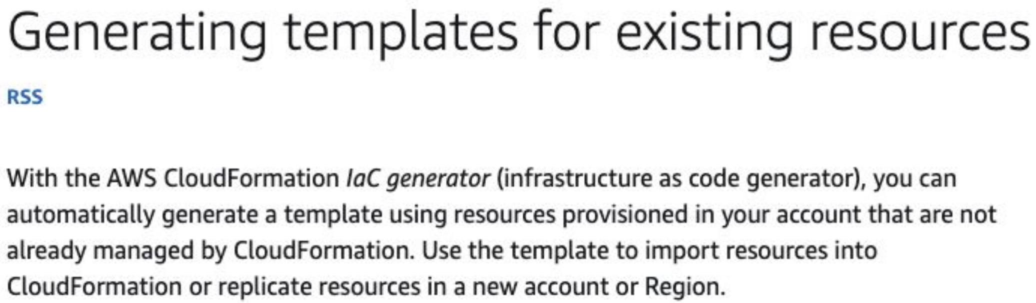
Identity and Access Management Cloud Security Part I

CS 40 | February 5, 2024

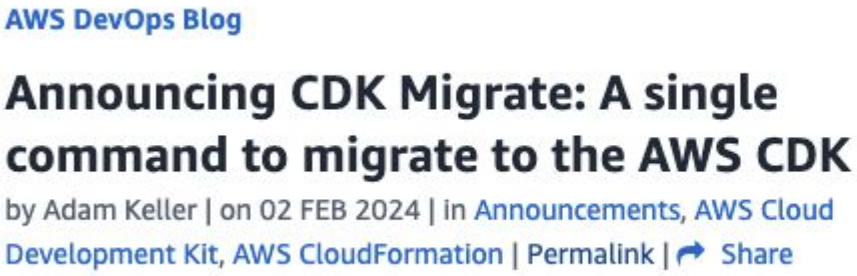
Aside: ClickOps → IaC



Generating templates for existing resources

RSS

With the AWS CloudFormation *IaC generator* (infrastructure as code generator), you can automatically generate a template using resources provisioned in your account that are not already managed by CloudFormation. Use the template to import resources into CloudFormation or replicate resources in a new account or Region.



AWS DevOps Blog

Announcing CDK Migrate: A single command to migrate to the AWS CDK

by Adam Keller | on 02 FEB 2024 | in Announcements, AWS Cloud Development Kit, AWS CloudFormation | Permalink | Share

Amazon Resource Name (ARN)

Unique identifiers for AWS resources.

```
arn:partition:service:region:account-id:resource-type:resource-id
```

- **Partition**: a group of AWS regions (`aws`, `aws-cn`, `aws-us-gov`)
- **Service**: an AWS product (e.g. **`s3`**, **`ec2`**, **`rds`**)
- **Region**: code for AWS region/datacenter (**`us-east-1`**, **`us-west-2`**)
    - Can sometimes be blank, e.g. for global services (Route 53, Cloudfront)
- **Account ID**: 12-digit Account ID of resource owner
    - Can sometimes be blank, e.g. for services uniquely named across accounts (S3)
- **Resource Type**: Type of the resource (e.g. **`vpc`** under **`ec2`**)
    - Can sometimes be blank, if the service has only one resource type (e.g. S3)
- **Resource ID**: Unique identifier for the resource (e.g. bucket name)

Amazon Resource Name (ARN)

Unique identifiers for AWS resources.

```
arn:aws:rds:us-west-2:850592110309:cluster:yoctogram-database
```

