DockerCon 2017 Austin - Exercises

Contents

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/?overlay=subscriptions.

1.1 Install ntp

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.

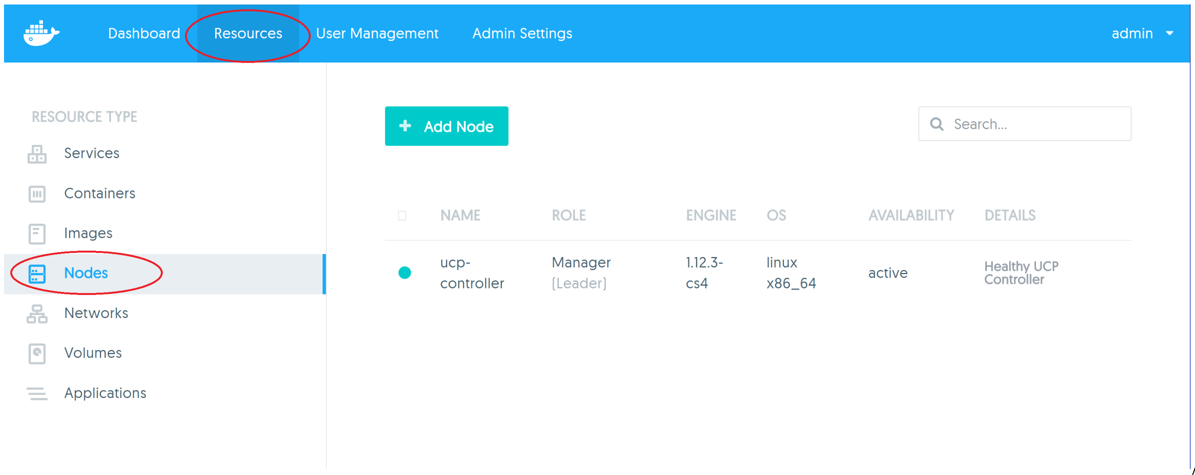
3. Once logged in you will see a prompt screen asking you to upload a license. Click the **Upload License** button and upload your UCP license; if you need a trial license, you can download one from Docker Store: https://store.docker.com/?overlay=subscriptions.

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.

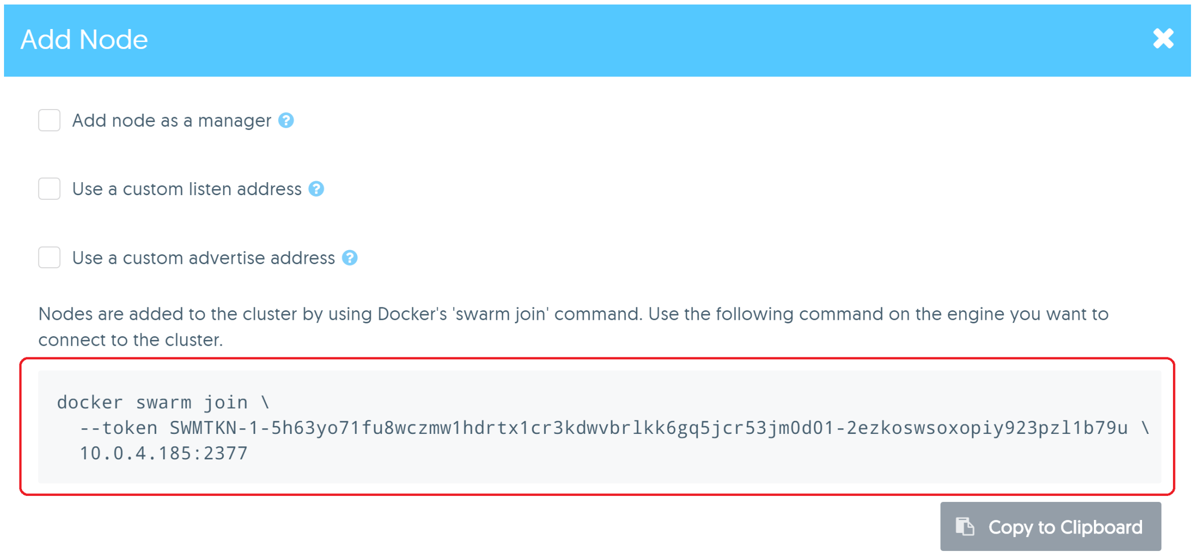
1.4 Add additional worker nodes

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`.



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

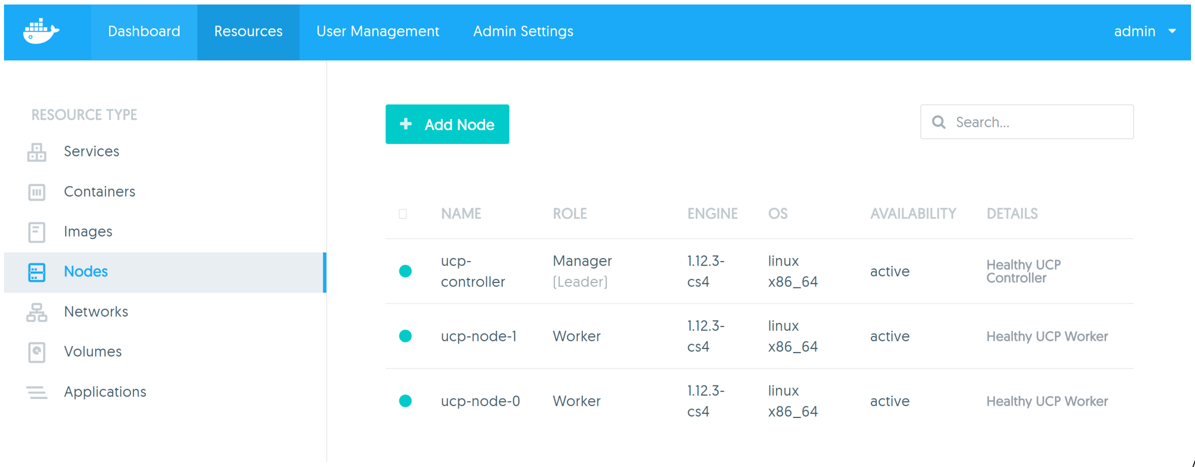
2. Click **Add Node**; a join command is presented.



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-1-5h63yo71fu8wczmw1hdrtx1cr3kdwvbrlkk6gq5jcr53jm0d01-2ezkoswsoxopiy923pzl1b79u \
  10.0.4.185:2377
```

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.



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

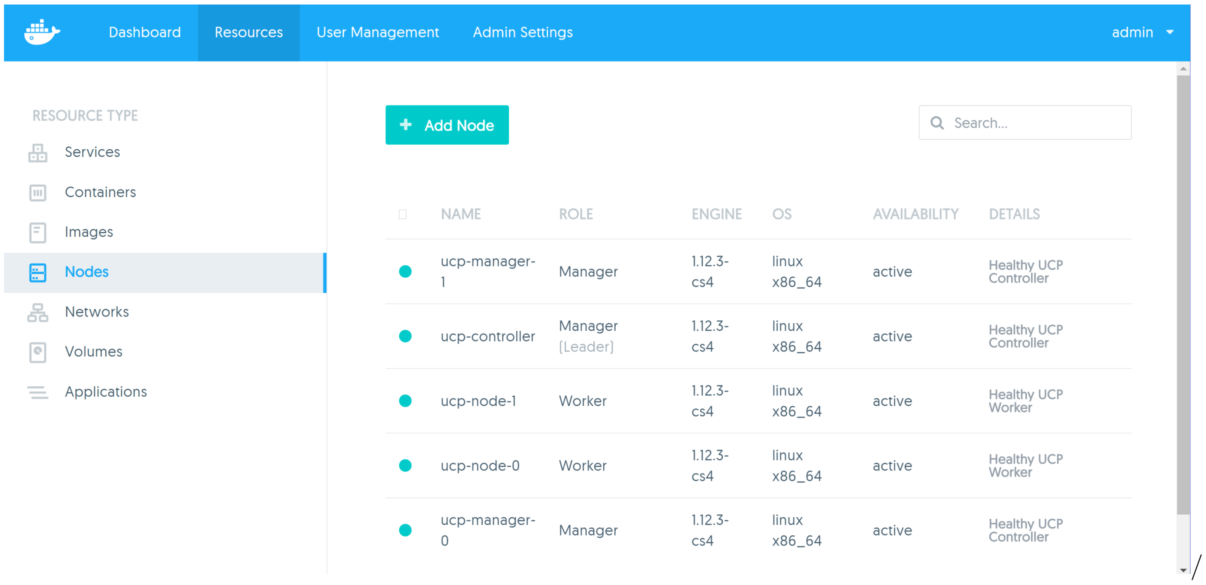
Like we did for our other UCP nodes, install ntp first:

```
$ 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*.

