
Portainer Documentation

Release 1.10.2

Portainer.io

December 13, 2016

1	Deployment	3
1.1	Quick start	3
1.2	Connect to a remote Docker engine	3
1.3	Connect to a local Docker engine	3
1.4	Connect to a Swarm cluster	4
1.5	Connect to a Docker engine with TLS enabled	4
1.6	Without Docker	4
2	Configuration	7
2.1	Hiding specific containers	7
2.2	Use your own logo	7
2.3	Use your own templates	7
2.4	Available flags	8
3	Templates	9
3.1	Template definition format	9
3.2	Build and host your own templates	11
4	Contribute	13
4.1	Build Portainer locally	13
4.2	Contribution guidelines	13
5	FAQ	15
5.1	How can I configure my reverse proxy to serve Portainer?	15
5.2	How can I setup basic HTTP authentication in front of Portainer?	15
5.3	How can I configure my reverse proxy to serve Portainer using HAProxy?	16
5.4	Exposed ports in the container view redirects me to 0.0.0.0, what can I do?	17
5.5	Why does Portainer not work in Kitematic?	17

Portainer is a simple management solution for Docker.

It consists of a web UI that allows you to easily manage your Docker containers, images, networks and volumes.

Contents:

Deployment

Portainer is built to run on Docker and is really simple to deploy.

Note: the following instructions target Docker for Linux, if your target is Docker for Windows please use the `portainer/portainer:windows` Docker image instead of the `portainer/portainer` Docker image.

Portainer deployment scenarios can be executed on both platforms unless specified.

1.1 Quick start

The most common deployment scenario is to deploy Portainer to manage a remote Docker host, it's as simple as:

```
$ docker run -d -p 9000:9000 portainer/portainer -H tcp://<REMOTE_HOST>:<REMOTE_PORT>
```

Voilà, you can now access Portainer by pointing your web browser at `http://DOCKER_HOST:9000`

Ensure you replace `DOCKER_HOST` with address of your Docker host where Portainer is running.

1.2 Connect to a remote Docker engine

In order to connect to a remote host, use the `-H` flag and the `tcp://` protocol:

```
$ docker run -d -p 9000:9000 portainer/portainer -H tcp://<REMOTE_HOST>:<REMOTE_PORT>
```

Ensure you replace `REMOTE_HOST` and `REMOTE_PORT` with the address/port of the Docker engine you want to manage.

1.3 Connect to a local Docker engine

Docker for Linux only

By default, Portainer will try to connect to the local Docker engine using the unix socket path at `/var/run/docker.sock`

You will need to bind mount the Docker socket to manage a local Docker engine:

```
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer
```

If your host is using SELinux, you'll need to pass the `--privileged` flag to the Docker run command:

```
$ docker run -d -p 9000:9000 --privileged -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer
```

You can also specify an alternative path to the Docker socket using the `-H` flag combined with the `unix://` protocol:

```
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/docker/docker.sock portainer/portainer -H unix://var/run/docker.sock
```

1.4 Connect to a Swarm cluster

If you want to manage a cluster created with Docker Swarm or using the swarm mode of Docker 1.12 all you need to do is to add the `--swarm` flag to the portainer command line.

Note: Ensure you connect to either a *primary* node when connecting to a Docker Swarm cluster or a *manager* node when connecting to a cluster created with Docker swarm mode.

For example, when connecting to a remote Swarm node:

```
$ docker run -d -p 9000:9000 portainer/portainer -H tcp://<REMOTE_HOST>:<REMOTE_PORT> --swarm
```

If you're using swarm mode, you can also deploy it as a service in your cluster:

```
$ docker service create \
  --name portainer \
  --publish 9000:9000 \
  --constraint 'node.role == manager' \
  --mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \
  portainer/portainer \
  --swarm
```

1.5 Connect to a Docker engine with TLS enabled

If your Docker engine is protected using TLS, you'll need to ensure that you have access to CA, the certificate and the public key used to access your Docker engine.

You can then use the `--tlsverify` flag to enable TLS communication with the Docker API.

Portainer will try to use the following paths to the files specified previously:

- CA: `/certs/ca.pem`
- certificate: `/certs/cert.pem`
- public key: `/certs/key.pem`

You must ensure these files are present in the container using a bind mount:

```
$ docker run -d -p 9000:9000 -v /path/to/certs:/certs portainer/portainer -H tcp://<DOCKER_HOST>:<DOCKER_PORT> --tlsverify
```

You can also use the `--tlscacert`, `--tlscert` and `--tlskey` flags if you want to change the default path to the CA, certificate and key file respectively:

```
$ docker run -d -p 9000:9000 -v /path/to/certs:/certs portainer/portainer -H tcp://<DOCKER_HOST>:<DOCKER_PORT> --tlscacert /certs/ca.pem --tlscert /certs/cert.pem --tlskey /certs/key.pem
```

1.6 Without Docker

Portainer binaries are available on each release page: [Portainer releases](#)

Download and extract the binary to a location on disk:

```
$ cd /opt
$ wget https://github.com/portainer/portainer/releases/download/1.10.2/portainer-1.10.2-linux-amd64.tar.gz
$ tar xvpfz portainer-1.10.2-linux-amd64.tar.gz
```

Then just use the portainer binary as you would use CLI flags with Docker.

```
$ /opt/portainer/portainer -H tcp://DOCKER_HOST:DOCKER_PORT
```

You can use the `-p` flag to serve Portainer on another port:

```
$ /opt/portainer/portainer -H tcp://DOCKER_HOST:DOCKER_PORT -p 8080
```

Configuration

Portainer can be easily tuned using CLI flags.

