**AWS Multi-Account Integration Setup**

This document provides detailed steps to configure AWS multi-account integration for Port’s software catalog, complete with an example.

**Understanding AWS Permissions Model**

Before setting up, it is essential to understand these key AWS components:

1. **Policy**: Defines allowed/denied actions on AWS resources.
2. **Role**: An identity with specific permissions.
3. **AssumeRole**: Allows a user/service to assume a role temporarily.
4. **Trust Policy**: Specifies principals that can assume a role.
5. **Account**: Container for AWS resources.
6. **Root Account**: The main account managing the AWS Organization.
7. **Integration Account**: The account where the Port integration is installed.
8. **Member Accounts**: Accounts from which resources are fetched.

**Infrastructure Requirements**

To set up the solution, ensure the following infrastructure requirements are met:

1. **AWS Organization**: Multi-account setup within an AWS Organization.
2. **GitHub Actions**: For CI/CD pipeline to trigger Terraform scripts.
3. **Terraform**: Used for provisioning roles, policies, and trust relationships.
4. **Port Integration**: Installed in the Integration Account.
5. **IAM Resources**: Configured to support cross-account role assumptions.

**Resource Requirements**

The following resources are required:

1. **IAM Roles and Policies**:
   * Integration Role: PortIntegrationRole
   * Root Account Role: OrganizationalOceanRole
   * Member Account Role: AccountReadRole
2. **Terraform State Storage**: Use an S3 bucket with versioning enabled.
3. **Remote Backend Configuration**: Store Terraform state files securely.
4. **GitHub Secrets**: Include AWS credentials for the GitHub Actions workflow.

**Roles and Permissions**

**1. Integration Account Setup**

* **Role Name**: PortIntegrationRole
  + **Policy**: Attach a policy that grants read-only access to resources in the Integration Account. Example policy:
  + {
  + "Version": "2012-10-17",
  + "Statement": [
  + {
  + "Effect": "Allow",
  + "Action": "sts:AssumeRole",
  + "Resource": "arn:aws:iam::<ROOT\_ACCOUNT\_ID>:role/OrganizationalOceanRole"
  + }
  + ]
  + }
* **Trust Policy**: Allow OrganizationalOceanRole from the Root Account to assume this role:
* {
* "Version": "2012-10-17",
* "Statement": [
* {
* "Effect": "Allow",
* "Principal": {
* "AWS": "arn:aws:iam::<ROOT\_ACCOUNT\_ID>:role/OrganizationalOceanRole"
* },
* "Action": "sts:AssumeRole"
* }
* ]
* }

**2. Root Account Setup**

* **Role Name**: OrganizationalOceanRole
  + **Policy**: Attach a policy that grants permissions to list accounts:
  + {
  + "Version": "2012-10-17",
  + "Statement": [
  + {
  + "Effect": "Allow",
  + "Action": "organizations:ListAccounts",
  + "Resource": "\*"
  + }
  + ]
  + }
* **Trust Policy**: Allow PortIntegrationRole to assume this role:
* {
* "Version": "2012-10-17",
* "Statement": [
* {
* "Effect": "Allow",
* "Principal": {
* "AWS": "arn:aws:iam::<INTEGRATION\_ACCOUNT\_ID>:role/PortIntegrationRole"
* },
* "Action": "sts:AssumeRole"
* }
* ]
* }

**3. Member Accounts Setup**

* **Role Name**: AccountReadRole
  + **Policy**: Attach a policy that grants read-only permissions for resources:
  + {
  + "Version": "2012-10-17",
  + "Statement": [
  + {
  + "Effect": "Allow",
  + "Action": [
  + "ec2:DescribeInstances",
  + "s3:ListAllMyBuckets",
  + "rds:DescribeDBInstances"
  + ],
  + "Resource": "\*"
  + }
  + ]
  + }
* **Trust Policy**: Allow PortIntegrationRole to assume this role:
* {
* "Version": "2012-10-17",
* "Statement": [
* {
* "Effect": "Allow",
* "Principal": {
* "AWS": "arn:aws:iam::<INTEGRATION\_ACCOUNT\_ID>:role/PortIntegrationRole"
* },
* "Action": "sts:AssumeRole"
* }
* ]
* }

