

Docker for Enterprise Operations Exercises

Contents

1	Configure UCP	3
1.1	Install ntp	3
1.2	License the installation	4
1.3	Add SANs	4
1.4	Add additional worker nodes	4
1.5	Conclusion	5
2	Adding UCP Manager Nodes	5
2.1	Install ntp	6
2.2	Add manager nodes	6
2.3	Test manager nodes	6
2.4	Configure Scheduler	6
2.5	Conclusion	7
3	Ingress Load Balancing	7
3.1	Deploy who-am-I service	7
3.2	Observe load-balancing and scale	7
3.3	Ingress Routing Mesh	8
3.4	Conclusion	9
4	Deploy an Application	9
4.1	Background info	9
4.2	Deploy Services	10
4.3	Scale the application	12
4.4	Conclusion	12
5	HTTP Routing Mesh	12
5.1	Enable HTTP Routing Mesh	12
5.2	Deploy Services with HRM Labels	12
5.3	Test the routing	13
5.4	Conclusion	14
6	Access Control with Users, Teams and Labels	14
6.1	Create new users	15
6.2	Create a team and add users	15
6.3	Assign permissions to team	16
6.4	Deploy Labeled Services	16
6.5	Test user access	17
6.6	Test container access from the web UI	20
6.7	Test container access from the command line	20
6.8	Test admin access	21
6.9	Test default permissions	21
6.10	Conclusion	22

7	User Management with LDAP	22
7.1	Examine the LDAP server	23
7.2	Integrate UCP and LDAP	24
7.3	Test User Access	24
7.4	Cleanup	25
7.5	Conclusion	25
8	Password Recovery	25
8.1	Recovering admin passwords	26
8.2	Conclusion	26
9	Troubleshooting User Access Scenarios	26
9.1	Setup	26
9.2	Container Access	27
9.3	Asset Creation & Inspection	27
10	Secret Management	27
10.1	Create a secret in UCP	27
10.2	Deploy MySQL Service with secret	28
10.3	Deploy WordPress Service with secret	29
10.4	Find a secret in a running container	30
10.5	Conclusion	31
11	Logging	31
11.1	Installing ELK Stack	31
11.2	Configuring all Swarm nodes	32
11.3	Stream all Docker logs to ELK	32
12	Health Checks	34
12.1	Analyzing the Dockerfile	34
12.2	Writing a Compose File	35
12.3	Deploying and using the Service	35
12.4	Summary	36
13	Demo: Install DTR	36
13.1	Add additional worker nodes to UCP	36
13.2	Install DTR	37
13.3	Check that DTR is running	37
13.4	Configure DTR to use S3	38
13.5	Integrate UCP and DTR	39
13.6	Test the integration	40
13.7	Conclusion	41
14	Demo: Install DTR Replicas	41
14.1	Install Replicas	41
14.2	Cleanup	41
14.3	Conclusion	42
15	Create a DTR Repository	42
15.1	Setup	42
15.2	Create a Repo	42
15.3	Push an image into the repository	43
15.4	Pull and Push an image from a DTR Repository	43
15.5	Conclusion	44
16	Working with Organizations and Teams	44
16.1	Demo: Creating Orgazinations and Teams	44
16.2	R/W in Org Repos	44

16.3 Conclusion	45
17 Enabling Image Signing	45
17.1 Install the Notary CLI	45
17.2 Configure the Notary CLI	45
17.3 Set Up Content Trust for a repository	46
17.4 Push a signed image	46
17.5 Conclusion	47
18 Delegate Image Signing	47
18.1 Rotate Snapshot key	47
18.2 Delegate image signing	47
18.3 Push a signed image	48
18.4 Conclusion	49
19 Image Scanning in DTR	49
19.1 Pre-requisites:	50
19.2 Enable Image Scanning	50
19.3 Create a repo	50
19.4 Investigate layers & components	51
19.5 Conclusion	52
20 DTR Webhooks	52
20.1 Setting up a Webhook	52
20.2 Conclusion	53

1 Configure UCP

In this exercise, you'll set up a single Universal Control Plane manager, with a pair of worker nodes.

Pre-requisites:

- Three nodes running Ubuntu 16.04 with linux kernel 4.4.0-22-generic or later, and Docker EE 17.03.0-ee, named as per:
 - ucp-controller
 - ucp-node-0
 - ucp-node-1
- UCP pre-installed on ucp-controller. If UCP is not installed, run:


```
$ docker container run --rm -it --name ucp \
-v /var/run/docker.sock:/var/run/docker.sock docker/ucp install -i
```

 and follow the prompts.
- A UCP license. If you don't have one, a trial license is available at <https://store.docker.com/bundles/docker-datacenter/purchase?plan=free-trial>.

1.1 Install ntp

1. While not strictly required, the ntp utility ensures clock synchronization across your cluster, helping you avoid a number of related problems. On ucp-controller, ucp-node-0 and ucp-node-1, run:

```
$ sudo apt install ntp
```

1.2 License the installation

1. Visit the UCP interface at <https://<public IP of ucp-controller>>.

Note: Most browsers will display some sort of warning message regarding the connection. This is because we are using the default self signed certificates as opposed to a certificate from a trusted source. You can tell the browser to ignore the warning and proceed.

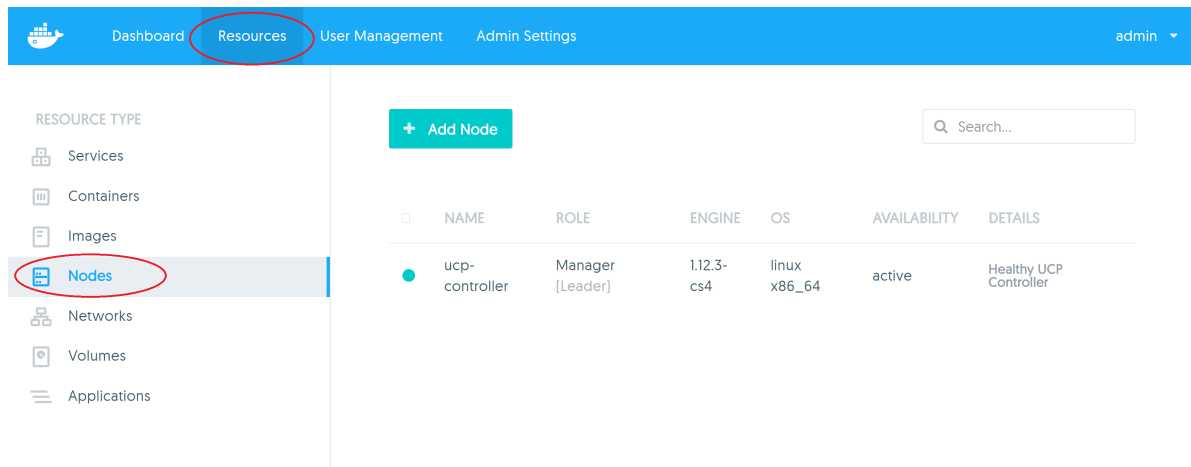
2. Login to your Admin account; if UCP was pre-installed for you, your instructor will provide the access credentials.
3. Once logged in you will see a prompt screen asking you to upload a license. Click the **Upload License** button and upload your UCP license; if you need a trial license, you can download one from Docker Store: <https://store.docker.com/?overlay=subscriptions>.

1.3 Add SANs

1. Navigate to **Resources** -> **Nodes** -> **ucp-controller** and scroll down the Details page to the section on SANs. Add any public IPs or public FQDNs for your controller node (necessary to make sure these names are registered in the certificate used for mutual TLS communication between nodes).
2. Click **Save Changes** in the top right when you're done adding aliases.
3. Wait a moment for node configuration to complete, then refresh your browser; you will likely have to make another security exception.

1.4 Add additional worker nodes

1. Click the **Resources** link on the top navigation bar and then click on **Nodes** on the left navigation panel. You should see a page listing all the nodes in your UCP cluster. At the moment, you should have one node, which is the ucp-controller.



The screenshot shows the UCP interface. The top navigation bar has 'Dashboard', 'Resources' (highlighted with a red circle), 'User Management', and 'Admin Settings'. The right side of the top bar shows 'admin'. The left sidebar lists 'RESOURCE TYPE' with options: Services, Containers, Images, Nodes (highlighted with a red circle), Networks, Volumes, and Applications. The main content area has a '+ Add Node' button and a search bar. Below is a table of nodes:

	NAME	ROLE	ENGINE	OS	AVAILABILITY	DETAILS
	ucp-controller	Manager [Leader]	1.12.3-cs4	linux x86_64	active	Healthy UCP Controller

2. Click **Add Node**; a join command is presented.

Add Node
✕

☐ Add node as a manager ?

☐ Use a custom listen address ?

☐ Use a custom advertise address ?

Nodes are added to the cluster by using Docker's 'swarm join' command. Use the following command on the engine you want to connect to the cluster.

```
docker swarm join \
  --token SWMTKN-1-5h63yo71fu8wczmw1hdx1cr3kdwvbrlkk6gq5jcr53jm0d01-2ezkoswsxopiy923pz11b79u \
  10.0.4.185:2377
```

📄 Copy to Clipboard

3. Open two new SSH terminals into your `ucp-node-0` and `ucp-node-1` nodes.
4. Cut and paste the join command provided by UCP into your `ucp-node-0` and `ucp-node-1` node terminals. Each should report: **This node joined a swarm as a worker.**
5. Click on the **Nodes** page and check that you have 3 nodes in your cluster.

Dashboard Resources User Management Admin Settings
admin ▾

RESOURCE TYPE

- Services
- Containers
- Images
- Nodes**
- Networks
- Volumes
- Applications

+ Add Node

🔍 Search...

	NAME	ROLE	ENGINE	OS	AVAILABILITY	DETAILS
●	ucp-controller	Manager (Leader)	1.12.3-cs4	linux x86_64	active	Healthy UCP Controller
●	ucp-node-1	Worker	1.12.3-cs4	linux x86_64	active	Healthy UCP Worker
●	ucp-node-0	Worker	1.12.3-cs4	linux x86_64	active	Healthy UCP Worker

1.5 Conclusion

At this point, UCP is installed with one manager and two worker nodes. Bear in mind that UCP is really just a collection of services running on a swarm. Alternatively, we could have set up a swarm from the command line first, and installed UCP on top of this if we preferred; UCP would have automatically recognized and integrated all nodes.

2 Adding UCP Manager Nodes

In this exercise, we will add 2 additional manager nodes to our UCP installation in order to make it highly available.

Pre-requisites:

- UCP installed with 2 worker nodes
- Two additional nodes running Ubuntu 16.04 with linux kernel 4.4.0-22-generic or later, and Docker EE 17.03.0-ee, named:
 - `ucp-manager-0`

– ucp-manager-1

2.1 Install ntp

1. Like we did for our other UCP nodes, install ntp first:

```
$ sudo apt install ntp
```

2.2 Add manager nodes

1. On the UCP web interface, navigate **Resources** -> **Nodes**, and click the **Add Node** button.
2. Tick the checkbox that says **Add node as a manager**.
3. Open a terminal connection to ucp-manager-0 and ucp-manager-1, and cut and paste the join command presented by UCP into each. Each should report *This node joined a swarm as a manager*.
4. Go back to the **Nodes** page on UCP. You should now see 5 nodes, similar to the following:

	NAME	ROLE	ENGINE	OS	AVAILABILITY	DETAILS
	ucp-manager-1	Manager	1.12.3-cs4	linux x86_64	active	Healthy UCP Controller
	ucp-controller	Manager [Leader]	1.12.3-cs4	linux x86_64	active	Healthy UCP Controller
	ucp-node-1	Worker	1.12.3-cs4	linux x86_64	active	Healthy UCP Worker
	ucp-node-0	Worker	1.12.3-cs4	linux x86_64	active	Healthy UCP Worker
	ucp-manager-0	Manager	1.12.3-cs4	linux x86_64	active	Healthy UCP Controller

Note: It takes a few minutes for a node to be set up as a UCP manager. This is reflected in the **Details** column. At first it will say that the node is being configured and when finished it will say *Healthy UCP Controller*.

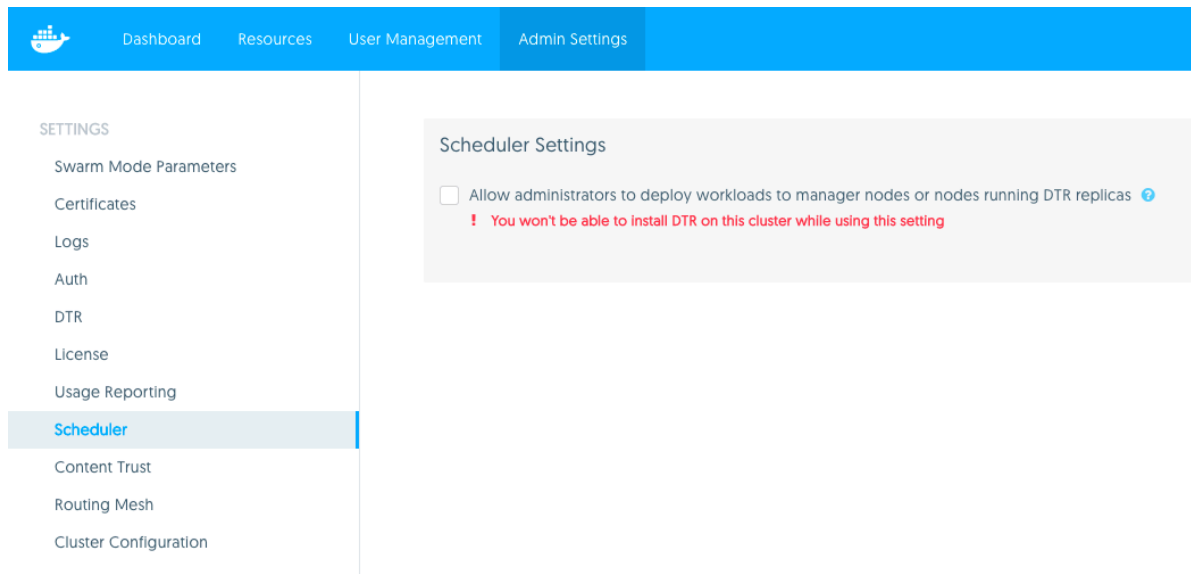
2.3 Test manager nodes

1. Open a new browser tab and put in the domain or public IP of either your ucp-manager-0 or ucp-manager-1 node (with https://). You should see the UCP login page.
2. Verify that you can login with the **Admin** account

2.4 Configure Scheduler

Now that we have 3 manager nodes setup, we want to make sure that our manager nodes are dedicated for their purpose and that they do not run any application containers.

1. In UCP, navigate **Admin Settings** -> **Scheduler**.
2. Uncheck both of the checkbox settings on the page as shown below:



This will prevent containers from being scheduled on any of our UCP manager nodes.

2.5 Conclusion

Your UCP cluster now has three manager nodes, preventing your cluster from collapsing if one of them fails. Bear in mind that this manager consensus is exactly a swarm manager consensus; UCP controllers are just swarm managers, with additional services running on top to control UCP.

3 Ingress Load Balancing

In this exercise, you will observe the behavior of the built in load balancing abilities of the Swarm mesh net.

Pre-requisites:

- UCP installed with 2 worker nodes (at least).

3.1 Deploy who-am-I service

1. Navigate to **Resources** -> **Services** -> **Create a Service**. This dialog can be used to define a service in UCP.
2. Fill in `who-am-I` as the service name, and `training/whoami:latest` as the image name.
3. Set the Replicas field to 3.
4. Click **Resources**, then **+ Publish Port**. Enter 8000 for both **Internal Port** and **Published Port**, and then click **Done** on the ports form; this will cause the mesh net to route inbound traffic on port 8000 to a `who-am-I` container's internal port 8000.
5. At the bottom, click **Deploy Now** to launch your service.
6. Once the service is deployed check the **Tasks** tab to confirm that there are 3 containers running.

3.2 Observe load-balancing and scale

1. SSH into the `ucp-controller` node.
2. Run `curl localhost:8000` and observe the output. You should see something similar to the following:

```
$ curl localhost:8000
I'm a7e5a21e6e26
```

- Run `curl localhost:8000` again. The `whoami` containers uniquely identify themselves by returning their respective hostname, so each one of our `whoami` instances should report a different value.
- Repeat the command two more times. You should see one new value and then on the 4th request it should revert back to the value of the first container.
- Now let's scale the number of Tasks for our `who-am-I` service. We will add 3 more tasks to the service so that we have a total of six `whoami` containers.
 - Navigate to **Resources** -> **Services**, and click on the `who-am-I` service.
 - Click on the **Scheduling** Tab in the service details and edit the **Scale** field to 6 as shown below:

Service: who-am-I

DETAILS SCHEDULING ENVIRONMENT RESOURCES TASKS Actions

Mode Replicated

Scale 6 ✓ ✗

- Once you have edited the field, click on the **Tick** symbol and then click **Save Changes** on the top right.
- Verify there are 6 tasks running for the `who-am-I` service via UCP.
 - Now switch back to your SSH terminal and run `curl localhost:8000` multiple times again. Use a script like this:

```
$ for n in {1..10}; do
> curl localhost:8000
> done;
I'm 263fc24d0789
I'm 57ca6c0c0eb1
I'm c2ee8032c828
I'm c20c1412f4ff
I'm e6a88a30481a
I'm 86e262733b1e
I'm 263fc24d0789
I'm 57ca6c0c0eb1
I'm c2ee8032c828
I'm c20c1412f4ff
```

You should be able to observe some new values. Note how the values repeat after the 6th `curl` command.