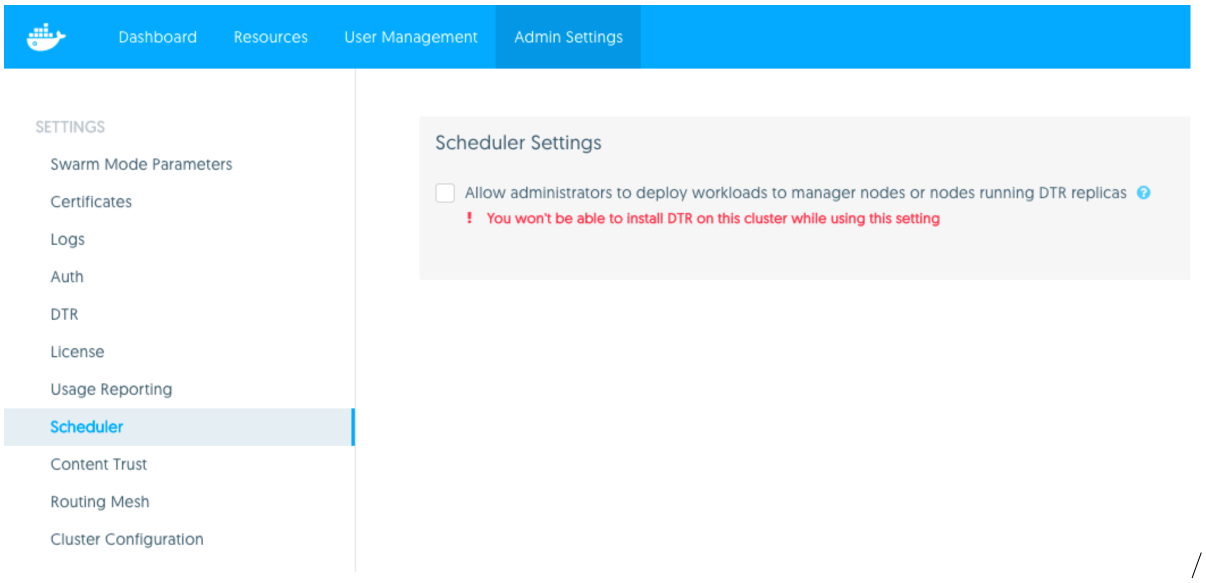
2.3 Test manager nodes

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:



Scheduler Settings

☐ Allow administrators to deploy workloads to manager nodes or nodes running DTR replicas

! You won't be able to install DTR on this cluster while using this setting

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 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:

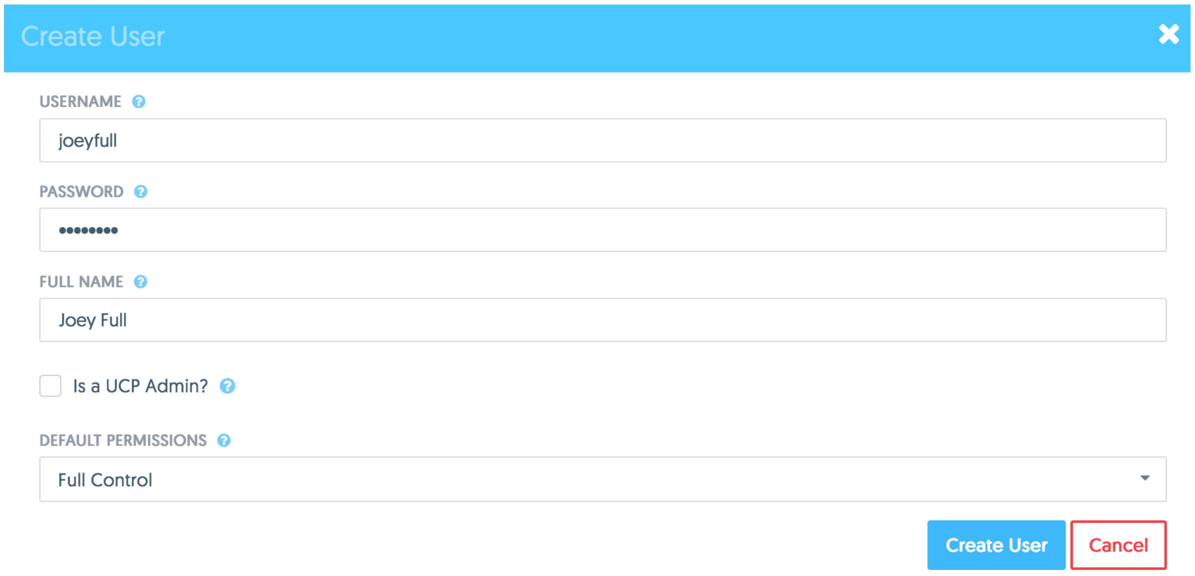
- A working UCP installation.

3.1 Create new users

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

Username	Full Name	Default Permissions
joeyfull	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.



USERNAME: joeyfull

PASSWORD: ••••••••

FULL NAME: Joey Full

☐ Is a UCP Admin?

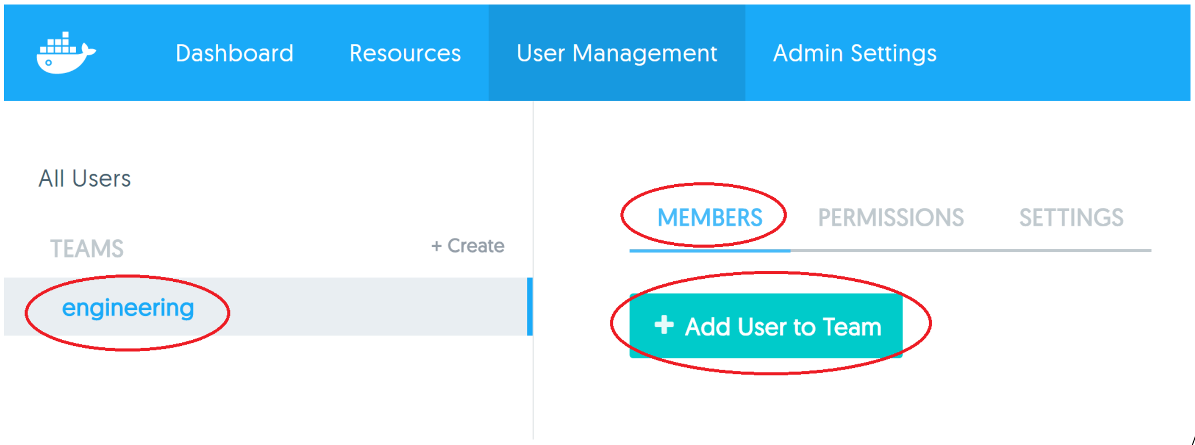
DEFAULT PERMISSIONS: Full Control

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

3.2 Create a team and add users

Users can be grouped into teams for simpler management.

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.

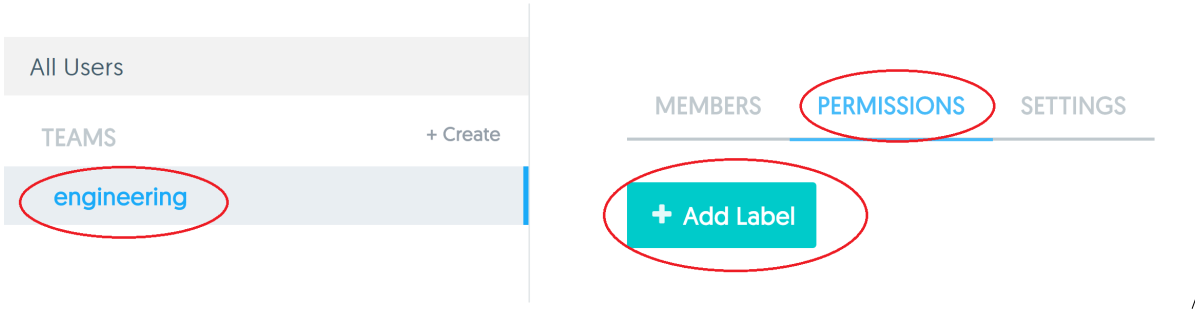


4. Add all four new users to the team by clicking the **Add to Team** button next to each of them and then click **Done**. Do not add yourself (usually "admin") to the team.

3.3 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**.

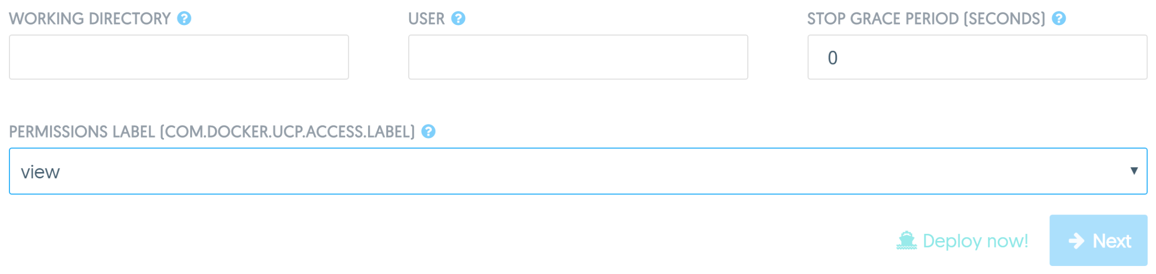
LABEL	PERMISSION
view	View Only
restricted	Restricted Control
run	Full Control

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

3.4 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!**:



WORKING DIRECTORY

USER

STOP GRACE PERIOD (SECONDS): 0

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

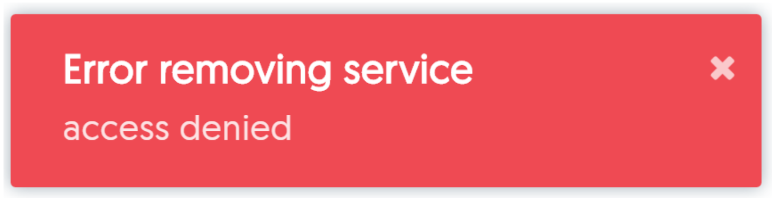
Repeat this step to deploy another service called `nginx-unlabeled`, but this time **do not use any label**.

