

## Lesson 1 - What is Cloud Computing?

### Characteristics

- On-Demand
- Broad Network Access
- Resource Pooling - Multi-tenant environment
- Rapid Elasticity
- Measured Service

### Service Models

- Infrastructure as a Service
- Platform as a Service
- Software as a Service

### Region and Availability Zones

- Oregon(us-west-2) us-west-2a/us-west-2b/us-west-2c

### Edge Locations

- More than 100 edge locations worldwide

### Scope of Services

- Global Services
  - AWS IAM
  - Amazon CloudFront
  - Amazon Route53
- Regional Services
  - Amazon DynamoDB
  - Amazon Simple Storage Service
  - Elastic Load Balancing
  - Amazon Virtual Private Cloud
  - Amazon Elastic Block Store
  - Amazon Elastic Compute Cloud
  - Subnets

## Lesson 2 - Identity and Access Management

Authentication

Authorization

- users
- groups
- password policy
- multifactor authentication (6 digits)

AWS API (Access ID + Secret Key)

- Operating System (User name + password/key pair)
- Application (User name + password)

CLI

SDKs - python or java

Web-based Management Console

Users and Groups

- Are created and exist within IAM service
- Login to Management Console
- Can have long-term access keys
- Can enable per-user MFA device

Types of credential types

Credential	Usage
Email address + Password	Master account (root) access
Username + Password	AWS Web Console
Access Key + Secret Key	CLI, SDK
Access/Secret Keys + Session Token	Role-based access

## Create Groups/Policies

- For example, groups like admin, developers, everyone

## Create Users

- For example, users with programmatic access (enable an access key and secret access key for AWS API, CLI, SDK, and other development tools), AWS Management Console access (enable a psw that allows users to sign-in to the AWS Management Console). Set permissions.

## Policies

- Determine authorization (permissions)
- Written in JSON
- Policy Types
  - Managed Policy
    - AWS managed
    - Customer managed
  - Inline Policy
- Create policies via generator, or hand written policy

```

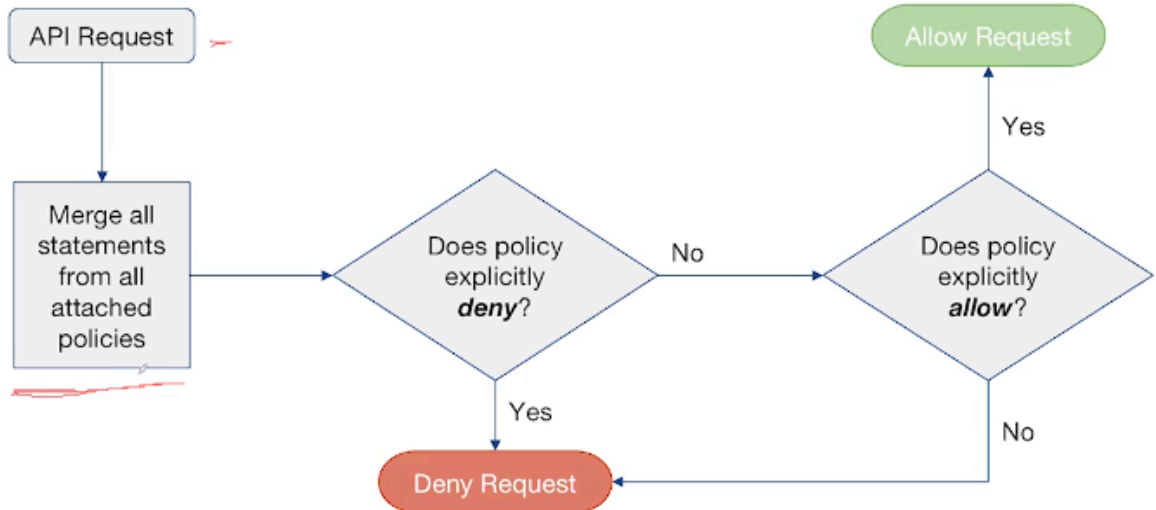
"GrantOptimusBucketPolicy": {
  "Type": "AWS::IAM::ManagedPolicy",
  "Properties": {
    "Description": {"Fn::Join": ["", [{"Ref": "OptimusVersion"}, {"data"}]]},
    "PolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [ {
        "Effect": "allow",
        "Action": [
          "s3:*"
        ],
        "Resource": [
          {"Fn::Join": ["-", [{"arn:aws:s3:::rokt"}, {"Ref": "Environment"}, {"us-west-2-feature-lake"}]]},
          {"Fn::Join": ["-", [{"arn:aws:s3:::rokt"}, {"Ref": "Environment"}, {"us-west-2-feature-lake/*"}]]},
          {"Fn::Join": ["-", [{"arn:aws:s3:::rokt"}, {"Ref": "Environment"}, {"Ref": "AWS::Region"}, {"optimus"}, {"Ref": "OptimusVersion"}, {"data"}]]},
          {"Fn::Join": ["-", [{"arn:aws:s3:::rokt"}, {"Ref": "Environment"}, {"Ref": "AWS::Region"}, {"optimus"}, {"Ref": "OptimusVersion"}, {"data/*"}]]}
        ]
      } ]
    }
  }
},

