APR 3 2016 · PRODUCTS

Running multiple applications in Dokku

Setting up multiple applications in a single DigitalOcean instance via Dokku.

Recently, I have looked how to develop multiple small applications and how to deploy them to the cloud. See the following blog posts

- Microservices with fuge
- Local microservice containers

I still could not find a way to actually deploy these *small* applications onto a single cloud instance. Different deployment services tried to create a separate instance to run each application; that is a huge overhead and costs a lot.

Finally, I found a solution that works for me, and is suitable for small to medium sized applications. If you have used Heroku, the concept would be very familiar to you, and in practice the steps are similar.

I am using Dokku - which is Docker-powered Heroku-like application. You can create a single DigitalOcean "droplet" with Dokku 0.4 with 1-click setup and have your own "Heroku" platform. The Dokku can then create and control multiple applications, each application running inside its own Docker container. If you need additional features, there are plugins for load balancing, common databases, etc.

Here are my steps to start and configure separate apps inside a single Dokku instance hosted on DigitalOcean. Hope these steps are useful, especially if you try to configure virtual hosting for application to avoid addressing apps by their transient port numbers.

Step zero

- Read the official Dokku installation guide - the steps might have changed since I have described them.
- **Important** make sure you have your public SSH key uploaded to DigitalOcean before

creating any new droplets for simple and secure login.

Create droplet

Read the Running Dokku on DigitalOcean instructions

- Pick one-click apps
- Select "Dokku 0.4.14 on Ubuntu"
- Pick at least 1GB size
- Add your SSH key for simplicity and security, instead of using the root password
- Choose internal host name, for example "dokku" instead of the default
- Click "Create" button and wait a few seconds

Check Dokku running in the droplet

- Copy droplet's IP4 address and run `ssh root@<ip4>` to login
- Check dokku's installation

```
root@dokku:~# dokku version
0.4.14
```

If the version is old(er), you can upgrade dokku, see instructions

```
sudo apt-get update
sudo apt-get install -qq -y dokku herokuish
```

Configure custom domain name (optional)

If you setup a custom domain, you will be able to use `<application name.domain.com>` for reach your application. Otherwise you will have to use `<ip4>:<port>` to reach the individual applications. Even worse, every time you redeploy an application X, you will get a new port number: `<ip4:port1>`, `<ip4:port2>`, `<ip4:port3>` which makes connecting to an application difficult.

With a custom domain name and virtual host names, the application stays at the same name `<name-of-the-app.domain.com>` and the port is remapped internally.

WARNING maybe it is possible, but I failed to switch existing an Dokku droplet from using

port numbers to the custom domain name; had to rebuild the droplet and create the applications again after configuring the domain. Thus it is recommended that you perform domain / virtual host name configuration first, before creating any apps.

Buy a domain and point DNS record at the droplet

1. First, you need a custom domain name for the droplet. Purchase a domain name `<domain.com>` , for example from domains.google.com
2. Go into the settings (where you bought the domain) and point DNS servers at the DigitalOcean's name servers: `ns1.digitalocean.com` , `ns2.digitalocean.com` and `ns3.digitalocean.com`
3. Verify that the new domain servers are listed in `$ whois <domain.com> | grep "Name Server"` response
4. From the DO droplet's actions pick "Add Domain" and enter the new domain `<domain.com>`
5. There should be nothing to do, there is already a default `@ <ip4>` A record created
6. Add wild card mapping for everything else to the same domain - A record `* <ip4>` **Note** just enter a wildcard "*" in the first column, and `<ip4>` address in the second.
7. You should see the following text in the record on the config page

```
1  domain.com. 1800 IN A <ip4>
2  *.domain.com. 1800 IN A <ip4>
```

1. Test the mapping by browsing to `<domain.com>` - you should see the same screen as browsing to `<ip4>` directly
2. Create CNAME record if you need aliases, I don't think you need them with Dokku
3. Test the domain resolution from the command line by using `ping <domain.com>`
4. Test the SSH works using domain name in addition to IP4

```
1  ssh root@<domain.com>
```

