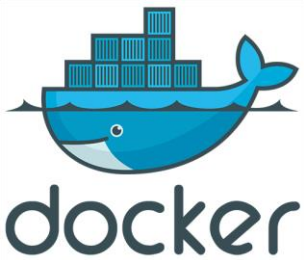
Sysgain

docker

Build, Run, & Ship Containerized Apps with Docker Enterprise Edition

Hands-on Lab Manual

This hands-on lab session will give you a brief introduction and lab experience for a Containerized Applications solution stack on Azure, built with Docker Enterprise and CloudBees Jenkins.



Introduction

The **Build, Deploy and Run Containerized Apps with Docker Enterprise Edition** Azure Partner Quick Start template launches Docker Enterprise, an integrated platform that enables agile application development and management along with Jenkins for continuous integration and continuous delivery with GitHub used as a source repository.

What You Will Learn

In this Launch & Learn session, you will learn how to leverage the Jenkins Continuous Integration/Continuous Deployment (CI/CD) platform to build and push a container-based application into the Docker Trusted Registry (DTR). Once the images are pushed, you will then leverage the Docker Universal Control Plane (UCP) to compose the application and run it on Docker Enterprise Edition container platform. You will also scale up the application and see how Docker Enterprise seamlessly scales your application.

Docker EE components:

Docker CS Engine (CS Engine)

The building block of Docker EE, the Docker Commercially Supported Engine (CS Engine) is responsible for container-level operations, interaction with the OS, providing the Docker API, and running the Swarm cluster. The Engine is also the integration point for infrastructure, including the OS resources, networking, and storage.

Universal Control Plane (UCP)

UCP extends CS Engine by providing an integrated application management platform. It is both the main interaction point for users and the integration point for applications. UCP runs an agent on all nodes in the cluster to monitor them and a set of services on the *controller nodes*. This includes *identity services* to manage users, *Certificate Authorities* (CA) for user and cluster PKI, the main *controller* providing the Web UI and API, data stores for UCP state, and a *Classic Swarm* service for backward compatibility.

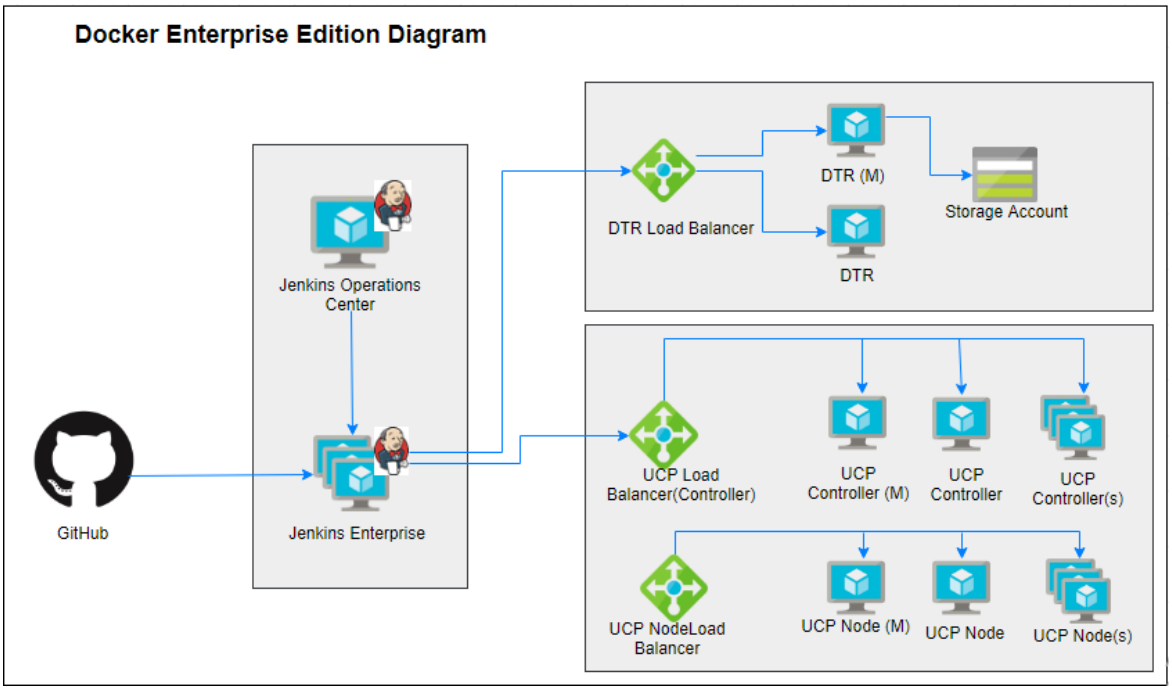
Docker Trusted Registry (DTR)

The DTR is an application managed by, and integrated with UCP, that provides Docker images distribution and security services. The DTR uses UCP's identity services to provide Single Sign-On (SSO), and establish a mutual trust to integrate with its PKI. It runs as a set of services on one or several *replicas*: the *registry* to store and distribute images, an image signing service, a Web UI, an API, and data stores for image metadata and DTR state.

Swarm Mode

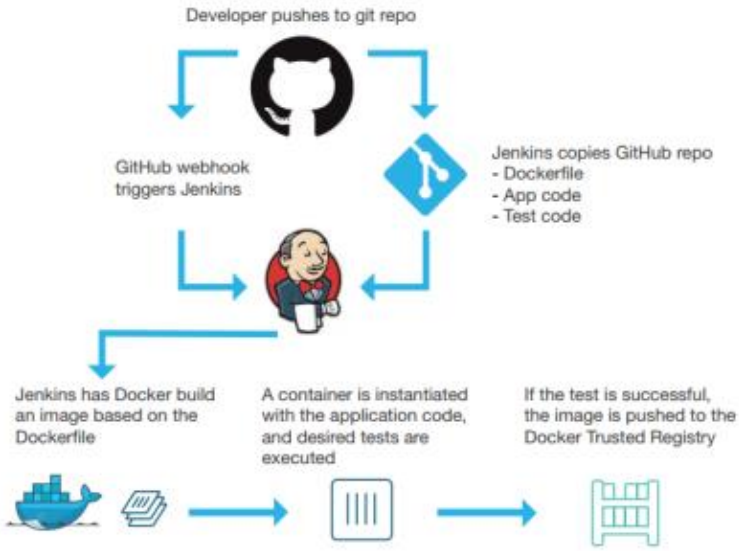
To provide a seamless cluster based on a number of nodes, DDC relies on CS Engine's swarm mode capability. Swarm mode divides nodes between *workers*, nodes running application workloads defined as services, and *managers*, nodes in charge of maintaining desired state, managing the cluster's internal PKI, and providing an API. Managers can also run workloads. In a Docker EE environment, they run UCP controllers and shouldn't run anything else.

The Swarm mode service model provides a declarative desired state for workloads, scalable to a number of *tasks* (the service's containers), accessible through a stable resolvable name, and optionally exposing an end-point. Exposed services are accessible from any node on a cluster-wide reserved port, reaching tasks through the *routing mesh*, a fast routing layer using high-performance switching in the Linux kernel. This set of features enable internal and external discoverability for services, UCP's *HTTP Routing Mesh* (HRM) adding hostname-to-service mapping.



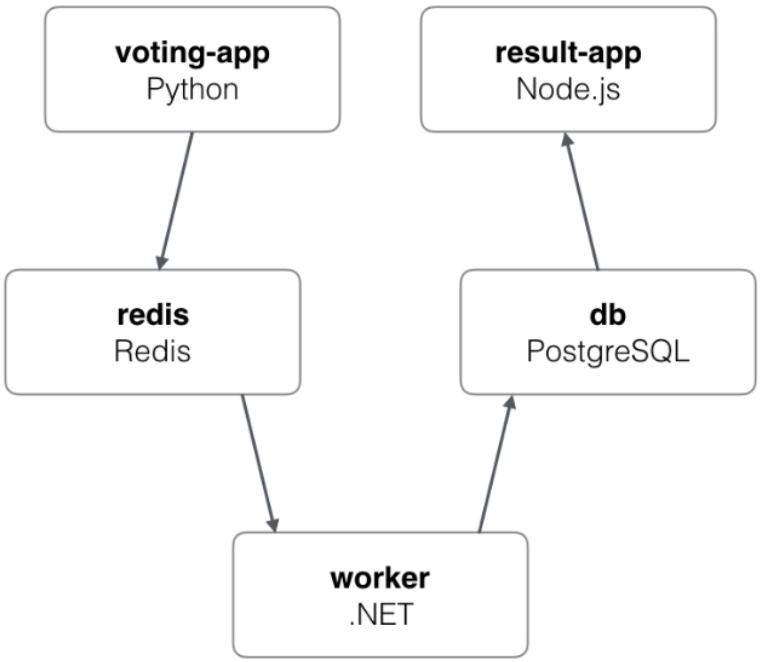
CI & CD Workflow

The diagram below shows a typical developer/operations (DevOps) workflow using Docker Enterprise Edition.



