
Portainer Documentation

Release 1.13.3

Portainer.io

Jun 20, 2017

1	Deployment	3
1.1	Quick start	3
1.2	Manage a new endpoint	3
1.3	Declare initial endpoint via CLI	4
1.4	Connect to a Swarm cluster	4
1.5	Connect to a Docker engine with TLS enabled	4
1.6	Persist Portainer data	5
1.7	Secure Portainer using SSL	5
1.8	Without Docker	6
2	Configuration	7
2.1	Disable authentication	7
2.2	Admin password	7
2.3	Hiding specific containers	8
2.4	Use your own logo	8
2.5	Use your own templates	8
2.6	Use an external endpoint source	9
2.7	Available flags	9
2.8	Deprecated flags	9
3	External endpoints	11
3.1	Endpoint definition format	11
3.2	Endpoint synchronization	12
4	Templates	15
4.1	Template definition format	15
4.2	Build and host your own templates	19
5	Contribute	21
5.1	Build Portainer locally	21
5.2	Contribution guidelines	22
6	Limitations	23
6.1	Docker	23
6.2	Swarm	23
6.3	Supported platforms	23

7	FAQ	25
7.1	How can I configure my reverse proxy to serve Portainer?	25
7.2	How can I configure my reverse proxy to serve Portainer using HAProxy?	25
7.3	Exposed ports in the container view redirects me to 0.0.0.0, what can I do?	27
7.4	I restarted Portainer and lost all my data, why?	27
7.5	How can I use a custom CSS file to customize Portainer look?	27
7.6	I am getting the error “Your session has expired” on login and cannot login. What’s wrong?	27
7.7	How can I access the Docker API on port 2375 on Windows?	27
7.8	How can I use Portainer behind a proxy?	28
7.9	How can I upgrade my version of Portainer?	28

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:

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

Portainer deployment scenarios can be executed on any platform unless specified.

Quick start

Deploying Portainer is as simple as:

```
$ docker run -d -p 9000:9000 portainer/portainer
```

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.

You'll then be prompted to specify a new password for the `admin` account. After specifying your password, you'll then be able to connect to the Portainer UI.

Manage a new endpoint

After your first authentication, Portainer will ask you information about the Docker endpoint you want to manage.

You'll have the following choices:

- **Not available for Windows Containers (Windows Server 2016)** - Manage the local engine where Portainer is running (you'll need to bind mount the Docker socket via `-v /var/run/docker.sock:/var/run/docker.sock` on the Docker CLI when running Portainer)
- Manage a remote Docker engine, you'll just have to specify the url to your Docker endpoint, give it a name and TLS info if needed

Declare initial endpoint via CLI

You can specify the initial endpoint you want Portainer to manage via the CLI, use the `-H` flag and the `tcp://` protocol to connect to a remote Docker endpoint:

```
$ 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.

You can also bind mount the Docker socket to manage a local Docker engine (**not available for Windows Containers (Windows Server 2016)**):

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

Note: 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
```

Connect to a Swarm cluster

Portainer will automatically detect if your endpoint is part of a Swarm cluster (either Docker Swarm or Swarm mode).

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.

As simple as:

```
$ docker run -d -p 9000:9000 portainer/portainer -H tcp://<SWARM_MANAGER_IP>:2375
```

Alternatively, 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 \
  -H unix:///var/run/docker.sock
```

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 upload the required files via the Portainer UI or use the `--tlsverify` flag on the CLI.

Portainer will try to use the following paths to the files specified previously (on Linux, see the configuration section for details about Windows):

- 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> --tlsverify --tlscacert /certs/myCa.pem --tlscert /
↳certs/myCert.pem --tlskey /certs/myKey.pem
```

Persist Portainer data

By default, Portainer will store its data inside the container in the `/data` folder on Linux (`C:\data` on Windows, this can be changed via CLI, see configuration).

You'll need to persist Portainer data to keep your changes after restart/upgrade of the Portainer container. You can use a bind mount to persist the data on the Docker host folder:

```
$ docker run -d -p 9000:9000 -v /path/on/host/data:/data portainer/portainer
```

On Windows:

```
$ docker run -d -p 9000:9000 -v C:\ProgramData\Portainer:C:\data portainer/portainer
```

If you deployed Portainer as a Docker Swarm service:

```
$ 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 \
  --mount type=bind,src=/path/on/host/data,dst=/data \
  portainer/portainer \
  -H unix:///var/run/docker.sock
```

Secure Portainer using SSL

By default, Portainer's web interface and API is exposed over HTTP. This is not secured, it's recommended to enable SSL in a production environment.

