Lesson 5 IAM

Michael Yang

AWS Account

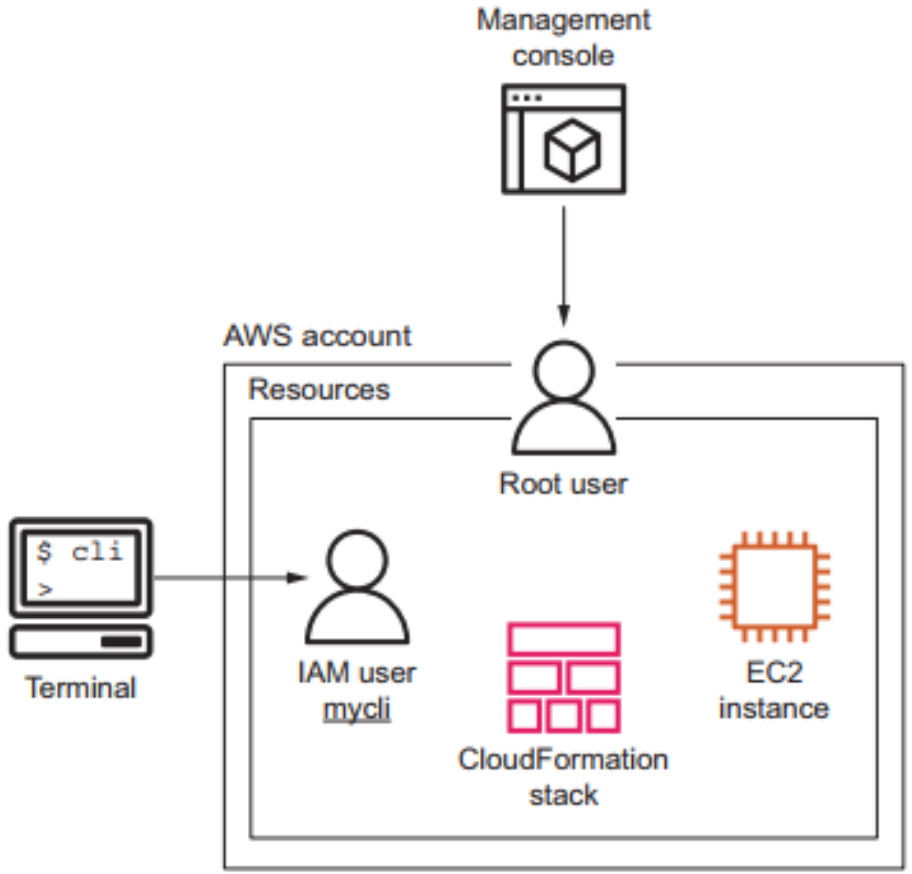