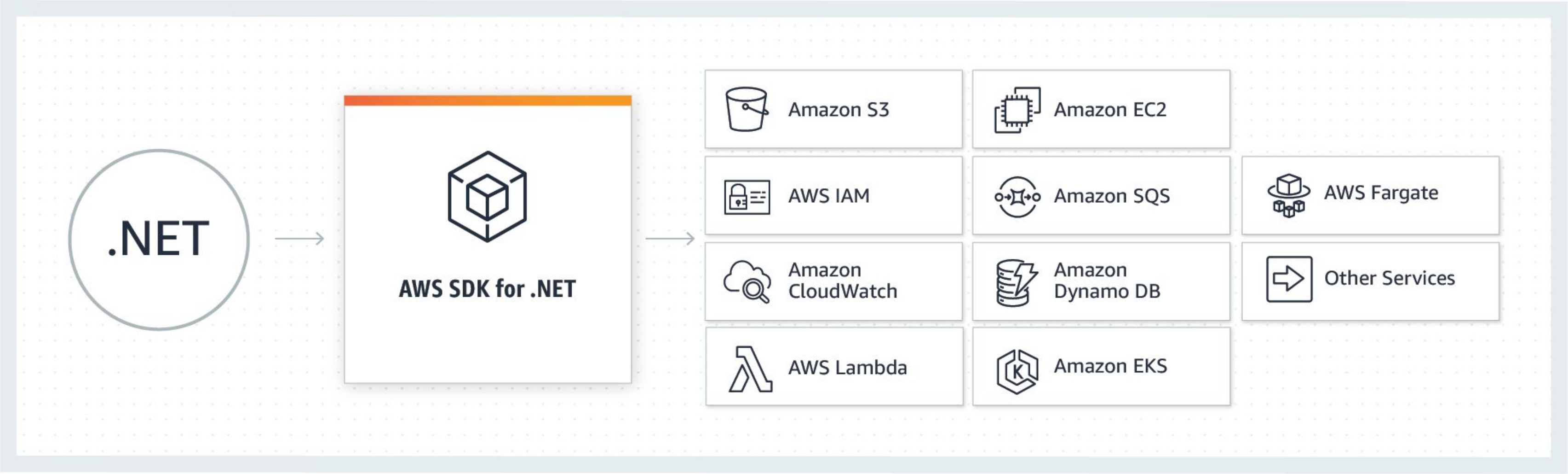
Methods of Accessing AWS

- AWS Console (“clickops”)
- AWS CLI
- **AWS Software Development Kit (SDK)**

*An interaction with AWS via any of these methods creates an API call (an **Action**) – keep this in mind for the rest of the lecture.*

AWS SDK

Interact with AWS services **from your application** using native language constructs



Note: In CS40, we usually talk about infrastructure as separate from app logic.
The AWS SDK allows your application to interact with the infrastructure.

Example: AWS SDK

```
presigned_url = s3_client.generate_presigned_url(
   "get_object",
   Params=parse_s3_uri(s3_uri)
   | {
       "ResponseContentType": content_type,
       "ResponseCacheControl": f"private, max-age={cache_age}, immutable",
   },
   ExpiresIn=settings.CLOUDFRONT_PRESIGNED_URL_EXPIRY,
)
```

Create a URL for external access to an S3 bucket.

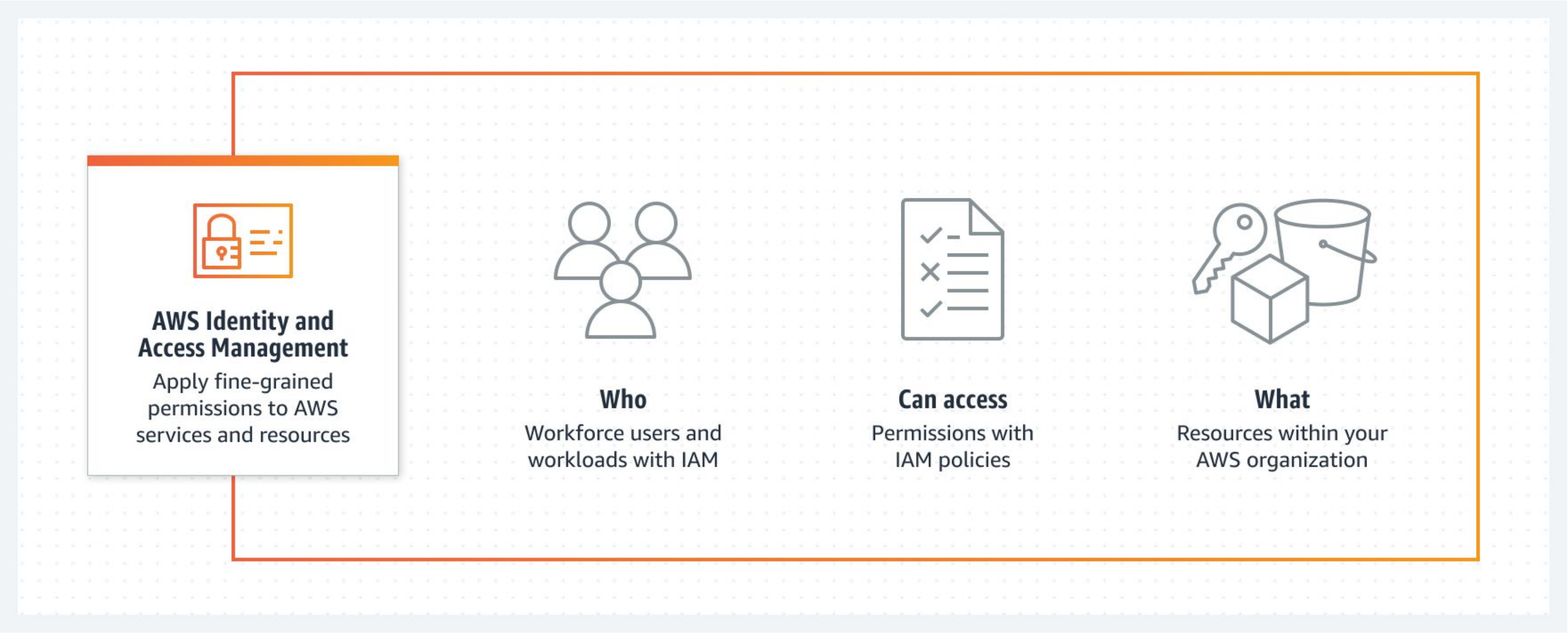
Example: AWS SDK

```go
stsSvc := sts.NewFromConfig(sdkConfig)
result, err := stsSvc.GetCallerIdentity(
    context.TODO(),
    &sts.GetCallerIdentityInput{}
)
if err != nil {
    log.Println(err)
    return err
}
accountID := *result.Account
```

