Automate rolling deployments with IBM DevOps Services for Bluemix

Daniel Berg
IBM Distinguished Engineer, CTO DevOps Tools and Strategy
IBM

22 September 2014

Developers who build applications on IBM Bluemix strive to achieve zero downtime to avoid disruptions in service and keep users happy. Achieving zero downtime while delivering new functions is challenging and requires the ability to perform a rolling deployment. This tutorial walks you through the setup and use of the Delivery Pipeline service from IBM DevOps Services for Bluemix to automate a rolling deployment process.

The best way to attract new users and keep existing ones is to create engaging applications that never go down. Consumers have little tolerance for sites that have outages (scheduled or otherwise). And as your team increases the velocity at which it delivers changes, it becomes even more important to avoid breaks in service.

Rolling deployments are a process by which a new version of an application is deployed into an environment with no disruption in service for the consumer. You *must* implement rolling deployments if you want to increase your user base.

IBM Bluemix™ provides a simple-to-use platform that enables rolling deployments. I'll show you how you can easily automate rolling deployments to Bluemix spaces by using IBM DevOps Services and the Delivery Pipeline service.

What you'll need

- Bluemix and IBM DevOps Services accounts.
- Basic understanding of the IBM Delivery Service capabilities, such as how to create a default Builder and Deployer. If necessary, review the Getting Started with IBM Bluemix and DevOps Services Using Java tutorial.
- Bash scripting knowledge.

See the pipeline
Get the example code

“ Rolling deployments are a necessity for any production-level application to avoid disruptions in service. ”

READ: Getting Started with IBM Bluemix and DevOps Services using Java

WATCH: BlueGreenDeployment http://martinfowler.com/bliki/BlueGreenDeployment.html

I set up a simple Hello World example with an automated rolling deployment pipeline. You can examine the code while reading. The pipeline is publicly available, so you can review the deployment automation configurations and deployment logs.

Rolling deployments are often called *blue-green* deployments, *red-black* deployments, or *zero-downtime* deployments. Although these deployment processes differ in subtle ways, they share the core principle of avoiding a disruption in service for the consumer.

I'll walk through the steps that I used to establish an automated rolling deployment in DevOps Services. Use these steps as an example only, adjusting the steps and scripts to fit your needs — or to improve upon my rudimentary bash scripting abilities.

Step 1: Fork the sample project

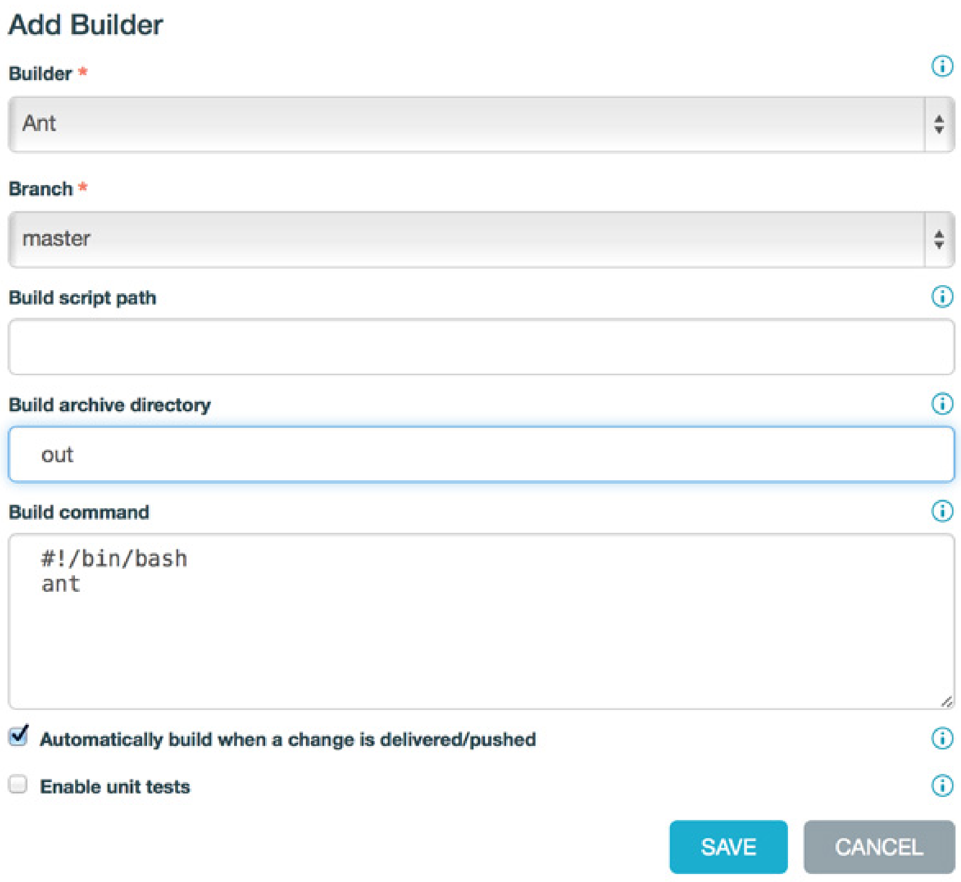
Start by forking the sample project so that you have your own DevOps Services project with the necessary source:

1. Scroll up and click this tutorial's **Get the example code** button.
2. In the danberg | Pipeline_hello project's overview page, click the **EDIT CODE** button (enter your DevOps Services credentials if you're not already logged in) and click the **FORK** button on the menu.
3. Enter a name for your new project, select the **Deploy to Bluemix** check box, and select an organization and space for billing purposes.
4. Click **Save** to create the new project with the source code.

Step 2: Add a new build stage

After the project is created, start setting up the delivery pipeline by adding a build stage to the pipeline:

1. Click the **BUILD & DEPLOY** button at the top of the page.
2. No delivery process is enabled by default. Click the **ADVANCED** button to turn on the advanced delivery pipeline feature.
3. Click the **add a builder** button to configure a builder for the project. Keep the defaults (Ant builder) and enter `out` in the **Build archive directory** field:



Add Builder

Builder *: Ant

Branch *: master

Build script path:

Build archive directory: out

Build command:

```
#!/bin/bash
ant
```

- [✓] Automatically build when a change is delivered/pushed
- [ ] Enable unit tests

4. Click **SAVE** to save the builder settings.
5. Click the **add a stage** button to configure a deployer. Click **SAVE** to save the default settings. You'll adjust the deployer for rolling releases in the subsequent steps.

Step 3: Adjust the delivery pipeline to use a deploy script

Because rolling deployments are a bit more involved than other deployment types, it's desirable to manage the deployment script within your SCM. You need to adjust your builder and deployer to use the deployment script:

1. Click your project's **EDIT CODE** button.
2. Select the build.xml file and notice that it includes the following snippet to copy the deploy.sh file — the script used for the rolling deployments — to the archiving directory. This code is necessary to ensure that the deploy.sh file is available during the deployment stage:

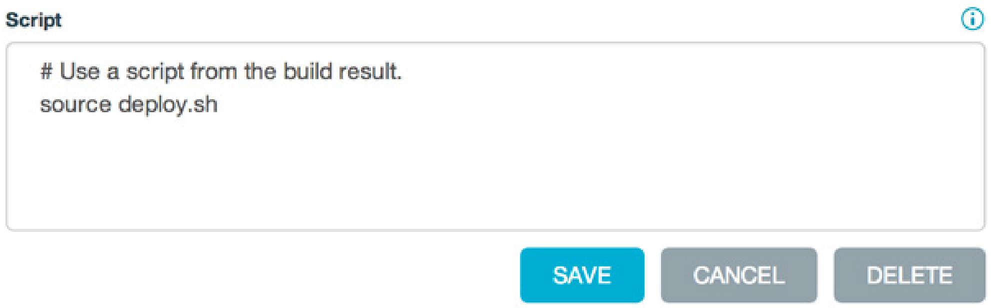
```
<echo message="Copy files to ${artifact_dir} for deployment"/>
<copy todir="${artifact_dir}">
    <fileset file="manifest.yml" />
    <fileset file="deploy*.sh" />
</copy>
```

3. Open the pipeline by clicking the **BUILD & DEPLOY** button.
4. Click the configure icon ⚙ in the upper-right corner of the Deployer to edit the configuration.
5. Set the **Application name** to a value that has meaning to your project. Note that this name will be used to compute a new application name during the deployment process.

