

Title: Docker Compose Continuous Deployment setup

Post Body:

I am looking for a way to deploy `docker-compose` images and / or builds to a remote sever, specifically but not limited to a DigitalOcean VPS.

`docker-compose` is currently working on the CircleCI Continuous Integration service, where it automatically verifies that tests pass. But, it **should deploy automatically on success**.

My `docker-compose.yml` is looking like this:

```
version: '2' services: web: image: name/repo:latest ports: - '3000:3000' volumes: - /app/node_modules
```

`docker-compose.override.yml`:

```
version: '2' services: web: build: .
```

`circle.yml` relevant part:

```
deployment: latest: branch: master commands: - docker login -e $DOCKER_EMAIL -u $DOCKER_USER -p $DOCKER_PASS
```

Accepted Answer:

Your `docker-compose` and `circle` configurations are already looking pretty good.

Your `docker-compose.yml` is already setup to gather the image from the [Docker Hub](#), which is being uploaded after tests have passed. We will use this image on the remote server, which instead of building the image up every time (which takes a long time), we'll use this already prepared one.

You did well into separating the `build: .` into a `docker-compose.override.yml` file, as [priority issues can arise if we use a `docker-compose.prod.yml` file](#).

Let's get started with the deployment:

There are various ways of getting your deployment done. The most popular ones are probably **SSH** and Webhooks.

We'll use SSH.

Edit your `circle.yml` config to take an additional step, which to load our `.scripts/deploy.sh` bash file:

```
deployment: latest: branch: master commands: - docker login -e $DOCKER_EMAIL -u $DOCKER_USER -p $DOCKER_PASS
```

`deploy.sh` will contain a few instructions to connect into our remote server through SSH and update both the repository and Docker images and reload Docker Compose services.

Prior executing it, you should have a remote server that contains your project folder (i.e. `git clone https://github.com/zurffy/my-project`), and both [Docker](#) and [Docker Compose](#) installed.

`deploy.sh`

```
#!/bin/bash DIR='${ cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd }' ( cd "$DIR/.." # Go to project dir. ssh $SSH_USERNAME
```

Notice: last EOF is not indented. That's how bash [HEREDOC](#) works.

`deploy.sh` steps explained:

1. `ssh $SSH_USERNAME@$SSH_HOSTNAME`: connects to the remote host through SSH. `-o StrictHostChecking=no` avoids the SSH asking whether we trust the server.
2. `cd $SSH_PROJECT_FOLDER`: browses to the project folder (the one you did gather through `git clone ...`)
3. `git pull`: updates project folder. That's important to keep `docker-compose` / `Dockerfile` updated, as well as any shared volume that depends on some source code file.
4. `docker-compose stop`: Our remote dependencies have just been downloaded. Stop the `docker-compose` services which are current running.
5. `docker-compose rm -f`: Remove `docker-compose` services. This step is really important, otherwise [we'll reuse old volumes](#).
6. `docker-compose -f docker-compose.yml -f docker-compose.prod.yml up -d`. Execute your `docker-compose.prod.yml` which extends `docker-compose.yml` in detached mode.

On your CI you will need to fill in the following environment variables (that the deployment script uses):

- `$SSH_USERNAME`: your SSH username (i.e. root)
- `$SSH_HOSTNAME`: your SSH hostname (i.e. stackoverflow.com)
- `$SSH_PROJECT_FOLDER`: the folder where the project is stored (either relative or absolute to where the `$SSH_USERNAME` is on login. (i.e. my-project/)

What about the SSH password? CircleCI in this case offers a way to store SSH keys, so password is no longer needed when logging in through SSH.

Otherwise simply edit the `deploy.sh` SSH connection to something like this:

```
sshpass -p your_password ssh user@hostname
```

More about SSH password [here](#).

In conclusion, all we had to do was to create a script that connected with our remote server to let it know that the source code had been updated. Well, and to perform the appropriate upgrading steps.

FYI, that's similar to how the alternative Webhooks method work.

Highest Rated Answer:

WatchTower solves this for you.

<https://github.com/v2tec/watchtower>

Your CI just needs to build the images and push to the registry. Then WatchTower polls the registry every N seconds and automagically restarts your services using the latest and greatest images. It's as simple as adding this code to your compose yaml:

```
watchtower: image: v2tec/watchtower volumes: - /var/run/docker.sock:/var/run/docker.sock - /root/.docker/config.json:/conf
```