Automatize implementações contínuas com o IBM DevOps Services for Bluemix

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

05/Dez/2014

Os desenvolvedores que desenvolvem aplicativos no IBM Bluemix se esforçam para atingir o tempo de inatividade zero para evitar interrupções no serviço e manter os usuários felizes. Atingir o tempo de inatividade zero enquanto entrega novas funções é desafiador e exige a capacidade de executar uma implementação contínua. Este tutorial irá guiá-lo pela configuração e uso do serviço Delivery Pipeline do IBM DevOps Services for Bluemix para automatizar um processo de implementação contínua.

A melhor maneira atrair novos usuários e manter os existentes é criar aplicativos atraentes que nunca deixam de funcionar. Os consumidores têm pouca tolerância para sites que têm indisponibilidades (planejadas ou não). E conforme sua equipe aumenta a velocidade em que entrega alterações, torna-se ainda mais importante evitar interrupções no serviço.

Implementações contínuas são um processo pelo qual uma nova versão de um aplicativo é implementada em um ambiente sem interrupção no serviço para o consumidor. Você*obrigatório* executar implementações contínuas se deseja aumentar sua base de usuários.

IBM® Bluemix™ fornece uma plataforma simples de usar que permite as implementações contínuas. Mostrarei como é possível automatizar facilmente as implementações contínuas para espaços do Bluemix usando o IBM DevOps Services e o serviço Delivery Pipeline.

O que você precisará?

- Contas do Bluemix e do IBM DevOps Services.
- Entendimento básico dos recursos do IBM Delivery Service, por exemplo, como criar um Builder e um Deployer padrão. Se necessário, revise o tutorial Getting Started with IBM Bluemix and DevOps Services Using Java .
- Conhecimento em script Bash.

Ver o pipeline
Obter o código de exemplo

> *“ As implementações contínuas são uma necessidade para qualquer aplicativo de nível de produção para evitar interrupções em um serviço. ”*

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

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

Eu configurei um simples exemplo de Hello World com um pipeline de implementação contínua automatizada. É possível examinar o código durante a leitura. O pipeline está disponível publicamente, portanto é possível revisar os logs de implementação e as configurações de automação da implementação.

> As implementações contínuas frequentemente são chamadas de implementações *azul-verde* , implementações *vermelho-preto* ou implementações com *tempo de inatividade zero* . Embora esses processos de implementação sejam diferentes de formas sutis, eles compartilham o princípio central de evitar a interrupção no serviço para o consumidor.

Irei guiá-lo pelas etapas que eu usei para estabelecer uma implementação contínua automatizada no DevOps Services. Use essas etapas apenas como um exemplo, ajustando as etapas e os scripts para se adequarem às suas necessidades — ou para aperfeiçoar com base nas minhas habilidades de script Bash rudimentares.

