**AWS CLOUD TECHNICAL ESSENTIALS**

**Modules:**

|  |  |
| --- | --- |
| Week - 1 | AWS Overview and Security |
| Week - 2 | Compute and Networking |
| Week – 3 | Storage and Databases on AWS |
| Week – 4 | Monitoring and Optimization |

Applications:

**App:1 – Employee Directory Application**

**A computer screen shot of a diagram

Description automatically generated**

* Application will be built on Amazon Private N/W (**Amazon VPC)**
* Host the application code on Amazon Elastic cloud/ **EC2** (offers VMs)
* The data will be stored in DB which will reside in this N/W and will be hosted using Amazon RDS
* Images for the employees will be stored using Object Storage Service (**S3)** – allows unlimited storage of any type of file.
* Use **Amazon CloudWatch** to monitor our solution and we will take care of scalability and fault tolerance (**Amazon Load Balancing** and **EC2 Auto Scaling**)
* For security and identity, we use Identity Access and Management (**IAM**)

**WEEK – 1: AWS OVERVIEW AND SECURITY**

***What is Cloud?***

Cloud computing is the on-demand delivery of IT resources over the internet with pay-as-you-go pricing. You no longer need to manage and maintain your own hardware in your own data centers. Companies like AWS own and maintain these data centers and provide virtualized data center technologies and services to users over the internet.

***Benefits of Cloud Computing:***

* Pay as you go
* Benefit from massive economies of scale
* Stop guessing capacity
* Increase speed and agility
* Stop spending money running and maintaining data centers
* Go global in minutes

***Types of Cloud Computing:***

* **IaaS** (Infrastructure as a Service):

*It contains the basic building blocks for cloudIT.*

*It provides access to networking features, computers (virtual/ dedicated hardware), and data storage space.*

* **PaaS** (Platform as a Service):

*It removes the need for you to manage underlying infrastructure (usually hardware and OS) and allows you to focus on the deployment and management of your apps.*

* **SaaS** (Software as a Service):

*It provides you with a complete product that is run and managed by a service provider.*

*Refers to end-user apps.*

**Cloud types based on deployment model:**

* Public cloud
* Private cloud / on-premises
* Hybrid cloud

1. **AWS Global Infrastructure**

**A diagram of data center

Description automatically generated**

AWS provides Redundancy:

|  |  |
| --- | --- |
| A diagram of a data center  Description automatically generated | A copy of your data is always stored in another data center that can provide the data with low latency when you lose the original data.  **Availability zones** basically contain two or more data centers. |
| A screen shot of a computer  Description automatically generated | A cluster of availability zones is called a **Region**. It is where AWS hosts its data centers/ They are named after the location where they reside. They are independent from one another (Data is not replicated from one region to another without your explicit consent and authorization). |
| **Eg:**   * + us-east-1 -> **N**.**virginia** –> first region created in the east of the US.   + Ap-northeast-1 -> **Tokyo** –> first region created in the northeast of Asia Pacific | |

AWS Region Considerations:

* + Compliance
  + Latency
  + Pricing
  + Service availability

**Global Edge Network:**

It consists of **Edge Locations** and **Regional Edge Cache.**

Edge locations and Regional edge caches are used to cache content closer to end users, thus reducing latency.

Note: At a minimum, you should use two AZs. If one entire AZ fails, your application will have infrastructure up and running in at least a second AZ to take over the traffic.

**AWS Service endpoints**

To connect programmatically to an AWS service, you use an endpoint.

“An endpoint is the URL of the entry point for an AWS web service”.

The AWS SDKs and AWS CLI automatically use the default endpoint for each service in an AWS Region. But you can specify an alternate endpoint for your API requests.

If a service supports Regions, the resources in each region are independent of similar resources in other regions.

**EG:** You can create EC2 instance/ AWS SQS queue in one region. When you do, the instance/ queue is independent of instances/ queues in all other regions.