Set virtualhost

1. Open the Dokku application in the browser using the new name `<domain.com>` and enter the new domain name in the input box instead of the `<ip4>` address.
2. Click "Finish setup"

Maybe

In order to properly route each application `foo` to `foo.domain.com` you might have to set

the hostname on the droplet. SSH into the droplet and check the current domain name, and set it to `domain.com`

```
sudo hostname -f
sudo hostname domain.com
```

Related info

- How To Set Up a Host Name with DigitalOcean
- How to Use the DigitalOcean Dokku Application

Install LetsEncrypt

In order to get free automatic SSL certificates for our applications and the new domain, need to install Dokku LetsEncrypt plugin.

```
sudo dokku plugin:install https://github.com/dokku/dokku-letsencrypt.
```

Create an application

- ssh into the box
- Create new `hello` application using `dokku apps:create hello`
- Exit SSH from the droplet and switch back to the local computer
- Since Dokku operates under its own user "dokku" we want to associate it with the user `root` to use our SSH key when push to the remote repo. From the local computer execute the following remote command

```
cat ~/.ssh/id_rsa.pub | ssh root@<ip4> "sudo sshcommand acl-add dokku
```

- From the local Git repo (for example I am using bahmutov/hello-world) add Dokku as another remote git server

```
git remote add dokku dokku@<ip4>:hello
or
git remote add dokku dokku@<domain.com>:hello
```

- push the code to the remote (you should see lots of build messages)

```
git push dokku master
```

You should see build process log messages, similar to the build log generated when pushing an app to Heroku.

Hint: you can push a different local branch to Dokku server. Just map it to Dokku's "master" like this `git push dokku local-branch:master` .

If the build step during the push fails, it might be due to the droplet's small size. Try resizing the drop let to have larger memory limit and push again. For example, I could run small HTTP server on a droplet with 512MB, but ExpressJS required at least 1 GB droplet.

Checking the application's state

- ssh into the box
- see the list of running applications

```
root@dokku:~# dokku apps
=====> My Apps
hello
```

- check the app's logs

```
root@dokku:~# dokku logs hello
2016-03-20T23:07:18.916218565Z app[web.1]:
2016-03-20T23:07:18.916313189Z app[web.1]: > hello-world@1.0.0 start /
2016-03-20T23:07:18.916324098Z app[web.1]: > node index.js
2016-03-20T23:07:18.916331059Z app[web.1]:
2016-03-20T23:07:19.032003761Z app[web.1]: Server running at port 500
```

- Make direct request to the application running inside the Docker container. To find the internal IP and port, see the `nginx.conf` file created with the application.

```
dokku:~# cat /home/dokku/hello-world/nginx.conf
...
upstream hello-world-5000 {
```

```
4     server 172.17.0.4:5000;
5   }
```

Use the above "server" IP to check the web application

```
1   dokku:~# curl 172.17.0.4:5000
2   Hello World
```

If you have configured custom domain and virtual host name resolution, skip the IP4 port step below.

- Find the **actual port** on the droplet mapped to the application (running inside Docker).

```
1   root@dokku:~# dokku urls hello
2   http://dokku:32769 (container)
3   http://dokku:80 (nginx)
```

- Check the application's response by running `curl` or whatever is your preferred HTTP client

```
1   $ http <ip4>:32769 // or http://<hello.domain.com>
2   HTTP/1.1 200 OK
3   Connection: keep-alive
4   Content-Type: text/plain
5   Date: Sun, 20 Mar 2016 23:24:22 GMT
6   Transfer-Encoding: chunked
7   Hello World
```

Our Hello World is running at that full address `<ip4>:32769` (or at `hello.domain.com`).

Setting up a database container (service)

If you need a database, for example MongoDB, you can easily create it to run in a separate container on Dokku.

