Deploy Docker containers in hybrid clouds that use IBM UrbanCode, Part 2: Deploying Docker containers

Sanjay Nayak
Noel Richard Nihill
Mike S. Samano

September 30, 2015

This article is Part 2 of a two-part how-to guide that describes how IBM UrbanCode can be used to quickly develop and deploy multitier and multi-container applications to hybrid clouds. In it, you learn how IBM UrbanCode Build, Docker Trusted Registry, and the new Docker plug-ins for IBM UrbanCode Deploy automate the delivery of Dockerized applications to different environments, such as on-premises or hybrid cloud, in a way that provides enterprise-grade governance, traceability, control, and self-service. IBM Bluemix provides the public cloud environment, while on-premises Docker Swarm acts as the private cloud. Combined, they provide a hybrid environment for multi-container applications that are orchestrated by IBM UrbanCode. The Docker application also uses the IBM Bluemix Secure Gateway service to connect to backend containers that are running on-premises in a Docker Swarm cluster.

View more content in this series

This article is Part 2 of a how-to guide that describes in detail how you can create a multi-container DevOps environment by using IBM UrbanCode™ and Docker Trusted Registry (DTR). You can learn more in the blog post titled Deploy containers to on-premises and hybrid clouds with IBM and Docker. The blog post introduced the development and deployment of multi-container applications to a hybrid cloud environment composed of IBM Bluemix and on-premises Docker Swarm cluster.

“In Part 2, you learn how UrbanCode Deploy orchestrates the hybrid application deployment to Bluemix and Docker Swarm.”

Learn more. Develop more. Connect more.

The new developerWorks Premium membership program provides an all-access pass to powerful development tools and resources, including 500 top technical titles (dozens specifically for web developers) through Safari Books Online, deep discounts on premier developer events, video replays of recent O’Reilly conferences, and more. Sign up today.
Part 1 presents an overview and walks you through the configuration of IBM Containers in Bluemix, Docker Trusted Registry, Docker Swarm and IBM UrbanCode Build necessary to deploy Docker containers by using UrbanCode. Here in Part 2, you learn how IBM UrbanCode Deploy orchestrates the hybrid application deployment to Bluemix and Docker Swarm.

What you need to create your hybrid deployment

- A sample multi-container application. You can use The Store application for this article.
- Docker Trusted Registry (DTR)
- IBM Containers in Bluemix
- IBM UrbanCode Deploy
- UrbanCode Deploy plug-ins for Docker and IBM Containers
- IBM UrbanCode Build
- Docker Swarm

View the demo
Get the code

Step 1. Set up IBM UrbanCode Deploy

1. Install an UrbanCode Deploy server.
2. Install two UrbanCode Deploy agents. Name them ucdiceagent and ucddheagent, and connect them to the UrbanCode Deploy server.
3. On ucdiceagent:
   a. Install the IBM Containers in the Bluemix ice-cli tool.
      
      $ pip install https://static-ice.ng.bluemix.net/icecli-3.0.zip -U
   b. Install the Cloud Foundry cf CLI.
   c. Install Docker CS Engine.
   d. Install DTR certificates.
4. On ucddheagent:
   a. Install Docker CS Engine.
   b. Set the DOCKER_HOST environment variable to the Swarm host and port. For example:
      
      tcp://<Docker Swarm host>:<Docker Swarm port>.
   c. Install DTR certificates.
5. After the UrbanCode Deploy server and agents are running and the server is responding properly at the Web UI URL, check that the two UrbanCode Deploy agents show as online and error free by navigating to the Resources > Agents link in the UI.

Step 2. Set up UrbanCode Deploy Docker plug-ins

To enable UrbanCode Deploy to operate properly with ICS and Docker containers, the Docker and ICS UrbanCode Deploy plug-ins must be installed at the server.

1. Navigate to the Settings > Automation plug-ins page at the UrbanCode Deploy server UI.
2. Click the Load Plugin button.
3. Download the plug-ins.
Step 3. Configure components

You can now configure UrbanCode Deploy for the application. This begins with the idea of components. For this exercise, a component in UrbanCode Deploy is one of the four Docker container images that comprise the Store application:

- `store/backend_was`
- `store/db2_expc_nonroot`
- `store/frontend`
- `store/mysql`

UrbanCode Deploy components are created for each image. The components are one of two types, in accordance with component templates provided by the plug-in.

The `store/frontend` container, which runs the Node.js application and provides the UI, is of component type **Hybrid from Docker Registry**, which means the container image resides in a Docker registry. From there, it can be pulled, then deployed to either a local on-premises service, such as VM or Docker Swarm, or on ICS.

The remaining three containers are of component type **Docker**. They also reside in a Docker registry, but they can be deployed only to a local service.

1. Navigate to the **Configuration > Basic Settings** dialog for each component and configure it with the required settings. For example, **Source Configuration Type** is **Docker Importer** in this case because they all reside in, and are pulled from, a Docker registry.

