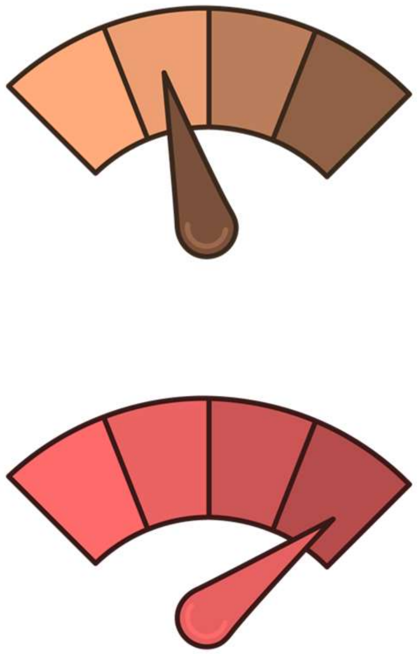
AWS Key Management Service (AWS KMS) features:

- Enables you to **create and manage encryption keys**
- Enables you to control the use of encryption across AWS services and in your applications.
- Integrates with AWS CloudTrail to log all key usage.
- Uses hardware security modules (HSMs) that are validated by Federal Information Processing Standards (FIPS) 140-2 to protect keys

AWS Key Management Service (AWS KMS)

Amazon CloudWatch

Amazon CloudWatch



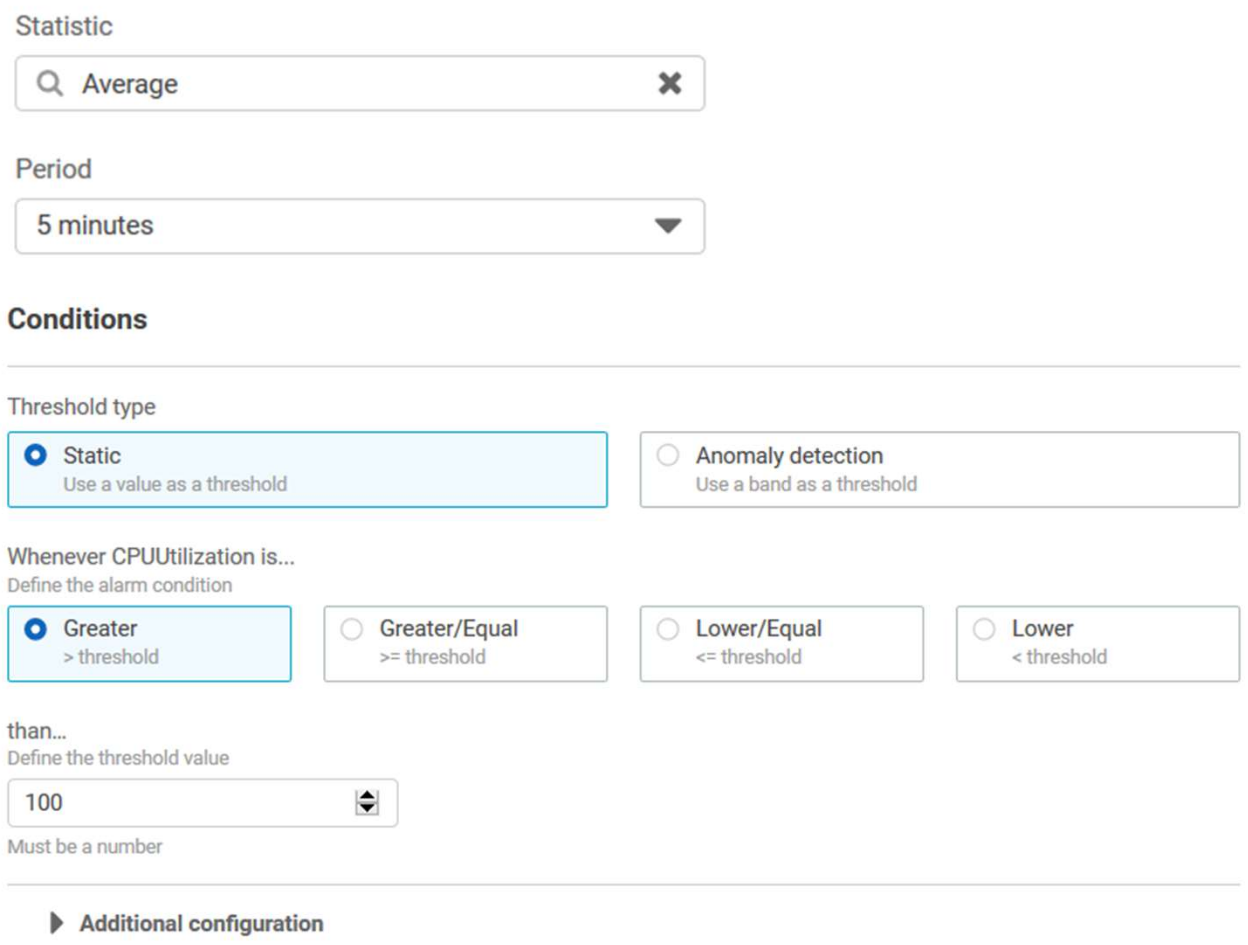
- Monitors –
  - AWS resources
  - Applications that run on AWS
- Collects and tracks –
  - Standard metrics
  - Custom metrics
- Alarms –
  - Send notifications to an Amazon SNS topic
  - Perform Amazon EC2 Auto Scaling or Amazon EC2 actions
- Events –
  - Define rules to match changes in AWS environment and route these events to one or more target functions or streams for processing