3.3 Ingress Routing Mesh

- To see the swarm mesh net in action from the browser, set up a service called 'NGINX', based on the `nginx:latest` image, with a single replica and container port 80 exposed on public port 8080. Once the service is live, determine which node the single NGINX container is running on.
- Open a browser tab and specify the public domain or IP address, followed by port 8080, of any of your UCP nodes. Try this on each of your nodes:
 - ucp-controller
 - ucp-node-0
 - ucp-node-1

- ucp-manager-0
- ucp-manager-1

You should be able to observe the NGINX welcome page on all of them, whether they are running your NGINX container or not.

3.4 Conclusion

In this exercise, we saw two examples of traffic being routed and load balanced (in a round robin fashion) across the Ingress network, to containers deployed by services with exposed ports. Under the hood, when traffic is sent to an exposed service port, it is assigned a virtual IP and load balanced across the Ingress overlay network by IP virtual servers situated on each node.

4 Deploy an Application

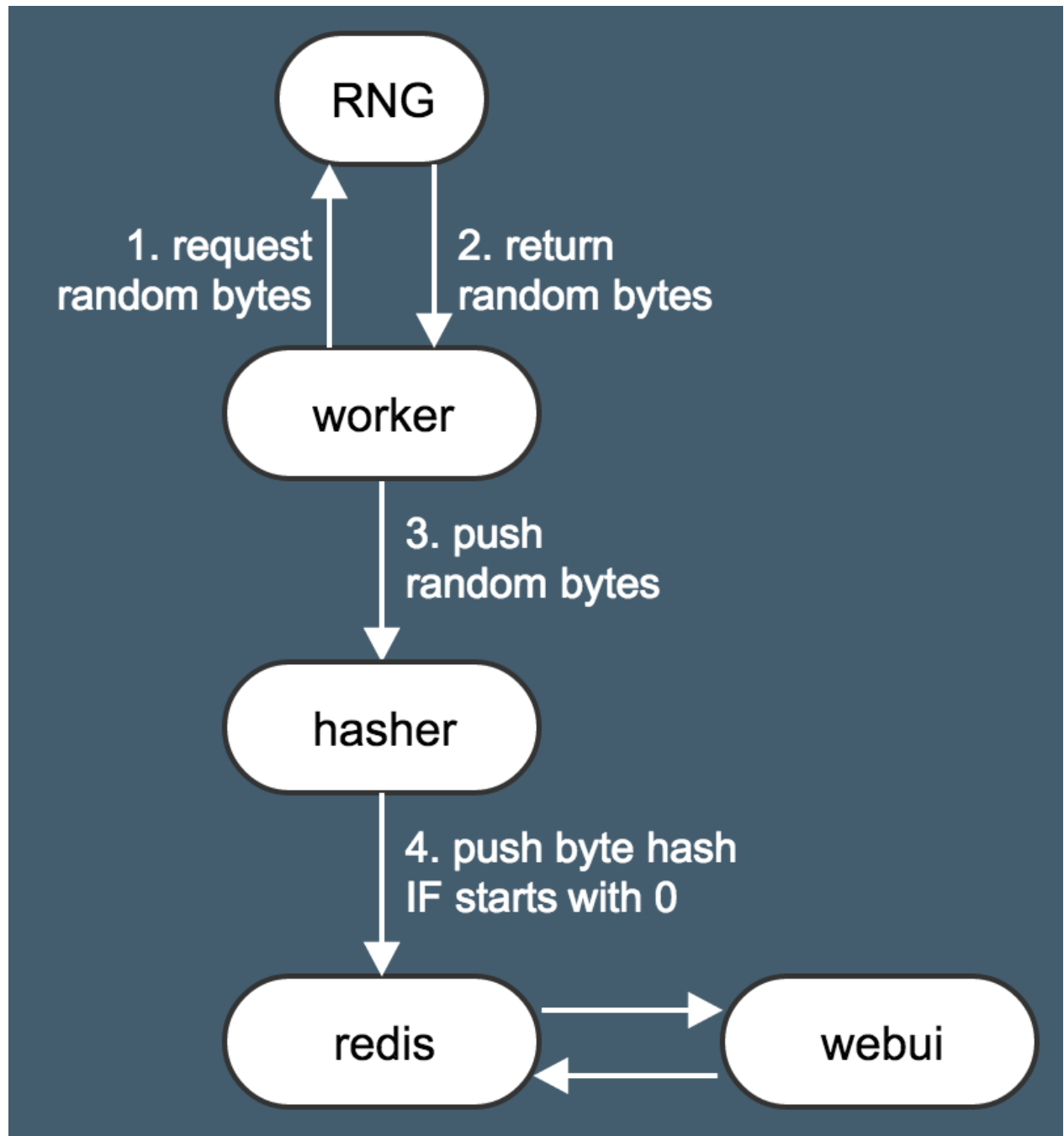
In this exercise you will learn how to deploy an application that consists of multiple services.

Pre-requisites:

- UCP installed with 2 worker nodes (or more).
- No existing services running. Before beginning this exercise, delete all your previous services that have been deployed, in order to avoid port conflicts.

4.1 Background info

The application that we are going to deploy is a toy Dockercoin miner that hashes random numbers and logs any hash beginning with a 0 as a 'Dockercoin', and reports real-time results in a web UI. The app consists of five services, that interact like this:



4.2 Deploy Services

The services that make up our app can be defined in a Compose v3.1 file and then deployed in UCP via the web UI.

1. In your UCP web UI, navigate **Resources** -> **Stacks & Applications** -> **Deploy**.
2. On the Deploy screen, specify 'Dockercoins' in the **Application Name** field, and paste the following Compose file into the **Application Definition** field:

```
1 version: "3.1"
2
3 services:
4   rng:
5     image: training/dockercoins_rng:1.0
```

```

6   networks:
7     - dockercoins
8   ports:
9     - "8001:80"
10  hasher:
11    image: training/dockercoins_hasher:1.0
12    networks:
13      - dockercoins
14    ports:
15      - "8002:80"
16  webui:
17    image: training/dockercoins_webui:1.0
18    networks:
19      - dockercoins
20    ports:
21      - "8000:80"
22  redis:
23    image: redis
24    networks:
25      - dockercoins
26  worker:
27    image: training/dockercoins_worker:1.0
28    networks:
29      - dockercoins
30 networks:
31   dockercoins:

```

Deploy compose.yml ✕

APPLICATION NAME ?

Dockercoins

DEPLOY AS ?

Services Containers

APPLICATION DEFINITION ?

```

1  version: "3.1"
2
3  services:
4    rng:
5      image: training/dockercoins_rng:1.0
6      networks:
7        - dockercoins
8      ports:
9        - "8001:80"
10
11   hasher:
12     image: training/dockercoins_hasher:1.0
13     networks:
14       - dockercoins
15     ports:

```

☐ Show verbose logs

Create Cancel

- Click **Create** and wait for the services to deploy and then check that the application is listed on the **Stacks and Applications** page.
- Open a browser tab and see Dockercoins in action by specifying the URL of any of your UCP nodes, followed

by port 8000.

4.3 Scale the application

We want more Dockercoins! To increase mining speed, we will scale the `worker` service so that we will have more containers to handle the workload.

1. Scale the `worker` service to run 2 containers; find the option to do this in the service's listing in UCP.
2. Return to the web UI after the new worker container has spun up; your mining speed should have doubled.
3. Try re-deploying the Dockercoins application, with the following modification to the Compose file:

```
worker:
  image: training/dockercoins_worker:1.0
  networks:
    - dockercoins
  deploy:
    replicas: 2
```

The end result is the same, but the scale of the `worker` service is captured right in the application definition, rather than having to manage it through UCP after deployment.

4. Finally, remove the Dockercoins application via the **Stacks and Applications** tab in UCP.

4.4 Conclusion

In this exercise, you deployed an entire application from a single compose file; note that in order for service containers to connect to each other, we had to make sure all services connected their containers to the same network, via the `networks` key above.

5 HTTP Routing Mesh

In this exercise, we'll investigate the optional application layer load balancer built into UCP.

Pre-requisites:

- UCP installed with at least 2 worker nodes

5.1 Enable HTTP Routing Mesh

1. From a UCP admin account, navigate **Admin Settings** -> **Routing Mesh**.
2. Tick the checkbox which says **Enable HTTP routing mesh**.
3. Specify port 80 in the **HTTP port** field and port 8443 in the **HTTPS port** field, and click **Update**.
4. Navigate **Resources** -> **Networks** on the left navigation bar. Verify that a network called `ucp-hrm` has been created.

5.2 Deploy Services with HRM Labels

1. Create a service called `webapp` using the `nginx:latest` image. On the **Deploy Service** screen, navigate **Resources** -> **Attach a Network**, and select the `ucp-hrm` network. Then click **Done** on this network attachment dialog.

- Click on **Publish Port** and map port 80 of the NGINX service to port 5000 on the host. Select Ingress for the **Publish Mode**. In the same pane, click on **Add hostname based route** and add a HTTP routing mesh hostname of **nginx.mycompany.com**. Your ports configuration should look like the following:

Ports ?

INTERNAL PORT ? 80

PROTOCOL ? tcp udp

PUBLISH MODE ? Ingress Host None

PUBLIC PORT ? 5000

EXTERNAL SCHEME ? http://

ROUTING MESH HOST ? nginx.mycompany.com

+ Add hostname based route

Done Cancel

- Click on **Done** for Ports and then click **Deploy Now** to deploy the service.
- Deploy another service called **enterprise-app**. Use the tomcat image and for port mapping, map the internal port 8080 to the public port 8080. Configure a HTTP routing mesh using the hostname **tomcat.mycompany.com**. Remember to attach the **ucp-hrm** network.

5.3 Test the routing

We now have two services with HTTP routing configured. To test this out we need to configure DNS so that both **nginx.mycompany.com** and **tomcat.mycompany.com** point to one of the nodes in our UCP cluster. To do this, we will configure the `/etc/hosts` file on one of our nodes.

- SSH into your **ucp-controller** node
- Open the `/etc/hosts`. You will need to specify root privileges with `sudo`:

```
$ sudo nano /etc/hosts
```

- Add a line at the beginning of the file specifying the following:

```
<Public IP address of ucp-node-0> nginx.mycompany.com tomcat.mycompany.com
```

(We specify the public IP address of one of our UCP worker nodes so that when we make a request to **nginx.mycompany.com** or **tomcat.mycompany.com**, we can resolve the hostname to an actual IP address).

Your `/etc/hosts` file should look something like the following:

```
1 54.202.59.103 nginx.mycompany.com tomcat.mycompany.com
2 127.0.0.1 ucp-controller localhost
3
4 # The following lines are desirable for IPv6 capable hosts
5 ::1 ip6-localhost ip6-loopback
6 fe00::0 ip6-localnet
7 ff00::0 ip6-mcastprefix
8 ff02::1 ip6-allnodes
9 ff02::2 ip6-allrouters
10 ff02::3 ip6-allhosts
```

- Run `curl -v nginx.mycompany.com`. You should see the request header followed by the HTML output of the NGINX welcome page. (If the curl appears to just hang and you're using AWS nodes for this lab, you may

need to open your security groups so your nodes can speak to one another). The request header should look like:

```
* Rebuilt URL to: nginx.mycompany.com/
* Trying 174.129.72.134...
* Connected to nginx.mycompany.com (174.129.72.134) port 80 (#0)
> GET / HTTP/1.1
> Host: nginx.mycompany.com
> User-Agent: curl/7.47.0
> Accept: */*
...
```

Note that the request proceeds on port 80, which is the port we configured our ucp-hrm routing mesh service to listen on when we first enabled the routing mesh; the `/etc/hosts` modification we made above routed `nginx.mycompany.com` to one of our swarm nodes, at which point the swarm routing mesh delivered the request to the ucp-hrm service which examined the header for the line that says `Host: nginx.mycompany.com`. This host value matched the routing mesh host we specified in creating our nginx service, so the request gets routed to nginx on the specified port.

5. Now run `curl -v` against the public IP of one of your UCP worker nodes. You will get a 503 error.

```
* Rebuilt URL to: 174.129.72.134/
* Trying 174.129.72.134...
* Connected to 174.129.72.134 (174.129.72.134) port 80 (#0)
> GET / HTTP/1.1
> Host: 174.129.72.134
> User-Agent: curl/7.47.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 503 Service Unavailable
< Cache-Control: no-cache
< Connection: close
< Content-Type: text/html
<
<html><body><h1>503 Service Unavailable</h1>
No server is available to handle this request.
</body></html>
```

In this case, the `Host:` value in the header was just the public IP of our worker - not a hostname that we told ucp-hrm what to do with, hence the 503.

6. Run `curl -v tomcat.mycompany.com`. You should see the HTML output of the default tomcat page; the HRM routes traffic to the correct app based on the header's `Host:` value, as above.

5.4 Conclusion

In this exercise, we set up the HTTP routing mesh, and saw it in action. Note that HRM is a *service*, and traffic is routed to this service exactly like traffic going to any exposed service: via the swarm routing mesh, in this case on port 80. Only once traffic arrives at the ucp-hrm service container is the header read for the `Host:` value, and the packet routed to the corresponding service.

6 Access Control with Users, Teams and Labels

In this exercise, we'll walk through the basics of creating and managing users and teams in UCP.

Pre-requisites:

- A working UCP installation.

6.1 Create new users

In this step you will create the 4 new users shown below:

Username	Full Name	Default Permissions
joeysfull	Joey Full	Full Control
kellyres	Kelly Restricted	Restricted Control
barryview	Barry View	View Only
traceyno	Tracey No	No Access

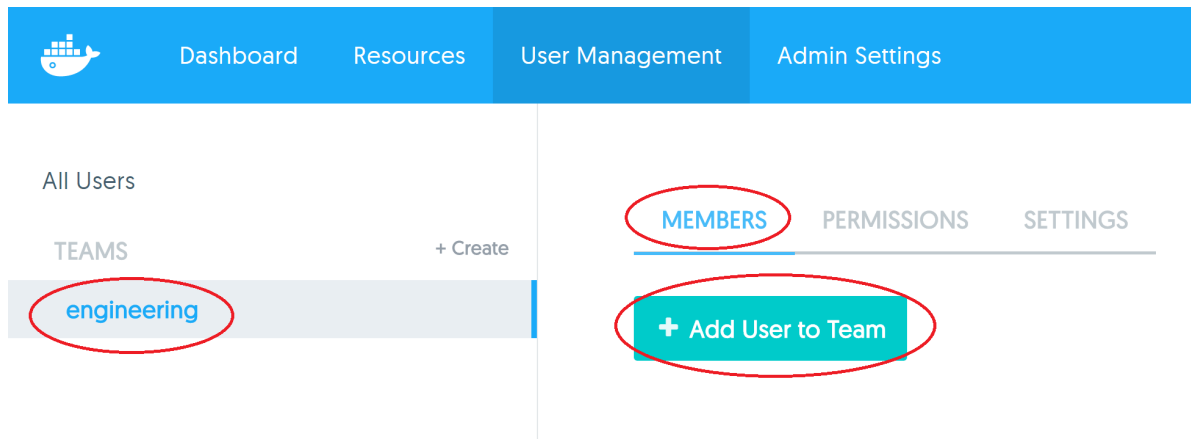
1. In UCP, navigate **User Management** -> **Create User**.
2. Fill out the **Create User** form with the details provided in the table above. The screenshot below shows the form filled out with the details for the *Joey Full* user; click **Create User** after finishing entering the details for each user.

Be sure to make a note of the password that you set for each user. You will need this in future labs.

6.2 Create a team and add users

Users can be grouped into teams for the purpose of creating shared resources.

1. Create a team called **Engineering** by clicking the **+ Create** button on the left hand side of the **User Management** view.
2. Set the **TEAM NAME** to "Engineering"
3. Make sure the Engineering team is selected and click the **Add User to Team** button from the **Members** tab.