The **Docker Registry** is the FQDN at which the Docker registry in question runs. **Docker Image Name** is the name of the image within the registry.
2. The `store/frontend` component is a little different. This type of component must be configured with information that pertains to ICS. This includes the Bluemix user ID and password, space, and organization names, image name by which the container is known
3. The UrbanCode Deploy server should now pull the newest versions of these components from the Docker registry. The server polls the registry for new image versions at a configurable interval, so this happens within the ICS registry, and the container name to be used when starting the container on ICS.
automatically. You can see this in the component's **Version** column.

Note:
The `store/frontend` component is configured to run a deployment process when a new version appears. Such processes can also be scheduled by using the UrbanCode Deploy Calendar.

**Step 4. Configure resources**

After the components are configured and available, you can configure UrbanCode Deploy Resources.

1. Under the **Resources** tab, click **Create Top Level Group** and create a top-level resource group named **all**.
2. Under **all**, you can create further resource groups, such as, in this case, **store all on swarm**, **store all on-prem**, and **store hybrid ics and on swarm**.
3. Add UrbanCode Deploy agents to these resource groups, then add the components. Add one agent to the **store all on swarm** resource group, the agent that is configured to interact with the Docker Swarm, and add all four application components to that.
4. Add two agents to the **store hybrid ics and on swarm** resource group, and add the **store/frontend** component to the agent that interacts with ICS.
5. Add the remaining three components to the agent that interacts with Docker Swarm.

![Image of UrbanCode Deploy interface]

**Step 5. Create an UrbanCode Deploy application**

1. Next, create an UrbanCode Deploy application that is named Store Application and add all four components to it.
2. Also, add two processes, which are created and configured via the UrbanCode Deploy Process Designer. The first process, `deploy`, deploys all four application components.
3. The second process, deploy store frontend only, deploys only the frontend component. This latter process is useful because it is this component that is developed and rebuilt regularly.

**Step 6. Create environments**

1. It is also necessary to create environments within the application. An UrbanCode Deploy environment can be thought of as a set of resources and configuration information that directs a particular type of application deployment. In this case, two environments are created, Hybrid Deploy to ICS.
and On-prem Swarm for Store Application, and Deploy to On-prem Swarm.

2. Each environment has resource groups and configuration information that are associated with it. The store hybrid ics and on swarm resource group is associated with the Hybrid Deploy to ICS and On-prem Swarm for Store Application environment.
3. The store all on swarm resource group is added to the Deploy to On-prem Swarm environment.

4. Under the Configuration tab for each environment are fields for entering Docker and ICS settings for the various application components at deployment time. In both environments, configure the various settings for the three on-premises containers. Settings such as Docker port-mappings, container linking directives, DB2 and MySQL authentication information. These are the Docker Environment Run Options, split out per-component. Within the Deploy to On-prem Swarm environment, the front end component is also configured in this way, for example, as a Docker component. Within Hybrid Deploy to ICS and On-prem Swarm for Store Application, however, the front end component receives its configuration not via the Docker Environment Run Options, but via the ICS Environment Run Options.
5. The key difference between the two environments is the value of the **Resource Type** property. For the Hybrid environment, set it to `ics`. For the on-premises environment, set it to `docker`. It is this that dictates the precise path that processes associated with the environment take, for example, whether to pull from the Docker registry and run locally, or to pull from the Docker registry, tag-and-push to ICS, and run on ICS by using the `ice-cli` tool.

6. The configuration of the front end component is also slightly different within each environment.
For ICS, the component is configured to bind to a Bluemix Cloud Foundry (CF) application so that it has access to information about the Bluemix Secure Gateway that is available through the CF application. This allows the store/frontend container to communicate with the rest of the component containers back on the on-premises system, in particular the store/backend_was container.

For Docker, the front end component is provided with a local environment file that contains the same VCAP gateway information that can be acquired programatically when running on ICS. This file is stored on the host or VM that the UrbanCode Deploy server and agents are running on, and is made available to the UrbanCode Deploy containers at the following location via the Docker volume:

```
/opt/urbancode/ibm-ucdagent/var/etc/vcap_services.env
```

7. In both environments, the front end component is provided with the Secure Gateway name, the Destination name within that Gateway, the Bluemix user ID and password, and the ports to expose.

You can run individual processes from either environment by either deploying the entire application, or deploying the front end component only. Depending on the environment, the front end is deployed either on ICS or on-premises.

When running a process, the versions of the components must be selected, as well as the option to deploy components even if their versions are unchanged since the last deployment.

**Step 7. Deploy application**

You created two UrbanCode Deploy environments:

- **On-premises**: To deploy the front end and three backend containers to on-premises Swarm.
- **Hybrid**: To deploy the front end container to IBM Containers in Bluemix, and three backend containers to on-premises Swarm.

For running either of those scenarios, run the following common steps first:

1. Go to the store/frontend application directory.
   ```bash
   $ cd store_frontend/public/css
   ```
2. Change `custom.css` by copying over `colorLight.css` or `colorDark.css`.
3. Run Git commit.
   ```bash
   $ git commit -a -m "changed store color"
   ```
4. The Git post-commit trigger kicks off the application build in UrbanCode Build.
5. Go to the UrbanCode Build portal and view the status of the build. When it completes you have a new Docker image, such as 1.0.n, where \( n \) is the incrementing build number.

