Docker Certified Associate Training

Source: https://docs.docker.com

Security

Learning Objectives

By the end of this lesson, you will be able to:

- Implement Docker Security and Default Engine Security
- Describe the process of signing an image
- Create the UCP client bundles
- Illustrate the significance of Roles and Secrets

Docker Security

Docker Security

Docker security prevents a compromised container from consuming a large amount of resources for disrupting service or performing malicious activities.

The kernel's intrinsic security and support for namespaces and cgroups.

The vulnerable surface on the Docker daemon.

Things to consider when reviewing Docker Security:

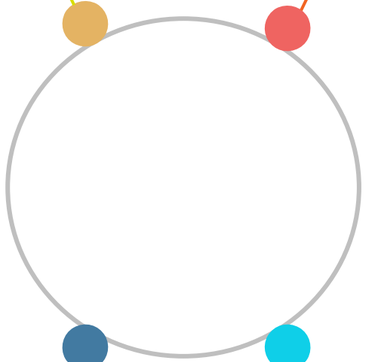
The container configuration profile's loopholes.

The kernel's "hardening" security features and how they interact with containers.

Default Engine Security

Configures the trust features of Docker to allow users to push and pull trusted images

Protects the Docker daemon socket and ensures only trusted Docker client connections



Has the right to access images in a registry

Sets up policies to secure the system calls in a container

Namespace

Docker creates namespaces in the container to provide the isolated workspace.

Docker Engine uses namespaces such as the following on Linux:

- **The pid namespace**: Isolates the process ID
- **The net namespace**: Manages network interfaces (**net**: Networking)
- **The ipc namespace**: Manages access to IPC resources (**ipc**: InterProcess Communication)
- **The mnt namespace**: Manages filesystem mount points (**mnt**: Mount)
- **The uts namespace**: Isolates kernel and version identifiers (**uts**: Unix Timesharing System)

Kernel Namespace

Docker containers are very similar to Linux containers, and they have similar security features.

- Namespaces provide the first and most straightforward form of isolation, processes running within a container cannot see, and even less affect, processes running in another container, or in the host system.
- Each container also gets its own network stack, meaning that a container doesn't get privileged access to the sockets or interfaces of another container.

Control Groups

Docker Engine on Linux relies on a technology called control groups (cgroups). They are a key component of Linux Containers.

Key features of control groups:

- Limit an application to a particular collection of resources
- Allow Docker Engine to share available container hardware resources and enforce limitations and constraints as an option
  - For example, the user can restrict the available space to a specific container
- Implement resource accounting

Control Groups

Key features of control groups (contd.):

- Provide many useful metrics
- Ensure that each container gets its fair share of memory, CPU, and disk I/O
- Ensure that a single container cannot bring the system down by exhausting one of the resources

Docker Daemon

Docker Daemon Attack Surface

It helps the Docker to allow the user to share a directory between the Docker host and a guest container; and it also allows the user to do so without limiting the access rights of the container.

Additional features of Docker Daemon:

- Running containers (and applications) with Docker implies running the Docker daemon.
- This daemon requires root privileges unless you opt-in to rootless mode (experimental). The user should be aware of some important details:
  - Only trusted users should be allowed to control your Docker daemon
  - The daemon is potentially vulnerable to inputs, such as image loading from either disk with docker load or from the network with docker pull

Linux Kernel Capabilities

Linux Kernel Capabilities

Docker runs the containers with certain restricted capabilities by default. This means the root capabilities are not provided to all processes operating inside a container.

For instance it is possible to:

- deny all **mount** operations
- deny access to raw sockets
- deny access to some filesystem operations, like creating new device nodes, changing the owner of files, or altering attributes
- deny module loading

Docker Content Trust

Docker Content Trust

Docker Content Trust (DCT) provides the ability to use digital signatures for data from remote Docker registries that are sent and received.

Image Tags and DCT:

An individual image record has the following identifier:
[REGISTRY_HOST[:REGISTRY_PORT]/]REPOSITORY[:TAG]

- A particular image **REPOSITORY** can have multiple tags.
- DCT is associated with the **TAG** portion of an image.

Docker Content Trust

Docker Content Trust Keys

Trust for an image tag is managed using signing keys. A key set is created when an operation using DCT is first invoked. A key set consists of the following classes of keys:

- An offline key that is the root of DCT for an image tag
- Repository or tagging keys that sign tags
- Server-managed keys, such as the timestamp key, which provides freshness security guarantees for the repository

Docker Content Trust

Sign Images with Docker Content Trust

- The user can use the **$docker** trust command syntax to sign and push a container image within the Docker CLI.
- A prerequisite for signing an image is a Docker Registry with an attached Notary server like the Docker Hub or Docker Trusted Registry.
- The user will need a delegation key pair to sign a Docker File.

Docker Content Trust

Runtime Enforcement with DCT

Docker Content Trust within the Docker Enterprise Engine prevents a user from using a container image from an unknown source, as well as prevents a user from building a container image from an unknown source.

Engine Signature Verification prevents the following:

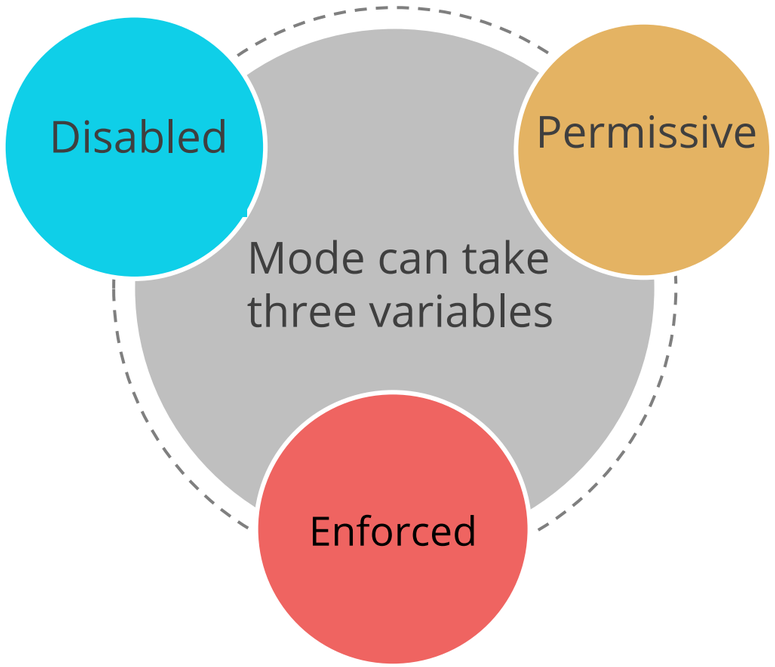
- $ docker container run of an unsigned or altered image.
- $ docker pull of an unsigned or altered image.
- $docker build where there is no FROM image signed or scratched.

Docker Content Trust

Enabling DCT within the Docker Enterprise Engine

DCT is controlled by the Docker Engine's configuration file.

The content-trust flag is based on a mode variable instructing the engine whether to implement signed images. The trustpinning variable tells the engine what sources to trust.



Docker Content Trust Signature Verification

It is a feature that allows the Docker Engine to run signed images.

How is it implemented?

- Built directly into the dockerd binary
- Configured in the Dockerd configuration file

Docker Content Trust Signature Verification

Advantages

- Allows to pull out and run repositories signed with a user-specified root key
- Trustpinning can be configured in daemon.json to allow this function

- Provides administrators with more information to implement and perform image signature validation with the CLI

