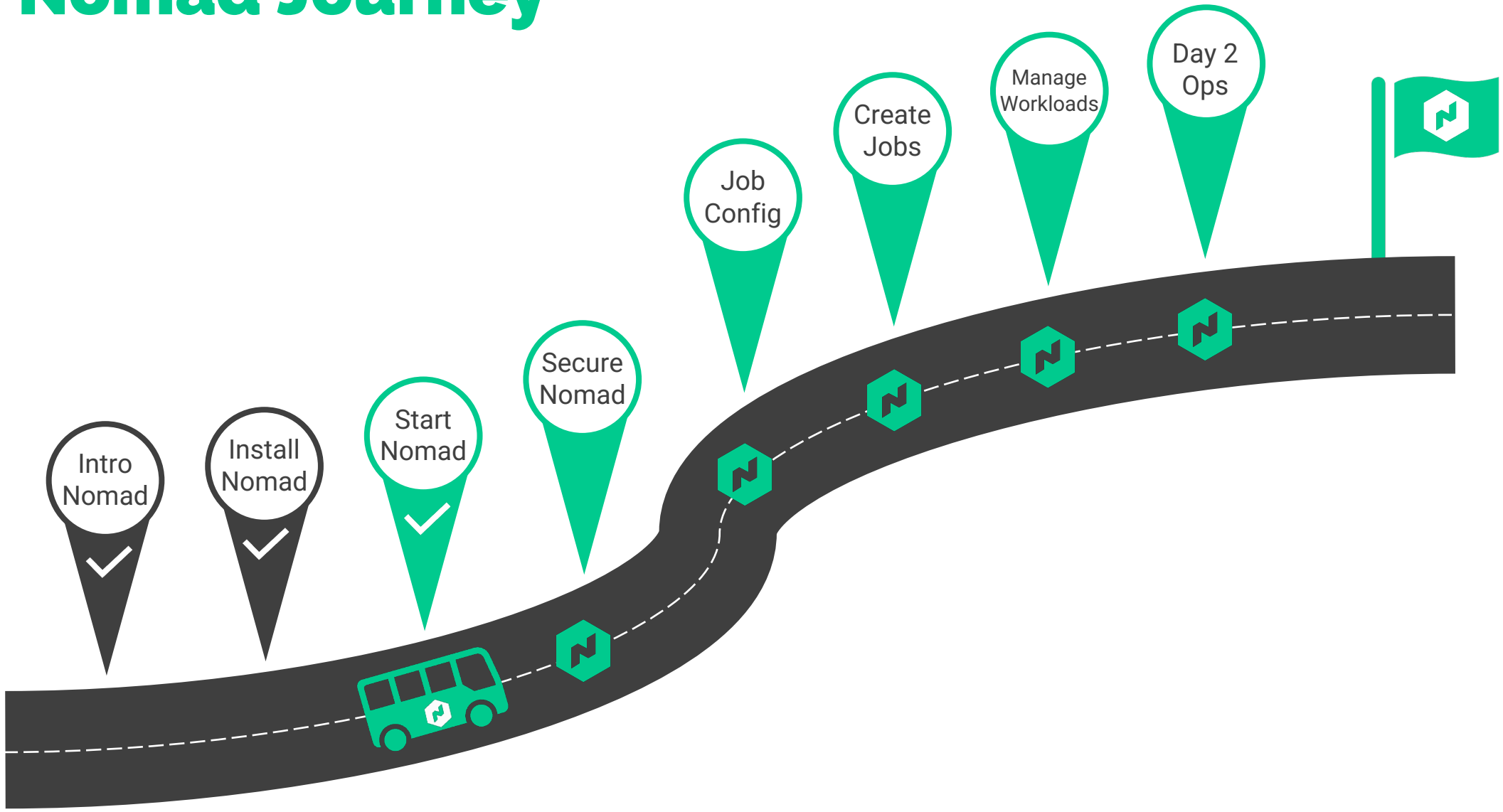
Securing Nomad

Nomad Journey



Security Model

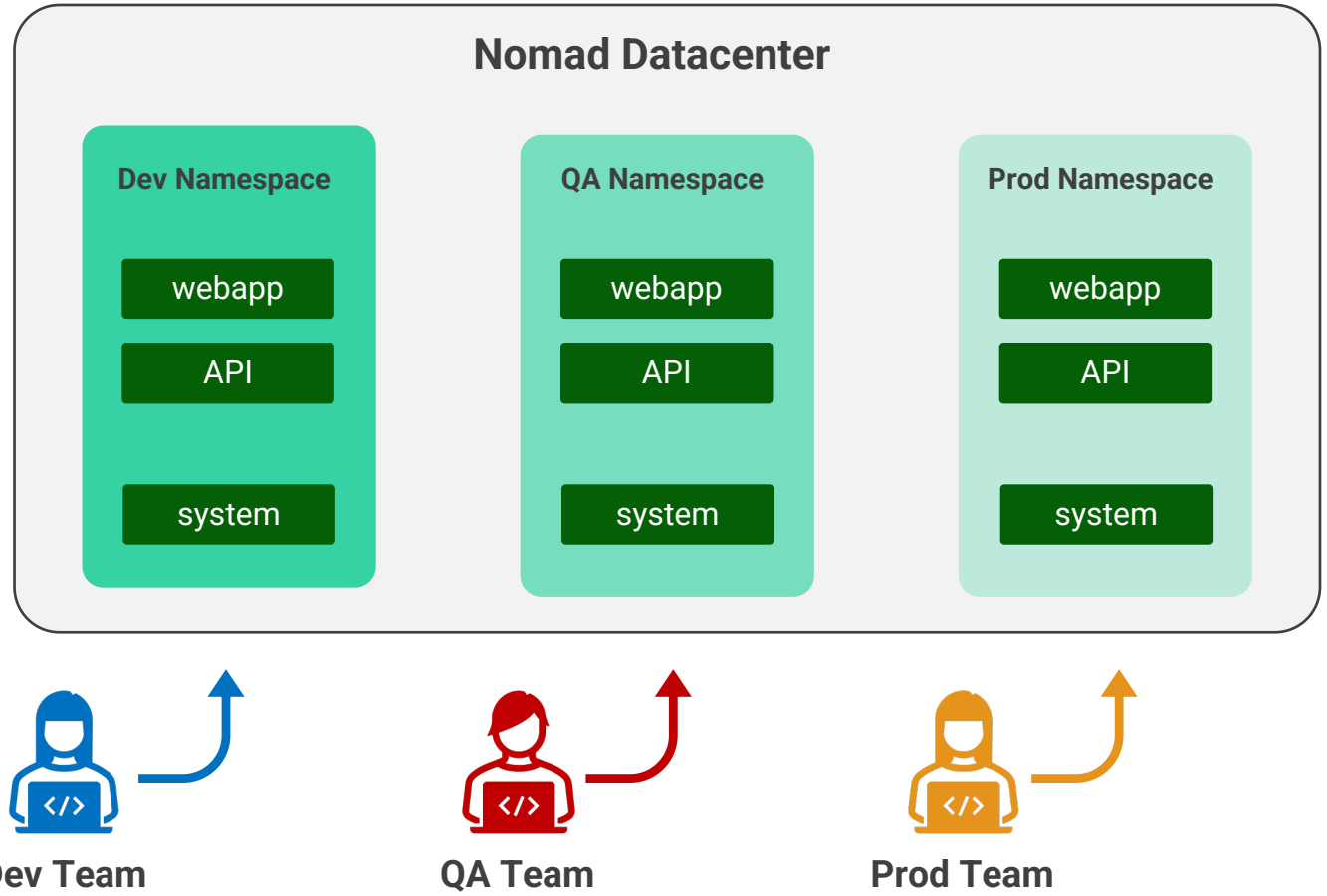
- Similar to most other HashiCorp products, it's up to **YOU** to secure the Nomad environment
- By default, Nomad is **not** secure:
  - **TLS encryption** is not configured – data between servers & clients are sent in clear text
  - **ACLs** are disabled meaning anybody can configured the Nomad environment
  - **Namespaces** are not used by default so there is no isolation between teams
  - **Sentinel** policies must be developed and applied where needed **(Enterprise feature)**
  - **Gossip** is not encrypted
  - **Resource quotes** are not configured so operators are not restricted to the underlying compute resources **(Enterprise feature)**

Security Model

Minimal security tasks for a secure production environment:

