AWS re:Invent

SEC316-R

Access control confidence: Grant the right access to the right things

Brigid Johnson
Senior Manager, AWS Identity
Amazon Web Services

AWS re:Invent

aws

Access control confidence

. . . it's a journey

Where you start

Business to innovate

Agility to move fast

Give builders freedom

Where you are going

Prevent dangerous actions

Accountable security posture

Least privilege

Let's go!

Agenda

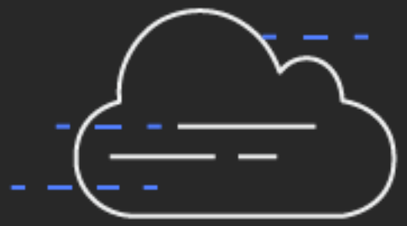
- Review permissions in AWS
- Understand a permission model that fosters growth
- Set yourself up for success with permission guardrails — Demo!
- Rely on attributes for fine-grained permission at scale — Demo!
- Use analytics to rein in permissions — Demo!

Review permissions in AWS

Two parts to permissions

Your job: Specification

Define which entities are allowed to perform which actions on specific resources and under which conditions



AWS's job: Enforcement

For each request, the service or application *evaluates* the permissions that you defined to allow or deny access

IAM policies enable granular access controls

```
{
 "Statement":[{
  "Effect":"effect",
  "Principal":"principal",
  "Action":"action",
  "Resource":"arn",
  "Condition":{
   "condition":{
    "key":"value" }
   }
  }
 ]
}
```

Principal: The entity that is allowed or denied access

"Principal":"AWS":"arn:aws:iam::123456789012:user/username"

Action: Type of access that is allowed or denied

"Action":"s3:GetObject"

Resource: The Amazon resource(s) the action will act on

"Resource":"arn:aws:sqs:us-west-2:123456789012:queue1"

Condition: The conditions that are valid under the access defined

"StringEqualsIfExists": {"aws:RequestTag/project": [“Pickles“]}

It is all about matching

Context of your request

The unique components of each AWS request

Matching

Allowed

Denied

Your defined policies

The policies you define on identities, resources, and organizations

Policy types – how they work together

All access requests start with DENY

If using service control policies	→	SCP must allow
If using permission boundaries	→	Permission boundary must allow
If same account access	→	Identity or resource policy must allow
If direct cross account access	→	Both the identity AND resource policy must allow
If using session policy	→	Session and identity policy must allow

Understand a permission framework that fosters growth

AWS re:Invent

aws

A term you've probably heard: Least privilege

The right access

To the right things

At the right time

To do their job

And nothing more

Least privilege is a journey

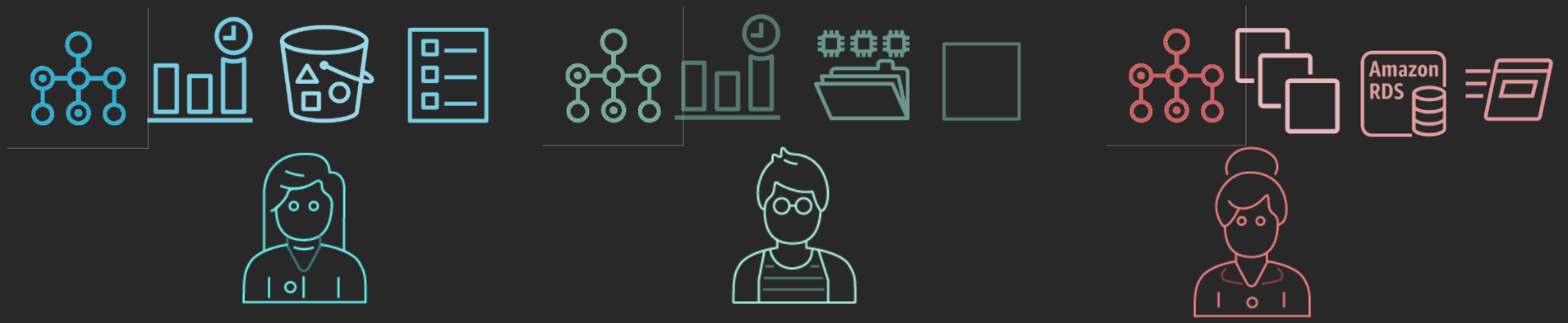
Here is how you make it a confident one

Permission guardrails

Powerful actions

Critical resources

Attribute-based access control



Rein in permissions

Set yourself up for success with permission guardrails

AWS re:Invent

aws

AWS tools to apply permission guardrails

VPC private link and endpoint policies

Require that traffic stays within your VPC

AWS Organizations service control policies (SCPs)