Etapa 1: Bifurque o projeto de amostra

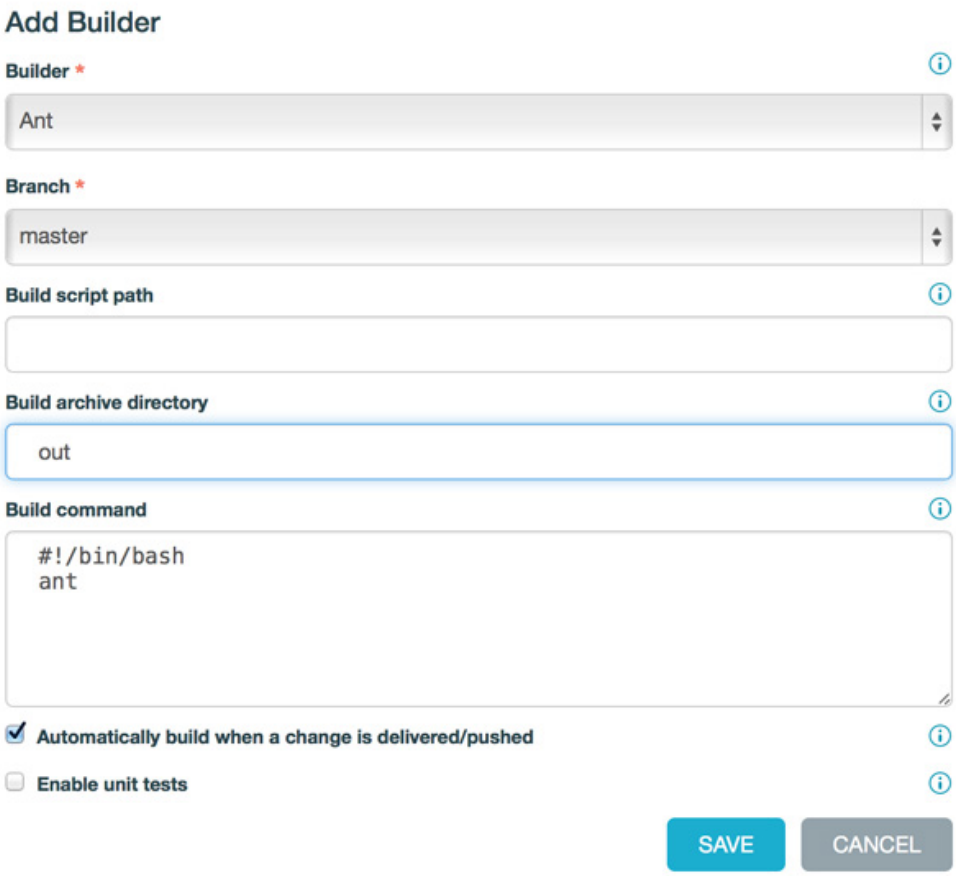
Comece bifurcando o projeto de amostra para que você tenha seu próprio projeto do DevOps Services com a fonte necessária:

1. Role para cima e clique no botão desse tutorial chamado **Obter o código de exemplo** .
2. Na página de visão geral do projeto danberg | Pipeline_hello, clique no botão **EDIT CODE** (insira suas credenciais do DevOps Services se ainda não estiver logado) e clique no botão **FORK** no menu.
3. Insira um nome para seu novo projeto, marque a caixa de seleção **Deploy to Bluemix** e selecione uma organização e um espaço para fins de faturamento.
4. Clique em **Save** para criar o novo projeto com o código-fonte.

Etapa 2: Inclua um novo estágio de desenvolvimento

Após o projeto ser criado, comece a configurar o pipeline de entrega incluindo um estágio de desenvolvimento no pipeline:

1. Clique na guia **BUILD & DEPLOY** na parte superior da página.
2. Nenhum processo de entrega está ativado por padrão. Clique na guia **ADVANCED** (botão) para ativar o recurso de pipeline de entrega avançado.
3. Clique na guia **add a builder** (botão) para configurar um construtor para o projeto. Mantenha os padrões (construtor Ant) e insira `out` na lista **Build archive directory** (campo):



4. Clique em **SAVE** para salvar as configurações do construtor.
5. Clique na guia **add a stage** (botão) para configurar um implementador. Clique em **SAVE** para salvar as configurações padrão. Você ajustará o implementador para os releases contínuos nas etapas subsequentes.

Etapa 3: Ajuste o pipeline de entrega para usar um script de implementação

Pelo fato de as implementações contínuas serem um pouco mais complexas que os outros tipos de implementação, é desejável gerenciar o script de implementação no seu SCM. É necessário ajustar seu construtor e implementador para usarem o script de implementação:

1. Clique no botão do seu projeto chamado **EDIT CODE** .
2. Selecione o arquivo build.xml e observe que ele inclui o seguinte fragmento para copiar o arquivo deploy.sh — o script usado para as implementações contínuas — para o diretório de archive. Esse código é necessário para garantir que o arquivo deploy.sh esteja disponível durante o estágio de implementação:

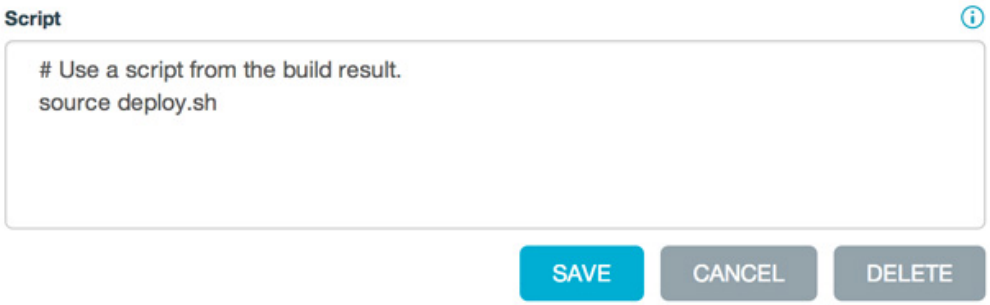
```
<echo message="Copy files to ${artifact_dir} for deployment"/>
<copy todir="${artifact_dir}">
    <fileset file="manifest.yml" />
    <fileset file="deploy*.sh" />
</copy>
```

3. Abra o pipeline clicando no botão **BUILD & DEPLOY** .

4. Clique no ícone de configuração no canto superior direito do Deployer para editar a configuração.
5. Defina o parâmetro **Nome do aplicativo** para um valor que tenha significado para seu projeto. Observe que esse nome será usado para calcular um novo nome do aplicativo durante o processo de implementação.
6. Substitua o conteúdo da seção Script por:

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

Usar `source` no corpo do script é a maneira correta de chamar um script Bash que está incluído nos seus artefatos de desenvolvimento:



7. Clique em **SAVE**.

Etapa 4: Examine o script da implementação contínua

Acompanhe no arquivo de script deploy.sh de exemplo conforme eu descrevo as principais seções necessárias para automatizar uma implementação contínua com um script Bash. (Não é necessário fazer nenhuma alteração nessa etapa.)

1. Na parte superior do script, defina as variáveis a serem usadas posteriormente:

```
#############
# 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` é importante porque mostra como eu calculo um novo nome do aplicativo para cada implementação com base no número do desenvolvimento e no registro de data e hora. As variáveis de cor são usadas para tornar os arquivos de log mais fáceis de ler. O nome do aplicativo especificado na configuração do Deployer é reverenciado no script com a variável de ambiente `CF_APP` e o espaço de destino com a variável `CF_SPACE` .

2. Localize os nomes dos aplicativos implementados existentes no espaço (a serem excluídos posteriormente). Eu procuro por todos os aplicativos no espaço que estão associados a `PUBLIC_HOST`, ignorando o número do desenvolvimento e o registro de data e hora calculados para a correspondência:

```
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. Realize o push das atualizações do desenvolvimento selecionado usando a rota e o nome calculados:

```
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}
```

Eu usei um comando básico `push` . Quando aplicar essa técnica em futuros aplicativos, é possível ajustar o comando `push` para se adequar às suas necessidades — incluindo a criação de serviços conforme o necessário.

4. Teste para ter certeza de que o novo aplicativo está funcionando. Aqui, estou usando um comando cURL básico para ter certeza de que o aplicativo está respondendo. Provavelmente você deseja mais testes nesse ponto para determinar se o aplicativo está pronto para aceitar carga:

```
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. Mapeie o tráfego para o novo aplicativo ligando o aplicativo novo e o existente à rota pública. O Bluemix realiza o roteamento do tráfego para as duas instâncias automaticamente:

```
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. Exclua a rota de teste temporária, que não é mais necessária. Essa etapa não é absolutamente necessária, mas ajuda a limpar as rotas que não estão mais sendo usadas:

```
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. Exclua a versão, ou versões, antiga do aplicativo agora, deixando apenas a nova versão para receber o tráfego na rota pública:

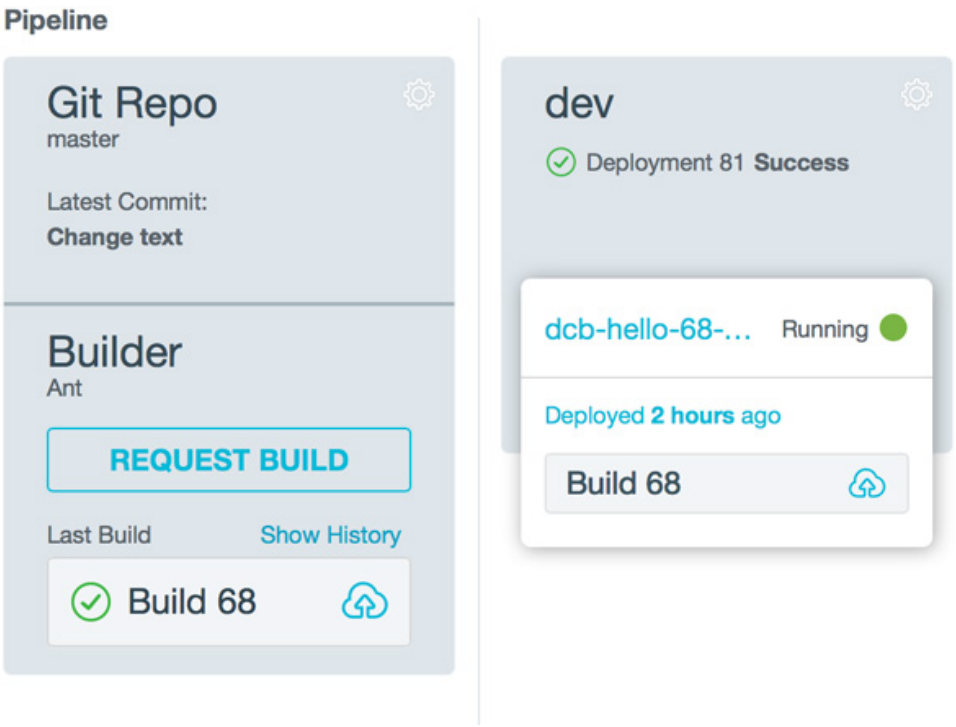
```
while read name
do
   if [ "$name" != "$NEW_APP_NAME" ]; then
       echo -e "${label_color}Deleting old deployed application - $name${no_color}"
      cf delete $name -f
   fi
done < app_names.txt