Sign an Image

Sign an Image

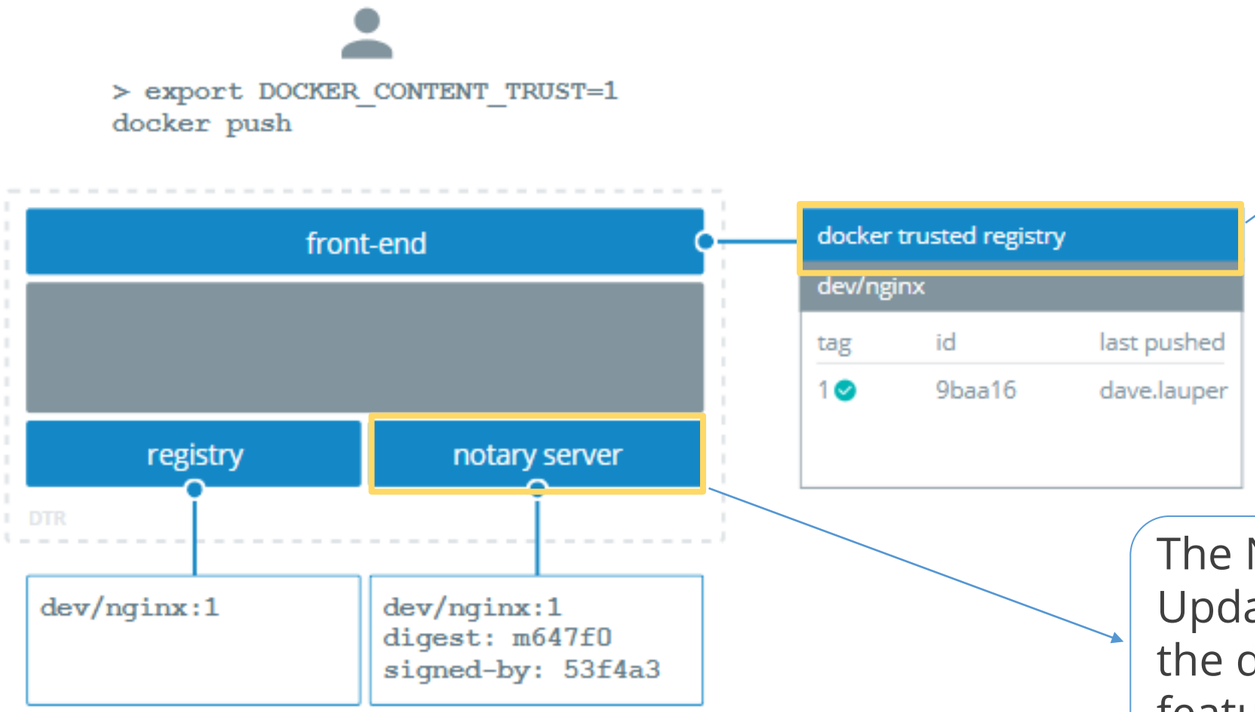
The user can configure the Docker CLI client to sign the images that the user pushes to DTR. This allows whoever pulls the image to validate if they are getting the image that is created, or a forged one.

To sign an image, the user can run:

export DOCKER_CONTENT_TRUST=1
docker push <dtr-domain>/<repository>/<image>:<tag>

The above command pushes the image to DTR and creates trust metadata. It also creates public and private key pairs to sign the trust metadata and push that metadata to the Notary Server internal to DTR.

Sign an Image



Docker Trusted Registry (DTR) is the enterprise-grade image storage solution from Docker.

The Notary server manages JSON formatted TUF (The Update Framework) metadata for Notary clients and the docker command line tool's Docker Content Trust features.

Source: https://docs.docker.com/v17.09/datacenter/dtr/2.2/guides/user/manage-images/sign-images/

Sign Images that UCP Can Trust

The user can sign the DTR images now, but UCP (Universal Control Panel) won't trust them because it can't tie the private key, which the user will use to sign the images to the UCP account.

To sign images in a way that UCP trusts them, the user needs to:

- Configure the Notary client
- Initialize trust metadata for the repository
- Delegate signing to the keys in the UCP client bundle

Vulnerabilities

Docker image security best practices:

- Prefer minimal base images
- Sign and verify images to mitigate MITM attack
- Find, fix, and monitor for open source vulnerabilities
- Don’t leak sensitive information to docker images
- Create a dedicated user with minimal permissions
- Use fixed tags for immutability
- Use **COPY** instead of **ADD**
- Use labels for metadata
- Use multi-stage builds for small secure images
- Use a linter

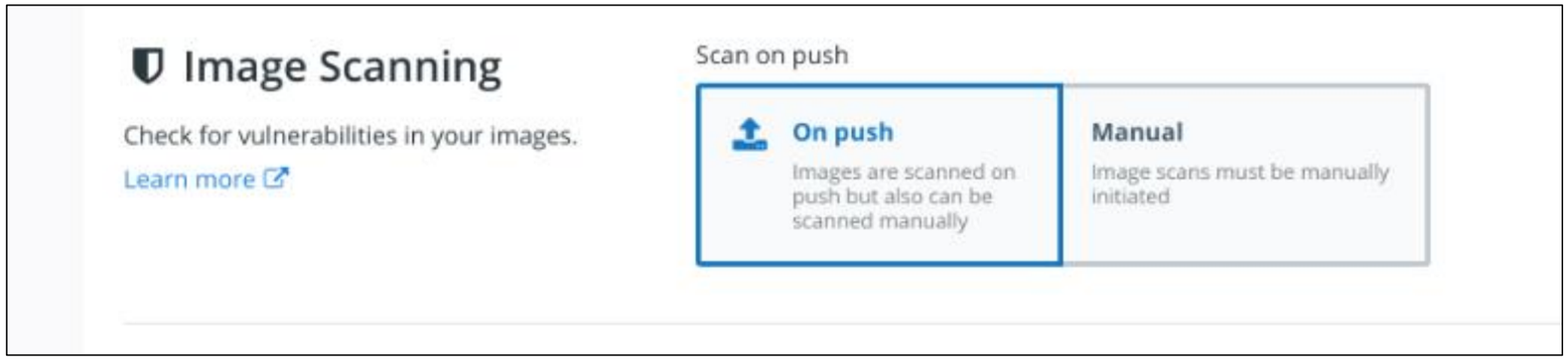
Scan Images for Vulnerabilities

Docker Trusted Registry can scan images in the repositories using Docker Security Scanning, to verify that they are free from known security vulnerabilities or exposures.

Docker Security Scan process:

- DTR scans both Linux and Windows images, but by default Docker doesn't push foreign image layers for Windows images so that DTR is not able to scan them.
- If the user wants DTR to scan the Windows images, the user will have to configure Docker to always push non-foreign layers and scan them.

Scan Images for Vulnerabilities



Security scan on push

Docker Security Scanning runs automatically on docker push to an image repository by default.

Manual Scanning

Manual Scanning can start scanning images manually in repositories to which the user has write access.

Source: https://docs.docker.com/ee/dtr/user/manage-images/scan-images-for-vulnerabilities/#manual-scanning

Client Bundle

Client Bundle

A client bundle is a group of certificates downloadable directly from the Docker Universal Control Plane (UCP) user interface within the admin section for “My Profile.”

Admin user certificate bundles: Allow the running of docker commands on the Docker Engine of any node

User certificate bundles: Only allow running docker commands through a UCP controller node

UCP Client Bundles

Features of UCP

- Universal control plane overview
- Centralized cluster management
- Deploy, manage, and monitor
- Built-in security and access control
- Use the Docker CLI client

Docker Universal Control Plane (UCP) is the enterprise-grade cluster management solution from Docker.

Features of UCP

- Universal control plane overview
- Centralized cluster management
- Deploy, manage, and monitor
- Built-in security and access control
- Use the Docker CLI client

Centralized cluster management helps the user to join up to thousands of physical or virtual machines together with Docker to create a container cluster that enables the user to deploy their applications on a scale.

Features of UCP

- Universal control plane overview
- Centralized cluster management
- **Deploy, manage, and monitor**
- Built-in security and access control
- Use the Docker CLI client

The user can deploy, manage, and monitor all the computing resources that are available from a centralized location, such as clusters, volumes, and networks with Docker UCP.

Features of UCP

- Universal control plane overview
- Centralized cluster management
- Deploy, manage, and monitor
- Built-in security and access control
- Use the Docker CLI client

Docker UCP has an integrated authentication mechanism of its own and is integrated with LDAP services. It also has Roll-Based Access Control (RBAC), which enables the user to control who can access and make changes to their cluster and applications.

Features of UCP

- Universal control plane overview
- Centralized cluster management
- Deploy, manage, and monitor
- Built-in security and access control
- Use the Docker CLI client

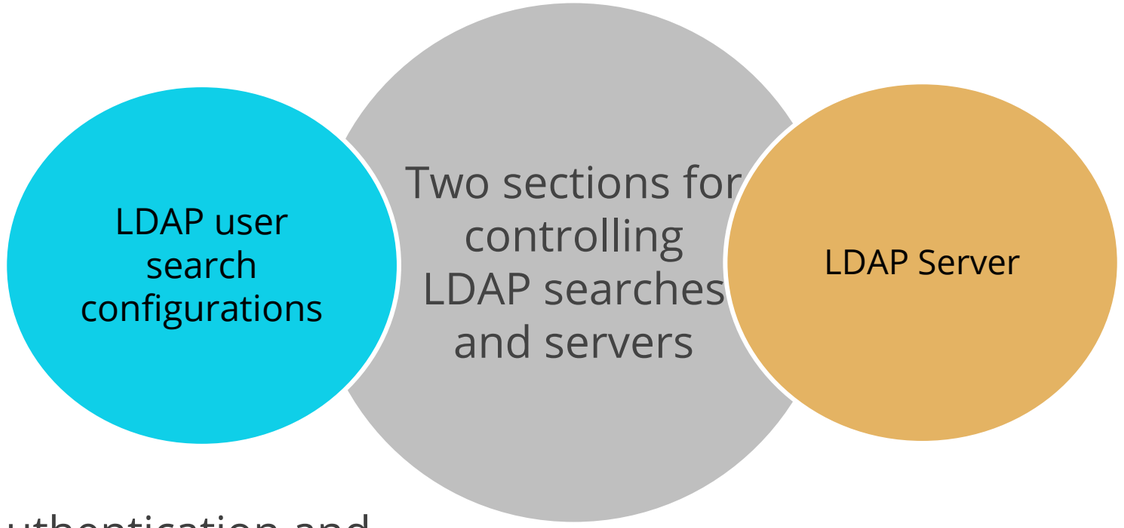
The user can continue to deploy and manage their applications including the Docker CLI client because UCP exposes the standard Docker API.