Permission guardrails to restrict access for principals across accounts

AWS Identity and Access Management (IAM) permission boundaries

Enable developers to create and manage permissions, while controlling the maximum permissions they grant

Service control policies as permission guardrails

Establish controls that all IAM principals (users and roles) adhere to across an account, organizational unit, or organization

What you can do

- Restrict access to specific AWS Regions
- Prevent your IAM principals from deleting common resources
- Restrict service actions for all IAM entities except a specific role

Pro tip: Push restrictions common among accounts up into SCPs

Demonstration characters

Central security team

Mission

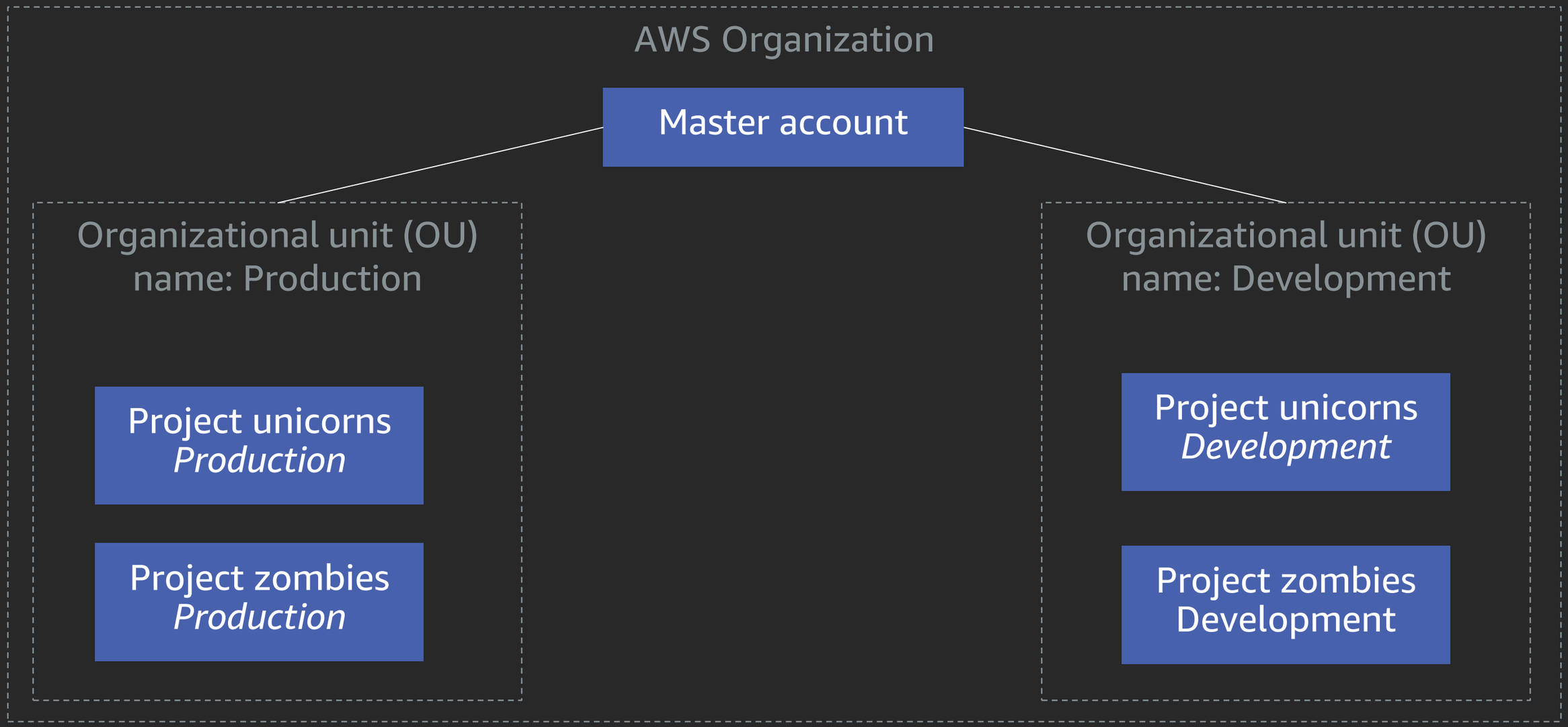
Access control confidence, while enabling developers to build.