3.5 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.

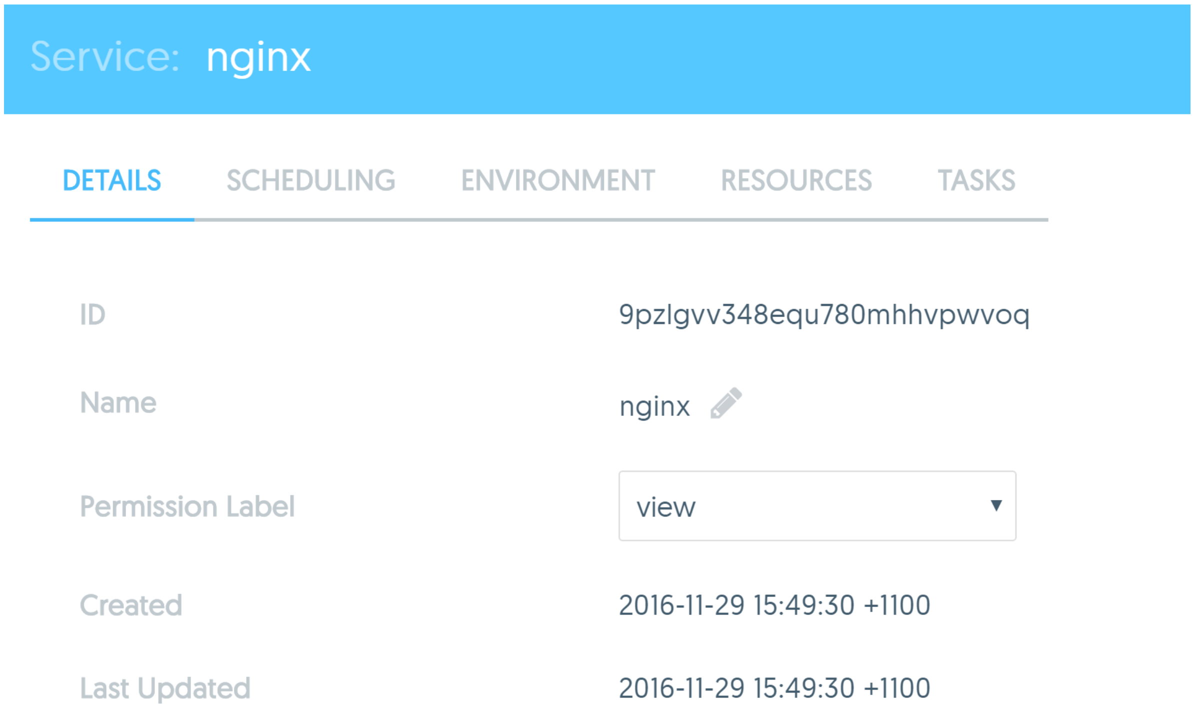
   You should get the following error message on the bottom of the screen:



Error removing service

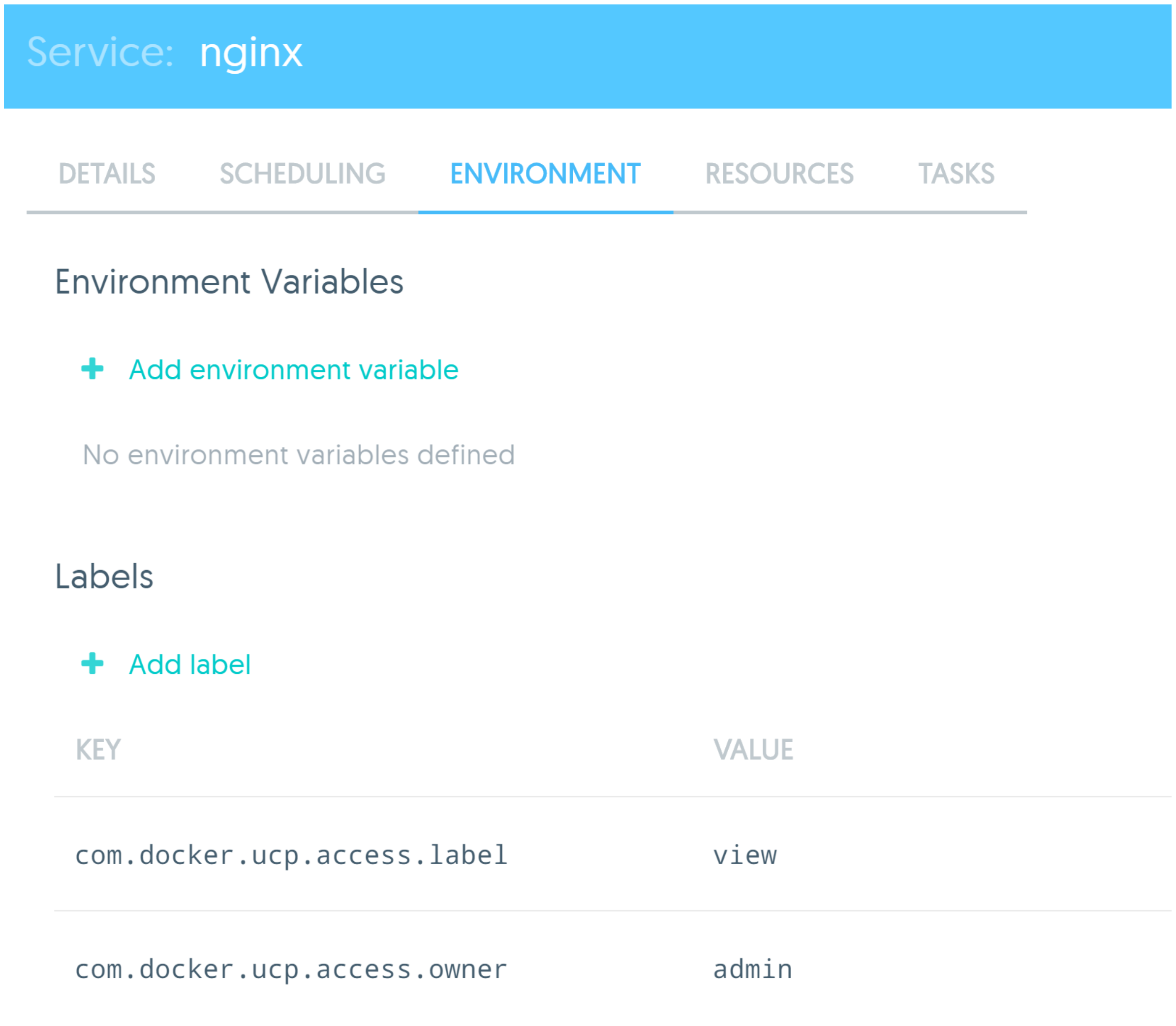
access denied

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


Service: nginx

ID	9pzlgvv348equ780mhhvpwvoq
Name	nginx
Permission Label	view
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

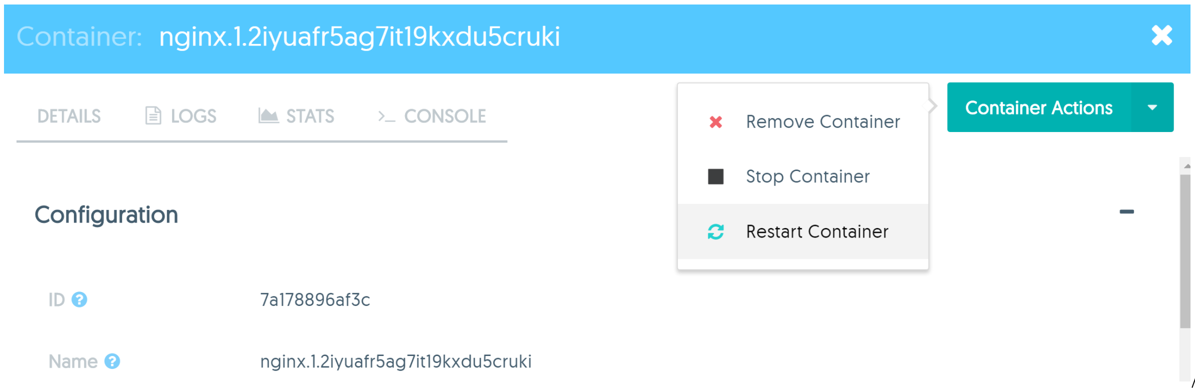
No environment variables defined

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

Configuration

ID	7a178896af3c
Name	nginx.1.2iyuafr5ag7it19kxdu5cruki

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:

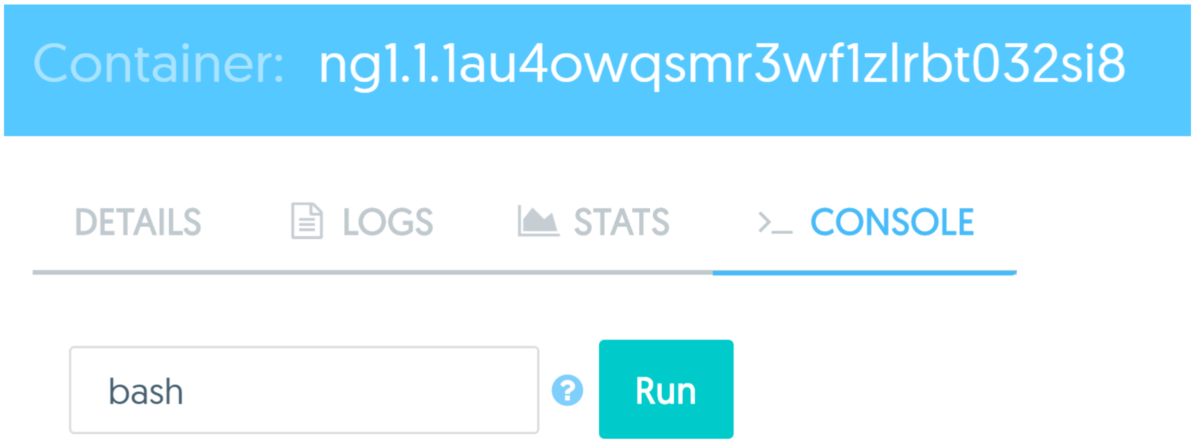
Image Name	Service Name	Permissions Label
nginx	ng1	restricted
nginx	ng2	run

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.

3.6 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 most privileged access to this container is via the Engineering team's *Restricted Control* privilege for the **restricted** label, which you applied to the `ng1` service. *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?

3.7 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
   ```

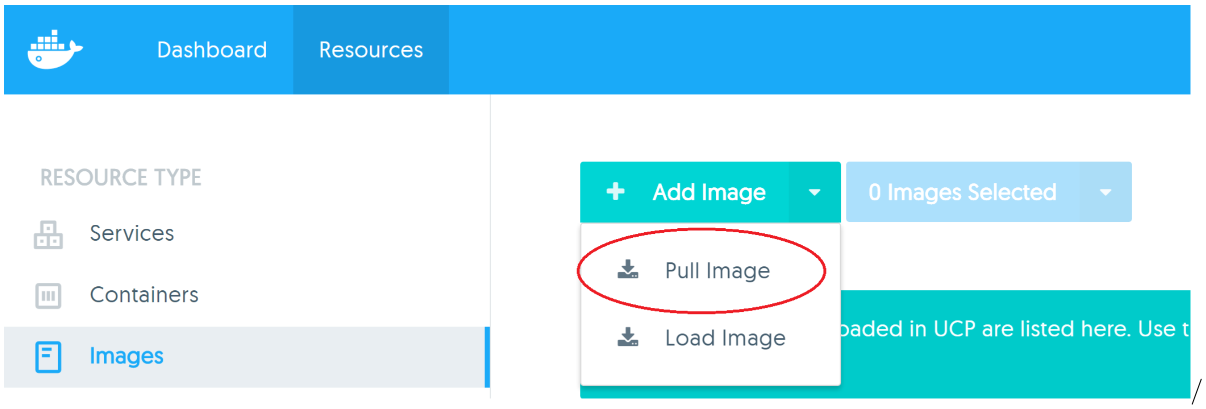
   to open a `bash` terminal on the `ng1.1` container. Does the command succeed? Why or why not?
7. Repeat the previous step for the **ng2.1** container. What happens and why?

3.8 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?

3.9 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".

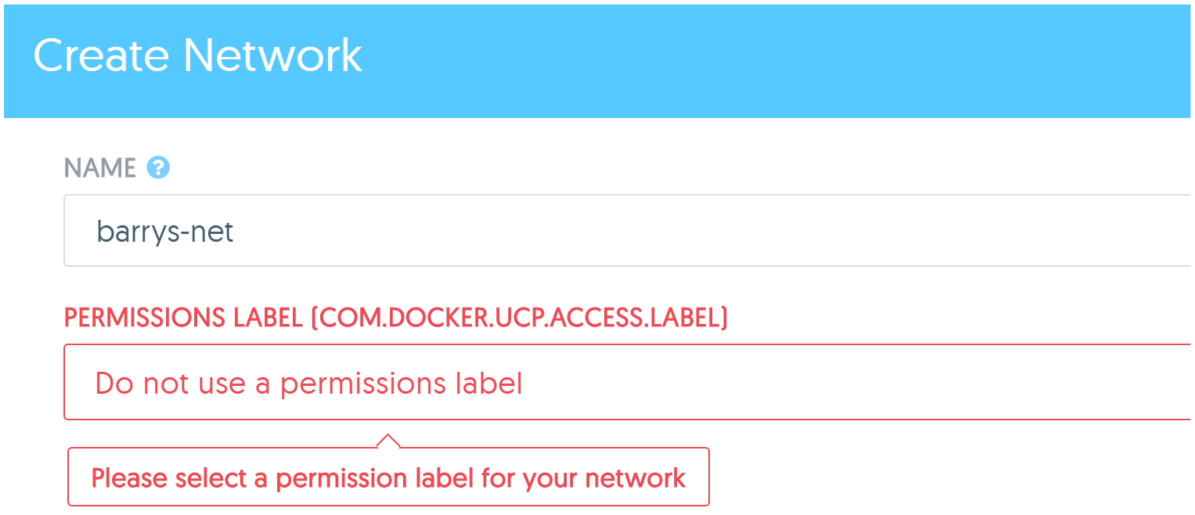
   So far, we can see that the user **joeyfull** has full access to create networks and pull images. This is because **joeyfull** has the *Full Access* default permission, giving them full access to create and manipulate new objects in UCP; note this does not confer the ability to delete objects, or see unlabeled containers 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. Log out of UCP as **kellyres** and log back in as **barryview**.
8. 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.

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

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



Create Network

NAME

barrys-net

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

Do not use a permissions label

Please select a permission label for your network

/

10. 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.
11. Logout of UCP as **barryview** and login as **traceyno**.
12. 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.
13. Click the **Services** link and notice that Tracey can see the three services `nginx-labeled`, `ng1` and `ng2`
14. Create a network using either the **restricted** or **run** label.

3.10 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 must have at least View Only permission, either as their default or as granted by a label in one of their teams, in order to see an object. 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).

4 Demo: Install DTR

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

Pre-requisites:

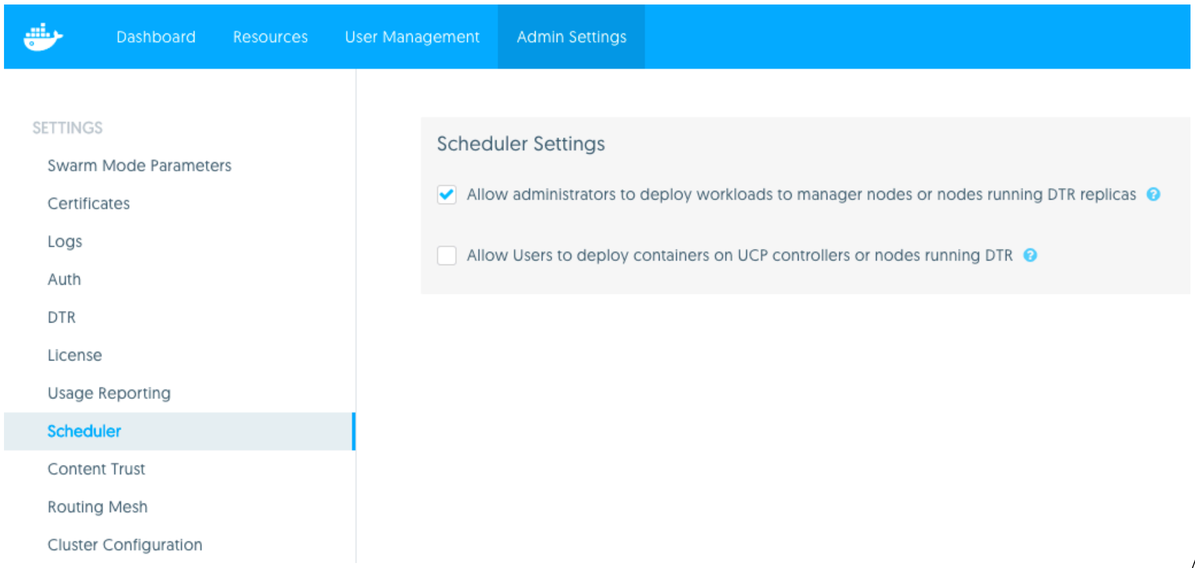
- UCP installed with 2 worker nodes
- Three additional nodes with the following names:
  - `dtr-replica-0`
  - `dtr-replica-1`
  - `dtr-replica-2`

4.1 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.

4.2 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**



Scheduler Settings

✓ Allow administrators to deploy workloads to manager nodes or nodes running DTR replicas

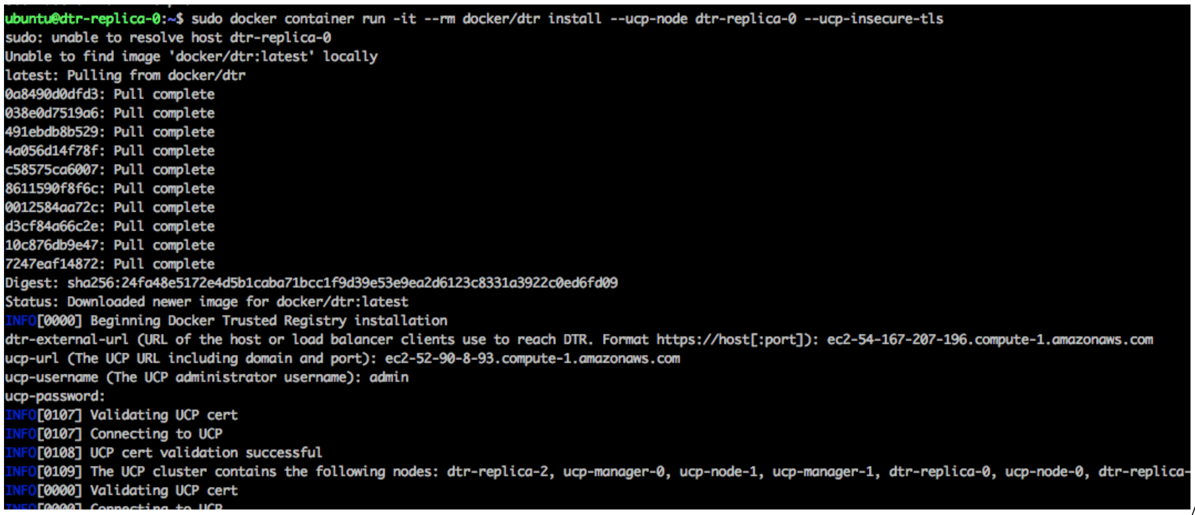
☐ Allow Users to deploy containers on UCP controllers or nodes running DTR

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
```

