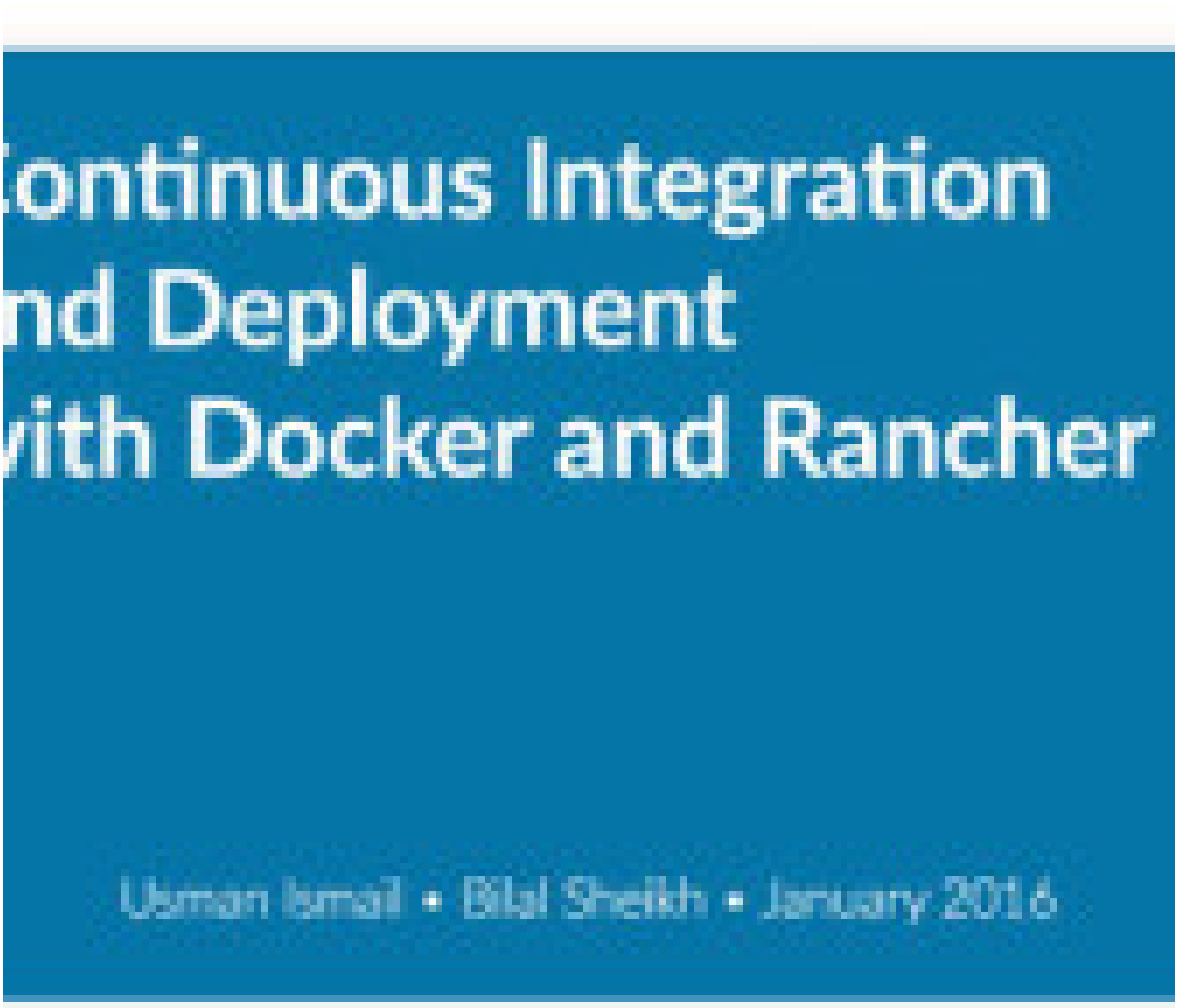
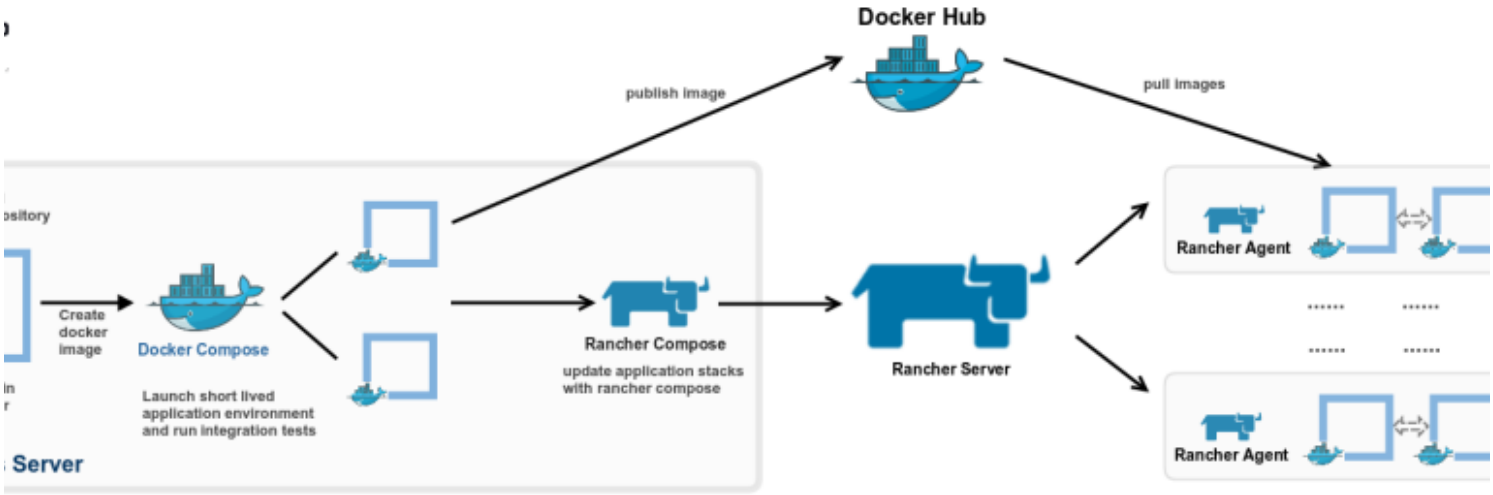
Docker-Based Build Pipelines (Part 2) - Continuous Deployment