Integrate UCP with LDAP

Docker UCP integrates with LDAP (Lightweight Directory Access Protocol) directory services, so that the user can manage users and groups from their organization's directory, and it will automatically propagate that information to UCP and DTR.

- The user can control how UCP integrates with LDAP by creating user searches.
- The user can specify multiple search configurations.
- The user can specify multiple LDAP servers for integration.
- The user can start searches with the **Base DN**, which is the distinguished name of the node in the LDAP directory tree where the search starts looking for users.

Integrate UCP with LDAP

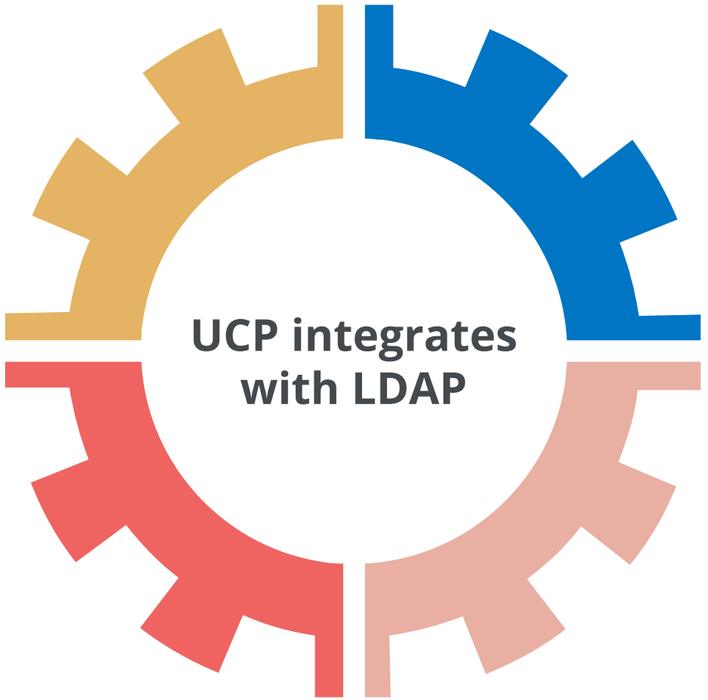


Authentication and Authorization section where a user specifies search parameters

Section where a user specifies the URL of an LDAP server

Integrate UCP with LDAP

UCP creates a set of search results by iterating over each of the user search configs, in the order that the user specifies.



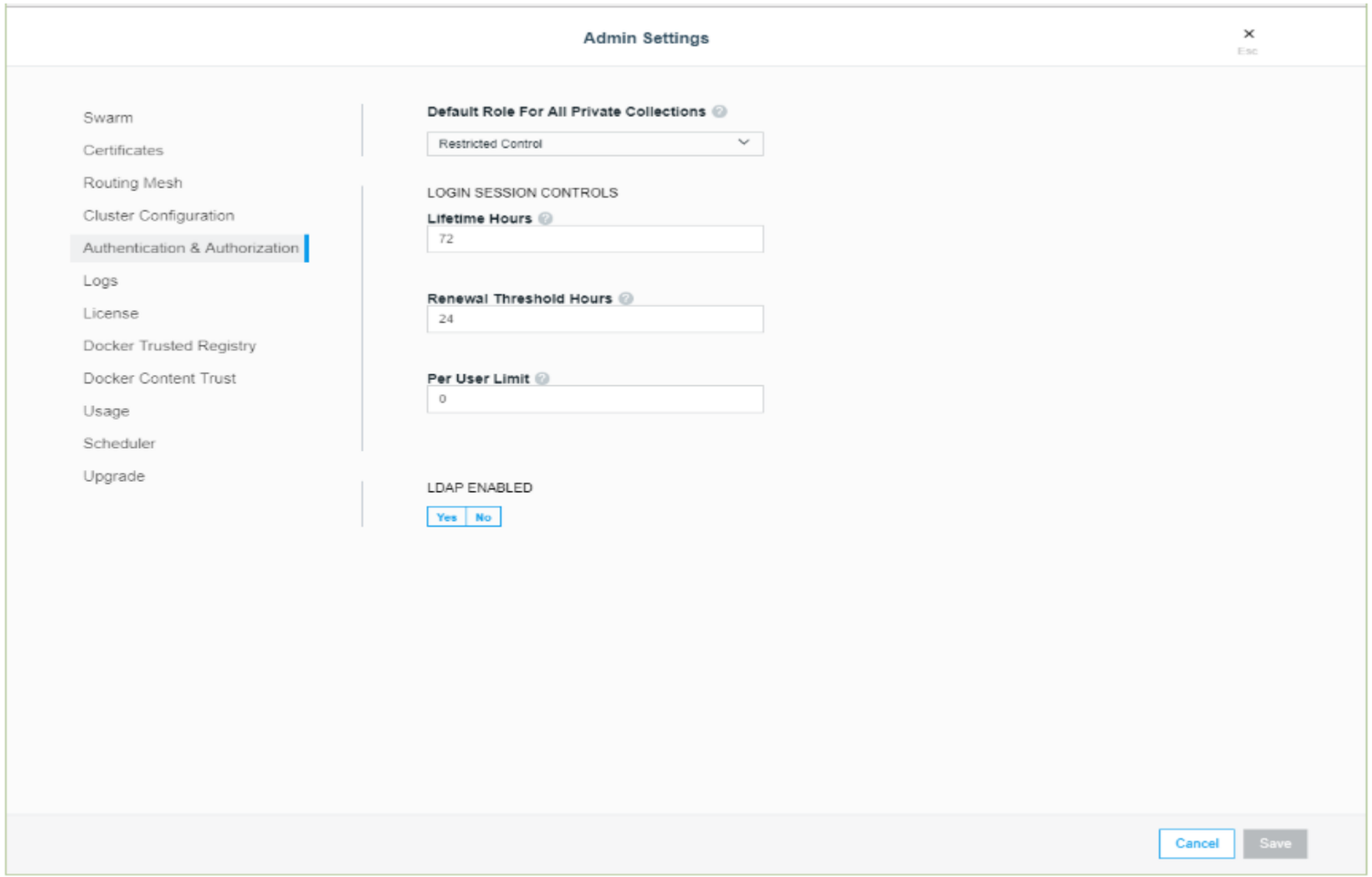
UCP chooses an LDAP server from the list of domain servers by considering the Base DN from the user search config and selecting the domain server that has the longest domain suffix match.

UCP combines the search results into a list of users and creates UCP accounts for them.

UCP uses the default domain server if no domain server has a domain suffix that matches the Base DN from the search configuration.

Configure the LDAP Integration

To configure UCP to create and authenticate users by using an LDAP directory, go to the UCP web UI, navigate to the **Admin Settings** page, and click **Authentication & Authorization** to select the method used to create and authenticate users.



Source: https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/external-auth/