Retrieve the AWS Account ID for the account the code is running in.

Identity and Access Management

IAM Conceptual Model



AWS Identity and Access Management

Apply fine-grained permissions to AWS services and resources

Who

Workforce users and workloads with IAM

Can access

Permissions with IAM policies

What

Resources within your AWS organization

IAM Users

- Give scoped access to AWS account resources to additional users beyond the root user
  - *Scoped*: limited permissions to accomplish specific tasks

- Typically, not best practice to assign human users IAM user accounts directly
  - Instead, use IAM Identity Center (later)

Key IAM Definitions (Agent-Side)

- **Principal**: A human user or workload that can make a request for an action or operation on an AWS resource
    - e.g. *Cody using the AWS CLI*, or *code running on an EC2 instance*

- **Role**: An IAM construct that can be assigned scoped permissions
    - Principals can be assigned, or *assume*, roles; multiple principals can assume a single role
    - Each principal can only assume one IAM role at a time, but may have permissions for multiple

- **Policy**: A listing of the permissions that IAM principals or roles are given
    - Written in JSON
    - e.g. *Allow read and write to all S3 buckets starting with* `cs40-teaching-assistant-`

Key IAM Definitions (Resource-Side)

- **Resource**: Objects within AWS services
  - e.g. *EC2 VMs, S3 buckets*

- **Action**: Operations performed on resources, specific to services
  - e.g. *create an EC2 VM, list objects in an S3 bucket*

- **Policy**: A listing of the permissions that govern access to the resource itself
  - e.g. *deny public downloads from the S3 bucket*

*Note that **IAM policies** can apply to both **principals** and **resources**!*

*Given a **principal** (maybe assuming a **role**) who wants to perform an **action** on a given **resource**, AWS decides whether to authorize or deny the request by evaluating the principal's or resource's **policy**.*

Example Principal Policy (1)

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

AdministratorAccess: Allow every action on every resource

Example Principal Policy (2)

```
{ "Version": "2012-10-17",
  "Statement": [{
     "Effect": "Allow",
     "Action": "rds:*",
     "Resource": ["arn:aws:rds:region:*:*"]
   }, {
     "Effect": "Allow",
     "Action": ["rds:Describe*"],
     "Resource": ["*"]
   }]
}
```

Wildcards allowed in ARNs and actions

Example Principal Policy (3)

```
{
  "Effect": "Deny",
  "Action": "ec2:RunInstances",
  "Resource": "*",
  "Condition": {
    "StringNotEquals": {
      "ec2:ResourceTag/Owner": "${aws:username}"
    }
  }
}
```

Policies can deny access too!
Prevent user from launching EC2 instances that are not tagged as being owned by them

Interactive: What permissions do we need here?

```
presigned_url = s3_client.generate_presigned_url(
    "get_object",
    Params=parse_s3_uri(s3_uri)
    | {
        "ResponseContentType": content_type,
        "ResponseCacheControl": f"private, max-age={cache_age}, immutable",
    },
    ExpiresIn=settings.CLOUDFRONT_PRESIGNED_URL_EXPIRY,
)
```

Create a URL for external access to an S3 bucket.

