GitHub Actions OIDC Setup Guide for AWS

This guide walks you through setting up OpenID Connect (OIDC) authentication between GitHub Actions and AWS, eliminating the need for long-lived access keys.

Why OIDC?

✅ **More Secure**: No long-lived credentials stored in GitHub
✅ **Short-lived Tokens**: Temporary credentials for each workflow run
✅ **Fine-grained Access**: Role-based permissions with conditions
✅ **Audit Trail**: Clear tracking of which workflows accessed what resources

Step 1: Create OIDC Identity Provider in AWS

Using AWS CLI

```
bash

# Create the OIDC identity provider
aws iam create-open-id-connect-provider \
  --url https://token.actions.githubusercontent.com \
  --thumbprint-list 6938fd4d98bab03faadb97b34396831e3780aea1 \
  --client-id-list sts.amazonaws.com
```

Using AWS Console

1. Go to **IAM Console** → **Identity Providers** → **Add Provider**

2. Choose **OpenID Connect**
3. Set **Provider URL**: `https://token.actions.githubusercontent.com`
4. Set **Audience**: `sts.amazonaws.com`
5. Click **Add Provider**

Step 2: Create IAM Role for GitHub Actions

Create a Terraform file for the OIDC role:

hcl

```
# github-oidc.tf - Add this to your existing Terraform configuration

# Data source for GitHub OIDC provider
data "aws_iam_openid_connect_provider" "github" {
  url = "https://token.actions.githubusercontent.com"
}

# IAM Role for GitHub Actions
resource "aws_iam_role" "github_actions" {
  name = "${local.name_prefix}-github-actions-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Principal = {
          Federated = data.aws_iam_openid_connect_provider.github.arn
        }
        Action = "sts:AssumeRoleWithWebIdentity"
        Condition = {
          StringEquals = {
            "token.actions.githubusercontent.com:aud" = "sts.amazonaws.com"
          }
          StringLike = {
            "token.actions.githubusercontent.com:sub" = "repo:YOUR_GITHUB_ORG/th
          }
        }
```

```
      }
    ]
  })

  tags = merge(local.common_tags, {
    Name = "${local.name_prefix}-github-actions-role"
    Purpose = "CI/CD"
  })
}

# Policy for ECR access
resource "aws_iam_policy" "github_actions_ecr" {
  name        = "${local.name_prefix}-github-actions-ecr"
  description = "ECR permissions for GitHub Actions"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "ecr:GetAuthorizationToken",
          "ecr:BatchCheckLayerAvailability",
          "ecr:GetDownloadUrlForLayer",
          "ecr:BatchGetImage",
          "ecr:InitiateLayerUpload",
          "ecr:UploadLayerPart",
          "ecr:CompleteLayerUpload",
          "ecr:PutImage"
```

```
      ]
      Resource = [
        aws_ecr_repository.flask_app.arn,
        aws_ecr_repository.celery_worker.arn,
        aws_ecr_repository.celery_beat.arn
      ]
    },
    {
      Effect = "Allow"
      Action = [
        "ecr:GetAuthorizationToken"
      ]
      Resource = "*"
    }
  ]
 })
}

# Policy for ECS deployment
resource "aws_iam_policy" "github_actions_ecs" {
 name        = "${local.name_prefix}-github-actions-ecs"
 description = "ECS deployment permissions for GitHub Actions"

 policy = jsonencode({
  Version = "2012-10-17"
  Statement = [
   {
    Effect = "Allow"
    Action = [
```

```
        "ecs:DescribeServices",
        "ecs:DescribeTaskDefinition",
        "ecs:DescribeTasks",
        "ecs:ListTasks",
        "ecs:RegisterTaskDefinition",
        "ecs:UpdateService",
        "ecs:RunTask",
        "ecs:CreateService",
        "ecs:DeleteService"
      ]
      Resource = [
        aws_ecs_cluster.main.arn,
        "${aws_ecs_cluster.main.arn}/*",
        "arn:aws:ecs:${var.aws_region}:${data.aws_caller_identity.current.account_id
        "arn:aws:ecs:${var.aws_region}:${data.aws_caller_identity.current.account_id
      ]
    },
    {
      Effect = "Allow"
      Action = [
        "ecs:DescribeClusters"
      ]
      Resource = "*"
    }
  ]
 })
}

# Policy for supporting AWS services
```

```
resource "aws_iam_policy" "github_actions_support" {
  name        = "${local.name_prefix}-github-actions-support"
  description = "Supporting AWS services for GitHub Actions"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "logs:CreateLogStream",
          "logs:PutLogEvents",
          "logs:DescribeLogGroups",
          "logs:DescribeLogStreams"
        ]
        Resource = [
          "arn:aws:logs:${var.aws_region}:${data.aws_caller_identity.current.account_i
        ]
      },
      {
        Effect = "Allow"
        Action = [
          "ec2:DescribeSubnets",
          "ec2:DescribeSecurityGroups",
          "elbv2:DescribeLoadBalancers",
          "elbv2:DescribeTargetGroups",
          "elbv2:DescribeTargetHealth"
        ]
        Resource = "*"
```

```
    },
    {
      Effect = "Allow"
      Action = [
        "iam:PassRole"
      ]
      Resource = [
        aws_iam_role.ecs_task_execution_role.arn,
        aws_iam_role.ecs_task_role.arn
      ]
    }
  ]
 })
}

# Attach policies to the GitHub Actions role
resource "aws_iam_role_policy_attachment" "github_actions_ecr" {
 role       = aws_iam_role.github_actions.name
 policy_arn = aws_iam_policy.github_actions_ecr.arn
}

resource "aws_iam_role_policy_attachment" "github_actions_ecs" {
 role       = aws_iam_role.github_actions.name
 policy_arn = aws_iam_policy.github_actions_ecs.arn
}

resource "aws_iam_role_policy_attachment" "github_actions_support" {
 role       = aws_iam_role.github_actions.name
 policy_arn = aws_iam_policy.github_actions_support.arn
```

```
}

# Output the role ARN for GitHub Actions configuration
output "github_actions_role_arn" {
  description = "ARN of the IAM role for GitHub Actions"
  value       = aws_iam_role.github_actions.arn
  sensitive   = false
}
```

