aws
COMMUNITY DAY
OCEANIA
VIRTUAL EDITION

Going Keyless: Eliminating Credential Management in AWS Compute Resources

Putu Sintia
Infrastructure Engineer at Zero One Group

Indonesia, 11 April 2026

aws
COMMUNITY DAY
OCEANIA
VIRTUAL EDITION

Hi!! I'm Sintia

Indonesian based IE with 2 years experience in cloud infrastructure, DevOps and observability.

Link:
https://linktr.ee/putusintia

aws
COMMUNITY DAY
OCEANIA
VIRTUAL EDITION

How It All Started

Client Request

"Can we eliminate hardcoded credentials from our infrastructure?"

My Resistance

I insisted hardcoded credentials were necessary. Multiple requests were needed before I investigated.

The Discovery

AWS documentation revealed my approach was fundamentally flawed. IAM Roles + IMDS is the right way.

aws
COMMUNITY DAY
OCEANIA
VIRTUAL EDITION

The Danger: What We Knew (and Didn't Know)

"Want to use AWS? Generate an Access Key and Secret Access Key from IAM, put it in the code."

That was what we knew. That was what tutorials taught. No .env pushed to repo — by basic standards, careful enough. But hardcoded credentials carry real risks: leaked keys in Git history, exposed in CI/CD logs, no automatic rotation, and broad blast radius if compromised.

But the real problem was more subtle:

IAM user with overly broad permissions — never narrowed down out of fear of breaking things

Single IAM user shared across all workloads — no granularity between services or environments

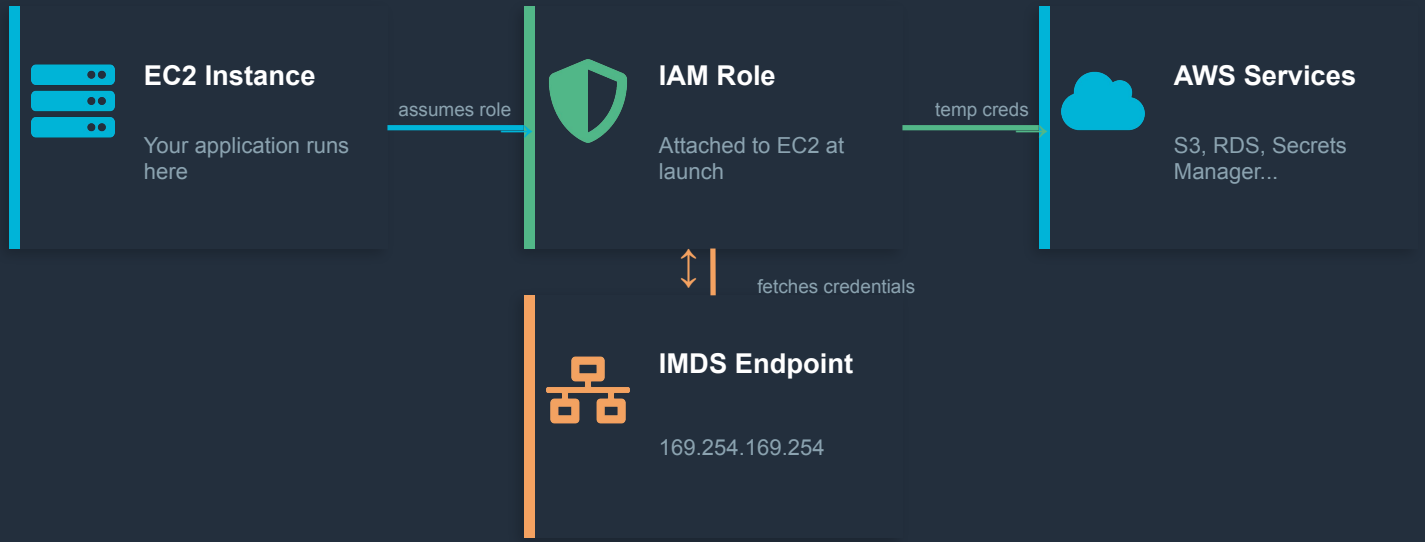
No expiry — same credentials could remain active for years without anyone noticing

Impossible to scope down — no one knew exactly where credentials were being used

Not a leak. A fundamentally flawed design — because we didn't know there was a better way.

aws
COMMUNITY DAY
OCEANIA
VIRTUAL EDITION

IAM Role Architecture



Key Insight: No credentials stored anywhere. AWS injects temporary, auto-rotating credentials via IMDS — your code just reads them.

aws

COMMUNITY DAY

OCEANIA

VIRTUAL EDITION

```
# main.tf — IAM Role + Instance Profile
resource "aws_iam_role" "ec2_role" {
  name = "ec2-app-role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect    = "Allow"
      Principal = { Service = "ec2.amazonaws.com" }
      Action    = "sts:AssumeRole"
    }]
  })
}

resource "aws_iam_role_policy" "app_policy" {
  role = aws_iam_role.ec2_role.id
  policy = jsonencode({
    Statement = [{
      Effect   = "Allow"
      Action   = ["s3:GetObject",
                  "secretsmanager:GetSecretValue"]
      Resource = "*"
    }]
  })
}

resource "aws_iam_instance_profile" "profile" {
  name = "ec2-app-profile"
  role = aws_iam_role.ec2_role.name
}

resource "aws_instance" "app" {
  ami                  = var.ami_id
  instance_type        = "t3.medium"
  iam_instance_profile =
    aws_iam_instance_profile.profile.name
}
```