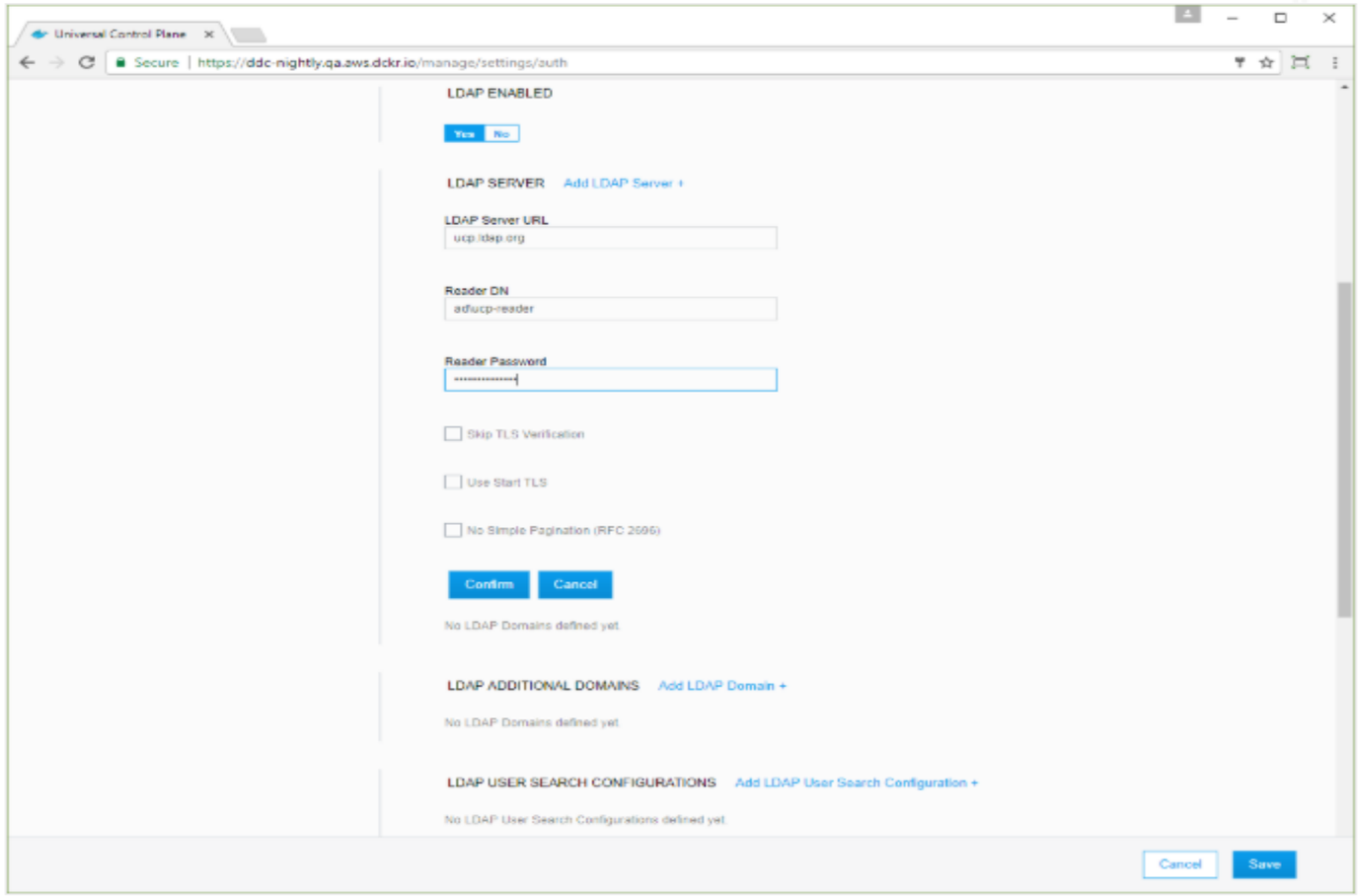
LDAP Enabled

Default role for all private collections:

Use this setting to change the default permissions of new users. Click the dropdown to select the permission level that UCP assigns by default to the private collections of new users.

LDAP enabled:

Click **Yes** to enable integrating UCP users and teams with LDAP servers.



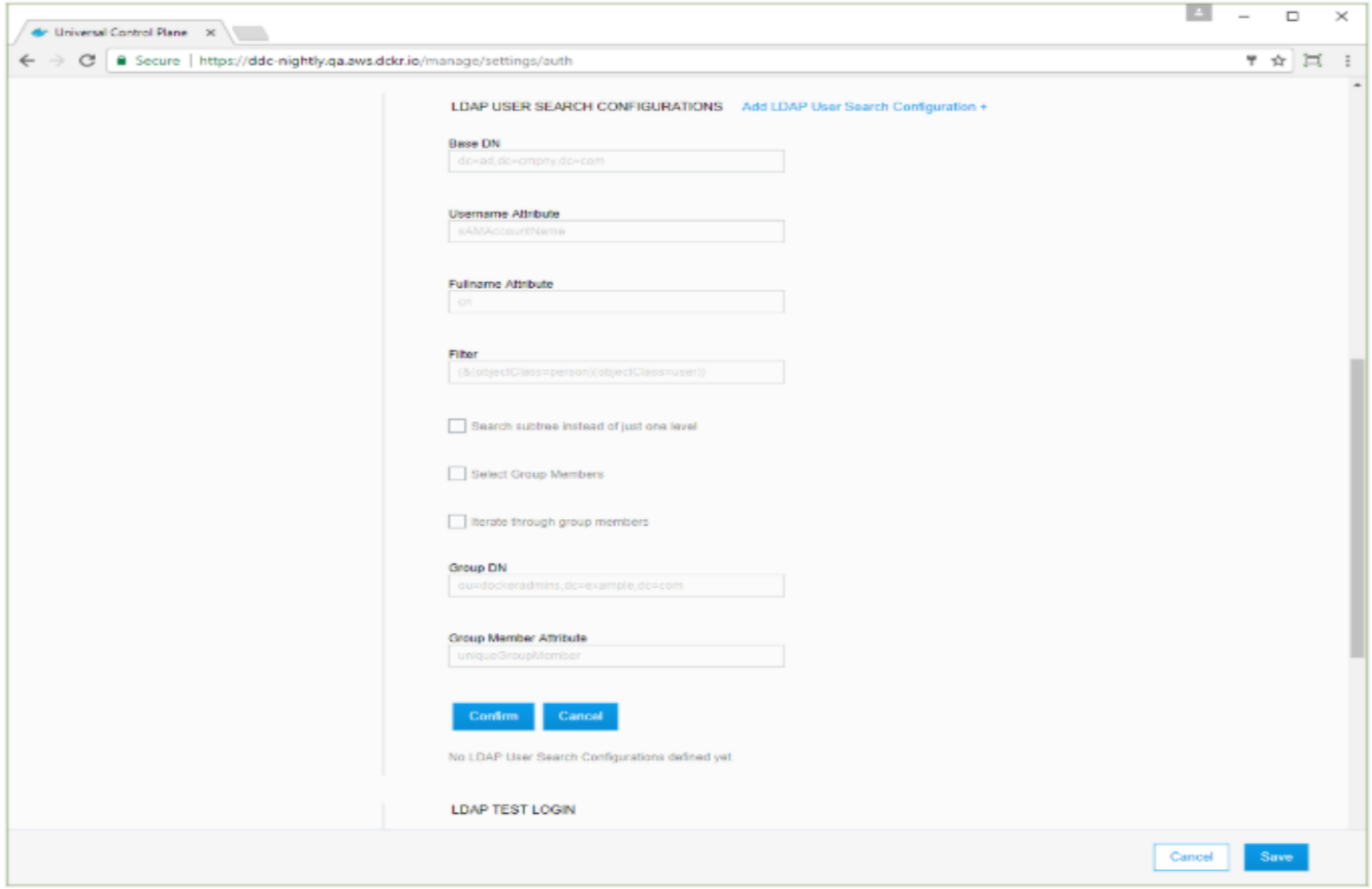
Source: https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/external-auth/

LDAP Server

Field	Description
LDAP server URL	The URL where the LDAP server can be reached.
Reader DN	The distinguished LDAP account name used in the LDAP server to search for entries. This should be a read-only client of LDAP as a best practice.
Reader password	The password of the account used for searching entries in the LDAP server.
Use Start TLS	The connection can be authenticated/encrypted only after connecting to the LDAP server over TCP.
Skip TLS verification	The LDAP server certificate will be verified using TLS.
No simple pagination	The LDAP server doesn’t support pagination.
Just-in-Time User Provisioning	The user can create user accounts only when users log in for the first time.

LDAP User Search Configurations

To configure more user search queries, click **Add LDAP User Search Configuration** again.



Source: https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/external-auth/

LDAP User Search Configurations

Field	Description
Base DN	The distinguished name of the node in the directory tree where the search should start looking for users.
Username attribute	The LDAP attribute to use as username on UCP.
Full name attribute	The LDAP attribute to use as the user's full name for display purposes.
Filter	The LDAP search filter used to find users.
Search subtree instead of just one level	The LDAP can be performed by searching on a single level of the LDAP tree, or searching through the full LDAP tree starting at the Base DN.
Select Group Members	This feature is helpful if the LDAP server does not support **memberOf** search filters.

LDAP User Search Configurations

Field	Description
Iterate through group members	This option searches for users by first iterating over the target group's membership, making a separate LDAP query for each member if **Select Group Members** is selected.
Group DN	This specifies the distinguished name of the group from which to select users if **Select Group Members** is selected.
Group Member Attribute	The value of this group attribute corresponds to the distinguished names of the members of the group if **Select Group Members** is selected.

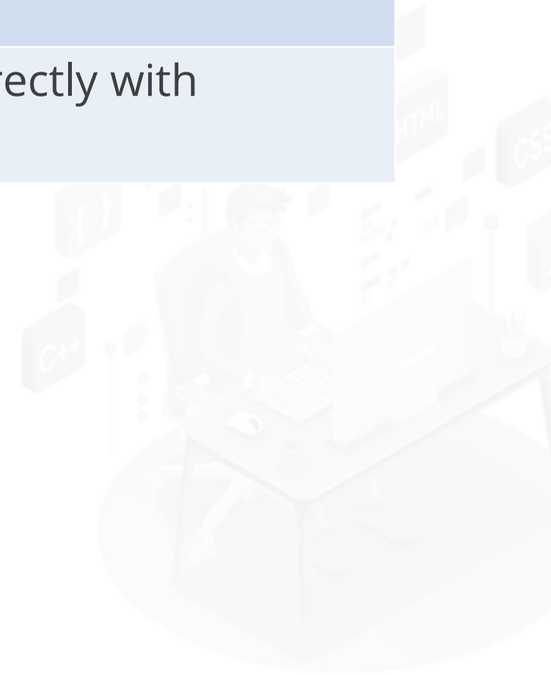
LDAP Test Login

Field	Description
Username	An LDAP username for testing authentication to this application. This value corresponds with the **Username Attribute** specified in the **LDAP user search configurations** section.
Password	The user's password is used to authenticate (BIND) to the directory server.