| JANUARY 12, 2016



seen how to setup a Jenkins CI system on top of docker and leverage docker in order to create a continuous integration pipeline. As part of that we used docker to create a c
lled out to any number of machines. We then setup the environment in Jenkins CI and automated the continuous building, packaging and testing of the source.

pipeline further (shown below) and see how we can continuously deploy the project to a long-running testing environment. This will allow manual human testing of the code
ironment will also allow you to get your customers' or QA's eyes on the latest changes before they hit production. Further, this will give you a good idea of how to build and d
cover in the next article. You can download the entire series in our eBook \"Continuous Integration and Deployment with Docker and Rancher.\"

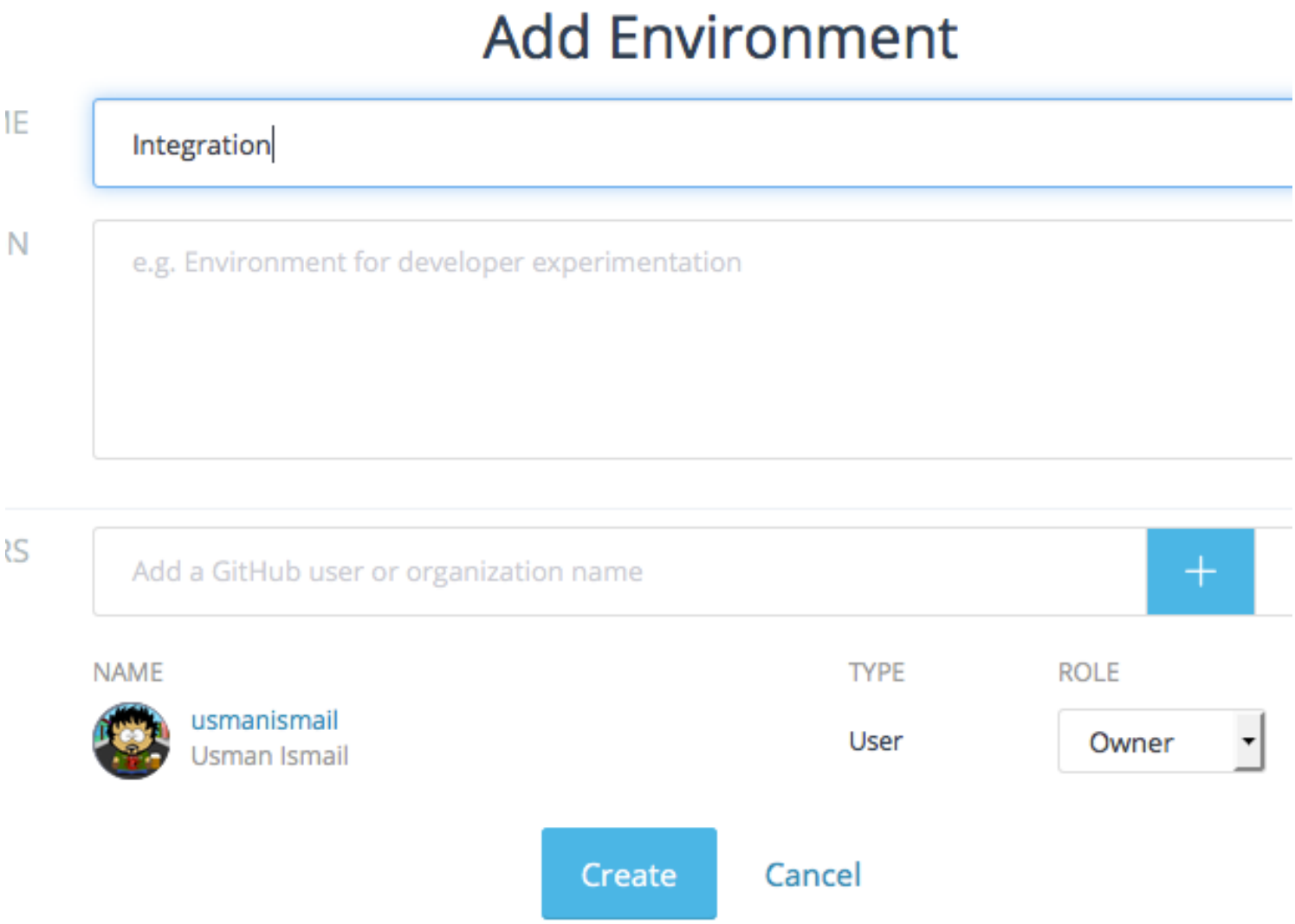


ng running application environments with Docker and Rancher

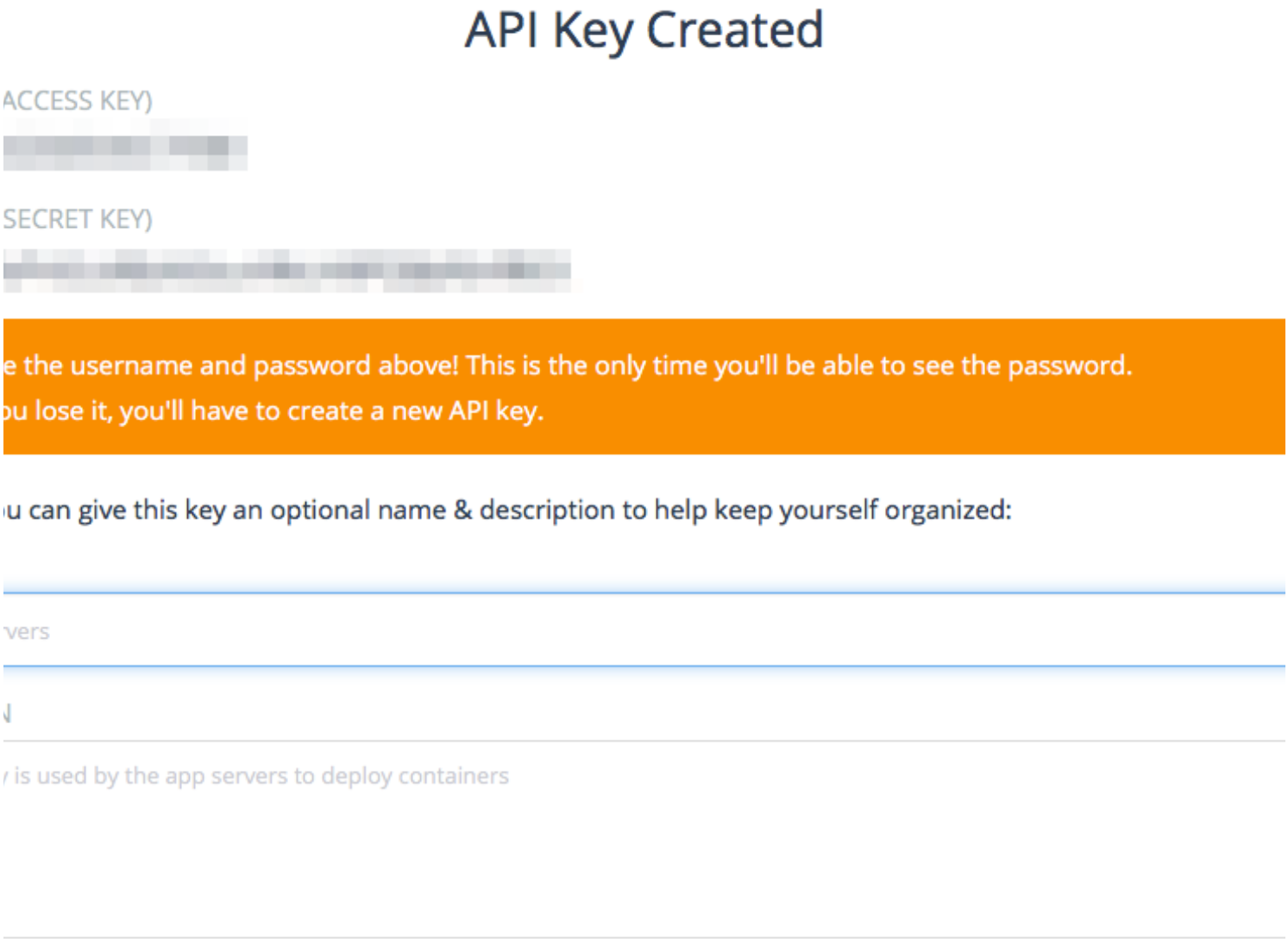
ur application, we can now deploy it to a long running, potentially externally facing environment. This environment will allow Quality Assurance (QA) or customers to see anc
production. This environment is an important step on the road to production as it allows us to unearth bugs that are only seen with real-world use and not automated integ
or Integration environment. Like our previous article, we'll be using the go-auth component of our go-messenger project to demonstrate how to create a test environment. W
ation environment:

ation environment in Rancher

op right corner and select *Manage Environments* and *Add Environment*. In the resulting screen (shown below) add the Name (Integration) and optionally a description for eac
sers and organizations that have access to the environments.



ıent setup, select the *Integration* environment from the drop down in the top left corner of the screen. We can now create the application stack for the integration environme
I & Keys and *Add API Key*. This will load a pop-up screen which allows you to create a named API Key pair. We need the key in subsequent steps to to use Rancher Compose
key pair named *JenkinsKey* to run rancher compose from our Jenkins instance. Copy the key and secret for use later as you will not be shown these values again. Note that
vill have to create a new key for each environment.



ompose and Rancher Compose templates

eated a docker compose template to define the container types required for our project. The compose template (docker-compose.yml) is shown below. We will be using the he addition of auth-lb service. This will add a load-balancer in front of our go-auth service and split traffic across all the containers running the service. Having a load balanc y and scalability, as it continues to serve traffic even if one (or more) of our service containers die. Additionally, it also spreads the load on multiple containers which may be

```
ORD: rootpass
messenger
enger
messenger
```

```
uth:${auth_version}
```

```
sql-master
```

```
ad-balancer-service

th-service
```

ose to launch the environment in a multi-host environment, this more closely mirrors production and also allows us to test integration with various services, e.g. Rancher an
sed environment which was explicitly designed to be independent of external services and launched on the CI server itself without pushing images to dockerhub.]