AWS CloudWatch Metrics

- Metrics are time-series data taken at a regular interval and which are identified by a name and (optionally) dimensions.
- A metric name could be “CPUUtilization” or “disk_used_percent.”
- Dimensions could be for example “InstanceId”, which would separate metric data for two different EC2 instances.
- Additionally for AWS, metric data is stored inside a “namespace,” which acts like a folder. Examples of namespaces are:

CloudWatch alarms

- Create alarms based on –
  - Static threshold
  - Anomaly detection
  - Metric math expression
- Specify –
  - Namespace
  - Metric
  - Statistic
  - Period
  - Conditions
  - Additional configuration
  - Actions



Statistic

Average

Period

5 minutes

Conditions

Threshold type

[✓ Static — Use a value as a threshold] Anomaly detection — Use a band as a threshold

Whenever CPUUtilization is...

Define the alarm condition

[✓ Greater — > threshold] Greater/Equal — >= threshold Lower/Equal — <= threshold Lower — < threshold

than...

Define the threshold value

100

Must be a number

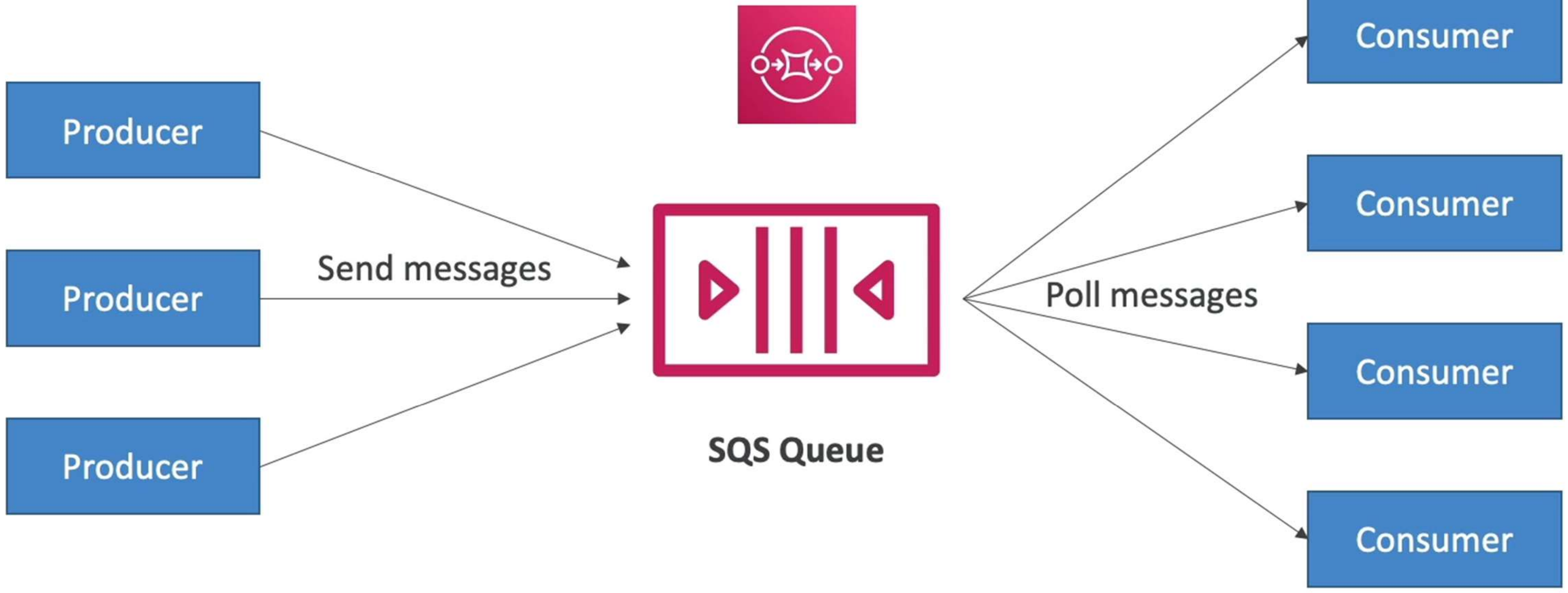
Additional configuration

AWS Messaging Services(SNS,SQS,SES).

Simple Notification Service