To do so, you can use the following flags `--ssl`, `--sslcert` and `--sslkey`:

```
$ docker run -p 443:9000 -v ~/local-certs:/certs portainer --ssl --sslcert /certs/
↳portainer.crt --sslkey /certs/portainer.key
```

You can use the following commands to generate the required files:

```
$ openssl genrsa -out portainer.key 2048
$ openssl ecparam -genkey -name secp384r1 -out portainer.key
$ openssl req -new -x509 -sha256 -key portainer.key -out portainer.crt -days 3650
```

Note that [Certbot](#) could be used as well to generate a certificate and a key.

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/1.13.3/portainer-1.13.3-linux-amd64.tar.gz
$ tar xvpfz portainer-1.13.3-linux-amd64.tar.gz
```

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

Note: Portainer will try to write its data into the `/data` folder by default. You must ensure this folder exists first.

```
$ mkdir /data
$ cd /opt
$ ./portainer/portainer
```

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

```
$ ./portainer/portainer -p :8080
```

You can change the folder used by Portainer to store its data with the `-d` flag:

```
$ ./portainer/portainer -d /opt/portainer-data
```

Portainer can be easily tuned using CLI flags.

Disable authentication

To disable Portainer internal authentication mechanism, start Portainer with the `--no-auth` flag:

```
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/  
↳portainer --no-auth
```

Admin password

Portainer allows you to specify an encrypted password from the command line for the admin account. You need to generate the encrypted password first.

You can generate an encrypted password with the following command:

```
$ htpasswd -nb -B admin <password> | cut -d ":" -f 2
```

or if your system does not provide `htpasswd` you can use a docker container with the command:

```
$ docker run --rm httpd:2.4-alpine htpasswd -nbB admin <password> | cut -d ":" -f 2
```

To specify the admin password from the command line, start Portainer with the `--admin-password` flag:

```
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/  
↳portainer --admin-password '$2y$05$qFHA1NAH0A.6oCDe1/4W.ueCWC/  
↳iTfBMXIHBI97QYfMWlMCJ7N.a6'
```

Hiding specific containers

Deprecation notice: the `-l` flag is deprecated and will be removed in future versions. Use the settings section in Portainer UI to manage hidden 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 service=secret
```

Use your own logo

Deprecation notice: the `--logo` flag is deprecated and will be removed in future versions. Use the settings section in Portainer UI to change the logo.

You do not like our logo? Want to make Portainer more corporate? Don't worry, you can easily switch for an external logo (it must be exactly 155px by 55px) using the `--logo` flag:

```
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/  
↪portainer --logo "https://www.docker.com/sites/all/themes/docker/assets/images/  
↪brand-full.svg"
```

Use your own templates

Deprecation notice: the `--templates` flag is deprecated and will be removed in future versions. Use the settings section in Portainer UI to change 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 http://my-host.my-domain/templates.json
```

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

Use an external endpoint source

Portainer gives you the option to define all the endpoints available in the UI from a JSON file.

You just need to start Portainer with the `--external-endpoints` flag and specify the path to the JSON file in the container.

Note: when using the external endpoint management, endpoint management will be disabled in the UI.

```
$ docker run -d -p 9000:9000 -v /tmp/endpoints:/endpoints portainer/portainer --
↪external-endpoints /endpoints/endpoints.json
```

For more information about the endpoint definition format see [External endpoints](#)

Available flags

The following CLI flags are available:

- `--host, -H`: Docker daemon endpoint
- `--bind, -p`: Address and port to serve Portainer (default: `:9000`)
- `--data, -d`: Directory where Portainer data will be stored (default: `/data` on Linux, `C:\data` on Windows)
- `--tlsverify`: TLS support (default: `false`)
- `--tlscacert`: Path to the CA (default: `/certs/ca.pem` on Linux, `C:\certs\ca.pem` on Windows)
- `--tlscert`: Path to the TLS certificate file (default: `/certs/cert.pem`, `C:\certs\cert.pem` on Windows)
- `--tlskey`: Path to the TLS key (default: `/certs/key.pem`, `C:\certs\key.pem` on Windows)
- `--no-analytics`: Disable analytics (default: `false`)
- `--no-auth`: Disable internal authentication mechanism (default: `false`)
- `--external-endpoints`: Enable external endpoint management by specifying the path to a JSON endpoint source in a file
- `--sync-interval`: Time interval between two endpoints synchronization requests expressed as a string, e.g. `30s`, `5m`, `1h...` as supported by the [time.ParseDuration method](#) (default: `60s`)
- `--admin-password`: Admin password in the form `admin:<hashed_password>`
- `--ssl`: Secure Portainer instance using SSL (default: `false`)
- `--sslcert`: Path to the SSL certificate used to secure the Portainer instance (default: `/certs/portainer.crt`, `C:\certs\portainer.crt` on Windows)
- `--sslkey`: Path to the SSL key used to secure the Portainer instance (default: `/certs/portainer.key`, `C:\certs\portainer.key` on Windows)

Deprecated flags

The following CLI flags are deprecated and should not be used anymore:

- `--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`)

External endpoints

External endpoint definitions are written in JSON.

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