Rancher compose to launch a multi-host test environment instead of docker compose, we also need to define a rancher compose template. Create a a file called *rancher-c*
we are defining that we need two containers of the auth service, one container running the database and another running the load-balancer container.

ck to the auth-service to make sure that we detect when containers are up and able to respond to requests. For this we will use the /health URI of the go-auth service. The a
now look something like this:

```
hold: 3
ET /health HTTP/1.0
ld: 2
t: 2000
```

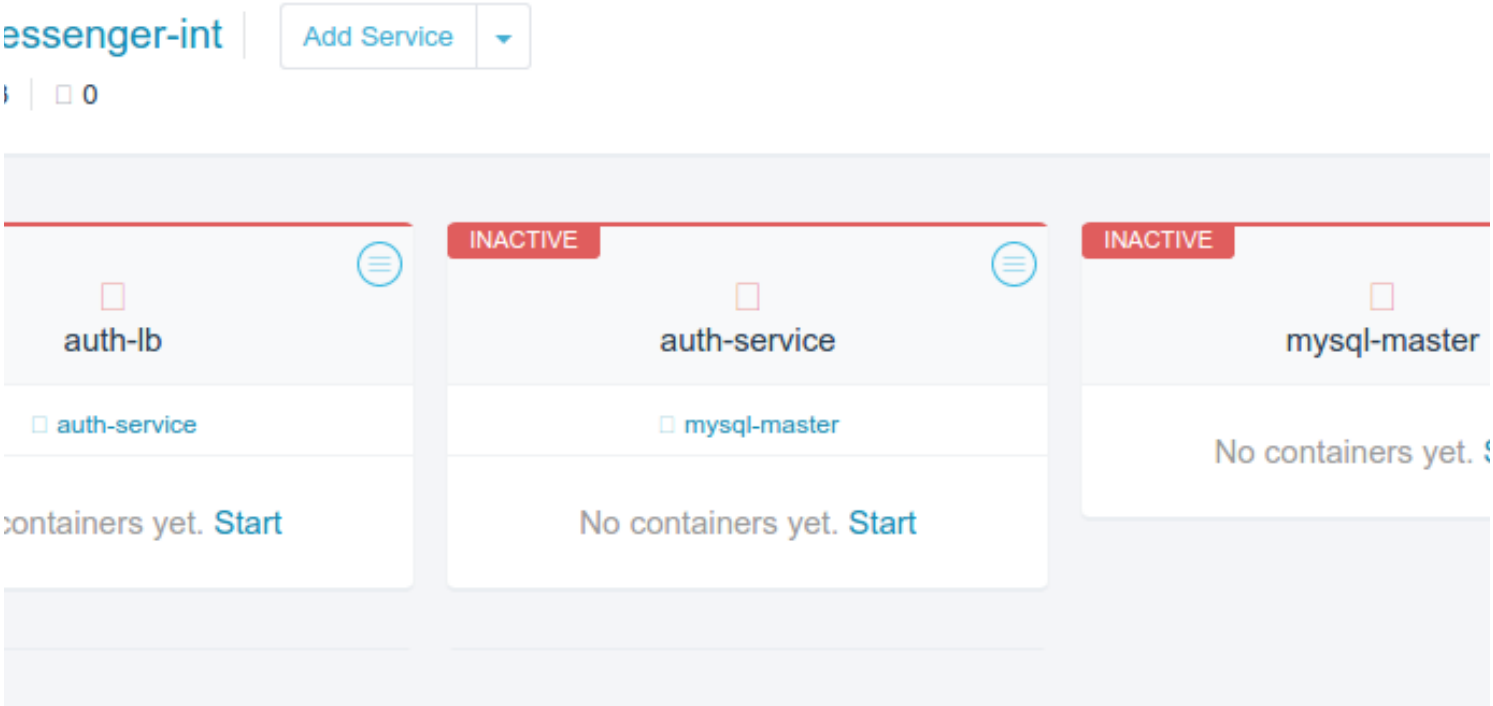
k on port 9000 of the service container which is run every 2 seconds (2000 milliseconds). The check makes a http request to the /health URI and 3 consecutive failed check
tive successes mark a container as healthy.

on stack with Rancher Compose

te defined we can use Rancher compose to launch our environment. To follow along, simply checkout the go-messenger project and download the rancher-compose CLI fro
velopment machine, follow the instructions here. Once you have rancher-compose setup you can use the create command shown below to setup your integration environme

```
ithub.com/usmanismail/go-messenger.git
loy

mpose with the latest version you downloaded from rancher UI
--project-name messenger-int \
UR_RANCHER_SERVER:PORT/v1/   \
PI_KEY>                      \
ECRET_KEY>                   \
e
```

able to see the stack and services for your project. Note that \“create\” command only creates the stack and doesn’t start services. You can either start the services from th
start all the services.



