

CONTENTS

Prerequisites

Step 1 — Installing Mosquitto

Step 2 — Installing Certbot for Let's Encrypt Certificates

Step 3 — Running Certbot

Step 4 — Setting up Certbot Automatic Renewals

Step 5 — Configuring MQTT Passwords

Step 6 — Configuring MQTT SSL

Step 7 — Configuring MQTT Over Websockets (Optional)

Conclusion

RELATED

Initial Server Setup with Ubuntu 12.04

[View](#) 

How To Install Ruby on Rails on Ubuntu 12.04 LTS (Precise Pangolin) with RVM

[View](#) 

// Tutorial //

How to Install and Secure the Mosquitto MQTT Messaging Broker

on Ubuntu 16.04

Published on December 10, 2016



By [Brian Boucheron](#)

Developer and author at DigitalOcean.



Not using Ubuntu 16.04?

Choose a different version or distribution.

Ubuntu 16.04 ▼

Introduction

[MQTT](#) is a machine-to-machine messaging protocol, designed to provide lightweight publish/subscribe communication to “Internet of Things” devices. It is commonly used for geo-tracking fleets of vehicles, home automation, environmental sensor networks, and utility-scale data collection.

[Mosquitto](#) is a popular MQTT server (or *broker*, in MQTT parlance) that has great community support and is easy to install and configure.

In this tutorial, we'll install Mosquitto, retrieve SSL certificates from Let's Encrypt, and set up our broker to use SSL to secure our password-protected MQTT communications.



Prerequisites

Before starting this tutorial, you will need:

- An Ubuntu 16.04 server with a non-root, sudo-enabled user and basic firewall set up, as detailed in [this Ubuntu 16.04 server setup tutorial](#).
- A domain name pointed at your server, as per [How to Set Up a Host Name with DigitalOcean](#). This tutorial will use `mqtt.example.com` throughout.

Step 1 – Installing Mosquitto

Ubuntu 16.04 has a fairly recent version of Mosquitto in its default software repository. Log in with your non-root user and install Mosquitto with `apt-get`.

```
$ sudo apt-get install mosquitto mosquitto-clients
```

Copy

By default, Ubuntu will start the Mosquitto service after install. Let's test the default configuration. We'll use one of the Mosquitto clients we just installed to subscribe to a topic on our broker.

Topics are labels that you publish messages to and subscribe to. They are arranged as a hierarchy, so you could have `sensors/outside/temp` and `sensors/outside/humidity`, for example. How you arrange topics is up to you and your needs. Throughout this tutorial we will use a simple test topic to test our configuration changes.

Log in to your server a second time, so you have two terminals side-by-side. In the new terminal, use `mosquitto_sub` to subscribe to the test topic:

```
$ mosquitto_sub -h localhost -t test
```

Copy

`-h` is used to specify the hostname of the MQTT server, and `-t` is the topic name. You'll see no output after hitting ENTER because `mosquitto_sub` is waiting for messages to arrive. Switch back to your other terminal and publish a message:

```
$ mosquitto_pub -h localhost -t test -m "hello world"
```

Copy

The options for `mosquitto_pub` are the same as `mosquitto_sub`, though this time we use the additional `-m` option to specify our message. Hit ENTER, and you should see **hello world** pop up in the other terminal. You've sent your first MQTT message!

Enter CTRL+C in the second terminal to exit out of `mosquitto_sub`, but keep the connection to the server open. We'll use it again for another test in Step 5.

Next, we'll secure our installation with SSL using Certbot, the new Let's Encrypt client.

Step 2 – Installing Certbot for Let's Encrypt Certificates

Let's Encrypt is a new service offering free SSL certificates through an automated API. There are many clients that can talk to the API, and Ubuntu includes the official client in their default repository, but it's a bit out of date and lacks one important feature we need.