- SSH into Dokku box and install Mongo plugin `dokku plugin:install https://github.com/dokku/dokku-mongo.git mongo`

- Then create a new mongo service, for example name it "my-db" `dokku mongo:create my-db`
  - the command will show full Mongo URI
  - You can always list created mongo service with `dokku mongo:list` and get details about a particular one with `dokku mongo:info my-db`

Let us say we want to use this MongoDB from a Parse server

Setting up Parse server

From your local machine:

- Clone the parse example to local folder
- Install dependencies `npm i -r http://registry.npmjs.org/` just to test the server locally
- SSH into Dokku box
- Create `parse` application on Dokku host `dokku apps:create parse`
- Link the mongo service to the new app `dokku mongo:link my-db parse`
- Note: you can see other environment variables in the index.js file
- Add remote Dokku as another git server `git remote add dokku dokku@<custom domain>:parse` and push the code `git push dokku master`
- SSH into the box and set app id and master key environment variables

```
dokku config:set parse APP_ID=<app id>
dokku config:set parse MASTER_KEY=<key>
```

Hint: you can set multiple variables at once using

```
dokku config:set <app> key1=value1 key2=value2
`
```

If everything goes well, you should have a fresh Parse server.

- Test the server (using the guide)

```
http http://<parse.domain.com>
{"error":"unauthorized"}
```

Try making a GET request

```
curl -X GET -H "X-Parse-Application-Id: <app>" -H "X-Parse-Master-Key
    http://<parse.domain.com>/parse/classes/Foo/1
```

Setting up SSL for an application

We have already installed LetsEncrypt plugin. Now we can use it to enable SSL for our `hello.domain.com` application.

```
dokku config:set --no-restart hello DOKKU_LETSENCRYPT_EMAIL=<me@gmai
dokku letsencrypt hello
-----> Configuring hello.domain.com...(using built-in template)
-----> Creating https nginx.conf
-----> Running nginx-pre-reload
       Reloading nginx
-----> Configuring hello.domain.com...(using built-in template)
-----> Creating https nginx.conf
-----> Running nginx-pre-reload
       Reloading nginx
-----> Disabling ACME proxy for hello...
       done
```

While we are at the Dokku box, print the two urls for "hello" app

```
dokku urls hello
http://hello.domain.com
https://hello.domain.com
```

Let us try regular request, and should get a redirect to the secure end point.

```
$ http http://hello.domain.com
HTTP/1.1 301 Moved Permanently
Connection: keep-alive
Content-Length: 178
Content-Type: text/html
Date: Wed, 06 Jul 2016 04:07:29 GMT
Location: https://hello.domain.com:443/
```

The secure endpoint works

```
$ http https://hello.domain.com
HTTP/1.1 200 OK
Hello World
```

Any modern browser is also very happy loading this secure page.

Note the free certificates expire after N months. Use the plugin's commands to setup a cron job to auto-renew them.

Note if you try to get a second certificate for a different application, it might fail. Check if the domain name is displayed correctly, sometimes it thinks the domain is *still the first one**

For example, this will work

```
dokku letsencrypt hello
...
-----> Enabling ACME proxy for api...
-----> Getting letsencrypt certificate for hello...
         - Domain 'hello.domain.com'
```

But when you try to get a certificate for second application "bye"

```
```
dokku letsencrypt bye
...
-----> Enabling ACME proxy for api...
-----> Getting letsencrypt certificate for bye...
         - Domain 'hello.domain.com'
ERROR: could not get ...
```

In this case try to cleanup "letsencrypt" for every application.

```
; do for each application
dokku letsencrypt:revoke <app>
dokku letsencrypt:cleanup <app>
; then get new certificate for each application
```
```

```
dokku letsencrypt <app>
```

Making SSL bulletproof

You can test your new SSL setup using SSL Labs Just open the browser at `https://www.ssllabs.com/ssltest/analyze.html?d=<app name.domain.com>` and wait a minute. Most likely your default setup will get a "C" due to SSLv3 enabled, which has Poodle vulnerability

Note the commands below could be executed using a single script I placed into bahmutov/secure-dokku.

Disable SSLv3

Let us disable the SSLv3 in the nginx configuration for all applications. SSH into the Dokku box, the nginx configuration should be in the file `/etc/nginx/conf.d/dokku.conf` . Add the following line:

```
echo "ssl_protocols TLSv1 TLSv1.1 TLSv1.2;" >> /etc/nginx/conf.d/dokku
service nginx reload
```

You can confirm that the SSLv3 was disabled from the command line - the following command should return an error

```
openssl s_client -connect <app.domain.com>:443 -ssl3
```

Boom! SSL Labs should bump the grade from "C" to "A-". If you want to go all the way to "A" you should configure forward secrecy.