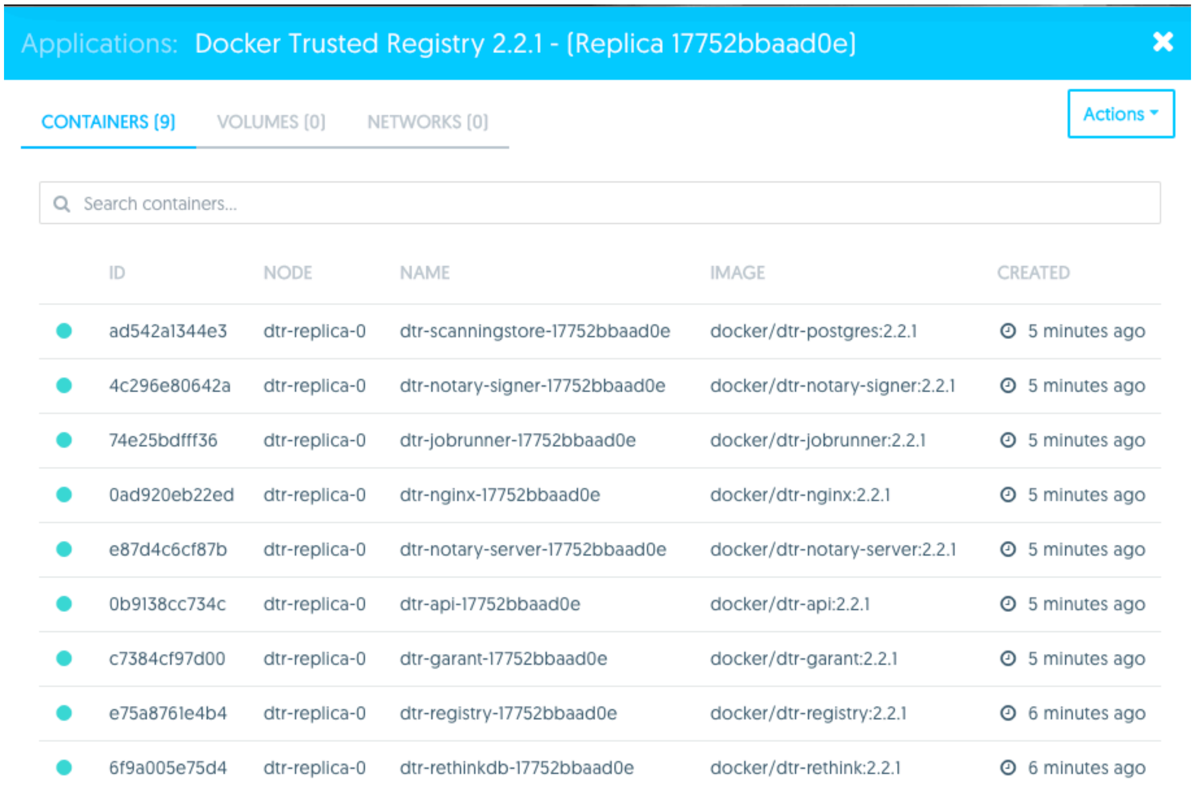
4. The first prompt you will get is for the **dtr-external-url**. This is the URL that users will use to access DTR. You must specify either the public IP or domain of the node you are installing DTR on.
5. Specify the URL to UCP along with the UCP admin username and password when prompted. The UCP URL 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-
INFO[0000] Validating UCP cert
INFO[0000] Connecting to UCP
```

4.3 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.



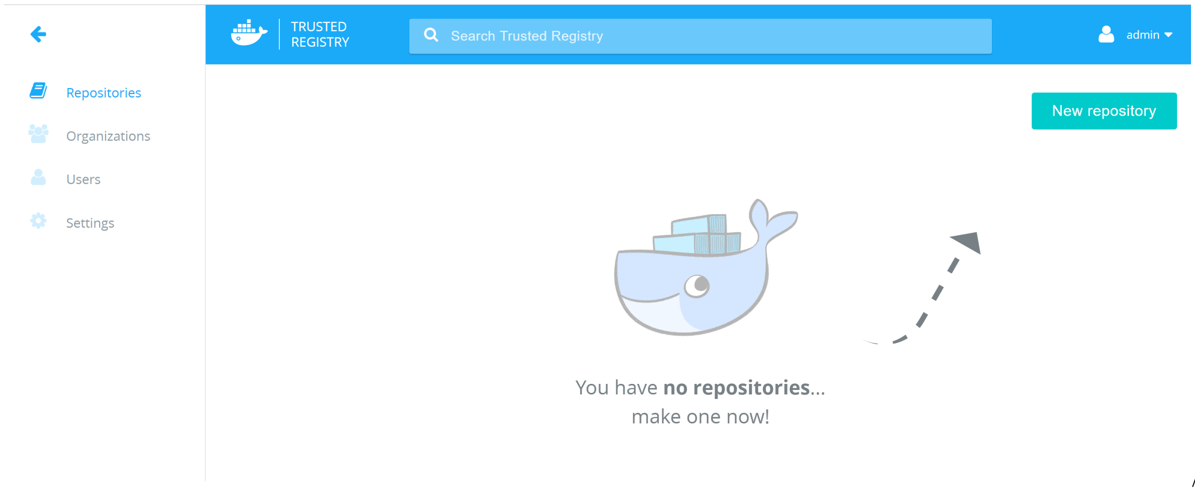
Applications: Docker Trusted Registry 2.2.1 - (Replica 17752bbaad0e)

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.



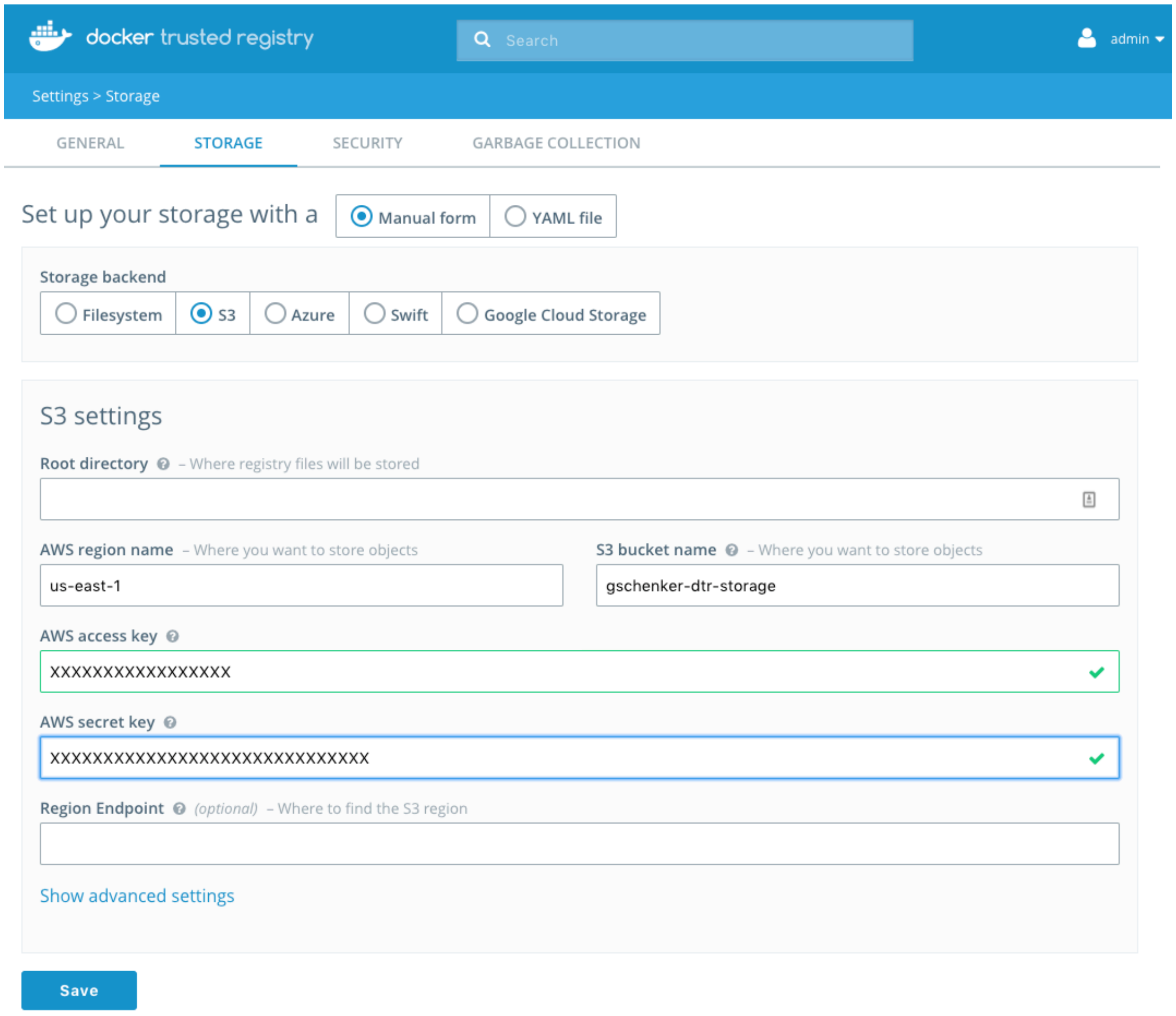
You have **no repositories**... make one now!

4.4 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.



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: XXXXXXXXXXXXXXXXX

AWS secret key: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

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

Show advanced settings

5. Click **Save**

4.5 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.

1. Switch into your `ucp-controller` terminal.
2. 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-domain-name/ca \
-o /usr/local/share/ca-certificates/dtr-domain-name.crt
```