Development

Mission

Build code!

Organize and govern accounts with AWS Organizations



Permission guardrail challenges

1. Restrict access to only the east and west US regions across your AWS organization

2. Restrict powerful service actions for all IAM entities except a specific role

Policy for permission guardrails – restrict regions

```
"Effect": "Deny",
"Action": [
    "codecommit:*",
    "codebuild:*",
    "s3:*",
    "secretsmanager:*",
    "elasticbeanstalk:*],
"Resource": ["*"],
"Condition": {
    "StringNotEquals": {
            "aws:RequestedRegion": ["us-west-2","us-west-1","us-east-1","us-east-2"]
     }}}]}
```

Deny these services,
when not in these regions

Pro tip: Use AWS RequestedRegion condition key

Policy for permission guardrails – powerful actions

```
"Effect": "Deny",
"Action": [
        "ec2:AssociateRouteTable",
        "ec2:CreateRoute",
        "ec2:CreateRouteTable",
        "ec2:DisassociateRouteTable",
        "ec2:ReplaceRoute",
        "ec2:DeleteRoute",
        "ec2:DeleteRouteTable",
        "ec2:DetachInternetGateway",
        "ec2:AttachInternetGateway",
        "ec2:CreateInternetGateway",
        "ec2:DeleteInternetGateway"
                ],
"Resource": "*",
"Condition": {
        "StringNotLike": {
                "aws:PrincipalARN": "arn:aws:iam::*:role/network-admin*"
                }
        }
```

Deny these actions, unless you are this role

Pro tip: Use AWS PrincipalArn condition key

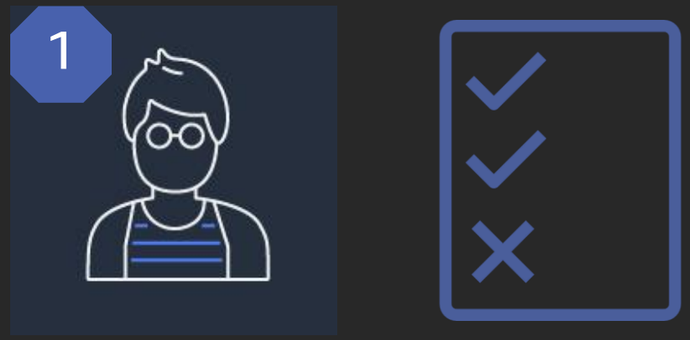
Permission guardrails: Demo

Set-up

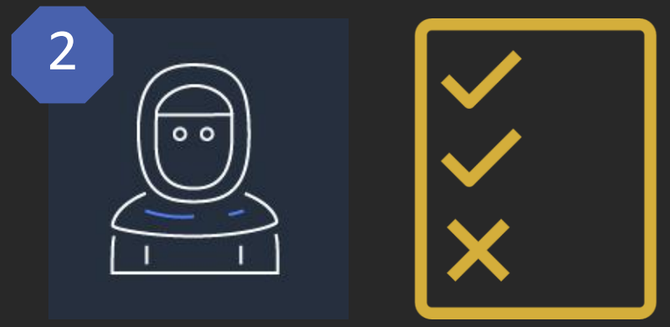
1. Create SCPs to restrict regions and powerful actions
2. Attach SCPs to the root

