AWS Identity and Access Management (IAM)

SwissLife Workshop - 4.7.2022

Version: 1.0.2

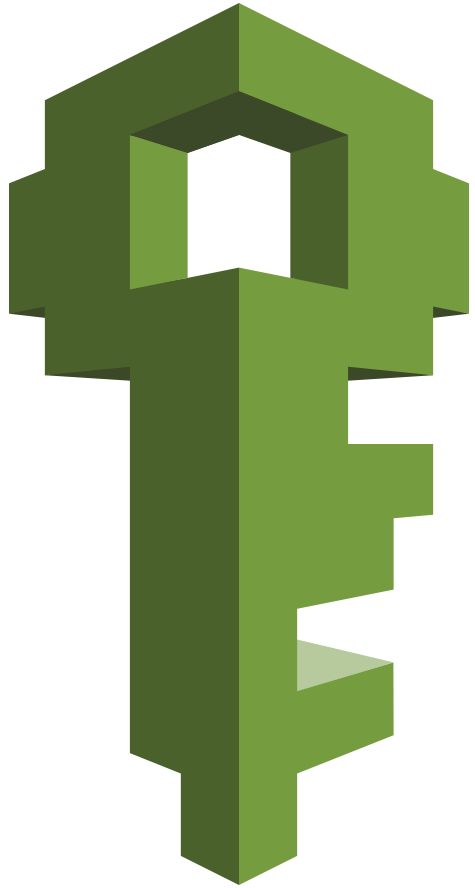
Agenda

- User / Groups
- Policies
- Roles
- Best Practice
- Terraform Examples

Goals of the Course

- getting a refresh / intro to AWS IAM
- Hands-On Time
- Best-Practices
- use Terrafrom to deploy some IAM-methods

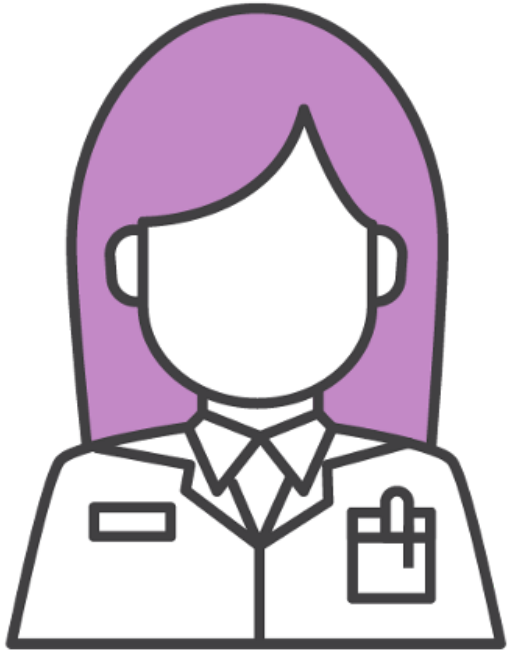
Users and Groups



The AWS Account Root User

This account has full access to all AWS services and resources.

- Billing information
- Personal data
- Your entire architecture and its components
- Tasks that require the Root User



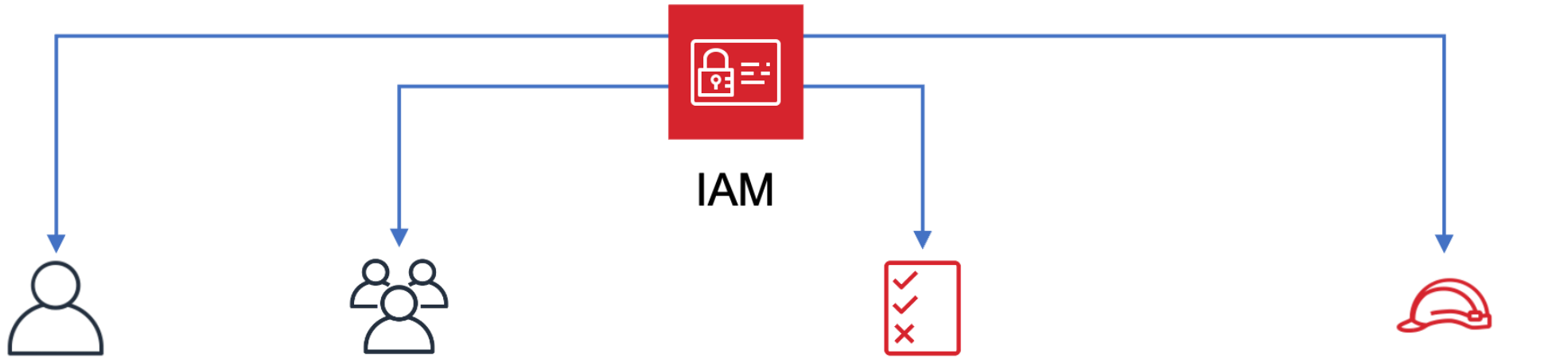
root user has extreme power w/o be limited