- **Secure Nomad with TLS certificates** from a trusted CA to avoid sending data in clear text and eliminate MITM attacks
- **Enable ACLs**. Otherwise, ANYBODY can make configuration changes to the cluster and workloads
- Don't run the Nomad service as the root user – **create an unprivileged user** with only the permissions needed to run the service
- **Lock down any directories** used on Nomad servers and clients to avoid accidental or intentional modifications to the binary, configuration files, drivers, systemd service files, etc.
- **Limit SSH/RPD access** to the Nomad servers and clients. Use immutable infrastructure if possible

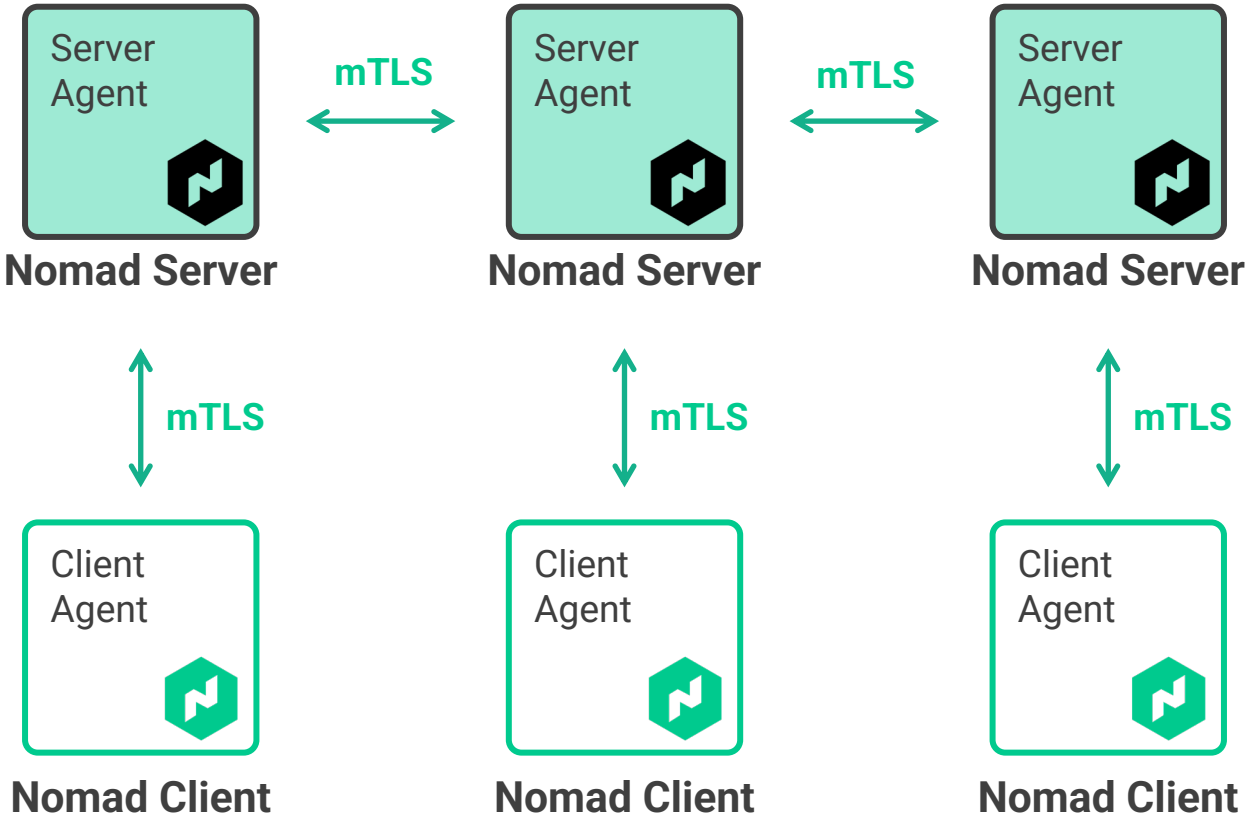
Namespaces



Allows many teams and projects to share a single multi-region Nomad deployment without conflict

- ACL policies provide enforcement of namespaces
- Job IDs are required to be unique with a namespace but not across namespaces
- Namespaces are automatically replicated across regions for easy, centralized administration at scale