Let's test it out using an administrator:	Allowed or denied?
Create a resource in an approved Region	Allowed
Create a resource in an unapproved Region	Denied
Use network role to modify/create a critical resource	Allowed
Use developer role to modify a critical resource	Denied

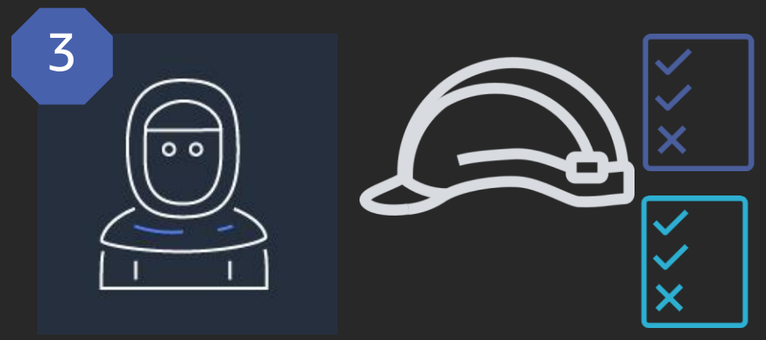
Permission boundary workflows



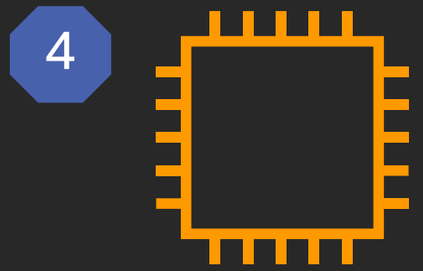
Admin creates maximum permissions



Admin allows developers to create managed policies, create roles with boundaries, attach policies, and pass specific roles



Developer creates role with maximum permissions and specific permissions



Developer passes the role to application resources

Permission boundaries

Enable developers to create and manage IAM roles but control the maximum permissions they can grant

What you can do

- Enable developer to create roles without escalating their access
- Require developers to create roles with a boundary

Pro tip: Require roles and managed policies start with a namespace

Permission boundaries challenge

Enable your developers to create

IAM roles to pass to Amazon

Elastic Compute Cloud (Amazon

EC2) and AWS Lambda, but

ensure they cannot exceed their

own permissions

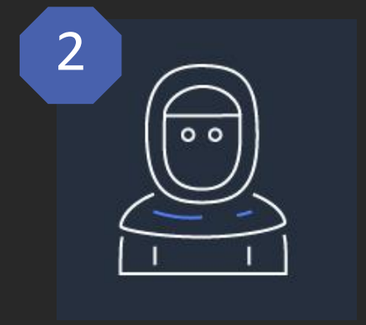
Admin creates maximum permissions


```
{
"Effect": "Allow",
"Action": [
        "secretsmanager:GetRandomPassword",
        "secretsmanager:GetResourcePolicy",
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret",
        "secretsmanager:ListSecrets",
        "secretsmanager:ListSecretVersionIds"],
"Resource": "*"
},
{
   "Effect": "Allow",
   "Action": "s3:GetObject",
   "Resource": "arn:aws:s3:::pickles-*/*"
}
```

Maximum for AWS Secrets Manager

Maximum for Amazon Simple Storage Service (Amazon S3)

Admin allows creation and management of roles

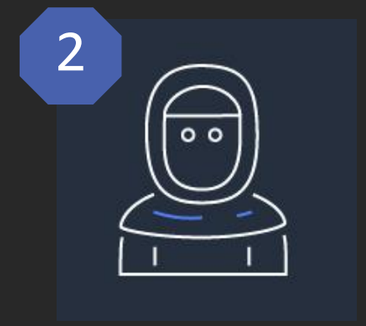


1. Create managed policies

```
"Effect": "Allow",
"Action": [
        "iam:CreatePolicy",
        "iam:CreatePolicyVersion",
        "iam:DeletePolicyVersion"
],
"Resource": "arn:aws:iam::432807222178:policy/${aws:PrincipalTag/project}-*"
```