3. Refresh the list of certificates to trust:

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

4. Restart the Docker engine:

```
$ sudo service docker restart
```

5. Log in to DTR from the command line:

```
$ docker login <DTR URL>
```

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

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

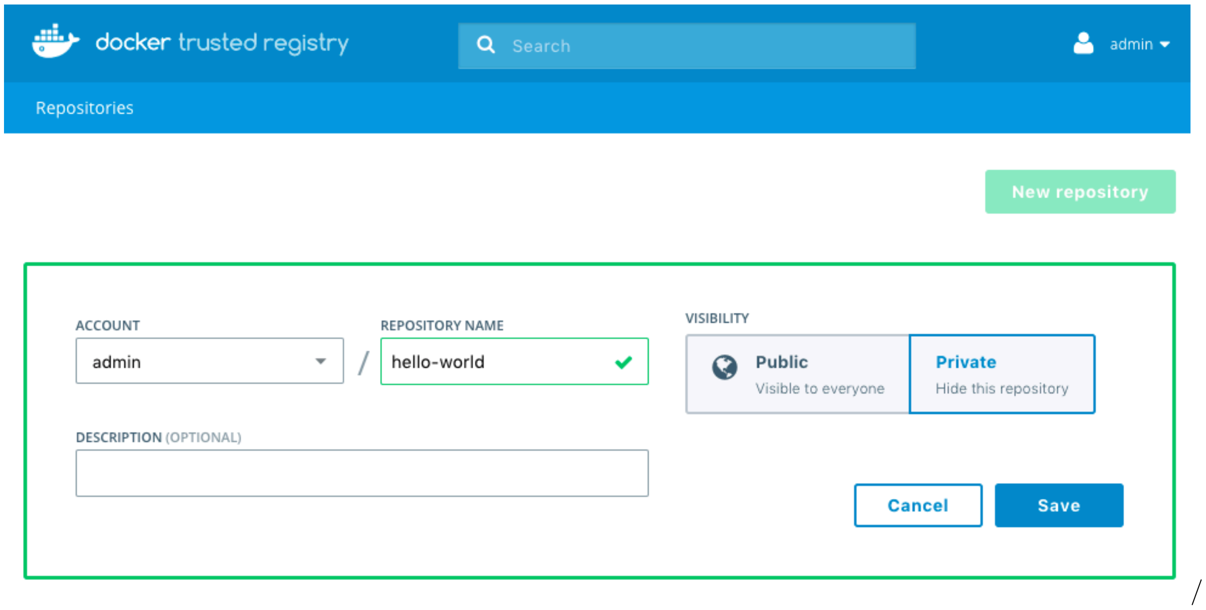
   Note: To speed up the whole process of configuring all nodes of our swarm we can use a bash script like the following:

```
CERT=./<your-pem-name>.pem
DTR_DOMAIN_NAME=<DTR-DOMAIN-Name>
NODE_LIST=<LIST-IP-ADDRESSES-OF-NODES-IN-SWARM>
# e.g. ("52.23.162.247" "52.3.234.184" "52.205.102.106" ...)
for NODE in "${NODE_LIST[@]}"; do
  ssh -i $CERT ubuntu@$NODE hostname
   ssh -i $CERT ubuntu@$NODE sudo curl -k https://$DTR_DOMAIN_NAME/ca \
  -o /usr/local/share/ca-certificates/dtr-domain-name.crt
  ssh -i $CERT ubuntu@$NODE sudo update-ca-certificates
  ssh -i $CERT ubuntu@$NODE sudo service docker restart
done;
```

   Replace the placeholders with the respective values of your infrastructure.

4.6 Test the integration

1. 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:



ACCOUNT: admin / REPOSITORY NAME: hello-world
VISIBILITY: Private
DESCRIPTION (OPTIONAL):

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

```
$ docker tag hello-world:latest \
ec2-54-244-191-106.us-west-2.compute.amazonaws.com/admin/hello-world:1.0
```

4. Push to DTR:

```
$ docker push <public DTR DNS>/admin/hello-world:1.0
```

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

4.7 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.

5 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:

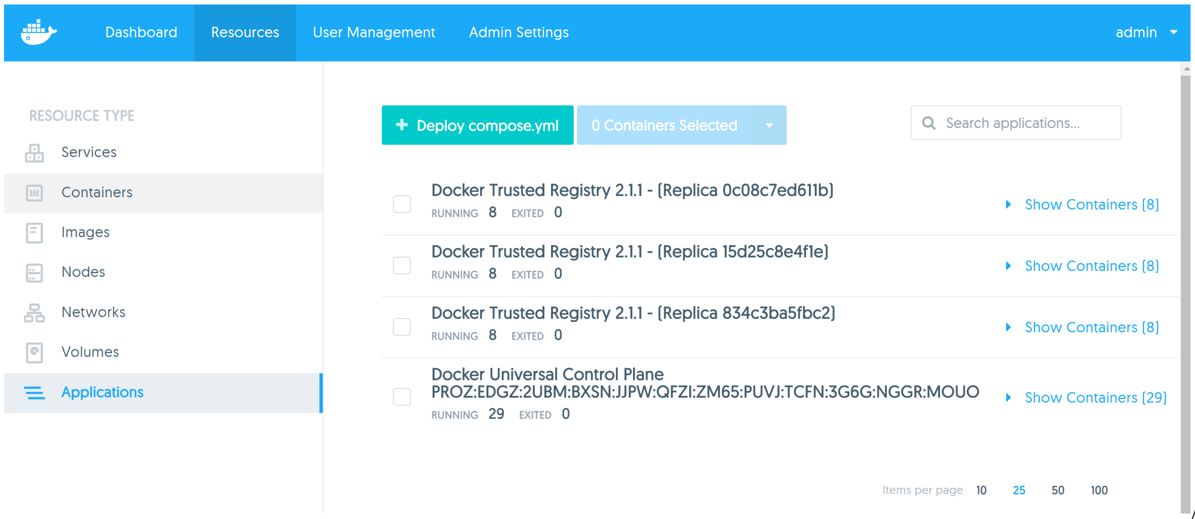
- UCP installed with 2 worker nodes
- DTR installed and integrated with UCP
- The following nodes joined to UCP as workers:
  - `dtr-replica-1`
  - `dtr-replica-2`

5.1 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.



Docker Trusted Registry 2.1.1 - [Replica 0c08c7ed611b]
RUNNING 8 EXITED 0 Show Containers (8)

Docker Trusted Registry 2.1.1 - [Replica 15d25c8e4f1e]
RUNNING 8 EXITED 0 Show Containers (8)

Docker Trusted Registry 2.1.1 - [Replica 834c3ba5fbc2]
RUNNING 8 EXITED 0 Show Containers (8)

Docker Universal Control Plane
PROZ:EDGZ:2UBM:BXSN:JJPW:QFZI:ZM65:PUVJ:TCFN:3G6G:NGGR:MOUO
RUNNING 29 EXITED 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`.

5.2 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.

5.3 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.

6 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.

6.1 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 instructions in the next section.

6.2 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.

