

30 Aug 2016

Enforcing Segregation of Duty With Docker UCP & DTR



roughly a 12 min read
by Benjamin Wootton

[BACK](#)

[SHARE](#)

Docker, Engineering

For various process or regulatory reasons, many software teams need to show tight access controls and segregation of duties in their software environments. For example ensuring that:

- Developers can't access or deploy to production
- Only testers can deploy to test environments
- Only certain classes of users have access to parts of the system (such as DBAs being able to access database servers)

Traditionally, this has been done at the UNIX level and within deployment tooling, though sometimes teams have to rely on processes and trust to ensure that this is in place. This is a common audit failure.

But how do we achieve this in a containerised environment? The Docker Datacenter - specifically Trusted Registry and Universal Control Plane punch very hard in allowing us to achieve this. In this article we will walk through how.

Step 1: Access Control On The Repository

The first step in locking down access comes at the repository level. The Docker Trusted Registry (DTR) gives us fine grained control over who can push specific artefacts to the repository and who can download them.

Within the Trusted Registry, we have created two users. A developer and a tester. They are allocated to development and testing teams with our organisation, Acme Corp.

The screenshot shows the 'MEMBERS' tab for the 'acme_corp' organization. The sidebar on the left displays the organization name 'acme_corp', a list of teams ('development', 'test'), and a search bar. The main content area shows a table of users with columns for 'USERNAME', 'FULL NAME', and 'Role'. The 'Role' column has a dropdown menu for each user, and an 'Add user' button is visible in the top right corner.

USERNAME	FULL NAME	Role
mr.test	Mr Tester	Member
mr.developer	Mr Developer	Member
admin		Admin

A repository has been created within acme_corp called my_application. The developers have read and write access to it.

The screenshot shows the 'REPOSITORIES' tab for the 'acme_corp' organization. The sidebar on the left displays the organization name 'acme_corp', a list of teams ('development', 'test'), and a search bar. The main content area shows a table of repositories with columns for 'Repository Name' and 'Access'. The 'Access' column has a dropdown menu for each repository, and an 'Add repository' button is visible in the top right corner.

Repository Name	Access
acme_corp/my_application	Read & Write

Whilst the testers are limited to read only access.

The screenshot shows the 'REPOSITORIES' tab for the 'acme_corp' organization. The sidebar on the left displays the organization name 'acme_corp', a list of teams ('development', 'test'), and a search bar. The main content area shows a table of repositories with columns for 'Repository Name' and 'Access'. The 'Access' column has a dropdown menu for each repository, and an 'Add repository' button is visible in the top right corner.

Repository Name	Access
acme_corp/my_application	Read only

When the developer logs in, he can push to the docker registry successfully.

```
ubuntu@ip-172-31-46-237:~$ sudo docker login 54.154.128.196
Username: mr.developer
Password:
Login Succeeded
ubuntu@ip-172-31-46-237:~$ sudo docker push
54.154.128.196/acme_corp/my_application
The push refers to a repository [54.154.128.196/acme_corp/my_application]
8ac8bfaff55a: Layer already exists
latest: digest:
sha256:77d282d0257d46b68cbfd8440bdeb6f94b8aa656a00dfc009829985ceebdb5cc
size: 2100
```

But when the tester logs in, he receives an unauthorised error when attempting to push:

```
ubuntu@ip-172-31-46-237:~$ sudo docker login 54.154.128.196
Username: mr.testter
Password:
Login Succeeded
ubuntu@ip-172-31-46-237:~$ sudo docker push
54.154.128.196/acme_corp/my_application
The push refers to a repository [54.154.128.196/acme_corp/my_application]
8ac8bfaff55a: Layer already exists
unauthorized: authentication required
```

This simple piece of role based access control means that only authorised people can change the images which represent our applications. Any changes made to our containers are audited, including, who, where and when the change was committed.

Step 2: Access Control On Universal Control Plane

Docker Trusted Registry integrates with the Universal Control Plane (UCP). UCP provides a GUI for managing containers on a back-end Swarm cluster, and provides role based access control for all actions within the GUI.

