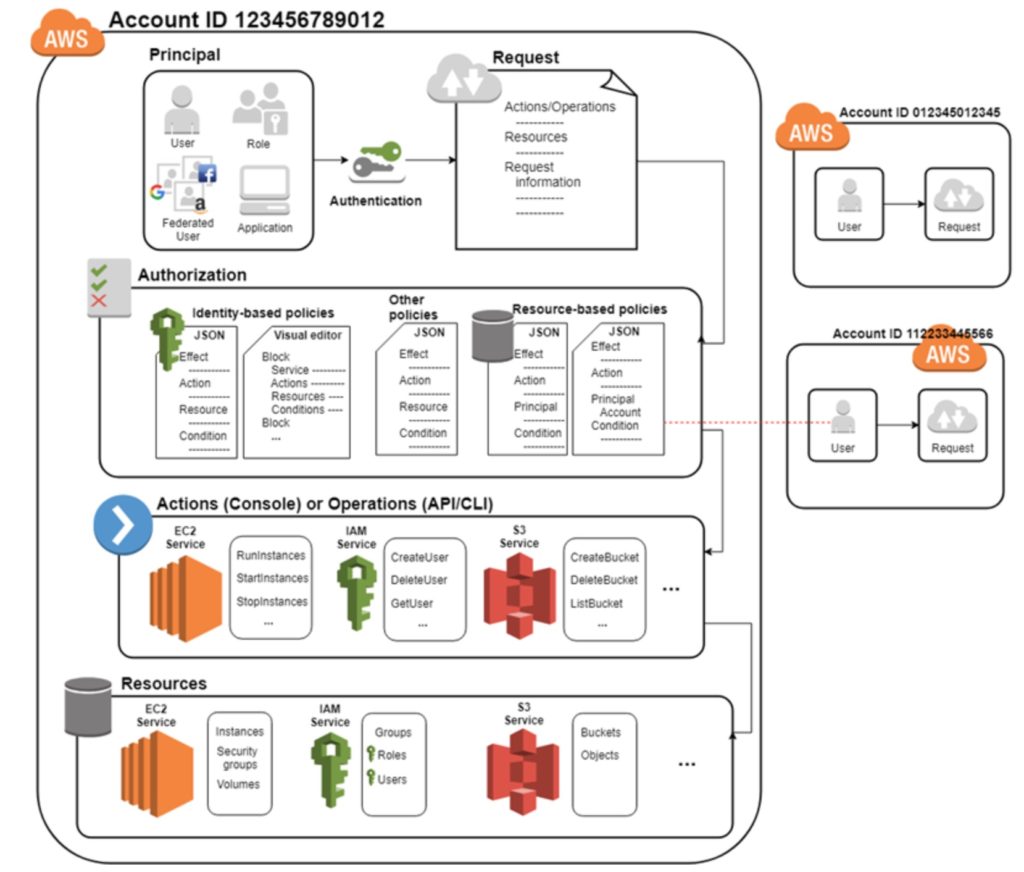
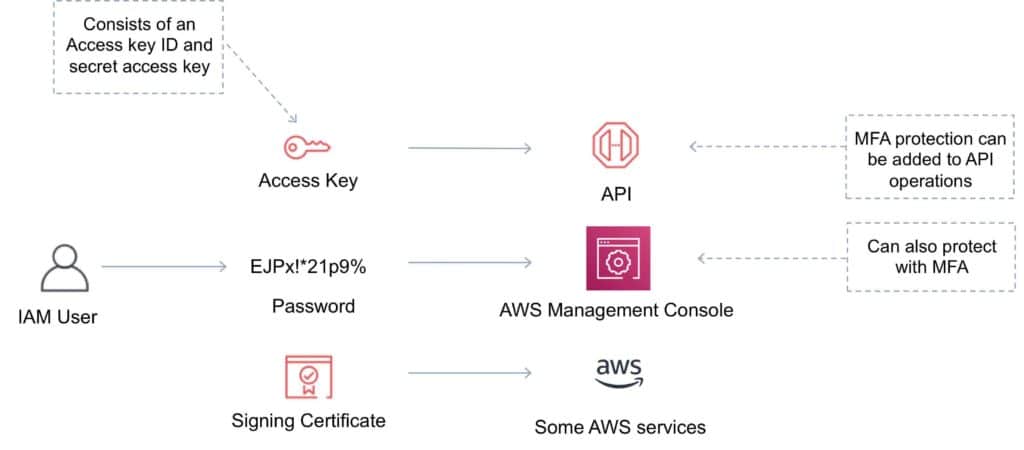
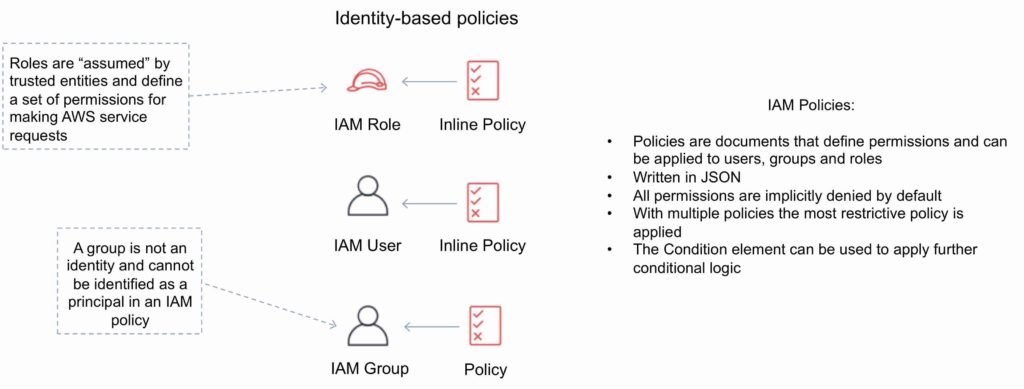
**Identity Access Management(IAM)**

* IAM is used to securely control individual and group access to AWS resources
* IAM makes it easy to provide multiple users secure access to AWS resources
* IAM can be used to manage
  + Users
  + Groups
  + Access Policies
  + Roles
  + User credentials
  + User password policies
  + Multi-factor authentiacation(MFA)
  + API keys for programmatic access(CLI)
* Provides centralized control of your AWS account
* Enables shared access to your AWS account
* By default new users are created with NO access to any services – they can only login to the console
* Permission must be explicitly granted to allow a user to access an AWS service
* IAM users are individuals who have been granted access to an AWS account
* Each IAM user has three main components
  + User-name
  + Password
  + Permissions to access various resources
* You can apply granular permissions with IAM
* You can assign users individual security credentials such as access keys, passwords and MFA deevices
* IAM is not used for application-level authentication
* Identity Federation (including AD, Facebook etc) can be configured allowing secure access to resources in an AWS accout without creating an IAM user account
* MFA can be enabled/enforced for the AWS account and for individual users under the account
* MFA uses an authentication device that continually generates random six-digit single use authentication codes
* You can authenticate using an MFA device in the following three ways:
  + Through the AWS Management Console – the user is prompted for a user name, password and authentication code
  + Using the AWS API – restrictions are added to IAMO policies and developers can request temporary security credentials and pass MFA parameters in ther AWS STS API requests
  + Using the AWS CLI by obtaining temporary security credentials from STS(aws sts get-session-token)
* It is a best practice to use MFA for all users and to use U2F or hardware MFA devices for all priveleged users
* IAM is universal and does not apply to regions
* IAM is eventually consistent
* IAM replicates data across multiple data centers around the world
* x
* It is a best practice to not use the root account for anything other than billing
* Power user access allows all permissions except the management of groups and users in the IAM
* Temporary security credentials consist of the AWS access key ID, secret access key, and security token
* IAM can assign temporary security credentials to provide users with temporary access to services/resources
* To sign-in you must provide your account ID or account alias in addition to a user name and password
* The sign-in URL includes the account ID or account alias, e.g.: <https://My_Aws_Account_ID.signin.aws.amazon.com/console>.
* Alternatively you can sign-in at the following URL and enter your account ID or alias manually
  + httpsL//console.aws.amazon.com/
* IAM integrates with many different AWS services
* IAM supports PCI DSS compliance
* AWS recommends that you use the AWS SDKs to make programmatic API calls to IAM
* However, you can also use the IAM Query API to make direct calls to the IAM web service
* **IAM Infrastructure elements**
* 
  + Principals:
    - An entity that can take an action on an AWS resource
    - Your administrative IAM user is your first principal
    - You can allow suers and services to assume a role
    - IAM supports federated users
    - IAM supports programmatic access to allow an application to access your AWS account
    - IAM users, roles, federated users, and applications are all AWS principals
  + Requests
    - Principals send requests via the Console, CLI, SDKs or APIs
    - Requests are:
      * Actions (or operations) that the principal wants to perform
      * Resources upon which the actions are performed
      * Principal information including the environment from which the request was made
    - Request context – AWS gathers the request information:
      * Principal(requester)
      * Aggregate permissions associated with the principal
      * Environment data, such as IP address, user agent, SSL status etc