Enabling TLS Encryption



- Prevent unauthorized access to Nomad
- Stop any observation or tampering with communications
- Prevent server/client misconfigurations (accidental or malicious)
- Prevent services from representing themselves as a Nomad agent

Enable TLS Encryption

- Nomad requires that you use certs from the same CA throughout the datacenter
- You will have multiple types of TLS certs for Nomad, including:
  - Server agents
  - Client agents
  - CLI and UI
- Certs need to be generated from a trusted CA – likely you have one deployed in your environment already, such as Vault. If not, you can use tools like `openssl` or `cfssl`
- Nomad requires that servers use a certificate that uses `server.<region>.nomad` and clients use `client.<region>.nomad`
  - This is somewhat different than traditional TLS certs where you usually create a cert for the DNS name
  - This strategy also prevents a client from presenting itself as a server

Enable TLS Encryption

```
# TLS configurations
tls {
  http = true
  rpc  = true

  ca_file   = "/etc/certs/ca.crt"
  cert_file = "/etc/certs/nomad.crt"
  key_file  = "/etc/certs/nomad.key"
}
```

TLS configuration as shown in the Nomad agent configuration file

CA cert should be the same across all agents

Use the `server` cert & key for servers

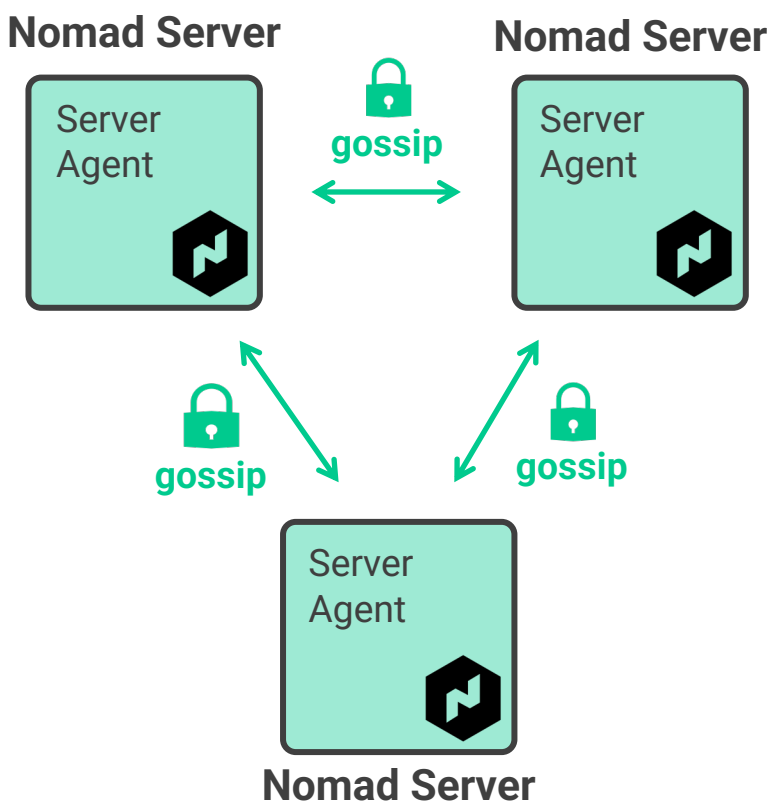
Use the `client` cert & key for servers

DEMO

Securing Nomad with TLS

Gossip Encryption

- By default, Gossip is **NOT** encrypted
- Gossip (Serf) uses an encryption key across **all servers** in the datacenter
- Federation requires that the same key be used on all other datacenters as well



Gossip Encryption

- The gossip encryption key is a **pre-shared key**, meaning you must create it and provide it in the agent configuration file

- You can use any method that can create 32 random bytes encoded in base64, however, I recommend using the built-in tool to avoid any issues

- Encryption key can be easily created by using the `nomad operator gossip keyring generate` command

TERMINAL

```
$ nomad operator gossip keyring generate
Do7GerAsNtzK527dxRZJwpJANdS2NTFbKJIxIod84u0=
```