Enable Forward Secrecy

To enable this feature and disable weak keys, I tweaked the configuration file following the advice in excellent Strong SSL Security on nginx; here is the result

/etc/nginx/conf.d/dokku.conf

```
include /home/dokku/*/nginx.conf;

server_tokens off;

```

```
ssl_session_cache shared:SSL:20m;
ssl_session_timeout 10m;

ssl_prefer_server_ciphers on;

# default ciphers
# ssl_ciphers EECDH+AES128:RSA+AES128:EECDH+AES256:RSA+AES256:EECDH+3

# strong ciphers only
# see https://weakdh.org/sysadmin.html
ssl_ciphers 'ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA25

ssl_dhparam /etc/nginx/conf.d/dhparams.pem;

# no SSLv3
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
```

The file `/etc/nginx/conf.d/dhparams.pem` was generated by me

```
openssl dhparam -out /etc/nginx/conf.d/dhparams.pem 2048
```

After editing the `/etc/nginx/conf.d/dokku.conf` file, restart the service again

```
service nginx reload
```

If there is a problem, inspect the log file, usually in `/var/log/nginx/error.log` . Most likely there is a spelling error.

After everything said and done, your website should get solid "A"!

Simple persistent storage

If you deploy a new version of the application, all its data inside the Docker container is lost. In order to save the data, and have it available after new deploy, you need to mount an external folder.

See docs for more information.

Let us mount external folder `/var/lib/dokku/data/storage/hello` as `/tmp/data` inside the application "hello"

```
mkdir -p /var/lib/dokku/data/storage/hello
dokku storage:mount hello /var/lib/dokku/data/storage/hello:/tmp/data
dokku ps:restart hello
```

And ... the application "hello" cannot actually write into that folder, see issue 2215. Seems every Dokku container runs a random non-privileged user, thus that user cannot actually write into the folder.

As a temporary workaround until Dokku v7, you can give everyone write access to the folder.

```
chmod a+w /var/lib/dokku/data/storage/hello
```

Since every deploy changes the user, if the new user creates new files or recreates the existing ones, we need to run this command again. I scheduled a quick cron job to do this very often; run `crontab -e` then enter the following to run the command every two minutes

```
# m h  dom mon dow   command
*/2 * * * * chmod a+w /var/lib/dokku/data/storage/hello/*
```

Hopefully, this would be fixed in a better way soon.

Setting up continuous deployment

We do not want to perform manual deploys, instead I prefer to deploy my applications automatically. For publishing NPM modules I prefer semantic release, similarly for deploys we can use a CI server to push the new code as soon as unit tests pass.

Simple case - Shippable

Let us follow the instructions from Shippable and setup the deployment pipeline for `hello-world` application.

First, we need to give Shippable access to Dokku so it can deploy applications. Browse to Shippable "CI / Settings" tab and copy the public "Deployment Key". Then from the local terminal add this key to Dokku (consult User Management / Adding Deploy Users). If the SSH key is in the clipboard, we can simply run

```
$ pbpaste | ssh root@domain.com "sudo sshcommand acl-add dokku shippal
```

The last word "shippable" is the name of the new user we have just created for Shippable connections.

Second, enable the build for the project, in this case `hello-world`. A simple `shippable.yml` file pushes to the Dokku remote server after a successful build

```
language: node_js
node_js:
  - 6
after_success:
  - if [ "$BRANCH" == "master" ]; then git remote add dokku dokku@dom
  - if [ "$BRANCH" == "master" ]; then git push dokku master; fi
```

Note the full git repo for the application "hello" is still `dokku@domain.com:hello` - even if we have created user "shippable", it is only to login via a specific SSH key.

Via Shippable user interface you can restrict the builds to run only for tagged commits, to make sure you only deploy a finished features for example. You can find the full configuration docs here.