**Deployment**

1. **GitHub Actions Workflow**:
   * Trigger Terraform scripts to deploy roles, policies, and trust relationships across accounts.
   * Example GitHub Actions workflow file:
   * name: Deploy Multi-Account Setup
   * on:
   * push:
   * branches:
   * - main
   * jobs:
   * deploy:
   * runs-on: ubuntu-latest
   * steps:
   * - name: Checkout Code
   * uses: actions/checkout@v2
   * - name: Setup Terraform
   * uses: hashicorp/setup-terraform@v1
   * - name: Terraform Init
   * run: terraform init
   * - name: Terraform Apply
   * run: terraform apply -auto-approve
2. **Terraform Deployment**:
   * Run terraform apply to provision resources and roles.
3. **Verification**:
   * Test role assumptions with the aws sts assume-role command.

**Sample Terraform Script**

Below is a sample Terraform script to provision roles and trust relationships for multi-account setup:

provider "aws" {

region = "us-east-1"

profile = "default"

}

resource "aws\_iam\_role" "PortIntegrationRole" {

name = "PortIntegrationRole"

assume\_role\_policy = jsonencode({

Version = "2012-10-17"

Statement = [

{

Effect = "Allow"

Principal = {

AWS = "arn:aws:iam::<ROOT\_ACCOUNT\_ID>:role/OrganizationalOceanRole"

}

Action = "sts:AssumeRole"

}

]

})

}

resource "aws\_iam\_role" "OrganizationalOceanRole" {

name = "OrganizationalOceanRole"

assume\_role\_policy = jsonencode({

Version = "2012-10-17"

Statement = [

{

Effect = "Allow"

Principal = {

AWS = "arn:aws:iam::<INTEGRATION\_ACCOUNT\_ID>:role/PortIntegrationRole"

}

Action = "sts:AssumeRole"

}

]

})

}

resource "aws\_iam\_role" "AccountReadRole" {

name = "AccountReadRole"

assume\_role\_policy = jsonencode({

Version = "2012-10-17"

Statement = [

{

Effect = "Allow"

Principal = {

AWS = "arn:aws:iam::<INTEGRATION\_ACCOUNT\_ID>:role/PortIntegrationRole"

}

Action = "sts:AssumeRole"

}

]

})

inline\_policy {

name = "ReadOnlyPolicy"

policy = jsonencode({

Version = "2012-10-17"

Statement = [

{

Effect = "Allow"

Action = ["ec2:DescribeInstances", "s3:ListAllMyBuckets", "rds:DescribeDBInstances"]

Resource = "\*"

}

]

})

}

}

**Rollback**

1. **Trigger Rollback**:
   * Use the same GitHub Actions workflow to run terraform destroy.
2. **Manual Cleanup**:
   * Verify and remove any residual roles or policies manually in the AWS Console if necessary.
3. **Validation**:
   * Ensure no cross-account roles or permissions remain post-rollback.

**Example Configuration**

**Integration Configuration Parameters**

organizationRoleArn: arn:aws:iam::<ROOT\_ACCOUNT\_ID>:role/OrganizationalOceanRole

accountReadRoleName: AccountReadRole

**Example Values**

* **Integration Account ID**: 123456789012
* **Root Account ID**: 234567890123
* **Member Account ID**: 345678901234

**Example Resource ARN**

* **Organization Role ARN**: arn:aws:iam::234567890123:role/OrganizationalOceanRole
* **Integration Role ARN**: arn:aws:iam::123456789012:role/PortIntegrationRole
* **Member Role ARN**: arn:aws:iam::345678901234:role/AccountReadRole

**Verification**

1. **Testing AssumeRole**: Run the following command to test role assumption:
2. aws sts assume-role --role-arn arn:aws:iam::<MEMBER\_ACCOUNT\_ID>:role/AccountReadRole --role-session-name TestSession
3. **Integration Health Check**: Verify data ingestion in the Port dashboard to ensure all accounts are synced successfully.

This document provides step-by-step guidance to configure AWS multi-account support for Port’s integration. Replace <ROOT\_ACCOUNT\_ID>, <INTEGRATION\_ACCOUNT\_ID>, and <MEMBER\_ACCOUNT\_ID> with actual AWS account IDs during setup.