      * Resource data, or data that is related to the resource being requested
    - Authentication
      * A principal sending a request must be authenticated and to send a request to AWS
      * To authenticate from the console you must sign in with your user name and password
      * To authenticate form the API or CLI you must provide your access key and secret key
    - Authorization:
      * IAM uses values form the requets context to check for matching policies and determines whether to allow or deny the request
      * IAM policies are stored in IAM as JSON documents and specify the permissions that are llowed or denied
      * IAM policeies can be:
        + User (identity) based policies
        + Resource-based policies
      * IAM checks each policy that matches the context of your request
      * If a single policy has a deny action IAM denies the request and stops evaluating(explicit deny)
      * Evaluation logic:
        + By default all requests are denied(implicit deny)
        + An explicit allow overrides the implicit deny
        + An explicit deny overrides any explicit allows
      * Only the root user has access to all resources in the account by default
    - Actions:
      * Actions are defined by a service
      * Actions are the things you can do to a resource such as viewing, creating, editing, deleting
      * Any actions on resources that are not explicitly allowed are deined
      * To allow a principal to perform an action you must include the necessay actions in a policy that pplies to the principal or the affected resource
    - Resources:
      * A resource is an entity that exists within a service
      * E.g. EC2 Instances, S3 buckets, IAM users, and DynamoDB tables
      * Each AWS service defines a set of actions that can be performed on the resource
      * After AWS approves the actions in your request, those actions can be performed on the related resources within your account
* **Authentication Methods**
  + Console password:
    - A passord that the user can enter to sign into interactive sessions such as the AWS Management Console
    - You can allow users to change their own passwords
    - Cb u s You can allow selected IAM users to change their passwords by disabling the option for all users and using an IAM policy to grant permissions for the selected users
  + Access Keys:
    - A combination of an **access key ID** and a **secret access key**
    - You can assign two active access keys to a user at a time
    - These can be used to make programmatic calls to AWS when using the **API** in program code or at a command prompt when using the **CLI**  or the **Powershell**
    - You can create, modify, view or rotate access keys
    - When created IAM returns the access key ID and secret access key
    - The secret access is returned only at creation time and if lost a new key must be created
    - Ensure access keys and secret access keys are stored securely
    - Users can be given access keys to change their own keys through IAM policy(not form the console)
    - You can disable a user’s access key which prevents it from being used for API calls
  + Server certificates:
    - SSL/TLS certificates that you can use to authenticate with some AWS services
    - AWS recommends that you use the AWS Certificate Manager(ACM) to provision, manage and deploy your server certificates
    - Use IAM only when you must support HTTPS connections in a region that is not supported by ACM
    - 
* **IAM Users**
  + An IAM user is an entity that represents a person or service
  + Can be assigned:
    - An access key ID and secret access key for programmatic access to the AWS API, CLI, SDK, and other development tools
    - A password for access to their management console
  + By default users cannot access anything in your account
  + The account root user credentials are the email address used to create the account and a password
  + The root account has full administrative permissions and these cannot be restricted
  + Best practice for root accounts:
    - Don’t use the root user credentials
    - Don’t share the root user credentials
    - Create an IAM user and assign administrative permissions as required
    - Enable MFA
  + IAM users can be created to represent applications and these are known as ‘service accounts’