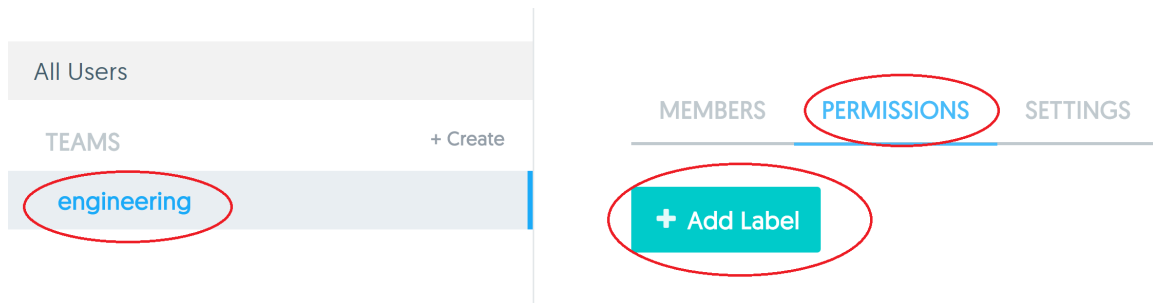


4. Add all four new users to the team by clicking the **Add to Team** button next to each of them and then click **Done**. Do not add yourself (usually "admin") to the team.

6.3 Assign permissions to team

Permissions are granted to teams via the use of labels.

1. Navigate **User Management** -> **Engineering** -> **Permissions**, and click + **Add Label**.



2. Create the following three labels and click **Add Label**.

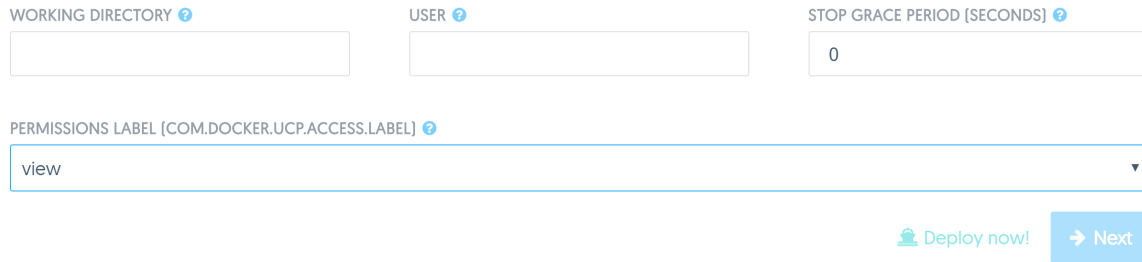
LABEL	PERMISSION
view	View Only
restricted	Restricted Control
run	Full Control

The labels will now be listed on the **Permissions** tab of the Engineering team.

6.4 Deploy Labeled Services

By deploying an asset like a service with a label, we can control which users are allowed to interact with this asset via their team membership and corresponding labels.

1. Create a service called `nginx-labeled` using the `nginx:latest` image. Before deploying, find the **Permissions Label** field, select the **View** label, and finally click **Deploy now!**:

A screenshot of a deployment configuration interface. It features three input fields at the top: 'WORKING DIRECTORY' with a help icon, 'USER' with a help icon, and 'STOP GRACE PERIOD (SECONDS)' with a help icon and the value '0'. Below these is a dropdown menu for 'PERMISSIONS LABEL (COM.DOCKER.UCP.ACCESS.LABEL)' with a help icon, currently showing 'view'. At the bottom right are two buttons: 'Deploy now!' with a person icon and 'Next' with a right arrow icon. A forward slash is visible to the right of the 'Next' button.

WORKING DIRECTORY ?

USER ?

STOP GRACE PERIOD (SECONDS) ? 0

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

view

Deploy now! Next

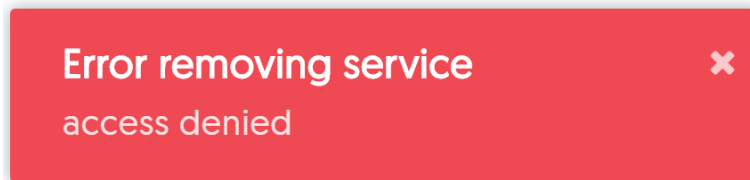
Repeat this step to deploy another service called `nginx-unlabeled`, but this time **do not use any label**.

6.5 Test user access

Now that we've set up some users, teams, labels, and services, let's see how all these things interact to govern which users can access which resources.

1. Log out of UCP and log back in as user **joeyfull**
2. Click on the **Resources** link in the top navigation bar. What services can you see, and why? Note that unlabeled containers are only ever visible to the user who created them, and admins.
3. Select the `nginx-labeled` service and try to delete it.

You should get the following error message on the bottom of the screen:



4. Click on the service to view its details.
5. Notice the **view** permission label on the **Details** tab

Service: nginx


DETAILS

SCHEDULING

ENVIRONMENT

RESOURCES

TASKS

ID	9pzlgvv348equ780mhhvpwvoq
Name	nginx 
Permission Label	<div>view ▼</div>
Created	2016-11-29 15:49:30 +1100
Last Updated	2016-11-29 15:49:30 +1100

/

You can also see this on the **Environment** tab. Look for the `com.docker.ucp.access.label` key in the **Labels** section

Service: nginx

DETAILS
SCHEDULING
ENVIRONMENT
RESOURCES
TASKS

Environment Variables

+ Add environment variable

No environment variables defined

Labels

+ Add label

KEY	VALUE
com.docker.ucp.access.label	view
com.docker.ucp.access.owner	admin

6. Click on the **Tasks** tab of the service and select the `nginx-labeled.1` task. This will bring up the container options.
7. Click on the **Container Actions** button on the top right side and select the **Restart Container** option (to confirm the action click the **Click to confirm restart (5s)** button).

Container: nginx.1.2iyuafr5ag7it19kxdu5cruki
✕

DETAILS
LOGS
STATS
CONSOLE

Configuration

ID ? 7a178896af3c

Name ? nginx.1.2iyuafr5ag7it19kxdu5cruki

✕ Remove Container

■ Stop Container

↺ Restart Container

Container Actions ▼

Notice how you get the same **Access Denied** error message. The label permissions apply to the service and also any containers that are part of the service.

8. Deploy two additional services using the details shown in the table below:

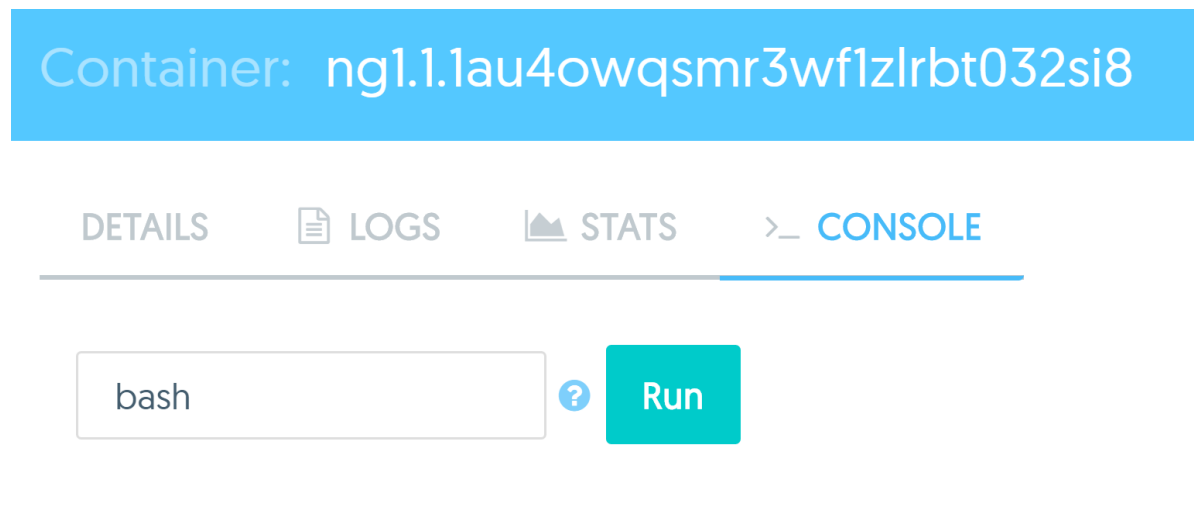
Image Name	Service Name	Permissions Label
nginx	ng1	restricted
nginx	ng2	run

While deploying the services, you will notice that only the **Restricted** and **Run** permission labels are available for selection. This is because members of the Engineering team have *Restricted Control* on the **restricted** label, and *Full Control* on the **run** label. Both of these permissions allow for the deployment of new containers. With the **view** label, the Engineering team only has *View Only* permissions, and thus cannot use this label when deploying a service.

6.6 Test container access from the web UI

In this step, we'll investigate **joeyfull**'s ability to interact with individual containers stood up as part of services.

1. Click on the **ng1** service. Then click the **Tasks** tab and select the **ng1.1** task. Then click on the **Console** tab.



2. Click on the **Run** button with "bash" specified in the field.

This action is the GUI equivalent of running a `docker container exec` command. In this case, you are trying to execute a `bash` terminal inside the **ng1** container. The request fails, since **joeyfull**'s membership in the Engineering team results in her having *Restricted Control* privilege to this container, via the **restricted** label; note this takes precedence over Joey's usual full control. *Restricted Control* prevents users from opening `exec` sessions to a container.

3. Now try the same thing with a **ng2** container. Is **joeyfull** able to do this? Why or why not?

6.7 Test container access from the command line

In this step you will create and download a **client bundle** for the **joeyfull** user, which is a credential package for allowing users to interact with UCP from the command line.

1. Navigate **joeyfull** -> **Profile**.
2. Scroll to the bottom of the profile screen and click **Create a Client Bundle**. This will download the client bundle to your local machine as a zipped archive file.
3. Upload the client bundle zip to any node in your swarm; in a bash shell, this will look something like

```
$ scp <your zip archive> ubuntu@<ip>:~/.
```

Windows users can download `winscp` to similar effect.

4. Unzip the client bundle on the node you uploaded it to. If your unzip utility places the archive contents in a new directory, move into that directory now; if you don't have a favorite such utility, try `unzip <archive>`
5. Source the `env.sh` shell script found in the unzipped archive via `source env.sh`.
6. Run a `docker container ls` command to list the containers. Look for the container with the name `ng1.1.<id>`. This is container 1 in our `ng1` service. Take note of the container ID and then run:

```
$ docker container exec -it <container ID> bash
```

to open a `bash` terminal on the `ng1.1` container. Does the command succeed? Why or why not?

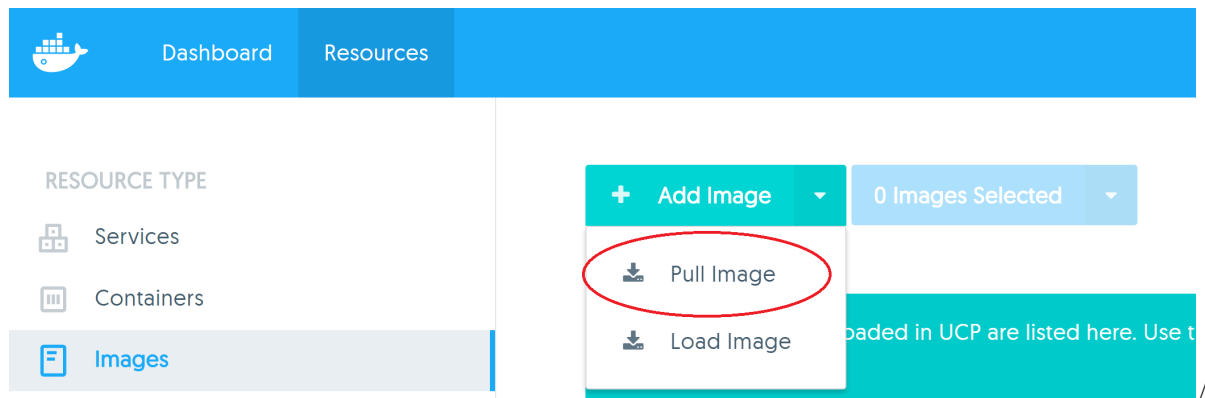
7. Repeat the previous step for the `ng2.1` container. What happens and why?

6.8 Test admin access

1. Logout of UCP as **joeyfull** and log back in as the **admin** user.
2. Navigate **Resources** -> **Services** -> **ng1** -> **Tasks**, and then select the **ng1.1** task.
3. Click the **Console** tab of the container task and run a **bash** terminal. Does the command succeed? Why or why not?

6.9 Test default permissions

1. Logout of UCP as the **admin** user and log back in as **joeyfull**.
2. Navigate **Resources** -> **Images** -> **Add Image** -> **Pull Image**, and pull the `hello-world` image.



3. Navigate **Networks** -> **+ Create Network**, and create a new network called "joeys-net".
So far, we can see that the user **joeyfull** has full access to create networks and pull images. This is because **joeyfull** has the *Full Access* default permission, giving them full access to create and manipulate new objects in UCP; note this does not confer the ability to see unlabeled objects deployed by other users.
4. Log out of UCP as **joeyfull** and log back in as **kellyres**.
5. Navigate **Resources** -> **Images**, and pull the `alpine` image.
6. Create a network called "kelly-net".
Similar to **joeyfull**, **kellyres** can also pull images and create networks despite only having the **Restricted Control** default permission. However, we saw above that **Restricted Control** blocks some actions, like starting an `exec` session in a container.
7. Can Kelly see the network "joeys-net"? Why or why not?
8. Log out of UCP as **kellyres** and log back in as **barryview**.

9. Navigate **Resources** -> **Images**.

Notice that Barry does not even have an **Add Image** button. This is because **barryview** has the **View Only** default permission. This permission does not allow operations such as pulling images.

10. Create a network called "barry-net".

Notice it won't let you create the Network but will highlight the **Permissions Label** field as follows:

NAME ?

barrys-net

PERMISSIONS LABEL [COM.DOCKER.UCP.ACCESS.LABEL]

Do not use a permissions label

Please select a permission label for your network

11. Select the **run** label on the **Permissions Label** field and then create the network. This time it should work.

Barry is able to create a network using a team label, since team labels also apply access control to networks. Previously the users Joey and Kelly were able to create networks without using any label because their *Default Permission was Full Control and Restricted Control*, thereby granting them the ability to create networks outside of their team permissions. Barry, however, only has *View Only* as his default permission.

12. What other users can see the network "barry-net"? Why?

13. Logout of UCP as **barryview** and login as **traceyno**.

14. Click on **Resources** and notice that Tracey cannot see the **Images** link.

This is because Tracey has the **No Access** default permission. However, because Tracey is a members of the Engineering team, they get access to all of the tagged resources that the Engineering team has access to. This includes the ability to create networks using one of the team labels.

15. Click the **Services** link and notice that Tracey can see the three services `nginx-labeled`, `ng1` and `ng2`

16. Create a network using either the **restricted** or **run** label.

6.10 Conclusion

In this exercise, you explored the effect of default user permissions and label-based team permissions on what assets a user can access. To summarize, a user can only see objects they have created themselves, or have at least view only access to via a team label. Furthermore, a user can only create objects they are able to have at least Restricted Control permissions to, again either by default (for creating unlabeled objects), or via a team permission label (for labeled objects).