Step 3: Configure GitHub Repository Secrets

Add the following secrets to your GitHub repository:

Required Secrets

Go to **GitHub Repository** → **Settings** → **Secrets and variables** → **Actions**

```bash
# Required for OIDC authentication
AWS_ROLE_TO_ASSUME = arn:aws:iam::123456789012:role/threatcompass-proc

# Optional: Notification secrets
SLACK_WEBHOOK_URL = https://hooks.slack.com/services/YOUR/SLACK/WEBH
ALERT_EMAIL = admin@yourdomain.com
SMTP_SERVER = smtp.youremailprovider.com
SMTP_USERNAME = your-smtp-username
SMTP_PASSWORD = your-smtp-password
```

Environment Configuration

For production deployments, set up a **GitHub Environment**:

1. Go to **Repository Settings** → **Environments**
2. Create environment named `production`
3. Add **Protection Rules**:
   - ✅ Required reviewers (add team members)
   - ✅ Wait timer (optional, e.g., 5 minutes)
   - ✅ Deployment branches (only `main` branch)

Step 4: Test OIDC Authentication

Create a simple test workflow to verify OIDC setup:

yaml

```yaml
# .github/workflows/test-oidc.yml
name: Test OIDC Authentication

on:
  workflow_dispatch:  # Manual trigger for testing

permissions:
  id-token: write
  contents: read

jobs:
  test-oidc:
    runs-on: ubuntu-latest
    steps:
      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: ${{ secrets.AWS_ROLE_TO_ASSUME }}
          role-session-name: GitHubActions-Test-${{ github.run_id }}
          aws-region: us-east-1

      - name: Test AWS access
        run: |
          aws sts get-caller-identity
          aws ecr describe-repositories --repository-names threatcompass-production
```

Step 5: Security Best Practices

1. Least Privilege Access

hcl

```
# Example: Restrict to specific branches and events
resource "aws_iam_role" "github_actions_restricted" {
  name = "${local.name_prefix}-github-actions-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Principal = {
          Federated = data.aws_iam_openid_connect_provider.github.arn
        }
        Action = "sts:AssumeRoleWithWebIdentity"
        Condition = {
          StringEquals = {
            "token.actions.githubusercontent.com:aud" = "sts.amazonaws.com"
          }
          StringLike = {
            # Only allow main branch and pull requests
            "token.actions.githubusercontent.com:sub" = [
              "repo:YOUR_ORG/threatcompass:ref:refs/heads/main",
              "repo:YOUR_ORG/threatcompass:pull_request"
            ]
          }
          # Optional: Restrict to specific GitHub environments
          # StringEquals = {
          #   "token.actions.githubusercontent.com:environment" = "production"
          # }
```

```
      }
    }
  ]
 })
}
```

2. Conditional Access Based on Repository

hcl

```
# Multiple repositories support
condition = {
 StringLike = {
  "token.actions.githubusercontent.com:sub" = [
   "repo:your-org/threatcompass:*",
   "repo:your-org/threatcompass-infrastructure:*"
  ]
 }
 StringEquals = {
  "token.actions.githubusercontent.com:aud" = "sts.amazonaws.com"
 }
}
```

