HOW TO BUILD DASHBOARD

Practical Scenarios

Version X.X

MM/DD/YYYY

Document Number: <document’s configuration item control number>

Contract Number: <current contract number of company maintaining document>

**Table of Contents**

[1. Requirements v](#__RefHeading___Toc1549_2114243365)

[1.1 Software v](#__RefHeading___Toc505_3504440665)

[1.2 Commands v](#__RefHeading___Toc507_3504440665)

[1.3 Confirmation vi](#__RefHeading___Toc509_3504440665)

[2. Why do we need dashboard? vii](#__RefHeading___Toc564_3256715104)

[3. Why do we need Docker? viii](#__RefHeading___Toc467_3256715104)

[3.1 Faster development process x](#__RefHeading___Toc544_3256715104)

[3.2 Handy application encapsulation x](#__RefHeading___Toc546_3256715104)

[3.3 Same behavior on local machine / dev / staging / production servers x](#__RefHeading___Toc548_3256715104)

[3.4 Easy and clear monitoring xi](#__RefHeading___Toc550_3256715104)

[3.5 Easy to scale xi](#__RefHeading___Toc552_3256715104)

[3.6 Supported platforms xi](#__RefHeading___Toc554_3256715104)

[4. Terminology xii](#__RefHeading___Toc477_3256715104)

[5. Example 1: flask compose sample (hello world) xiii](#__RefHeading___Toc518_3256715104)

[5.1 Required Software xiii](#__RefHeading___Toc543_2486985292)

[5.2 Files to be created in the app xiii](#__RefHeading___Toc545_2486985292)

[5.3 Create the Parent Directory xiii](#__RefHeading___Toc584_2486985292)

[5.3.1 Flask Application xiii](#__RefHeading___Toc586_2486985292)

[5.4 Template file (Page that being displayed on browser) xiv](#__RefHeading___Toc588_2486985292)

[5.4.1 Requirements File xv](#__RefHeading___Toc590_2486985292)

[5.4.2 Docker File xv](#__RefHeading___Toc592_2486985292)

[5.4.3 Docker Compose File xv](#__RefHeading___Toc594_2486985292)

[5.5 Build and Run the Service using Docker Compose xvi](#__RefHeading___Toc596_2486985292)

[5.5.1 Output xvi](#__RefHeading___Toc898_2789233850)

[6. Example 2: flask compose sample (seeding) xix](#__RefHeading___Toc520_3256715104)

[6.1 Required Software xix](#__RefHeading___Toc543_24869852921)

[6.2 Files to be created in the app xix](#__RefHeading___Toc545_24869852921)

[6.3 Create the Parent Directory xxi](#__RefHeading___Toc584_24869852921)

[6.3.1 Flask Application xxi](#__RefHeading___Toc586_24869852921)

[6.3.2 Template file (Page that being displayed on browser) xxii](#__RefHeading___Toc1563_2114243365)

[6.3.3 Template file (Page that being displayed on browser) xxiii](#__RefHeading___Toc1565_2114243365)

[Requirements File xxiii](#__RefHeading___Toc1536_2114243365)

[Models File xxiv](#__RefHeading___Toc1538_2114243365)

[Seeds File xxiv](#__RefHeading___Toc1540_2114243365)

[Docker File (root) xxv](#__RefHeading___Toc592_24869852921)

[Docker File (seed) xxv](#__RefHeading___Toc1544_2114243365)

[Docker Compose File (root) xxvi](#__RefHeading___Toc594_24869852921)

[Seed Data (Json file) xxvii](#__RefHeading___Toc1546_2114243365)

[Build and Run the Service using Docker Compose xxviii](#__RefHeading___Toc596_24869852921)

[Output xxviii](#__RefHeading___Toc898_27892338501)

[Docker way xxx](#__RefHeading___Toc530_3256715104)

[Conclusion xxxi](#__RefHeading___Toc524_3504440665)

[1. References xxxii](#__RefHeading___Toc585_3256715104)

## **Requirements**

As a Product Owner, I struggled to decide which way to go, so we decided to ask other users for opinion. For the next few days, my team paired up and interviewed a group of users – every pair had to explain both designs to a random user, making sure we ask different users every time. This was an excellent exercise – our users felt that they had

Our project and/or scenario is to build centralized dashboard that will give single page view of all necessary information required by product owner. Product owner is responsible for entire project and needs to know lot many details than any one like developer and/or business analysts.