Before the user saves the configuration changes, test that the integration is configured correctly. To do this, provide the LDAP user credentials and click on the Test button.

LDAP Sync Configuration

Field	Description
Sync interval	This interval between UCP and the LDAP server helps users to synchronize in hours.
Enable sync of admin users	This option specifies that system admins should be synced directly with members of a group in the organization’s LDAP directory.



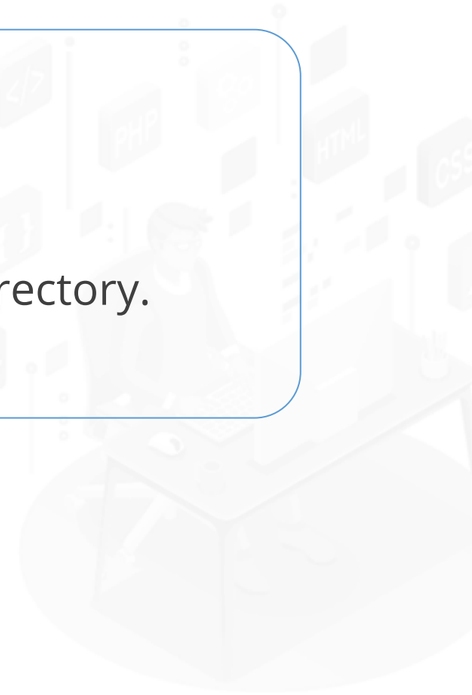
LDAP Sync Configuration

When a user is removed from LDAP, the effect on the user's UCP account depends on the Just-in-Time User Provisioning setting:

- Just-in-Time User Provisioning is false: Users deleted from LDAP become inactive in UCP after the next LDAP synchronization runs.
- Just-in-Time User Provisioning is true: Users deleted from LDAP can't authenticate, but their UCP accounts remain active.

LDAP Sync Configuration

- UCP saves a minimum amount of user data required to operate.
- UCP does not store any additional data from the directory server.
- UCP enables syncing teams with a search query or group in the organization’s LDAP directory.



Create UCP Client Bundles

Problem Statement: Your manager has asked you to create UCP client bundles that help run Docker commands on a UCP node.

Steps to Perform:

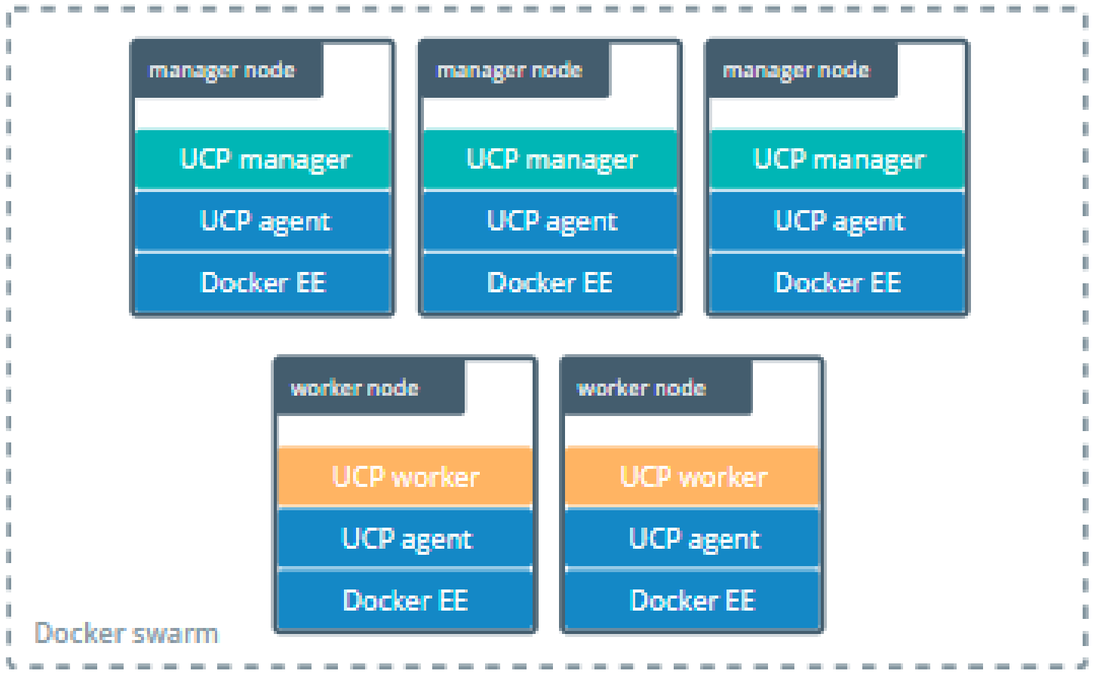
1. Sign in to Docker UCP with your admin credentials and navigate to *My Profile.*
2. Click on *New Client Bundle* dropdown and select *Generate Client Bundle* to download the certificate bundle.
3. Unzip the *client-bundle.zip* file and start the client certificates.
4. Use Docker CLI with client certificates.

ASSISTED PRACTICE

UCP: Manager and Worker Nodes

UCP: Manager and Worker Nodes

A swarm is a collection of nodes that are in the same Docker cluster. Nodes in a Docker swarm operate in one of two nodes: Manager or Worker.



Source: https://docs.docker.com/datacenter/ucp/2.2/guides/architecture/#under-the-hood

UCP: Manager and Worker Nodes

Managers

The **ucp-agent** service starts to serve all UCP components automatically, including the UCP Web UI and data stores used by UCP. This is accomplished by the **ucp-agent** by deploying several containers on the node. UCP automatically becomes highly available and fault tolerant by promoting a node to the manager.

Workers

The **ucp-agent** service starts to provide a proxy network on the worker nodes, ensuring that only authorized users and other UCP services can execute Docker commands in that node. The **ucp-agent** deploys a subset of containers on worker nodes.

UCP Components in Manager Nodes

UCP Component	Description
ucp-agent	The running service monitors the node and ensures the right UCP services are present.
ucp-reconcile	The ucp-agent detects that the node is not running the right UCP components. The ucp-reconcile container starts to converge the node to its desired state.
ucp-auth-api	The centralized service for identity and authentication used by UCP and DTR.
ucp-auth-store	The service stores authentication configurations and data for users, organizations, and teams.
ucp-auth-worker	The service performs scheduled LDAP synchronizations and cleans authentication and authorization data.
ucp-client-root-ca	A certificate authority to sign client bundles.
ucp-cluster-root-ca	A certificate authority used for TLS communication between UCP components.

UCP Components in Manager Nodes

UCP Component	Description
ucp-controller	It is a UCP web server.
ucp-dsinfo	It helps Docker system information collection script to assist with troubleshooting.
ucp-kv	It is used to store the UCP configurations. It can't be used in the applications, since it's for internal use only.
ucp-metrics	It is used to collect and process metrics for a node, like the disk space available.
ucp-proxy	It is a TLS proxy. It allows secure access to the local Docker Engine to UCP components.
ucp-swarm-manager	It is used to provide backwards-compatibility with Docker Swarm.