Users are integrated between DTR and UCP, so our users created in DTR already exist in UCP. However, the teams are not integrated into UCP. I'm not sure if this is by design or if they will be integrated in future.

Docker Universal Control Plane

admin

Dashboard / Users & Teams

All Users

TEAMS

+ Create

developers

prod-support

testers

Create User

Search users...

USERNAME	FULL NAME		
admin		AdminYou	<div></div> <div></div>
mr.developer	Mr Developer		<div></div> <div></div>
mr.prodsupport	Mr Prod Support		<div></div> <div></div>
mr.prodsupport2	Mr Prod Support 2		<div></div> <div></div>
mr.testers	Mr Tester		<div></div> <div></div>

Our aim is to ensure that *only* production support are able to deploy into the environment in order to demonstrate segregation of duties. To start with, we can ensure that developers have no permissions by default:

Account Details

FULL NAME

Mr Developer

PASSWORD

☐ Disable account

☐ Is a UCP admin?

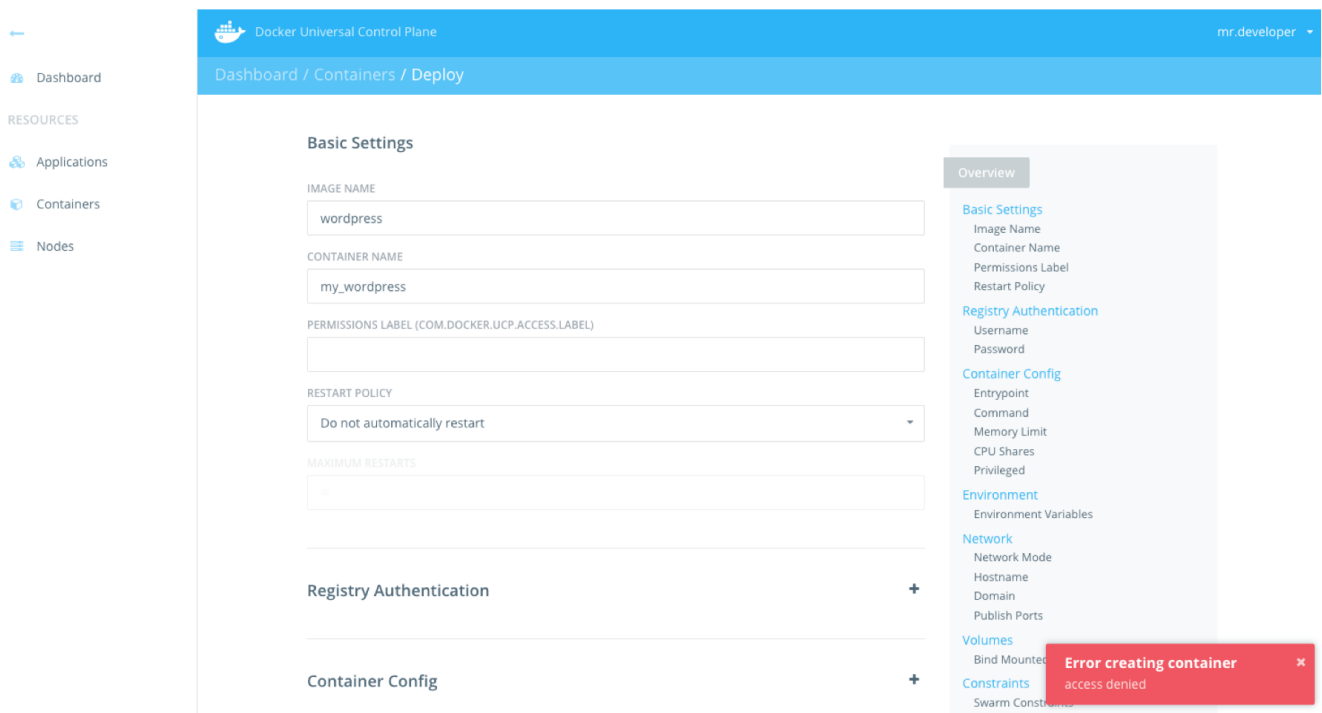
DEFAULT PERMISSIONS ?