Instead, we'll install the official client from an Ubuntu PPA, or *Personal Package Archive*. These are alternative repositories that package more recent or more obscure software. First, add the repository.

```
$ sudo add-apt-repository ppa:certbot/certbot
```

Copy

You'll need to press `ENTER` to accept. Afterwards, update the package list to pick up the new repository's package information.

```
$ sudo apt-get update
```

Copy

And finally, install the official Let's Encrypt client, called `certbot`.

```
$ sudo apt-get install certbot
```

Copy

Now that we have `certbot` installed, let's run it to get our certificate.

Step 3 – Running Certbot

`certbot` needs to answer a cryptographic challenge issued by the Let's Encrypt API in order to prove we control our domain. It uses ports `80` (HTTP) and/or `443` (HTTPS) to accomplish this. We'll only use port `80`, so let's allow incoming traffic on that port now:

```
$ sudo ufw allow http
```

Copy

Output

```
Rule added
```

We can now run Certbot to get our certificate. We'll use the `--standalone` option to tell Certbot to handle the HTTP challenge request on its own, and `--standalone-supported-challenges http-01` limits the communication to port `80`. `-d` is used to specify the domain you'd like a certificate for, and `certonly` tells Certbot to just retrieve the certificate without doing any other configuration steps.

```
$ sudo certbot certonly --standalone --standalone-supported-challenges http-01 -d Copy ⌘
```



When running the command, you will be prompted to enter an email address and agree to the terms of service. After doing so, you should see a message telling you the process was successful and where your certificates are stored.



We've got our certificates. Now we need to make sure Certbot renews them automatically when they're about to expire.

Step 4 – Setting up Certbot Automatic Renewals

Let's Encrypt's certificates are only valid for ninety days. This is to encourage users to automate their certificate renewal process. We'll need to set up a regularly run command to check for expiring certificates and renew them automatically.

To run the renewal check daily, we will use `cron`, a standard system service for running periodic jobs. We tell `cron` what to do by opening and editing a file called a `crontab`.

```
$ sudo crontab -e
```

Copy

You'll be prompted to select a text editor. Choose your favorite, and you'll be presented with the default `crontab` which has some help text in it. Paste in the following line at the end of the file, then save and close it.

`crontab`

```
. . .  
15 3 * * * certbot renew --noninteractive --post-hook "systemctl restart mosquitto"
```

The `15 3 * * *` part of this line means "run the following command at 3:15 am, every day". The `renew` command for Certbot will check all certificates installed on the system and update any that are set to expire in less than thirty days. `--noninteractive` tells Certbot not to wait for user input.

`--post-hook "systemctl restart mosquitto"` will restart Mosquitto to pick up the new certificate, but only if the certificate was renewed. This `post-hook` feature is what older versions of the Let's Encrypt client lacked, and why we installed from a PPA instead of the default Ubuntu repository. Without it, we'd have to restart Mosquitto every day, even if no certificates were actually updated. Though your MQTT clients should be configured to reconnect automatically, it's wise to avoid interrupting them daily for no good reason.

Now that automatic certificate renewal is all set, we'll get back to configuring Mosquitto to be more secure.

Step 5 – Configuring MQTT Passwords

Let's configure Mosquitto to use passwords. Mosquitto includes a utility to generate a special password file called `mosquitto_passwd`. This command will prompt you to enter a password for the specified username, and place the results in `/etc/mosquitto/passwd`.

```
$ sudo mosquitto_passwd -c /etc/mosquitto/passwd sammy
```

Now we'll open up a new configuration file for Mosquitto and tell it to use this password file to require logins for all connections:

```
$ sudo nano /etc/mosquitto/conf.d/default.conf
```

Copy

This should open an empty file. Paste in the following:

```
/etc/mosquitto/conf.d/default.conf
```

```
allow_anonymous false
password_file /etc/mosquitto/passwd
```

`allow_anonymous false` will disable all non-authenticated connections, and the `password_file` line tells Mosquitto where to look for user and password information. Save and exit the file.