Gossip Encryption

- Key is placed in the configuration file in the server configuration stanza
- Each server agent configuration file should include this SAME key

```
# Server & Raft configuration
server {
  enabled          = true
  bootstrap_expect = 3
  encrypt          = "Do7GerAsNtzK527dxRZJwpJANdS2NTFbKJIxIod84u0="
  license_path     = "/etc/nomad.d/nomad.hclic"
  server_join {
    retry_join = ["server_a.example.com", "server_b.example.com", "server_c.example.com"]
  }
}
```


Gossip Encryption

- Use the command `nomad agent-info` to validate gossip is encrypted

TERMINAL

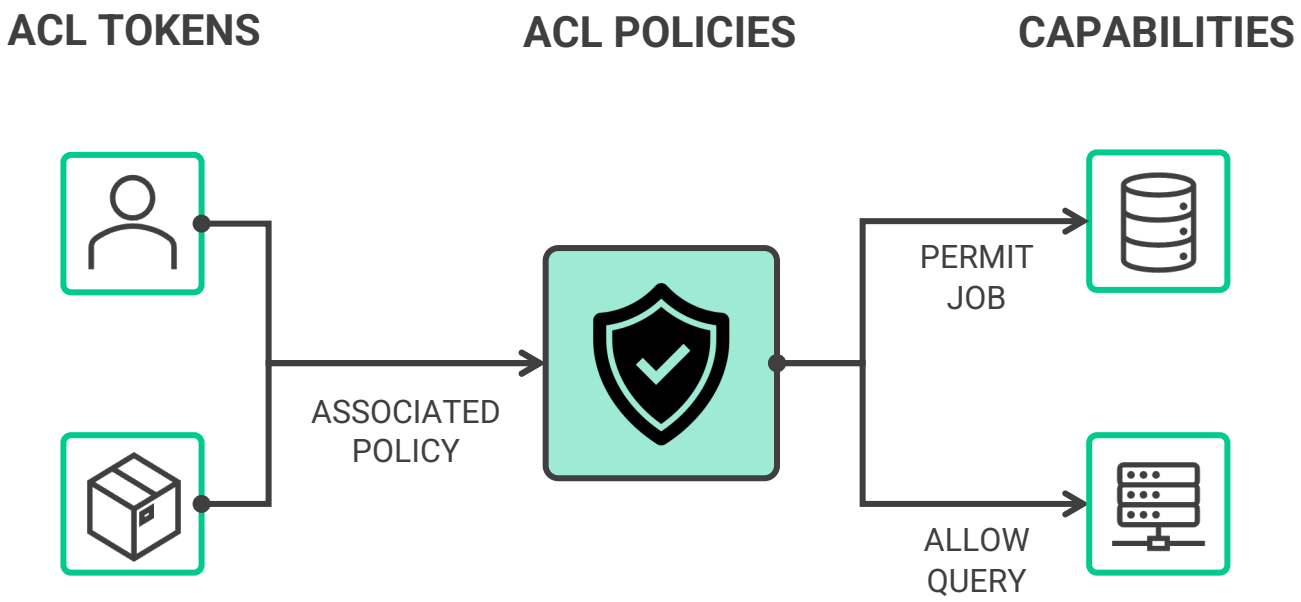
```
$ nomad agent-info
..
serf
  intent_queue = 0
  member_time = 1
  query_queue = 0
  event_time = 1
  event_queue = 0
  failed = 0
  left = 0
  members = 5
  query_time = 1
  encrypted = true
```


DEMO

Securing Nomad Gossip

Secure Nomad with ACLs

- Token-based authentication
- Tokens are associated with a policy that permits/denies access to capabilities in Nomad
- Policies are centrally managed
- Policies and tokens are automatically replicated across regions for easy, centralized administration at scale



Secure Nomad with ACLs

There are three components to the ACL system:

Tokens

- Management tokens
- Client tokens (default)