7 User Management with LDAP

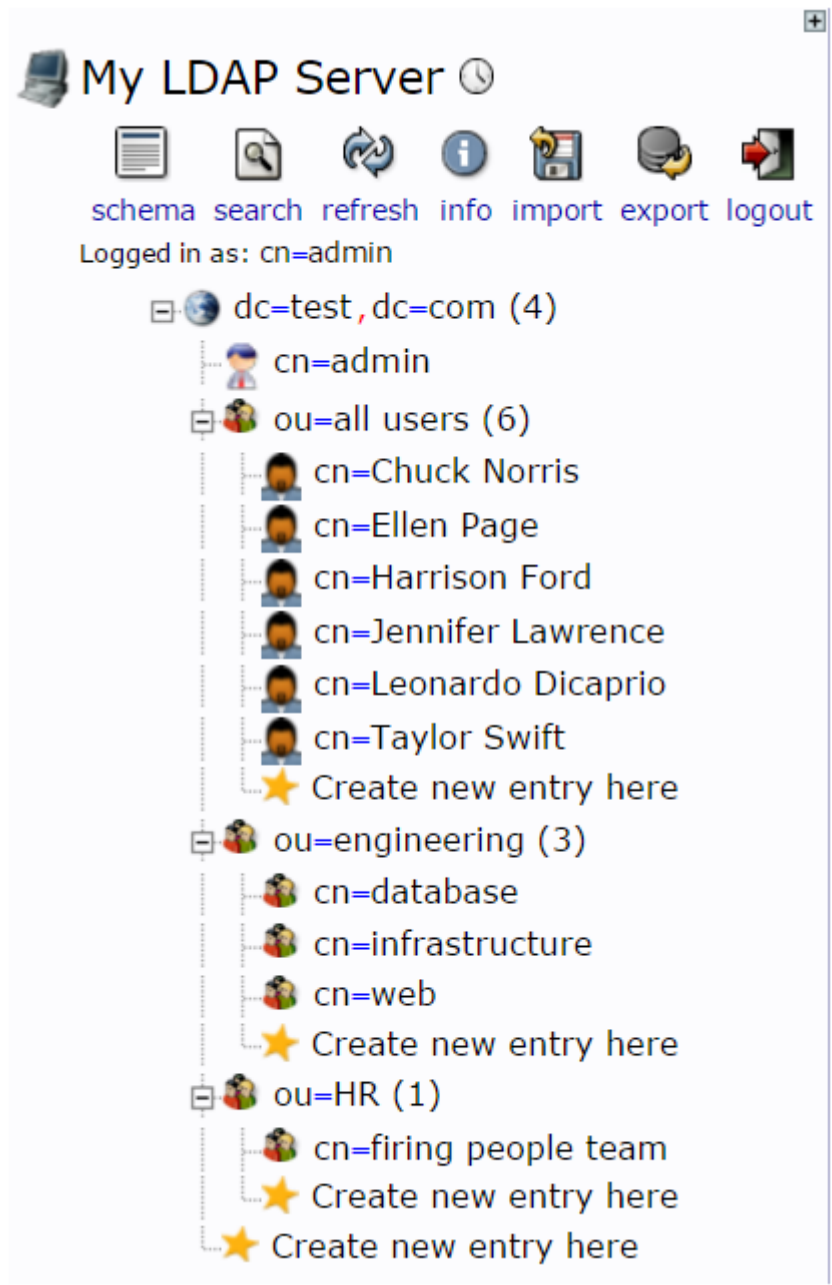
Instead of managing users directly from UCP, an LDAP server can define users.

Pre-requisites:

- UCP installation.
- Pre-configured LDAP server with users and groups.

7.1 Examine the LDAP server

1. Open a browser and go to `http://LDAP_SERVER/phpldapadmin` - LDAP_SERVER should be the domain name or public IP address of your LDAP node. This will open up the GUI for your LDAP server.
2. Login to the server with the following credentials:
Login DN: `cn=admin,dc=test,dc=com` **Password:** `admin`
3. Expand the left navigation bar and take note of the LDAP entries underneath `ou=all users`, `ou=engineering` and `ou=HR`



In this setup, we have 6 users.

7.2 Integrate UCP and LDAP

1. Login to UCP as the admin user.
2. navigate **Admin Settings -> Auth**, and change the method from **Managed** to **LDAP**.
3. Specify your **LDAP server FQDN**. Use the format `ldap://<LDAP_FQDN>`
4. Select **Full Control** on the **DEFAULT PERMISSION FOR NEW USERS** field
5. For the **LDAP Server Configuration** section, fill in the following details.

Field	Value
Recovery Admin Username	admin
Recovery Admin Password	orca1orca2
Reader DN	cn=Chuck Norris,ou=all users,dc=test,dc=com
Reader Password	password

6. For the **LDAP Security Options**, leave both options unchecked.
7. For the **User Search Configurations** section, fill in the following details.

Field	Value
Base DN	dc=test,dc=com
Username Attribute	uid
Full Name Attribute	cn
Filter	objectClass=inetOrgPerson

8. Under **SEARCH SCOPE** tick the **Subtree** radio button.

User Search Configurations

BASE DN ⓘ *

dc=test,dc=com

USERNAME ATTRIBUTE ⓘ

uid

FULL NAME ATTRIBUTE ⓘ

cn

FILTER ⓘ

objectClass=inetOrgPerson

SEARCH SCOPE ⓘ

☐ One Level ☒ Subtree

☐ Match Group Members ⓘ

9. Scroll down to the **Test LDAP Login** section and specify the following:

Field	Value
USERNAME	cnorris
PASSWORD	password

10. Click the **Test** button and verify that you get a **Success** message.
11. Click **Update Auth Settings**.

7.3 Test User Access

1. In the **LDAP Sync Status** section, click on the **Sync Now** button.

2. Click on **User Management** and check to see that all the users from our LDAP server are present. You will also notice that the previous users you created are still present.
3. Logout of UCP as the admin user. Try and login as user joeyfull. What do you notice?

Once UCP is integrated with LDAP, all user accounts will come from LDAP. **Managed** accounts that were previously setup in UCP will be disabled, except for the account that your specified in the **Recovery Admin Username** configuration field.

4. Login as the user cnorris. The password is password.
5. Logout as cnorris and log back in as admin. Use the password orca1orca2 as defined above.

7.4 Cleanup

To make sure that the subsequent exercises work as expected it is important to cleanup using the following steps:

1. Disable the LDAP integration by switching the Authentication back to **Managed**
2. **Important:** Make sure to re-enable all users (barryview, joeyfull, kellyres and traceyno) that have been previously disabled when we turned on LDAP authentication

Account: joeyfull ✕

[DETAILS](#)
[TEAM MEMBERSHIP](#)

[Save Changes](#)
[Cancel](#)

Full Name ? jf

Default Permissions ?
[No Access](#)
[View Only](#)
[Restricted Control](#)
[Full Control](#)
[Admin](#)

Account Status ?
[Enabled](#)
[Disabled](#)

Change User's Password

NEW PASSWORD ?

Client Bundles and Public Keys

[Create a Client Bundle](#)
[Add an Existing Public Key](#)

7.5 Conclusion

In this exercise, we replaced UCP's default managed users with a collection of users registered in an LDAP server; bear in mind that only one user management scheme - managed by UCP or managed by LDAP - is allowed at one time. If you wish to return to a UCP-managed scheme, all such accounts must be manually re-enabled after deactivating LDAP.

8 Password Recovery

In this exercise you will see how to recover from a scenario where the password to the built-in UCP admin account has been lost.

Pre-requisites:

- UCP configured to use the **Managed** authentication method (**Admin Settings** -> **Auth**).

8.1 Recovering admin passwords

1. SSH into the ucp-controller node.
2. If we do a quick `docker container ls` we can see all our UCP containers; the container we want to access is `ucp-auth-api`.
3. Run the following command:

```
$ docker container exec -it ucp-auth-api enzi \
"$(docker container inspect --format '{{ index .Args 0 }}' ucp-auth-api)" \
passwd --interactive
```

4. You will be prompted for the username. Specify `admin`, then choose a new password. Finally, check that you can log in to the UCP UI with your new credentials.

8.2 Conclusion

Note that node access to a UCP controller and the ability to `exec-attach` as per the above command is all that is required to gain admin control to UCP; make sure access to these nodes is appropriately restricted.

9 Troubleshooting User Access Scenarios

In the following, you will use your knowledge of access control labels in UCP to troubleshoot some access problems.

Pre-requisites:

- UCP installed and running.
- User accounts and labels described in exercises 'Access Control with Users, Teams and Labels' created.

9.1 Setup

1. Create a new user, **Chloe**, to lead your operations team:

Username	Full Name	Default Permissions
chloe	Chloe Operate	Full Control

2. Create a new team called **Operations**, and add **chloe** to this team.
3. Create the following label permissions for the **Operations** team:

Resource Label	Permission
production	Full Control
development	Full Control
qa	Full Control

4. Create the following label permissions for the **Engineering** team:

Resource Label	Permission
production	View Only
development	Full Control

Resource Label	Permission
qa	View Only

9.2 Container Access

You are a UCP admin. The user **Chloe** has launched an application with the following `docker-compose.yml`:

```

1 version: '3.1'
2
3 services:
4   javaclient:
5     image: training/hello-redis:1.0
6     labels:
7       com.docker.ucp.access.label: "production"
8   redis:
9     image: redis
10    labels:
11      com.docker.ucp.access.label: "production"

```

However, the application is not behaving correctly and Chloe has asked Joey to debug the application. However, Joey complains that running `docker exec` to connect to a container returns an `Access Denied` error.

1. Explain to Joey why this is the case. Discuss your answer with people around you.
2. Change the permissions so that Joey can `docker exec` into the container. Do this in such a way that only Joey and Chloe are able to `docker exec` into the container.

9.3 Asset Creation & Inspection

Barry wants to set up a custom network for his application deployment. However, when creating a network via the UCP GUI or the client bundle, Barry receives an `Access Denied` error message.

1. Explain to Barry why he is getting that message. Discuss your answer with people around you.
2. Change Barry's permissions so he is able to create the network.
3. Create a new user "Sam Tester"
4. Create a new team called "testing" and add Sam to the team
5. Login to UCP as Chloe and deploy an Ubuntu container, using the "qa" label.
6. Chloe has now asked the "testing" team to test the newly deployed Ubuntu container. However the "testing" team cannot see the new container. Fix the permissions so that the "testing" team can inspect the container.

10 Secret Management

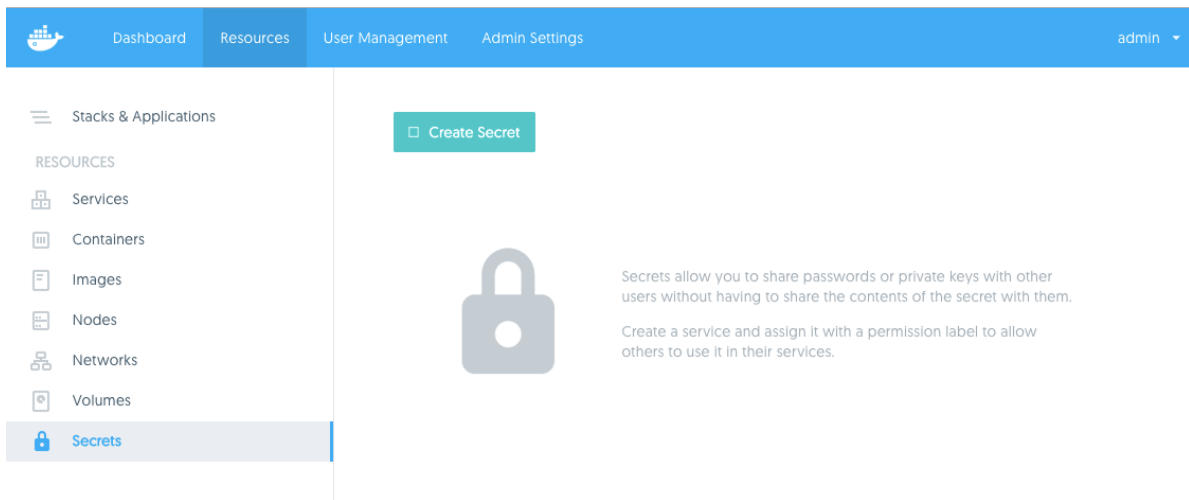
In this exercise, you'll explore creating and using secrets in UCP by creating a Wordpress app that requires a password to access its database backend.

Pre-requisites:

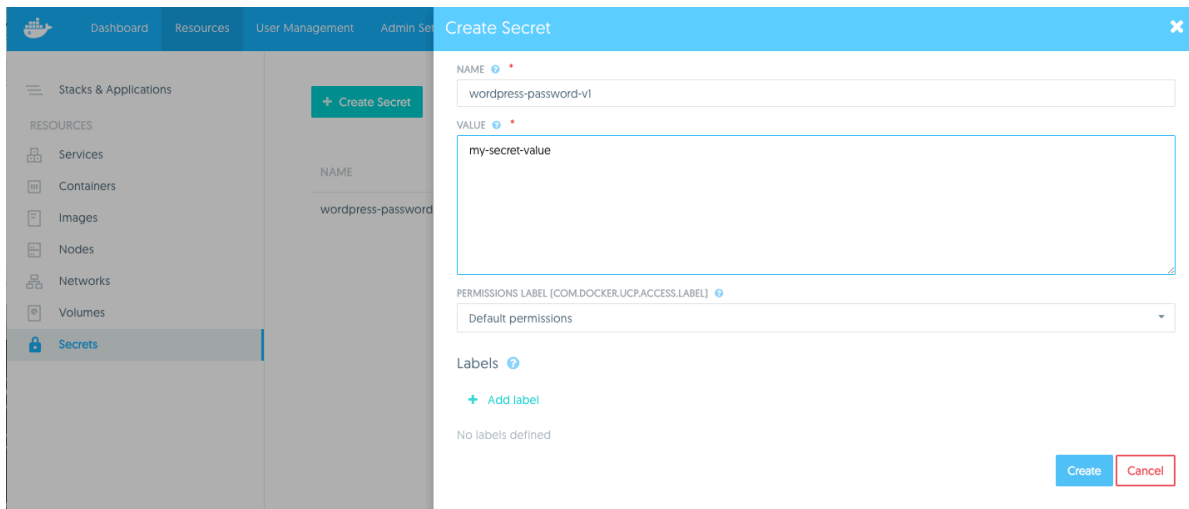
- UCP installed with at least two worker nodes.

10.1 Create a secret in UCP

1. Navigate **Resources -> Secrets -> + Create Secret**

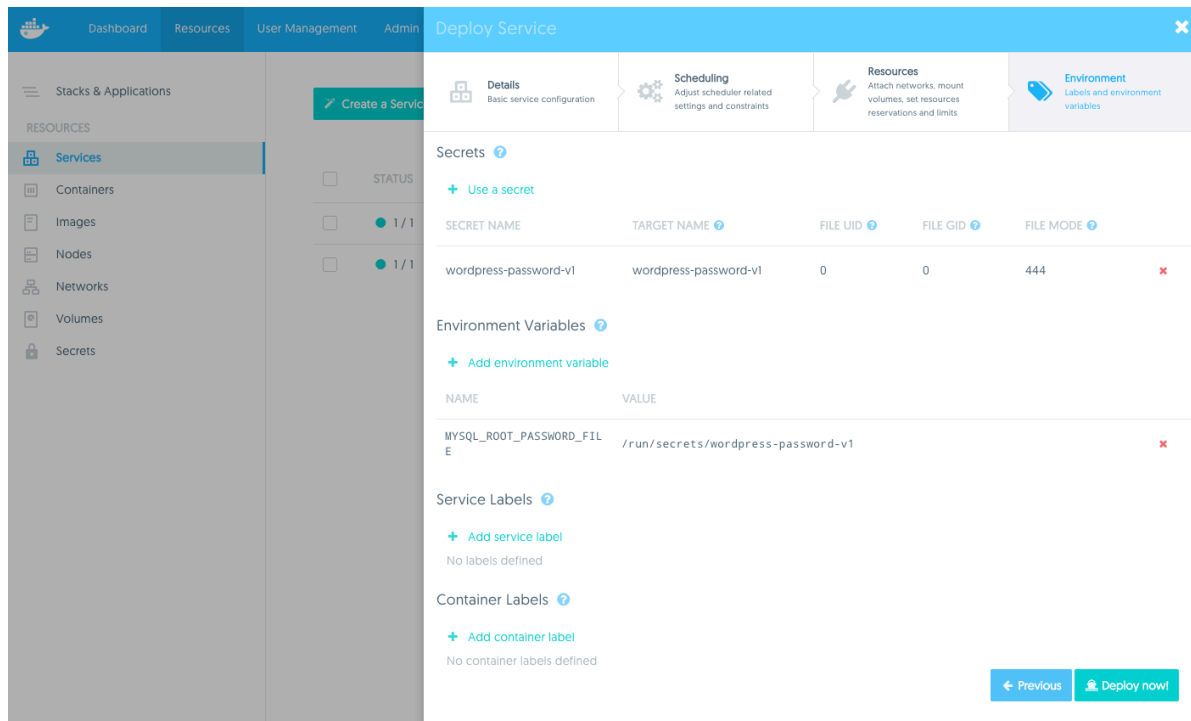


2. Create a secret named `wordpress-password-v1` and a value of `my-secret-value`.



10.2 Deploy MySQL Service with secret

1. Create a network called `wordpress-network`.
2. Deploy a service named `wordpress-db`, based on the `mysql:5.7` image, with the following additional specifications:
 - a) Attach the `wordpress-network` network
 - b) Under **Environment** -> **Secrets**, select **wordpress-password-v1** to associate this secret with this service.
 - c) Under **Environment** -> **Environment Variables**, create a variable called `MYSQL_ROOT_PASSWORD_FILE` with the value `/run/secrets/wordpress-password-v1`.