Sample Application

- To better understand the Continuous Integration and Continuous Deployment (CI/CD) process, we will use the Sample Voting Application that has been pulled from GitHub.
- The Components of the Sample Application are shown below:



- **Voting App:** A Python web-app to let you vote between two options.
- **Caching App:** A Redis queue to collect new votes.
- **Worker App:** A .NET worker to consume votes and store them in a Postgres database.
- **Results App:** A Node.js web-app to show the results of the voting in real-time

Lab Prerequisites

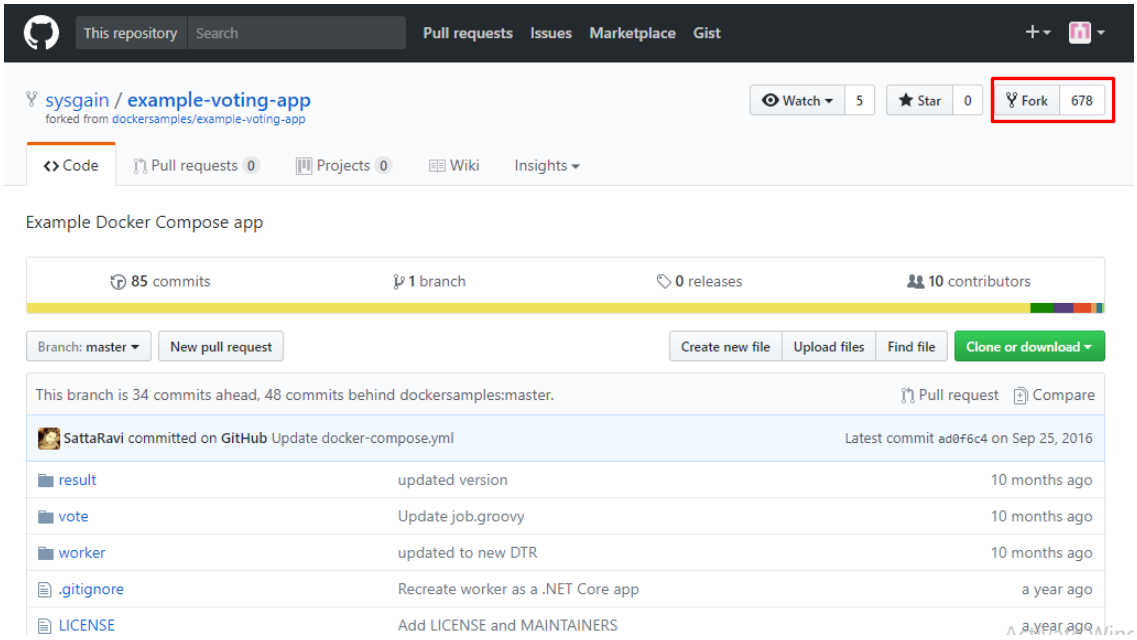
The following will already be provided by the lab instructor before or during the session:

- GitHub account. (You need to have a GitHub Account). Please sign up for a GitHub account if you do not have one at https://www.github.com
- Voting App Git repository.
- A Jenkins environment.
- Jenkins Credentials plugin ID.
- Docker Trusted Registry.
- Universal Control Panel.
- Login credentials for all the above, except GitHub account.

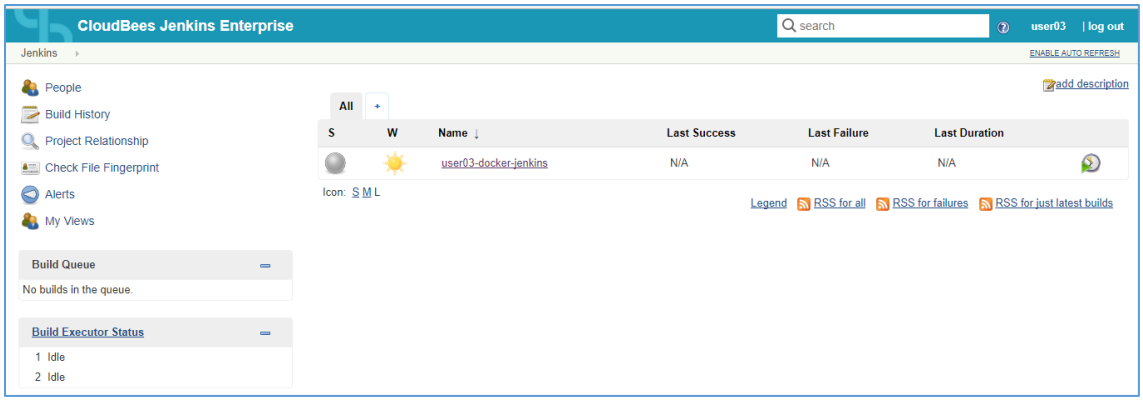
Lab Steps

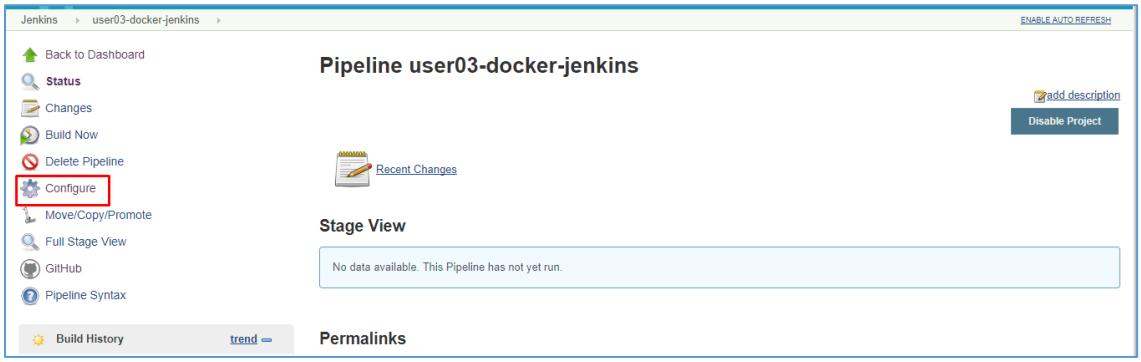
1. Login to your GitHub account (http://www.github.com) and fork the sample voting application by going to the below URL and selecting the **"Fork"** option in the top right of the page.

    - Available here: https://github.com/sysgain/example-voting-app

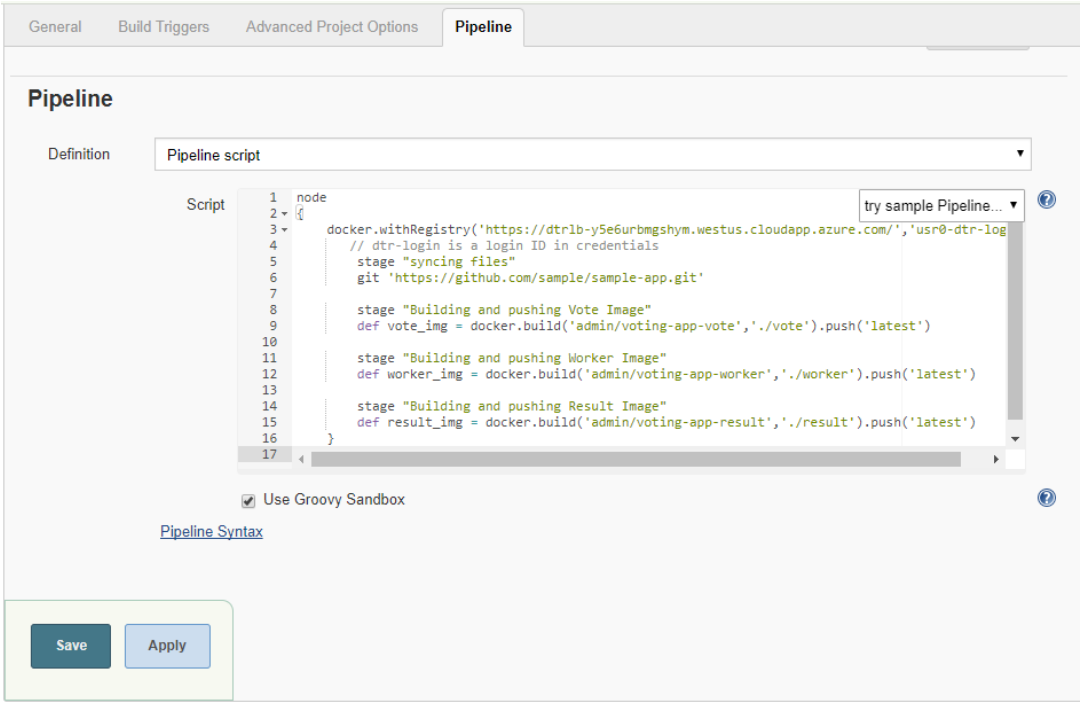


2. Once you've forked the repo, go to the Jenkins environment using the URL provided in the invitation. There, you can configure the job assigned to you (Ends with the username assigned to you). Select the Job from the list, then select '**Configure'** from the left side navigation.

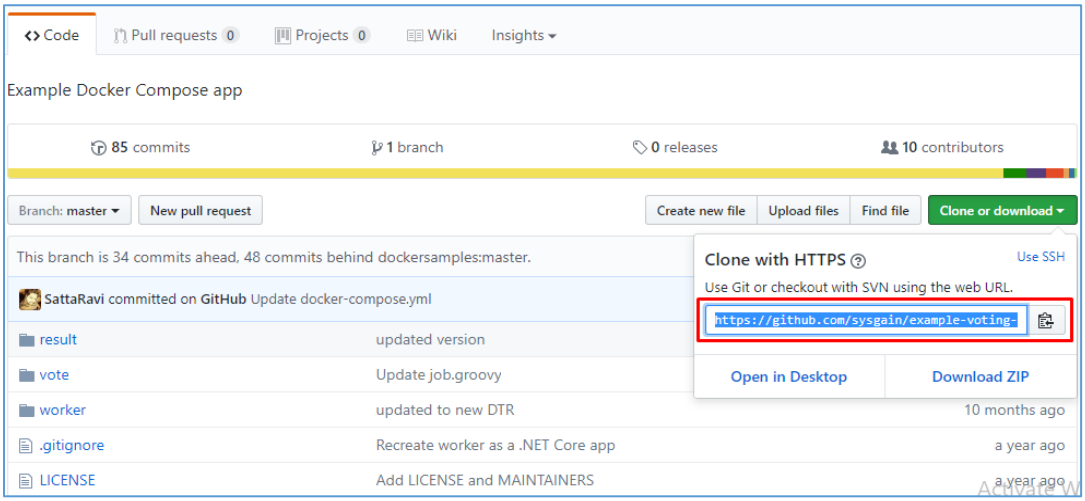




3. After selecting Configure, go to the bottom of the page and perform a search with Ctrl+F for "**usr0-dtr-login**" in the Pipeline text box. Replace that on **line 3** with the DTR credential ID you were provided at the start of the session.