  + You can have up to 5000 users per AWS account
  + Each user account has a friendly name and an ARN which uniquely identifies the user across AWS
  + A unique ID is also created which is returned only when you create the user using the API, Tools for Windows PowerShell or the AWS CLI
  + You should create individual IAM accounts for users(best practice not to share accounts)
  + The Access Key ID and Secret Access Key are not the same as a password and cannot be used to login to the console
  + The Access Key ID and Secret Access Key can only be generated once and must be regnerated if lost
  + A password policy can be defined for enforcing password length, complexity etc
  + You can allow or disallow the ability to change passwords using an IAM policy
  + Access keys and passwords should be changed regularly
* **Groups**
  + Groups are collecitons of users and have policies attached to them
  + A group is not an identity and cannot be identified as a principal in an IAM policy
  + Use groups to assign permissions to users
  + Use the principal of least privilege when assigning permissions
  + You cannot nest groups(groups within groups)
* **Roles**
  + Roles are created and then ‘assumed’ by trusted entities and define a set of permissions for making AWS service requests
  + With IAM Roles you can delegate permissions to resources for users and services without using permanent credentials(e.g. user name and password)
  + IAM users or AWS services can assume a role to obtain temporary security credentials that can be used to make AWS API calls
  + You can delegate using roles
  + There are no credentials associated with a role(password or access keys)
  + IAM users can temporarily assume a role to take on permissions for a specific task
  + A role can be assigned to a federated user who signs in using an external identity provider
  + Temporary credentials are primarily used with IAM roles and automatically expire
  + Roles can be assumed temporarily through the console or programmatically with the CLI, Powershell or API
  + IAM Roles with EC2 instances:
    - IAM roles can be used for granting applications running on EC2 instances permissions to AWS API requests using instance profiles
    - Only one role can be assigned to an EC2 instance at a time
    - A role can be assigned at the **EC2 instace creation time or at any time afterwards**
    - When using the AWS CLI or API instance profiles must be created manually(it’s automatic and transparent through the console)
    - Applications retrieve temporary security credentials from the instance metadata
  + Role Delegation:
    - Create an IAM role with two policies:
      * Permissions policy – grans the user of the role the reuired permissions on a resource
      * Trust policy – specifies the trusted accounts that are allowed to assume the role
      * Wildcards(\*) cannot be specified as a principal
      * A permissions policy must also be attached to the user in the trusted account
* **Policies**
  + Policies are documents that define permissions and can be applied to users, gorups and roles
  + Policy deocuments are written in JSON(key value pair that consists of an attribute and a value)
  + All permissions are implicitly denied by default
  + The most restrictive policy is applied
  + The IAM policy simulator is a tool to help you understand, test, and validate the effects of access control policies
  + The Condition element can be used to apply further conditional logic
  + 
* **Security Token Service**
  + STS is a web service that enables you to request temporary, limited-privilege credentials for IAM users or for users that you authenticate(federated users)
  + By default, AWS STS is available as a global service, and all STS requests go to a single endpoint <https://sts.amazonaws.com>
  + You can optionally send your STS requests to endpoints in any region(can reduce latency)
  + All regions are enabled for STS by default but can be disabled
  + The region in which temporary credentials re requestsed must be enabled
  + Credentials will always work globally
  + STS supports CT, which records AWS calls for your AWS account and delivers log files to an S3 bucket