gain to start the services:

```
--project-name messenger-int \
UR_RANCHER_SERVER:PORT/v1/   \
PI_KEY>                      \
ECRET_KEY>                   \
```

orking, head over to the public IP for the host running the \"auth-lb\" service and create a user using the command shown below. You should get a 200 OK. Repeating the ab
t with an existing user in the database. At this point we have a basic integration environment for our application which is intended to be a long running environment.

```
PUT -d userid=<TEST_USERNAME> -d password=<TEST_PASS> <HOST_IP_FOR_AUTH_LB>:9000/user
```

ords with Rancher and AWS Route53

ant to be long running and externally facing, we are going to be using DNS entries and HTTPS. This allows us to distribute the application outside corporate firewalls securel
stent DNS rather than IPs which may change. You may use a DNS provider of your choice, however, we are going to illustrate how to setup DNS entries in Amazon Route53. T
d *Create Hosted Zone*. In the hosted zone you will have to specify a domain name of your choice e.g. gomessegner.*com*. While you are in the AWS console you can also crea
o to AWS Console > IAMS > Users and select *Create New Users*. Keep the the Access Key and Secret Key of this user handy as you will need a little later on. Once you have c
3FullAccess policy to the user so that it can make updates to route53.

d Zone and IAMs user setup we can add the Route53 integration to our Rancher Server. The detailed instructions on how to do so can be found here. In short you need to bro
er and select Route 53 DNS. You will be asked to specify the Hosted Zone that you setup earlier as well as the AWS Access and Secert Keys for you Rancher IAMs user with
n and click create, you should see new stack created in your environment with a service called route53.

Launch

Route53 DNS

Rancher External DNS service powered by Amazon Route53

Launch

ncher events and catch any load balancer instance launches and terminations. Using this information it will automatically create DNS entries for all the Hosts on which your
of the form [Loadbalancer].[stack].[environment].[domain], e.g. goauth.integration.testing.gomessenger.com. As more containers are launched and taken down on your vari
your DNS records consistent. This is essential for our integration test environments because as we will see later we need to relaunch the environment containers in order to
Route53 DNS integration we do not have to worry about getting the latest hostnames to our clients and testers.

HTTPS

ds for our environment it is a good idea to support HTTPS. To do that, first, we need an SSL certificate for our domain. You can purchase a root SSL certificate for your doma
such as Comodo. If you don't have a certificate you can generate a self-signed certificate to complete the setup and replace it with a trusted one at a later time. The implica
ll get a \"This connection is untusted\" warnings in browsers, however, the communication is still encrypted. In order to generate the self-signed certificate you will first neec
enrsa command of openssl. Then you can use the key file to generate the certificate using the req command. The steps to do so are listed below. Its also a good idea to prin
so that you can manually ensure that the same certificate is presented to you when making HTTPS requests. In the absence of a trusted certificate manually matching finger
nan-in-the-middle attacks.

```
 integration.gomessenger.com.key 2048
509                    \
n.gomessenger.com.key    \
n.gomessenger.com.crt   \
 /CN=integration.gomessenger.com
rprint -sha256 -in integration.gomessenger.com.crt
E2:E5:86:09:F0:91:F4:3C:C2:DE:D1:40:9C:DD:AF:A2:0A:88:EE:19:0C:C5:A6:03:C9:9B:17:6E:8F:58:D2:C3
```

cate and the private key file we need to upload these into Rancher. We can upload certs by clicking the *Add Certificate* button in the *Certificates* Section of the *Infrastructure*
name for your certificate and optionally a description as well. Copy the contents of *integration.gomessenger.com.key* and *integration.gomessenger.com.crt *into the *Priva*
Read from File and select the respective files). Once you have completed the form click save and wait a few moments for the certificate to become active.

Add Certificate

ME

integration.gomessenger.com_selfsigned

ON

The certificate for the Integration test environment

Y*

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAm9eneN1Efhmq7ftLZMEeGK0OcNbmQ0QoeMj1+FxW3vHU5ZVD
uICN4B4u4+yA/uCmMnaUstGCQIM28xrzYXB7NF1nPyBsds6cF44oZKwKBu820n4/
oESkT45gGDTDq/4iDI9B6HFUWoXm3v8L1m+H0ByKARjrHs+xrVtg8hYF/M5n1VAs
jgMa9x+lTvrAf6RTz4w6dFbaGVKt6Ece519P4Uw2GxqIqv+UqpNgTVh1kgZLeDBH

E*

lqRahE2nadw0ftrFeUrULTOjnku/sk9ja4M02uOWr4gDZem014rAn7MPQ1brj8M+
TpLjalpun2Hru6RBG+yR+eNH9PJmMJq78isy0FvYNXSa7eDiw1OJgl2dW59nnqut
f8+RDPrrWjDTuvobPy3lsDpCWQFkUyfw9JKm/VQ+TDsR61TVd/k=
-----END CERTIFICATE-----

TS

Optional; Paste in the additional chained certificates, starting with -----BEGIN CERTIFICATE-----

Save Cancel

e we can add the HTTPS endpoint to our environment. In order to do so we have to modify our docker-compose file to include the SSL port configuration. We add a second po
outside the load balancer container and we use the *io.rancher.loadbalancer.ssl.ports* label to specify that '9001' will be the public load balancer port with SSL termination. F
alancer we can route requests to our actual service container using plain HTTP over the original 9000 port. We specify this mapping from 9001 to 9000 using the *io.rancher.*

```
ancer.ssl.ports: '9001'
ancer.target.auth-service: 9000=9000,9001=9000

d-balancer-service

h-service
```

ancher-compose file to specify the SSL certificate we should use in the load balancer service for SSL termination. Add the *default_cert* parameter with the name of the certif
need to delete and recreate your stack as there is currently no way to add these properties to a deployed stack.

```
egration.gomessenger.com_selfsigned
fig:
onfig
```

is working, you can use the following curl command. When you try the same command with the https protocol specifier and the 9001 port you should see a failure complain
use the *--insecure* switch to turn of trusted certificate checking and use https without it.

```
PUT            \
_USERNAME>     \
ST_PASS>       \
ion.gomessenger.com:9000/user

h secure checking
900(1)
PUT            \
ERNAME>        \
PASS>          \
n.gomessenger.com:9001/user
ificate problem, verify that the CA cert is OK. Details: error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certif

h insecure checking
PUT            \
               \
ERNAME>        \
PASS>          \
n.gomessenger.com:9001/user
```

ontinuous Deployment pipelines with Rancher and Jenkins

r test environment we can finally get back to the original intent of this article and build out a continuous deployment pipeline by extending our Jenkins CI pipeline which buil
egration tests against it.