- Simple Notification Service (SNS) is a fully managed service offered by Amazon. It is helpful for application-to-application (A2A) and application-to-person (A2P) communication.
- It is highly used for **event notifications, monitoring various applications,** time-sensitive information updates, workflow systems, etc.
- It can also be used in workflow systems to relay events among various distributed computing applications.
- It helps developers build applications that are based on **real-time events**.

Amazon SQS – Simple Queue Service What's a queue?



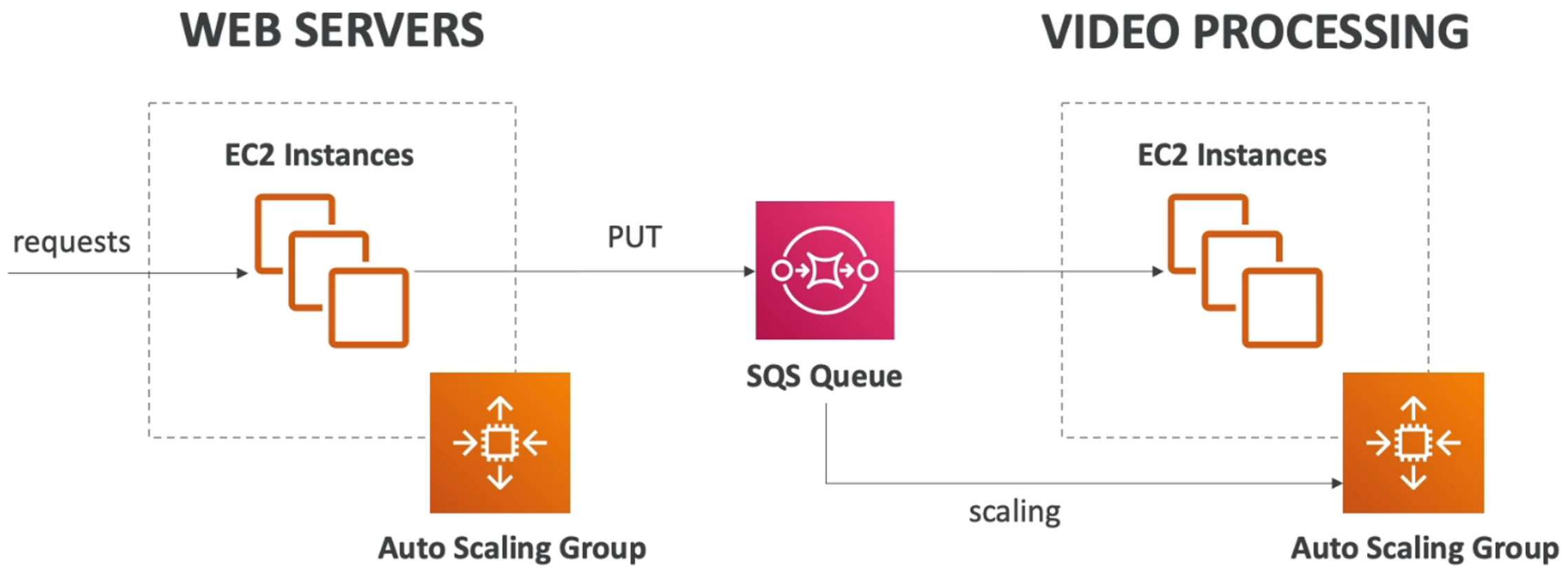
Simple Queue Service

- Simple Queue Service (SQS) is a **fully managed service** offering from Amazon. It helps to **decouple and scale microservices, serverless applications,** and **distributed systems.**
- It removes the complexity and overhead associated with managing and operating message-oriented middleware.
- SQL lets you send, store and receive messages of any volume. It is a simple process to start with an SQS service on an AWS console or command-line interface.
- SQS is a great choice to **determine the load on the application**, and it can scale up and down as per the load.
- The combination of both SQS and SNS helps to **send identical copies of a message to multiple queues**.