Interactive: What permissions do we need here?

```
{ "Action": [
       "s3:GetBucket*",
       "s3:GetObject*",
   ],
   "Resource": [
       "arn:aws:s3:::yoctogram-private-images",
       "arn:aws:s3:::yoctogram-public-images",
       "arn:aws:s3:::yoctogram-private-images/*",
       "arn:aws:s3:::yoctogram-public-images/*"
   ],
   "Effect": "Allow" }
```

Read access to all objects in the private and public S3 buckets.

Example Resource Policy

```
{
  "Effect": "Deny",
  "Principal": { "AWS": "*" },
  "Action": "s3:*",
  "Resource": [
    "arn:aws:s3:::yoctogram-public-images",
    "arn:aws:s3:::yoctogram-public-images/*"
  ],
  "Condition": {
    "Bool": { "aws:SecureTransport": "false" }
  }
}
```

Deny access to the S3 bucket if the request doesn’t use HTTPS.

IAM Roles: Attaching Policies to Principals

- IAM roles are a way to temporarily grant specific permissions to specific principals
    - Principal *assumes* role that has policies (allow / deny) *attached*

- Two components
    - **Permission Policy**: *What can the role do?* (previous slides)
    - **Trust Policy**: *Who can assume the role?*

Assuming IAM Roles

- Access to roles is granted via *Security Token Service* (STS)

```
aws sts assume-role \
    --role-arn arn:aws::iam:123456789012:role/my_role \
    --role-session-name my_session
```

- Outputs:
  - *Access Key ID*
  - *Access Key Secret*
  - *Session Token*
  - Setting as environment variables for AWS API calls (via CLI) grants access to role permissions

Note: AWS services assume roles through internal STS API calls (e.g. EC2 thru IMDS)

Demo: Assuming IAM roles

IAM Role Trust Policies

Motivation: don’t want arbitrary principals to assume roles with access to sensitive resources.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/btripp"
  },
  "Action": "sts:AssumeRole"
}
```

All trust policies apply to **Principals** and allow the **`sts:AssumeRole`** action.

IAM Role Trust Policies

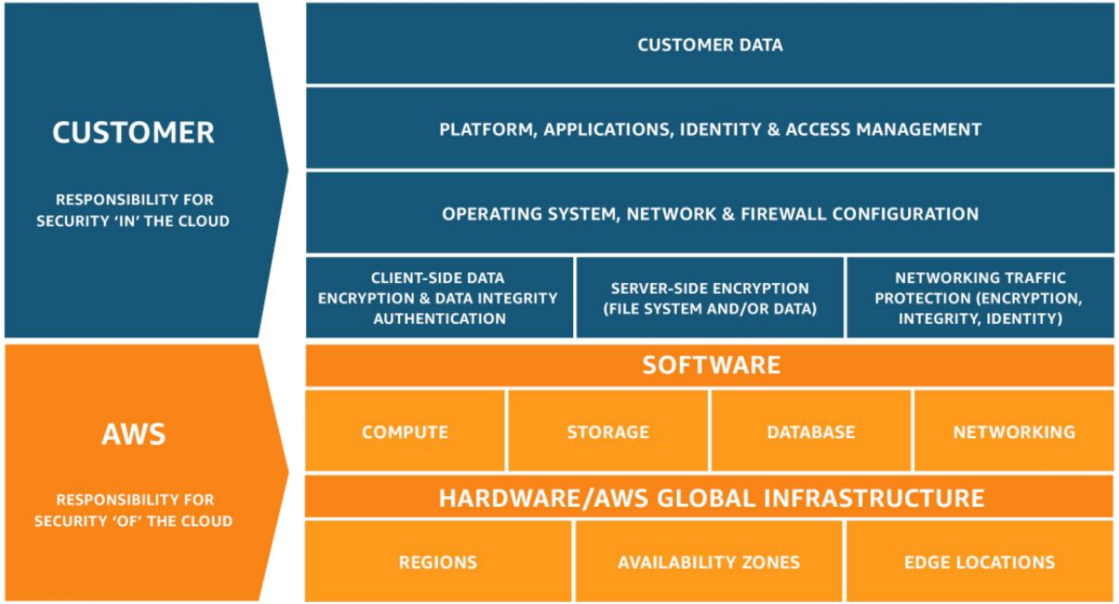
Motivation: don’t want arbitrary principals to assume roles with access to sensitive resources.

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "ecs.amazonaws.com"
  },
  "Action": "sts:AssumeRole"
}
```

Trust policy principals can be services, too!