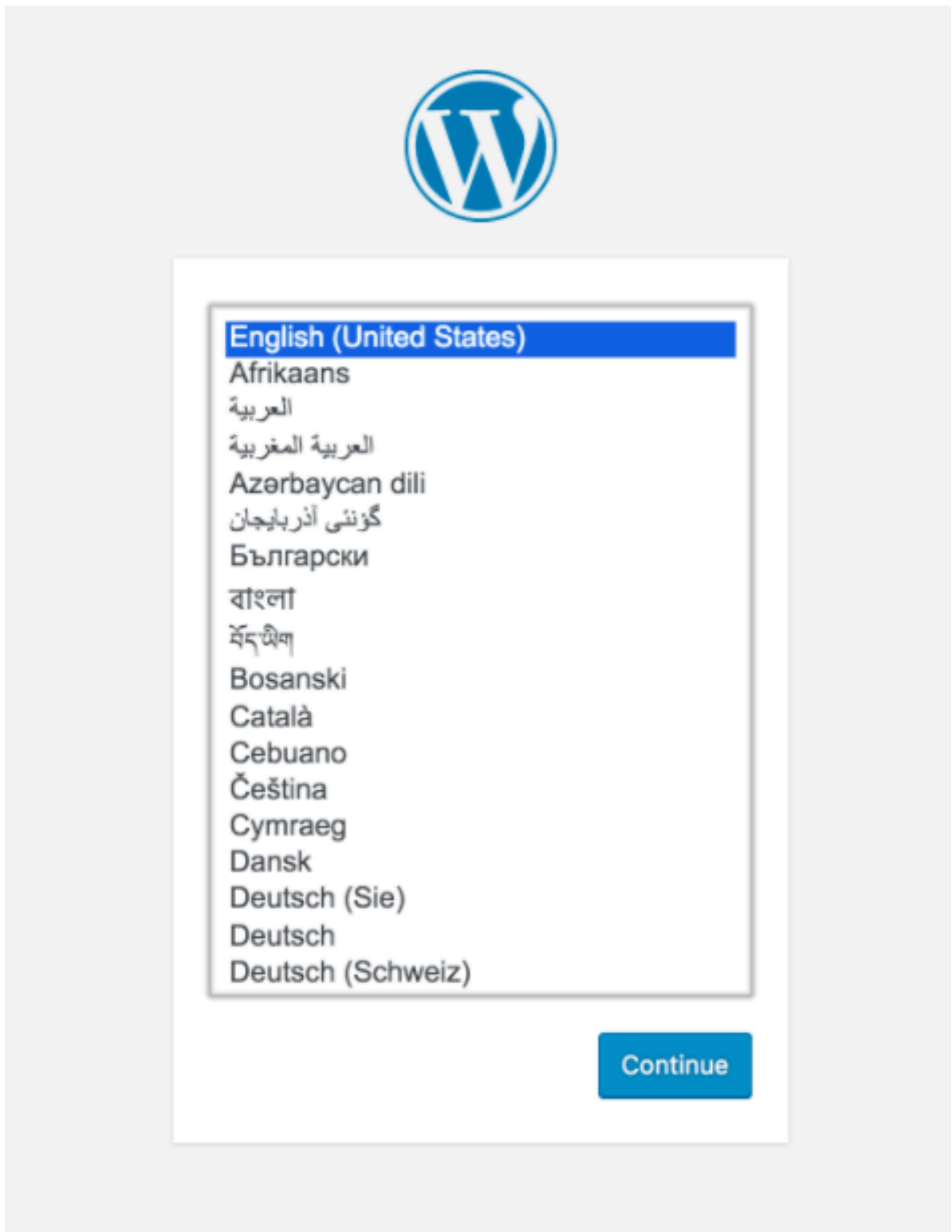


10.3 Deploy WordPress Service with secret

1. Deploy a WordPress service that uses MySQL as a storage backend with the following configuration.

Name	Value
Service name	wordpress
Image name	wordpress:latest
Published ports	80, ingress, 8080
Attached network	wordpress-network
Secret	wordpress-password-v1
Environment variable	WORDPRESS_DB_HOST=wordpress-db:3306
Environment variable	WORDPRESS_DB_PASSWORD_FILE=/run/secrets/wordpress-password-v1

2. Confirm your new blog is running by visiting it at <ucp node IP>:8080



10.4 Find a secret in a running container

1. In UCP, navigate **Resources** -> **Containers**, and determine which node is running your wordpress container.
2. SSH into that node, and attach to the container via `docker exec -it <container ID> bash`. Look for the Wordpress environment variables you defined:

```
$ env | grep WORDPRESS

WORDPRESS_DB_HOST=wordpress-db:3306
WORDPRESS_VERSION=4.7.2
WORDPRESS_SHA1=7b687f1af589c337124e6247229af209ec1d52c3
WORDPRESS_DB_PASSWORD_FILE=/run/secrets/wordpress-password-v1
```

cat the secrets file to verify that the secret value was delivered as you defined it:

```
$ cat /run/secrets/wordpress-password-v1
```

You should see your secret, made available to this container.

10.5 Conclusion

In this exercise, you set up a service (Wordpress) to have access to the password it needed to connect to its MySQL database. Note that at the end of the exercise, you saw the secret sitting unencrypted in the wordpress container; destination containers are the only place secrets are available unencrypted. Secrets are stored encrypted at rest in the Raft consensus database outside of services, and are transmitted to containers encrypted via mutual TLS. Furthermore, even in the destination container, the unencrypted secret is only present mounted in an in-memory, temporary filesystem on top of the container's usual union filesystem. When the container stops, the temporary filesystem containing the secret is destroyed.

11 Logging

11.1 Installing ELK Stack

In this section we are going to install an ELK stack which serves as a log aggregator and offers a Web UI for the users to drill into the logging data.

1. SSH into the node called `elk`. Prepare the node to run `elasticsearch` by running the following commands:

```
$ sudo sysctl -w vm.max_map_count=262144
$ sudo echo "vm.max_map_count=262144" >> /etc/sysctl.conf
```

2. Clone a GitHub repository which contains the code necessary to define and run the ELK stack:

```
$ git clone https://github.com/docker-training/elk-dee.git
$ cd elk-dee
```

3. Since we want to deploy the ELK stack as a (Docker) **stack** we need to make the node part of a swarm. To do this run:

```
$ docker swarm init
```

4. Finally we deploy the ELK stack:

```
$ docker stack deploy -c elk-docker-compose.yml elk
```

Double check that everything runs by using this command:

```
$ watch docker service ls
```

and wait until every service is running. You should see something like this:

```
Every 2.0s: docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE
13sksh79tim2	elk_elasticsearch	replicated	1/1	elasticsearch:5.2

cluyclmtaw48	elk_logstash	replicated	1/1	logstash:5.2-alpine
whni0r8ddm8i	elk_kibana	replicated	1/1	kibana:5.2

specifically note the column **Replicas** saying 1/1 for every service. This indicates that we're ready to roll.

11.2 Configuring all Swarm nodes

Now we need to configure the swarm that every node in it reports all its logs to the ELK stack. We need to set up Docker daemon logging configuration to default to using `journald`. The reason we're doing this is to ensure that we can have a local copy of the logs to be able to use docker logs and docker service logs. Note that at the time of this writing **docker service logs** is still in experimental mode.

1. SSH into the `ucp-controller` node.
2. Add this (recommended) logging configuration to the file `/etc/docker/daemon.json`:

```
{
  "log-driver": "journald",
  "log-level": "debug",
  "log-opts": {
    "tag": "{{.ImageName}}/{{.Name}}/{{.ID}}"
  }
}
```

3. Restart the Docker daemon:

```
$ sudo service docker restart
```

Note: If you cannot restart the Docker daemon you can send it a `SGHUP` signal instead.

4. Repeat the above steps **every single** node of the swarm.

11.3 Stream all Docker logs to ELK

Now that we have configured all nodes to generate their respective logs using the `journald` driver we need to stream the data to our ELK stack.

1. SSH into one of the manager nodes, e.g. our `ucp-controller`.
2. Run the following commands on this swarm node:

```
$ git clone https://github.com/docker-training/elk-dee.git
$ cd elk-dee
$ export LOGSTASH_HOST=<ELK NODE IP>
$ docker stack deploy -c journalbeat-docker-compose.yml journalbeat
```

where `<ELK NODE IP>` corresponds to the **internal** IP of the `elk` node. The result of this can be visualized as follows:

3. Make sure the logs are forwarded correctly to the ELK node by using the `journalctl` tool on the current node:

```
$ journalctl --since "2017-04-03 22:20"
```

Replace the date/time stamp in the above command with the current date/time minus a few minutes. Watch out for events from `dockerd` and make sure there are no DNS resolution or connection errors visible in the log.

4. Open a browser at `http://<ELK_PUBLIC_IP_ADDRESS>:5601` to access Kibana. Upon first usage you should be asked to create an index for the events before you can proceed. Accept the defaults and click **Create**.
5. In Kibana navigate to **Discover** and view the list of captured events.

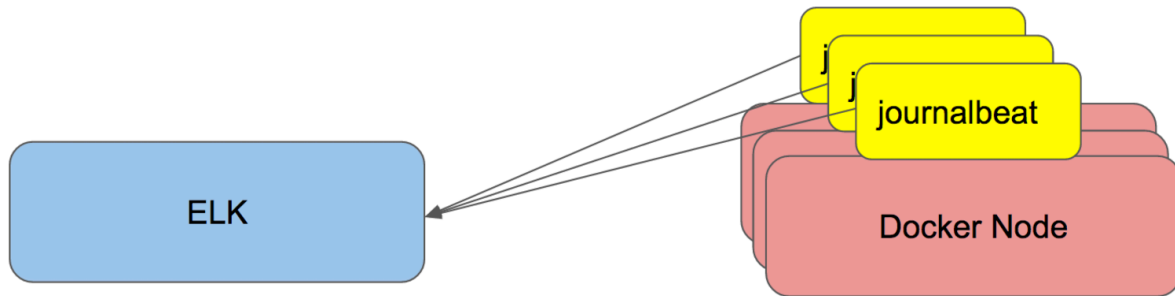
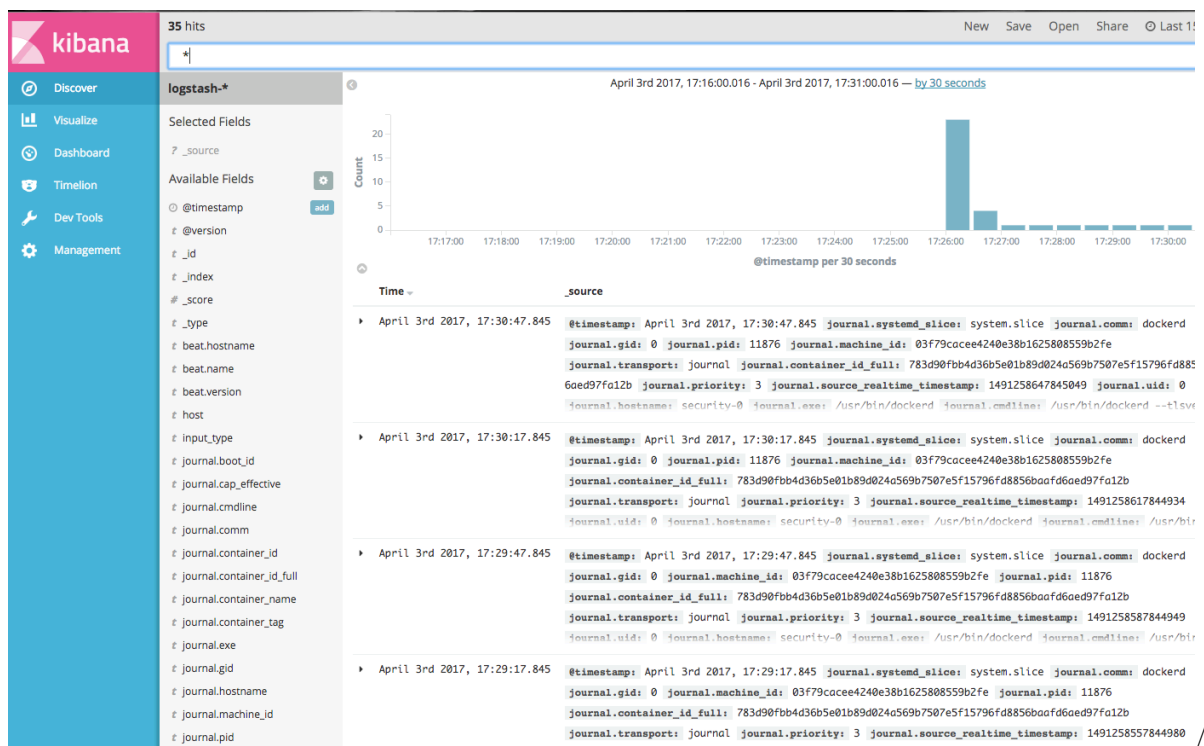


Figure 1: logging

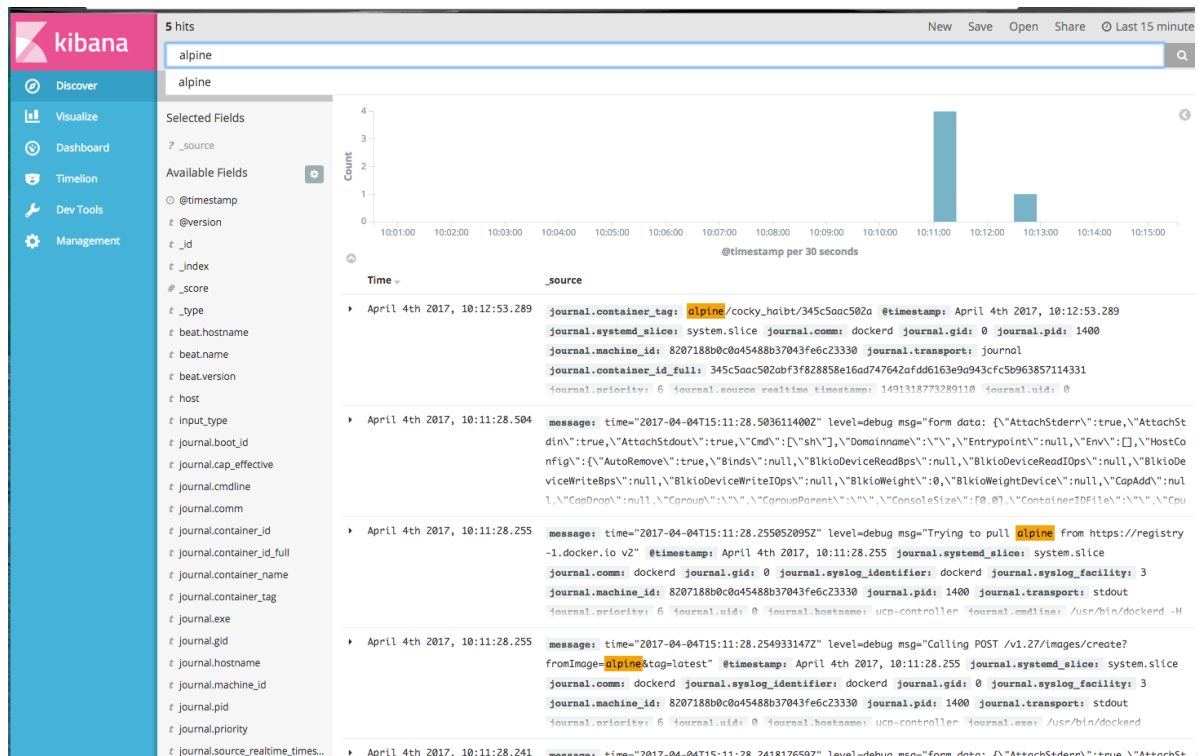


6. If you don't see any events yet then try to run a container:

```
$ docker run --rm -it alpine sh
```

Events should pour in and appear in Kibana as you do so (refresh the browser window)...

7. In the filter box at the top of the Kibana page enter `alpine` and hit enter. Events containing the word `alpine` should be filtered out:



12 Health Checks

In this exercise we are going to show how the health check mechanism for services works. For this we're going to use a simple service that provides a `/health` endpoint which returns either OK (status 200) or NOT OK (status 500) depending on an internal setting. Initially the service reports its status as healthy. We can then call the endpoint `/kill` to put the service in an unhealthy status.

The source code to this service can be found here: <https://github.com/docker-training/healthcheck>.

12.1 Analyzing the Dockerfile

1. First let's look at the Dockerfile for this service. It looks as follows:

```
1 FROM ubuntu:16.04
2
3 RUN apt-get update && apt-get -y upgrade
4 RUN apt-get -y install python-pip curl
5 RUN pip install flask==0.10.1
6
7 ADD /app.py /app/app.py
8 WORKDIR /app
9
10 HEALTHCHECK CMD curl --fail http://localhost:5000/health || exit 1
11
12 CMD python app.py
```

2. Please specifically note the second to last line in the Dockerfile where the command is defined with which the health of the process is determined at runtime. Evidently the command probes the endpoint `/health` using `curl`.
3. Have a look at the application code itself; the most important part for healthchecking is the `/health` route:

```

1  @app.route('/health')
2  def health():
3      global healthy
4
5      if healthy:
6          return 'OK', 200
7      else:
8          return 'NOT OK', 500

```

The app performs some logic to decide if it is healthy or not when /health is visited. This toy example just checks a bit, but a real example would have the same structure.

12.2 Writing a Compose File

Next we are going to write a compose file that we'll use to define a stack to be deployed into the swarm.

1. Create a file called my-stack.yml and add the following content:

```

1  version: '3.1'
2  services:
3      app:
4          image: training/healthcheck:17.03
5          ports:
6              - 5000:5000
7          healthcheck:
8              interval: 2s
9              timeout: 2s
10             retries: 3

```

2. Save the file.

12.3 Deploying and using the Service

1. Deploy the stack using the above compose file:

```
$ docker stack deploy -c my-stack.yml test
```

2. Now open a second terminal and run the following command to observe the paas service:

```
$ watch docker service ps test_app
```

you should see something like this:

```
Every 2.0s: docker service ps test_app
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
ralw1apn8mgs	test_app.1	training/healthcheck:17.03	ucp-node-0	Running	Running 25 seconds

As we can see, the service is up and running happily.