2.1 Hiding specific containers

Portainer allows you to hide containers with a specific label by using the `-l` flag.

For example, take a container started with the label `owner=acme` (note that this is an example label, you can define your own labels):

```
$ docker run -d --label owner=acme nginx
```

To hide this container, simply add the `-l owner=acme` option on the CLI when starting Portainer:

```
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer -l owner=acme
```

Note that the `-l` flag can be repeated multiple times to specify multiple labels:

```
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer -l owner=acme -l another-label
```

2.2 Use your own logo

You do not like our logo? Want to make Portainer more corporate? Don't worry, you can easily switch for an external logo using the `--logo` flag:

```
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer --logo /path/to/logo.png
```

2.3 Use your own templates

Portainer allows you to rapidly deploy containers using App Templates.

By default [Portainer templates](#) will be used but you can also define your own templates.

Add the `--templates` flag and specify the external location of your templates when starting Portainer:

```
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer --templates /path/to/templates
```

For more information about hosting your own template definitions see [Templates](#)

2.4 Available flags

The following CLI flags are available:

- `--host, -H`: Docker daemon endpoint (default: `unix:///var/run/docker.sock`)
- `--bind, -p`: Address and port to serve Portainer (default: `:9000`)
- `--swarm, -s`: Docker Swarm cluster / swarm-mode support (default: `false`)
- `--tlsverify`: TLS support (default: `false`)
- `--tlscacert`: Path to the CA (default: `/certs/ca.pem`)
- `--tlscert`: Path to the TLS certificate file (default: `/certs/cert.pem`)
- `--tlskey`: Path to the TLS key (default: `/certs/key.pem`)
- `--hide-label, -l`: Hide containers with a specific label in the UI
- `--logo`: URL to a picture to be displayed as a logo in the UI, use Portainer logo if not specified
- `--templates, -t`: URL to templates (apps) definitions (default: `https://raw.githubusercontent.com/portainer/templates/master/templates.json`)

Templates

Template definitions are written in JSON.

It must consist of an array with every template definition consisting of one element.

3.1 Template definition format

A template element must be a valid `JSON` object.

Example:

```
{
  "title": "Nginx",
  "description": "High performance web server",
  "logo": "https://cloudinovasi.id/assets/img/logos/nginx.png",
  "image": "nginx:latest",
  "ports": [
    "80/tcp",
    "443/tcp"
  ]
}
```

It is composed of multiple fields, some mandatory and some optional.

3.1.1 title

Title of the template.

This field is **mandatory**.

3.1.2 description

Description of the template.

This field is **mandatory**.

3.1.3 logo

URL of the template's logo.

This field is **mandatory**.

3.1.4 image

The Docker image associated to the template. The image tag **must** be included.

This field is **mandatory**.

3.1.5 env

A JSON array describing the environment variables required by the template. Each element in the array must be a valid JSON object.

An input will be generated in the templates view for each element in the array.

This field is **optional**.

Element format:

```
{
  "name": "the name of the environment variable, as supported in the container image (mandatory)",
  "label": "label for the input in the UI (mandatory)",
  "set": "pre-defined value for the variable, will not generate an input in the UI (optional)"
}
```

Example:

```
{
  "env": [
    {
      "name": "MYSQL_ROOT_PASSWORD",
      "label": "Root password"
    },
    {
      "name": "MYSQL_USER",
      "label": "MySQL user",
      "set": "myuser"
    },
    {
      "name": "MYSQL_PASSWORD",
      "label": "MySQL password",
      "set": "mypassword"
    }
  ]
}
```

3.1.6 volumes

A JSON array describing the associated volumes of the template. Each element in the array must be a valid JSON string.

For each element in the array, a Docker volume will be created and associated when starting the container.

This field is **optional**.

Example:

```
{  
  "volumes": ["/var/lib/mysql", "/var/log/mysql"]  
}
```

3.1.7 ports

A JSON array describing the ports exposed by template. Each element in the array must be a valid JSON string specifying the port number in the container and the protocol.

Each port will be automatically bound on the host by Docker when starting the container.

This field is **optional**.

Example:

```
{  
  "ports": ["80/tcp", "443/tcp"]  
}
```

3.2 Build and host your own templates

You can build your own container that will use [Nginx](#) to serve the templates definitions.

