Dynamic Identities and Secrets for your applications

whoami

- ex-dev, ex-SRE
- Staff Solutions Architect @HashiCorp
- HashiCorp Ambassador 2021 and 2022
- Passionate about automation and infrastructure

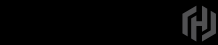
https://atodorov.me/

Introduction to Nomad

A flexible and modern orchestrator

Orchestrate Any Application/workload

- in any format:
  - Docker, Podman containers
  - `exec` any existing binary/script/artifact
  - Virtual Machines with QEMU/KVM
  - Firecracker microVMs
- on any OS
  - Linux, Windows, macOS, FreeBSD
- on any* hardware
  - from Raspberry Pis to mainframes, RISC-V and x86 VMs and bare metal



Nomad Main Features

Secure identity

Each application has a unique and cryptographically verifiable identity which can be used to authenticate to third parties and to Nomad itself for secrets and API access.

Advanced deployments

- Rolling
- Blue/Green
- Canary

Geo Distributed

A cluster can have its nodes spread anywhere (multiple DCs, cloud regions, on ships/trains/submarines all around the world).

Multiple clusters can be federated for a common control plane while keeping local control/decision making.

Service Discovery

Register services in a catalogue, and discover them from any other application.

Dynamic Templating

Configuration files/env variables can be provisioned for applications using metadata, secrets, service discovery, etc. Updates are handled.

Low overhead, low maintenance

Single binary, updates are just replace it + restart.

HashiStack

Integrates with Consul for Service Discovery and Mesh, Vault for secrets management, Terraform for complex deployments.

Application Deployment as Code

Job

Self-contained HCL file.

Declaratively specifies the rules for running your application.

Group

A set of tasks that should be co-located and co-scheduled on the same client.

Task

The smallest atomic unit of work, the actual thing to run.

TERMINAL

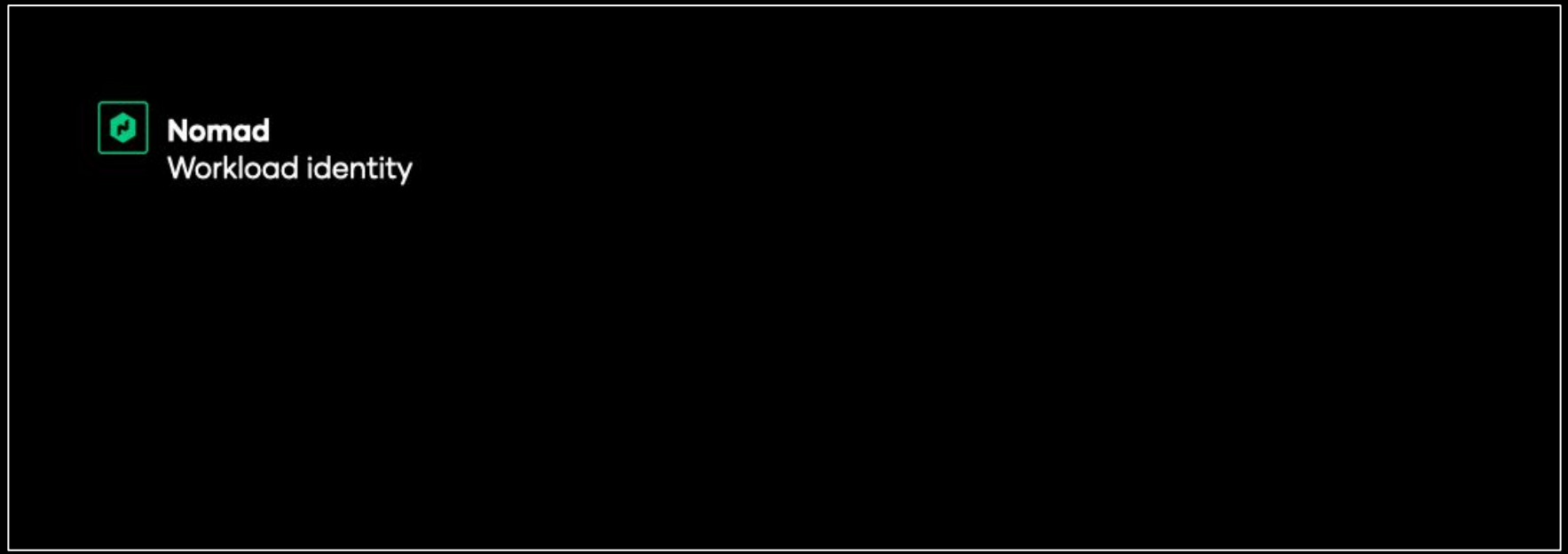
```
job "my_job" {

  region      = "eu"
  datacenters = ["gcp-europe-west9", "aws-eu-west-3", "pa3"]
  node_pool   = "arm64"

  type = "service"

  group "web" {
    count = 5

    task "frontend" {
      driver = "docker"

      config {
        image = "hashicorp/web-frontend"
      }

      resources {
        cpu    = 500 # MHz
        memory = 128 # MB
```