Allow create policy, but require starts with your project

Admin allows creation and management of roles



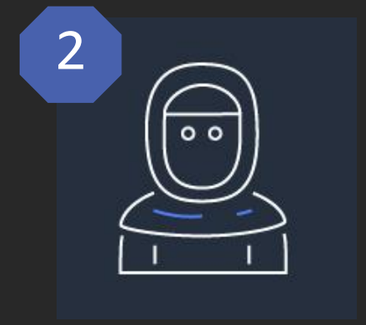
2. Create roles and attach policies with specific boundary

```
"Effect": "Allow",
"Action": [
        "iam:DetachRolePolicy",
        "iam:CreateRole",
        "iam:AttachRolePolicy"
],
"Resource": "arn:aws:iam::432807222178:role/${aws:PrincipalTag/project}-*",
"Condition": {
        "StringEquals": {
                "iam:PermissionsBoundary": "arn:aws:iam::432807222178:policy/read-content-
boundary"
        }
}
```

Require boundary

Pro tip: Use the PermissionsBoundary condition key

Admin allows creation and management of roles



3. Pass roles they created

```
{
        "Effect": "Allow",
        "Action": [
                "iam:passrole"],
        "Resource": "arn:aws:iam::432807222178:role/${aws:PrincipalTag/project}-*"
}
```

Rely on attributes for fine-grained permission at scale

AWS re:Invent

aws

Examples of attribute-based permissions

- Grant developers read and write access to their project resources
- Require developers to assign their project to new resources
- Grant developers read access to resources that are common to their team
- Manage only the resources that you own

A scalable permissions model based on attributes

Workforce users

Permissions

Resources

AWS tools to apply attribute-based access control (ABAC)

AWS IAM principal tags

New! AWS IAM session tags

Tag entities and sessions with access control attributes

Tags on AWS resources

Tag resources with access control attributes

AWS IAM policies

Control access based on tags

New! Tag policies with AWS Organizations

Standardizing tag names, values, and capitalization. Control allowable values. Investigate differences.

Session tags for ABAC

New!

Identity provider is the source of truth

Pass in user attributes as tags specific to each federated AWS session

Permissions automatically apply

Access adjusts as user attributes change or new users are added to your directory

Track user activity

AWS logs attributes in AWS CloudTrail, enabling you to track the user identity for a role session