6. Replace the contents of the Script section with:

```
# Use a script from the build result
source deploy.sh
```

   Using `source` within the script body is the correct way to call a bash script that is included with your build artifacts:



Script

```
# Use a script from the build result.
source deploy.sh
```

7. Click **SAVE**.

Step 4: Examine the rolling deployment script

Follow along in the example deploy.sh script file as I describe the main sections needed to automate a rolling deployment with a bash script. (You don't need to make any changes in this step.)

1. At the top of the script, define variables to be used later:

```
#############
# Colors    #
#############
green='\e[0;32m'
red='\e[0;31m'
label_color='\e[0;33m'
no_color='\e[0m'

# Used to create a public route within the space for accessing the deployed
# application. The reserved CF_SPACE variable (configured in the deployer) is
# used to unique the route name per space.
if [ "${CF_SPACE}" == "prod" ]; then
   PUBLIC_HOST=${CF_APP}
else
   PUBLIC_HOST="${CF_SPACE}-${CF_APP}"
fi

# Extract the selected build number from the reserved BUILD_SELECTOR variable.
SELECTED_BUILD=$(grep -Eo '[0-9]{1,100}' <<< "${BUILD_SELECTOR}")

# Compute a unique app name using the reserved CF_APP name (configured in the
# deployer or from the manifest.yml file), the build number, and a
# timestamp (allowing multiple deploys for the same build).
NEW_APP_NAME="${CF_APP}-$SELECTED_BUILD-$(date +%s)"

# Domain can be customized if needed.
DOMAIN=mybluemix.net
# Used to define a temporary route to test the newly deployed app
TEST_HOST=$NEW_APP_NAME
# Java artifact being deployed. Needs to match output from
# the build process.
WAR=Pipeline_hello1.war
MEMORY=256M
```

`NEW_APP_NAME` is important because it shows how I compute a new application name for each deployment based on the build number and the time stamp. The color variables are used to make the log files easier to read. The application name specified in the Deployer configuration is referenced in the script with the `CF_APP` environment variable, and the target space with the `CF_SPACE` variable.

2. Find the existing deployed application names within the space (to be deleted later). I search for all apps within the space that are associated with the `PUBLIC_HOST`, ignoring the computed build number and time stamp for the match:

```
rm -f apps.txt
echo -e "${label_color}Find all existing deployments:${no_color}"
cf apps | { grep $PUBLIC_HOST || true; } | sed -r "s/\x1B\[([0-9]{1,2}(;[0-9]{1,2})?)?[mGK]//g"
   &> apps.txt
cat apps.txt
# Store just the deployed application names
awk '{ print $1 }' apps.txt > app_names.txt
rm -f apps.txt
```

3. Push the updates from the selected build using the computed name and route:

```
echo -e "${label_color}Pushing new deployment - ${green}$NEW_APP_NAME${no_color}"
cf push $NEW_APP_NAME -p $WAR -d $DOMAIN -n $TEST_HOST -m $MEMORY
DEPLOY_RESULT=$?
if [ $DEPLOY_RESULT -ne 0 ]; then
   echo -e "${red}Deployment of $NEW_APP_NAME failed!!"
   cf logs $NEW_APP_NAME --recent
   return $DEPLOY_RESULT
fi
echo -e "${label_color}APPS:${no_color}"
cf apps | grep ${CF_APP}
```

   I've used a basic `push` command. When you apply this technique to future applications, you can adjust the `push` command to suit your needs — including the creation of services as required.