rm -f app_names.txt
```

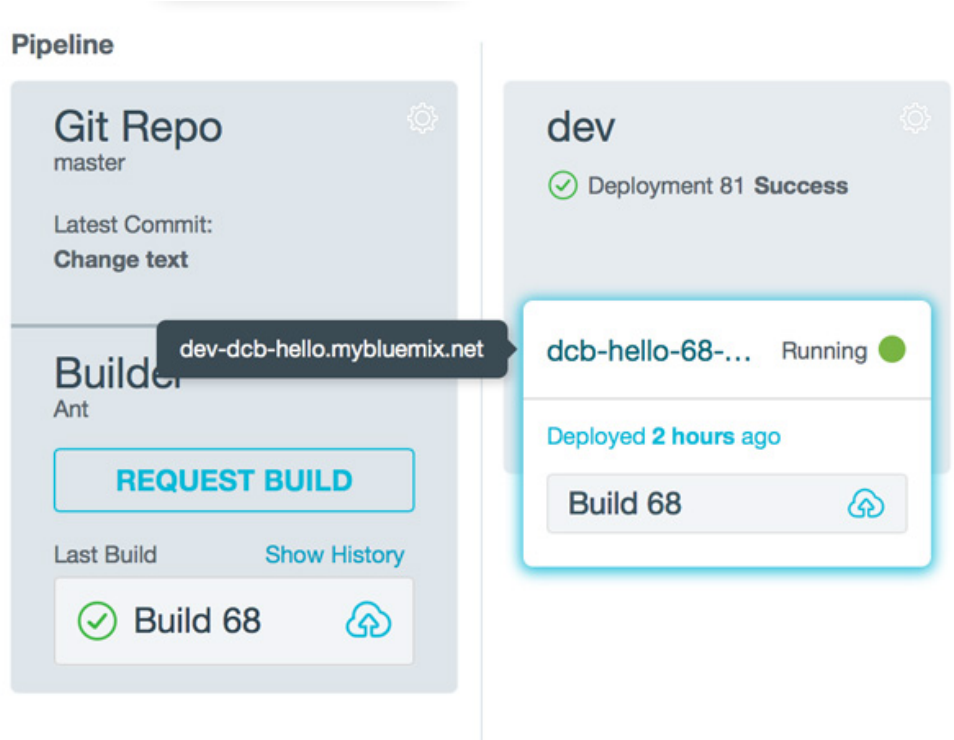
Etapa 5: Revise os logs de uma solicitação de implementação

Agora que o pipeline está configurado, é possível solicitar um novo desenvolvimento e tê-lo implementado:

1. Clique na guia **REQUEST BUILD** (botão) para desenvolver seu projeto e automaticamente acionar uma implementação.
2. Solicite outro desenvolvimento clicando no botão **REQUEST BUILD** . Nesse exemplo, a Build 68 é solicitada para implementação no espaço dev após o desenvolvimento. Seu número de desenvolvimento será diferente, mas o fluxo é o mesmo:



3. Na ajuda instantânea (mantenha o ponteiro do mouse sobre o nome do aplicativo em execução), é possível ver a rota usada para acessar o aplicativo no espaço dev. Clique na solicitação de implementação (Deployment 81 nesse exemplo) para ver os logs da implementação:



4. É possível ver as variáveis designadas e o aplicativo implementado existente que foi implementado a partir da Build 68. Observe que o `NEW_APP_NAME` calculado possui como sufixo o número de desenvolvimento e um registro de data e hora:

```
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. O push dos conteúdos da Build 68 para o Bluemix é realizado usando o nome calculado:

```
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
```

É possível ver que o aplicativo novo e o antigo estão implementados, mas em rotas diferentes. A Build 67 mais antiga está usando a rota pública e a nova implementação está em uma rota privada:

```
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. Agora é possível testar o novo aplicativo da Build 68 usando a rota de teste:

```
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. Com um teste bem-sucedido, você liga o novo aplicativo ao host público (dev-dcb-hello.mybluemix.net). Observe que agora o aplicativo novo e o antigo estão ligados à rota pública:

```
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. Exclua a rota de teste:

```
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. Por fim, exclua o aplicativo antigo, deixando apenas o novo aplicativo para receber o tráfego do cliente:

```
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.
```

Conclusão

As implementações contínuas são uma necessidade para qualquer aplicativo de nível de produção para evitar interrupções em um serviço e manter os usuários felizes. É possível executar um release contínuo de um aplicativo manualmente no Bluemix, mas fazer isso seria enfadonho e propenso a erros. É melhor automatizar suas implementações contínuas para obter implementações altamente confiáveis e repetíveis.

Eu mostrei uma abordagem para obter implementações contínuas automatizadas com o serviço Delivery Pipeline do DevOps Services. Eu usei um aplicativo trivial para focar no script da implementação contínua. Na realidade também seria necessário automatizar a configuração (exclusiva ou compartilhada) dos serviços.

Com as informações desse tutorial e com o exemplo, você deve ser capaz de aplicar uma implementação contínua a qualquer um dos seus aplicativos do Bluemix. Boa sorte com sua própria implementação contínua e, como sempre, poste qualquer dúvida que possa ter em nosso fórum.

Temas relacionados: DevOps bash

Sobre o autor

Daniel Berg

Siga-me no G+