# Managing Security in a Docker Swarm Cluster

**Nigel Brown**

@n_brownuk www.windsock.io

## Module Outline

Securing communication between nodes

Using and managing secrets in a cluster

Protecting the cluster with autolocking

Managing the availability of the cluster

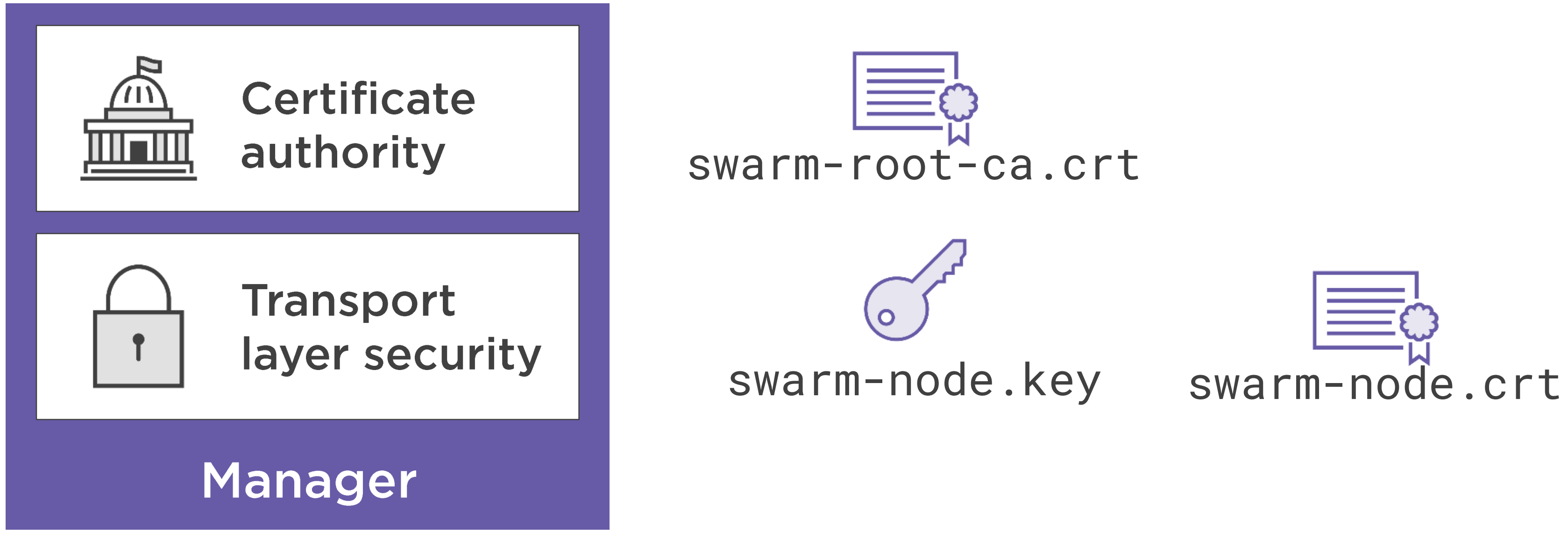
```
# Initialize a swarm cluster
$ docker swarm init --advertise-addr 192.168.99.101:2377


# Join a node to an existing cluster
$ docker swarm join --token <token> 192.168.99.101:2377
```