1. Regional endpoints:

**Syntax:** *protocol*://*service-code*.*region-code*.amazonaws.com

Example: https://dynamodb.us-west-2.amazonaws.com

General endpoints: The following services support general endpoints also which doesn’t include region name

|  |  |
| --- | --- |
| **Service** | **General Endpoint** |
| EC2 | Ec2.amazonaws.com |
| EC2 Auto Scaling | Autoscaling.amazonaws.com |
| AWS EMR | Elasticmapreduce.amazonaws.com |

1. Global endpoints: Global services do not support regions.

The following services have a single global endpoint:

* + - Amazon CloudFront
    - AWS Global Accelerator
    - AWS IAM
    - AWS Network Manager
    - AWS Organizations
    - AWS Route 53
    - AWS Shield Advanced
    - AWS WAF Classic

1. View the Service endpoints:
   * + Open service endpoints and Quotas -> search for service name and open -> view the information in the page.
     + To programmatically check using SDK for java -> <https://docs.aws.amazon.com/sdk-for-java/v2/developer-guide/java-dg-region-selection.html#region-selection-query-service>
     + To programmatically view region and service info using Systems manager -> <https://docs.aws.amazon.com/systems-manager/latest/userguide/parameter-store-public-parameters.html#parameter-store-public-parameters-global-infrastructure>
2. FIPS endpoints (Federal Information Processing Standard):

They use a TLS software lib that complies with FIPS 140-2.

These endpoints might be required by enterprises that interact with US govt.

To specify FIPS endpoint when you call an AWS operation, use a mechanism provided by the tool that you are using to make the call.

**EG:** AWS SDKs provide:

* + - Set the **AWS\_USE\_FIPD\_ENDPOINT** env var to **TRUE**
    - Add **use\_fipd\_enpoint = true** to your **~/.aws/config** file.

AWS CLI supports these mechanisms, and also provides the **–endpoint-url** option.

aws kms create-key --endpoint-url https://kms-fips.us-west-2.amazonaws.com

1. Dual stack endpoints: Some AWS services offer dual stack endpoints, so that you can access them using either IPv4 or IPv6 requests.
   * + For both dual and single stack endpoints:

Syntax: *protocol*://*service-code*.*region-code*.api.aws

* + - For only dual stack endpoints

Syntax: *protocol*://*service-code*.*region-code*.amazonaws.com

**Ways to interact with AWS API:**

1. AWS Management console
2. AWS Command Line Interface (CLI))

|  |  |
| --- | --- |
| **AWS configure** | $ aws configure  AWS Access Key ID: MYACCESSKEY  AWS Secret Access Key: MYSECRETKEY  Default region name [us-west-2]: us-west-2  Default output format [None]: json |
| **To use environment variables** | $ export AWS\_ACCESS\_KEY\_ID = <access\_key>  $ export AWS\_SECRET\_ACCESS\_KEY=<secret\_key> |
| **To use shared credentials file** | [default]  aws\_access\_key\_id = MYACCESSKEY  aws\_secret\_access\_key = MYSECRETKEY  [testing]  aws\_access\_key\_id = MYACCESSKEY  aws\_secret\_access\_key = MYSECRETKEY  and place it in **~/.aws/credentials** or in **%UserProfile%\.aws/credentials** on windows |
| **To tell aws-cli where to find credentials** | $export AWS\_SHARED\_CREDENTIALS\_FILE=/path/to/shared\_credentials\_file |

**Basic AWS CLI commands:**

|  |  |
| --- | --- |
| List s3 buckets | $ aws s3 ls |
| To view help documentation | $ aws help  $ aws <command> help  $ aws <command> <subcommand> help |
| The get the version of AWS CLI | $ aws –version |
| To turn on debugging output | $ aws –debug <command> <subcommand> |

1. AWS Software Development Kit (SDK) (application+AWS)
2. **AWS Identity and Access Management (IAM) best practices:**

***What is an IAM?***

IAM is a web service that enables you to manage access to your AWS account and resources.