```
[
  {
    "Name": "my-first-endpoint",
    "URL": "tcp://myendpoint.mydomain:2375"
  },
  {
    "Name": "my-second-endpoint",
    "URL": "tcp://mysecondendpoint.mydomain:2375",
    "TLS": true,
    "TLSCACert": "/tmp/ca.pem",
    "TLSCert": "/tmp/cert.pem",
    "TLSKey": "/tmp/key.pem"
  }
]
```

Endpoint definition format

An endpoint element must be a valid **JSON** object.

Example:

```
{
  "Name": "my-secure-endpoint",
  "URL": "tcp://myendpoint.mydomain:2375",
  "TLS": true,
  "TLSCACert": "/tmp/ca.pem",
  "TLSCert": "/tmp/cert.pem",
  "TLSKey": "/tmp/key.pem"
}
```

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

Name

Name of the endpoint. Used to check if an endpoint already exists in the database during a synchronization request. It will also be displayed in the UI.

This field is **mandatory**.

URL

How to reach the endpoint.

Protocol **must** be specified, only `tcp://` and `unix://` are supported at the moment. Any definition not using one of these 2 protocols will be skipped.

This field is **mandatory**.

TLS

Specify this field to true if you need to use TLS to connect to the endpoint. Defaults to `false`. When applying the true value to this field, Portainer will expect the `TLSCACertPath`, `TLSCertPath` and `TLSKeyPath` fields to be defined too.

This field is **optional**.

TLSCACert

Path to the CA used to connect to the endpoint.

This field is **optional**.

TLSCert

Path to the certificate used to connect to the endpoint.

This field is **optional**.

TLSKey

Path to the key used to connect to the endpoint.

This field is **optional**.

Endpoint synchronization

When using the `--external-endpoints` flag, Portainer will read the specified JSON file at startup and automatically create the endpoints.

Portainer will then read the file based on the interval defined in `--sync-interval` (every 60s by default) and will automatically do the following:

- For each endpoint in the database, it will automatically merge any configuration found in the file using the endpoint name as the comparison key
- If an endpoint exists in the database but is not present in the file, it will be removed from the database
- If an endpoint exists in the file but not in the database it will be created in the database

When using external endpoint management, endpoint management via the UI will be disabled to avoid any possible configuration overwrite (the endpoints view is still accessible but will only display the list of endpoints without giving the possibility to create/update endpoints). A simple warning message will be displayed in the endpoints view.

Template definitions are written in JSON.

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

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.

title

Title of the template.

This field is **mandatory**.

description

Description of the template.

This field is **mandatory**.

logo

URL of the template's logo.

This field is **mandatory**.

image

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

This field is **mandatory**.

registry

The registry where the Docker image is stored. If not specified, Portainer will use the Dockerhub as the default registry.

This field is **optional**.

command

The command to run in the container. If not specified, the container will use the default command specified in its Dockerfile.

This field is **optional**.

Example:

```
{
  "command": "/bin/bash -c \"echo hello\" && exit 777"
}
```

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.

Depending on the value in *type* field, the view will display a different input. For example, when using the value *container* for the *type* field, the UI will display a dropdown with all the running containers. The container hostname will then be inserted as a value in the environment variable.

Supported types:

- *container*

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)",
  "type": "only container is available at the moment (optional)",
  "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"
    }
  ]
}
```

network

A string corresponding to the name of an existing Docker network.

Will auto-select the network (if it exists) in the templates view.

This field is **optional**.

Example:

```
{
  "network": "host"
}
```

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"]
}
```

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"]
}
```

privileged

Should the container be started in privileged mode. Boolean, will default to false if not specified.

This field is **optional**.

```
{
  "privileged": true
}
```

interactive

Should the container be started in foreground (equivalent of `-i -t` flags). Boolean, will default to false if not specified.

This field is **optional**.

```
{
  "interactive": true
}
```

restart_policy

Restart policy associated to the container. Value must be one of the following:

- no
- unless-stopped
- on-failure
- always

This field is **optional**. Will default to `always` if not specified.

```
{
  "restart_policy": "unless-stopped"
}
```

note

Usage / extra information about the template. This will be displayed inside the template creation form in the Portainer UI.