Common Cloud Security Footguns

The Shared Responsibility Model



CUSTOMER

RESPONSIBILITY FOR SECURITY 'IN' THE CLOUD

CUSTOMER DATA

PLATFORM, APPLICATIONS, IDENTITY & ACCESS MANAGEMENT

OPERATING SYSTEM, NETWORK & FIREWALL CONFIGURATION

CLIENT-SIDE DATA ENCRYPTION & DATA INTEGRITY AUTHENTICATION

SERVER-SIDE ENCRYPTION (FILE SYSTEM AND/OR DATA)

NETWORKING TRAFFIC PROTECTION (ENCRYPTION, INTEGRITY, IDENTITY)

AWS

RESPONSIBILITY FOR SECURITY 'OF' THE CLOUD

SOFTWARE

COMPUTE

STORAGE

DATABASE

NETWORKING

HARDWARE/AWS GLOBAL INFRASTRUCTURE

REGIONS

AVAILABILITY ZONES

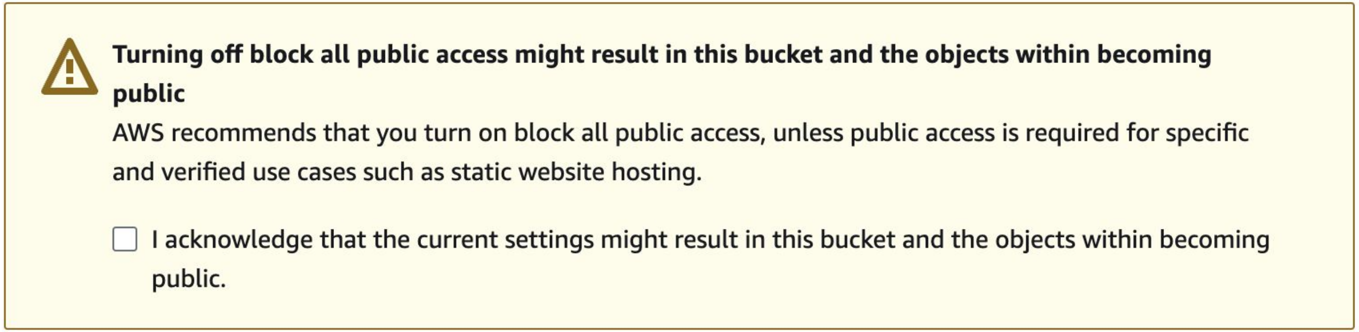
EDGE LOCATIONS

AWS assumes responsibility for its own infrastructure.
You assume responsibility for how you use AWS's infrastructure.

Publicly Exposed S3 Buckets

- Occurs when S3 buckets containing sensitive data don’t have a **block all public access** resource policy

- AWS will warn you about this, but some let the warnings go unheeded – especially if trying to get things to *just work*

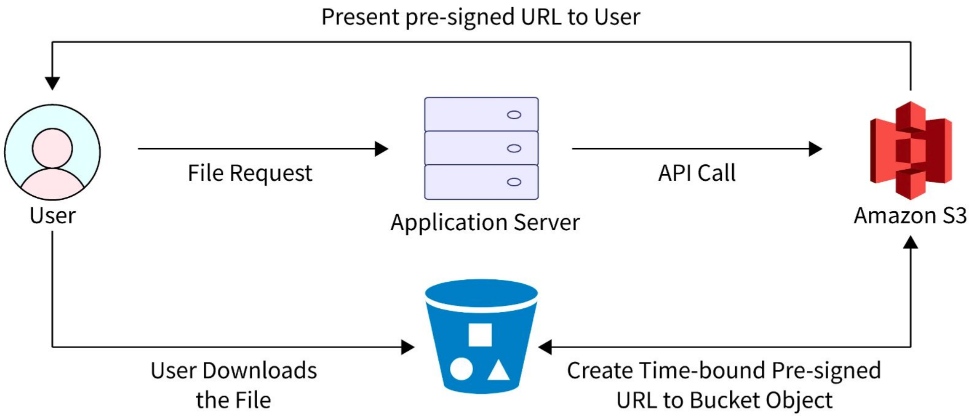


Turning off block all public access might result in this bucket and the objects within becoming public
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