It also provides a centralized view of who and what are allowed inside your AWS account (authentication), and who and what have permissions to use and work with your AWS resources (authorization).

With IAM, you can share access to an AWS account and resources without having to share your set of access keys and passwords.

You can also provide granular access to those working in your account, so that people and services only have permissions to the resources they need. Eg: read-only access to an AWS service.

***IAM Features:***

* Global and not specific to a region -> you can see and use your IAM configs from any region.
* Integrated with many AWS services by default.
* You can establish password policies in IAM to specify complexity requirements and mandatory rotation periods for users.
* Supports MFA.
* Supports identity federation -> allows users who already have passwords elsewhere to get temporary access to your AWS account.
* Any AWS customer can use IAM; the service is offered at no additional charge.

***What is an IAM user?***

Represents a person/ service that interacts with AWS. You can define the user within your AWS account and any activity done by that user is billed to your account.

Once you create a user, that user can sign in to gain access to the AWS resources in your account.

You can also add more users to your account as needed. Each person should have their own login credentials which prevents sharing of credentials.

***IAM User Credentials:***

IAM user has **name** and **set of credentials.**

IAM user credentials are considered permanent, in that they stay with the user until there is a forced rotation by admin.

When you create IAM user, you have the option to grant permissions directly at the user level. This is a good idea if you have few users.

For too many users you can consider the group level access permissions instead.

When creating a user, you can choose to provide the user:

* + Access to the AWS management console
  + Programmatic access to the AWS CLI and AWS API

***What is an IAM Group?***

Collection of users. All users in the group inherit the permissions assigned to the group. Makes it easy to give permissions to multiple users at once. Most scalable way of managing permissions for the users in AWS account.

Note:

* Groups can have many users.
* Users can belong to many groups.
* Groups cannot belong to groups.

***What is an IAM Policy?***

To manage access and provide permissions to AWS services and resources, you create IAM policies and attach them to IAM users, groups and roles. Whenever a user or role makes a req, AWS evaluates the policies associated with them to determine if the req should be allowed/ denied.

Most policies are stored in AWS as JSON documents with several policy elements.

**Example**: A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

IAM Policies: Granular permissions

* + Grant or deny permission to take actions
  + Actions are AWS API calls
  + Attach policies to AWS identities

Example:

A screen shot of a computer code

Description automatically generated

For IAM policies, grant least privilege, or only the permissions required to perform a task.

A diagram of a group of people

Description automatically generated IAM for single user or group

Login with root user -> Root user authentication -> create IAM user -> logout of root user -> login with IAM user credentials

You cannot apply a policy to root user but you can apply policy to IAM user.

A screenshot of a computer

Description automatically generated

***When to use IAM?***

* Central set of identities and assignments.
* Access to accounts across an entire amazon organization.
* Connection to your existing identity provider.
* Temporary credentials.
* MFA.
* Self-service MFA config for end users.
* Administrative enforcement of MFA usage.
* Single sign-on to all Amazon web services account entitlements.

***When to create an IAM user?*** (Recommended for use cases not supported by federated users). Some use cases are as follows:

* Workloads that cannot use IAM roles.
* Third party Amazon clients
* Amazon CodeCommit access
* Amazon Keyspaces access
* Emergency access

***When to create an IAM role?***

* You’re creating an app that runs on EC2 and that app makes reqs to amazon.
* You’re creating an app that runs on mobile phones and that makes reqs to amazon.
* Users in your company are authenticated in your corporate network and want to be able to use Amazon without having to sign in again; you want to allow users to federate into amazon.

***Role Based Access in AWS:***

IAM roles:

* + Not static login credentials.
  + Are assumed programmatically.
  + Credentials are temporary for a configurable amount of time.
  + Credentials expire and are rotated.

For our application example:

A screenshot of a computer

Description automatically generated

When one service needs to send an API call to another service then it uses role-based access.