Ping Identity

Auth0

RSA

ForgeRock®

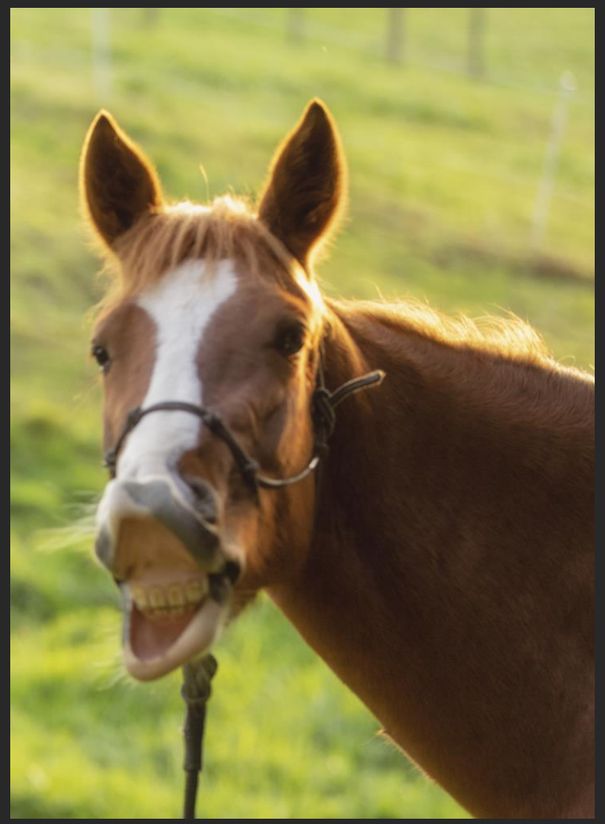
okta

IBM Security

onelogin

Demonstration setup

Project Pickles

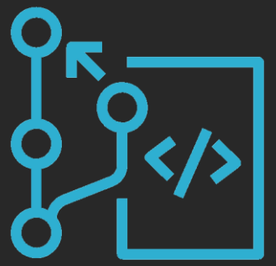


Project Bubbles

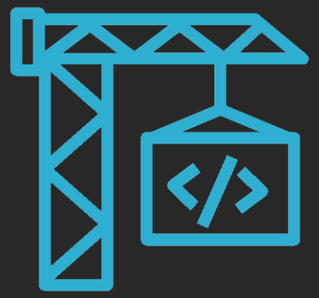
Demonstration setup — Application

Store secrets in AWS Secrets Manager

Store content in Amazon S3



Check in code with AWS CodeCommit



Build AWS CodeBuild

Deploy with AWS Elastic Beanstalk

ABAC challenge

Enable developers to create,

manage, and build applications

based on their project

Steps to applying ABAC in your organization

1

Identities with attributes and federate to AWS with attributes

2

Configure tagging controls

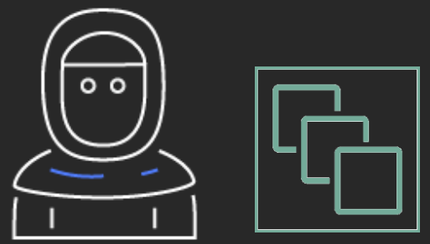
3

Require attributes for new resources

4

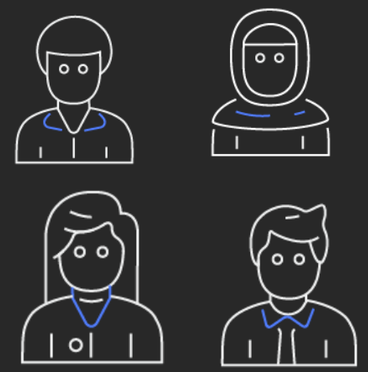
Set permissions based on attributes

5



Create new resources

6



Permissions automatically apply

1. Set up users to federate to AWS with attributes

1

Identities with attributes and federate to AWS with attributes

Ping Identity

Set-up steps

1. User working on project pickles
2. User work on project bubbles
3. Update trust policy to require specific attributes
4. Configure identity provider to pass in required attributes

Pro tip: Reserve specific attributes to use for access control

Trust policy to require specific session tags

```
"Effect": "Allow",
"Principal": {
        "Federated": "arn:aws:iam::432807222178:saml-provider/Ping"
  },
"Action": [
        "sts:AssumeRoleWithSAML",
        "sts:TagSession"
  ],
"Condition": {
        "StringEquals": {
          "SAML:aud": "https://signin.aws.amazon.com/saml"
        },
        "StringLike": {
          "aws:RequestTag/project": "*"
}
```

Trusted IdP

Allowed to pass in session tags

Must pass in project tag

Pro tip: You need to update trust policies to include TagSession

Example SAML assertion to pass in new attributes

```
<Attribute
Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:project">
                    < AttributeValue >pickles<AttributeValue>
</ Attribute>

 <Attribute
Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:userID">
                    < AttributeValue >casey<AttributeValue>
</ Attribute>
```

Configure AWS federation to pass in attributes

2

Configure approved AWS tag keys and values

Set-up steps

1. Create a tag policy with AWS Organizations
    a. Require ‘project’ capitalization match
    b. Only allow pickles and bubbles as acceptable values
2. Apply tag policy to root of organization

Require attributes for new resources

3

Require attributes for new resources

Set-up steps

1. Create a policy
2. Add permissions to require project tag on new resources
3. Add permissions to also allow developers to tag with name tag if they need it

Permission policy to require attributes on new resources

```
{
  "Version": "2012-10-17",
  "Statement": [  {
              "Effect": "Allow",
     "Action": ["secretsmanager:CreateSecret“,
                “codecommit:CreateRepository”,
                “codebuild:CreateProject”],
     "Resource": [ "arn:aws:codecommit:*:*:${aws:PrincipalTag/project}-*",
                   "arn:aws:codebuild:*:*:project/${aws:PrincipalTag/project}-*",
                   "arn:aws:secretsmanager:*:*:secret:${aws:PrincipalTag/project}-*"]
     "Condition": {
         "StringEquals": {
             "aws:RequestTag/project": "${aws:PrincipalTag/project}",
         "ForAllValues:StringEquals": {
              "aws:TagKeys,project": [
                  "project",
                  “name"] } } } ] }
```