REFER:

<http://evo-cloud.github.io/hmake/quickguide/install/>

Here is the list of all the software's that we need to install to begin with:

### **Software**

All the software's:

* docker (1.6.0 or above)
* docker-compose (1.3.1+)
* docker-machine
* python 3.6 or above
* virtualenv (used by python)
* Flask (used by python)
* Linux VM - (We used Ubuntu 18.04 64 bit)
* git (for version control)
* skype (for communication)

### **Commands**

Here is the list of all commands:

sudo apt update

sudo apt upgrade

sudo apt dist-upgrade

sudo snap install docker

sudo apt install docker.io

sudo apt install docker-compose

sudo apt install python3

sudo apt install python-pip

sudo apt-get install apt-transport-https ca-certificates curl software-properties-common

sudo apt install virtualenv python3-flask

base=https://github.com/docker/machine/releases/download/v0.14.0

curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine

sudo install /tmp/docker-machine /usr/local/bin/docker-machine

sudo usermod -aG docker `$(whoami)`

### **Confirmation**

Confirming versions:

* $ docker --version
* Docker version 17.12.1-ce, build 7390fc6
* $ docker-compose --version
* docker-compose version 1.17.1, build unknown
* $ docker-machine --version
* docker-machine version 0.14.0, build 89b8332
* $ pip --version
* pip 9.0.1 from /usr/lib/python2.7/dist-packages (python 2.7)
* $ virtualenv --version
* 16.0.0
* $ flask --version
* Flask 1.0.2
* Python 2.7.15rc1 (default, Apr 15 2018, 21:51:34)
* [GCC 7.3.0]
* $ python3 --version
* Python 3.6.6

## **Why do we need dashboard?**

As a Product Owner, I struggled to decide which way to go, so we decided to ask other users for opinion. For the next few days, my team paired up and interviewed a group of users – every pair had to explain both designs to a random user, making sure we ask different users every time. This was an excellent exercise – our users felt that they had influence on the product development and we learnt a lot about the users. Win-win!

What we have here is a simple dashboard that will single single view of the project to the Product Owner:

Key benefits:

* Single view of builds and releases
* Single view of tests conducted
* Single view of stories completed.

## **Why do we need Docker?**

The short list of benefits includes:

* Faster development process
* Handy application encapsulation
* The same behavior on local machine / dev / staging / production servers
* Easy and clear monitoring
* Easy to scale

The following table gives a direct comparison between virtual machines and containers:

|  |  |
| --- | --- |
| Virtual Machines (VMs) | Containers |
| Represents hardware-level virtualization | Represents operating system virtualization |
| Heavyweight | Lightweight |
| Slow provisioning | Real-time provisioning and scalability |
| Limited performance | Native performance |
| Fully isolated and hence more secure | Process-level isolation and hence less secure |

### **Faster development process**

There is no need to install 3rd-party apps like PostgreSQL, Redis, Elasticsearch on the system – you can run it in containers. Docker also gives you the ability to run different versions of same application simultaneously. For example, say you need to do some manual data migration from an older version of Postgres to a newer version. You can have such a situation in micro-service architecture when you want to create a new micro-service with a new version of the 3rd-party software.

It could be quite complex to keep two different versions of the same app on one host OS. In this case, Docker containers could be a perfect solution – you receive isolated environments for your applications and 3rd-parties.

### **Handy application encapsulation**

You can deliver your application in one piece. Most programming languages, frameworks and all operating systems have their own packaging managers. And even if your application can be packed with its native package manager, it could be hard to create a port for another system.