Slightly more complex - CircleCI

Similarly, you can configure other CI services like CircleCI, Codeship to push code to Dokku after a successful build. First, we need to generate a new SSH key pair, upload the private key to CircleCI and public to Dokku.

```
ssh-keygen -q -t rsa -b 2048 -f "/tmp/ci_rsa" -N ""
```

This has generated two files in `/tmp` folder. Grab the private key and place it in the clipboard

```
cat /tmp/ci_rsa | pbcopy
```

Hint: even better is to store the generated key pair in your folder for future use

```
folder=$(pwd)
ssh-keygen -q -t rsa -b 2048 -f "$folder/.ssh/dokku-circle-ci_rsa" -N
```

Browse to the CircleCI page `https://circleci.com/gh/<username>/<project>`

`/edit#ssh` and paste the text into the text box (leave the `hostname` blank to allow reusing this key for any dokku domain name).

Now grab the public key file and pipe it directly to the Dokku ssh command

```
cat /tmp/ci_rsa.pub | ssh root@domain.com "sudo sshcommand acl-add dol
```

That is it, we should be able to push new code directly from the build using the command `git remote add dokku dokku@domain.com:hello` first.

```
deployment:
  production:
    branch: master
    commands:
      - echo Deploying master
      - git remote add dokku dokku@domain.com:hello
      - git push dokku master
```

- How to setup deployment from CircleCI

Beefing up server security

This is a list of advanced steps to make the Dokku box even more secure.

Disable SSH login using passwords

We should only login using SSH key, not passwords. Edit file `/etc/ssh/sshd_config` and insert line

```
PasswordAuthentication no
```

If you are only logging in from specific IP(s) you can restrict the Dokku host to only allow logins from these addresses, see this guide **Note** this will probably break continuous deployment!

Restart the SSH daemon

```
service ssh restart
```

Login with non-root user

It is a good idea to make a new non-root user just to be able to SSH into the box, and disable SSH login for the root user. See how.

Enable 2FA for SSH login

This is an advanced step to make sure only you can login. Read the Configuring SSH with 2FA and Key based authentication

Conclusion

I love Dokku - it seems mature, feature complete, simple and allows me to spend very little money to run a plenty of small applications. While not as simple as Zeit now, Dokku has a huge advantage - every tool that runs in a Docker container can be deployed in seconds.