er images

hing the packaged image to a docker repository. For simplicity we are using a public DockerHub repository, however, for actual development projects you would want to pus
e a new *Free Style Project* job in Jenkins by clicking the *New Item* button and name our job *push-go-auth-image*. Once you do so, you will be taken to the Jenkins job configu
ush your go-auth image up to Dockerhub.

f the pipeline we built in our previous article, the job will have similar configuration to *go-auth-integration-test* job. The first setting you need is to make it *parameterized bui
er.

build is parameterized

String Parameter

Name GO_AUTH_VERS

Default Value develop

image we will select the *Add build step* drop down and then the *Execute shell* option. In the resulting text box add the commands shown below. In the commands we are goi
earlier. We're pushing to the *usman/go-auth* repository, however, you will need to push to your own DockerHub repository.

ticle, we're using git-flow branching model where all feature branches are merged into the 'develop' branch. To continuously deploy changes to our integration environment w
based off of develop. In our package job we tagged the docker container using the GO_AUTH_VERSION (e.g., docker build -t usman/go-auth:\${GO_AUTH_VERSION} ….). By c
s article we'll create new releases for our application and use the CI/CD pipeline to build, package, test and deploy them to our integration environment. Note that with this s
develop branch (usman/go-auth:develop) which prevents us from referencing historical builds and do rollbacks. One simple change that you can make to the pipeline is to a
e.g., usman/go-auth:develop-14.

ecify your DockerHub username, password and email. You can either use a parameterized build to specify these for each run or use the Jenkins Mask Passwords Plugin to d
on and inject them into the build. Make sure to enable 'Mask passwords (and enable global passwords)' under *Build Environment *for your job.

```
ION}
OCKERHUB_USERNAME} -p ${DOCKERHUB_PASSWORD} -e ${DOCKERHUB_EMAIL}
o-auth:${GO_AUTH_VERSION}
```

at this job is triggered after our integration test job. To do that we need to update our integration test job to trigger *parameterized build* with *current build parameters. *Thi
tion test job we will push the tested image up to Dockerhub.

uild on other projects

o build push-go-auth-image,

hen build is Stable

uild without parameters

nt build parameters

deployment job once the image is successfully pushed to DockerHub. Again, we can do that by adding a post-build action as we did for other jobs.

egration environment

her compose CLI to; stop the running environment, pull latest images from DockerHub, and restart the environment. A brief word of caution, the Updates API is under heavy d
e newer features added in the coming weeks and months so check the Documentation to see if there are updated options. Before we create a Jenkins job to achieve contin
ly.

to stop all services (auth service, load balancer and mysql), pull the latest images and start all services. This however would be less than ideal for long running environment
date our application, we're first going to stop auth-service. You can do this by using the stop command with Rancher Compose.

```
lready done so
/github.com/usmanismail/go-messenger.git
eploy

roject-name messenger-int      \
UR_RANCHER_SERVER:PORT/v1/     \
PI_KEY>                        \
ECRET_KEY>                     \
auth-service
```

unning for auth-service which you can verify by opening the stack in the Rancher UI and verifying that the status of the service is set to *Inactive.* Next, we'll tell rancher to pu
on we specify here will be substituted in our docker compose file for the auth service (*image: usman/go-auth:\${auth_version}*).

```
ion} rancher-compose --project-name messenger-int \
RANCHER_SERVER:PORT/v1/   \
KEY>                      \
ET_KEY>                   \
h-service
```