4. Next, in the same Pipeline text box, replace the **GitHub URL** on **line 6** with your forked GitHub repository's URL from step 1. See the below screenshot for reference.



5. Search for the admin and replace with your user name in the same pipeline text box. (ex: user03)

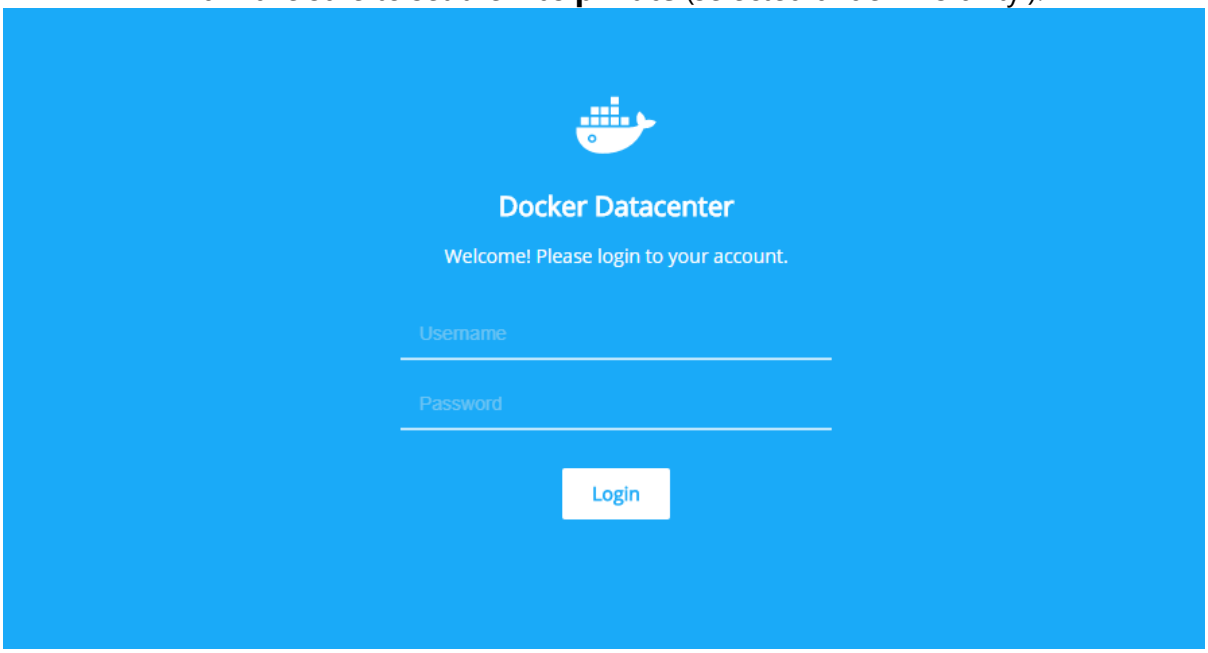
6. Next, replace the username **usr0** from **lines 9**, **12**, and **15** with your user name.

7. Select **save** to save your changes and close the job configuration. You have now configured the Jenkins job to be used later.

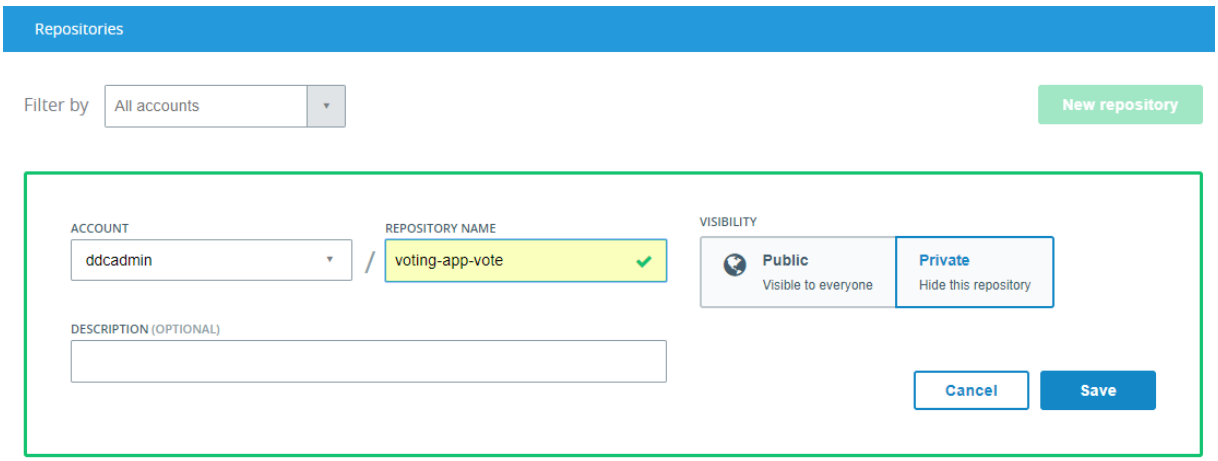
8. Login to the DTR URL in a new tab/window using the DTR Login URL provided you at the starting of the session and select '**Create'.** Create repositories with the following names:

- voting-app-result
- voting-app-worker
- voting-app-vote

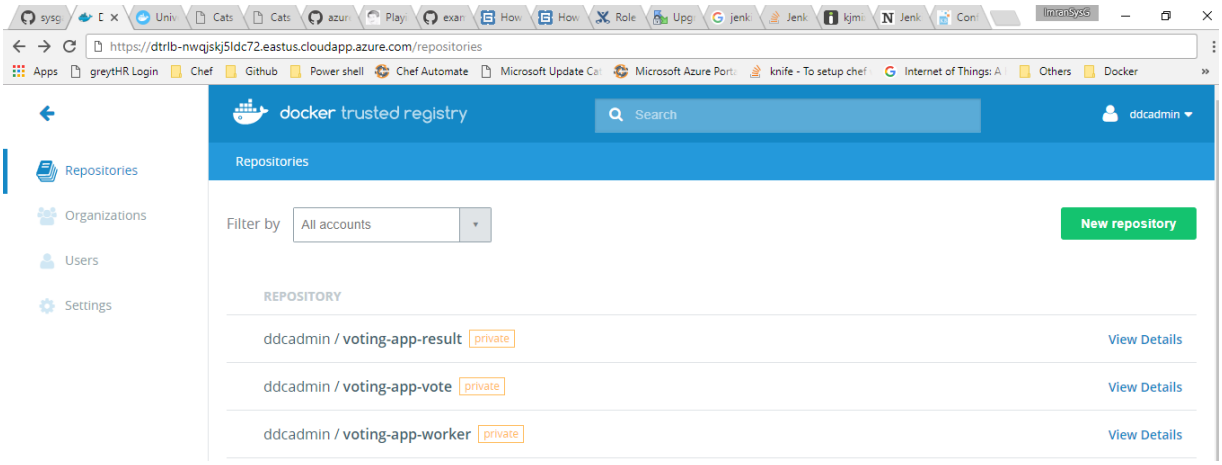
And make sure to set them as **private** (selected under 'Visibility').



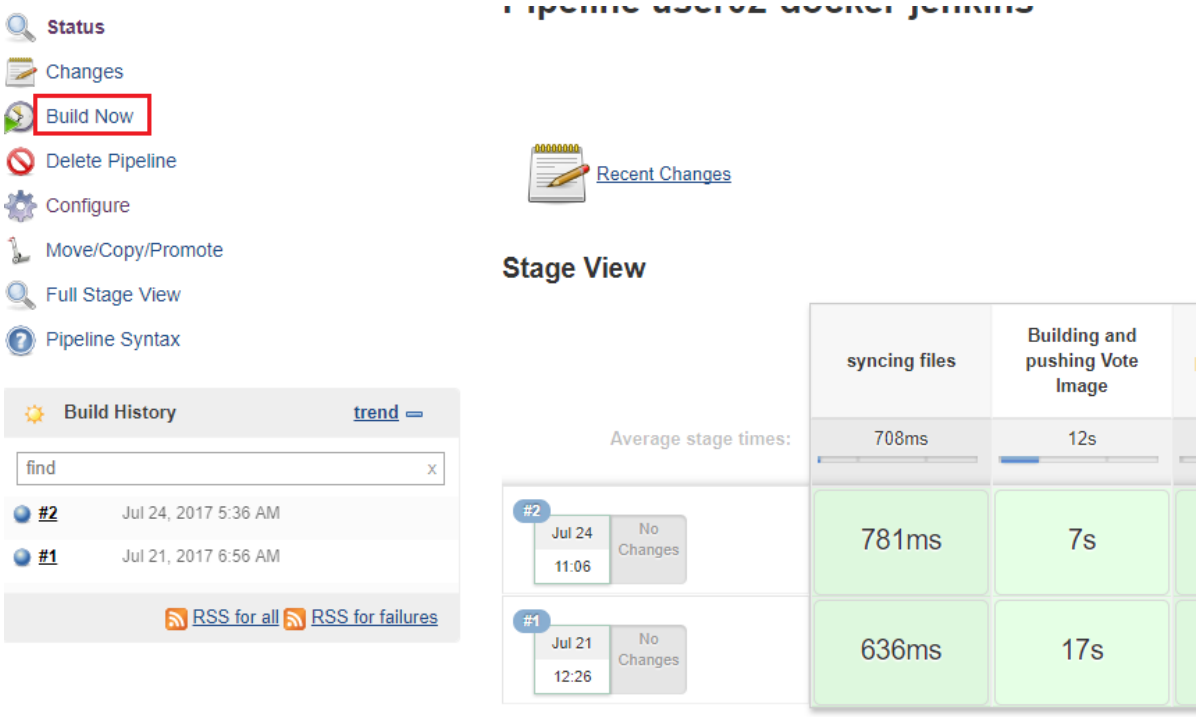
- Enter your Username and the password for the Docker Trusted Registry at the login page. You can find the Username & Password in the information provided previously – replace "**userxx**" with your assigned user name. (Example: "user03")
- The Repository creation screen looks like the screen shot below.