Now we need to restart Mosquitto and test our changes.

```
$ sudo systemctl restart mosquitto
```

Copy

Try to publish a message without a password:

```
$ mosquitto_pub -h localhost -t "test" -m "hello world"
```

Copy

The message should be rejected:

Output

```
Connection Refused: not authorised.
Error: The connection was refused.
```

Before we try again with the password, switch to your second terminal window again, and subscribe to the 'test' topic, using the username and password this time:

```
$ mosquitto_sub -h localhost -t test -u "sammy" -P "password"
```

Copy

It should connect and sit, waiting for messages. You can leave this terminal open and connected for the rest of the tutorial, as we'll periodically send it test messages.

Now publish a message with your other terminal, again using the username and password:

```
$ mosquitto_pub -h localhost -t "test" -m "hello world" -u "sammy" -P "password" Copy
```



The message should go through as in Step 1. We've successfully added password protection to Mosquitto. Unfortunately, we're sending passwords unencrypted over the internet. We'll fix that next by adding SSL encryption to Mosquitto.

Step 6 – Configuring MQTT SSL

To enable SSL encryption, we need to tell Mosquitto where our Let's Encrypt certificates are stored. Open up the configuration file we previously started:

```
$ sudo nano /etc/mosquitto/conf.d/default.conf
```

Copy

Paste in the following at the end of the file, leaving the two lines we already added:

```
/etc/mosquitto/conf.d/default.conf
```

```
. . .
listener 1883 localhost

listener 8883
certfile /etc/letsencrypt/live/mqtt.example.com/cert.pem
cafile /etc/letsencrypt/live/mqtt.example.com/chain.pem
keyfile /etc/letsencrypt/live/mqtt.example.com/privkey.pem
```

We're adding two separate `listener` blocks to the config. The first, `listener 1883 localhost`, updates the default MQTT listener on port `1883`, which is what we've been connecting to so far. `1883` is the standard unencrypted MQTT port. The `localhost` portion of the line instructs Mosquitto to only bind this port to the localhost interface, so it's not accessible externally. External requests would have been blocked by our firewall anyway, but it's good to be explicit.

`listener 8883` sets up an encrypted listener on port `8883`. This is the standard port for MQTT + SSL, often referred to as MQTTS. The next three lines, `certfile`, `cafile`, and `keyfile`, all point Mosquitto to the appropriate Let's Encrypt files to set up the encrypted connections.

Save and exit the file, then restart Mosquitto to update the settings:

```
$ sudo systemctl restart mosquitto
```

Copy

Update the firewall to allow connections to port `8883`.

```
$ sudo ufw allow 8883
```

Copy



Now we test again using `mosquitto_pub`, with a few different options for SSL:

```
$ mosquitto_pub -h mqtt.example.com -t test -m "hello again" -p 8883 --capath /etc/ssl/certs/
```

Note that we're using the full hostname instead of `localhost`. Because our SSL certificate is issued for `mqtt.example.com`, if we attempt a secure connection to `localhost` we'll get an error saying the hostname does not match the certificate hostname (even though they both point to the same Mosquitto server).

`--capath /etc/ssl/certs/` enables SSL for `mosquitto_pub`, and tells it where to look for root certificates. These are typically installed by your operating system, so the path is different for Mac OS, Windows, etc. `mosquitto_pub` uses the root certificate to verify that the Mosquitto server's certificate was properly signed by the Let's Encrypt certificate authority. It's important to note that `mosquitto_pub` and `mosquitto_sub` will not attempt an SSL connection without this option (or the similar `--cafile` option), even if you're connecting to the standard secure port of `8883`.

If all goes well with the test, we'll see **hello again** show up in the other `mosquitto_sub` terminal. This means your server is fully set up! If you'd like to extend the MQTT protocol to work with websockets, you can follow the final step.

Step 7 – Configuring MQTT Over Websockets (Optional)

In order to speak MQTT using JavaScript from within web browsers, the protocol was adapted to work over standard websockets. If you don't need this functionality, you may skip this step.