## Docker CLI Cluster Commands

**Initiate a swarm cluster on an existing Docker host**

**New nodes join the cluster using a secret called a token**

# Bootstrapping a Swarm Cluster



# Node Certificate

`O=vtbiuefnd0nitg284qnh8oy0w`

**Organization attribute contains the cluster ID the node belongs to**

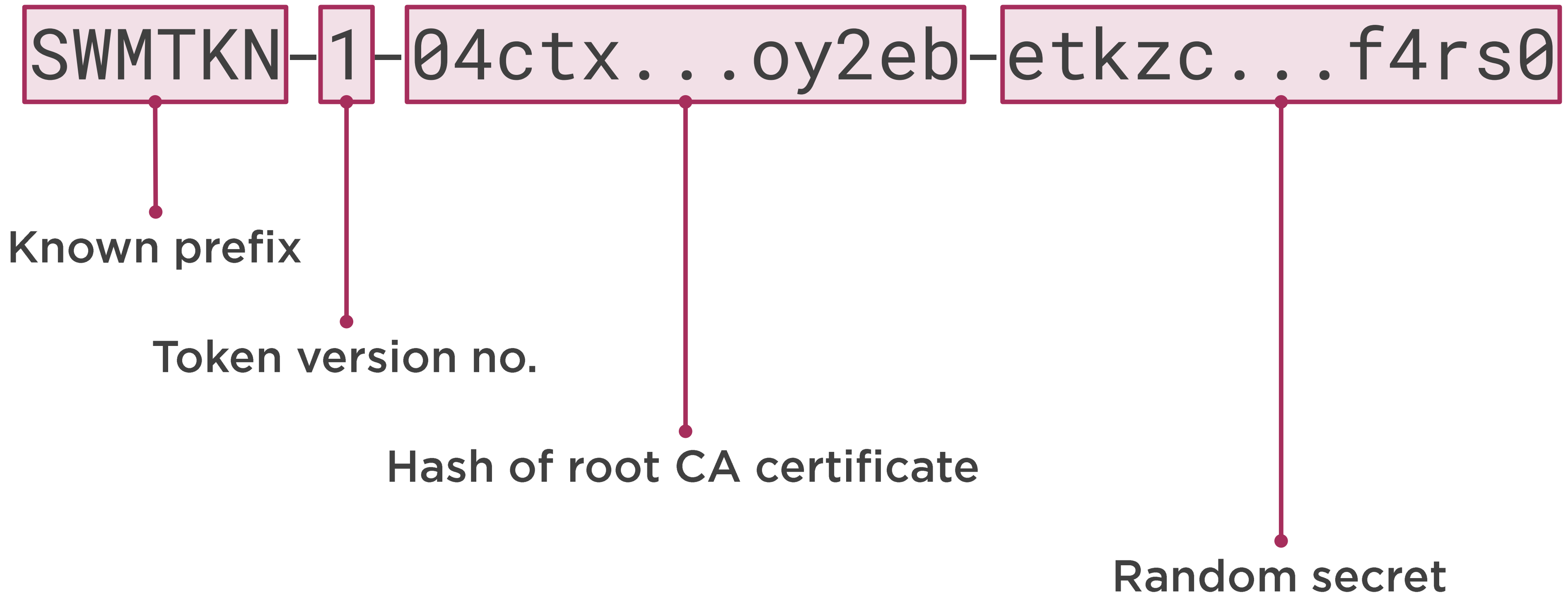
`OU=swarm-manager`

**Organizational Unit attribute contains the node’s role in the cluster**

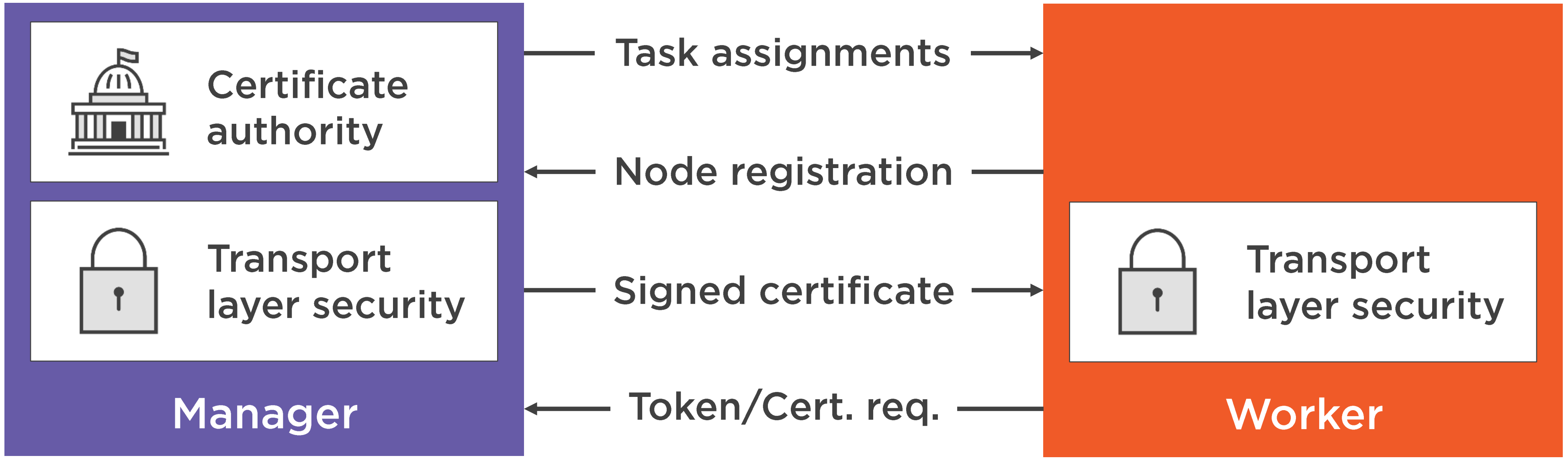
`CN=jlo1nxuq6f1hzmjs70el0y08r`

**Common name contains the node’s unique ID in the cluster**

# Join Token


SWMTKN-1-04ctx...oy2eb-etkzc...f4rs0
Known prefix
Token version no.
Hash of root CA certificate
Random secret

