# **How To Deploy Docker Container with Ansible**

**Prerequisites**

Before you begin this guide you'll need the following:

* One Debian 8 server set up by following the [Initial Server Setup with Debian 8 tutorial](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-debian-8) , including a sudo non-root user.
* Docker installed on the server.
* Ansible 2.4 or higher installed on your local machine by following this [How to Install and Configure Ansible on Ubuntu 18.04](https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-ansible-on-ubuntu-18-04).

*Make sure that your local machine can communicate with the server over SSH. Use SSH key instead of typed password for security reason.*

**Step 1 — Build a docker image with Dockerfile**

There are some number of way to build a docker image but we recommend that you build image using a definition file called a Dockerfile.

First, create a directory on the workstation and a Dockerfile within the created directory.

mkdir docker

cd docker

touch Dockerfile

For our example, we are going to build a Docker image that contains a simple web server.

Edit Dockerfile with nano and add the following content:

*# Version: 0.0.1*

**FROM** debian:8.5

**MAINTAINER** maintainer\_name "maintainer\_email"

**RUN** apt-get update

**RUN** apt-get install -y nginx

**RUN** echo 'Default page. Nginx is in your container. ' \

>/usr/share/nginx/html/index.html

**EXPOSE** 80

Now that we have a Dockerfile, Ansible can help us to ease the building and the deployment on the server.

**Step 2 — Edit inventory file**

Inventory file contains IP addresses or domain names where we want deploy container. Add this file in your project:

touch ../hosts

Edit the hosts file with nano and add the following content within this file.

[webserver]

your\_server\_ip

Now, check if your local machine can communicate with the server via Ansible. Type the following command in terminal:

ansible webserver -m ping -i ../hosts

The output of this command should be:

your\_server\_ip | success >> {

"changed": false,

"ping": "pong"

}

That means you can reach your server over SSH and you can use Ansible to configure it.

**Step 3 — Edit Ansible playbook**

First, change the working directory to the project root directory and create a new Ansible playbook. The directory layout should look like:

docker\_project/

main.yml

hosts

docker/

Dockerfile

Next, add a task to Ansible playbook to install docker-py on the Docker host. Ansible’s Docker integration requires you to install this python library. We use pip to install the required python library. So we add also a task that install pip before docker-py.

Package installation and Docker require a sudo privilege on the server. You can handle this privilege in Ansible by specifying the sudo user name via remote\_user and adding become: yes and become\_method: sudo directives.

Edit the playbook main.yml with nano and add the following:

---

- hosts: webserver

remote\_user: remote\_username

become: yes

become\_method: sudo

tasks:

- name: Install pip

apt: name=python-pip state=present

- name: install docker-py package

pip: name=docker-py

The playbook uses the docker module to build an image. We provide a name for the image, the path to the Dockerfile (in our case, inside the docker directory), and tell Ansible via the state parameter whether the image should be present ,absent , or build.

So, let's add the following content to main.yml playbook:

...

- name: Build Docker image from Dockerfile

docker\_image:

name: web

path: docker

state: build

With this playbook, you only built the new web image but this image never run. You need to add another task to the playbook for running it. Provide the image name, the path to the Dockerfile and use state paramater to tell Ansible the image should be running(via running). Use the latest image by tagging its name with :latest. The playbook should look like this:

---

- hosts: webserver

remote\_user: remote\_username

become: yes

become\_method: sudo

tasks:

- name: Install pip

apt: name=python-pip state=present

- name: install docker-py

pip: name=docker-py

- name: Build Docker image from Dockerfile

docker\_image:

name: web

path: docker

state: build

- name: Running the container

docker\_container:

image: web:latest

path: docker

state: running

Now add a task to check if the new container works fine on the server. This task runs docker ps on the server.

Finally, your playbook become:

---

- hosts: webserver

remote\_user: linx

become: yes

become\_method: sudo

tasks:

- name: Install pip

apt: name=python-pip state=present

- name: install docker-py

pip: name=docker-py

- name: Build Docker image from Dockerfile

docker\_image:

name: web

path: docker

state: build

- name: Running the container

docker\_container:

image: web:latest

path: docker

state: running

- name: Check if container is running

shell: docker ps

Our Ansible playbook is now ready for use.

**Step 3 — Running the playbook**

Run the playbook by using the following command on your workstation:

ansible-playbook -i hosts main.yml --ask-become-pass

You are now prompted to type the remote sudo user password.

To check if our web server container works fine, type:

curl your\_server\_ip:your\_docker\_port

This should return the Nginx default page created by the Dockerfile.

**Default** page. Nginx **is** **in** your container.