Follow Gleb Bahmutov @bahmutov, see his projects at glebbahmutov.com

---

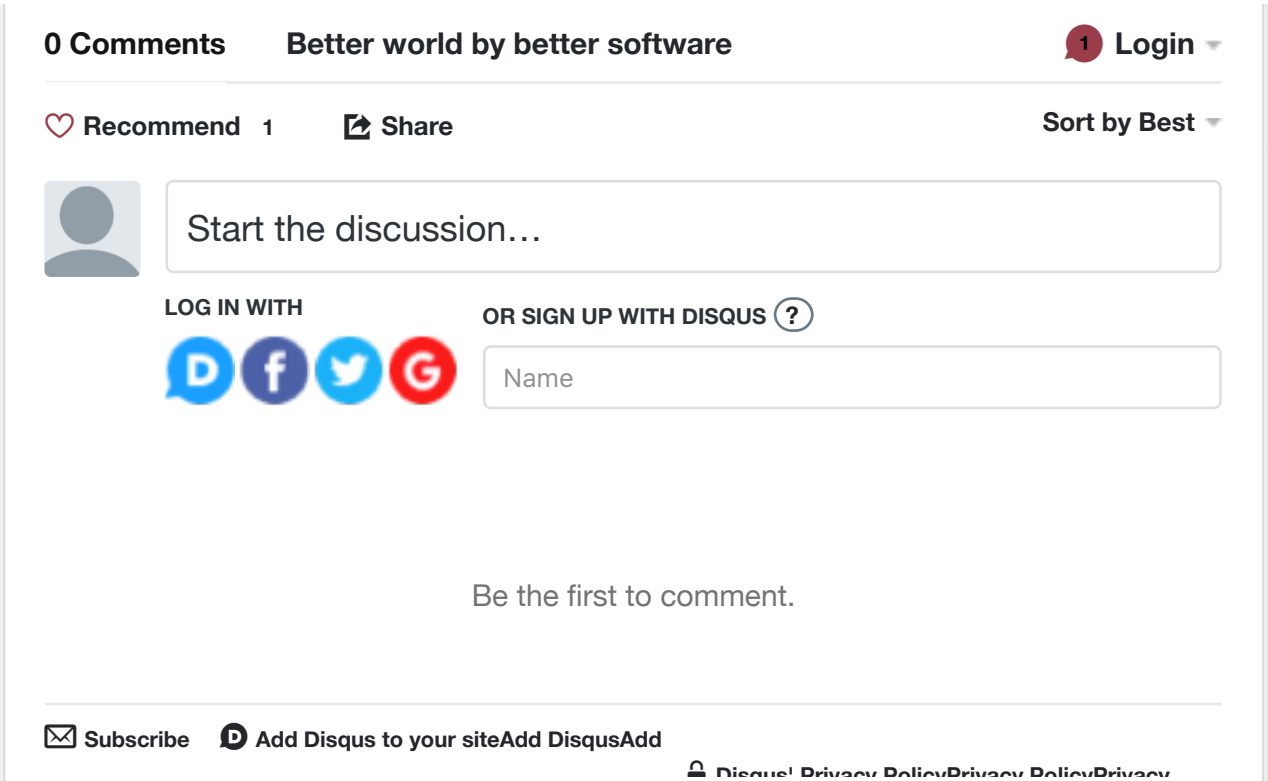
#docker Comments Share

NEWER

Test if a function is pure revisited

OLDER

JavaScript nuggets



CATEGORIES

- book review (15)
- people (17)
- process (102)
- products (281)

TAGS

- QUnit (5)
- advice (98)
- angularjs (58)
- angularjs2 (1)
- assertions (9)
- ast (8)
- boilerplate (14)
- browser (18)
- ci (7)
- concurrency (1)
- cyclejs (7)

cypress (11)
d3 (3)
db (11)
docker (9)
es6 (13)
es7 (1)
functional (67)
generators (5)
git (11)
grunt (7)
gulp (3)
hyperapp (5)
immutable (4)
interview (3)
jade (4)
javascript (160)
jshint (3)
modular development (27)
nodejs (71)
performance (17)
presentation (4)
promises (31)
proposal (2)
ramda (1)
reactive (12)
reactjs (2)
screencast (1)
security (7)
sentry (10)
service workers (6)
testing (73)
tutorial (8)
typescript (2)
ui (2)
vuejs (4)
web workers (6)

TAG CLOUD

QUnit advice angularjs angularjs2 assertions ast boilerplate browser ci concurrency cyclejs cypress d3 db docker es6 es7 functional generators git grunt gulp hyperapp immutable interview jade javascript jshint modular development nodejs performance presentation promises proposal ramda reactive reactjs screencast security sentry service workers testing tutorial typescript ui vuejs web workers

ARCHIVES

April 2018 (3)
March 2018 (4)
February 2018 (1)
January 2018 (5)
December 2017 (3)
November 2017 (4)
September 2017 (2)
August 2017 (9)
July 2017 (4)
June 2017 (4)
May 2017 (2)
April 2017 (9)
March 2017 (10)
February 2017 (5)
January 2017 (4)
December 2016 (5)
November 2016 (1)
October 2016 (2)
September 2016 (1)
August 2016 (5)
July 2016 (2)
June 2016 (7)
May 2016 (5)
April 2016 (8)
March 2016 (10)
February 2016 (5)
January 2016 (8)
December 2015 (9)
November 2015 (8)
October 2015 (4)
September 2015 (4)
August 2015 (11)
July 2015 (4)
June 2015 (9)
May 2015 (8)
April 2015 (10)
March 2015 (15)
February 2015 (13)
January 2015 (10)
December 2014 (8)
November 2014 (19)
October 2014 (8)
September 2014 (13)
August 2014 (12)
July 2014 (13)
June 2014 (9)
May 2014 (11)
April 2014 (13)

RECENTS