Create resources

With this name

With this tag

Only with these keys

Set permissions based on attributes

1

Require attributes for new resources

Set-up steps

1. Add permissions to developer role to manage resources with the same project tag
2. Enable developers to add or update name tags

Permission policy to manage resources using tags

```
{"Effect": "Allow",
"Action": ["codebuild:StartBuild",
           "codecommit:CreateCommit",
           "codecommit:GetRepository"],
"Resource": "*",
"Condition": {
      "StringEquals": {“aws:ResourceTag/project": "${aws:PrincipalTag/project}"} } }

{"Effect": "Allow",
 "Action": ["secretsmanager:GetSecretValue",
            "secretsmanager:DescribeSecret",
            "secretsmanager:PutSecretValue",
            "secretsmanager:DeleteSecret",
            "secretsmanager:UpdateSecret"],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                     "secretsmanager:ResourceTag/project": "${aws:PrincipalTag/project}“ }
            } }
```

Manage only resources with these tags

Permission policy to manage tags

```
"Effect": "Allow",
"Action": [“codecommit:TagResource”],
"Resource": "*",
"Condition": {
        "StringEquals": {
                “aws:ResourceTag/project": "${aws:PrincipalTag/project}"},
        "ForAllValues:StringEquals": {
                "aws:TagKeys": [
                        "project",
                        “name" ] },
        "StringEqualsIfExists": {
                "aws:RequestTag/project": ["${aws:PrincipalTag/project}"]
        ]}}},
```

Tag resources

But only if part of your project

For project, specify only your project

Permission policy to manage other tag values

```
"Effect": "Allow",
"Action": [“codecommit:UntagResource” ],
"Resource": "*",
"Condition": {
        "StringEquals": {
                “aws:ResourceTag/project": "${aws:PrincipalTag/project}" },
        "ForAllValues:StringEquals": {
                "aws:TagKeys": [
                        “name" ] } } } ] }
```

Change tags, but only for your project and with the name tag

Watch developers build

4

Demo steps

1. Sign in as Casey who is working on the pickles application
2. Create a secret
3. Check-in code to use the latest secret
4. Build the latest code
5. Deploy the latest package
6. Check out the Pickle application!

Session attributes in AWS CloudTrail


```
"requestParameters":
        {
        "sAMLAssertionID": "k4d_hYQVN74StIT5_lbUwCNUjUC",
        "roleSessionName": "username",
        "principalTags": {
                "jobfunction": "SystemsEngineer",
                "project": "pickles" },
        "durationSeconds": 3600,
        "roleArn": "arn:aws:iam::432807222178:role/acc-developer-abac",
        "principalArn": "arn:aws:iam::432807222178:saml-provider/Ping"
},
```

Use analytics to rein in permissions

AWS tools to rein in your permission

New! Role and access key last-used information

Easily identify and confidently remove unused IAM users and roles

Service last-accessed information

Analyze permissions and remove unused permissions across IAM and account entities

New! IAM Access Analyzer

Identify and remediate cross account access to resources in your account

Rein in permissions challenge

1. Remove unused roles in your production account

2. Analyze role permissions and service control policies in order to remove unused permissions

📣 Pro tip: Channel your inner Marie Kondo

Rein in permissions: Demo steps

1. Analyze roles last used timestamp and delete those older than 6 months
2. Analyze SCPs to identify unused services
3. Analyze developer role policies and identify unused services

IAM Access Analyzer

New!