Supports HTML.

This field is **optional**.

```
{
  "note": "You can use this field to specify extra information. <br/> It supports <b>
  ↪HTML</b>."
}
```

platform

Supported platform. This field value must be set to **linux** or **windows**. This will display a small platform related icon in the Portainer UI.

This field is **optional**.

```
{
  "platform": "linux"
}
```

categories

An array of categories that will be associated to the template. Portainer UI category filter will be populated based on all available categories.

This field is **optional**.

```
{
  "categories": ["webserver", "open-source"]
}
```

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
```


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

Build Portainer locally

Requirements

Ensure you have [Docker](#), [Node.js](#) $\geq 0.8.4$ and [npm](#) installed locally.

Build

Checkout the project and go inside the root directory:

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

Install the dependencies using `npm`:

```
$ npm install -g bower && npm install
```

Ensure that a folder named `bower_components` is created in the root directory, if not run the following command:

```
$ bower install --allow-root
```

Note for CentOS users, you'll need to create a symlink to the `shasum` binary:

```
$ ln -s /usr/bin/shasum /usr/bin/shasum
```

Build the app locally:

```
$ grunt build
```

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
```

Contribution guidelines

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

Information about supported platforms and Docker versions.

Docker

Portainer is compatible with the following versions of Docker:

- Docker > 1.9

Portainer has partial support for the following versions of Docker:

- Docker 1.9

Portainer is **not** compatible with the following versions of Docker:

- Docker < 1.9

Swarm

Portainer is compatible with the following versions of Docker Swarm standalone:

- Docker Swarm \geq 1.2.3

Note: this is not related to Docker Swarm mode, see https://docs.docker.com/swarm/swarm_at_scale/deploy-app/

Supported platforms

Portainer can be deployed on the following platforms:

- Linux amd64
- Linux 386

- Linux arm
- Linux arm64
- Linux ppc64le
- Windows amd64
- Darwin amd64

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

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

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

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.

I restarted Portainer and lost all my data, why?

Portainer data is stored inside the Docker container. If you want to keep the data of your Portainer instance after reboot/upgrade, you'll need to persist the data. See [Deployment](#)

How can I use a custom CSS file to customize Portainer look?

A workaround can be used to specify your own *vendor.css* and *portainer.css* files. Simply bind mount the folder of your choice to the *css* folder inside the container:

```
$ docker run -d -p 9000:9000 -v <your-absolute-path>/css:/css/ portainer/portainer
```

I am getting the error “Your session has expired” on login and cannot login. What’s wrong?

When running Portainer inside a container, it will use your Docker engine system time to calculate the authentication token expiry time. A timedrift in your Docker system time might occur when using computer/VM hibernation. You need to ensure that your Docker engine system time is the same as your machine system time and if not, restart your Docker engine.

As simple way to check your Docker system time is to use `docker info` or if the information is not available `docker run busybox date`.

Users of Docker for Windows can also fix this by navigating to hyper-v-management -> virtual machines -> right-click on MobyLinuxVM -> settings -> integration services and enabling the time sync checkbox in the services list.

How can I access the Docker API on port 2375 on Windows?

On some Windows setup, Docker is listening on the local loopback address and cannot be accessed from within the Portainer container. You can use `netsh` to create a port redirection, and then use the newly created IP address to connect from Portainer.

Create a redirection from the loopback address on port 2375 to a newly created address **10.0.75.1** on port 2375 (DOS/Powershell command):

```
> netsh interface portproxy add v4tov4 listenaddress=10.0.75.1 listenport=2375_
↳connectaddress=127.0.0.1 connectport=2375
```

You'll then be able to use **10.0.75.1:2375** as the URL of your endpoint.

How can I use Portainer behind a proxy?

When using Portainer behind a proxy, some features requiring access to the Internet (such as Apps Templates) might be unavailable.

When running Portainer as a container, you can specify the `HTTP_PROXY` and `HTTPS_PROXY` env var to specify which proxy should be used.

Example:

```
$ docker run -d -p 9000:9000 -e HTTP_PROXY=my.proxy.domain:7777 portainer/portainer
```

How can I upgrade my version of Portainer?

If you're running Portainer as a container, it's simply a matter of Docker image version. Just stop your existing Portainer container, pull the latest `portainer/portainer` image and create a new Portainer container (using the same options you used to create the previous one).

If you're running Portainer as a service in a Swarm cluster, you can issue the following command to update the image (assuming your Docker service is called `portainer`):

```
$ docker service update --image portainer/portainer:latest portainer
```

If you're running Portainer outside of Docker, download and extract the new binaries and restart the Portainer binary using the same options you used before.