4. Test to make sure that the new application is functioning. Here, I'm using a basic cURL command to make sure that the application is responding. You would likely want more testing at this point to determine if the new application is ready to take on load:

```
echo -e "${label_color}Testing new app - ${green}$NEW_APP_NAME${no_color}"
curl "http://$TEST_HOST.$DOMAIN"
TEST_RESULT=$?
if [ $TEST_RESULT -ne 0 ]; then
   echo -e "${red}New app did not deploy properly - ${green}$NEW_APP_NAME${no_color}"
   return $TEST_RESULT
else
   echo -e "${green}Test PASSED!!!!${no_color}"
fi
```

5. Map traffic to the new application by binding both the existing and new application to the public route. Bluemix automatically routes traffic to both instances:

```
echo -e "${label_color}Map public space route to new app - ${green}$NEW_APP_NAME${no_color}"
cf map-route $NEW_APP_NAME $DOMAIN -n $PUBLIC_HOST

echo -e "${label_color}Public route bindings:${no_color}"
cf routes | { grep $PUBLIC_HOST || true; }
```

6. Delete the temporary test route, which is no longer needed. This step isn't absolutely necessary, but it does help to clean up routes that are no longer being used:

```
echo -e "${label_color}Remove and delete test route - ${green}$TEST_HOST${no_color}"
cf unmap-route $NEW_APP_NAME $DOMAIN -n $TEST_HOST
if [ $? -ne 0 ]; then
   echo -e "${label_color}Test route isn't mapped and doesn't need to be removed.${no_color}"
fi
cf delete-route $DOMAIN -n $TEST_HOST -f
```

7. Delete the old application version or versions now, leaving only the new version to take traffic on the public route:

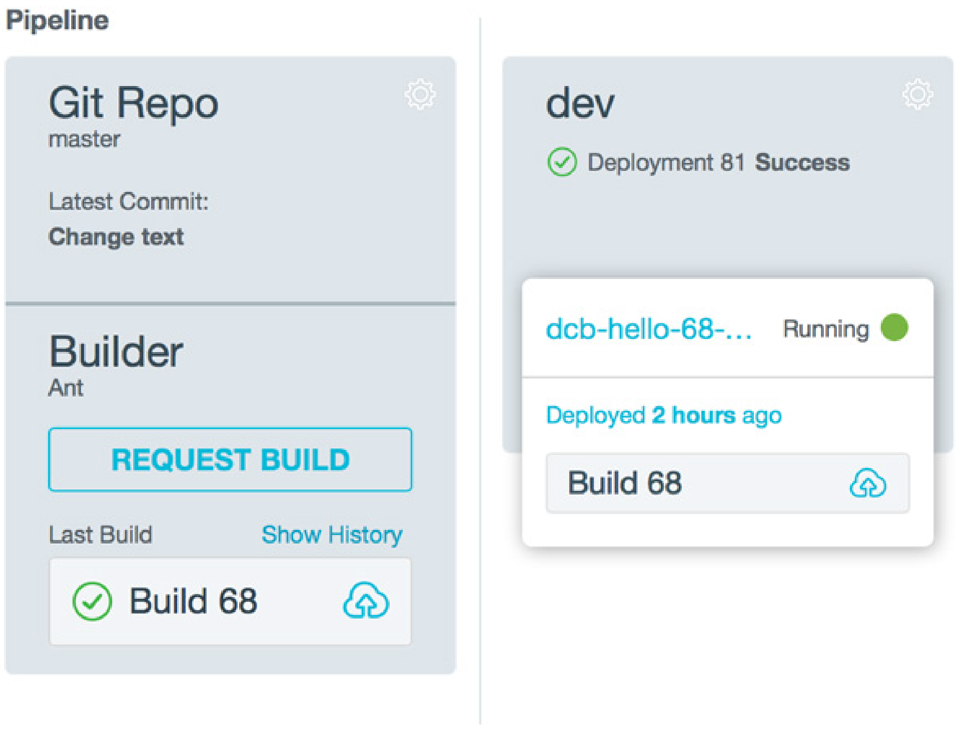
```
while read name
do
   if [ "$name" != "$NEW_APP_NAME" ]; then
       echo -e "${label_color}Deleting old deployed application - $name${no_color}"
      cf delete $name -f
   fi
done < app_names.txt

rm -f app_names.txt
```