1. Switch into your `ucp-controller` terminal.
2. Download the DTR certificate by running the following command (replace `<dtr-domain-name>` with the domain of your instrucor's first DTR instance):

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

3. Refresh the list of certificates to trust:

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

4. Restart the Docker engine:

   ```
   $ sudo service docker restart
   ```

5. Log in to DTR from the command line:

   ```
   $ docker login <DTR URL>
   ```

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

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

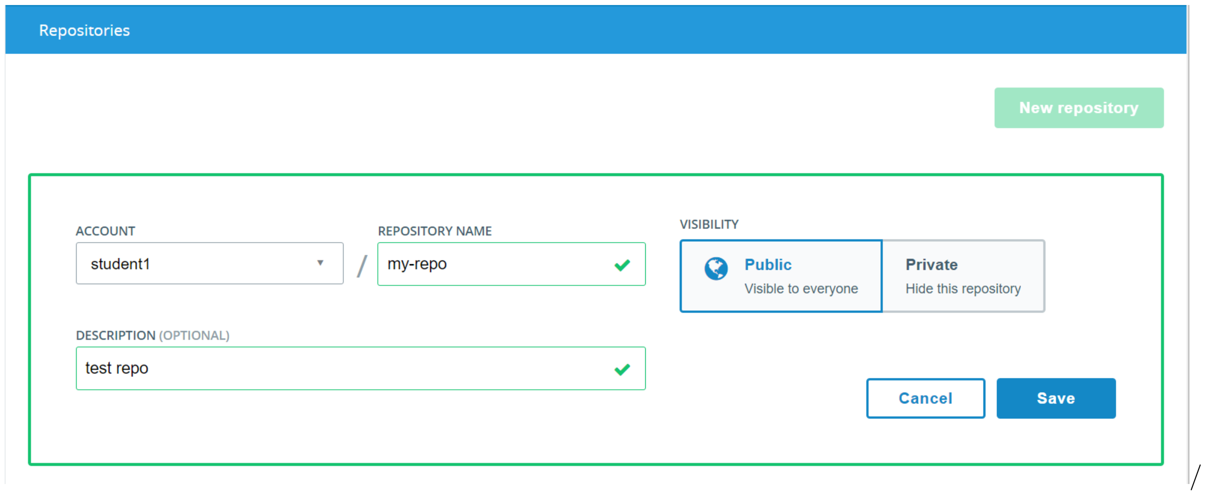
   Note: To speed up the whole process of configuring all nodes of our swarm we can use a bash script like the following:

```
CERT=./<your-pem-name>.pem
DTR_DOMAIN_NAME=<DTR-DOMAIN-Name>
NODE_LIST=<LIST-IP-ADDRESSES-OF-NODES-IN-SWARM>
# e.g. ("52.23.162.247" "52.3.234.184" "52.205.102.106" ...)
for NODE in "${NODE_LIST[@]}"; do
  ssh -i $CERT ubuntu@$NODE hostname
   ssh -i $CERT ubuntu@$NODE sudo curl -k https://$DTR_DOMAIN_NAME/ca \
  -o /usr/local/share/ca-certificates/dtr-domain-name.crt
  ssh -i $CERT ubuntu@$NODE sudo update-ca-certificates
  ssh -i $CERT ubuntu@$NODE sudo service docker restart
done;
```

Replace the placeholders with the respective values of your infrastructure.

6.3 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".