Policy

- Policies provide role-based access control
- Associated with tokens

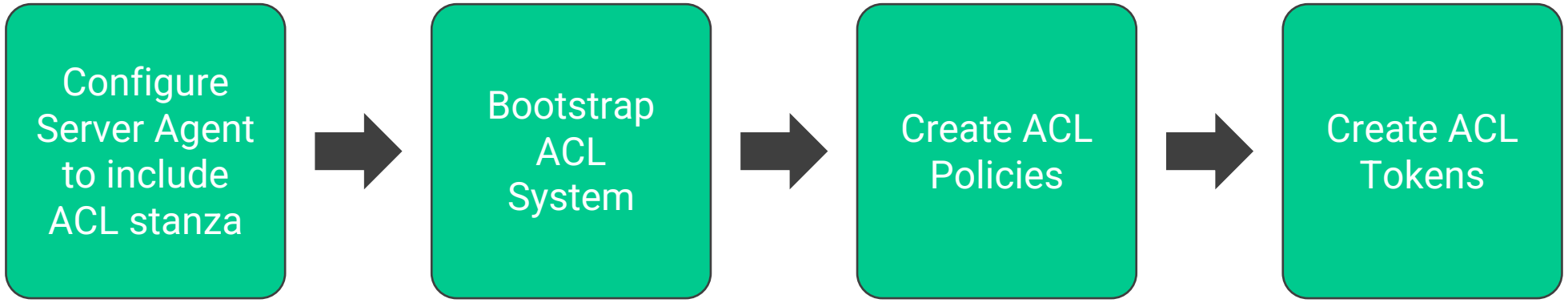
Capabilities

- Defines the actions that can be performed on the designated path

Secure Nomad with ACLs

- Nomad ACLs are NOT enabled by default and therefore the ACL must be bootstrapped before you can use them to secure Nomad
- All servers must include the `acl` stanza and parameters in the agent config, otherwise you'll get an error message stating that ACL support is disabled

Steps Required to Use Nomad ACLs



Secure Nomad with ACLs

```
# Server & Raft configuration
server {
  enabled          = true
  bootstrap_expect = 3
  encrypt          = "Do7GerAsNtzK527dxRZJwpJANdS2NTFbKJIxIod84u0="
  license_path     = "/etc/nomad.d/nomad.hclic"
  server_join {
    retry_join = ["server_a.example.com", "server_b.example.com", "server_c.example.com"]
  }
}

# Client Configuration - Disable for Server nodes
client {
  enabled = false
}

# Enable and configure ACLs
acl {
  enabled    = true
  token_ttl  = "30s"
  policy_ttl = "60s"
  role_ttl   = "60s"
}
```


Bootstrap ACLs

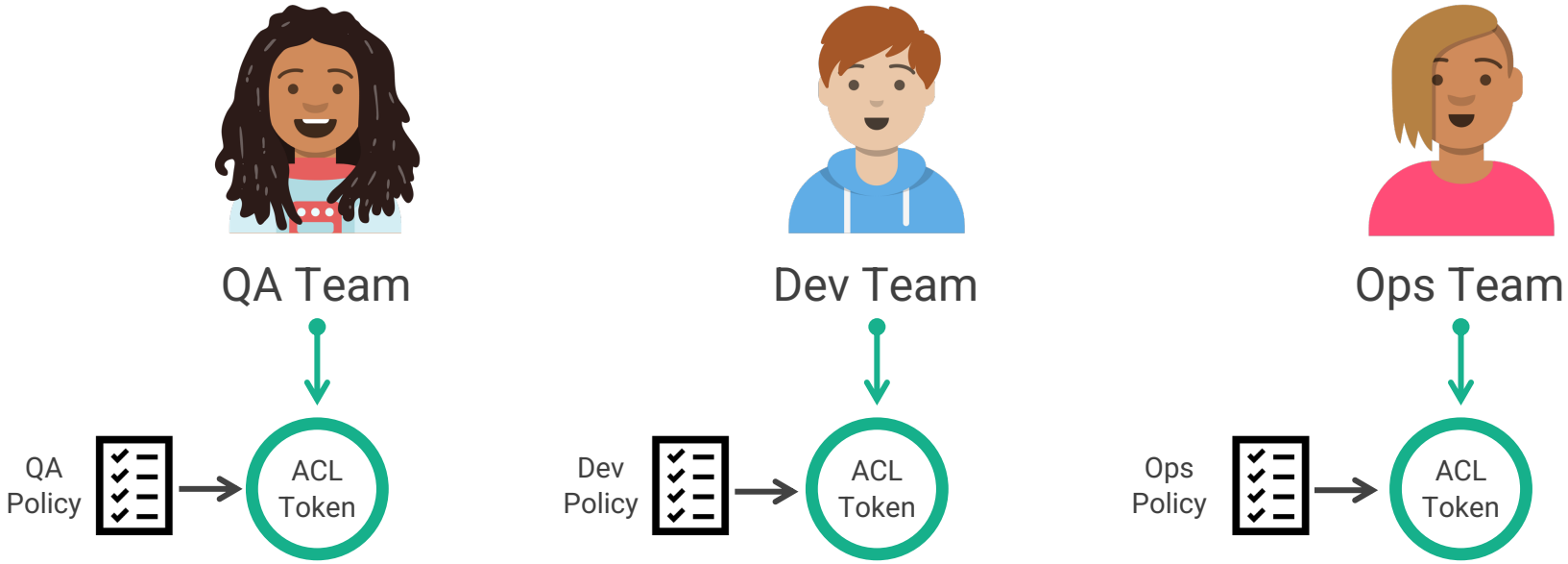
To bootstrap the ACL system, use the `nomad acl bootstrap` command:

TERMINAL

```
$ nomad acl bootstrap
Accessor ID  = 400a8f88-8f73-ef48-0750-fd122e2abe8d
Secret ID    = 4a8be0a9-459c-6598-ac8b-d80f26a6e8f0
Name         = Bootstrap Token
Type         = management
Global       = true
Create Time  = 2023-01-03 14:19:04.509226313 +0000 UTC
Expiry Time  = <none>
Create Index = 9877
Modify Index = 9877
Policies     = n/a
Roles        = n/a
```

ACL Tokens

- When the ACL system is bootstrapped, you get the **bootstrap token**
- The bootstrap token is a management token that provides access to everything
- It is **NOT** recommended that you use this token for day-to-day operations



ACL Tokens

TERMINAL

```
$ nomad acl token create -name="nomad_is_awesome" -policy="krausen"
Accessor ID  = ebea4525-51a4-3b6d-6511-da8fecf63eb1
Secret ID    = 0f8b37f4-6f22-ef1d-c9aa-e1f04a86dd76
Name         = nomad_is_awesome
Type         = client
Global       = false
Create Time  = 2023-01-03 18:41:06.447109751 +0000 UTC
Expiry Time  = <none>
Create Index = 10166
Modify Index = 10166
Policies     = krausen
Roles        = n/a
```

ACL Policies

- Control access to Nomad data and APIs (RBAC)
- Written in HCL (or JSON) and contains one or more rules
- Policies generally have the following dispositions:
  - `read` – allows read and list of Nomad resources
  - `write` – allows read and write of Nomad resources
  - `deny` – denies read or write – takes precedence over any other permission
  - `list` – list resources but not provide details
- Other rules also allow more fine-grained controls and capabilities

ACL Policies

Example

- Policy written in HCL
- Provides write to most resources in Nomad
- More aligned to a Nomad operator

```
namespace "default" {
  policy = "read"
}

node {
  policy = "write"
}

agent {
  policy = "write"
}

operator {
  policy = "write"
}

plugin {
  policy = "list"
}
```


Namespace Capabilities

Policy	Capabilities
read	list-jobs
	parse-job
	read-job
	csi-list-volume
	csi-read-volume
	list-scaling-policies
	read-scaling-policies
	read-job-scaling

Policy	Capabilities
write	list-jobs
	parse-job
	read-job
	submit-job
	dispatch-job
	read-logs
	read-fs
	alloc-exec
	alloc-lifecycle
	csi-write-volume
	csi-mount-volume
	list-scaling-policies
	read-scaling-policies
	read-job-scaling
	scale-job

Namespace Capabilities

```
namespace "default" {
  policy = "read"
  capabilities = ["submit-job", "read-logs", "alloc-exec", "scale-job"]
}

node {
  policy = "write"
}

plugin {
  policy = "list"
}
```

Provides read and four additional rights

```
namespace "default" {
  capabilities = ["submit-job", "read-logs", "alloc-exec", "scale-job"]
}
```

Only provides rights explicitly listed here

Nomad ACLs

- Interaction via CLI requires an ACL token to perform almost all operations
- There are a few ways you can provide the token:
  - `-token` flag on the CLI with the desired command to be executed
  - Setting the `NOMAD_TOKEN` environment variable