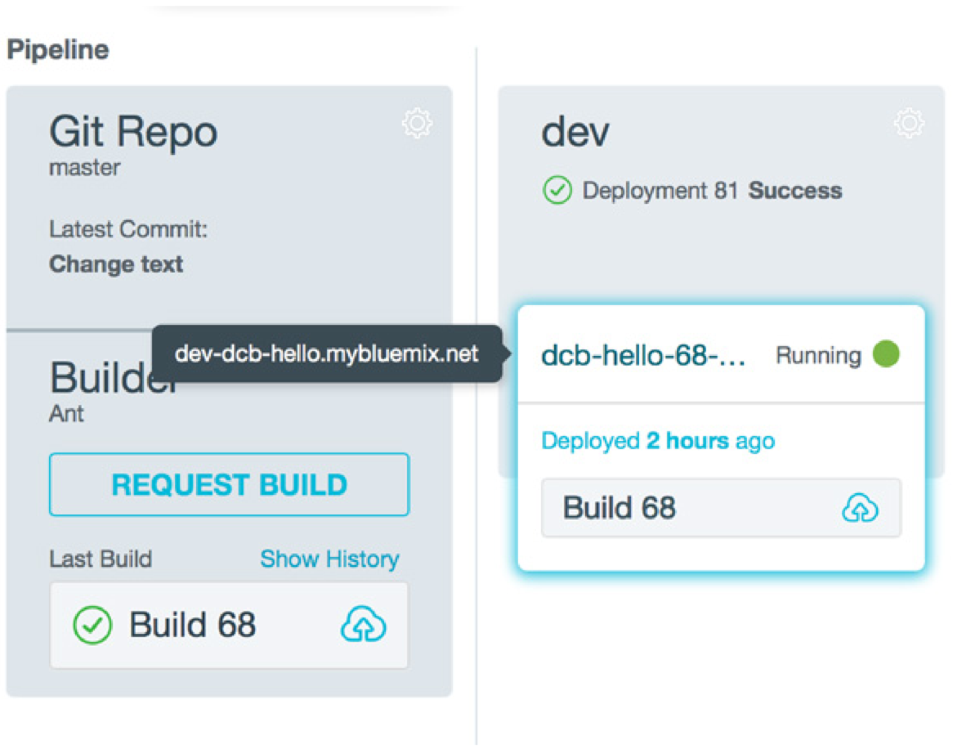
Step 5: Review the logs of a deployment request

Now that the pipeline is set up, you can request a new build and have it deployed:

1. Click the **REQUEST BUILD** button to build your project and automatically trigger a deployment.
2. Request another build by clicking the **REQUEST BUILD** button. In this example, Build 68 is requested for deployment into the dev space after the build. Your build number will be different, but the flow is the same:



3. From the hover (hold your mouse pointer over the name of the running app), you can see the route used to access the application within the dev space. Click the deployment request (Deployment 81 in this example) to see the logs for the deployment:



4. You can see the assigned variables and the existing deployed application that was deployed from Build 68. Notice that the computed `NEW_APP_NAME` has as a suffix the build number and a time stamp:

```
Variables:
PUBLIC_HOST=dev-dcb-hello
DOMAIN=mybluemix.net
NEW_APP_NAME=dcb-hello-68-1408373927
TEST_HOST=dcb-hello-68-1408373927
WAR=Pipeline_hello1.war
MEMORY=256M
Find all existing deployments:
dcb-hello-67-1408373457   started          1/1         256M     1G     dev-dcb-hello.mybluemix.net
```

5. The contents of Build 68 are pushed to Bluemix using the computed name:

```
Pushing new deployment - dcb-hello-68-1408373927
Creating app dcb-hello-68-1408373927 in org danberg@us.ibm.com / space dev as danberg@us.ibm.com...
OK

Creating route dcb-hello-68-1408373927.mybluemix.net...
OK

Binding dcb-hello-68-1408373927.mybluemix.net to dcb-hello-68-1408373927...
OK

Uploading dcb-hello-68-1408373927...
Uploading from: /home/jenkins/workspace/b3a8fa2f-27f8-5b88-4007-76445be089ce/
   0b7a50d5-2d8f-4dac-ac7b-c5b9aa3e6f27/out/Pipeline_hello1.war
2.5K, 3 files
OK

Starting app dcb-hello-68-1408373927 in org danberg@us.ibm.com / space dev as danberg@us.ibm.com...
OK
-----> Downloaded app package (4.0K)
-----> Uploading droplet (104M)

0 of 1 instances running, 1 starting
```

```
0 of 1 instances running, 1 starting
0 of 1 instances running, 1 starting
0 of 1 instances running, 1 starting
0 of 1 instances running, 1 starting
0 of 1 instances running, 1 starting
1 of 1 instances running

App started
```