LAB

create admin user (console)

- create an iam user `admin`
- grant the `AdministratorAccess` policy
- give it `AWS Console Access`
- note the Login-URL
- log out and use Administrator

IAM terms and concepts



IAM Users

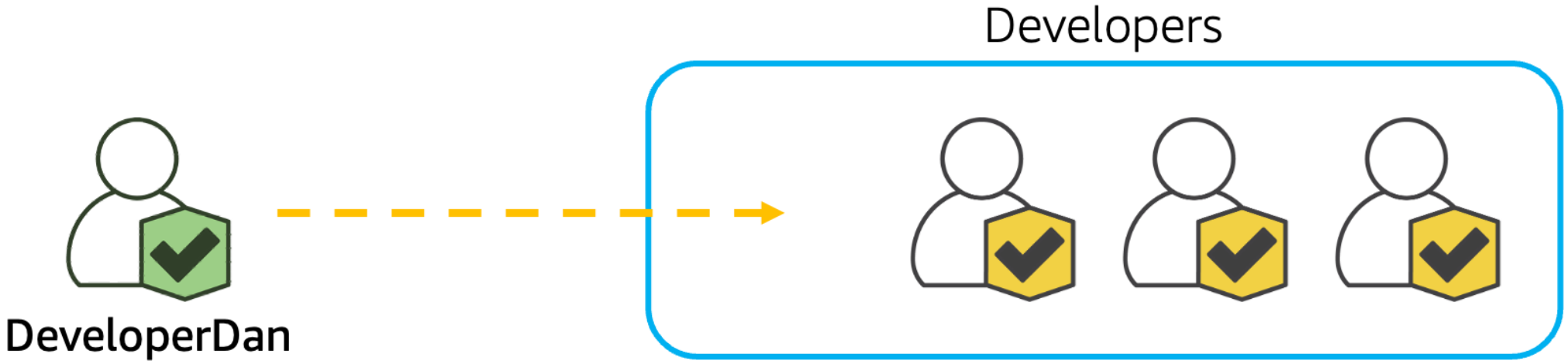
- IAM users are not separate AWS accounts; they are users within your account.
- Each user has their own credentials.
- IAM users are authorized to perform specific AWS actions based on their permissions (chapter permissions).

Origin of an IAM User

- There are ***NO*** default permissions.
- Access to the AWS Management Console or CLI must be explicitly granted.
- global Ressource