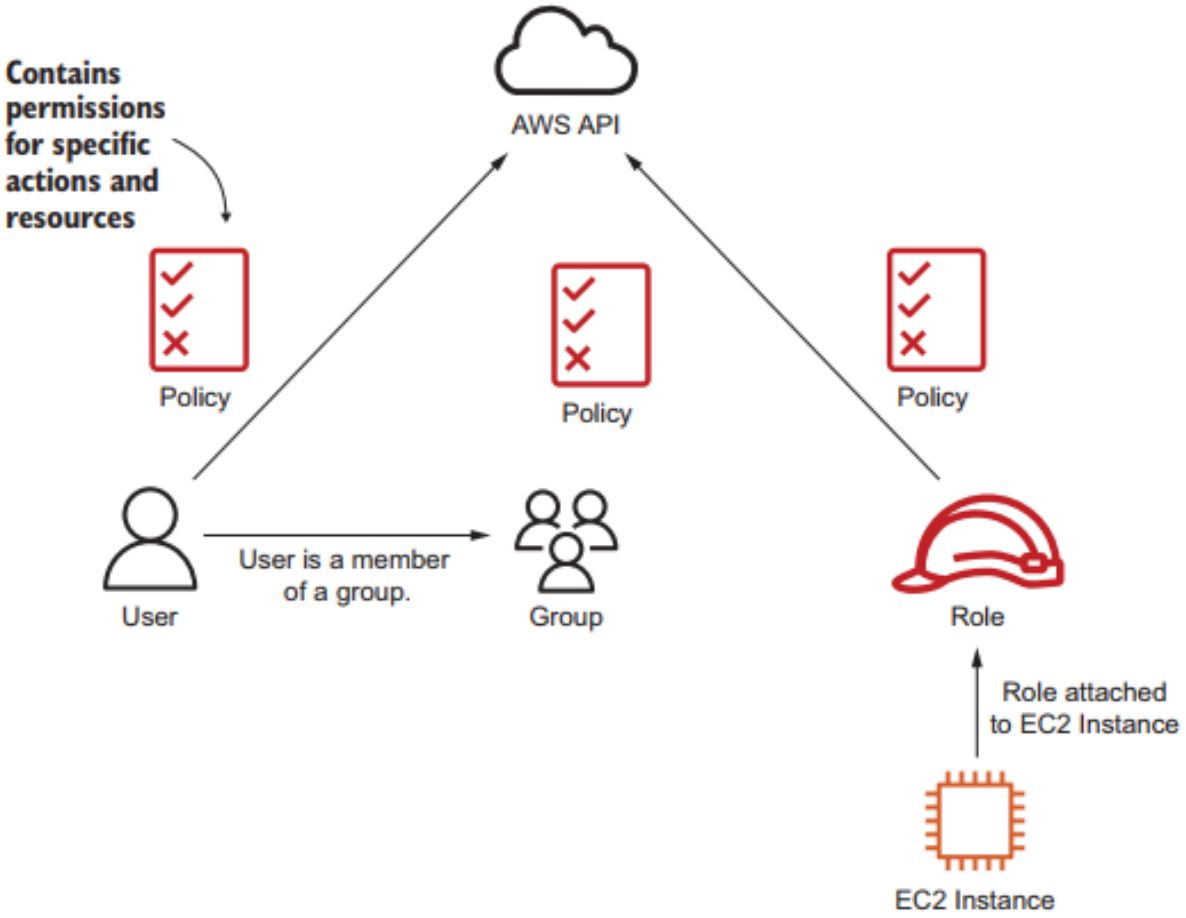
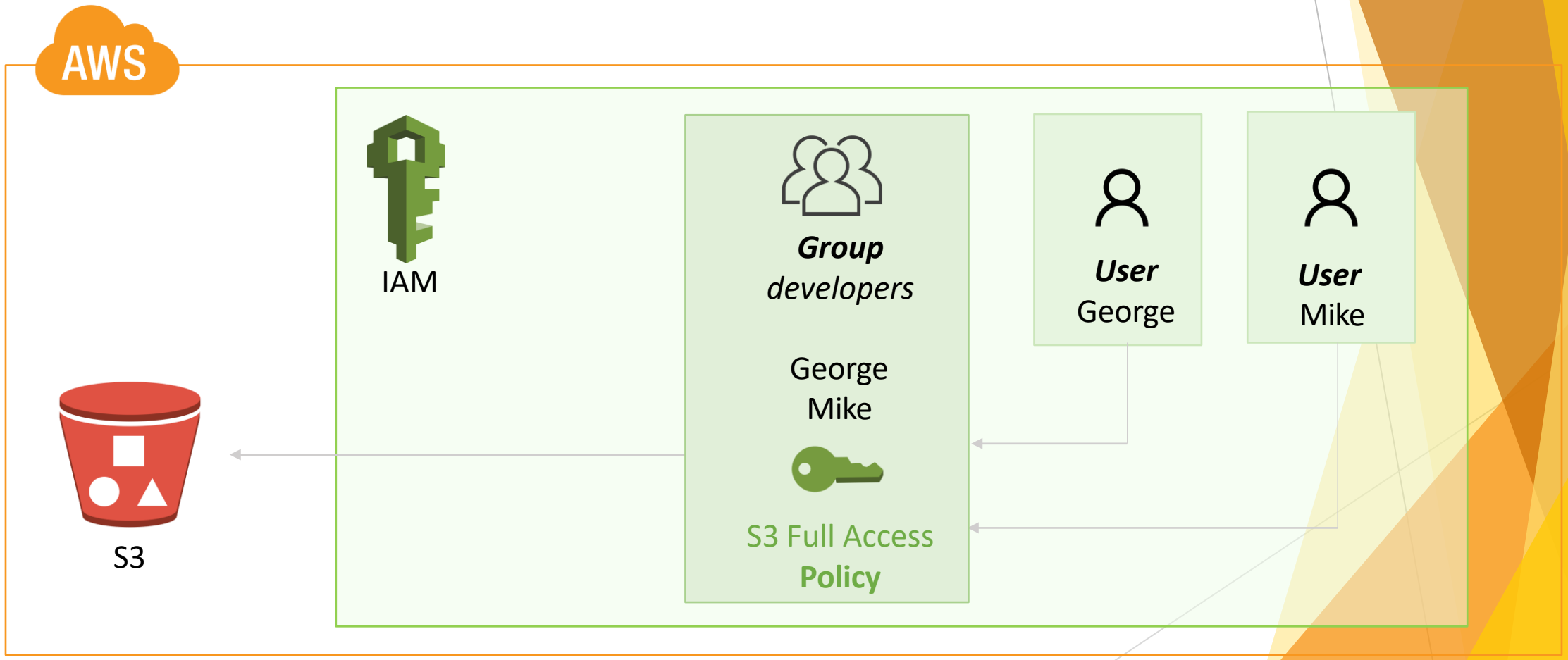