You can see that both the new application and the old application are deployed but on different routes. The older Build 67 is using the public route, and the new deployment is on a private route:

```
APPS:
dcb-hello-67-1408373457   started           1/1         256M     1G     dev-dcb-hello.mybluemix.net
dcb-hello-68-1408373927   started           1/1         256M     1G     dcb-
hello-68-1408373927.mybluemix.net
```

6. Now you can test the new application from Build 68 by using the test route:

```
Testing new app - dcb-hello-68-1408373927
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100    46    0    46    0     0     76      0 --:--:-- --:--:-- --:--:--    89
100    46    0    46    0     0     76      0 --:--:-- --:--:-- --:--:--    89
Rolling deployments made easy!!!0.0.0.0:63645
Test PASSED!!!!
```

7. With a successful test, you bind the new application to the public host (dev-dcb-hello.mybluemix.net). Notice that both the old and new application are now bound to the public route:

```
Map public space route to new app - dcb-hello-68-1408373927
Creating route dev-dcb-hello.mybluemix.net for org danberg@us.ibm.com / space dev as
 danberg@us.ibm.com...
OK
Route dev-dcb-hello.mybluemix.net already exists
Adding route dev-dcb-hello.mybluemix.net to app dcb-hello-68-1408373927 in org danberg@us.ibm.com /
   space dev as danberg@us.ibm.com...
OK
Public route bindings:
dev-dcb-hello                                                  mybluemix.net   dcb-hello-68-1408373927,
 dcb-hello-67-1408373457
```

8. Delete the test route:

```
Remove and delete test route - dcb-hello-68-1408373927
Removing route dcb-hello-68-1408373927.mybluemix.net from app dcb-hello-68-1408373927 in org
   danberg@us.ibm.com / space dev as danberg@us.ibm.com...
OK
Deleting route dcb-hello-68-1408373927.mybluemix.net...
OK
```

9. Finally, delete the old application, leaving only the new application to take customer traffic:

```
Deleting old deployed application - dcb-hello-67-1408373457
Deleting app dcb-hello-67-1408373457 in org danberg@us.ibm.com / space dev as danberg@us.ibm.com...
OK
Sending deployment success of dcb-hello-67-1408373457 to IBM DevOps Services...
IBM DevOps Services notified successfully.
Deployed Applications:
dcb-hello-68-1408373927   started           1/1         256M     1G     dev-dcb-hello.mybluemix.net
Public route bindings:
dev-dcb-hello                           mybluemix.net   dcb-hello-68-1408373927
You have successfully executed a rolling deployment of dcb-hello-68-1408373927.
```

Conclusion

Rolling deployments are a necessity for any production-level application to avoid disruptions in service and keep users happy. You can execute a rolling release of an application manually within Bluemix, but doing so would be tedious and error-prone. It is best to automate your rolling deployments to achieve highly repeatable and reliable deployments.

I've shown one approach for achieving automated rolling deployments with the Delivery Pipeline service from DevOps Services. I used a trivial application to focus on the rolling deployment script; in reality, you'd also need to automate the setup (unique or sharing) of services.

With the information from this tutorial and the example, you should be able to apply a rolling deployment to any of your Bluemix applications. Good luck with your own rolling deployment and, as always, post any questions that you might have to our forum.

The Delivery Pipeline servicehttp://www.ibm.com/developerworks/topics/delivery pipeline serviceautomates builds and deployments, as well as test execution and build script configuration.

RELATED TOPICS: DevOps bash

About the author

Daniel Berg

Follow me on G+