Docker gives you a unified image format to distribute you applications across different host systems and cloud services. You can deliver your application in one piece with all the required dependencies (included in an image) ready to run.

### **Same behavior on local machine / dev / staging / production servers**

Docker can’t guarantee 100% dev / staging / production parity, because there is always the human factor. But it reduces to almost zero the probability of error caused by different versions of operating systems, system-dependencies, etc.

With right approach to building Docker images, your application will use the same base image with the same OS version and the required dependencies.

### **Easy and clear monitoring**

Out of the box, you have a unified way to read log files from all running containers. You don't need to remember all the specific paths where your app and its dependencies store log files and write custom hooks to handle this.   
You can integrate an [external logging driver](https://docs.docker.com/config/containers/logging/configure/" \l "supported-logging-drivers) and monitor your app log files in one place.

### **Easy to scale**

A correctly wrapped application will cover most of the [Twelve Factors](https://12factor.net/). By design, Docker forces you follow its core principles, such as configuration over environment variables, communication over TCP/UDP ports, etc. And if you’ve done your application right, it will be ready for scaling not only in Docker.

### **Supported platforms**

Docker’s native platform is Linux, as it’s based on features provided by the Linux kernel. However, you can still run it on macOS and Windows. The only difference is that on macOS and Windows, Docker is encapsulated into a tiny virtual machine. At the moment, Docker for macOS and Windows has reached a significant level of usability and feels more like a native app.

## **Terminology**

* **Container** – a running instance that encapsulates required software. Containers are always created from images. A container can expose ports and volumes to interact with other containers or/and the outer world. Containers can be easily killed / removed and re-created again in a very short time. Containers don't keep state.
* **Image** – the basic element for every container. When you create an image, every step is cached and can be reused ([Copy On Write model](https://en.wikipedia.org/wiki/Copy-on-write)). Depending on the image, it can take some time to build. Containers, on the other hand, can be started from images right away.
* **Port** – a TCP/UDP port in its original meaning. To keep things simple, let’s assume that ports can be exposed to the outer world (accessible from the host OS) or connected to other containers – i.e., accessible only from those containers and invisible to the outer world.
* **Volume** – can be described as a shared folder. Volumes are initialized when a container is created. Volumes are designed to persist data, independent of the container’s lifecycle.
* **Registry** – the server that stores Docker images. It can be compared to Github – you can pull an image from the registry to deploy it locally, and push locally built images to the registry.
* [**Docker Hub**](https://hub.docker.com/explore/) – a registry with web interface provided by Docker Inc. It stores a lot of Docker images with different software. Docker Hub is a source of the "official" Docker images made by the Docker team or in cooperation with the original software manufacturer (it doesn't necessary mean that these "original" images are from official software manufacturers). Official images list their potential vulnerabilities. This information is available to any logged-in user. There are both free and paid accounts available. You can have one private image per account and an infinite amount of public images for free. [**Docker Store**](https://store.docker.com/search?type=image&source=verified) – a service very similar to Docker Hub. It's a marketplace with ratings, reviews, etc. My personal opinion is that it's marketing stuff. I'm totally happy with Docker Hub.

## Example 1: flask compose sample (hello world)

In this tutorial you will learn how to create a simple Flask App with MongoDB integration, deploy and run it inside a docker container using docker Compose. Link to refer is => <http://containertutorials.com/docker-compose/flask-mongo-compose.html>

### **Required Software**

* docker (1.6.0 or above)
* docker-compose (1.3.1+)
* python 2.7 or above
* Linux VM - (We used ubuntu 14.04 64 bit)

### **Files to be created in the app**

├── app.py

├── docker-compose.yml

├── Dockerfile

├── README.md

├── requirements.txt

└── templates

└── todo.html

### **Create the Parent Directory**

Create a parent directory flask\_mongo\_compose\_sample

$ mkdir flask\_compose\_sample

$ cd flask\_compose\_sample

### **Flask Application**

Create a new file app.py and add the following python code

**import** **os**

**from** **flask** **import** Flask, redirect, url\_for, request, render\_template

**from** **pymongo** **import** MongoClient

app = Flask(\_\_name\_\_)

client = MongoClient(

os.environ['DB\_PORT\_27017\_TCP\_ADDR'],

27017)

db = client.tododb

**@app.route**('/')

**def** todo():

\_items = db.tododb.find()

items = [item **for** item **in** \_items]