UCP Components in Worker Nodes

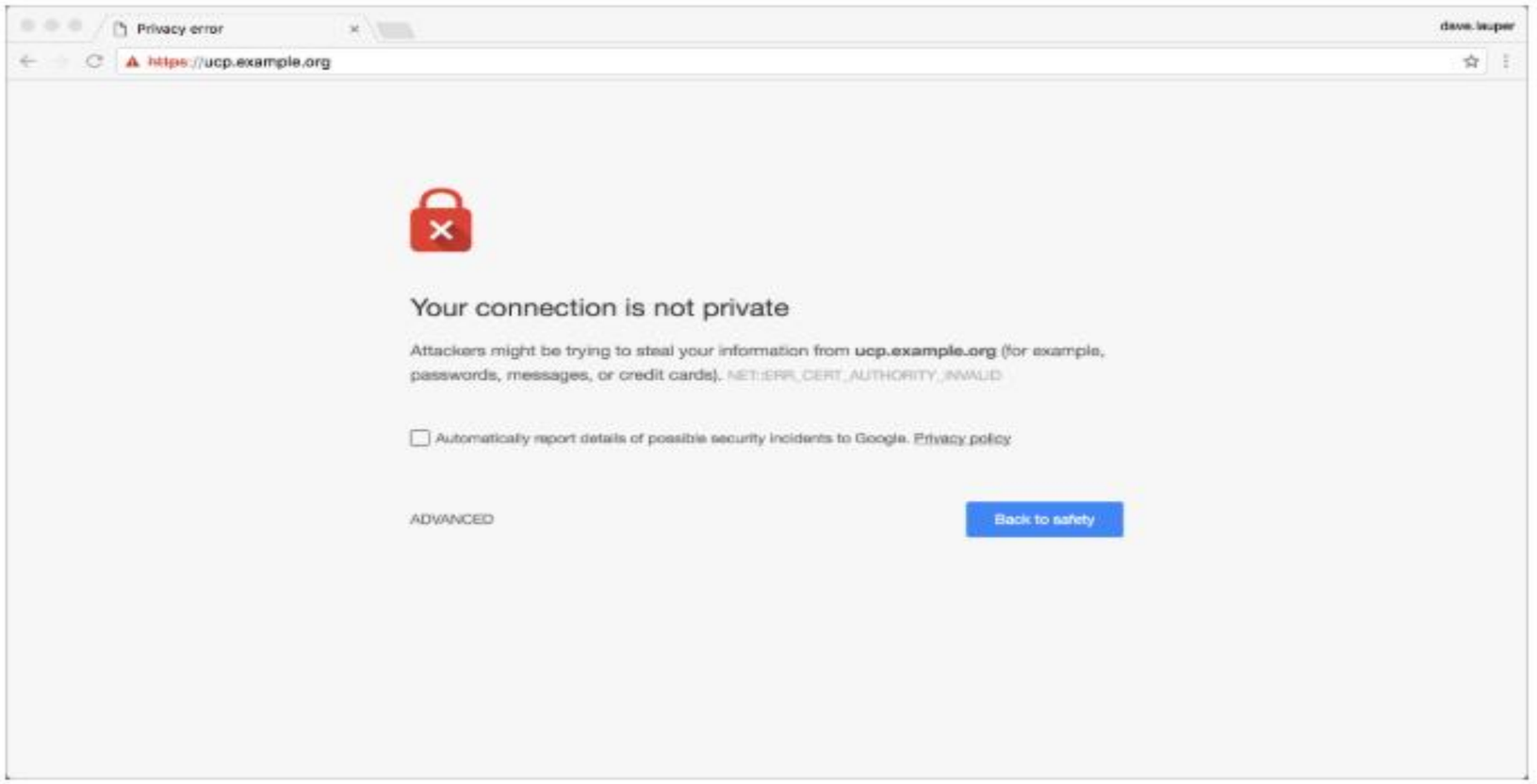
UCP Component	Description
ucp-agent	It is the service that runs and monitors the node and ensures the right UCP services.
ucp-dsinfo	It is the Docker system information collection script that assists with troubleshooting.
ucp-reconcile	It is the ucp-agent that detects that the node is not running the right UCP components. The ucp-reconcile container starts to converge the node to its desired state.
ucp-proxy	It is a TLS proxy. It allows secure access to the local Docker Engine to UCP components.

Configuration of Certificates

External Certificates with UCP

Use TLS Certificates

To ensure that all communications between clients and UCP are encrypted, all UCP services are exposed using HTTPS.

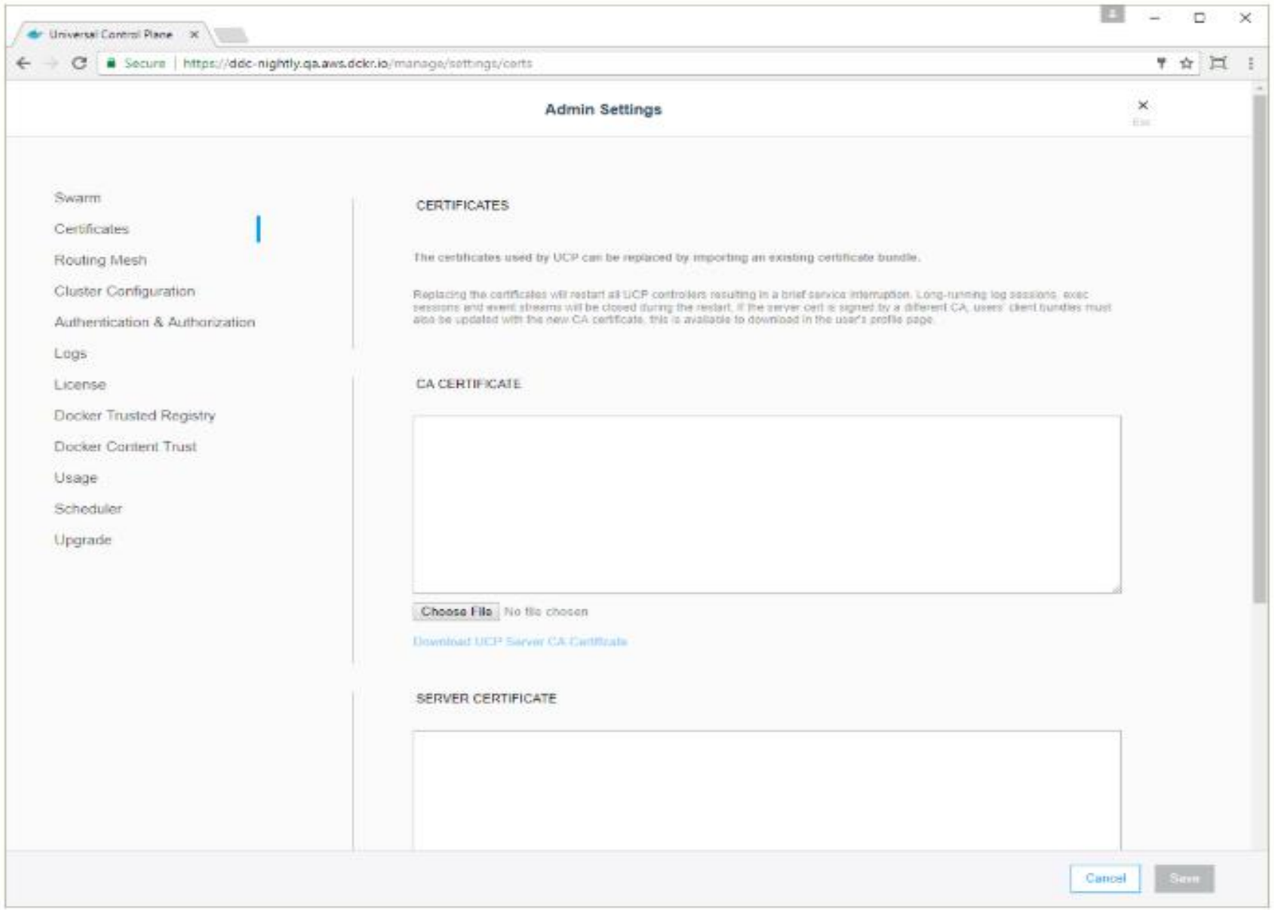


Source: https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/use-your-own-tls-certificates/

External Certificates with UCP

Configure UCP to use TLS certificates and keys

In the UCP web UI, log in with administrator credentials and navigate to the **Admin Settings** page. In the left pane, click **Certificates**.

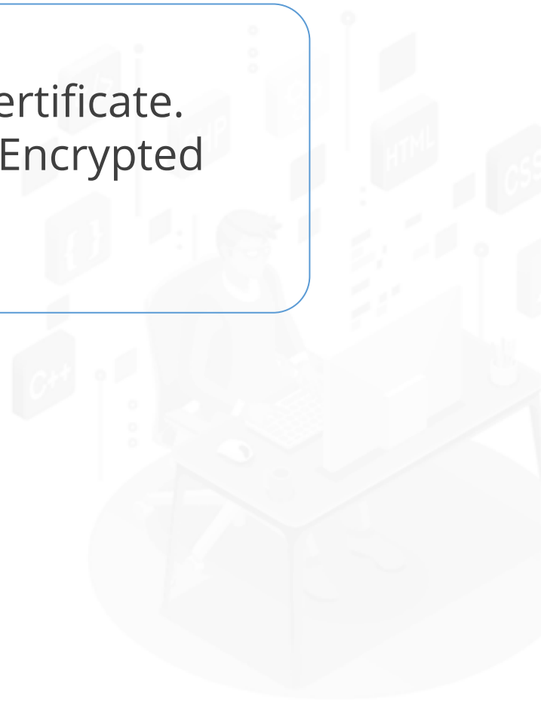


Source: https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/use-your-own-tls-certificates/

External Certificates with UCP

Upload the following certificates and keys:

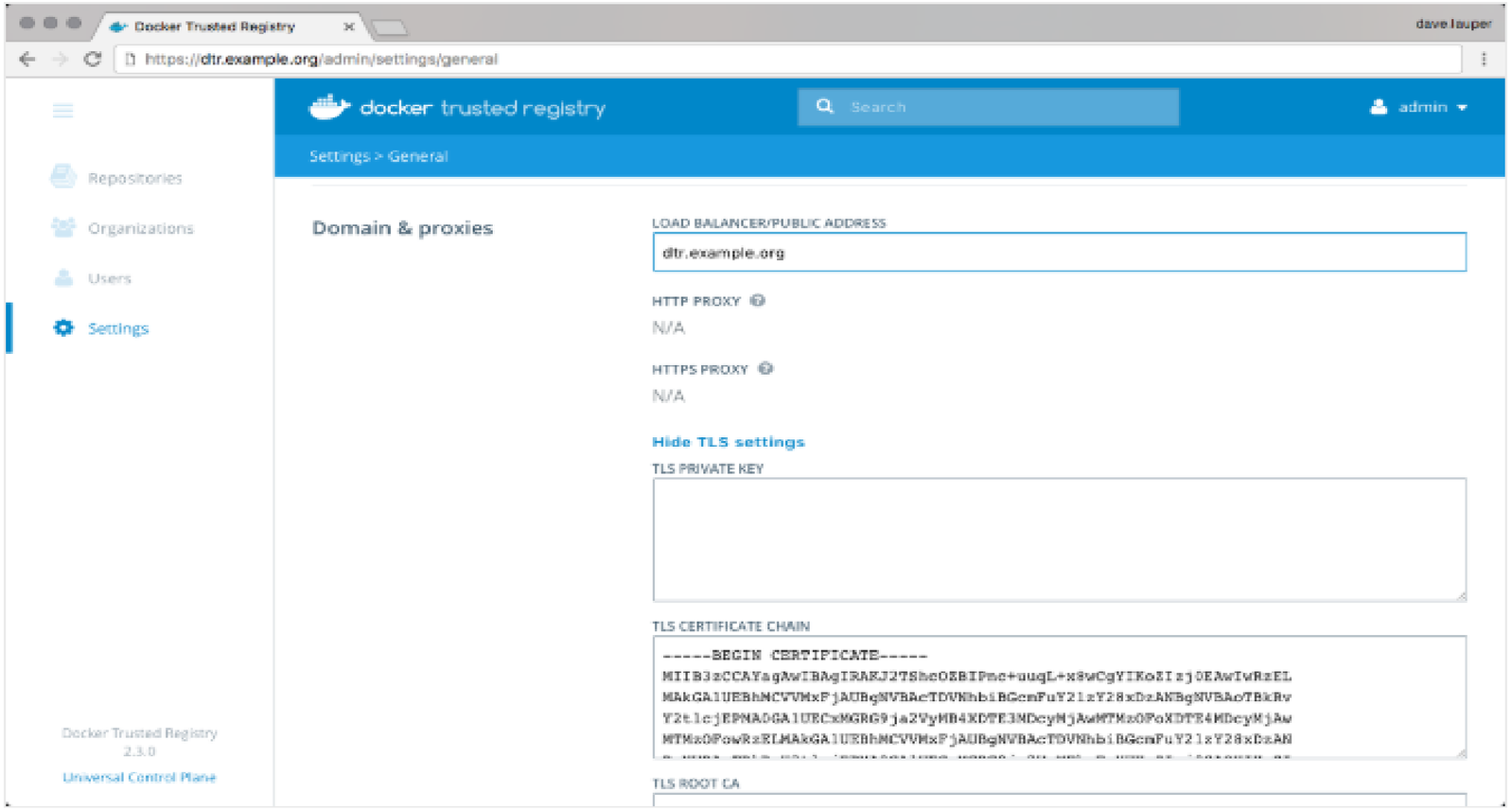
- A **ca.pem** file with the root CA public certificate.
- A **cert.pem** file containing the domain TLS certificate and any intermediate public certificate.
- A **key.pem** file with TLS private key. Make sure it is not encrypted with a password. Encrypted keys should have ENCRYPTED in the first line.



External Certificates with DTR

Replace the server certificates

To configure DTR to use the certificates and keys, go to the **DTR web UI**, navigate to the **Settings** page, and scroll down to the **Domain** section.

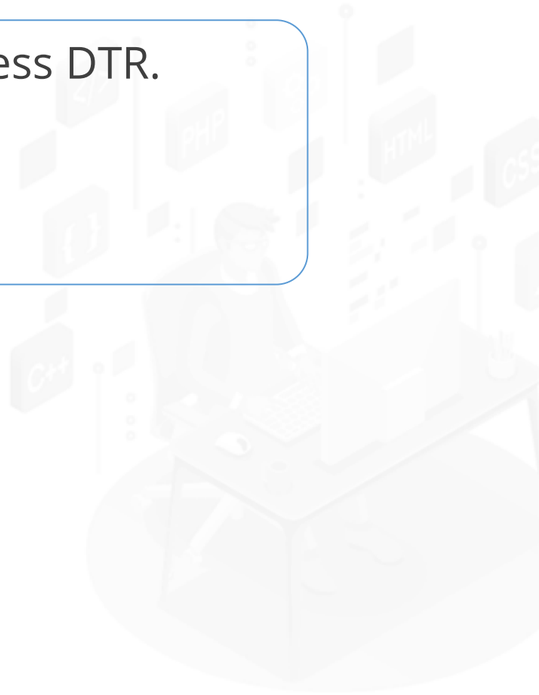


Source: https://docs.docker.com/datacenter/dtr/2.3/guides/admin/configure/use-your-own-tls-certificates/

External Certificates with DTR

Upload the following certificates and keys:

- **Load balancer/public address**: This is the domain name client that will help to access DTR.
- **TLS certificate**: This is the server certificate.
- **TLS private key**: This is the server private key.
- **TLS CA**: This is the root CA public certificate.



Configuration of Certificates

Understanding the Configuration

A custom certificate is configured by creating a directory under **/etc/docker/certs.d** using the same name as the registry's hostname, such as **localhost**. All ***.crt** files are added to this directory as CA roots.

The presence of one or more **<filename>.key/cert** pairs in Docker indicates that there are custom certificates required for access to the desired repository.

Configuration of Certificates

The following illustrates a configuration with custom certificates:

```
/etc/docker/certs.d/          <-- Certificate directory
    └── localhost:5000        <-- Hostname:port
        ├── client.cert       <-- Client certificate
        ├── client.key        <-- Client key
        └── ca.crt            <-- Certificate authority that signed the registry certificate
```

Configuration of Certificates

Create the client certificates

Use OpenSSL's **genrsa** and **req** commands to first generate an RSA key and then use the key to create the certificate.

```
$ openssl genrsa -out client.key 4096
$ openssl req -new -x509 -text -key client.key -out client.cert
```

Note: These TLS commands only generate a working set of certificates on Linux. The version of OpenSSL in macOS is incompatible with the type of certificate Docker requires.

Configuration of Certificates

Troubleshooting tips

The Docker daemon interprets **.crt** files as CA certificates and **.cert** files as client certificates. The Docker daemon logs the following error message if a CA certificate is accidentally given the **.cert** extension instead of the correct **.crt** extension:

> Missing key KEY_NAME for client certificate CERT_NAME. CA certificates should use the extension **.crt**.

Configuration of Certificates

If the Docker registry is accessed without a port number, do not add the port to the directory name. The following shows the configuration for a registry on default port 443 which is accessed with docker login my-https.registry.example.com:

```
/etc/docker/certs.d/
    └── my-https.registry.example.com        <-- Hostname without port
        ├── client.cert
        ├── client.key
        └── ca.crt
```

Swarm Security and TLS

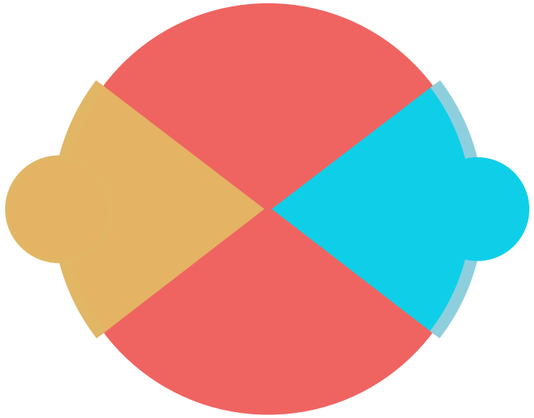
Swarm Security

- The swarm mode public key infrastructure (PKI) system built into Docker makes it simple to securely deploy a container orchestration system.
- The nodes in a swarm use mutual Transport Layer Security (TLS) to authenticate, authorize, and encrypt the communications with other nodes in the swarm.
- When the user creates a swarm by running **docker swarm init**, Docker designates itself as a manager node.
- The user can specify their own externally-generated root CA, using the **--external-ca** flag of the **docker swarm init** command.