3. Time-based Access Control

```hcl
  # Optional: Restrict deployment times (business hours only)
  condition = {
    StringEquals = {
      "token.actions.githubusercontent.com:aud" = "sts.amazonaws.com"
    }
    StringLike = {
      "token.actions.githubusercontent.com:sub" = "repo:your-org/threatcompass:*"
    }
    DateGreaterThan = {
      "aws:RequestedRegion" = "2024-01-01T00:00:00Z"
    }
    IpAddress = {
      "aws:SourceIp" = [
        "140.82.112.0/20",   # GitHub Actions IP ranges
        "185.199.108.0/22",
        "192.30.252.0/22"
      ]
    }
  }
```

Step 6: Monitoring and Auditing

CloudTrail Events

Monitor OIDC usage with CloudTrail:

bash

```
# Query CloudTrail for GitHub Actions activity
aws logs filter-log-events \
  --log-group-name CloudTrail/YourLogGroup \
  --filter-pattern "{ $.userIdentity.type = \"AssumedRole\" && $.userIdentity.princip
```

CloudWatch Dashboard

hcl

```hcl
# CloudWatch dashboard for CI/CD monitoring
resource "aws_cloudwatch_dashboard" "cicd" {
  dashboard_name = "${local.name_prefix}-cicd-dashboard"

  dashboard_body = jsonencode({
    widgets = [
      {
        type   = "metric"
        width  = 12
        height = 6
        properties = {
          metrics = [
            ["AWS/ECS", "RunningTaskCount", "ServiceName", aws_ecs_service.flask_a
            [".", "DesiredCount", ".", ".", ".", "."],
            [".", "PendingTaskCount", ".", ".", ".", "."]
          ]
          view   = "timeSeries"
          region = var.aws_region
          title  = "ECS Deployment Status"
          period = 300
        }
      }
    ]
  })
}
```

Step 7: Troubleshooting Common Issues

Issue 1: "No identity-based policy allows the sts:AssumeRoleWithWebIdentity action"

Solution: Check the trust policy conditions and ensure the repository name matches exactly.

```bash
# Debug: Check what GitHub is sending
echo "GitHub Repository: $GITHUB_REPOSITORY"
echo "GitHub Ref: $GITHUB_REF"
echo "GitHub SHA: $GITHUB_SHA"
```

Issue 2: "Token audience validation failed"

Solution: Ensure the audience is set to `sts.amazonaws.com` in both the workflow and IAM role.

Issue 3: ECR Authentication Fails

Solution: Verify ECR permissions and repository names:

```bash
# Test ECR access
aws ecr describe-repositories --region us-east-1
aws ecr get-authorization-token --region us-east-1
```

Issue 4: ECS Deployment Permissions

Solution: Ensure the role has `iam:PassRole` permissions for ECS task roles:

```bash
# Test ECS access
aws ecs describe-clusters --clusters threatcompass-production-cluster
aws ecs describe-services --cluster threatcompass-production-cluster --services
```

Step 8: Advanced Configuration

Multi-Environment Support

hcl

```hcl
# Different roles for different environments
resource "aws_iam_role" "github_actions_staging" {
  name = "${local.name_prefix}-github-actions-staging-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Principal = {
          Federated = data.aws_iam_openid_connect_provider.github.arn
        }
        Action = "sts:AssumeRoleWithWebIdentity"
        Condition = {
          StringEquals = {
            "token.actions.githubusercontent.com:aud" = "sts.amazonaws.com"
            "token.actions.githubusercontent.com:environment" = "staging"
          }
          StringLike = {
            "token.actions.githubusercontent.com:sub" = "repo:your-org/threatcompass
          }
        }
      }
    ]
  })
}
```

Cross-Account Deployments

hcl

```hcl
# For deploying to different AWS accounts
resource "aws_iam_role" "github_actions_cross_account" {
  provider = aws.target_account
  name     = "${local.name_prefix}-github-actions-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Principal = {
          Federated = "arn:aws:iam::TARGET_ACCOUNT:oidc-provider/token.actions.gi
        }
        Action = "sts:AssumeRoleWithWebIdentity"
        Condition = {
          StringEquals = {
            "token.actions.githubusercontent.com:aud" = "sts.amazonaws.com"
          }
          StringLike = {
            "token.actions.githubusercontent.com:sub" = "repo:your-org/threatcompass
          }
        }
      }
    ]
  })
}
```

Summary

With OIDC configured, your CI/CD pipeline now:

✅ **Authenticates securely** without long-lived credentials
✅ **Uses short-lived tokens** (1 hour expiration)
✅ **Provides fine-grained access** control with conditions
✅ **Maintains audit trails** in CloudTrail
✅ **Supports multiple environments** and repositories

The next step is to deploy your Terraform changes and test the complete CI/CD pipeline!