Figure 5.4 An AWS account contains all the AWS resources and comes with an AWS account root user by default.

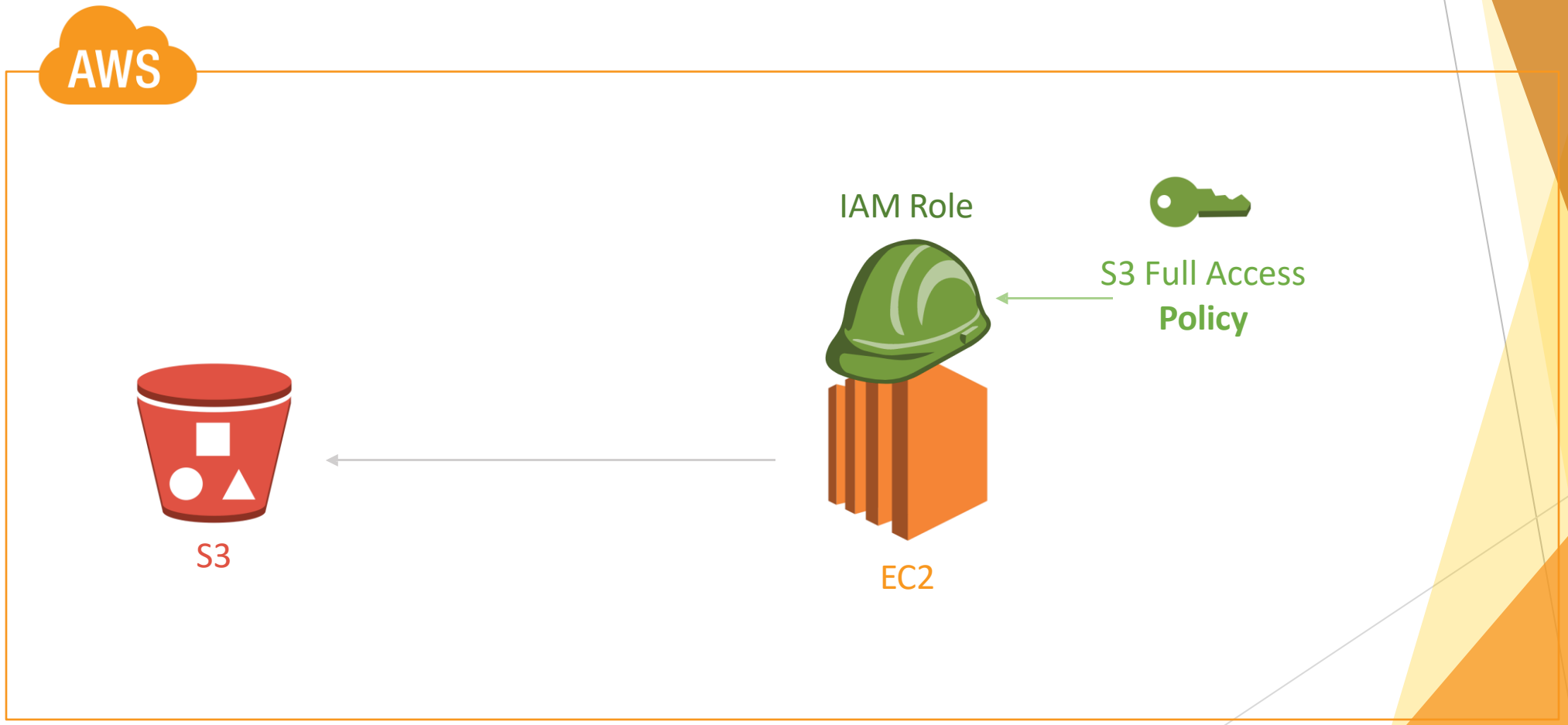
IAM Core Concepts



AWS IAM Group



AWS IAM Role



IAM user vs IAM role

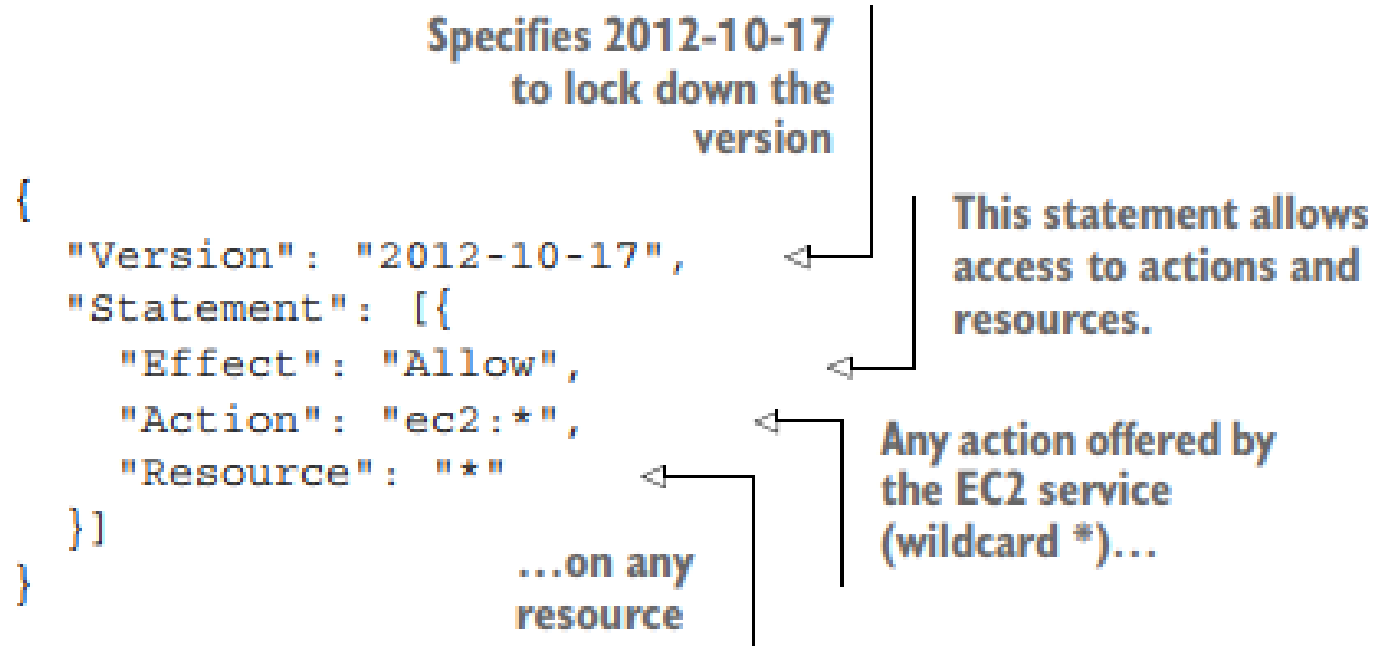
	AWS account root user	IAM user	IAM role
Can have a password (needed to log in to the AWS Management Console)	Always	Yes	No
Can have access keys (needed to send requests to the AWS API (e.g., for CLI or SDK)	Yes (not recommended)	Yes	No
Can belong to a group	No	Yes	No
Can be associated with an EC2 instance, ECS container, Lambda function	No	No	Yes