3. Open yet another terminal and ssh into the node on which the app is running. In the above sample it would be node ucp-node-0. On the node execute

```
$ docker system events
```

to observe the event stream generated by the app container.

4. But now we want to disrupt this peaceful state a bit. In your first terminal execute the command to put the service into an unhealthy status:

```
$ curl localhost:5000/kill
```

This flips the health bit in our app, so it starts reporting as unhealthy.

5. Observe what's happening in the terminal where you run the `docker system events` command. Did you notice how container `health_status: unhealthy` is reported? After the third occurrence container `kill` and container `die` events are triggered.
6. Observe what's happening in the second terminal where you run the `watch` command. After approximately 8 seconds you should see that the running instance gets killed and a new instance is started instead.
7. Explain the delay between the `kill` command and the moment the service gets effectively killed.
8. Now execute the `kill` command again and observe what's happening. Eventually you should see something like this:

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
pd9fri908svl	test_app.1	training/healthcheck:17.03	moby	Running	Running 2 minutes a
zizfogf5dhoz	_ test_app.1	training/healthcheck:17.03	moby	Shutdown	Failed 2 minutes a
s7ui88soxzlt	_ test_app.1	training/healthcheck:17.03	moby	Shutdown	Failed 2 minutes a

9. To get some detailed information about the last five healthchecks on a container, find the node the container is running on and do:

```
$ docker inspect --format '{{json .State.Health.Log}}' <container id> | json_pp
```

10. Clean up the system by removing the stack:

```
$ docker stack rm test
```

12.4 Summary

In this exercise we have shown a sample of how we can define a command in our `Dockerfile` which reports the health status of the containerized service. We have created a `compose` file that defines how frequently the health check should be executed and how many times we should allow the check to fail before the system kills the container running the service and starts a new instance instead.

Each service should implement a health check endpoint. It is advised to come up with a convention in your company or team about how the health-check endpoint of each service shall be called and what the possible responses are.

Next to the health check endpoint we need to define a command in the `Dockerfile` which is used to probe the endpoint and finally we can define upon creation of the service (e.g. in the `docker-compose` file) how frequently the endpoint shall be probed as well how many failures shall be tolerated until the service is reported as unhealthy.

13 Demo: Install DTR

In this demo, we'll walk through installing Docker Trusted Registry with an S3 storage backend.

Pre-requisites:

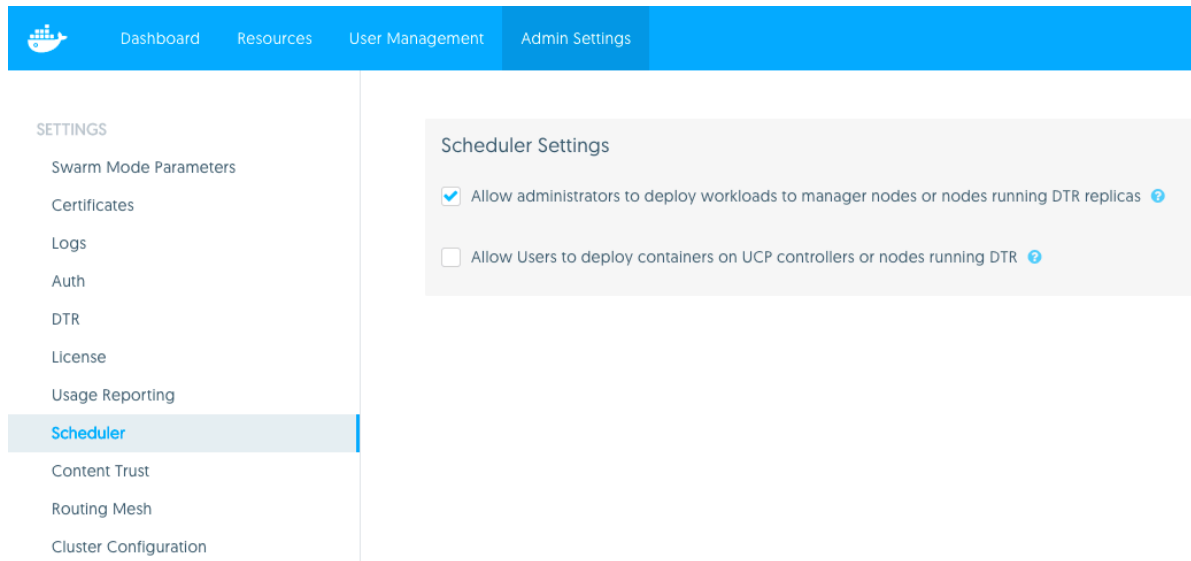
- UCP installed with 2 worker nodes
- Three additional nodes with the following names:
 - `dtr-replica-0`
 - `dtr-replica-1`
 - `dtr-replica-2`

13.1 Add additional worker nodes to UCP

DTR is installed on top of UCP worker nodes. Start by adding all three DTR nodes to UCP as worker nodes.

13.2 Install DTR

1. Navigate **Admin Settings** -> **Scheduler**.
2. Tick the box that says **Allow Administrators to deploy workloads to manager nodes or nodes running DTR replicas**



We need to enable this in order to be able to install DTR on one of our worker nodes.

3. On `ucp-controller`, run the following command to install DTR via the `docker/dtr` bootstrap image:

```
$ docker container run -it --rm docker/dtr install \
--ucp-node dtr-replica-0 --ucp-insecure-tls
```

4. The first prompt you will get is for the **dtr-external-url**. This is the URL that users will use to access DTR. You must specify either the public IP or domain of the node you are installing DTR on.
5. Specify the URL to UCP along with the UCP admin username and password when prompted. The UCP URL is the URL to the node where you first installed UCP.

```
ubuntu@dtr-replica-0:~$ sudo docker container run -it --rm docker/dtr install --ucp-node dtr-replica-0 --ucp-insecure-tls
sudo: unable to resolve host dtr-replica-0
Unable to find image 'docker/dtr:latest' locally
latest: Pulling from docker/dtr
0a8490d0dfd3: Pull complete
038e0d7519a6: Pull complete
491ebdb8b529: Pull complete
4a056d14f78f: Pull complete
c58575ca6007: Pull complete
8611590f8f6c: Pull complete
0012584aa72c: Pull complete
d3cf84a66c2e: Pull complete
10c876db9e47: Pull complete
7247eaf14872: Pull complete
Digest: sha256:24fa48e5172e4d5b1caba71bcc1f9d39e53e9ea2d6123c8331a3922c0ed6fd09
Status: Downloaded newer image for docker/dtr:latest
INFO[0000] Beginning Docker Trusted Registry installation
dtr-external-url (URL of the host or load balancer clients use to reach DTR. Format https://host[:port]): ec2-54-167-207-196.compute-1.amazonaws.com
ucp-url (The UCP URL including domain and port): ec2-52-90-8-93.compute-1.amazonaws.com
ucp-username (The UCP administrator username): admin
ucp-password:
INFO[0107] Validating UCP cert
INFO[0107] Connecting to UCP
INFO[0108] UCP cert validation successful
INFO[0109] The UCP cluster contains the following nodes: dtr-replica-2, ucp-manager-0, ucp-node-1, ucp-manager-1, dtr-replica-0, ucp-node-0, dtr-replica-1
INFO[0000] Validating UCP cert
INFO[0000] Connecting to UCP
```

13.3 Check that DTR is running



















1. Navigate **Resources** -> **Stacks & Applications** in UCP. You should see DTR listed and when expanded, there should be 9 services running.

Applications: Docker Trusted Registry 2.2.1 - [Replica 17752bbaad0e]

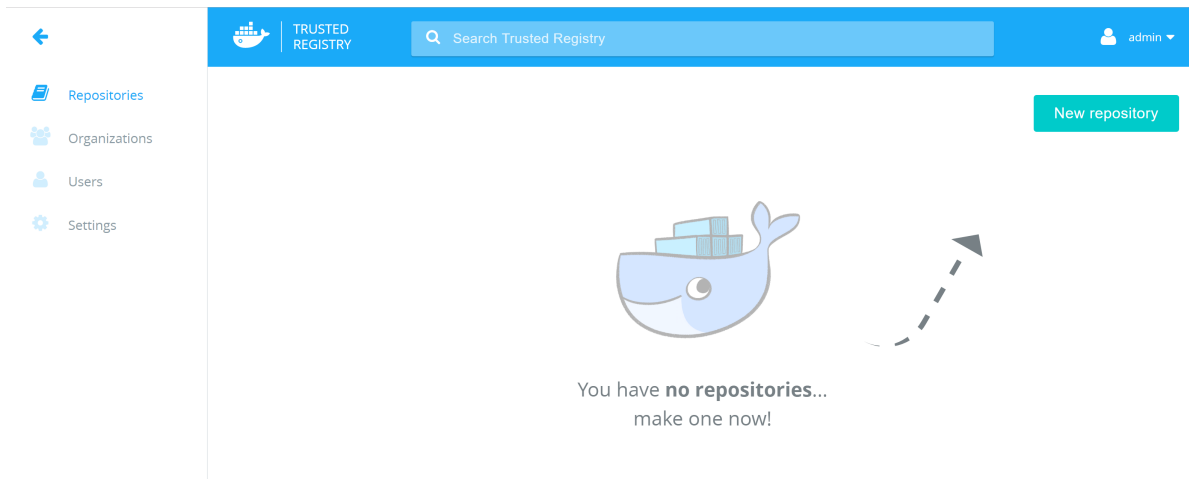
CONTAINERS [9] VOLUMES [0] NETWORKS [0]

Actions

Search containers...

ID	NODE	NAME	IMAGE	CREATED
 ad542a1344e3	dtr-replica-0	dtr-scanningstore-17752bbaad0e	docker/dtr-postgres:2.2.1	 5 minutes ago
 4c296e80642a	dtr-replica-0	dtr-notary-signer-17752bbaad0e	docker/dtr-notary-signer:2.2.1	 5 minutes ago
 74e25bdfff36	dtr-replica-0	dtr-jobrunner-17752bbaad0e	docker/dtr-jobrunner:2.2.1	 5 minutes ago
 0ad920eb22ed	dtr-replica-0	dtr-nginx-17752bbaad0e	docker/dtr-nginx:2.2.1	 5 minutes ago
 e87d4c6cf87b	dtr-replica-0	dtr-notary-server-17752bbaad0e	docker/dtr-notary-server:2.2.1	 5 minutes ago
 0b9138cc734c	dtr-replica-0	dtr-api-17752bbaad0e	docker/dtr-api:2.2.1	 5 minutes ago
 c7384cf97d00	dtr-replica-0	dtr-garant-17752bbaad0e	docker/dtr-garant:2.2.1	 5 minutes ago
 e75a8761e4b4	dtr-replica-0	dtr-registry-17752bbaad0e	docker/dtr-registry:2.2.1	 6 minutes ago
 6f9a005e75d4	dtr-replica-0	dtr-rethinkdb-17752bbaad0e	docker/dtr-rethink:2.2.1	 6 minutes ago

2. Open a new browser tab and go enter the URL to your DTR installation. You will have to add a security exception in you browser when prompted, like you did when installing UCP.



13.4 Configure DTR to use S3

Since we will run DTR in high availability mode on multiple nodes we need to configure it to use shared storage. In our case we will use an S3 bucket.

1. In your AWS console select the S3 service and create a new bucket.
2. Login to your DTR as **admin**.
3. Navigate **Settings** -> **Storage**.
4. Select **S3** and fill out the form below.

docker trusted registry

Search

admin

Settings > Storage

GENERAL **STORAGE** SECURITY GARBAGE COLLECTION

Set up your storage with a ☒ Manual form ☐ YAML file

Storage backend

☐ Filesystem ☒ S3 ☐ Azure ☐ Swift ☐ Google Cloud Storage

S3 settings

Root directory ⓘ – Where registry files will be stored

AWS region name ⓘ – Where you want to store objects

us-east-1

S3 bucket name ⓘ – Where you want to store objects

gschenker-dtr-storage

AWS access key ⓘ

XXXXXXXXXXXXXXXXXXXX

AWS secret key ⓘ

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Region Endpoint ⓘ (optional) – Where to find the S3 region

Show advanced settings

Save

- Click **Save**

13.5 Integrate UCP and DTR

By default Docker engine uses TLS when pushing and pulling images to an image registry like Docker Trusted Registry; as such, each Docker engine must be configured to trust DTR.

- Switch into your `ucp-controller` terminal.
- Download the DTR certificate by running the following command (replace `dtr-domain-name` with the domain of your DTR instance):

```
$ sudo curl -k https://<DTR_FQDN>/ca \
-o /usr/local/share/ca-certificates/<DTR_FQDN>.crt
```

- Refresh the list of certificates to trust:

```
$ sudo update-ca-certificates
```

- Restart the Docker engine:

```
$ sudo service docker restart
```

- Log in to DTR from the command line:

```
$ docker login <DTR_FQDN>
```

If your login is successful, you have configured your Docker engine to trust DTR.

- Repeat steps 2-5 on all your VMs; every Docker engine in the Swarm cluster must be configured to trust DTR.

Note: To speed up the whole process of configuring all nodes of our swarm we can use a bash script like the following:

```

1 CERT=./<your-pem-name>.pem
2 DTR_FQDN=<DTR_FQDN>
3 NODE_LIST=<LIST-IP-ADDRESSES-OF-NODES-IN-SWARM>
4 # e.g. ("52.23.162.247" "52.3.234.184" "52.205.102.106" ...)
5 for NODE in "${NODE_LIST[@]}; do
6     ssh -i $CERT ubuntu@$NODE hostname
7     ssh -i $CERT ubuntu@$NODE sudo curl -k https://$DTR_FQDN/ca \
8     -o /usr/local/share/ca-certificates/$DTR_FQDN.crt
9     ssh -i $CERT ubuntu@$NODE sudo update-ca-certificates
10    ssh -i $CERT ubuntu@$NODE sudo service docker restart
11 done;
```

Replace the placeholders with the respective values of your infrastructure.

13.6 Test the integration

- Open your browser to DTR and click on the **New Repository** button to create a repository called `hello-world`. The repository should go under your **admin** account:

- On ucp-controller, pull the `hello-world` image from Docker Store: `docker pull hello-world:latest`
- Re-tag the image :

```
$ docker tag hello-world:latest <DTR_FQDN>/admin/hello-world:1.0
```

- Push to DTR:

```
$ docker push <DTR_FQDN>/admin/hello-world:1.0
```

and verify that you can see your image both in DTR and in your S3 bucket.

13.7 Conclusion

In this exercise we have installed Docker Trusted Registry, which is now running on top of UCP, with an S3 backing for your images. Note that DTR installation expects ports 443 and 80 to be available; if they aren't, double check whether one of those ports are occupied by a service in UCP (including the HTTP routing mesh). Furthermore, x509 errors when logging into DTR are usually a symptom of not correctly setting up trust between Docker engine and DTR.

14 Demo: Install DTR Replicas

In this exercise, the instructor will demo setting up 2 additional DTR replicas in order to implement high availability.

Pre-requisites:

- UCP installed with 2 worker nodes
- DTR installed and integrated with UCP
- The following nodes joined to UCP as workers:
 - dtr-replica-1
 - dtr-replica-2

14.1 Install Replicas

1. On your ucp-controller node, run the docker/dtr bootstrapper to configure dtr-replica-1 as a DTR replica:

```
$ docker container run -it --rm docker/dtr join \
--ucp-node dtr-replica-1 --ucp-insecure-tls
```

2. Accept the default ID of your existing DTR installation when prompted.
3. Run the bootstrapper again but this time, specify to install the third replica on the dtr-replica-2 node.
4. Check the **Applications** page inside the UCP web UI. You should now see three instances of DTR.