No Access

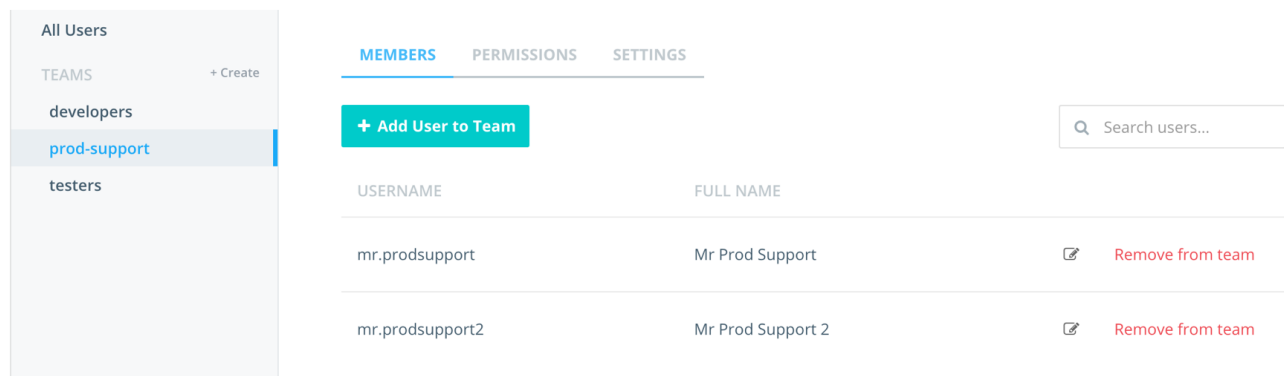
Update Account

Cancel

When they try to deploy, they will receive the access denied error shown below.



Our production support user however has the required access to deploy.



When we combine steps 1 and 2, we have demonstrated how only developers are the only ones who have access to commit artefacts, and production support are the only ones with ability to deploy. An intermediate step could be included where only testers can deploy to test environments, or we could also model the role of deployment engineers.

Step 3: Fine Grained Control and Visibility With Access Labels

By default, users can only see their own containers. The Universal Control Plane allows us to group users into teams so they can see each others containers, and then add a layer of fine grained controls to certain classes of containers.

In the image below, we have created a production support team and assigned the two production support users into the team.

All Users

TEAMS

+ Create

developers

prod-support

testers

MEMBERS

PERMISSIONS

SETTINGS

+ Add User to Team

Search users...

USERNAME	FULL NAME	
mr.prodsupport	Mr Prod Support	Remove from team
mr.prodsupport2	Mr Prod Support 2	Remove from team

We have then specified that the production support team have full control over resources with the label 'prod'

All Users

TEAMS

+ Create

developers

prod-support

testers

MEMBERS

PERMISSIONS

SETTINGS

+ Add Label

Search labels...

RESOURCE LABEL	PERMISSION	
prod	Full Control	

Another group, testers, have view only access on resources with the label 'prod'. They aren't allowed to start or change anything about containers with this access label.

All Users

TEAMS

+ Create

developers

prod-support

testers

MEMBERS

PERMISSIONS

SETTINGS

+ Add Label

Search labels...

RESOURCE LABEL	PERMISSION	
prod	View Only	

This permission access label is set at container deploy time. This results in it being added a label with the key COM.DOCKER.UCP.ACCESS.LABEL:

Basic Settings

IMAGE NAME

tomcat

CONTAINER NAME

tomcat

PERMISSIONS LABEL (COM.DOCKER.UCP.ACCESS.LABEL)

prod

RESTART POLICY

Do not automatically restart

MAXIMUM RESTARTS

Overview

Basic Settings

Image Name
Container Name
Permissions Label
Restart Policy


Registry Authentication

Username
Password

Container Config

Entrypoint
Command
Memory Limit
CPU Shares
Privileged

Any member of the production support team will see and have ability to administer this container. However, our tester does not have access to stop the container.

 Docker Universal Control Plane mr.testor

Dashboard / Containers

+ Deploy Container

1 Containers Selected

View Options

Search containers...