**return** render\_template('todo.html', items=items)

**@app.route**('/new', methods=['POST'])

**def** new():

item\_doc = {

'name': request.form['name'],

'description': request.form['description']

}

db.tododb.insert\_one(item\_doc)

**return** redirect(url\_for('todo'))

**if** \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0', debug=True)

### **Template file (Page that being displayed on browser)**

This file todo.html contains the Jinja template to display HTML. These will be stored in templatesfolder

<form action="/new" method="POST">

<input type="text" name="name"></input>

<input type="text" name="description"></input>

<input type="submit"></input>

</form>

{% for item in items %}

<h1> {{ item.name }} </h1>

<p> {{ item.description }} <p>

{% endfor %}

### **Requirements File**

Requirements file states the software required to be installed in the container. Create a file requirements.txt inside web folder

flask

pymongo

### **Docker File**

This file is needed to create a docker image and deploy it

FROM python:2.7

ADD . /todo

WORKDIR /todo

RUN pip install -r requirements.txt

### **Docker Compose File**

Go to the parent directory flask\_mongo\_compose\_sample and create a file docker-compose.yml. For each service there is a parent tag and child tags which specify

web

* Builds from the Dockerfile in the current directory.
* Forwards the exposed port 5000 on the container to port 5000 on the host machine..
* Mounts the current directory on the host to /todo inside the container allowing you to modify the code without having to rebuild the image.
* Links to the container name db which is the MongoDB container

db

* Creates a standard MongoDB container from the image mongo:3.0.2

web:

build: .

container\_name: "web"

command: python -u app.py

ports:

- "5000:5000"

volumes:

- .:/todo

links:

- db

db:

image: mongo:3.0.2

container\_name: "mongodb"

ports:

- "27017:27017"

command: mongod --smallfiles --logpath=/dev/null # --quiet

### **Build and Run the Service using Docker Compose**

Run the following command to build the docker image flask\_mongo\_compose\_sample from web directory and deploy is as a service

$ docker-compose build --force-rm

$ docker-compose up

You can go to the browser and open the url http://localhost:5000 to see the HTML rendered

### **Output**

**~/Public/flask\_compose\_sample$ docker-compose build --force-rm**

db uses an image, skipping

Building web

Step 1/4 : FROM python:2.7

---> 4ee4ea2f0113

Step 2/4 : ADD . /todo

---> Using cache

---> 49c2dabb5b22

Step 3/4 : WORKDIR /todo

---> Using cache

---> 9395ff6be88b

Step 4/4 : RUN pip install -r requirements.txt

---> Using cache

---> d5d19830267e

Successfully built d5d19830267e

Successfully tagged flaskcomposesample\_web:latest

**~/Public/flask\_compose\_sample$ docker-compose up**

Starting mongodb ...

Starting mongodb ... done

Starting web ...

Starting web ... done

Attaching to mongodb, web

web | \* Serving Flask app "app" (lazy loading)

web | \* Environment: production

web | WARNING: Do not use the development server in a production environment.

web | Use a production WSGI server instead.

web | \* Debug mode: on

web | \* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)

web | \* Restarting with stat

web | \* Debugger is active!

web | \* Debugger PIN: 120-130-453

## Example 2: flask compose sample (seeding)

In this tutorial you will learn how to create a simple Flask App with MongoDB integration, deploy and run it inside a docker container using docker Compose. Link to refer is => <http://containertutorials.com/docker-compose/flask-mongo-compose.html>

### **Required Software**

* docker (1.6.0 or above)
* docker-compose (1.3.1+)
* python 2.7 or above
* Linux VM - (We used ubuntu 14.04 64 bit)