Here, the role gains temporary creds and sends API call to the other service who then verified the request.

***Identity Provider:***

1. **AWS Account root user best practices**

|  |  |
| --- | --- |
| **Authentication** | **Authorization** |
| * If the user types in the correct email and password, the system assumes the user is allowed to enter and grants them access. This process is authentication. * It ensures that the **user is who they say they are.** * **Egs:** Usernames and passwords, token-based, biometric data etc | * Authorization is the process of giving users permission to access AWS resources and services. * It determines whether the user can perform an action – R/W/D/create resources. * It ensures **what actions you can perform.** |

Protect the AWS root user:

Email address you create AWS account – root user – unrestricted to all services.

AWS root user has two sets of credentials associated with it.

1. **Email address and password** – used to create the account
2. **Access keys** – allows you to make programmatic requests from AWS CLI or AWS API.

Access keys consist of 2 parts:

* + - * **Access key ID** -> eg: A2lAl5EXAMPLE
      * **Secret access key** -> eg: wJalrFE/KbEKxE

You need both the access key ID and secret access key to authenticate your requests via AWS CLI or AWS API.

* + - MFA – Multi Factor Authentication

A screenshot of a phone

Description automatically generated

Best Practices for root user:

* + - User a strong root user pwd
    - Secure root user sign-in with MFA
    - Do not create access keys for the root user
    - Use multi-person approval for root user sign-in wherever possible
    - Use a group email address for root user credentials
    - Restrict access to account recovery mechanisms
    - Monitor access and usage
    - Secure your organizations account root user credentials

To lockdown the root user:

* + Don’t share the creds associated with the root user.
  + Consider deleting the root user ace keys.
  + Enable MFA on the root account.

**Follow the principle of Least privilege:**

It is a standard security principle that advises you to grant only the necessary permissions to do a particular job and nothing more. To implement least privilege for access control, start with the minimum set of permissions in an IAM policy and then grant additional permissions as necessary for a user/ group/ role.

**Use IAM appropriately:**

Not used for authentication and authorization, does not support security controls for protecting OS and networks. It is used to secure access to your AWS account and resources and provides a way to create and manage users, groups and roles to access resources within a single AWS account.

**Use IAM roles when possible:**

IAM dynamically provides temp creds that expire after a defined period between 15mins and 36 hrs.

Users, on the other hand, have long-term credentials in the form of user name and password combinations or a set of access keys. User access keys only expire when you or the admin of you account rotates these keys. User login creds expire if you have applied a password policy to your account that forces users to rotate their passwords.

**Consider using an Identity Provider:**

To manage employee identity.

Using and IdP, whether it be an AWS service such as AWS IAM Identity center or a third-party identity provider, provides you a single source of truth for all identities in your organization. You no longer have to create separate IAM users. You can instead use IAM roles to provide permissions to identities that are federated from your IdP.

**Consider an AWS IAM Identity Center:**

It is an IdP that lets your users sign in to a user portal with a single set of credentials (like SSO). It then provides them access to all their assigned accounts and applications in one central location.

A diagram of a diagram

Description automatically generated

It is similar to IAM, in that it offers a dir where you can create users, organize them in groups, and set permissions across those groups and grant access to AWS resources. However, in AWS IAM Identity center has some advantages over IAM. It removes the burden of having to re-create users that already exists elsewhere and it enables you to manage those users from your IdP. It separates the duties between your IdP and AWS, ensuring that your cloud access management is not inside or dependent on your IdP.

1. **AWS APIs**
2. **AWS Shared responsibility model:**

In AWS Cloud, managing security and compliance is a shared responsibility between AWS and you.