ACCOUNT: student1 / REPOSITORY NAME: my-repo ✓

VISIBILITY: [✓ Public — Visible to everyone] Private — Hide this repository

DESCRIPTION (OPTIONAL): test repo ✓

6.4 Push an image into the repository

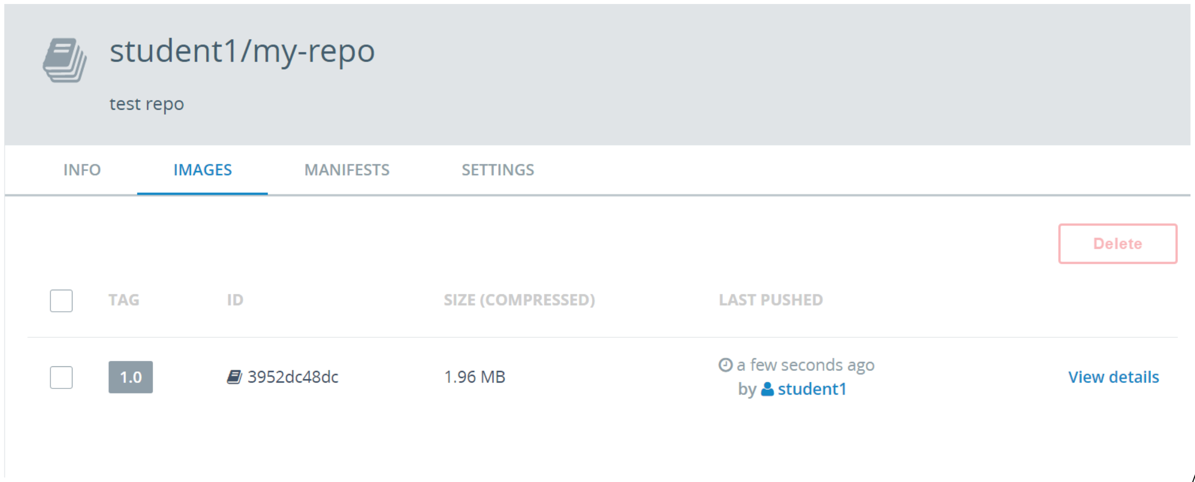
1. From your `ucp-controller` node, pull the `alpine` image.
2. Re-tag `alpine` with your DTR repository name and a tag of 1.0:

```
$ 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.



TAG	ID	SIZE (COMPRESSED)	LAST PUSHED
1.0	3952dc48dc	1.96 MB	a few seconds ago by student1

6.5 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.

6.6 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.

7 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

7.1 Demo: Creating Orgazinations 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`.

7.2 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?

7.3 Conlcusion

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.

8 Image Scanning in DTR

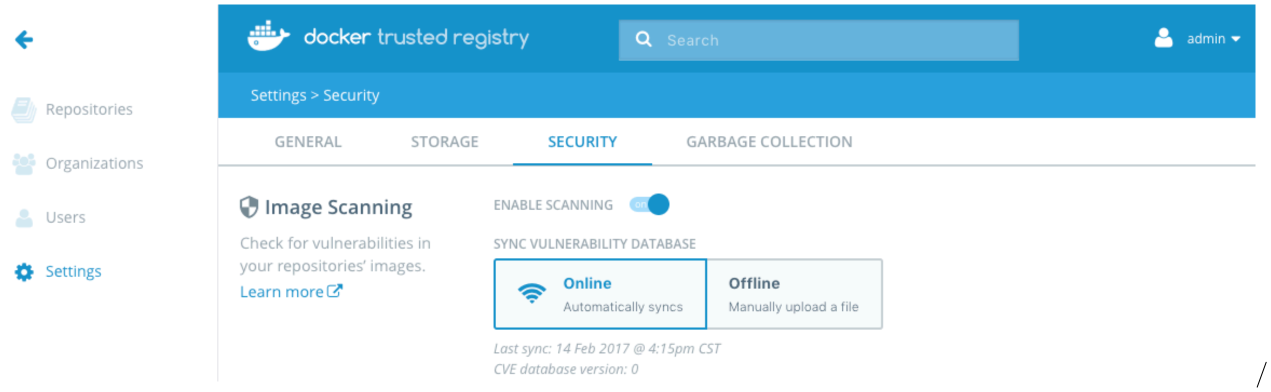
In this exercise, you'll walk through enabling image signing in DTR, and inspecting the results of a scan.

8.1 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

8.2 Enable Image Scanning

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



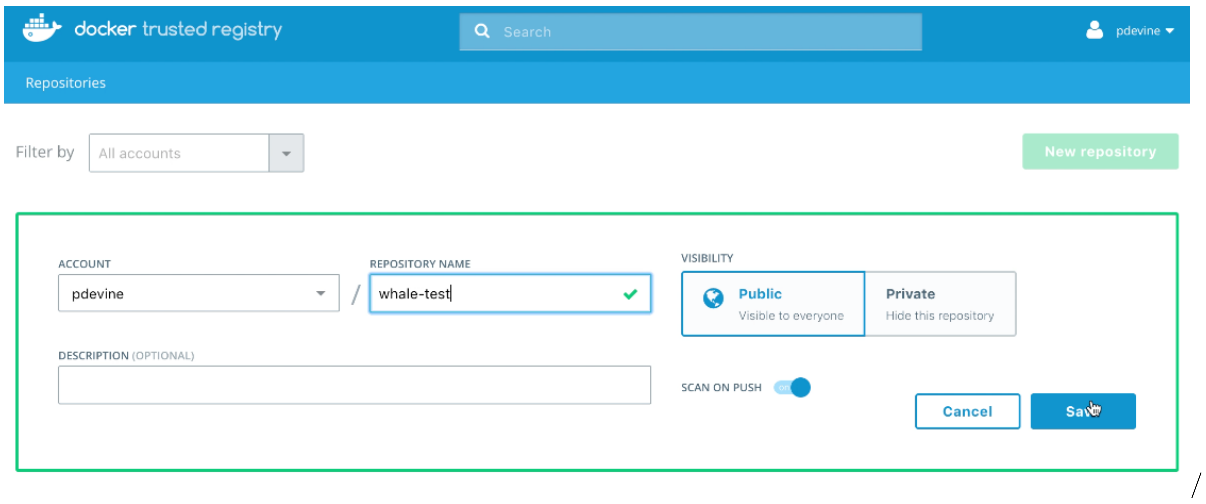
ENABLE SCANNING: on

SYNC VULNERABILITY DATABASE: Online

Last sync: 14 Feb 2017 @ 4:15pm CST
CVE database version: 0

8.3 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.



ACCOUNT: pdevine / REPOSITORY NAME: whale-test

DESCRIPTION (OPTIONAL)

VISIBILITY: Public

SCAN ON PUSH: on

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.

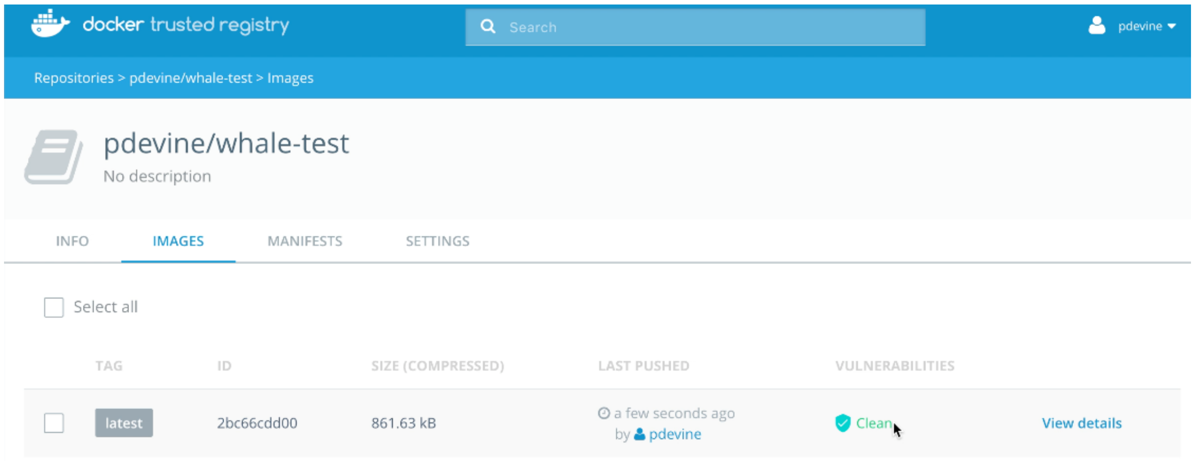
```
$ docker image tag ubuntu:16.04 \
<DTR public DNS>/yourUsername/whale-test:1.0
```

5. Log in and push the image to DTR:

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

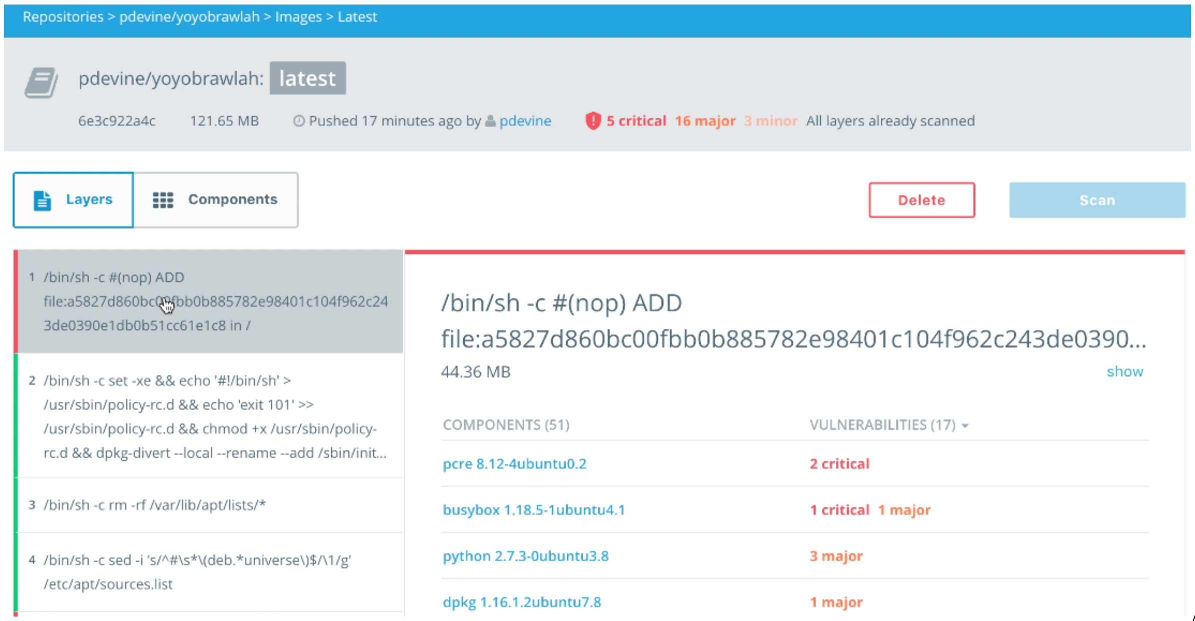
8.4 Investigate layers & components

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



TAG	ID	SIZE (COMPRESSED)	LAST PUSHED	VULNERABILITIES	
latest	2bc66cdd00	861.63 kB	a few seconds ago by pdevine	Clean	View details

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.


Repositories > pdevine/yoyobrawlah > Images > Latest
pdevine/yoyobrawlah: latest
Layers
Components
Delete
Scan
show

6e3c922a4c 121.65 MB Pushed 17 minutes ago by pdevine 5 critical 16 major 3 minor All layers already scanned

1 /bin/sh -c #(nop) ADD file:a5827d860bc00fbb0b885782e98401c104f962c243de0390e1db0b51cc61e1c8 in /

2 /bin/sh -c set -xe && echo '#!/bin/sh' > /usr/sbin/policy-rc.d && echo 'exit 101' >> /usr/sbin/policy-rc.d && chmod +x /usr/sbin/policy-rc.d && dpkg-divert --local --rename --add /sbin/init...

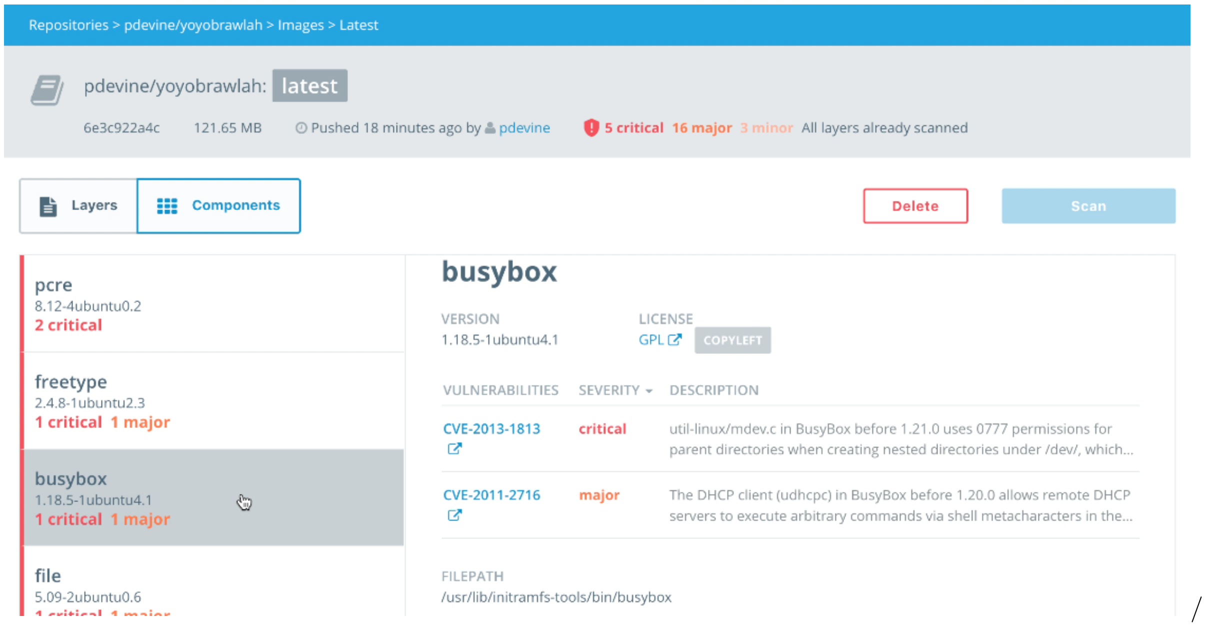
3 /bin/sh -c rm -rf /var/lib/apt/lists/*

4 /bin/sh -c sed -i 's/^#\s*\(deb.*universe\)$/\1/g' /etc/apt/sources.list

/bin/sh -c #(nop) ADD file:a5827d860bc00fbb0b885782e98401c104f962c243de0390...
44.36 MB

COMPONENTS (51)	VULNERABILITIES (17)
pcre 8.12-4ubuntu0.2	2 critical
busybox 1.18.5-1ubuntu4.1	1 critical 1 major
python 2.7.3-0ubuntu3.8	3 major
dpkg 1.16.1.2ubuntu7.8	1 major

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

pcre
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

busybox

VERSION	LICENSE
1.18.5-1ubuntu4.1	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 (udhcpc) 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

8.5 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.

9 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.

9.1 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:

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

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.

9.2 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.