### **Files to be created in the app**

<Ignore Blue folders>

**.**

├── **1.27.2-shared.sock**

├── app.py

├── **Backups**

│   └── workspaces.json

├── **Cache**

│   ├── data\_0

│   ├── data\_1

│   ├── data\_2

│   ├── data\_3

│   ├── f\_000001

│   ├── f\_000002

│   └── index

├── **CachedData**

│   └── **f46c4c469d6e6d8c46f268d1553c5dc4b475840f**

├── **CachedExtensions**

│   ├── builtin

│   └── user

├── Cookies

├── Cookies-journal

├── docker-compose.yml

├── Dockerfile

├── **GPUCache**

│   ├── data\_0

│   ├── data\_1

│   ├── data\_2

│   ├── data\_3

│   └── index

├── HOW TO BUILD DASHBOARD.docx

├── languagepacks.json

├── **Local Storage**

│   ├── file\_\_0.localstorage

│   └── file\_\_0.localstorage-journal

├── **logs**

│   ├── **20180930T230020**

│   └── **20181001T063854**

├── machineid

├── models.py

├── models.pyc

├── mongodb.dealers.json

├── mongorun.sh

├── Preferences

├── README.md

├── requirements.txt

├── seed.py

├── storage.json

├── **templates**

│   ├── todo.html

│   └── users.html

└── **User**

├── settings.json

├── **snippets**

└── **workspaceStorage**

14 directories, 38 files

### **Create the Parent Directory**

Create a parent directory flask\_mongo\_compose\_sample

$ mkdir flask\_compose\_sample

$ cd flask\_compose\_sample

### **Flask Application**

Create a new file app.py and add the following python code

import os

from flask import Flask, redirect, url\_for, request, render\_template

from pymongo import MongoClient

from random import randint

from pprint import pprint

import testdata

#Step 1: Start app

app = Flask(\_\_name\_\_)

#Step 2: Connect to MongoDB - Note: Change connection string as needed

client = MongoClient(

os.environ['DB\_PORT\_27017\_TCP\_ADDR'],

27017)

@app.route('/')

def todo():

db = client.tododb

\_items = db.tododb.find()

items = [item for item in \_items]

return render\_template('todo.html', items=items)

@app.route('/users')

def users():

db = client.tododb

\_users = db.users.find()

users = [users for users in \_users]

return render\_template('users.html', users=users)

@app.route('/new', methods=['POST'])

def new():

db = client.tododb

item\_doc = {

'name': request.form['name'],

'description': request.form['description']

}

db.tododb.insert\_one(item\_doc)

return redirect(url\_for('todo'))

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0', debug=True)

### T**emplate file (Page that being displayed on browser)**

This file todo.html contains the Jinja template to display HTML. These will be stored in templatesfolder

<form action="/new" method="POST">

<input type="text" name="name"></input>

<input type="text" name="description"></input>

<input type="submit"></input>

</form>

{% for item in items %}

<h1> {{ item.name }} </h1>

<p> {{ item.description }} <p>

{% endfor %}

### **Template file (Page that being displayed on browser)**

1. This file users.html contains the Jinja template to display HTML. These will be stored in templatesfolder
2. <form action="/new" method="POST">

<input type="text" name="firstname"></input>

<input type="text" name="lastname"></input>

<input type="text" name="address"></input>

<input type="submit"></input>

</form>

{% for user in users %}

<h1> {{ user.firstname }} </h1>

<p> {{ user.lastname }} <p>

{% endfor %}

### **Requirements File**

Requirements file states the software required to be installed in the container. Create a file requirements.txt inside web folder

flask

pymongo

python-testdata

### **Models File**

Create a new file models.py and add the following python code

# Models declared

class Users(testdata.DictFactory):

id = testdata.CountingFactory(10)

firstname = testdata.FakeDataFactory('firstName')

lastname = testdata.FakeDataFactory('lastName')

address = testdata.FakeDataFactory('address')

age = testdata.RandomInteger(10, 30)

gender = testdata.RandomSelection(['female', 'male'])