Amazon SQS – Standard Queue

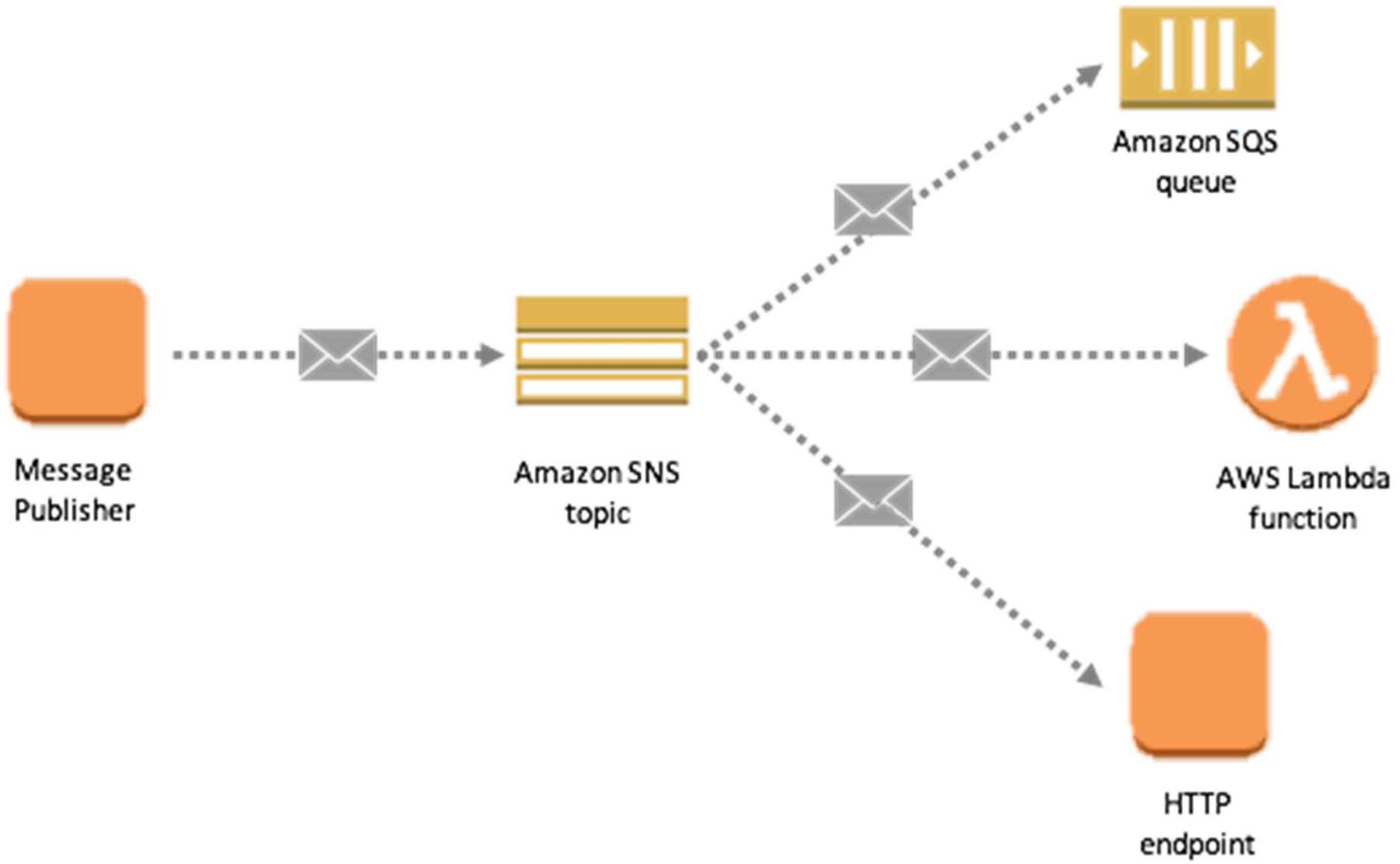
- Oldest AWS offering (over 10 years old)
- Fully managed service (~serverless), use to **decouple** applications
- Scales from 1 message per second to 10,000s per second
- Default retention of messages: 4 days, maximum of 14 days
- No limit to how many messages can be in the queue
- **Messages are deleted after they're read by consumers**
- Low latency (<10 ms on publish and receive)
- **Consumers share the work to read messages & scale horizontally**