  + Temporary security credentials work almost identically to long-term access key credentials that IAM users can use, with the following differences:
    - Temporary security credentials are short-term
    - **They can be configured to last anywhere from minutes to hours**
    - After the credentials expire, AWS no longer recognizes them or allows any kind of access to API requests made with them
    - Temporary security credentials are not stored with the user but are generated dynamically and provided to the user when requested
    - When(or even before) the temporary security credentials expire, the user can request new credentials, as long as the user requesting them till has permissions to do so
  + Advantagest of STS are:
    - You do not have to distribute or embed long-term AWS security credentials with an application
    - You can provide access to you AWS resources to users without having to define an AWS identity for them(temporary security credentials are the basis for IAM Roles and ID Federation)
    - The temporary security credentials have a limited lifetime, so you don not have to rotate them or explicitly revoke them when they’re no longer needed
    - After temporary security credentials expire, they cannot be reused(you can specify how long the credentials are valid for, up to a maximum limit)
  + AWS STS API actions returns temporary security credentials that consist of:
    - An access key which consists of an access key and a secret ID
    - A session token
    - Expiration or duration of validity
    - Users(or an application that the user runs) can use these credentials to access your resources
  + With STS you can request a session token using one of the following APIs:
    - AssumeRole – can be used by IAM users(can be used for MFA)
    - AssumeRoleWithSAML – can be used by any user who passes a SAML authentication response that indicates authentication from a known(trusted) identity provider
    - AssumeRoleWithWebIdentity – can be used by a user who passes web identity token that indicates authentication from a known(trusted) identity provider
    - GetSessionToken – can be used by an IAM user or AWS account root user(can be used for MFA)
    - Get FederationTOken – can be used by an IAM user or AWS account root user
  + AWS recommends using Cognito for identity federation with Internet identity providers
  + Users can come from three sources
    - **Federation(typically AD):**
      * Uses SAML 2.0
      * Grants temporary access based on the users AD credentials
      * Does not need to be a user in IAM
      * Signle sign-on allows users to login to the AWS console without assigning IAM credentials
    - **Federation with Mobile Apps**
      * Use FB/Amazon/Google or other OpenID providers to login
    - **Cross Account Access:**
      * Lets users from one AWS account access resources in another
      * To make a request in a different account the resource in that account must have an attached resource-based policy with the permissions you need
      * Or you must assume a role(identity-based policy) within that account with the permissions you need
  + There are a couple of ways STS can be used
    - Scenario 1
      * Develop an Identity Broker to communicate with LDAP and STS
      * Identity Broker always authenticates with LDAP first, then with STS
      * Application then gets temporary access to AWS resources
    - Scenario 2
      * Develop an Identity Broker to communicate with LDAP and STS
      * Identity Broker authenticates with LDAP first, then gets an IAM role associated with the user
      * Application then authenticates with STS and assumes that IAM role
      * Application uses that IAM role to interact with the service
* **IAM Best Practices**
  + Lock Away your AWS Account Root User Access Keys
  + Create Individual IAM Users
  + Use Groups to Assign Permissions to IAM Users
  + 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
  + Configure a strong password policy for your users
  + Enable MFA
  + Use Roles for Applications that run on EC2 instances
  + Use roles to delegate permissions
  + Do not share access keys
  + Rotate credentials regularly
  + Remove unnecessary credentials
  + Use policy conditions for extra security
  + Monitor activity in your AWS account