Application Name	Running	Exited	Action
Docker Trusted Registry 2.1.1 - (Replica 0c08c7ed611b)	8	0	Show Containers (8)
Docker Trusted Registry 2.1.1 - (Replica 15d25c8e4fle)	8	0	Show Containers (8)
Docker Trusted Registry 2.1.1 - (Replica 834c3ba5fbc2)	8	0	Show Containers (8)
Docker Universal Control Plane PROZ:EDGZ:2UBM:BXSN:JJPW:QFZI:ZM65:PUVJ:TCFN:3G6G:NGGR:MOUO	29	0	Show Containers (29)

5. Verify that you can visit the DTR dashboard via the public IP of both dtr-replica-1 and dtr-replica-2.

14.2 Cleanup

1. Navigate **Admin Settings** -> **Scheduler** in UCP

2. Untick both checkboxes that says “**Allow Administrators to deploy workloads to manager nodes or nodes running DTR replicas**”

This is so that later on, when we run containers, we do not accidentally get them scheduled on our DTR nodes or on our UCP manager nodes.

14.3 Conclusion

After completing this exercise, your DTR instance is running in high availability mode. Just like the UCP controller consensus, the DTR replicas maintain their state in a rethink database, and elect a leader via a Raft consensus.

15 Create a DTR Repository

In this exercise, you will tour the basics of creating user accounts on DTR. You will then use these accounts to push and pull images.

Pre-requisites:

- Instructor has DTR installed.
- Student has UCP set up with at least one manager node.

15.1 Setup

1. The instructor will provide the admin username and password to their instance of DTR; use this information to log in and create yourself a user account in DTR.
2. Log out of the DTR admin account.
3. Log into DTR from one of your UCP terminals with your new credentials via `docker login <dtr url>`.
4. Integrate your UCP deployment with DTR by following the ‘Integrate UCP and DTR’ step in the ‘Install DTR’ example in this book; use the public DNS of your instructor’s first DTR node.

15.2 Create a Repo

1. Log in to the Docker Trusted Registry web UI using your new account.
2. Create a new repository, named however you like. Set the visibility to “public”.

The screenshot shows the 'New repository' form in the Docker Trusted Registry web UI. The form is titled 'Repositories' and has a 'New repository' button. It contains the following fields and options:

- ACCOUNT:** A dropdown menu with 'student1' selected.
- REPOSITORY NAME:** A text input field with 'my-repo' and a green checkmark.
- DESCRIPTION (OPTIONAL):** A text input field with 'test repo' and a green checkmark.
- VISIBILITY:** Two radio buttons: 'Public' (Visible to everyone) and 'Private' (Hide this repository). The 'Public' option is selected.
- Buttons:** 'Cancel' and 'Save' buttons at the bottom right.

15.3 Push an image into the repository

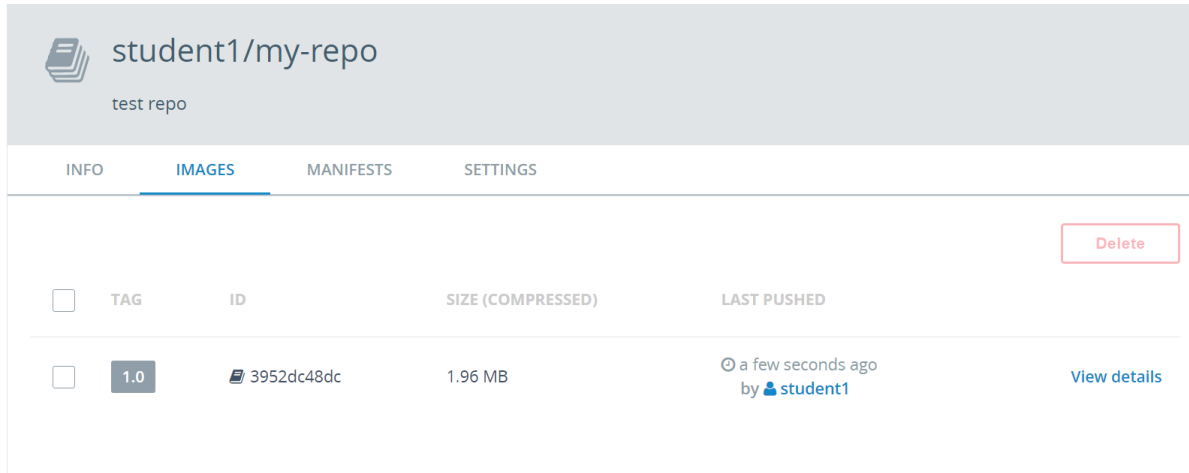
1. From your ucp-controller node, pull the alpine image.
2. Re-tag alpine with your DTR repository name and a tag of 1.0:

```
$ docker image tag alpine \
<DTR-URL>/<username>/<your-repo>:1.0
```

3. Push the image into DTR:

```
$ <DTR-URL>/<username>/<your-repo>:1.0
```

4. Go to the repository on the DTR web UI and verify that you now have a 1.0 tag on the repository.



15.4 Pull and Push an image from a DTR Repository

1. Log out of DTR from the command line:

```
$ docker logout <DTR-URL>
```

2. Find a partner in the class, and pull their image:

```
$ docker pull \
<DTR-URL>/<partner-username>/<partner-repo>:1.0
```

Notice this works even when logged out, since these are public repos.

3. Re-tag the image you just pulled with your own DTR namespace and the tag 1.1:

```
$ docker image tag \
<DTR-URL>/<partner-username>/<partner-repo>:1.0 \
<DTR-URL>/<username>/<your-repo>:1.1
```

Push to DTR:

```
$ docker image push \
<DTR-URL>/<username>/<your-repo>:1.1
```

Of course this fails, because you aren't logged in. Log back into DTR and push again, then confirm you can see your 1.1 tag in the DTR web UI.

4. Finally, re-tag your partner's image with their namespace and the tag partner, and try to push to *their* repo:

```
$ docker image tag \
<DTR-URL>/<partner-username>/<partner-repo>:1.0 \
<DTR-URL>/<partner-username>/<partner-repo>:partner
```

```
$ docker image push \  
<DTR-URL>/<partner-username>/<partner-repo>:partner
```

The push fails, since even though this is a public repo, it belongs to your partner; only they and admins can push to it at this time.

15.5 Conclusion

In this demo and exercise, we saw how user-owned repos appear in DTR; only owners and admins can write to user-owned repos, even if they are public. Org members can, however, see public user-owned repos for other users in the same org.

16 Working with Organizations and Teams

Pre-requisites:

- UCP integrated with instructor's DTR
- At least one Docker engine configured to trust DTR certificate.
- User account created on instructor's DTR

16.1 Demo: Creating Organizations and Teams

First, the instructor will demo setting up organizations and teams for everyone to participate in:

1. In the Engineering org, the instructor will create two teams: database and web, and add half the user accounts to each.
2. Also as part of the Engineering org, the instructor will create two repos: db and ui.
3. Finally, the instructor will give team database R/W permission to db and R/O access to ui, and vice-versa for team web.

16.2 R/W in Org Repos

1. Log into the DTR web console, and click on Repositories on the left sidebar to see a list of all repos you have read access to. Use the dropdown to filter only repos that belong to the Engineering org.
2. At the console, tag and push a copy of alpine to your team's repo (db for team database and ui for team web). Tag it with your lucky number, so we get a collection of different tags:

```
$ docker image tag alpine \  
ec2-54-213-179-82.us-west-2.compute.amazonaws.com/engineering/your-repo:13.0  
  
$ docker image push \  
ec2-54-213-179-82.us-west-2.compute.amazonaws.com/engineering/your-repo:13.0
```

3. Once you can see some tags in the other team's repo, try pulling one of them. Try retagging what you pulled, and pushing it back to the other team's repo; the pull will work but the push will fail, since you have read only access to these repos.
4. Next, have the instructor set both db and ui to private repos (Repositories -> db -> Settings -> Private). Go to the web UI - can you see the other team's repo? Finally, have the instructor revoke your team's read only permission for the other team's repo (Organizations -> Engineering -> database -> Repositories -> 'x' beside Read Only permission for ui). Refresh your DTR browser - can you see the other team's repo now?

16.3 Conclusion

In this exercise, we saw how organization-owned repositories appear on DTR. Users can read and write to repos based on the permissions afforded them by their team; private org-owned repos become invisible to any user without at least explicit read access to that repo.

17 Enabling Image Signing






In this exercise you will install and use Notary to sign and push a trusted image.

Pre-requisites:

- Instructor has DTR installed
- Instructor has created a DTR account for each student
- Instructor has added each student account to the **Engineering** organization as an Admin.
- Docker engine on **ucp-controller** configured to trust the instructor DTR instance

17.1 Install the Notary CLI

1. Open your web browser and go to <https://github.com/docker/notary/releases>. Look for the **Downloads** section of the latest release on the page:

Downloads	
 notary-Darwin-amd64	8.09 MB
 notary-Linux-amd64	9.39 MB
 notary-Windows-amd64.exe	7.78 MB
 Source code (zip)	
 Source code (tar.gz)	

2. Right click on the **notary-linux-amd64** link and copy the link location.
3. SSH into the **ucp-controller** VM.
4. Run `wget` and paste in the link location:


```
$ wget https://github.com/docker/notary/releases/download/v0.4.3/notary-Linux-amd64
```
5. Copy the downloaded binary file into the `/usr/local/bin` folder and rename it to just `notary`:


```
$ sudo cp notary-Linux-amd64 /usr/local/bin/notary
```
6. Set the permissions on the notary client:


```
$ sudo chmod +x /usr/local/bin/notary
```

17.2 Configure the Notary CLI

Notary commands require pointers to the Notary server, and DTR certificates; after acquiring the Notary binary, it's convenient to alias the `notary` command to handle this automatically:

1. Get a copy of the DTR CA certificate:


```
$ curl https://<DTR_FQDN>/ca > dtr-ca.pem
```

Where DTR_FQDN is the public IP address or DNS name to the DTR.

2. In your ~/.bashrc file, add the following alias:

```
alias notary="notary -s https://<DTR_FQDN> -d ~/.docker/trust --tlscacert ~/dtr-ca.pem"
```

3. Source your new .bashrc so the changes take effect:

```
$ source ~/.bashrc
```

17.3 Set Up Content Trust for a repository

1. Login to the DTR web client with your user account.
2. Create a new repository under the **Engineering** organization called <yourname>-enterprise-app. Make the repo private and replace with your DTR username.
3. Check if Notary has content trust data for your yourname-enterprise-app repository by running, back at the command line:

```
$ notary list <DTR_FQDN>/engineering/yourname-enterprise-app
```

When prompted for a username and password, enter your DTR credentials. Notice the error message since we do not have any trust data at the moment.

4. Initialise the Content trust collection by running:

```
$ notary init -p <DTR_FQDN>/engineering/<yourname>-enterprise-app
```

You will be prompted to pick as passphrase for a **root key**, **targets key** and **snapshot key**. Do not lose these.

Note: If you get an error like this: fatal: you are not authorized to perform this operation: server returned 401., make sure you're not using https:// in <DTR_FQDN> but only the public IP address or DNS name of your assigned DTR.

17.4 Push a signed image

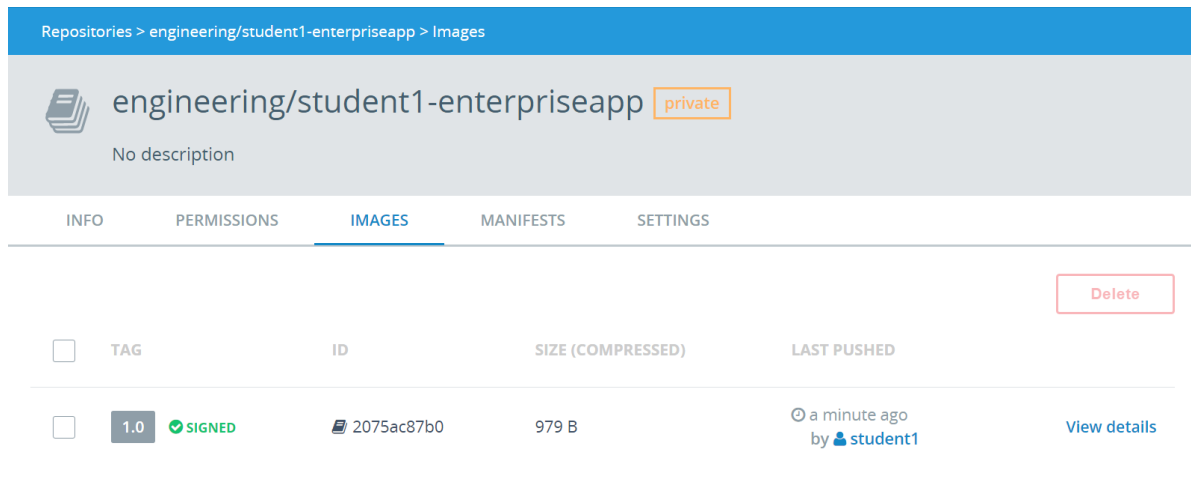
1. Pull the hello-world image into your ucp-controller node.
2. Tag the image with your DTR enterprise-app repo

```
$ docker image tag hello-world <DTR_FQDN>/engineering/<yourname>-enterprise-app:1.0
```

3. Enable content trust by running:

```
$ export DOCKER_CONTENT_TRUST=1
```

4. Login to DTR on the Docker CLI if you haven't already.
5. Push the <DTR_FQDN>/engineering/<yourname>-enterprise-app:1.0 image to DTR. You will be prompted for your repository key passphrase.
6. Go to the visit your <yourname>-enterprise-app repository in DTR via the web client, and click on the **Images** tab. You should be able to see the 1.0 image you pushed. It should be marked as signed.



17.5 Conclusion

Now that Notary is installed and content trust is activated, the Notary server integrated in DTR maintains signed metadata that describes the contents of a repository (targets), how old the content is (timestamp), and what combinations of those two are valid (snapshot). A client's Notary server will sync this metadata locally, and use it to defeat downgrade, mix-and-match, and other man-in-the-middle attacks on `docker pull`.

18 Delegate Image Signing

Now that basic trust has been established via Notary, we can delegate signing privileges to other DTR users, without having to directly share snapshot or target keys.

Pre-requisites:

- Instructor has DTR installed
- Instructor has created a DTR account for each student
- Instructor has added each student account to the **Engineering** organization as an Admin.
- Docker engine on **ucp-controller** configured to trust the instructor DTR instance
- Completed the Enabling Image Signing exercise

18.1 Rotate Snapshot key

The snapshot key can be managed by DTR's Notary server, eliminating the need to share this with users.

1. Rotate the repository **snapshot** key:

```
$ notary key rotate <DTR_FQDN>/engineering/<yourname>-enterprise-app \
  snapshot --server-managed
```

You'll be required to provide your DTR login credentials, as well as your root key password.

18.2 Delegate image signing

For each user that you want to delegate image signing to, you need to get their `cert.pem` file. This is the `cert.pem` file that comes with each user's client bundle. We'll delegate our image signing to the user **Chloe**.

1. Login to the your instructor's UCP web UI as **Chloe**. Make sure this is the instructor's UCP, and not your own!
2. Download the client bundle to your local machine (**Chloe** -> **Profile**, scroll to the bottom and click 'Create a Client Bundle').

3. Upload the zip archive to your **ucp-controller** node (note that you may need to include `-i keyfile.pem` after `scp` if your instances require it):

```
$ scp ucp-bundle-chloe.zip ubuntu@<ucp-manager-IP>:
```

4. Unzip the archive (you may need to do `sudo apt install unzip` to get the unzip util first). Run the following command to create a new Notary delegation role, using the user certificate:

```
$ notary delegation add -p <DTR_FQDN>/engineering/<yourname>-enterprise-app \
  targets/releases --all-paths cert.pem
```

When prompted for authentication details, specify your own DTR credentials.

18.3 Push a signed image

For this step, we will pretend that we are the user Chloe.

1. Open a new ssh session into your **ucp-controller** node
2. In the folder you unzipped Chloe's client bundle, run the command:

```
$ notary key import key.pem
```

This is needed because before delegated users can publish signed content with Notary or Docker Content Trust, they must import the private key associated with the user certificate. You will be promoted for a passphrase for the delegation key. Pick a passphrase and make sure to remember it.