We need to add one more `listener` block to our Mosquitto config:

```
$ sudo nano /etc/mosquitto/conf.d/default.conf
```

Copy

At the end of the file, add the following:

```
/etc/mosquitto/conf.d/default.conf
```

```
...
listener 8083
protocol websockets
certfile /etc/letsencrypt/live/mqtt.example.com/cert.pem
cafile /etc/letsencrypt/live/mqtt.example.com/chain.pem
keyfile /etc/letsencrypt/live/mqtt.example.com/privkey.pem
```

This is mostly the same as the previous block, except for the port number and the `protocol websockets` line. There is no official standardized port for MQTT over websockets, but `8083` is the most common.

Save and exit the file, then restart Mosquitto.

```
$ sudo systemctl restart mosquitto
```

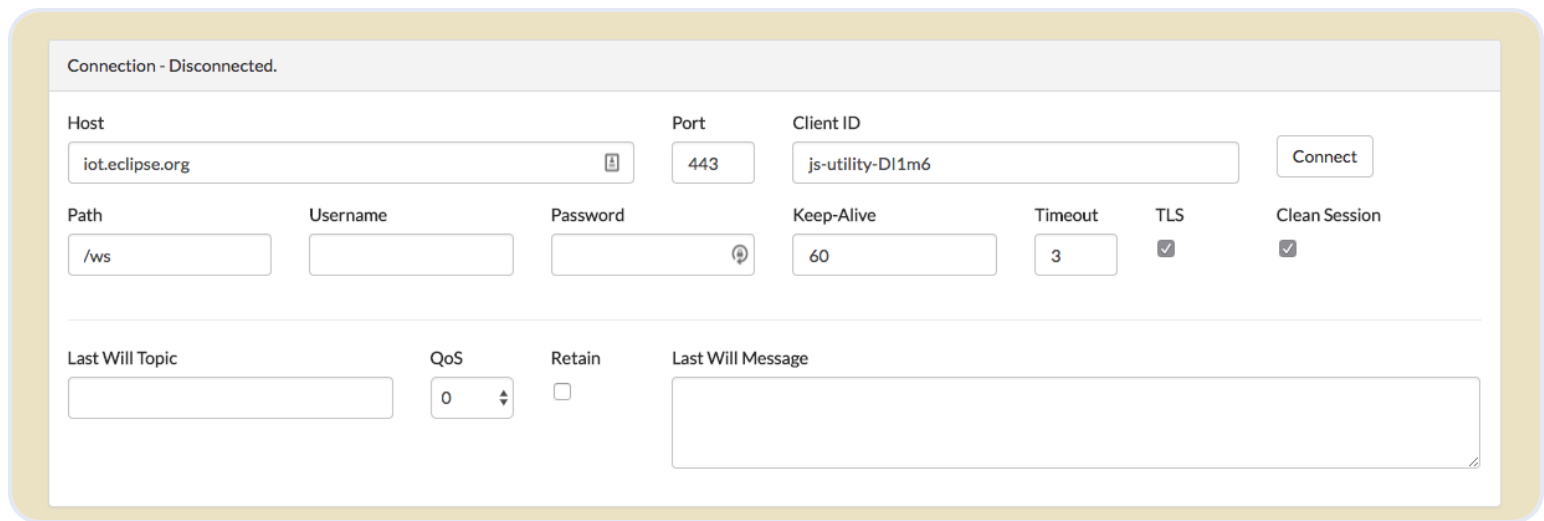
Copy

Now, open up port 8083 in the firewall.

```
$ sudo ufw allow 8083
```

Copy

To test this functionality, we'll use a public, browser-based MQTT client. There are a few out there, but the [Eclipse Paho JavaScript Client](#) is simple and straightforward to use. [Open the Paho client in your browser](#). You'll see the following:

A screenshot of the Eclipse Paho JavaScript Client web interface. The title bar says "Connection - Disconnected." The form has several input fields: "Host" (iot.eclipse.org), "Port" (443), "Client ID" (js-utility-Dl1m6), "Path" (/ws), "Username" (empty), "Password" (empty), "Keep-Alive" (60), "Timeout" (3), "TLS" (checked), and "Clean Session" (checked). There are also fields for "Last Will Topic", "QoS" (0), "Retain" (unchecked), and "Last Will Message". A "Connect" button is in the top right.