### **Seeds File**

Create a new file seeds.py and add the following python code

import os

from pymongo import MongoClient

from random import randint

from pprint import pprint

import testdata

from models import Users

#Step 2: Connect to MongoDB - Note: Change connection string as needed

client = MongoClient(

os.environ['DB\_PORT\_27017\_TCP\_ADDR'],

27017)

# Issue the serverStatus command and print the results

db = client.tododb

def add\_users():

for user in Users().generate(10): # let say we only want 10 users

db.tododb.insert\_one(user)

print('finished creating business reviews')

if \_\_name\_\_ == '\_\_main\_\_':

add\_users()

### **Docker File (root)**

This file is needed to create a docker image and deploy it

FROM python:2.7

ADD . /todo

WORKDIR /todo

RUN chmod +x ./mongorun.sh

# Expose port #27017 from the container to the host

EXPOSE 27017

RUN pip install -r requirements.txt

ENTRYPOINT ["sh", "./mongorun.sh"]

### **Docker File (seed)**

This file is needed to create a docker image and deploy it

FROM mongo:3.0.2

COPY data.json /data.json

CMD mongoimport --db tododb --collection users --type json --file /data.json --jsonArray

### **Docker Compose File (root)**

Go to the parent directory flask\_mongo\_compose\_sample and create a file docker-compose.yml. For each service there is a parent tag and child tags which specify

web

* Builds from the Dockerfile in the current directory.
* Forwards the exposed port 5000 on the container to port 5000 on the host machine..
* Mounts the current directory on the host to /todo inside the container allowing you to modify the code without having to rebuild the image.
* Links to the container name db which is the MongoDB container

seed

* Creates a standard MongoDB container from the image mongo:3.0.2
* Data.json file will have all necessary details

db

* Creates a standard MongoDB container from the image mongo:3.0.2

web:

build: .

container\_name: "web"

command: python -u app.py

ports:

- "5000:5000"

volumes:

- .:/todo

links:

- seed

seed:

build: ./mongo-seed

links:

- db

db:

image: mongo:3.0.2

container\_name: "mongodb"

ports:

- "27017:27017"

command: mongod --smallfiles --logpath=/dev/null # --quiet

### **Seed Data (Json file)**

Following is the input data that will be used for seeding

[

{

"name": "Joe Smith",

"email": "jsmith@gmail.com",

"age": 40,

"admin": false

},

{

"name": "Jen Ford",

"email": "jford@gmail.com",

"age": 45,

"admin": true

}

]

### **Build and Run the Service using Docker Compose**

Run the following command to build the docker image flask\_mongo\_compose\_sample from web directory and deploy is as a service

$ docker system prune

$ docker-compose build --force-rm --no-cache

$ docker-compose up

You can go to the browser and open the url http://localhost:5000 to see the HTML rendered

### **Output**

**~/Public/flask\_compose\_sample$ docker-compose build --force-rm**

db uses an image, skipping

Building web

Step 1/4 : FROM python:2.7

---> 4ee4ea2f0113

Step 2/4 : ADD . /todo

---> Using cache

---> 49c2dabb5b22

Step 3/4 : WORKDIR /todo

---> Using cache

---> 9395ff6be88b

Step 4/4 : RUN pip install -r requirements.txt

---> Using cache

---> d5d19830267e

Successfully built d5d19830267e

Successfully tagged flaskcomposesample\_web:latest

**~/Public/flask\_compose\_sample$ docker-compose up**

Starting mongodb ...

Starting mongodb ... done

Starting web ...

Starting web ... done

Attaching to mongodb, web

web | \* Serving Flask app "app" (lazy loading)

web | \* Environment: production

web | WARNING: Do not use the development server in a production environment.

web | Use a production WSGI server instead.

web | \* Debug mode: on

web | \* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)

web | \* Restarting with stat

web | \* Debugger is active!

web | \* Debugger PIN: 120-130-453

## Docker way

Docker has some restrictions and requirements, depending on the architecture of your system (applications that you pack into containers). You can ignore these requirements or find some workarounds, but in this case, you won't get all the benefits of using Docker. My strong advice is to follow these recommendations:

* **1 application = 1 container**.
* Run the process in the **foreground** (don't use systemd, upstart or any other similar tools).
* **Keep data out of containers** – use volumes.
* **Do not use SSH** (if you need to step into container, you can use the docker exec command).
* **Avoid manual configurations** (or actions) inside container.