3. Pull the `hello-world` image into the node

```
$ docker image pull hello-world
```

4. Re-tag the `hello-world` image with the `<yourname>-enterprise-app` repository and use a `2.0` tag.

```
$ docker image tag hello-world \
  <DTR_FQDN>/engineering/<yourname>-enterprise-app:2.0
```

5. Enable content trust

```
$ export DOCKER_CONTENT_TRUST=1
```

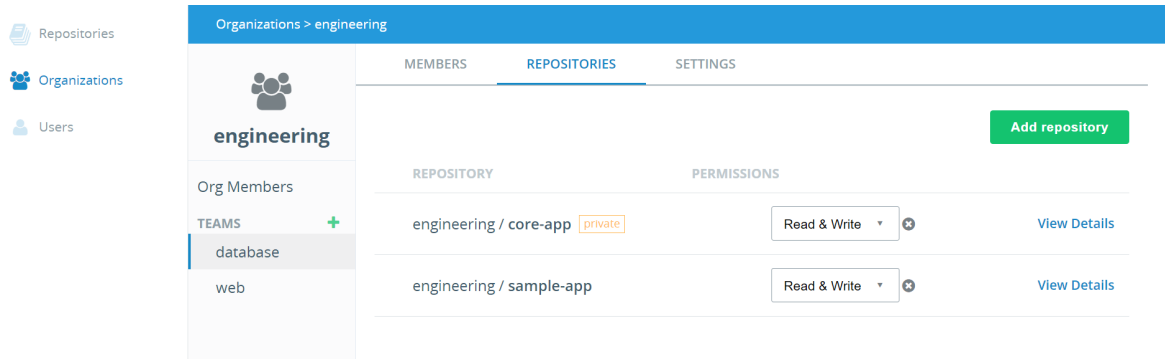
6. Login to DTR on the Docker CLI as Chloe.

```
$ docker login <DTR_FQDN>
Username: chloe
Password:
Login Succeeded
```

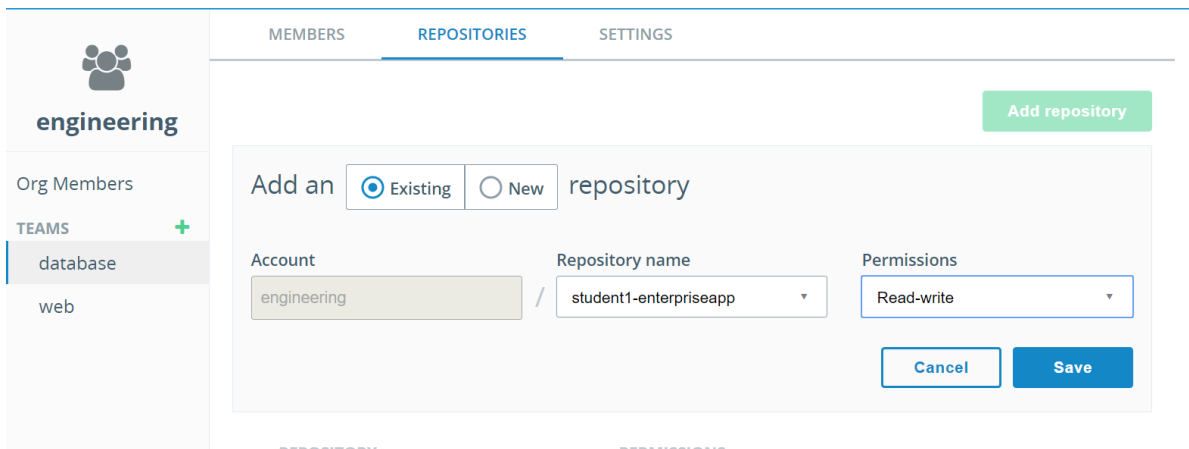
7. Push the `engineering/<yourname>-enterprise-app` repo to DTR.

What do you notice? You should see a **denied** error message. This is because Chloe does not have permissions to push to the repository.

8. Login to the DTR web UI using your account.
9. Navigate **Organizations** -> **Engineering**.
10. From here, click on your team and then click on the **Members** tab at the top. Add **chloe** as a team member (note that since a bunch of your fellow students are also on the same team, someone may beat you to this step - that's ok, as long as Chloe is on your team).
11. Now click on the **Repositories** tab at the top.



12. Click on **Add Repository**, select the <yourname>-enterprise-app repo and select **Read-Write** on the permissions. Then click **Save**.



We've now given your team, including Chloe, read-write access to <yourname>-enterprise-app.

13. Try and push the engineering/<yourname>-enterprise-app repo to DTR again, as Chloe. This time, you will be prompted for your delegation key passphrase and the push should succeed.

```
$ docker push <DTR_FQDN>/engineering/<yourname>-enterprise-app:2.0
The push refers to a repository [<DTR_FQDN>/engineering/<yourname>-enterprise-app]
98c944e98de8: Layer already exists
2.0: digest: sha256:2075ac87b043415d35bb6351b4a59df19b8ad154e578f7048335feeb02d0f759 size: 524
Signing and pushing trust metadata
Enter passphrase for delegation key with ID 0491cf9:
Successfully signed "<DTR_FQDN>/engineering/<yourname>-enterprise-app":2.
```

14. Go to the <yourname>-enterprise-app repository on your web browser and check that the 2.0 tag reports being signed.

18.4 Conclusion

In this exercise, you began with a content trust enabled repo that only you could push signed tags to. By allowing the appropriate keys to be managed by the notary server, you were able to delegate signing authority to another user on your team, without ever having to transmit your content trust keys to another human.

19 Image Scanning in DTR

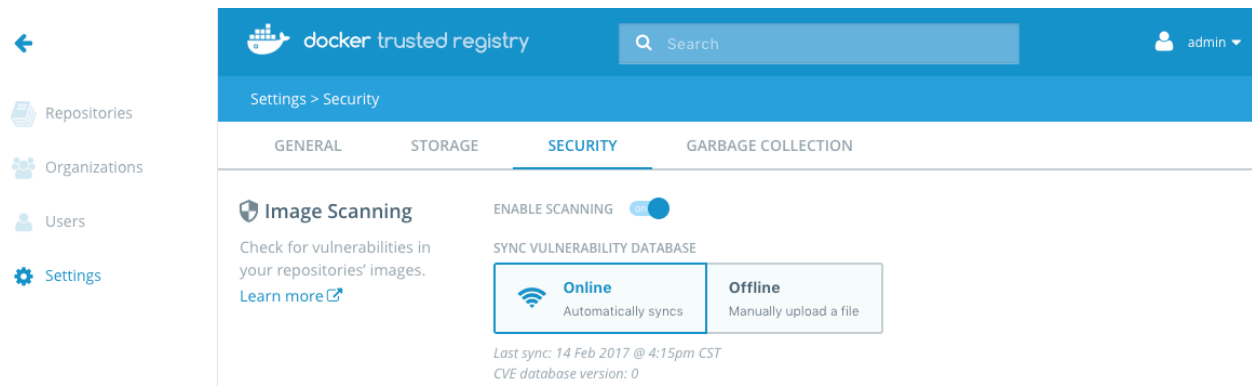
In this exercise, you'll walk through enabling image signing in DTR, and inspecting the results of a scan.

19.1 Pre-requisites:

- DTR installed and integrated with UCP
- All Docker engines configured to trust DTR certificate.
- Must have completed all user management exercises in order to have the necessary user accounts set up in DTR

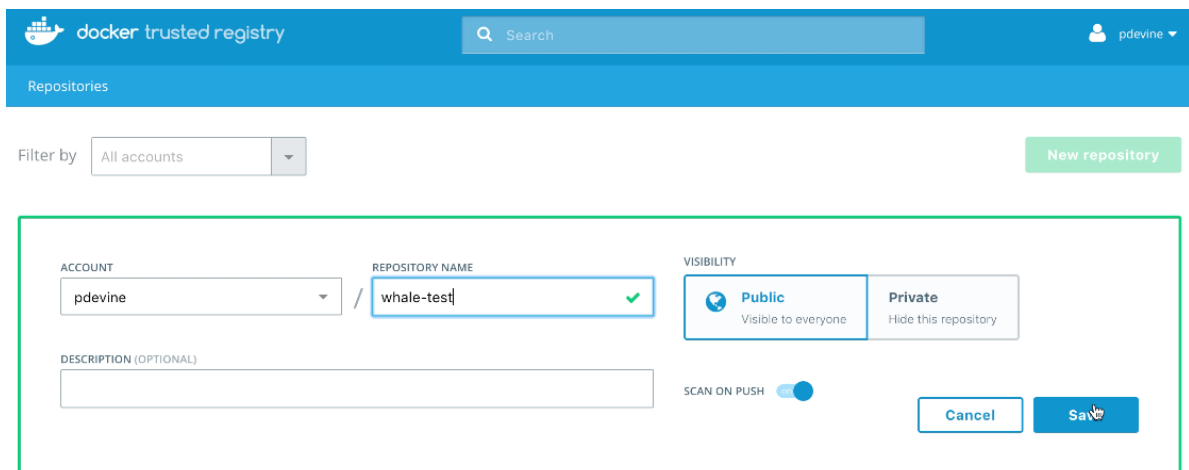
19.2 Enable Image Scanning

First, a DTR admin needs to enable Image Scanning - your instructor can do this.



19.3 Create a repo

1. Log in to the Docker Trusted Registry web UI with your user account.
2. Create a new user-owned repository called `whale-test`. Set the visibility to “public” and enable ‘SCAN ON PUSH’ radio button.



3. Pull the `ubuntu:16.04` image or any image that you would like to perform security image scanning on:

```
$ docker pull ubuntu:16.04
```

4. Re-tag the image with your DTR repository name and a tag of 1.0.

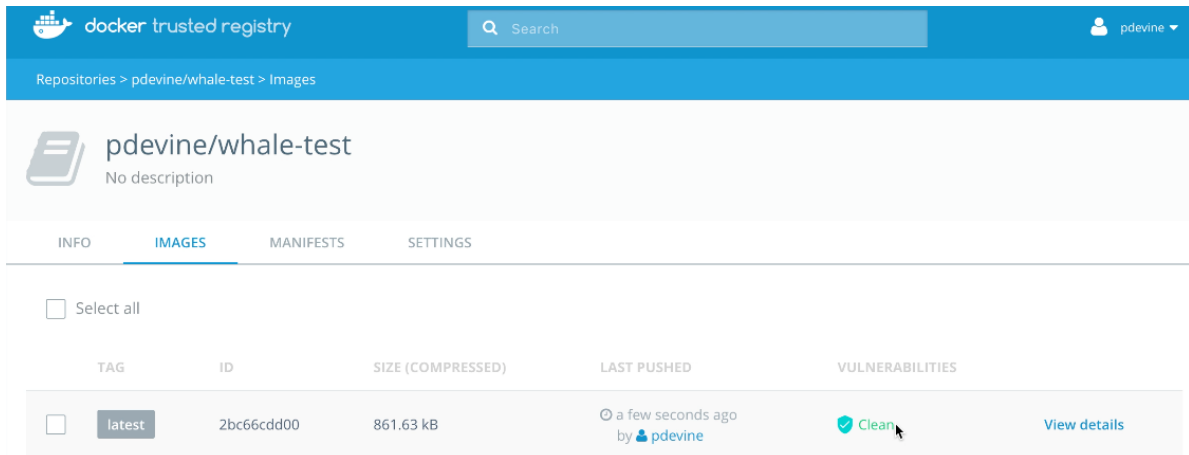
```
$ docker image tag ubuntu:16.04 \
<DTR public DNS>/yourUsername/whale-test:1.0
```

5. Log in and push the image to DTR:

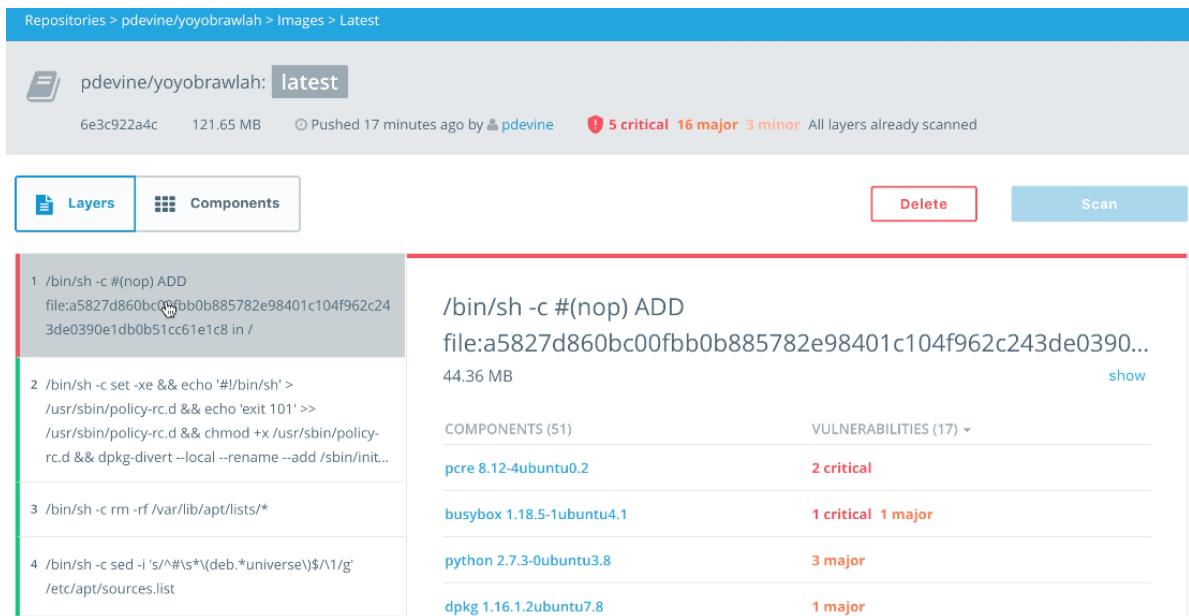
```
$ docker login <DTR public DNS>
$ docker push <DTR public DNS>/yourUsername/whale-test:1.0
```

19.4 Investigate layers & components

1. Go to the repository on the DTR web UI and verify that you now have a 1.0 tag on the repository:



2. Click 'View Details' link of your image, it will take you to 'Layers' view. Dockerfile entries are listed on the left; clicking on them will display the report for that layer.



3. Click on 'Components'; the same vulnerabilities are listed by component instead of layer, with links to the relevant CVE reports.

Repositories > pdevine/yoyobrawlah > Images > Latest

pdevine/yoyobrawlah: latest

6e3c922a4c 121.65 MB Pushed 18 minutes ago by pdevine 5 critical 16 major 3 minor All layers already scanned

Layers Components Delete Scan

pcr
8.12-4ubuntu0.2
2 critical

freetype
2.4.8-1ubuntu2.3
1 critical 1 major

busybox
1.18.5-1ubuntu4.1
1 critical 1 major

file
5.09-2ubuntu0.6
1 critical 1 major

busybox

VERSION 1.18.5-1ubuntu4.1 LICENSE GPL COPYLEFT

VULNERABILITIES	SEVERITY	DESCRIPTION
CVE-2013-1813	critical	util-linux/mdev.c in BusyBox before 1.21.0 uses 0777 permissions for parent directories when creating nested directories under /dev/, which...
CVE-2011-2716	major	The DHCP client (udhcp) in BusyBox before 1.20.0 allows remote DHCP servers to execute arbitrary commands via shell metacharacters in the...

FILEPATH
/usr/lib/initramfs-tools/bin/busybox

19.5 Conclusion

At this point, layers pushed to the repo you created in this exercise will be scanned when they are received by DTR, and when the local CVE database is updated. These scans in turn can be set to trigger webhooks on scan completion or vulnerability detection, allowing users to integrate image scanning into CI/CD pipelines.

20 DTR Webhooks

In order to integrate seamlessly with a CI/CD pipeline, Docker Trusted Registry provides a system of webhooks. In this exercise, you'll walk through setting up and triggering a webhook that fires when an image is pushed to a repository.

20.1 Setting up a Webhook

1. Create a service that will catch and display the webhook's POST payload (in reality, this would be the part of your CI/CD chain that needs to take action when the webhook fires):

```
$ docker service create --name whale-webhook -p 8000:8000 training/whale-server
```

2. Create a DTR repo to attach your webhook to. Make it a user-owned repo named username/whale-test.
3. Navigate to API Docs through the top-right dropdown menu on DTR's dashboard. Find the row for POST api/v0/webhooks (near the bottom), and add:

```
1 {
2   "type": "TAG_PUSH",
3   "key": "username/whale-test",
4   "endpoint": "http://<public DNS of a UCP node>:8000"
5 }
```

4. Click 'Try it Out!' to register the webhook.
5. Tag and push training/whale-server to your new DTR repo username/whale-test; once the push finishes, check the container logs of your running whale-server container to see a record of the webhook POST.
6. Click 'Try it Out!' on the GET method of api/v0/webhooks to see a list of all webhooks you've registered in DTR.

7. Finally, to delete a webhook, copy the `id` field from the JSON returned by the GET method, and paste it into the DELETE method for `api/v0/webhooks`; clicking 'Try it Out!' will delete the corresponding webhook.

20.2 Conclusion

Webhooks are available for most Registry events, like repository creation and deletion, pushes, and security scanning events. In this demo, you created a toy service that caught your webhook, but in reality this could be any part of a CI/CD chain or other service integrated with DTR activity.