Swarm Security

The manager node also generates two tokens to use when the user joins additional nodes to the swarm:

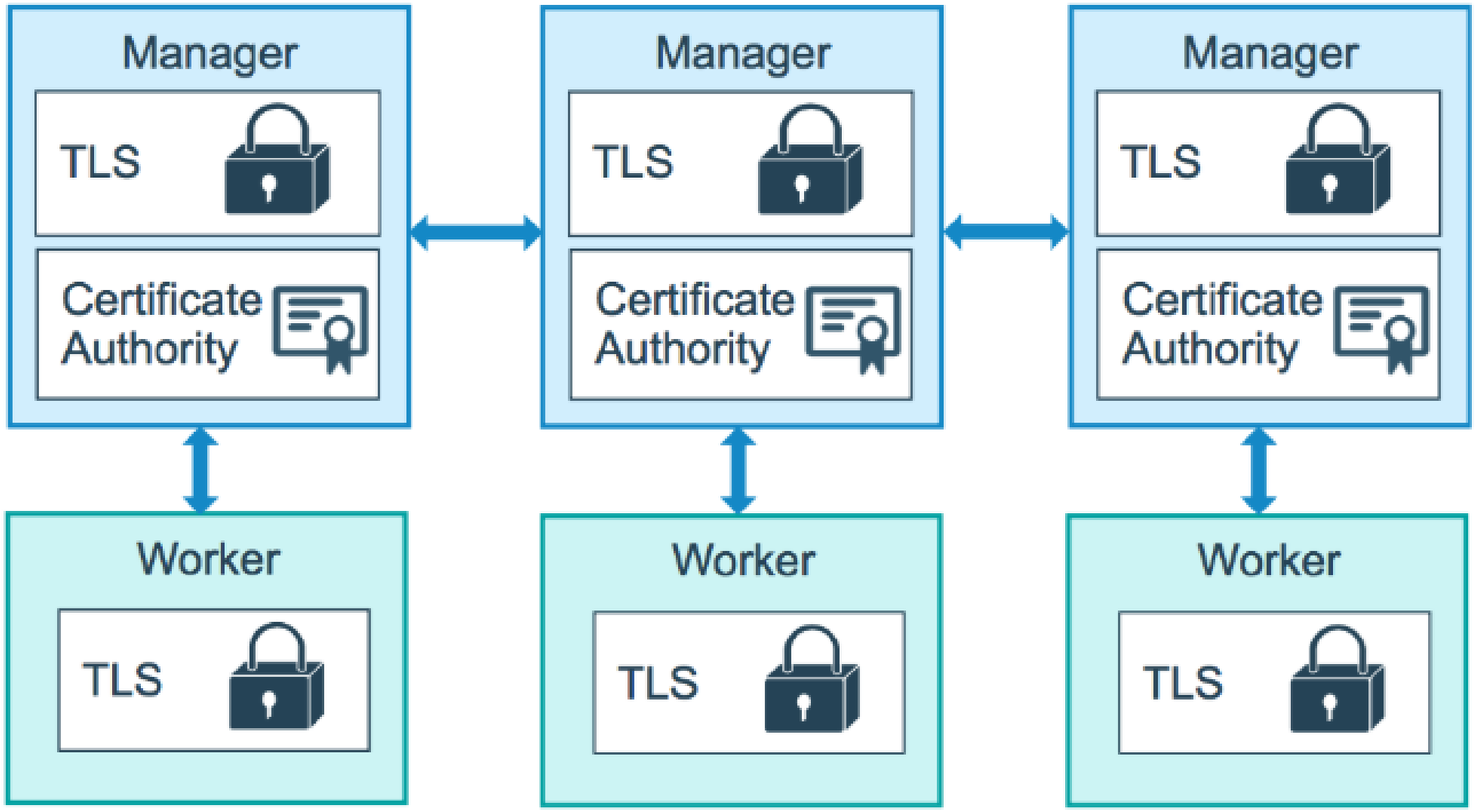
Worker Token



Manager Token

Swarm Security

The diagram below illustrates how manager nodes and worker nodes encrypt communications using a minimum of TLS 1.2.



Source: https://docs.docker.com/datacenter/dtr/2.3/guides/admin/configure/use-your-own-tls-certificates/

Swarm Security

The example below shows the information from a certificate of a worker node:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            3b:1c:06:91:73:fb:16:ff:69:c3:f7:a2:fe:96:c1:73:e2:80:97:3b
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: CN=swarm-ca
        Validity
            Not Before: Aug 30 02:39:00 2016 GMT
            Not After : Nov 28 03:39:00 2016 GMT
        Subject: O=ec2adilxf4ngv7ev8fwsi61i7, OU=swarm-worker,
CN=dw02poa4vqvzxi5c10gm4pq2g
...snip...
```

By default, each node in the swarm renews its certificate every three months. The user can configure this interval by running the docker swarm update **--cert-expiry <TIME PERIOD>** command.

Swarm Security

Rotating the CA Certificate:

- To generate a new CA certificate and password, run the **docker swarm ca—rotate**.
- To specify the root certificate and to use a root CA external to the swarm, the user can pass the **—ca-cert** and **—external-ca** flags if they prefer.
- Alternatively, to specify the exact certificate and key the user wants the swarm to use, they can pass the **—ca-cert** and **—ca-key** flags.

Swarm Security

When the user issues the **docker swarm ca --rotate command**, the following things happen in sequence:

1. Docker generates a cross-signed certificate.

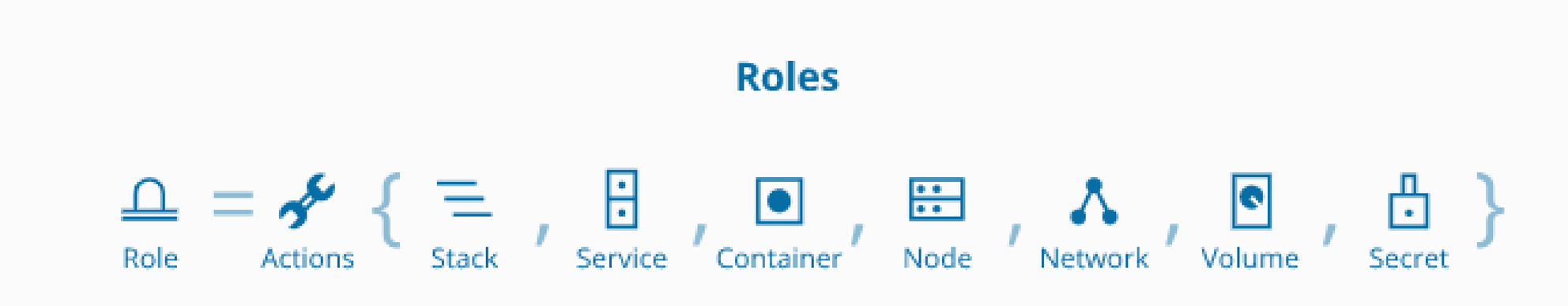
1. In Docker 17.06 and higher, Docker also tells all nodes to immediately renew their TLS certificates.

1. After every node in the swarm has a new TLS certificate signed by the new CA, Docker forgets about the old CA certificate and key material and tells all the nodes to trust only the new CA certificate.

Roles and Secrets

Roles

- Docker Universal Control Plane has two types of users: administrators and regular users.
- Administrators can make changes to the UCP swarm, while regular users have permissions that range from no access to full control over resources like volumes, networks, images, and containers.
- Users are grouped into teams and organizations.



Source: https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/permission-levels/#roles

Secrets

- Docker secrets to centrally manage this data and securely transmit it to only those containers that need access to it.
- Secrets are encrypted during transit and at rest in a Docker swarm.
- A given secret is only accessible to those services which have been granted explicit access to it, and only while those service tasks are running.

Secrets

The user can use secrets to manage any sensitive data which a container needs at runtime, but the user doesn't want to store the sensitive data in the image or in source control. Such sensitive data are:

- Usernames and passwords
- TLS certificates and keys
- SSH keys
- Other important data, such as the name of a database or internal server
- Generic strings or binary content (up to 500 kb in size)

How Docker Manages Secrets

Docker sends the secrets to the swarm manager over a mutual TLS connection when the user adds a secret to the swarm.

The secret is stored in the Raft log, which is encrypted.

The entire Raft log is replicated across the other managers, guaranteeing high availability for secrets.

Docker Secret Commands

Command	Description
docker secret create	Creates a secret from a file or STDIN as content
docker secret inspect	Displays detailed information on one or more secrets
docker secret ls	Lists secrets
docker secret rm	Removes one or more secrets

Key Takeaways

- Docker security prevents a compromised container from consuming a large amount of resources for disrupting service or performing malicious activities.
- Docker Trusted Registry can scan images in the repositories using Docker Security Scanning.
- A client bundle is a group of certificates downloadable directly from the Docker Universal Control Plane (UCP).
- Docker secrets centrally manage the sensitive data and securely transmit it to only those containers that need access to it.