DeveloperDan

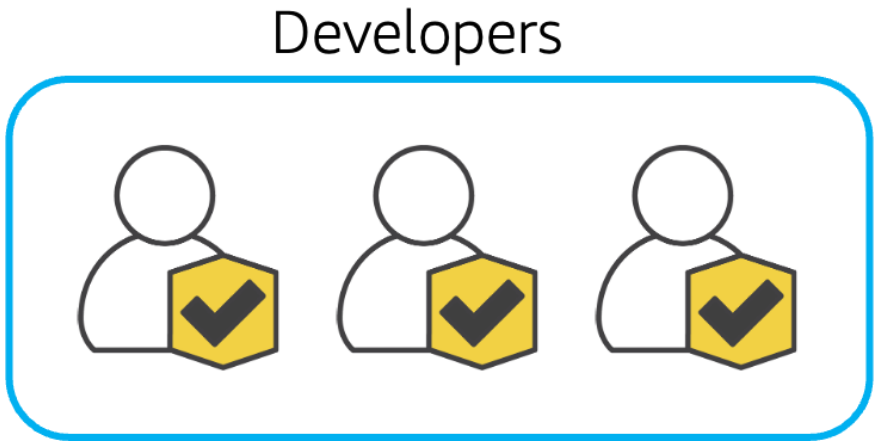
IAM Groups



IAM Groups

An IAM user group is a collection of IAM users. User groups let you specify permissions for multiple users.

- user groups can contain many users
- a user can belong to multiple user groups
- user groups ***can't*** be nested
- can contain only users
- there is no `default user` group



LAB

create a User and a Group

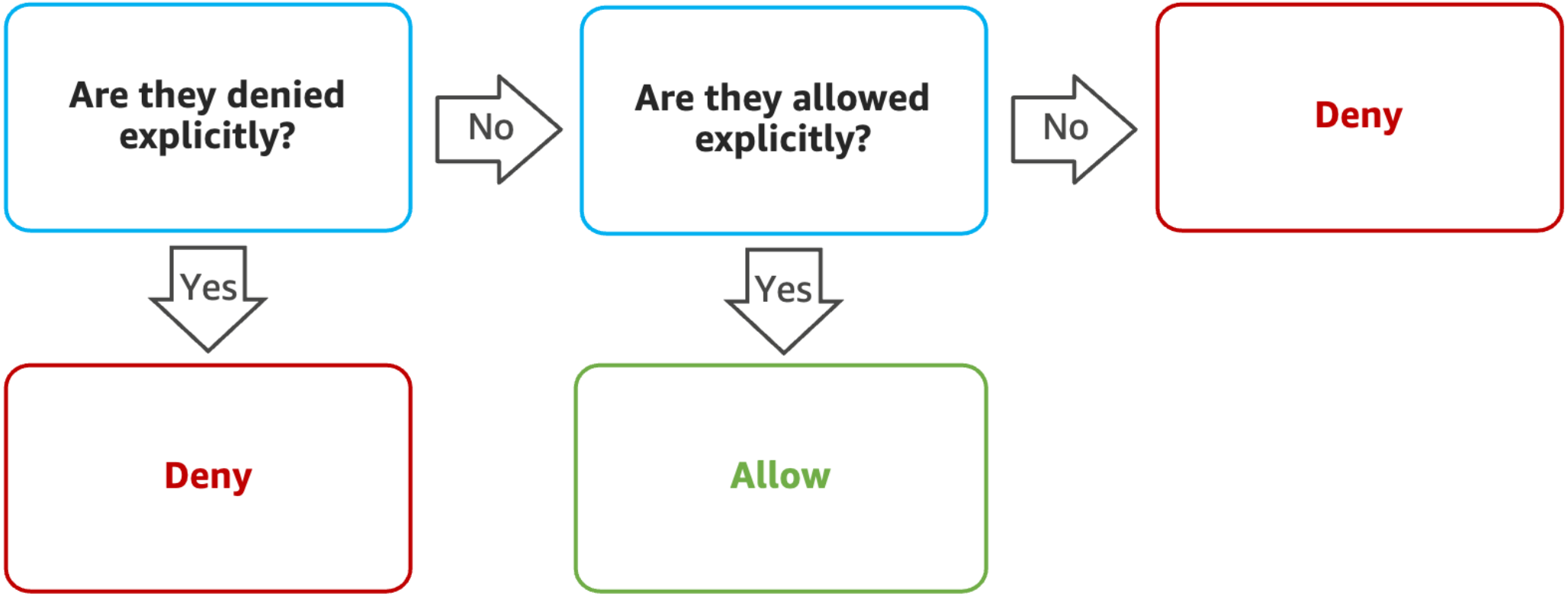
- create user `infrasally`
- grant her AWS Console access
- create group `infra-staff`
- attach `infrasally` to group `infra-staff`