	ID	NODE	NAME	IMAGE	CREATED	
<input checked="" type="checkbox"/>	d550f1d3c727	ip-172-31-35-182	tomcat	tomcat	2016-08-28 21:49:36 +0100	⋮
<input type="checkbox"/>	3fcb7c503895	ip-172-31-35-182	xxasdasd	wordpress	2016-08-28 19:40:38 +0100	⋮

Universal Control Plane 1.1.2 (bbb5902) | API: 1.22

Error stopping container
access denied

This demonstrates how we can very finely target certain classes of container to certain classes of user.

It also enables multi-tenancy, where different groups or different application teams can share a UCP instance. Environments such as development, test, and staging can also be hosted on the same UCP and access controlled using labels in the same way.

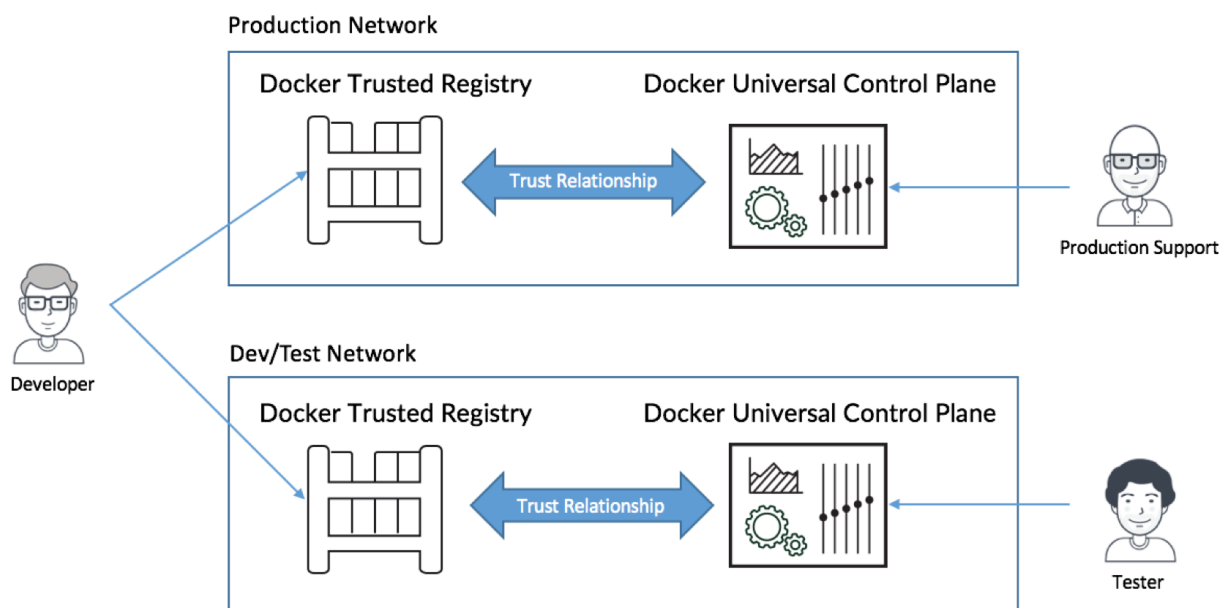
Step 4: Segregating Production & Non Production Trusted Registries

Another step we can take to further segregate the two instances and demonstrate clean separation between environments is to create two segregated instances of the Docker Datacenter runtime.

Development takes place in one segregated network. The developer checks into a DTR which is integrated with UCP. DTR & UCP have exchanged certificates to ensure that they are trusted. Only testers are able to deploy into the environment.

When a release candidate is ready, the developer checks into the production trusted registry. Only a production support engineer has access to perform the deployment through UCP. Again, certificates are exchanged and the network is segregated so the UCP cannot pull from the non production DTR.

Again, there are various options here such as introducing a deployment engineer, or using a CI server such as Jenkins to handle the build promotion.



A Word On DevOps

We are obviously hoping to move away from traditional teams, release engineers, and rigid handovers described here. For many organisations, it would be a terrible shame to recreate these processes as they adopt containers.

However, sometimes we can't get away from the need for segregation of duties for regulatory reasons.

One thing that we advise at Contino is that even if need to show segregation of duty, we can still do it using the kind of tooling supported role based access control even if the people sit as part of the same team.