Some Concepts

- OIDC - OpenID Connect protocol, Identity Federation / Single Sign On
- JWT - JSON Web Tokens, signed JSON representing claims between two parties
- Workload Identity - the Identity of a workload, represented by a JWT
- Workload Identity Federation - setting up “trust” between different Identity Providers via OIDC and JWT, to allow one workload’s identity to be trusted by others. Examples:
  - AWS IAM identity to be trusted by GCP to allow assuming a GCP Service Account
  - Microsoft Entra ID identity to be trusted by GitLab to allow users to authenticate
  - Nomad WI to be trusted by Vault, GCP, AWS, etc. to allow workloads to authenticate

Workload Identity in motion

The end to end flow



Nomad JWT format

"aud": "nomadproject.io", # audience, who is this for, arbitrary value

"iat": 1733164029, #issued at, in unix timestamp

"iss": "https://nuc-nomad.lab", # issuer, as defined in the Nomad configuration, arbitrary value

"jti": "bfa76498-16ea-45b5-165e-97a78c0940e4",

"nbf": 1733164029, #not before, not to be used before this time

"nomad_allocation_id": "19faa4d6-18c6-c69b-34e4-df0a6ce2a59c", # Nomad allocation ID

"nomad_job_id": "traefik", # Nomad job ID/name

"nomad_namespace": "default", # Nomad Namespace name

"nomad_task": "traefik", # Nomad Task Name

"sub": "global:default:traefik:traefik:traefik:default" # region+namespace+job+taskgroup+task+identity

GCP Workload Identity Federation

GCP WIF

GCP Workload Identity Federation allows GCP IAM access to be given to trusted identities from other Identity providers, such as:

- AWS, Azure
- Active Directory
- X.509 Certificates
- Any OIDC or SAML compliant provider (like Nomad, GitLab, GitHub)

Importantly for OIDC, it allows importing the public signing keys, and does not require GCP to have access to the OIDC provider itself.

GCP WIF Configuration

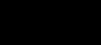
GCP WIF configuration flow:

1. WIF Pool
2. WIF Pool Provider - OIDC, mapping claims from the JWTs to GCP IAM attributes
3. IAM Service Account with roles giving them access to stuff
4. IAM Service Account binding to bind WIF identities to the Service Account
5. Profit

Demo

Demo contents

1. Terraform to configure the GCP WIF, Identity Provider, Service Account + a GCS bucket
2. A Nomad job using the gcloud CLI which authenticates with its Workload Identity to the GCP Service Account
3. With its Service Account access, it pushes a dynamically generated file to a GCS bucket containing a secret + some metadata



Links

Code: https://github.com/sofixa/nomad-dynamic-identities-and-secrets

Slide deck: https://atodorov.me/talks/nomad-workload-identity

Nomad Workload Identity Federation with GCP tutorial:
https://developer.hashicorp.com/nomad/tutorials/fed-workload-identity/integration-gcp

Nomad Workload Identity with AWS tutorial:
https://developer.hashicorp.com/nomad/docs/operations/aws-oidc-provider

https://developer.hashicorp.com/nomad