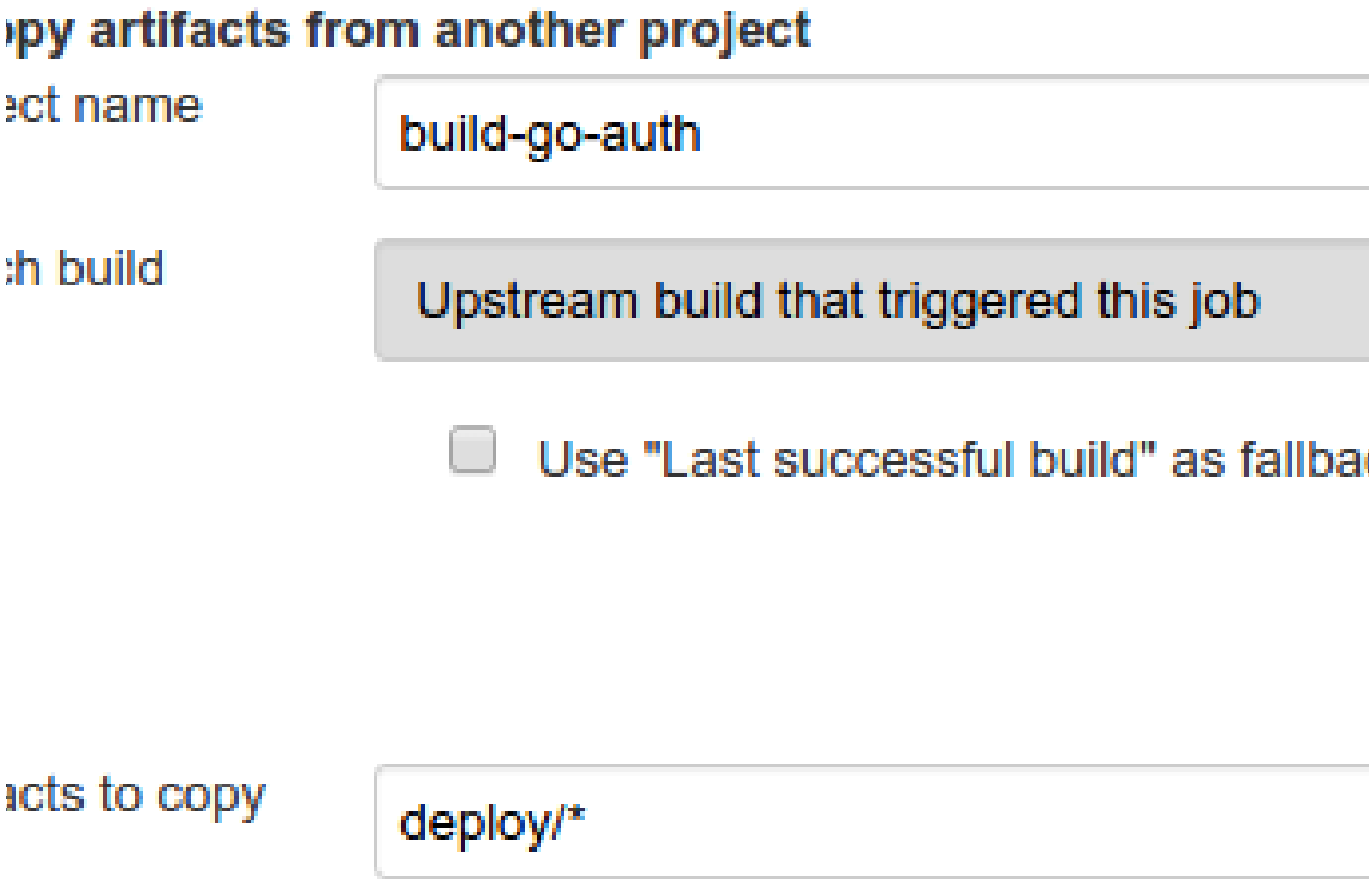
image we want, all that is needed is to start the application.

```
ion} rancher-compose --project-name messenger-int \
RANCHER_SERVER:PORT/v1/   \
KEY>                      \
ET_KEY>                   \
```

n 0.44.0, the three steps listed above can be run by a single up command using the *--force-upgrade* switch as follows:

```
ion} rancher-compose  --project-name messenger-int \
RANCHER_SERVER:PORT/v1/   \
KEY>                      \
ET_KEY>                   \
force-upgrade --pull --confirm-upgrade auth-service
```

our update lets create a Jenkins job in our pipeline to do so. As before create a new freestyle project and name it *deploy-integration. *As with all other jobs, this will also b
ng parameter. Next we need to copy over artifacts from the upstream *build-go-auth* job.



ecute Shell build step with the Rancher compose up command that we specified earlier. Note that you will also need to setup rancher-compose on Jenkins ahead of time an
are setting up our job to reinstall compose every time for the sake of simplicity. You will need to specify the Rancher API key, Rancher API Secret and your Rancher server U
e the Parameterized build option or the Masked Passwords plugin to avoid exposing your secret or having to enter it every time. The complete contents of the execute shell s
ss if you have multiple Rancher compose nodes the load balancer containers may launch on different host and hence your route 53 record-set may need to be updated.

```
.com/rancher/rancher-compose/releases/download/v0.5.1/rancher-compose-linux-amd64-v0.5.1.tar.gz -O - | tar -zx
v0.5.1/rancher-compose .
ose-v0.5.1

-project-name messenger-int \
RANCHER_SERVER:PORT/v1/   \
KEY>                      \
ET_KEY>                   \
-force-upgrade --pull --confirm-upgrade auth-service
```

s the Pipeline we started in the Docker Based Build Pipelines article, now looks like the image shown below. Every check-in to our sample application now gets compiled to
tomated tests pass. That change then gets packaged, and tested with integration testes and finally deployed for manual testing. The five steps below provide a good baselin
y move code from development to testing and deployment stages. Having a continuous deployment pipeline ensures that all code is not only tested by automated systems b
as a model for production deployment automation and can test the operations tooling and code to deploy your application on a continual basis.

eploying a new version

:ode to a persistent testable environment we will let QA (Quality Assurance) team test the changes for a period of time. Once they certify that the code is ready, we can creat production. The way releases work with git-flow is similar to how feature branches (which we talked about in the previous article work. We start a release using the *git flow* is will create a new named release branch. In this branch we will perform house-keeping actions such as incrementing version numbers and making any last minute changes

```
art v1
ranch 'release/v1'

ease/v1' was created, based on 'develop'
anch 'release/v1'

number now!
last-minute fixes in preparing your release

nish 'v1'
```