SQS to decouple between application tiers



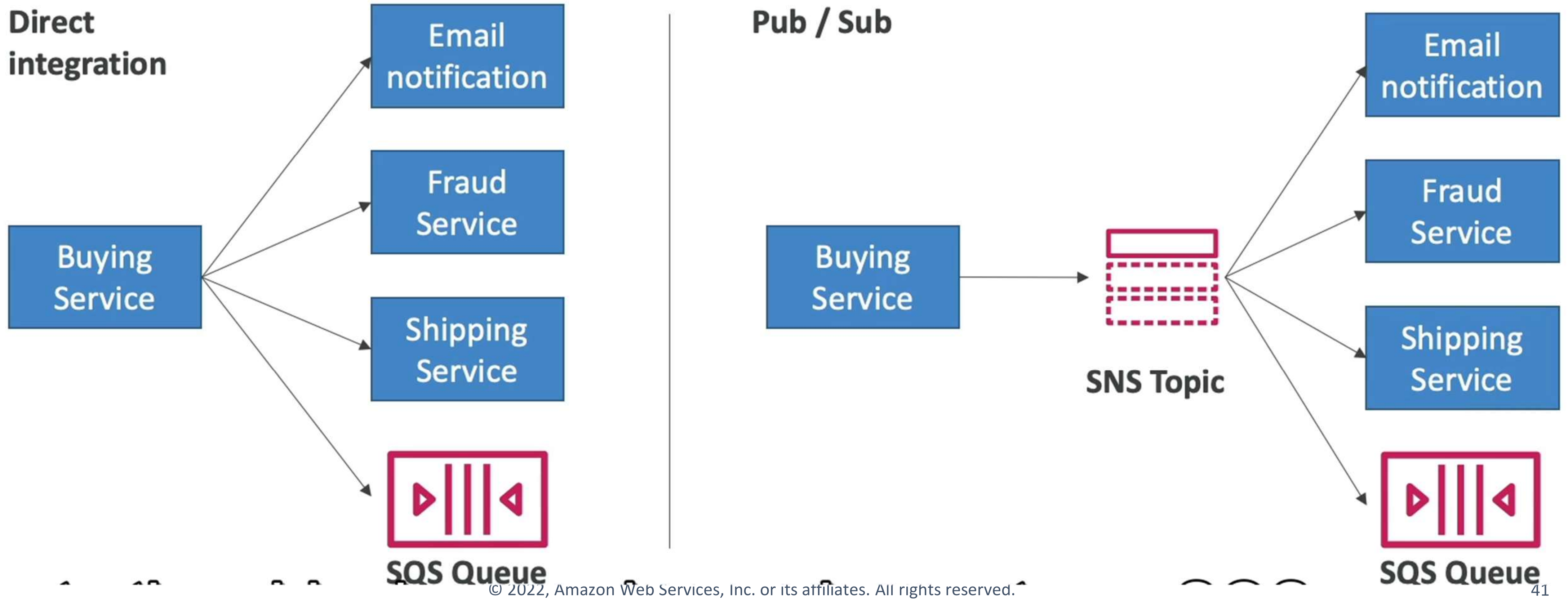
Simple Email Service

- Simple Email Service (SES) is a flexible, highly scalable, and cost-effective service that allows application developers to send emails from within the application.
- There are straightforward steps to configure SES service quickly.
- We can use it for transactional emails, marketing emails, and sending emails in bulk.
- You can use it for **Transactional Emails** such as purchase confirmation or password resets etc.
- You can also use the Amazon SES service for **marketing purposes** such as sending special offers, newsletters, etc.
- You can **send bulk emails** to such as notifications and announcements to large organizations or communities.


Amazon SQS queue
Message Publisher
Amazon SNS topic
AWS Lambda function
HTTP endpoint

Amazon SNS

- What if you want to send one message to many receivers?



Direct integration
Buying Service
Email notification
Fraud Service
Shipping Service
SQS Queue
Pub / Sub
Buying Service
SNS Topic
Email notification
Fraud Service
Shipping Service
SQS Queue