IAM Policy

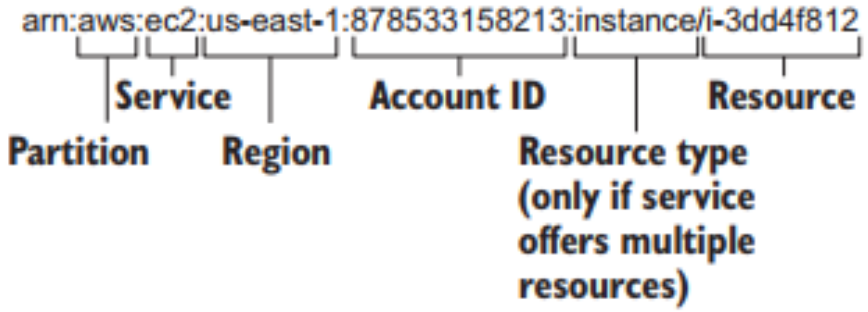
- IAM policies come in two types. *Identity policies* are attached to users, groups, or roles. *Resource policies* are attached to resources. Very few resource types support resource policies. One common example is the S3 bucket policy attached to S3 buckets.
  If a policy contains the property `Principal`, it is a resource policy. The `Principal` defines who is allowed to perform the action. Keep in mind that the principal can be set to public.

IAM Policy

- The following is an identity policy, it has one statement that allows every action for the EC2 service, for all resources:



ARN



- If you know your account ID, you can use ARNs to allow access to specific resources of a service like this:

```
{
"Version": "2012-10-17",
"Statement": [{
"Effect": "Allow",
"Action": "ec2:TerminateInstances",
"Resource": "arn:aws:ec2:us-east-
1:111111111111:instance/i-
0b5c991e026104db9"
}]
}
```

IAM Policy

- If you have multiple statements that apply to the same action, `Deny` overrides `Allow`. The following identity policy allows all EC2 actions except terminating EC2 instances

```
{
    "Version": "2012-10-17",
    "Statement": [{
      "Effect": "Allow",
      "Action": "ec2:*",
      "Resource": "*"
    }, {
      "Effect": "Deny",
      "Action": "ec2:TerminateInstances",
      "Resource": "*"
    }]
}
```

Action is denied.