Fill out the connection information as follows:

- **Host** should be the domain for your Mosquitto server, `mqtt.example.com`.
- **Port** should be `8083`.
- **ClientId** can be left to the default value, `js-utility-Dl1m6`.
- **Path** can be left to the default value, `/ws`.
- **Username** should be your Mosquitto username; here, we used **sammy**.
- **Password** should be the password you chose.

The remaining fields can be left to their default values.

After pressing **Connect**, the Paho browser-based client will connect to your Mosquitto server.

To publish a message, navigate to the **Publish Message** pane, fill out **Topic** as **test**, and enter any message in the **Message** section. Next, press **Publish**. The message will show up in your `mosquitto_sub` terminal.

Conclusion



We've now set up a secure, password-protected MQTT server, with auto-renewing SSL certificates from the Let's Encrypt service. This will serve as a robust and secure messaging platform.

whatever projects you dream up. Some popular software and hardware that works well with the MQTT protocol includes:

- [OwnTracks](#), an open-source geo-tracking app you can install on your phone. OwnTracks will periodically report position information to your MQTT server, which you could then store and display on a map, or create alerts and activate IoT hardware based on your location.
- [Node-RED](#) is a browser-based graphical interface for ‘wiring’ together the Internet of Things. You drag the output of one node to the input of another, and can route information through filters, between various protocols, into databases, and so on. MQTT is very well supported by Node-RED.
- The [ESP8266](#) is an inexpensive wifi microcontroller with MQTT capabilities. You could wire one up to publish temperature data to a topic, or perhaps subscribe to a barometric pressure topic and sound a buzzer when a storm is coming!

These are just a few popular examples from the MQTT ecosystem. There is much more hardware and software out there that speaks the protocol. If you already have a favorite hardware platform, or software language, it probably has MQTT capabilities. Have fun getting your “things” talking to each other!

Thanks for learning with the DigitalOcean Community. Check out our offerings for compute, storage, networking, and managed databases.

[Learn more about us](#) →

Get \$200 to try DigitalOcean – and do all the below for free!

Build applications, host websites, run open source software, learn cloud computing, and more – every cloud resource you need. If you’ve never tried DigitalOcean’s products or services before, we’ll cover your first \$200 in the next 60 days.

[Sign up now to activate this offer](#) →



About the authors



[Brian Boucheron](#) Author
Developer and author at DigitalOcean.



[Hazel Virdó](#) Editor
senior technical writer

hi! i write [do.co/docs](#) now, but i used to be the senior tech editor publishing tutorials here in the community.

Still looking for an answer?

Ask a question

Search for more help

Was this helpful?

Yes

No



Comments

10 Comments

B *I* U ☺ 📎 🖼️ ✎ H₁ H₂ H₃ ☰ ☷ “” ⓘ ☐ <>



Leave a comment...

This textbox defaults to using **Markdown** to format your answer.

You can type `!ref` in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!



Sign In or Sign Up to Comment

[28ebc2c372c548539fbff934d9](#) • January 23, 2023



Hi, I believe there may something missing in step 5. When creating the default.conf file there should be a line `listener 1883` added as well. If not the restart won't change anything and publish without username and password will still be possible.