- After you have created the 3 repositories the DTR screen should look similar to the one below.

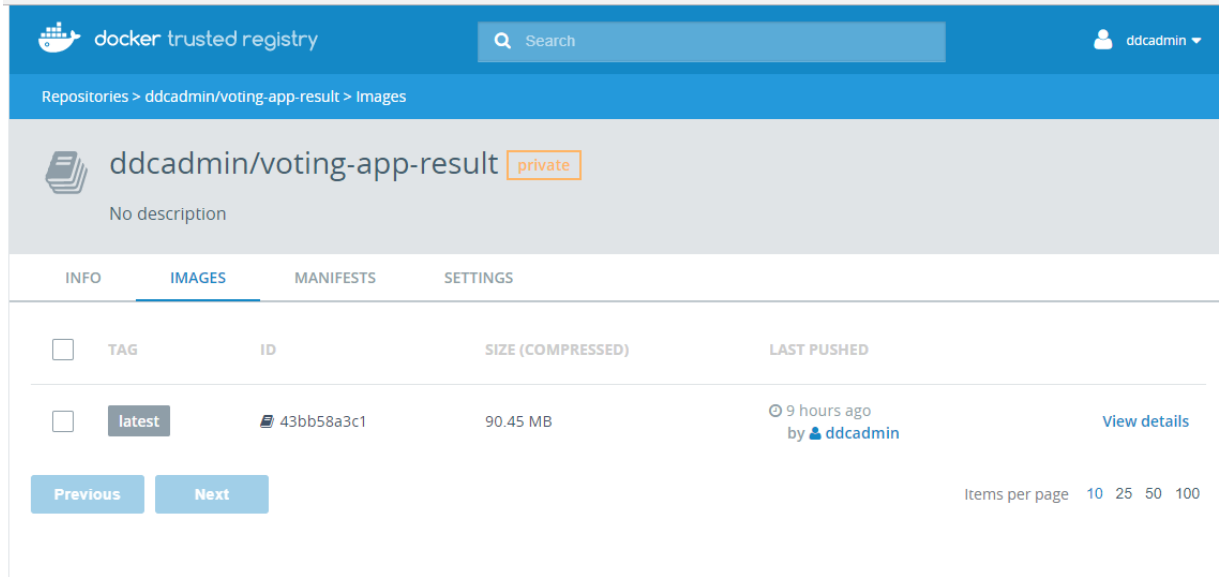


9. With the repositories in place we can now build Docker images to be pushed to the DTR using the Jenkins job you configured earlier. Return to the Jenkins page that was previously opened and click on '**Build Now'** button on the left side navigation to trigger the build process. This process builds the **Three Docker images** (Vote, Worker and Result images) and pushes them to DTR.



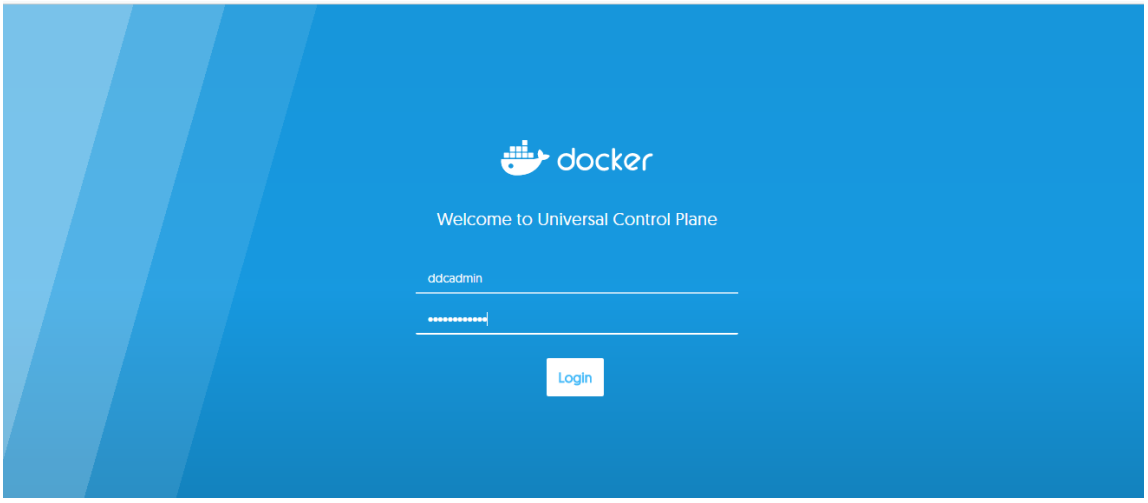
10. Once the job succeeds, we can find the images being pushed to the respective repositories we created in DTR from step 8.

    - Select one of the images. It will show details about that image, such as its **tags**, **size**, and **ID** – as shown below.

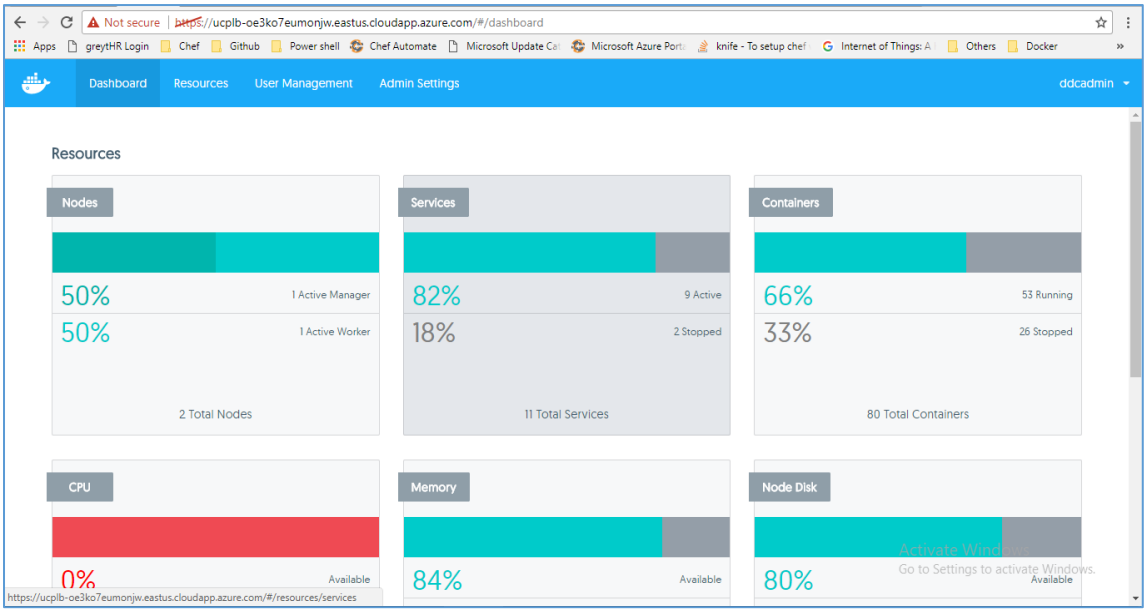


Now that you've seen these images in DTR, in the next section we will see how these images will be deployed as a containerized application in Docker Enterprise.

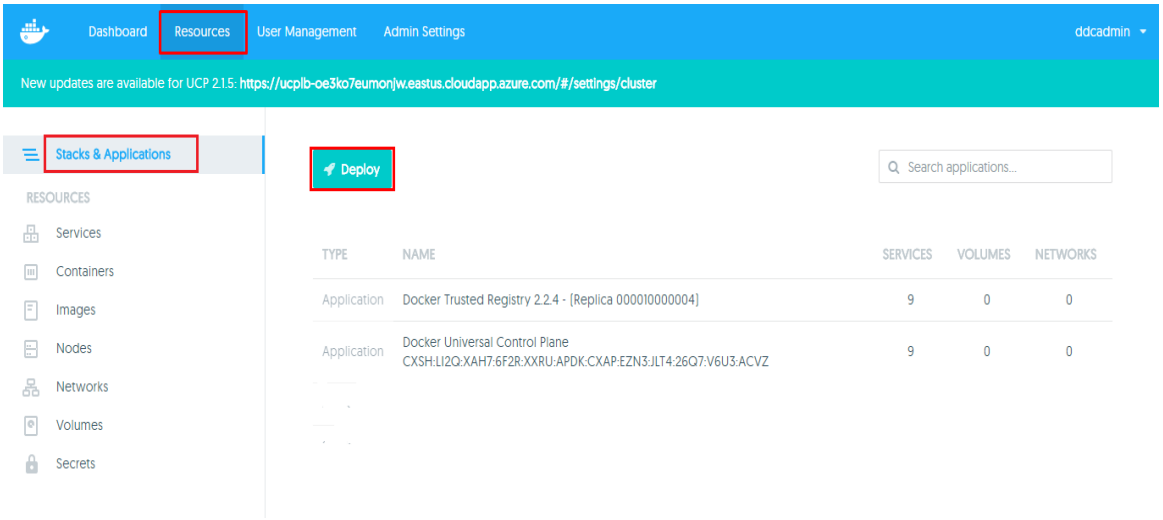
11. With the images pushed, now you can deploy the voting application from the UCP. Login to the UCP URL provided as part of the session. Open the UCP URL in a new browser tab/window. The UCP login page will look like the screenshot shown below:



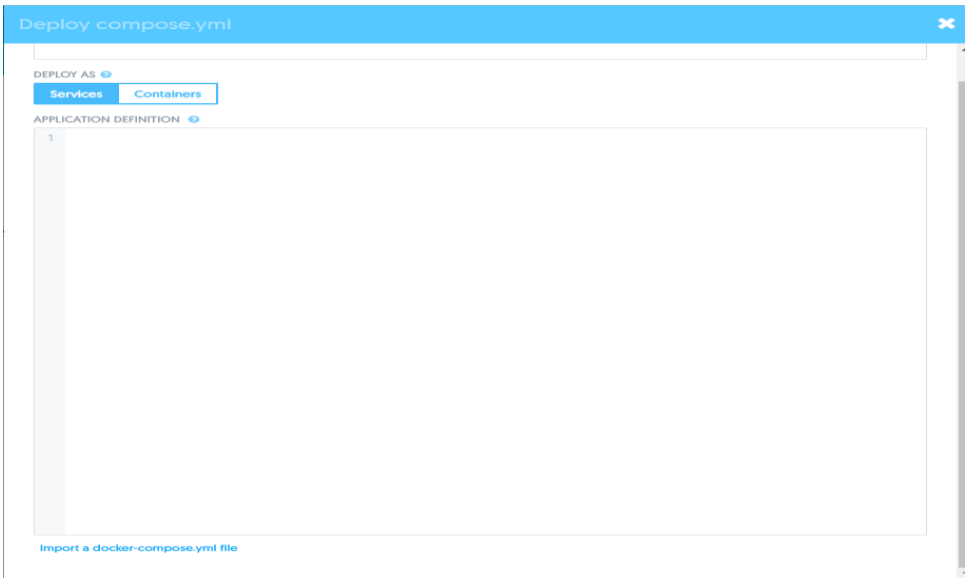
12. Once you have logged in to UCP, the screen should look like the screenshot below:



13. Next, you will deploy the **Sample Voting Application** to the Docker EE cluster running in Azure using the docker-compose.yml file. Click the **Resources** button and then go to **Stacks & Applications** and click on **Deploy**. The screenshot below shows the steps described so far.



14. After Clicking on **deploy** button the **Create Application** dialog box should appear, as shown below:



Keep this browser tab open, as we will return to it in the next section.

15. Go to the below GitHub URL and Copy the contents of the **docker-compose.yml** file to compose and deploy the application.

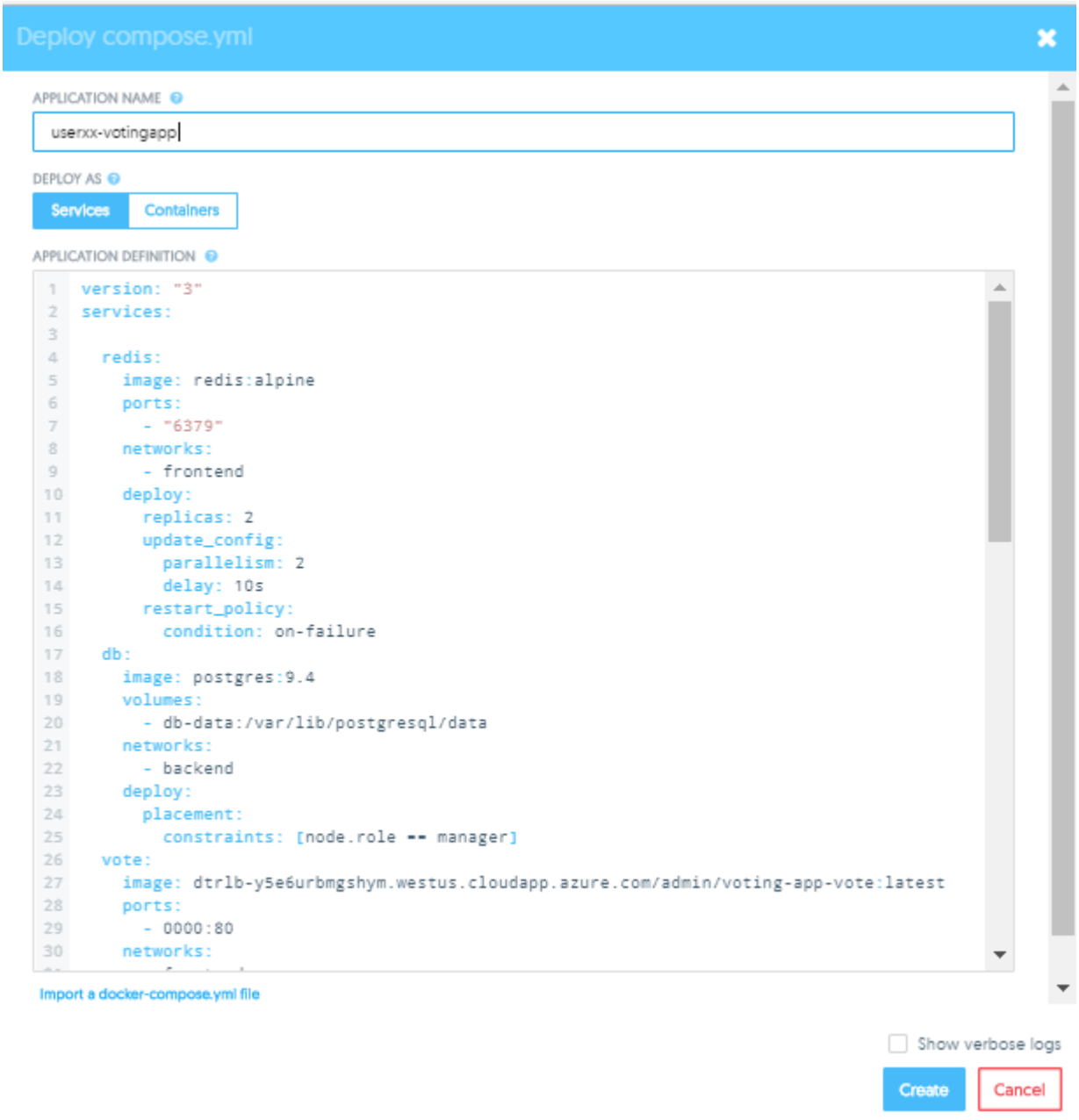
GitHub URL: https://raw.githubusercontent.com/sysgain/Docker-HOL/master/docker-compose.yml

16. Copy the entire contents of the **docker-compose.yml** file, as shown below:

```
version: "3"
services:

  redis:
    image: redis:alpine
    ports:
      - "6379"
    networks:
      - frontend
    deploy:
      replicas: 2
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
  db:
    image: postgres:9.4
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    deploy:
      placement:
        constraints: [node.role == manager]
  vote:
    image: dtrlb-oe3ko7eumonjw.eastus.cloudapp.azure.com/ddcadmin/voting-app-vote:latest
    ports:
      - 5000:80
    networks:
      - frontend
    depends_on:
      - redis
    deploy:
      replicas: 6
      update_config:
        parallelism: 2
```

17. Then, return to the UCP window/tab in your web browser and paste the copied docker-compose.yml file code into the window as shown below:
18. For the **Application Name** field, enter a unique name with the following format, as shown below:
    a. [your_username-app_name]



19. Go to lines **27, 41, and 55** in the docker-compose.yml window and replace the user "**admin**" with your own username and change the port numbers, as shown below:

APPLICATION DEFINITION

```
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    deploy:
      placement:
        constraints: [node.role == manager]
  vote:
    image: dtrlb-y5e6urbmgshym.westus.cloudapp.azure.com/admin/voting-app-vote:latest
    ports:
      - 0000:80
    networks:
      - frontend
    depends_on:
      - redis
    deploy:
      replicas: 6
      update_config:
        parallelism: 2
      restart_policy:
        condition: on-failure
  result:
    image: dtrlb-y5e6urbmgshym.westus.cloudapp.azure.com/admin/voting-app-result:latest
    ports:
      - 0001:80
    networks:
      - backend
    deploy:
      replicas: 2
      update_config:
        parallelism: 2
        delay: 10s
```

20. Go to lines **29 and 43** in the docker-compose.yml window. The default outbound port number will be 0000 & 0001. Replace the first digit with username number. If the user name is **user20,** the port should be 20000 and 20001, respectively. If the user name is **user01** then change the port number as 1000 and 1001.