## Conclusion

To summarize this tutorial, alongside with IDE and Git, Docker has become a must-have developer tool. It's a production-ready tool with a rich and mature infrastructure.

Docker can be used on all types of projects, regardless of size and complexity. In the beginning, you can start with [compose](https://docs.docker.com/compose/overview/) and [Swarm](https://docs.docker.com/engine/swarm/). When the project grows, you can migrate to cloud services like [Amazon Container Services](https://aws.amazon.com/containers/) or [Kubernetes](https://kubernetes.io/).

Like standard containers used in cargo transportation, wrapping your code in Docker containers will help you build faster and more efficient CI/CD processes. This is not just another technological trend promoted by a bunch of geeks – it's a new paradigm that is already being used in the architecture of large companies like [PayPal](https://blog.docker.com/2017/12/containers-at-paypal/), [Visa](https://blog.docker.com/2017/04/visa-inc-gains-speed-operational-efficiency-docker-enterprise-edition/), [Swisscom](https://www.docker.com/customers/swisscom-goes-400vms-20vms-docker), [General Electric](https://www.docker.com/customers/ge-uses-docker-enable-self-service-their-developers), [Splink](https://www.docker.com/customers/docker-datacenter-delivers-splunks-house-demos), etc.

## References

Some useful links for installations and lots of examples:

* [https://docs.docker.com/machine/install-machine/#install-machine-directly](https://docs.docker.com/machine/install-machine/" \l "install-machine-directly)
* <https://blog.hackingcode.io/docker-for-beginners-tutorial-installation-mac-ubuntu/>
* [https://mail.google.com/mail/u/0/#search/virtualenv/WhctKJTrZQkmgVWdmNNldCPCBLhNcKDvtQgRPWGrlDLRczXbJWhRlGHDFVsrcKTPjvFMpqV](https://mail.google.com/mail/u/0/" \l "search/virtualenv/WhctKJTrZQkmgVWdmNNldCPCBLhNcKDvtQgRPWGrlDLRczXbJWhRlGHDFVsrcKTPjvFMpqV)
* <https://web.whatsapp.com/>
* <http://containertutorials.com/docker-compose/flask-mongo-compose.html>
* <https://blog.hackingcode.io/docker-for-beginners-tutorial-installation-mac-ubuntu/>
* [https://djangostars.com/blog/what-is-docker-and-how-to-use-it-with-python/#13](https://djangostars.com/blog/what-is-docker-and-how-to-use-it-with-python/" \l "13)
* <https://runnable.com/docker/python/dockerize-your-python-application>
* <https://medium.freecodecamp.org/a-recipe-for-website-automated-tests-with-python-selenium-headless-chrome-in-docker-8d344a97afb5>
* <https://github.com/tylerharter/open-lambda>
* <https://medium.com/@cramirez92/build-a-nodejs-cinema-microservice-and-deploying-it-with-docker-part-1-7e28e25bfa8b>
* <https://azure.microsoft.com/en-us/resources/samples/?sort=0&platform=python>
* <https://pypi.org/project/jira/>
* <https://media.readthedocs.org/pdf/jira/stable/jira.pdf>
* <https://realpython.com/python-virtual-environments-a-primer/>
* <https://pencil.evolus.vn/Downloads.html>
* [https://pusher.com/tutorials/live-dashboard-python#prerequisites](https://pusher.com/tutorials/live-dashboard-python" \l "prerequisites)
* [https://docs.docker.com/machine/install-machine/#install-machine-directly](https://docs.docker.com/machine/install-machine/" \l "install-machine-directly)
* <https://productownerblog.wordpress.com/tag/dashboard/>