[Reply](#)

[dndonatosileo](#) • February 26, 2021



Hi, could you please help me? I'm starting learning and install let's encrypt certificate with DNS challenge. I dont have pem.file but inside the folder `/home/pi/.acme.sh/example.com` I have these files: `example.com.cer` `example.com.conf` `example.com.csr` `example.com.csr.conf` `example.com.key` `ca.cer` `fullchain.cer`. Which ones are the certfile, cafile, and keyfile? I've tryed several combination looking at the names but the command `mosquitto_pub -h example.com -t test -m "hello again" -p 8883 --capath /etc/ssl/certs/ -u "sammy" -P "password"` doesn't work. Moreover inside the `/etc/ssl/certs/` I don't find the certificate. Thanks a lot

[Reply](#)

[LWGShane](#) • October 3, 2020



This needs to be updated as the Paho JavaScript client is no longer available publicly and needs to be downloaded. Is there any other publicly available MQTT browser based clients?

[Reply](#)

[AKBAR SALEEM T](#) • July 31, 2019



I am Totally new to this,I want to run `simple_subscriber` and `simple_publisher` in Ubuntu 14.04 .I tried buy `./bin/simple_subscriber` and same for publisher in other terminal.But data not received by subscriber .I downloaded application from following git account and compiled it. → <https://github.com/LiamBindle/MQTT-C.git> SUBSCRIBER RUNNING*****
aleem@IM-PC-102:~/akbar/mqqt2/MQTT-C/bin\$./simple_subscriber 127.0.0.1 1888
./simple_subscriber listening for 'datetime' messages. Press CTRL-D to exit.



./simple_subscriber disconnecting from 127.0.0.1 akbarsaleem@IM-PC-102:~/akbar/mqqt2/MQTT-C/bin\$**

PUBLISHER*** akbarsaleem@IM-PC-102:~/akbar/mqqt2/MQTT-C/bin\$./simple_publisher 127.0.0.1 1883 ./simple_publisher is ready to begin publishing the time. Press ENTER to publish the current time. Press CTRL-D (or any other key) to exit.

./simple_publisher published : "The time is 2019-07-30 17:58:35" ./simple_publisher published : "The time is 2019-07-30 17:58:41" akbarsaleem@IM-PC-102:~/akbar/mqqt2/MQTT-C/bin\$

Please provide me details to run subscriber and publisher.

[Reply](#)

[kondlemohan1](#) • November 26, 2018

i tried to configure bridge local to cloud mosquito server. am getting this error in cloud mosquito server log.

OpenSSL Error: error:140780E5:SSL routines:ssl23_read:ssl handshake failure

please someone help me which files need to be add in bridge configuration.

[Reply](#)

[arun2619](#) • July 23, 2018

mosquitto_sub -h [motorinkz.in](#) -t test -p 1883 -u "username" -P "password" . when am hitting enter it showing as connection refused but when i change the ports (i.e) -p 8883 or 8083 mosquitto_sub doesnt show any error its ready to receive message from mosquitto_pub, but when i type mosquitto_pub -h [motorinkz.in](#) -t test -p 8883 Or 8083 -u "username" -P "password" -m hello. it shows Error: the connection was lost. please explain me why it happens and give me a solution

[Reply](#)



[Ahmed Taha](#) • May 18, 2018



COOKIE PREFERENCES

Hi, I am always getting this error, can anyone help me please

Failed authorization procedure. mqtt.example.com (http-01): urn:acme:error:dns :: DNS problem: NXDOMAIN looking up A for mqtt.example.com

IMPORTANT NOTES:

- The following errors were reported by the server:

Domain: mqtt.example.com Type: None Detail: DNS problem: NXDOMAIN looking up A for mqtt.example.com

[Reply](#)

[kater](#) • May 11, 2018



Thank you for the recipe. I'm actually using 2 Mosquitto servers now. The existing one for the intranet runs on the standard MQTT port 1883 without passwords and is used by already running ESP nodes on the LAN. The new one on the other port (8883) has TLS and user/password authorization and is reachable from the internet. It just forwards all messages to the first server using the bridge configuration. This has proven to work very well with off-site MQTT nodes for almost three months...