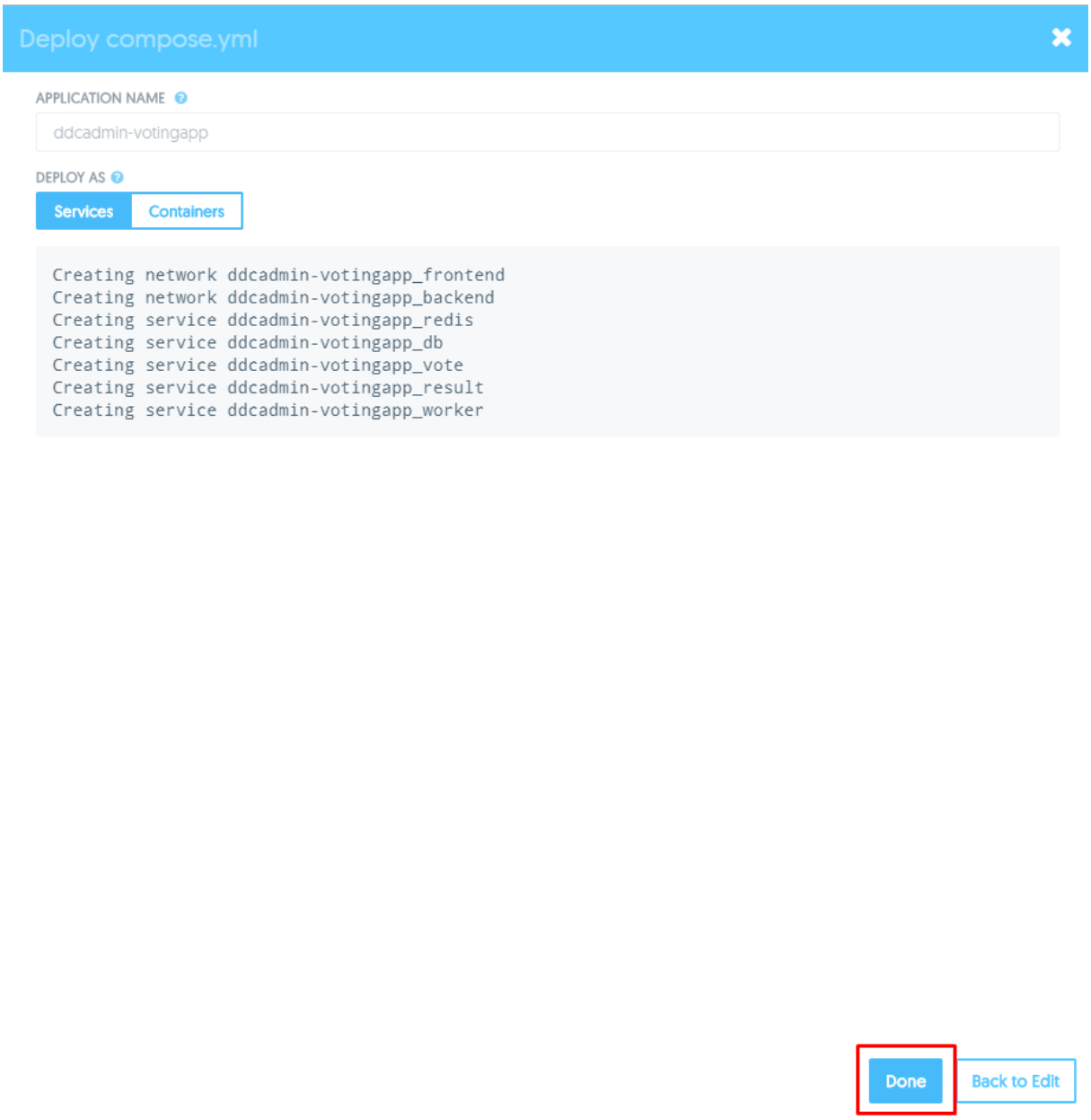
```
27    image: dtrlb-y5e6urbmgshym.westus.cloudapp.azure.com/admin/voting-app-vote:latest
28    ports:
29      - 0000:80
30    networks:
31      - frontend
32    depends_on:
33      - redis
34    deploy:
35      replicas: 6
36      update_config:
37        parallelism: 2
38      restart_policy:
39        condition: on-failure
40  result:
41    image: dtrlb-y5e6urbmgshym.westus.cloudapp.azure.com/admin/voting-app-result:latest
42    ports:
43      - 0001:80
44    networks:
```

21. Go to line **71** in the docker-compose.yml window and change the port number as per the username. If the user is **user01**, change it to **8001:8001** and for user20 the number will be **8020:8020**, as shown below:

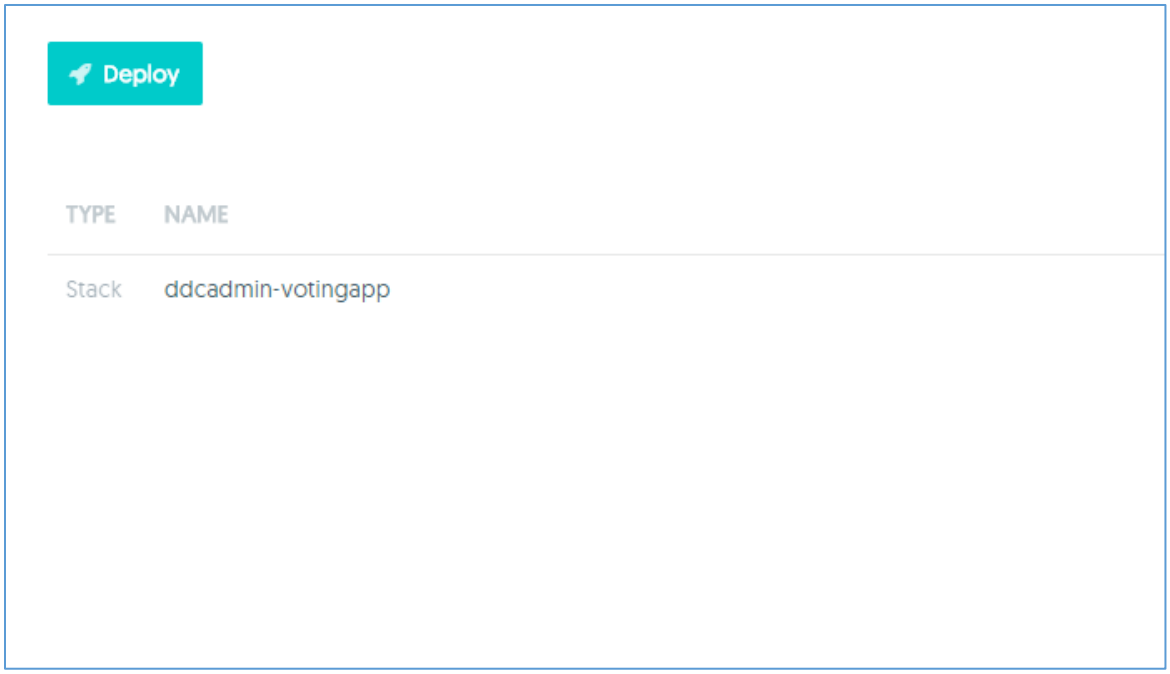
```
69          constraints: [node.role == worker]
70      ports:
71        - "8001:8001"
72      stop_grace_period: 1m30s
73      volumes:
74        - "/var/run/docker.sock:/var/run/docker.sock"
75
76  networks:
77    frontend:
```

22. Enable the **Show Verbose Logs** checkbox near the bottom of the dialog box, and click **Create** to deploy the application.

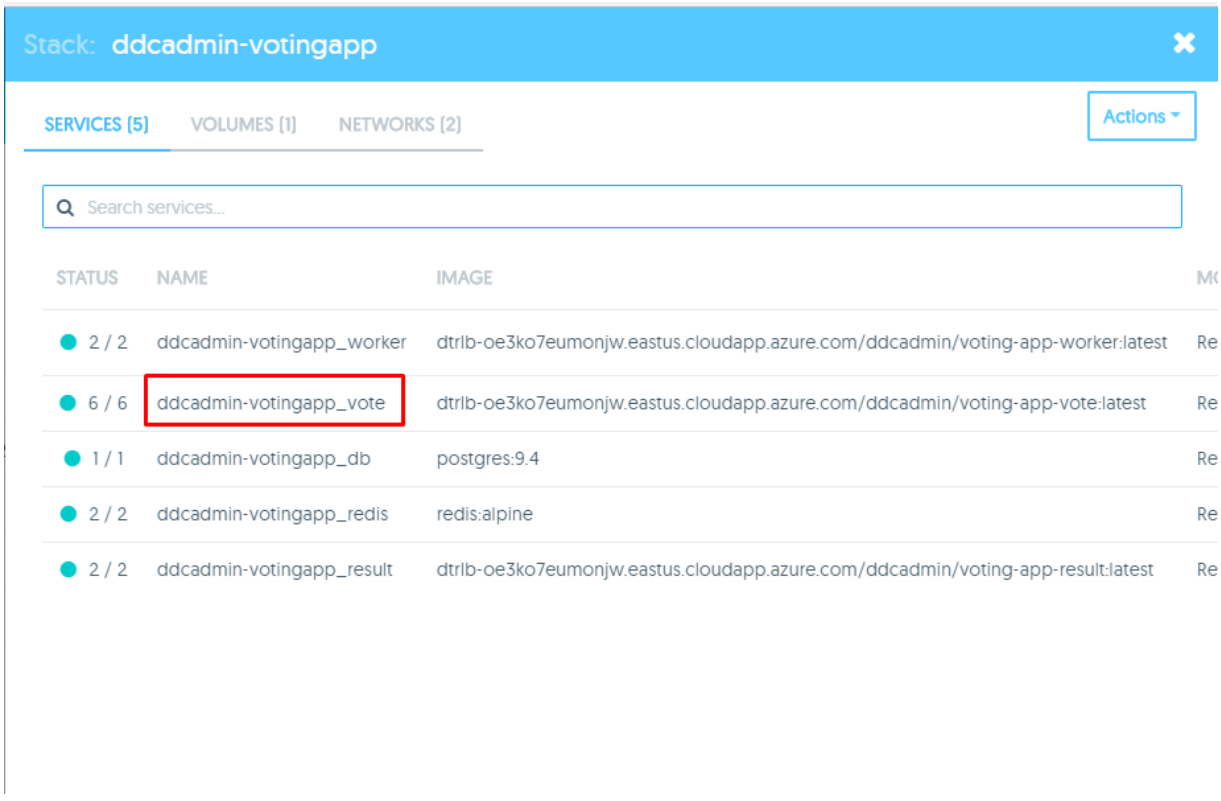
23. Once the deployment succeeds (it may take a few minutes) you will now see that your application deployed successfully, as shown below:



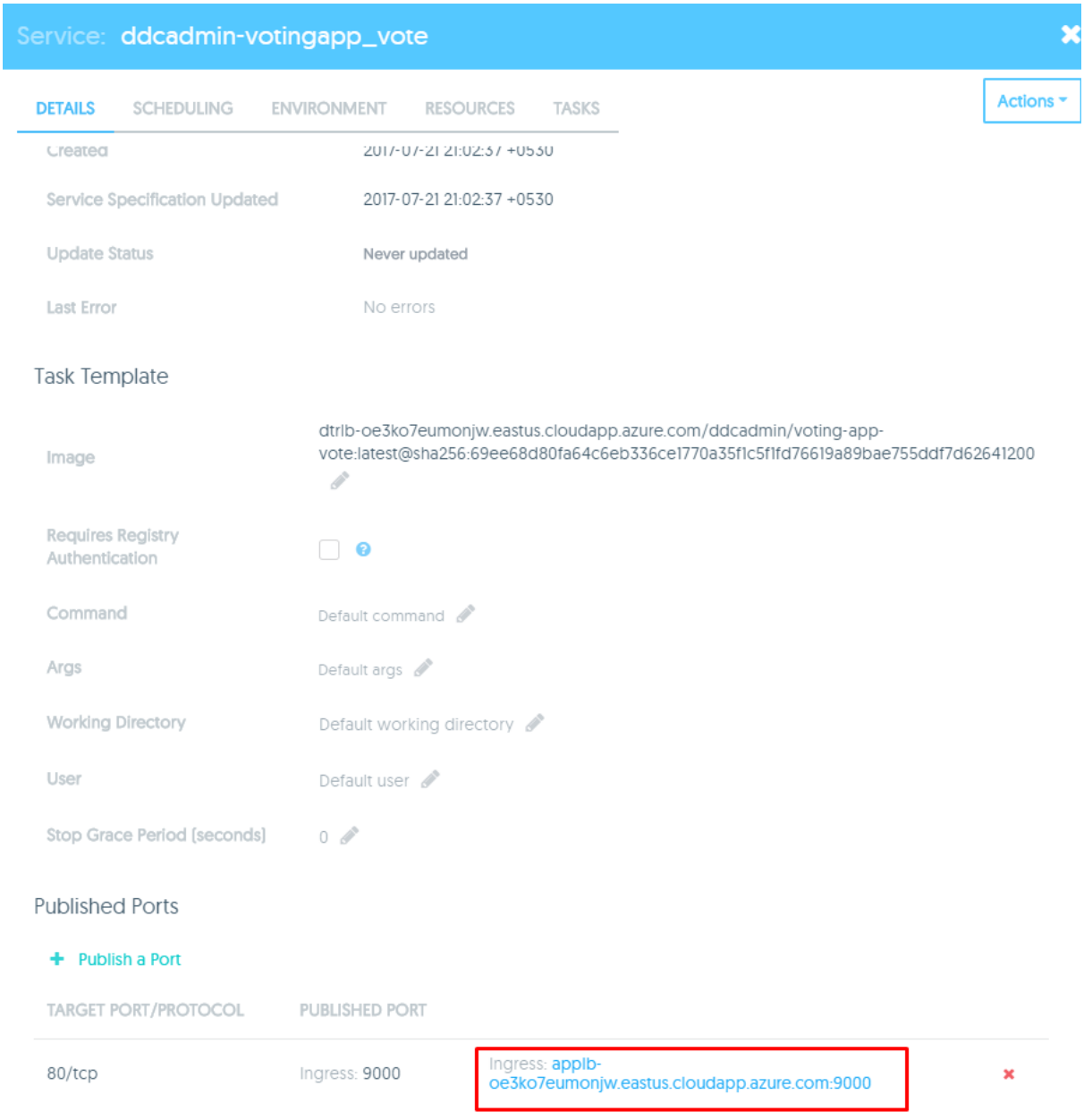
- Click the **Done** button to see your application deployed in the Universal Control Plane, as shown below:



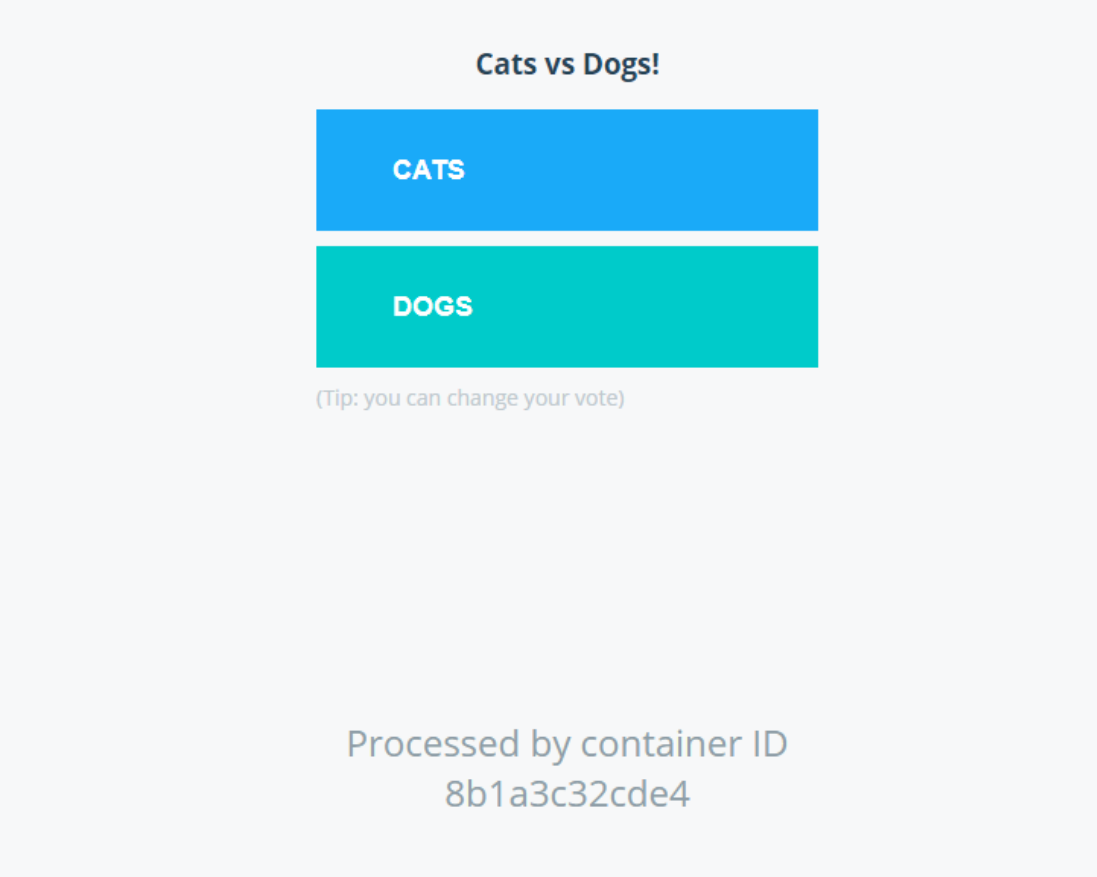
24. Click on the app that we have just created, then click on the votingapp, as shown below.



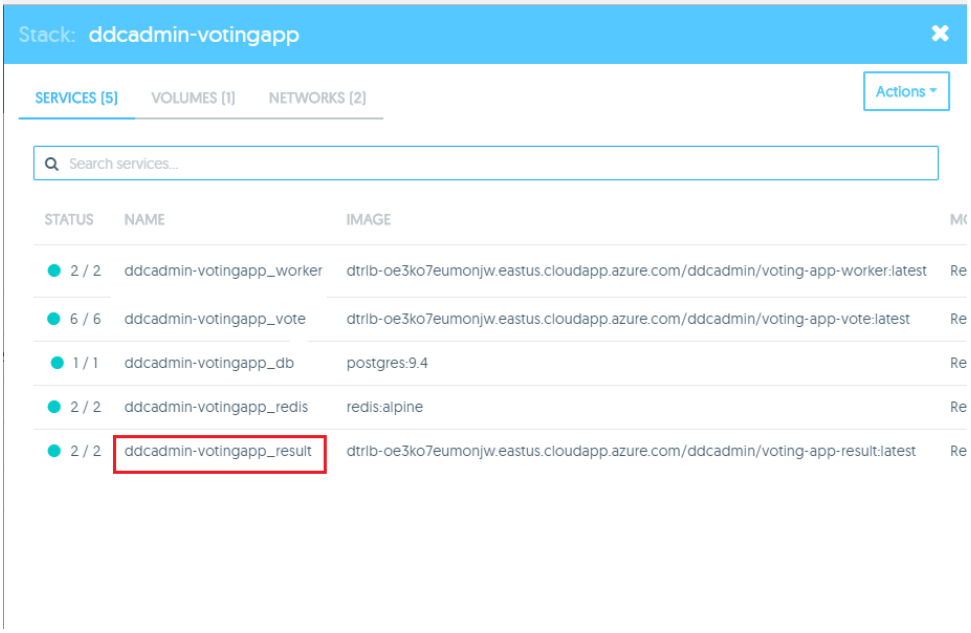
25. The voting app will now be displayed in the same browser tab. On the voting app screen, scroll down and find the URL as highlighted. Click the url to vote for Cat or Dog.