```

- Version is a date-based version that represents when the policy standard was published by AWS
- Statement is a list of permissions being granted
  - Effects states whether an action is allowed or denied
- Actions are namespace according to service, specifically load(put) and download(get)
  - Resource specifies a single or list of resources on which the actions may or may not be performed
- Conditions - under the conditions
  - Policies will merged and denies always win

# Policies

## Policy Evaluation Logic



- Attach Policy to the groups (Managed Policies)
  - AWSLambdaFullAccess
  - AmazonS3FullAccess
  - AmazonDynamoDBFullAccess
  - IAMReadOnlyAccess
  - IAMSelfManagedService
  - IAMSelfUserChangePassword
- Inline Policies that are embedded in the group - Example DenyDangeous
  - Policy generator (Allow/Deny)
  - AWS Service (AmazonDynamoDB)
  - Actions>DeleteTable)
  - Amazon Resource Name(ARN) - \*

# Policies

The screenshot shows the AWS IAM console's 'Create policy' page. The top navigation bar includes the AWS logo, 'Services', 'Resource Groups', and user information 'richard @ tmj'. The page title is 'Create policy' with two numbered steps (1 and 2). A description states: 'A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. [Learn more](#)'. Below this are tabs for 'Visual editor' and 'JSON', with an 'Import managed policy' link. The 'JSON' tab is active, showing a JSON policy document in a code editor. The document is as follows:

```
6   "Effect": "Allow",
7   "Action": [
8     "iam:DeleteAccessKey",
9     "iam:GetAccessKeyLastUsed",
10    "iam:UpdateAccessKey",
11    "iam:CreateAccessKey",
12    "iam:ListAccessKeys"
13  ],
14  "Resource": "arn:aws:iam::*:user/*"
15  }
16  ]
17 }
```

The bottom of the screenshot shows a video player interface with a progress bar, a search bar, and a footer with copyright information: '© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.' and links to 'Privacy Policy' and 'Terms of Use'.

Use access keys to make secure REST or HTTP Query protocol requests to AWS service APIs.

## Credentials

- Should not embed access keys in code
- Should not embed in environment variable
- Should not share with third parties, hundreds of enterprise users, millions of web/app users
- Use temporary credentials
  - delegate permissions to EC2 instance, AWS service, A user/elevate privileges
  - separate account, one you own, third party account

## Roles for EC2

- Allows AWS services to perform actions on your behalf. e.g., EC2, Lambda, etc
- For example, the application on EC2 (APP+SDK) needs to authenticate against the AWS API for accessing other services
- Apply a Role to the EC2 instance which provides a way to apply the permissions and retrieve temporary credentials
- Policy is applied to the role, which specifies the actions and resources allowed or denied.
- AWS STS, Security Token Service generates temporary credentials
- AWS SDK will automatically retrieve temporary credentials from the IMDS

## **Roles for Cross-Accounts Access**

- Roles (Admin account; Production account) -> trust relationship tells AWS that our production account is trusting, or delegating permission to the Admin account
- Policy (Allow assume role)
- Assuming a role requires a call to STS to retrieve temporary credentials
- Use temporary credentials are then used to access resources in the other account

## **Existing Users**

- Organizational Users -
  - LDAP
  - Microsoft Active Directory
  - Web/mobile application users: Facebook, Google

## **Federation Options**

- SAML AWS Management Console, CLI, and API access authenticating with enterprise credentials.
- AWS Management Console access authenticating with Active Directory username + password.
- Amazon Cognito API access for web/mobile app users authenticating with OpenID Connect (OIDC).
- AWS SSO - Single sign-on provides access to multiple AWS accounts and business applications.

## **First Steps**

- Enable AWS CloudTrail
- Create an admin user in IAM (with optional condition)
- Enable Multi-Factor Authentication on root account
- Enable Cost and Usage Report
- Log out of **root account**
- Log in with admin user
- Create additional users, groups, etc

## **IAM Best Practices**

- Master account (root) credentials
- Email address + password
- Protect at all costs
- Delete any existing access/secret keys
- Do not use for day-to-day operations
- Follow principle of least privilege
- Rotate long-term credentials (access keys/passwords)
- Enable Multi-Factor Authentication (MFA)