Alas, now the time for renewal of the certificate has come, and the crontab call fails. The server runs behind a NAT router (AVM FritzBox), all relevant ports have been exposed to the outside. I now get an "unexpected error" saying:

Failed authorization procedure. xyyzz.defdc.com (tls-sni-01): urn:acme:error:unauthorized :: The client lacks sufficient authorization :: Incorrect validation certificate for tls-sni-01 challenge. Requested f5xxxcc.6cxxx18.acme.invalid from 87.xx.yy.zz:443. Received 1 certificate(s), first certificate had names "fritz.box, fritz.nas, xyyzz.mofoo.com, myfritz.box, www.fritz.box, www.fritz.nas, www.myfritz.box".

xyyzz.defdc.com and xyyzz.mofoo.com are both Dynamic DNS domains resolving correctly to the cited IPv4 address. All "fritz" domains are made up by the FritzBox, I suppose. I already tried different options to "certbot renew", but got errors in each case.

Any chance to renew the certificate (even manually)? Or do I have to recreate it completely with different parameters? (which ones?)

[Show replies](#) ▾ [Reply](#)



[GooPag](#) • April 29, 2018



Nice article!

by [Go](#)

[Reply](#)

[uborzz](#) • April 23, 2018



Nice, it works =)

I would like to create a bridge from my local broker (mosquitto on centos) to my secured digital oceans server (mosquitto in ubuntu). Any guide or help about this topic? Couldn't find info about it :S

[Reply](#)

Load More Comments



This work is licensed under a Creative Commons Attribution-NonCommercial- ShareAlike 4.0 International License.

Try DigitalOcean for free

Click below to sign up and get **\$200 of credit** to try our products over 60 days!

Sign up →



Ubuntu

Linux Basics

JavaScript

React

Python

Security

MySQL

Docker

Kubernetes

Browse all topic tags

[Free Managed Hosting →](#)

[All tutorials →](#)

Questions

Q&A Forum

Ask a question

DigitalOcean Support



Congratulations on unlocking the whale ambience easter egg! Click the whale button in the bottom left of your screen to toggle some ambient whale noises while you read.



Thank you to the [Glacier Bay National Park & Preserve](#) and [Merrick079](#) for the sounds behind this easter egg.

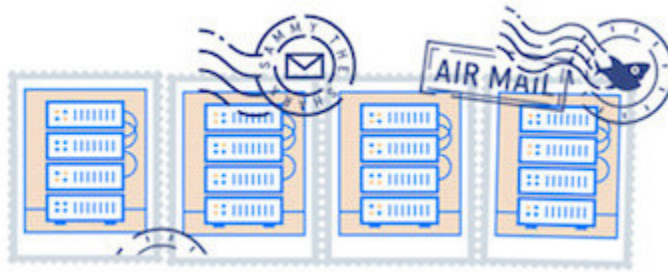


Interested in whales, protecting them, and their connection to helping prevent climate change? We recommend checking out the [Whale and Dolphin Conservation](#).

[Reset easter egg to be discovered again](#) / [Permanently dismiss and hide easter egg](#)