☐ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

Mitigation: Pre-Signed S3 URLs

- Temporarily grant public access to S3 objects by having a trusted party (e.g., your backend) *pre-sign* a URL to access a specific resource
    - Works for GET/POST/PUT access for retrieve, modify, and create


Present pre-signed URL to User
User
File Request
Application Server
API Call
Amazon S3
User Downloads the File
Create Time-bound Pre-signed URL to Bucket Object

Yoctogram (Assignment 2) serves images this way!

Overscoped IAM Policies

- Ensure IAM permissions attached to principals / roles only allow least possible access to make things work
  - “With granular power comes granular responsibility”
- Ensure arbitrary principals can’t assume IAM roles with elevated privileges

Interactive: Overscoped IAM Permissions Policy

```
{
  "Effect": "Allow",
  "Action": [
    "s3:ListBucket",
  ],
  "Resource": [
    "arn:aws:s3:::demo",
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "iam:CreateAccessKey",
  ],
  "Resource": "*"
}
```

What’s wrong
with this policy?

Interactive: Overscoped IAM Permissions Policy

```
{
  "Effect": "Allow",
  "Action": [
    "s3:ListBucket",
  ],
  "Resource": [
    "arn:aws:s3:::demo",
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "iam:CreateAccessKey",
  ],
  "Resource": "*"
}
```

Allows you to create an access key for the root user!

Interactive: Overscoped IAM Permissions Policy

```
{
  "Effect": "Allow",
  "Action": [
    "s3:ListBucket",
  ],
  "Resource": [
    "arn:aws:s3:::demo",
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "iam:CreateAccessKey",
  ],
  "Resource": "arn:aws:iam::*:user/${aws:username}"
}
```

Restrict to creating access keys for the specific user only.

Wildcard IAM Trust Policy

```
{
  "Effect": "Allow",
  "Principal": { "*" },
  "Action": "sts:AssumeRole"
}
```

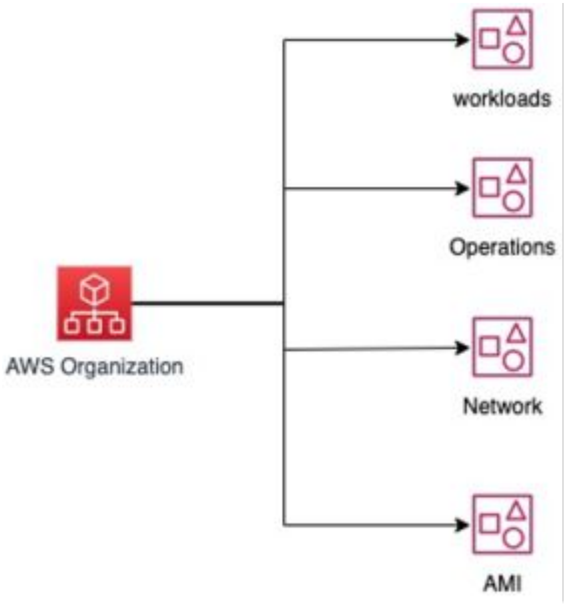
Allows anyone to assume a role with potentially elevated privileges.

Easy Cloud Security Best Practices

AWS Organizations and IAM Identity Center (SSO)

- Instead of having one account with all AWS resources for an organization, use **AWS Organizations** to separate distinct concerns into separate hierarchical accounts

- Use **AWS IAM Identity Center** to delegate user access to AWS accounts
  - This is an easy way to implement Single Sign On, even without a real SSO provider!
  - IAM Identity Center is free



Demo: IAM Identity Center

Security for Human IAM Users

- Within IAM Identity Center: enforce multi-factor authentication for all users
    - AWS accounts are a ***significant*** target for cyberattacks – even for small startups!

- Don’t use long-lived credentials for command-line authentication
    - **`aws sso login`** is your friend

Deploy Using IaC Only

- As with last lecture: IaC gives you a consistent source of truth on the state of your infrastructure

- Allows you to more easily audit your resources, and enforce some previously mentioned security policies

- More next lecture

Next Lecture:
Auditing, Logging, and Observability (2/7)