Analyze access continuously

Identify resources with public or cross-account access

Achieve the highest levels of security assurance

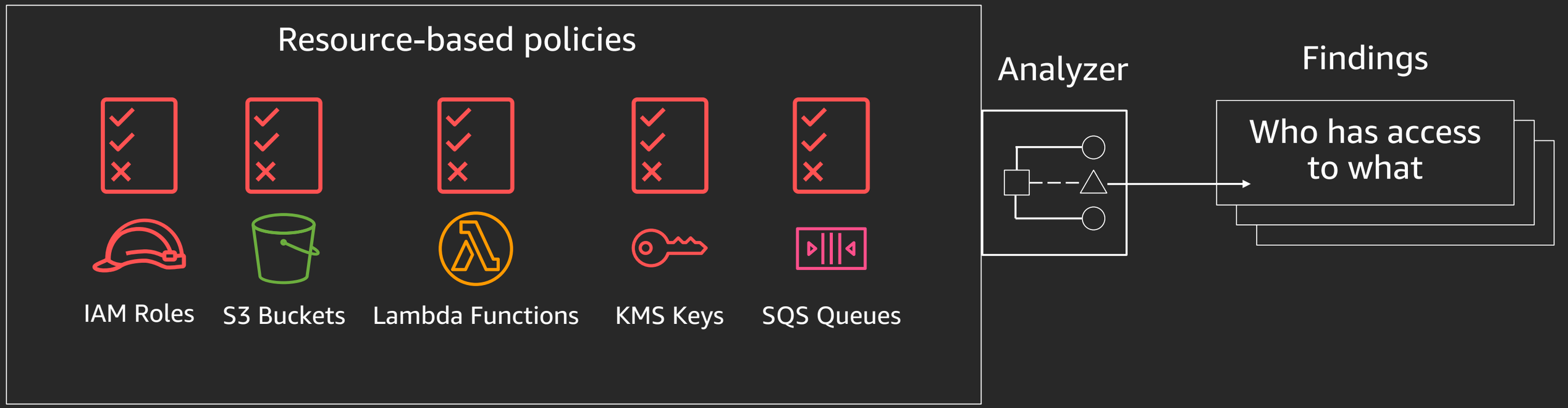
Uses automated reasoning, a form of mathematical logic & inference, to determine all access paths

Remediate broad access

Resolve or archive findings based on your security requirements

COMING SOON! Use IAM Access Analyzer to centrally analyze access across your AWS organization

How IAM Access Analyzer works



Quickly and continuously analyze policies for public and cross account access

Let’s analyze some access

1. Visit Access Analyzer in the IAM console
2. See the finding for a bucket with broad permissions
3. Determine our next step

Resource policy for your buckets

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGetObject",
            "Effect": "Allow",
            "Principal": “*”,
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::pickles-demo-video-public/*",
            "Condition":{
              "ForAnyValue:StringLike":{
                    "aws:PrincipalOrgPaths":[
                          "o-f69sujcm46/r-wiyi/ou-wiyi-csqr4xrj/",
                           "o-f69sujcm46/r-wiyi/ou-wiyi-w0h20sda/"] }}}]}
```

Pro tip: Use PrinciaplOUPath condition key in resource policies

Quick recap

Set yourself up for success with permission guardrails

Rely on attributes for fine-grained permission at scale with ABAC

Use analytics to rein in permissions

Additional resources

Previous talks on policies

Become an IAM Policy Master in 60 Minutes or Less

https://www.youtube.com/watch?v=YQsK4MtsELU

AWS re:Invent 2017: IAM Policy Ninja

https://www.youtube.com/watch?v=aISWoPf_XNE&t=38s

Scale Permissions Management with Attribute-based Access Control

https://www.youtube.com/watch?v=Iq_hDc385t4

Service specific permission documentation

A central location of services, actions, resource-level permissions, and conditions supported across AWS.

Page: Actions, Resources, and Condition Keys for AWS Services

Learn security with AWS Training and Certification

Resources created by the experts at AWS to help you build and validate cloud security skills

30+ free digital courses cover topics related to cloud security, including Introduction to Amazon GuardDuty and Deep Dive on Container Security

Classroom offerings, like AWS Security Engineering on AWS, feature AWS expert instructors and hands-on activities

Validate expertise with the **AWS Certified Security - Specialty** exam

Visit aws.amazon.com/training/paths-specialty/

aws training and certification

Thank you!

Brigid Johnson

@bjohnso5y

AWS re:Invent

aws

Please complete the session survey in the mobile app.