Clone the [Portainer templates repository](#), edit the templates file, build and run the container:

```
$ git clone https://github.com/portainer/templates.git portainer-templates  
$ cd portainer-templates  
# Edit the file templates.json  
$ docker build -t portainer-templates .  
$ docker run -d -p "8080:80" portainer-templates
```

Now you can access your templates definitions at `http://docker-host:8080/templates.json`.

You can also mount the `templates.json` file inside the container, so you can edit the file and see live changes:

```
$ docker run -d -p "8080:80" -v "${PWD}/templates.json:/usr/share/nginx/html/templates.json" portainer-templates
```

Contribute

Use the following instructions and guidelines to contribute to the Portainer project.

4.1 Build Portainer locally

4.1.1 Requirements

Ensure you have [Docker](#), [Node.js](#) `>= 0.8.4` and [npm](#) installed locally.

4.1.2 Build

Checkout the project and go inside the root directory:

```
$ git checkout https://github.com/portainer/portainer.git  
$ cd portainer
```

Install the dependencies using `npm`:

```
$ npm install
```

Start a live-reload process, the local application will be updated when you save your changes:

```
$ grunt run-dev
```

Access Portainer at <http://localhost:9000>

Do not forget to [lint](#) your code:

```
$ grunt lint
```

4.2 Contribution guidelines

Please follow the contribution guidelines on [the repository](#).

5.1 How can I configure my reverse proxy to serve Portainer?

Here is a working configuration for Nginx (tested on 1.11) to serve Portainer at *myhost.mydomain/portainer*:

```
upstream portainer {
    server ADDRESS:PORT;
}

server {
    listen 80;

    location /portainer/ {
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        proxy_pass http://portainer/;
    }
    location /portainer/ws/ {
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_http_version 1.1;
        proxy_pass http://portainer/ws/;
    }
}
```

Replace `ADDRESS:PORT` with the Portainer server/container details.

5.2 How can I setup basic HTTP authentication in front of Portainer?

Here is a working configuration for Nginx (tested on 1.11 with **bcrypt** support) to serve Portainer at *myhost.mydomain/portainer* with basic HTTP authentication:

```
upstream portainer {
    server ADDRESS:PORT;
}

server {
    listen 80;

    auth_basic "myhost.mydomain/portainer";
    auth_basic_user_file /etc/nginx/conf.d/portainer.htpasswd;
}
```

```
location /portainer/ {
    proxy_http_version 1.1;
    proxy_set_header    Connection "";
    proxy_pass http://portainer/;
}
location /portainer/ws/ {
    proxy_set_header    Upgrade $http_upgrade;
    proxy_set_header    Connection "upgrade";
    proxy_http_version 1.1;
    proxy_pass http://portainer/ws/;
}
}
```

Replace ADDRESS : PORT with the Portainer server/container details.

You can generate the authentication file using the following command:

```
docker run --rm httpd htpasswd -nbB <USERNAME> <PASSWORD> /etc/nginx/conf.d/portainer.htpasswd
```

You can also checkout our pre-configured setup using Docker compose [here](#).

5.3 How can I configure my reverse proxy to serve Portainer using HAProxy?

Here is a working configuration for HAProxy to serve Portainer at *portainer.127.0.0.1.xip.io*:

```
global
    maxconn          10000
    daemon
    ssl-server-verify none
    tune.ssl.default-dh-param 2048

defaults
    mode            http
    log             global
    option          httplog
    option          dontlognull
    option          http-server-close
    option          forwardfor except 127.0.0.0/8
    option          redispatch
    retries         30
    timeout http-request 300s
    timeout queue    1m
    timeout connect  10s
    timeout client   1m
    timeout server   1m
    timeout http-keep-alive 10s
    timeout check    10s
    maxconn 10000

userlist users
    group all
    group demo
    group haproxy

listen stats
```

```

bind            *:2100
mode            http
stats           enable
maxconn         10
timeout client  10s
timeout server  10s
timeout connect 10s
timeout         queue    10s
stats           hide-version
stats           refresh 30s
stats           show-node
stats           realm Haproxy\ Statistics
stats           uri      /
stats           admin if TRUE

frontend www-http
  bind          *:80
  stats         enable
  mode          http
  option        http-keep-alive

  acl portainer  hdr_end(host)    -i portainer.127.0.0.1.xip.io

  use_backend   portainer        if portainer

backend portainer
  stats         enable
  option        forwardfor
  option        http-keep-alive
  server        portainer        127.0.0.1:9000 check

```

Note: http-keep-alive must be set for both frontend and backend

5.4 Exposed ports in the container view redirects me to 0.0.0.0, what can I do?

In order for Portainer to be able to redirect you to your Docker host IP address and not the 0.0.0.0 address, you will have to change the configuration of your Docker daemon and add the `--ip` option.

Have a look at the [Docker documentation](#) for more details.

Note that you will have to restart your Docker daemon for the changes to be taken in effect.

5.5 Why does Portainer not work in Kitematic?

Portainer has to be connected to a Docker instance either by bind-mounting the Docker socket or using the `-H` flag in the container command to specify the Docker host address. Unfortunately, Kitematic does not allow you to specify any of these at the moment.