Production Terraform Implementation

Trust Policy

Allows EC2 service to assume this role — only EC2 can use it

Least Privilege

Grant only the specific permissions your app needs

Instance Profile

The bridge that attaches the IAM Role to EC2 — attach at launch

Minimal Code Adjustments — Retrieving Credentials from IAM Role

Credential Provider Chain — SDK lookup order:

1. **Explicit credentials in code**
   Remove these
2. **Environment variables**
   AWS_ACCESS_KEY_ID etc.
3. **AWS credentials file**
   ~/.aws/credentials (local dev)
4. **IAM Role via IMDS**
   ← On EC2, this is what we rely on

What happens at Step 4 (behind the scenes):

- PUT → 169.254.169.254/latest/api/token → get session token (IMDSv2)
- GET → .../iam/security-credentials/<role> → returns AccessKeyId, SecretKey, Token, Expiration
- SDK uses temp credentials for all API calls — auto-refreshes before expiry

❌ **BEFORE**

```
import boto3
s3 = boto3.client(
  "s3",
  aws_access_key_id=
    "AKIAIOSFODNN7",
  aws_secret_access_key=
    "wJalrXUtnFEMI"
)
```

✅ **AFTER — IAM Role via IMDS**

```
import boto3
# SDK auto-fetches from IMDS:
# AccessKeyId (ASIA...)
# SecretAccessKey
# SessionToken
# Expiration (auto-refresh)
s3 = boto3.client("s3")
```

Docker Swarm Considerations

IMDS Access from Containers

Containers need hop limit = 2 (container → host → IMDS). Most AMIs default to 1 — except Ubuntu 22.04+ which ships with 2 out of the box. Best practice: set explicitly in Terraform regardless of AMI.

```
# Don't rely on AMI defaults — be explicit
metadata_options {
  http_tokens                 = "required"
  http_put_response_hop_limit = 2
  http_endpoint               = "enabled"
}
```

Shared IAM Role in Swarm

All containers on one EC2 node share the node's IAM Role. No per-service roles like ECS Task Roles. Separate workloads with different permissions into different node pools.

```
# docker-compose.yml (Swarm mode)
services:
  app:
    image: myapp:latest
    environment:
      - AWS_DEFAULT_REGION=ap-southeast-1
    # No credentials needed
```

Instance Metadata Service (IMDS) Deep Dive

Feature	IMDSv1 (Legacy)	IMDSv2 (Recommended)
Request type	Simple GET	Session-oriented (PUT + GET)
SSRF protection	❌ None	✅ Protected
Token required	❌ No	✅ Yes (TTL-based)
Container support	✅ Default hop=1	⚠️ Needs hop-limit=2
AWS Recommendation	Deprecated	Use this!

Fetching Credentials via IMDSv2 (for debugging / verification)

```
TOKEN=$(curl -s -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")
curl -s -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-data/iam/security-credentials/ec2-app-role
```

Real Troubleshooting + Proof It Works

NoCredentialProviders

Instance Profile not attached → re-launch with correct Terraform

Container timeout to IMDS

Hop limit = 1 → set --http-put-response-hop-limit 2

Access Denied on S3

IAM policy incomplete → use IAM Policy Simulator

Credentials expired

Don't cache raw tokens → use boto3 Session (auto-refresh)

✅ Proof It Works

Temporary Credentials from IMDS:

```
X-Amz-Credential=
 ASIAR72PHFFRXC7PDEDU/...
```

ASIA... = Temporary (STS) ✅

Not AKIA... (permanent IAM user) ✅

X-Amz-Security-Token present ✅

Valid Presigned URL:

```
?X-Amz-Algorithm=AWS4
&X-Amz-Expires=900
&X-Amz-Signature=7f976e...
```

Signed with IAM Role creds ✅

Expiry 900s — short-lived as expected ✅

aws
COMMUNITY DAY
OCEANIA
VIRTUAL EDITION

Key Takeaways

Stop using hardcoded credentials

Not a blame — just a better way now known.

IAM Roles are the right approach

Temporary, auto-rotating, auditable. Zero credential storage.

Code changes are minimal

Remove explicit credentials. SDK retrieves them from IAM Role via IMDS automatically.

Docker Swarm: set hop-limit to 2

One parameter. Resolves all container IMDS access.

Least privilege from day one

One role per workload, specific permissions only. Use IAM Policy Simulator.

aws
COMMUNITY DAY
OCEANIA
VIRTUAL EDITION

"If clients hadn't pushed back,
I'd still be storing secrets in plaintext."

Go keyless. Your future self will thank you.


aws
COMMUNITY DAY
OCEANIA
VIRTUAL EDITION