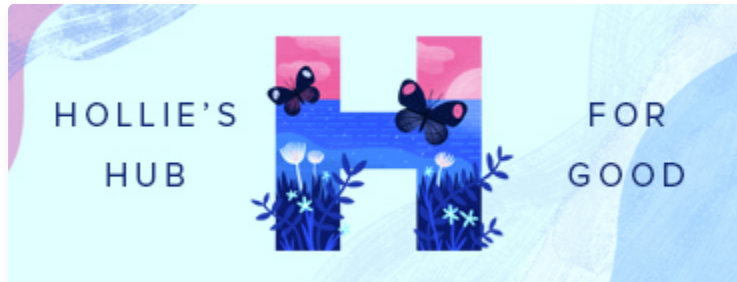


COOKIE PREFERENCES



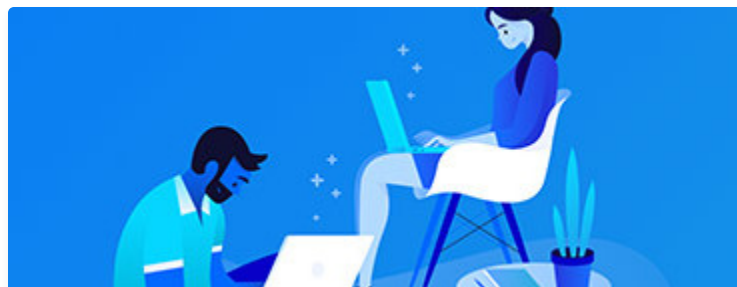
GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a
Newsletter.



HOLLIE'S HUB FOR GOOD

Working on improving health and
education, reducing inequality,
and spurring economic growth?
We'd like to help.



BECOME A CONTRIBUTOR

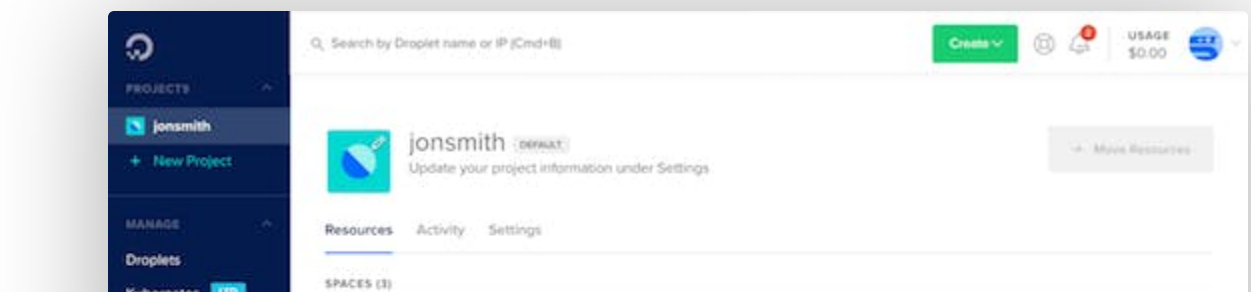
You get paid; we donate to tech
nonprofits.



Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you’re running one virtual machine or ten thousand.

[Learn More](#)



Company

[About](#)
[Leadership](#)
[Blog](#)
[Careers](#)
[Customers](#)
[Partners](#)
[Channel Partners](#)
[Referral Program](#)
[Affiliate Program](#)
[Press](#)
[Legal](#)
[Security](#)
[Investor Relations](#)
[DO Impact](#)

Products

[Products Overview](#)
[Droplets](#)
[Kubernetes](#)
[App Platform](#)
[Functions](#)
[Cloudways](#)
[Managed Databases](#)
[Spaces](#)
[Marketplace](#)
[Load Balancers](#)
[Block Storage](#)
[Tools & Integrations](#)

Community

[Tutorials](#)
[Q&A](#)
[CSS-Tricks](#)
[Write for DONations](#)
[Currents Research](#)
[Hatch Startup Program](#)
[deploy by DigitalOcean](#)
[Shop Swag](#)
[Research Program](#)
[Open Source](#)
[Code of Conduct](#)
[Newsletter Signup](#)

Solutions

[Website Hosting](#)
[VPS Hosting](#)
[Web & Mobile Apps](#)
[Game Development](#)
[Streaming](#)
[VPN](#)
[SaaS Platforms](#)
[Cloud Hosting for Blockchain](#)
[Startup Resources](#)

Contact

[Support](#)
[Sales](#)
[Report Abuse](#)
[System Status](#)
[Share your ideas](#)

[API](#)

[Meetups](#)

[Pricing](#)

[Documentation](#)

[Release Notes](#)

[Uptime](#)

© 2023 DigitalOcean, LLC. All rights reserved.