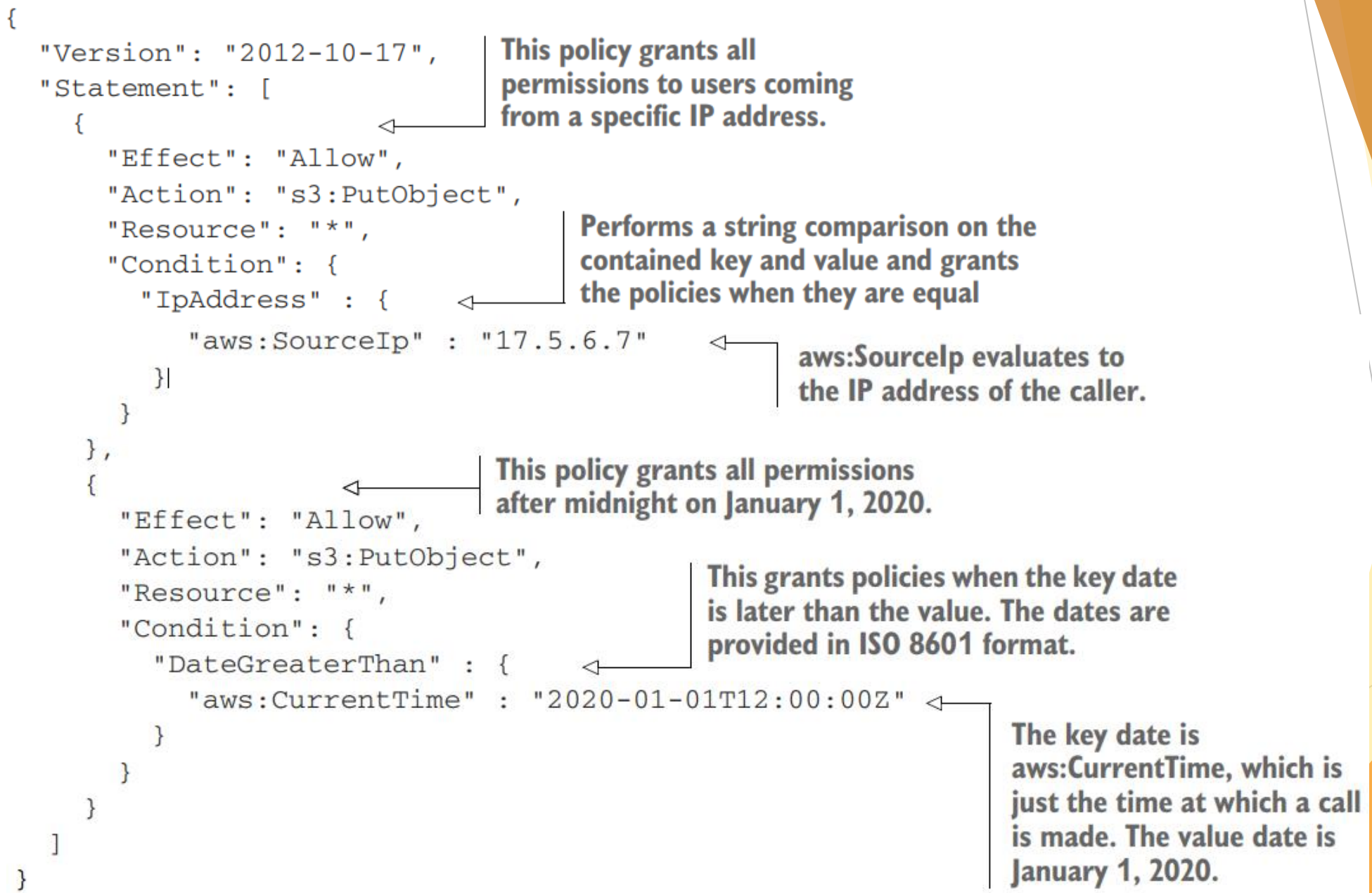
Terminate EC2 instances.

IAM policy

The Condition element (or Condition block) lets you specify conditions for when a policy is in effect.
In the Condition element, you build expressions in which you use condition operators (equal, less than, etc.) to match the condition keys and values in the policy against keys and values in the request context.
Learn more: IAM JSON policy elements: Condition

```
"Condition" : { "{condition-operator}" : { "{condition-key}" : "{condition-value}" }}
```

```
"Condition" : { "StringEqualsIgnoreCase" : { "aws:username" : "johndoe" }}
```


{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": "s3:PutObject",
"Resource": "*",
"Condition": {
"IpAddress" : {
"aws:SourceIp" : "17.5.6.7"
}
}
},
{
"Effect": "Allow",
"Action": "s3:PutObject",
"Resource": "*",
"Condition": {
"DateGreaterThan" : {
"aws:CurrentTime" : "2020-01-01T12:00:00Z"
}
}
}
]
}
This policy grants all permissions to users coming from a specific IP address.
Performs a string comparison on the contained key and value and grants the policies when they are equal
aws:SourceIp evaluates to the IP address of the caller.
This policy grants all permissions after midnight on January 1, 2020.
This grants policies when the key date is later than the value. The dates are provided in ISO 8601 format.
The key date is aws:CurrentTime, which is just the time at which a call is made. The value date is January 1, 2020.

Explicit Deny wins

- The following identity policy denies all EC2 actions. The `ec2:TerminateInstances` statement isn't crucial, because `Deny` overrides `Allow`. When you deny an action, you can't allow that action with another statement:

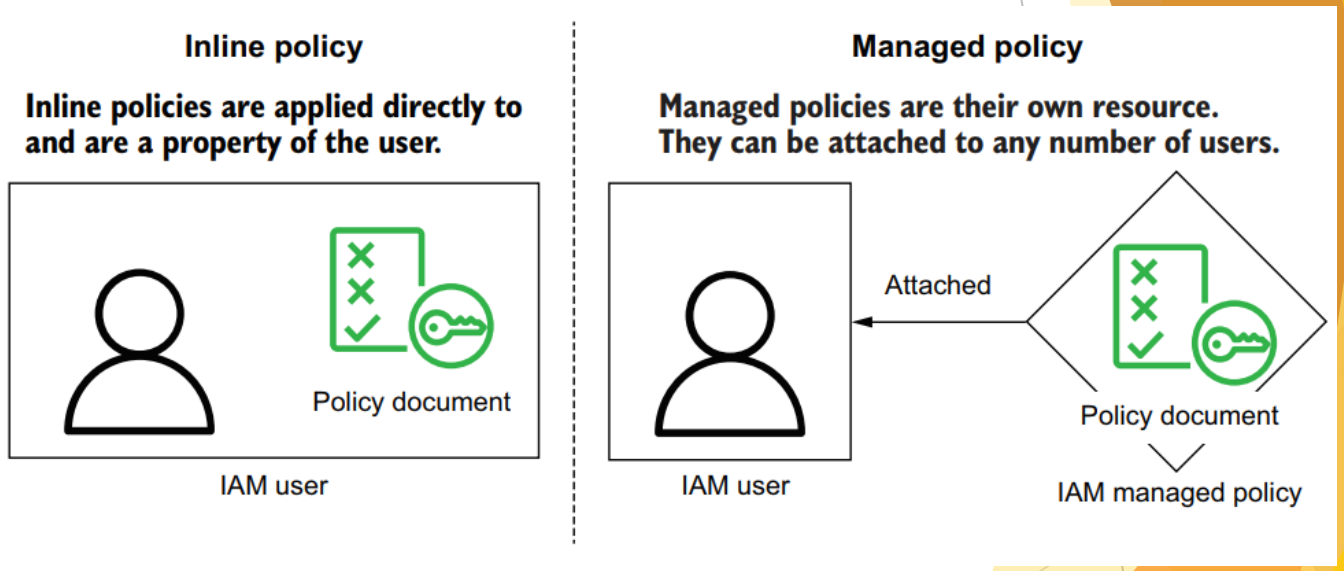
```
{
   "Version": "2012-10-17",
   "Statement": [{
     "Effect": "Deny",
     "Action": "ec2:*",
     "Resource": "*"
   }, {
     "Effect": "Allow",
     "Action": "ec2:TerminateInstances",
     "Resource": "*"
   }]
}
```

Denies every EC2 action

Allow isn't crucial; Deny overrides Allow.

Two Identity Polic

- The following two types of identity policies exist:
Managed policy—If you want to create identity policies that can be reused in your account, a managed policy is what you're looking for. There are two types of managed policies:
AWS managed policy—An identity policy maintained by AWS. There are identity policies that grant admin rights, read-only rights, and so on.
Customer managed—An identity policy maintained by you. It could be an identity policy that represents the roles in your organization, for example.
Inline policy—An identity policy that belongs to a certain IAM role, user, or group. An inline identity policy can't exist without the IAM role, user, or group that it belongs to.



Managed Policy

```
$ aws iam create-policy \
   --policy-name SamplePolicy \
   --policy-document file://policy.json

$ aws iam attach-user-policy \
   --user-name Alice
   --policy-arn arn:aws:iam::123456789012:policy/SamplePolicy
```

Creates a new managed policy

Attaches a managed policy to a user

This is the ARN of the managed policy we just created. It will be in the response of the create-policy call.

Inline Policy

```
$ aws iam create-user \
    --user-name Bob

$ aws iam put-user-policy \
    --user-name Bob \
    --policy-name SampleInlinePolicy \
    --policy-document file://policy.json
```

Puts the policy on the user Bob, created before

Names the policy for later reference

This is a local file that contains the policy document you want to use.