Policies and Permission

Granting Permission

- a formal declaration of one or more permissions
- evaluated at the time of request
- IAM policies ONLY control access to AWS services

IAM permission evaluation

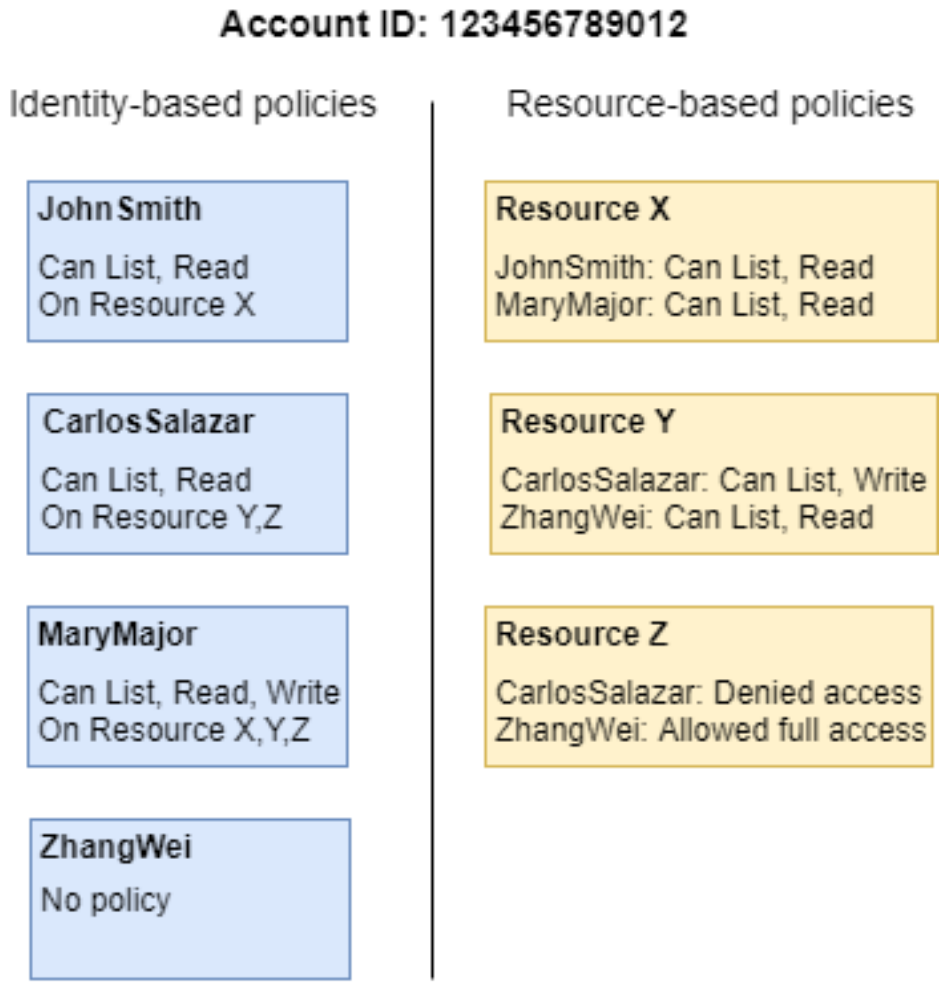


Identity-Based Policy

Identity-based policies are attached to an IAM `user`, `group`, or `role`.

Control:

- Actions performed
- Which resources
- What conditions are required



Syntax - Identity-Based Policy

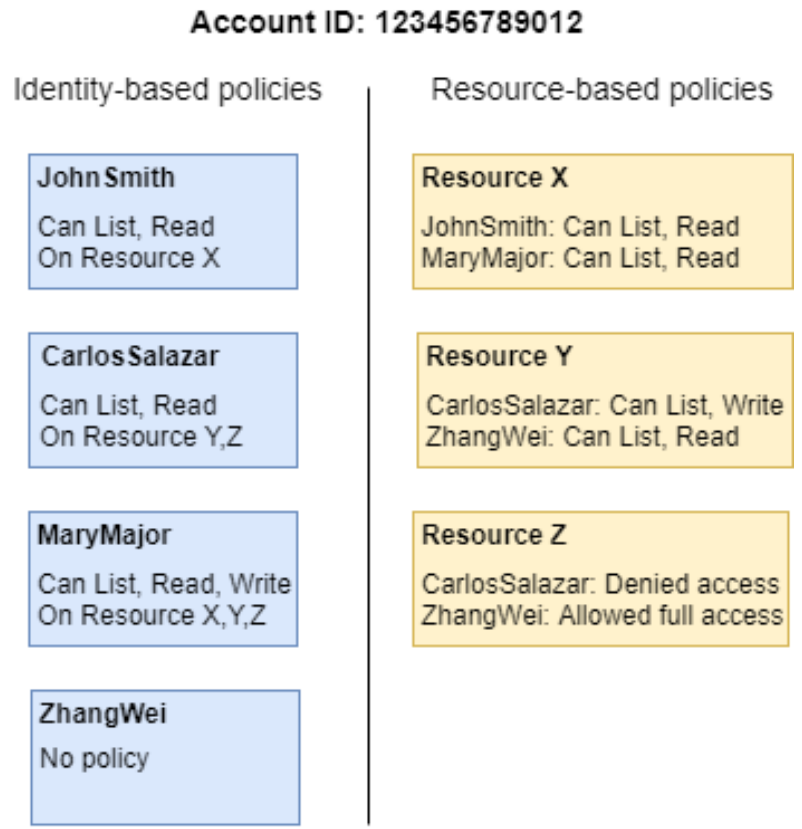
```
{
  "Version": "2012-10-17",
  "Id": "S3PolicyId1",
  "Statement": [
    {
      "Sid": "ListObjectsInBucket",
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::notes"
    },
    {
      "Sid": "AllObjectActions",
      "Effect": "Allow",
      "Action": "s3:*Object",
      "Resource":["arn:aws:s3:::notes/*",]
    }
  ]
}
```

Resource-Based Policies

Resource-based policies are attached to a `resource` (like S3, Glacier, KMS).

Control:

- Actions allowed by specific principal
- What conditions are required
- Are always inline policies
- No AWS managed resource-based policies



Syntax - Resource-based policies with Condition

```
{
  "Version": "2012-10-17",
  "Id": "S3PolicyId1",
  "Statement": [
    {
      "Sid": "IPAllow",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::notes",
        "arn:aws:s3:::notes/*"
      ],
      "Condition": {
        "NotIpAddress": {"aws:SourceIp": "54.240.143.0/24"}
      }
    }
  ]
}
```

LAB

add a policy to group

- create EC2-Instance `t2.micro` as admin
- add `AmazonEC2ReadOnlyAccess` to group infra-staff
- login as `infrasally` and check EC2

Policy with Tag-Condition

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "ec2:*",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceTag/env": "prod"
        }
      }
    }
  ]
}
```

Is there something missing - a better Version?

LAB

create a policy with tags

- create an iam-policy `ec2-admin-non-prod`
- allow all EC2-Actions on Systems with the Tag `env` and any Value that is NOT `prod`
- attach the policy to `infra-sally`

Policy with Tag-Condition

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:*",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "ec2:Describe*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": [
        "ec2:CreateTags",
        "ec2:DeleteTags"
      ],
      "Resource": "*"
    }
  ]
}
```

Roles

Role

A role lets you define a set of permissions to access the resources that a user or service needs.

- The permissions are not attached to an IAM user or group.
- The permissions are attached to a role and the role is assumed by the user or the service.