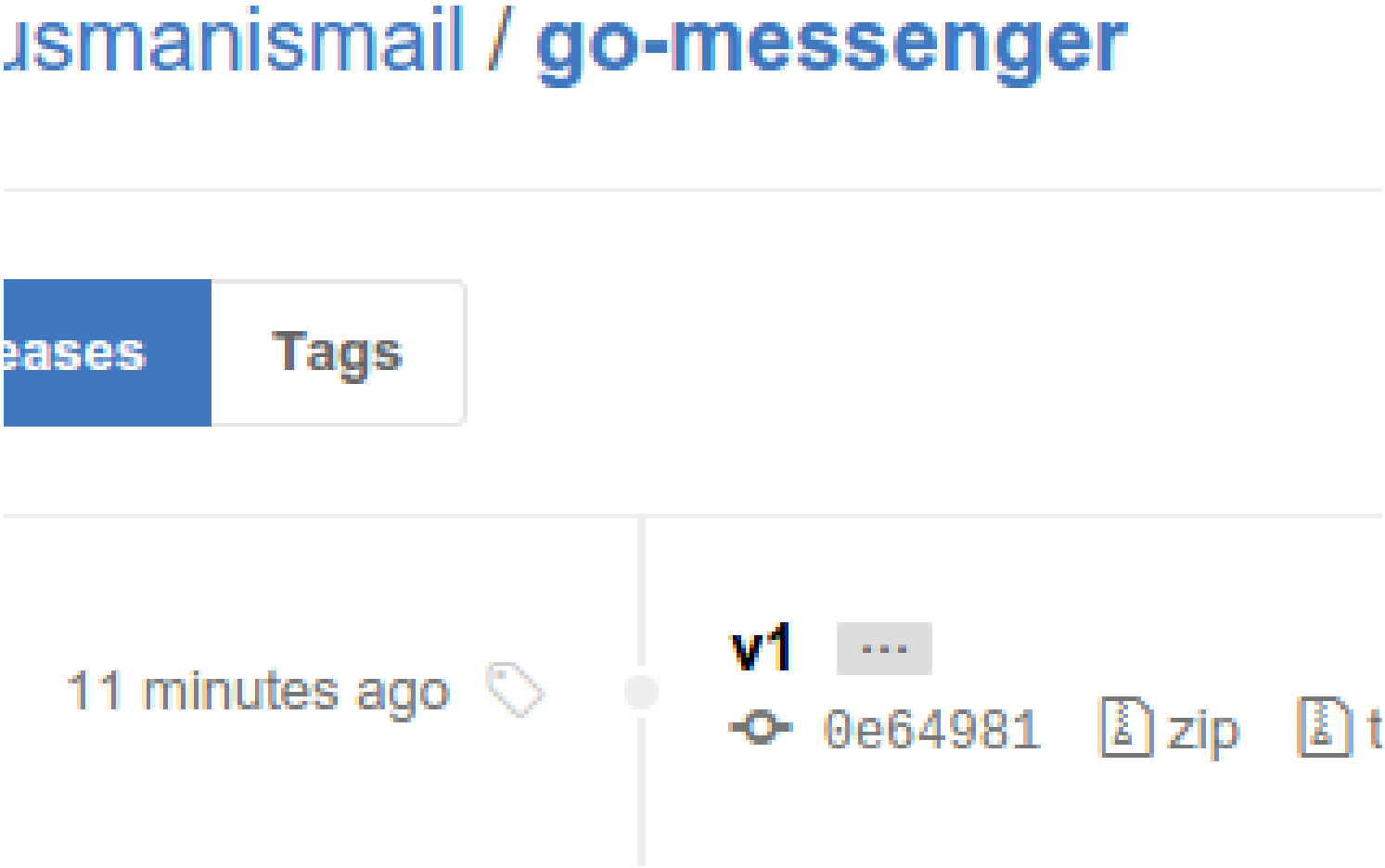
n the *release finish* command to merge the release branch into the master branch. This way master always reflects the latest released code. Further each release is tagged s h release. Since we don't want any other changes to go in, let's finalize the release.

```
'master'
recursive' strategy.

insertion(+)
ase/v1 (was 7ae8ca4).

ve been fetched from 'origin'
s been merged into 'master'
agged 'v1'
s been back-merged into 'develop'
elease/v1' has been deleted
```

the release to remote repository.

```
ter
ushes the v1 tag to remote repository
```

ting your git repository, you should now have a new release.



images to DockerHub with a version that matches the release name. To do so, let's trigger our CD pipeline by running the first job. If you recall, we setup Git Parameter plugi
ur filter from git. This normally defaults to *develop* however, when we trigger the pipeline manually we can choose from git tags. For example in the section below, we have tv
of them and kick off the integration and deployment pipeline.

oject build-go-auth

uild requires parameters:

AUTH_VERSION v1.1
v1

You must have built the projec
If you wipe out your workspac
Version of go-auth application based

uild

wing steps and deploy our application with version 1.1 to our long running integration environment all with a couple of clicks:

ed release from git

tion and run unit tests

age with tag v1.1 (e.g., usman/go-auth:v1.1)

ests

(usman/go-auth:v1.1) to DockerHub

on to our integration environment

creating a continuous deployment pipeline which can put our sample application on an integration environment. We also looked at integrating DNS and HTTPS support in or which clients can integrate. In the next article we will look at running production environments. Deploying to production environments presents it's own set of challenges as (ideally zero) downtime. Furthermore, Production environments present challenges as they have to scale out to meet load while also scaling back to control cost. Lastly, we order to provide automatic fail over and high availability. In subsequent articles we will look at operations management of docker environments in production as well as diffe services. To get the entire series, please download our eBook: Continous Integration and Deployment with Docker and Rancher. You can also join us for this months online based operations processes.

nd infrastructure engineers, with experience in building large scale distributed services on top of various cloud platforms. You can read more of their work at techtraits.com, eikh respectively.