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| **AWS** | **You** |
| Responsible for security of the cloud (infrastructure that runs all the services in AWS).  Protecting & securing AWS regions, AZs, and data centers down to the physical security of the buildings.  Managing the hardware, software and networking components that run AWS services, such as physical server, host OS, virtualization layers, and AWS networking components.  Container services refer to AWS abstracting app containers behind the scenes, not Docker container services. This enables AWS to move the responsibility of managing that platform away from customers. | You are responsible for security in cloud.  For properly configuring the service and your apps, ensuring your data is secure.  Level of responsibility depends on AWS service you are using.  You are responsible for Packing and upgrades.  Examples of your responsibilities:   * Choosing a region for AWS resources according to data regulations. * Implementing data protection mechanisms. * Using access control to limit who has access to your data and AWS resources. |

Responsibility of AWS:

A screenshot of a white and black list

Description automatically generated

Responsibility of AWS vs Customer:

Using the three categories of AWS services, you can determine your level of responsibility for each AWS service you use.

A screenshot of a computer

Description automatically generated

* **Inherited controls:**

Controls which a customer fully inherits from AWS.

* + Physical and Environmental controls
* **Shared controls:**

Controls which apply to both the infrastructure and customer layers, but in completely separate contexts/ perspectives. In a shared control, AWS provides the requirements for the infrastructure and the customer must provide their own control implementation within their use of AWS services.

* + Patch management
  + Configuration management
  + Awareness and Training
* **Customer Specific:**

Controls which are solely the responsibility of the customer based on the application they are deploying within the AWS Services.

* + Service and communication protection or Zone security which may require a customer or route or zone data within specific security environments.

**LAB-1: INTRODUCTION TO IAM**

* Explore IAM Users and Groups.
* Inspect IAM policies applied to groups.
* Follow a real-world scenario that adds users to groups and explores group permissions.
* Locate and use the IAM sign-in URL.
* Experiment with policies and service access.

Lab Overview:

3 users -> **user-1, user-2, user-3**

3 groups -> **S3 Support** (Read-Only access); **EC2 Support** (Read-Only); **EC2 Admin** (View, Start, stop Access)

**IAM for employee directory application:**

**Create Roles:**

* + AWS service – trusted entity type – EC2, lambda and other services
  + AWS account – Allows you to allow cross accounts access to permissions for resources in your account.
  + Web identity – Allows for federated users to assume a role.
  + SAML 2.0 federation – if you have corporate directory that is on premises that would be using SAML.

**Permission Policies:** example:

**AmazonS3FullAccessPolicy:**

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3:\*",

"s3-object-lambda:\*"

],

"Resource": "\*"

}

]

}

\*-> any resource

**EC2:** I a compute engine that allows you to host VMs.

Every EC2 must reside in a N/W in AWS. By default AWS services reside in VPC (Virtual Private Network)

**Instance:** One single virtual machine

**Amazon Linux 2 user data script:**

#!/bin/bash -ex

wget <https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/DEV-AWS-MO-GCNv2/FlaskApp.zip>

unzip FlaskApp.zip

cd FlaskApp/

yum -y install python3 mysql

pip3 install -r requirements.txt

amazon-linux-extras install epel

yum -y install stress

export PHOTOS\_BUCKET=${SUB\_PHOTOS\_BUCKET}

export AWS\_DEFAULT\_REGION=<INSERT REGION HERE>

export DYNAMO\_MODE=on

FLASK\_APP=application.py /usr/local/bin/flask run --host=0.0.0.0 --port=80

**Amazon Linux 2023 user data script:**

#!/bin/bash -ex

wget <https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/DEV-AWS-MO-GCNv2/FlaskApp.zip>

unzip FlaskApp.zip

cd FlaskApp/

yum -y install python3-pip

pip install -r requirements.txt

yum -y install stress

export PHOTOS\_BUCKET=${SUB\_PHOTOS\_BUCKET}

export AWS\_DEFAULT\_REGION=<INSERT REGION HERE>

export DYNAMO\_MODE=on

FLASK\_APP=application.py /usr/local/bin/flask run --host=0.0.0.0 --port=80