Role Use cases:

- Provide AWS resources with access to AWS services
- Provide access to externally authenticated users
- Provide access to third parties
- Switch roles to access resources in:
    - Your AWS account
    - Any other AWS account (cross-account access)

Assume-Role

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::589905756388:user/infrasally"
            },
            "Action": "sts:AssumeRole",
            "Condition": {}
        }
    ]
}
```

LAB

create a role & attach

- create a policy `ec2-admin`
  with all actions on ec2.
- create an `Assume role` for `infrasally`
- test everything
- optional challage:
  do it for the group `infra-staff`

Step 1. - create a Policy that Allow to Assume-Role

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AssumeRoleForRoleEC2Administration",
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Resource": [
                "arn:aws:iam::589905756388:role/Role-EC2-Administration"
            ]
        }
    ]
}
```

Stept 2 - attach the Policy to the Group

Best Practice

Best Practices

- Lock away your AWS account root user access keys
- Use roles to delegate permissions (assume role)
- Grant least privilege (only permissions required to perform a task)
- Use customer managed policies instead of inline policies
- Configure a strong password policy for your users
- Enable MFA
- Use policy conditions for extra security
- Monitor activity in your AWS account

Terraform Examples

Terraform IAM Ressources

User Ressource

```
resource "aws_iam_user" "audit-user" {
  name = "auditjack"
}
```

User Login

```
resource "aws_iam_user_login_profile" "devdan-profile" {
  user                    = aws_iam_user.devdan.name
  password_length         = 20
  password_reset_required = false
}
```

Terraform IAM Ressources

Group Ressource

```
resource "aws_iam_group" "audit-group" {
  name = "audit-staff"
}
```

Group Membership

```
resource "aws_iam_group_membership" "devdan-membership" {
  name  = "tf-testing-group-membership"
  group = aws_iam_group.audit-group.name
  users = [
    aws_iam_user.audit-user.name,
  ]
}
```

LAB

create user & group

- create a user `devdan`
- create a group `dev-staff`
- attach the user to the group

Terraform IAM Ressources

Policy Ressource

```
resource "aws_iam_policy" "s3_admin_policy" {
  name        = "s3_admin_policy"
  description = "Allow all S3 actions"
  policy = <POLICY-STRING>
}
```

IAM Policy

```
resource "aws_iam_user_policy_attachment" "attach_s3_policy" {
  user       = aws_iam_user.devdan.name
  policy_arn = aws_iam_policy.s3_admin_policy.arn
}
```

policy loading options

- `policy = jsonencode({..})`
- `policy = file()`
- `policy = templatefile()`
- or just as String

```
policy = <<EOT
{
  ...Lines ommited...
}
EOT
```

Generate a policy through a Datasource

```
data "aws_iam_policy_document" "generated-audit-policy" {
  statement {
    actions   = ["s3:ListAllMyBuckets"]
    resources = ["arn:aws:s3:::*"]
    effect    = "Allow"
  }
  statement {
    actions   = ["s3:*"]
    resources = [var.bucket_arn]
    effect    = "Allow"
  }
}
```

use it with .json to render:

```
policy = data.aws_iam_policy_document.generated-audit-policy.json
```

LAB

create policies & attach

- create a policy `s3_admin` with all actions on s3.
- attach the policy to the user `devdan`
- add the policy `AmazonEC2ReadOnlyAccess` to the group `dev-staff` (Hint: Datasource)
- test everything

```
resource "aws_iam_policy" "policy" {
  name        = "${random_pet.pet_name.id}-policy"
  description = "My test policy"

  policy = <<EOT
{
  "Version": "2012-10-17",
  "Statement": [
  ...Lines ommited...
  ]
}
EOT
}
```

LAB

create assume-role for ec2

- create a private S3 bucket `sw-test-XX`
- create a policy `sw-test-read` with all actions on s3.
- start an `t2.micro instance`
- add the role `ec2-s3-access` to the instance-profile
- test access via `aws s3 ls` everything

IAM-Live (3rd-Party)

- Install IAM-Live
- Terminal #2 - `$ iamlive --mode proxy`
- Terminal #1 - add Environment-Variables

```
export AWS_CA_BUNDLE=~/.iamlive/ca.pem
export HTTP_PROXY=http://127.0.0.1:10080
export HTTPS_PROXY=http://127.0.0.1:10080
export AWS_ACCESS_KEY_ID="Real Key"
export AWS_SECRET_ACCESS_KEY="Real Key"
```

- Terminal #1 - `terraform apply`