# Joining a Node



```
$ docker swarm ca --rotate

$ docker swarm ca --rotate --ca-cert <path> --ca-key <path>

$ docker swarm ca --rotate --ca-cert <path> \
--external-ca <protocol,url>
```

## Root CA Certificate Rotation

**The root CA certificate/key can be regenerated by Docker**

**The PKI artifacts can be supplied with explicit pathnames**

**A certificate signing endpoint can be specified**

```
# Manager or worker tokens can be rotated
$ docker swarm join-token --rotate manager|worker
```

## Join Token Rotation

**A compromised manager or token secret, necessitates token rotation**

## What Is a Secret?

Data item used for accessing service

Passwords, X.509 certificates, and keys

Proper management of secrets is imperative

# Secrets in a Swarm Cluster

Secrets are first-class objects, but can only exist in context of swarm

Secrets are stored in encrypted form, in the swarm cluster's Raft log

Secrets are associated with services, and only available as required

Secrets are made available using an in-memory filesystem mount

```
# Create secret on a manager node
$ echo cheese | docker secret create password -

# Create (or update) service and specify (or add) secret
$ docker service create --secret password ...

# Secret available inside the service task container
$ docker container exec b0b cat /run/secrets/password
cheese
```

## Creating and Using a Swarm Secret

**Secrets are created using the** `docker secret create` **CLI command**

**Secrets are associated with a service at creation or on update**

**The secret is available inside any of the service's task containers**

# Secret Rotation

**Immutable Secrets**

Secrets cannot be changed once they've been created

**Service Updates**

Secrets can be added or removed during service updates

## Encrypting the Encryption Key

A copy of the Raft log is stored on disk, in encrypted form

The encryption key for the Raft log is also stored on disk

If a manager node is compromised, the Raft log can be easily read

Swarm managers can be locked with an offline encryption key

```
# Autolocking is enabled at initiation or on update
$ docker swarm init --autolock

$ docker swarm unlock-key -q
SWMKEY-1-Fs3BJSKXK1F0tjyW8//ULVQ8gjtiyov+yk6zTeBSqFo

# After a restart, the cluster requires unlocking
$ docker swarm unlock
```

## Using Autolock in Swarm Clusters

**The `--autolock` config option is used to enable autolocking**

**Unlock key can be obtained from a manager node that is running**

**After a Docker daemon restart, the Raft log requires decrypting**

## Manager Node Consensus

Multiple manager nodes enable a cluster to be resilient to failure

Swarm manages state using Raft Consensus (https://bit.ly/1z0Obaw)

The cluster requires a quorum of functioning manager nodes

## Maintaining a Quorum

| No. of Managers | Quorum Required | Tolerated Failures |
|---|---|---|
| 1 | 1 | 0 |
| 2 | 2 | 0 |
| 3 | 2 | 1 |
| 4 | 3 | 1 |
| 5 | 3 | 2 |
| 6 | 4 | 2 |
| 7 | 4 | 3 |
| 8 | 5 | 3 |
| 9 | 5 | 4 |

```
# A manager node demotes the node
$ docker node demote <node>

# Now a worker, the node leaves the cluster
$ docker swarm leave

# Remove the node from the cluster, --force to force
$ docker node rm <node>
```

## Removing a Manager Node

Try and demote the node from a manager to a worker

The node should then elect to leave the cluster

After the node leaves, remove its registration, with force if necessary

A lost quorum renders a cluster incapable of performing any further management tasks

```
# Command should be run against a healthy manager node
$ docker swarm init --force-new-cluster
```

## Recovering from a Lost Quorum

A quorum loss may require the forcing of a new cluster

A forced cluster has a single manager node, but retains state

New manager nodes can then be added to the cluster

## Backing up a Swarm Cluster

Swarm data is located within the data root

If autolock enabled, take copy of unlock key

Stop the daemon during the backup

Perform backup using preferred method

# Recovering a Swarm Cluster

Restore a backup of the swarm directory

Start the daemon, and unlock if required

Re-initialize the swarm on a new node

Add new manager and worker nodes

## Module Summary

Swarm automatically secures intra-node communication

Secrets are a first class API object

Sensitive data can be protected using autolocking

Taking precautions to minimize node loss, helps to ensure cluster availability