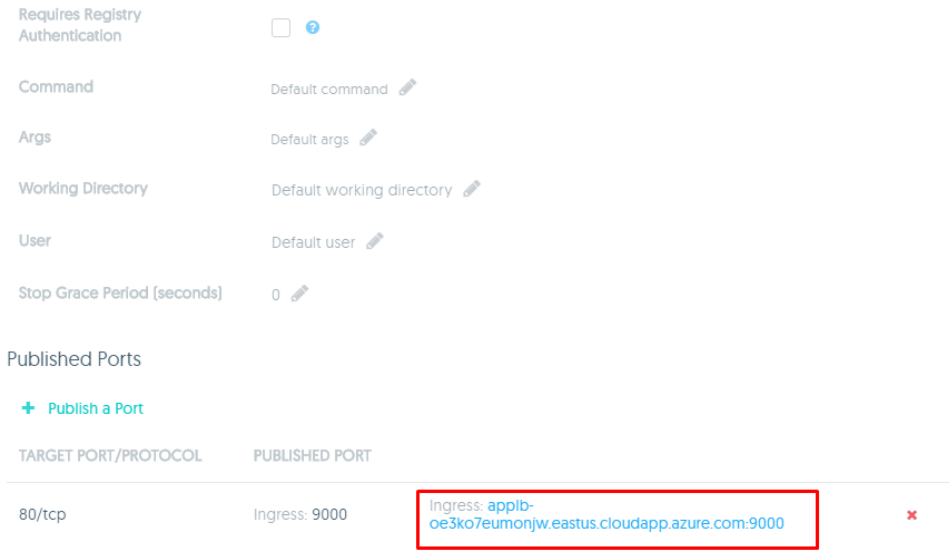
- You should see the running **Voting Application**, as shown below:



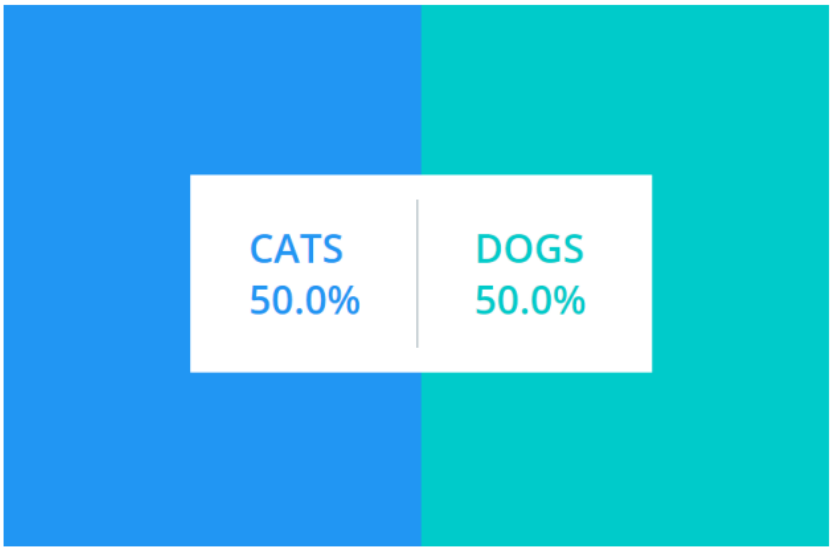
26. Click on the app that we have just created, then click on the result app, as shown below.



27. In the application, go to result app and click on it. Scroll down to get the result URL. Copy and paste that URL in the browser to see the result. As show in below.



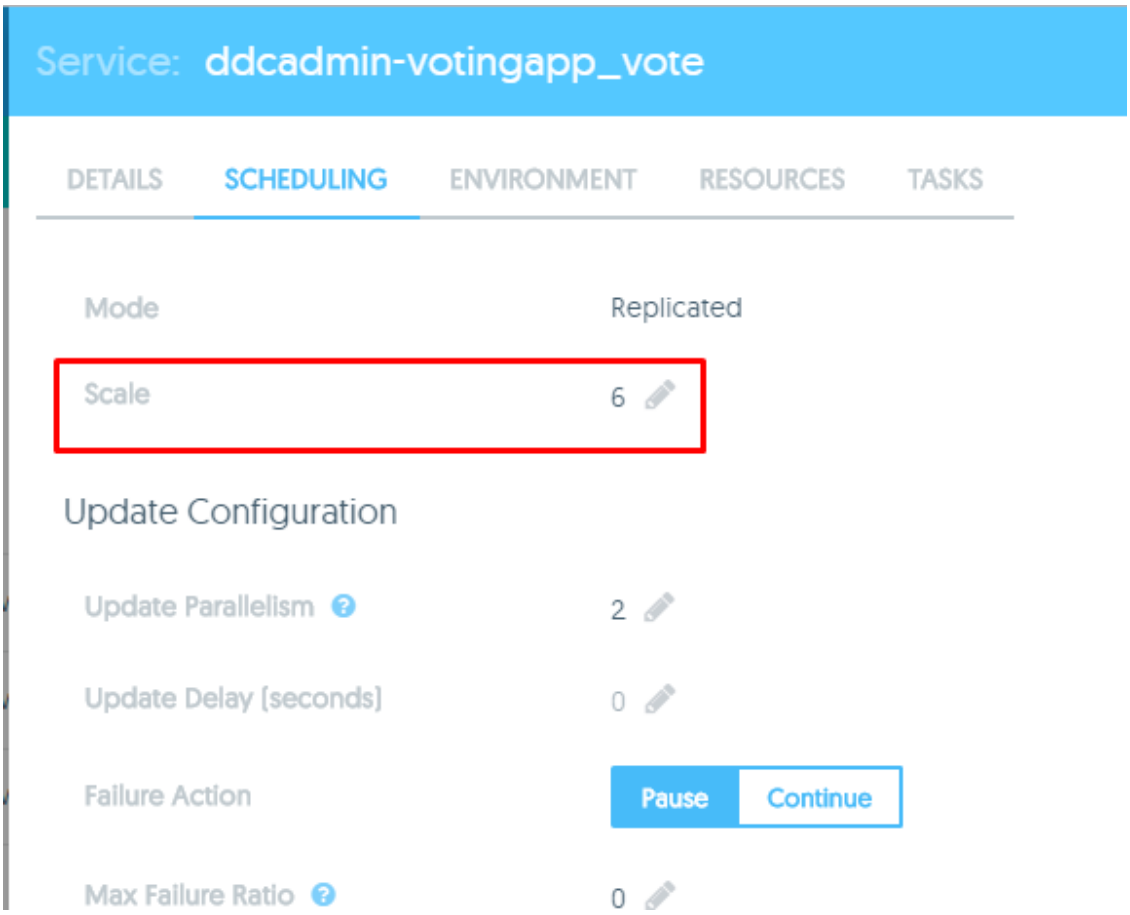
28. The result app will now be displayed in the same browser tab. Now you can see the result.



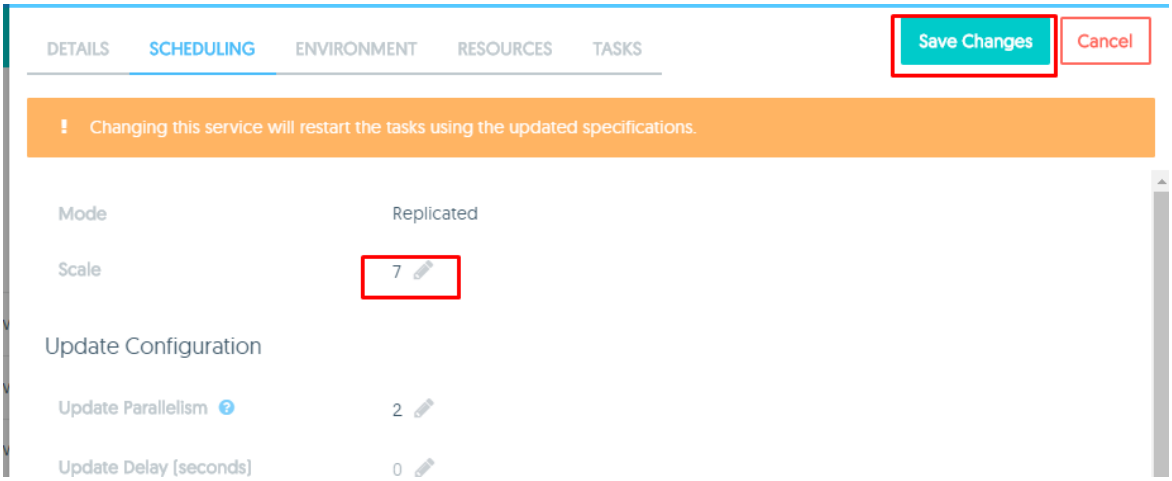
- Congratulations! You now have both the Voting and Results applications running.

29. Next, we will scale up the Voting App to 7 containers, because we believe we need additional resources to handle the load.

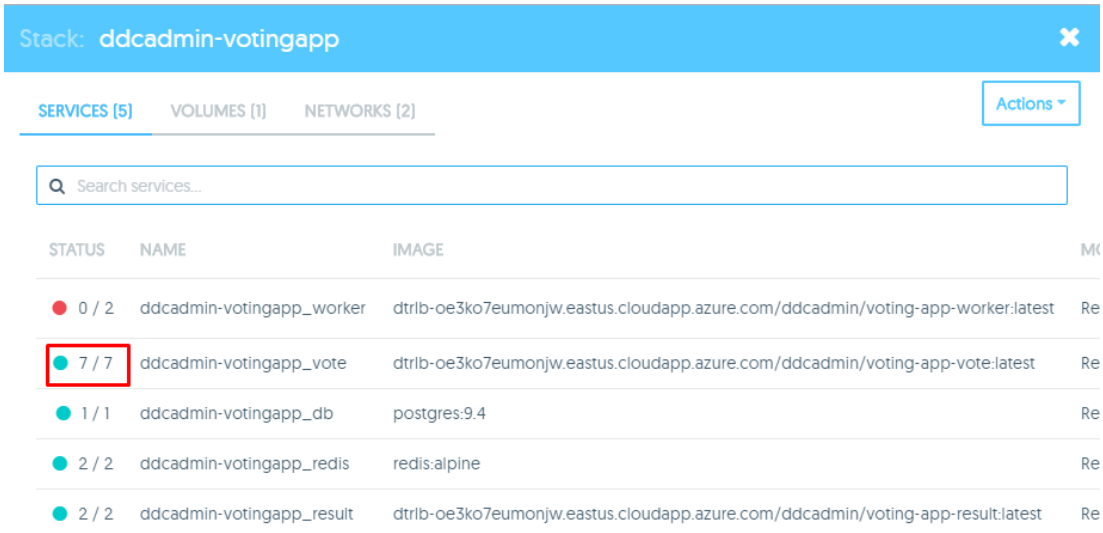
- Return to the UCP main page and select '**Resources'.**
- Go to your app and select the "**voting-app**" item. In the next pane, go to scheduling, where you see the scaling.



- Change the scaling to 7 and save it.



You should see that the total containers running on your application have increased by 1.



- **Congratulations**! You've now scaled up your containers running in the application you created in UCP.