6. On the Component: store/frontend page in the UrbanCode Deploy portal, check the Version column to make sure that the new 1.0.n build was imported.

7. In UrbanCode Deploy, go to the environments of the Store Application and deploy either the On-premises or Hybrid environment as described in the following sections.

On-premises deployment with UrbanCode Deployment

For this example, run the deployment of all containers to on-premises first.

1. Log in to your UrbanCode Deploy portal and go to Store Application environments.
2. Run the Store: Deploy to On-prem Swarm environment.
3. Choose its **Deploy** process. Clear the **Only Changed Versions** check box so that backend containers that were not updated are also deployed.

4. Choose the newest available versions of `store/frontend` and the three backend containers.

Click OK.

5. The status and completion of deployment is displayed, as shown below:

6. Check the Swarm Manager to see the containers that are running.
sanjay@dhe:/swarm$ docker -H 192.168.56.30:3375 ps
CONTAINER ID        IMAGE                                               COMMAND                STATUS              PORTS
   e96ef9f6affe        dhe.cork.ie.ibm.com/store/store_frontend:1.0.109    "/usr/bin/supervisor"   Up 3 minutes        22/tcp, 192.168.56.30:3000->3000/tcp
                      vagrant-ubuntu-trusty-64/store_frontend
   c8fd9c7b3247        dhe.cork.ie.ibm.com/store/backend_was:latest        "/opt/startup.sh"      Up 4 minutes        8879/tcp, 192.168.56.30:9060->9060/tcp, 9043/tcp, 9443/tcp, 9080->9080/tcp
                      vagrant-ubuntu-trusty-64/store_backend_was
   1bb80b195645        dhe.cork.ie.ibm.com/store/mysql:latest              "/opt/startup.sh"      Up 4 minutes        22/tcp, 192.168.56.30:3306->3306/tcp, 192.168.56.30:32768->3000/tcp
                      vagrant-ubuntu-trusty-64/store_backend_was/mysql,vagrant-ubuntu-trusty-64/store_mysql
   cff666d21789        dhe.cork.ie.ibm.com/store/db2_expc_nonroot:memfix   "/opt/startup.sh"      Up 4 minutes        192.168.56.30:5000->5000/tcp
                      vagrant-ubuntu-trusty-64/store_backend_was/db2,vagrant-ubuntu-trusty-64/store_db2_expc_nonroot

7. Check the logs of the store_backend_was container to verify that its initialization is completed and server is running.
   $ docker logs -f store_backend_was

8. Go to The Store front end URL, http://<host>:3000, to verify that it is running.

Hybrid deployment with UrbanCode Deploy

Now that all containers are running in on-premises swarm, run the store_frontend container in Bluemix and connect it to backend containers that are running on-premises.

1. Log in to your UrbanCode Deploy portal and go to the Store Application environments.
2. Run the Store: Hybrid Deploy to ICS and On-prem Swarm environment.
3. Choose the `Deploy store frontend only` process and click **Submit**.

![Run Process on Store: Hybrid Deploy to ICS and On-prem Swarm](image)

4. An approval step for deployments to the Hybrid environment is required, so a designated approver must approve the submission.

![Approval Progress](image)

5. After approval is submitted, the deployment continues. The new Docker image is pushed to the IBM Container in Bluemix, the currently running `store_frontend` container is stopped, a new `store_frontend` container is started with the new image, and a public IP is bound to it. Upon completion, the status is similar that shown below.

![Deployment Steps](image)

6. Now go to your Bluemix user interface and check that a `store_frontend` container is running with your newly built Docker image version. Note the public IP of your `store_frontend` container.

7. Go to The Store front end URL, `http://<host>:3000`, to verify that it is running. This front end container is connected via the Bluemix Secure Gateway service to the `store_backend_was` container that is running in the on-premises Docker Swarm cluster that was started in the previous On-premises scenario.
Conclusion

In Part 2 of this two-part article, you saw how a multitier and multi-container application can be quickly deployed to both an on-premises Docker Swarm cluster, and an off-premises cloud environment such as IBM Containers in Bluemix. You used the enterprise features of IBM UrbanCode Deploy along with the IBM-supported Docker Trusted Registry offering as the trusted repository for Docker images.

You learned how UrbanCode products integrate with Docker Trusted Registry, as well as orchestrate and automate the deployment of multiple containers to multiple cloud environments as part of a typical DevOps pipeline. And you saw how an application that is running in IBM Containers in Bluemix can securely connect to on-premises infrastructure by using the Bluemix Secure Gateway service, with the potential to use the vast catalog of other Bluemix services.

Please provide feedback on this two-part article by posting your comments. Our aim is to help customers make the most of Docker containerization in their enterprise cloud environments!
Related topics

- IBM UrbanCode – Docker Trusted Registry Blog
- IBM UrbanCode – Docker Trusted Registry Demo

© Copyright IBM Corporation 2015
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)