Resource Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/Alice"
      },
      "Resource": "arn:aws:s3:::my-sample-bucket/*"
    }
  ]
}
```

This is the Principal block of the resource policy. It determines which entities the policy refers to.

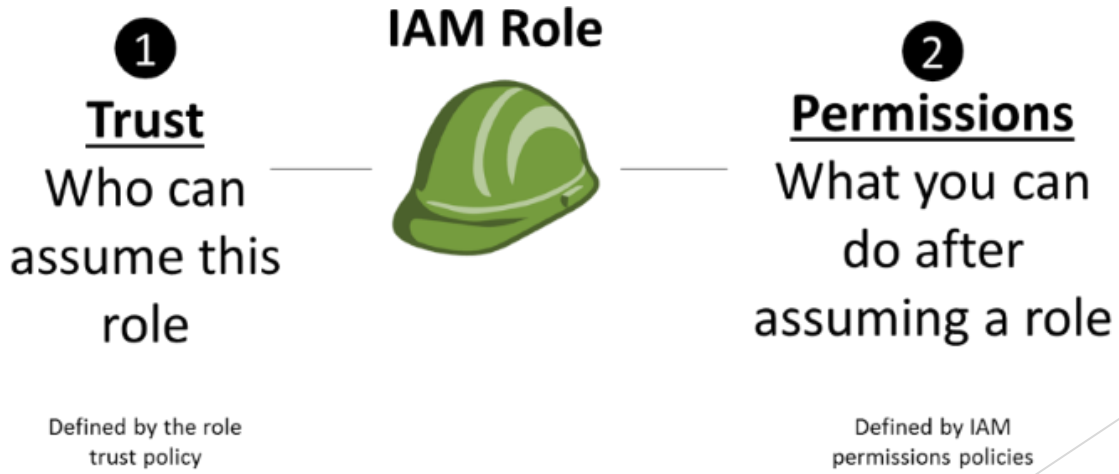
The "AWS": ARN notation is used to reference an IAM entity—in this case, a user.

The Resource block is required in bucket policies and, in this instance, refers to all objects in this bucket.

IAM role Under the hood

IAM role is similar to IAM user but you assign that to a resource, not a developer. There is a trust policy that defines which service can assume (use) that role. An IAM user and role have one thing in common, permission policies.

When the service in the trust policy assumes the role, AWS STS (Security Token Service) returns **temporary** tokens (access key id, secret access key, and **session token**), and those tokens are rotated automatically.



AWS Security Token Service (AWS STS)

Under the hood, tokens are generated and used to access AWS services. STS is a web service that enables you to request **temporary** credentials for IAM role. The example is AWS Academy, you receive a temporary token.

The temporary credentials consist of:

1. **Access key ID** - Access keys are long-term credentials for an IAM user. Like username.
2. **Secret access key** - Like password.
3. **Session token** - Validates temporary credentials.
4. **Duration** - defines how long the temporary credentials lasts. Most cases 12 hours. 15-min is min. You will not see these in the permanent tokens for users.

STS - AssumeRole

The underlying service makes an implicit “AssumeRole” call to STS on behalf of the resource and fetch the temporary tokens. It all happens under the hood.

For instance, when fetching images from S3 in EC2, EC2 makes the AssumeRole call to STS, then receives the token and provides the token to S3. You don’t have to manually rotate it. The rotation is done by AWS.

IAM User vs Role

IAM User	**IAM Role**
IAM entity assigned to a **person**.	IAM entity assigned to a **service**. It has a trust policy that specifies what services can use the role.
Tokens are permanent. It is on you to rotate that regularly.	**Tokens are temporary**. Tokens are generated by AWS STS. It has an extra token “aws_session_token”.

Exercise

```
[ec2-user@ip-172-31-17-250 ~]$ aws --version
aws-cli/1.18.147 Python/2.7.18 Linux/4.14.225-169.362.amzn2.x86_64 botocore/1.18.6
[ec2-user@ip-172-31-17-250 ~]$ aws iam list-users
Unable to locate credentials. You can configure credentials by running "aws configure".
[ec2-user@ip-172-31-17-250 ~]$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [None]: ^C
[ec2-user@ip-172-31-17-250 ~]$ aws iam list-users
{
    "Users": [
        {
            "UserName": "stephane",
            "PasswordLastUsed": "2021-04-19T20:41:59Z",
            "CreateDate": "2021-04-19T20:38:18Z",
            "UserId": "AIDATCOXNUAWA5KFV6MM7",
            "Path": "/",
            "Arn": "arn:aws:iam::211442049068:user/stephane"
        }
    ]
}
[ec2-user@ip-172-31-17-250 ~]$
```

IAM Summary

IAM is about:

1. allow/deny
2. what actions?
3. on which resources?
4. who?
    a. principle propriety (resource-based)
    b. IAM user or role (identity-based)
5. condition (optional)