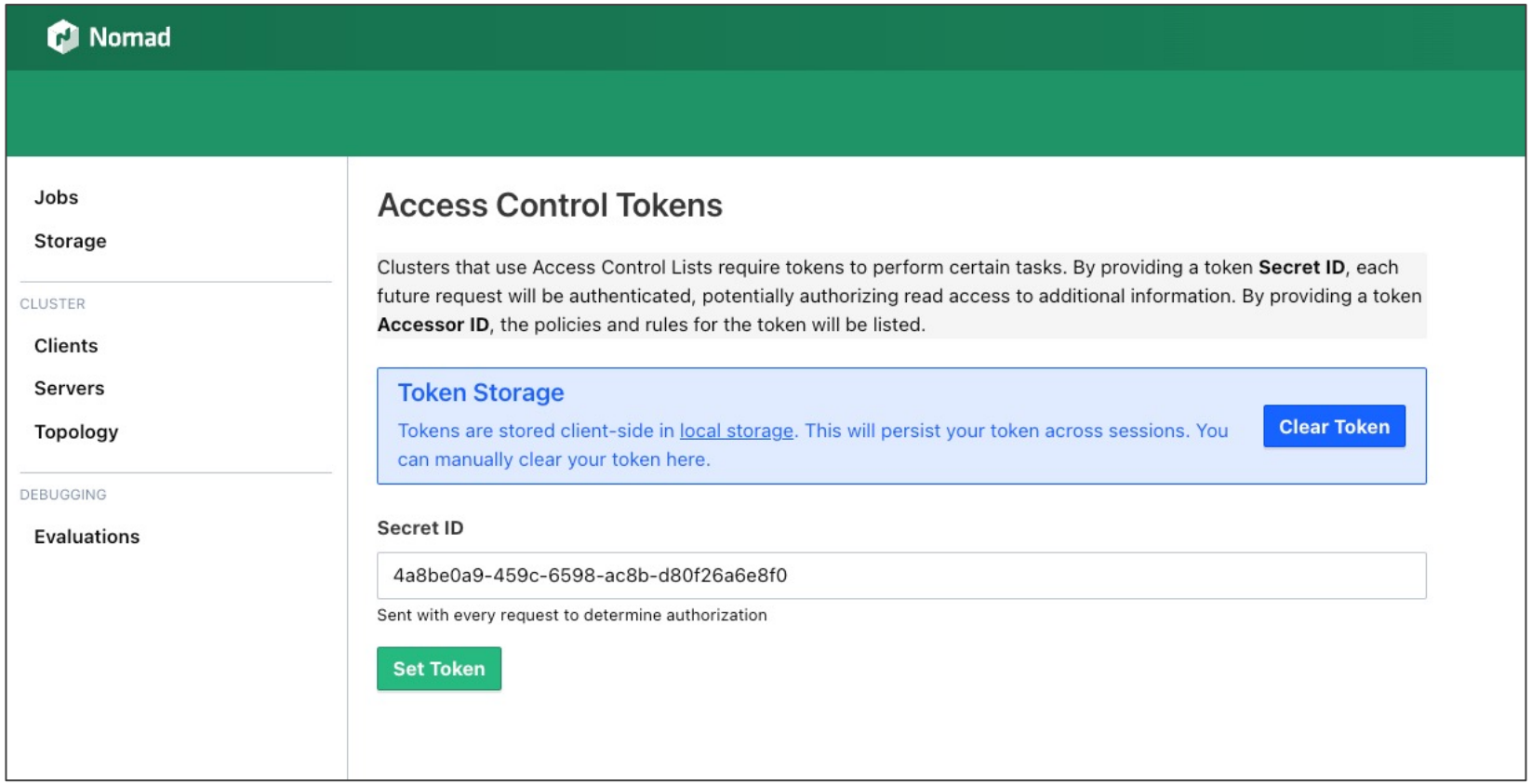
TERMINAL

```
# Use the -token flag for authentication
$ nomad job run webapp.nomad -token=4a8be0a9-459c-6598-ac8b-d80f26a6e8f0

# Set the NOMAD_TOKEN environment variable to authenticate
$ export NOMAD_TOKEN=4a8be0a9-459c-6598-ac8b-d80f26a6e8f0
$ nomad job run webapp.nomad
```

Nomad ACLs

Authenticate to the Nomad UI without exposing the token to the browser's history



DEMO

Securing Nomad with ACLs