Portainer Documentation

Release 1.18.0

Portainer.io

Jun 21, 2018

Contents

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:

CHAPTER 1

Deployment

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

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

1.1 Quick start

Deploying Portainer is as simple as:

```
$ docker volume create portainer_data
$ docker run -d -p 9000:9000 --name portainer --restart always -v /var/run/docker.
↪sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer
```

Voilà, you can now use Portainer by accessing the port 9000 on the server where Portainer is running.

1.2 Inside a Swarm cluster

Use our agent setup to deploy Portainer inside a Swarm cluster.

Note: This setup will assume that you're executing the following instructions on a Swarm manager node.

```
$ curl -L https://portainer.io/download/portainer-agent-stack.yml -o portainer-agent-
↪stack.yml
$ docker stack deploy --compose-file=portainer-agent-stack.yml portainer
```

Have a look at the *Agent* section to find more details on how to connect an existing Portainer instance to a manually deployed Portainer agent.

1.3 Persist Portainer data

By default, Portainer store its data inside the container in the `/data` folder on Linux (`C:\\data` on Windows).

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 --name portainer --restart always -v /var/run/docker.
↪sock:/var/run/docker.sock -v /path/on/host/data:/data portainer/portainer
```

Example on Windows:

```
$ docker run -d -p 9000:9000 --name portainer --restart always -v /var/run/docker.
↪sock:/var/run/docker.sock -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 \
    --replicas=1 \
    --constraint 'node.role == manager' \
    --mount type=bind,src=//path/on/host/data,dst=/data \
    portainer/portainer
```

Note: The Swarm service example will persist Portainer data in `/path/on/host/data` for each host in the cluster. If the container is re-scheduled on another node, existing Portainer data might not be available. Persisting data across all nodes of a Swarm cluster is outside the scope of this documentation.

1.4 Advanced deployment

Advanced Portainer deployment scenarios.

1.4.1 Declaring the Docker environment to manage upon deployment

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

```
$ docker run -d -p 9000:9000 --name portainer --restart always -v portainer_data:/
↪data portainer/portainer -H tcp://<REMOTE_HOST>:<REMOTE_PORT>
```

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

You can also bind mount the Docker socket to manage a local Docker environment (**only possible on environments where the Unix socket is available**):

```
$ docker run -d -p 9000:9000 --name portainer --restart always -v /var/run/docker.
↪sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer -H unix:///
↪var/run/docker.sock
```

If your Docker environment 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 --name portainer --restart always  -v /path/to/certs:/
↪certs -v portainer_data:/data 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 --name portainer -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
$ docker run -d -p 9000:9000 --name portainer --restart always  -v /path/to/certs:/
↪certs -v portainer_data:/data portainer/portainer -H tcp://<DOCKER_HOST>:<DOCKER_
↪PORT> --tlsverify --tlscacert /certs/myCa.pem --tlscert /certs/myCert.pem --tlskey /
↪certs/myKey.pem
```

1.4.2 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 -d -p 443:9000 --name portainer --restart always -v ~/local-certs:/certs␣
↪-v portainer_data:/data portainer/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.

1.4.3 Deploy Portainer via docker-compose

You can use docker-compose to deploy Portainer.

Here is an example compose file:

```
version: '2'

services:
  portainer:
    image: portainer/portainer
    command: -H unix:///var/run/docker.sock
```

(continues on next page)

(continued from previous page)

```
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - portainer_data:/data

volumes:
  portainer_data:
```

1.4.4 Deploy Portainer 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/1.18.0/portainer-1.18.
↪0-linux-amd64.tar.gz
$ tar xvpfz portainer-1.18.0-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 (or change the path it will use via the `--data`, see below).

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

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

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

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

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

CHAPTER 2

Configuration

Portainer can be easily tuned using CLI flags.

2.1 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
```

2.2 Admin password

2.2.1 From the command line

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$05$qFHAlNAH0A.6oCDe1/4W.ueCWC/
↪iTfBMXIHBI97QYfMWlMCJ7N.a6'
```

2.2.2 Inside a file

You can also store the plaintext password inside a file and use the `--admin-password-file` flag:

```
# mypassword is plaintext here
$ echo -n mypassword > /tmp/portainer_password
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock -v /tmp/
↪portainer_password:/tmp/portainer_password portainer/portainer --admin-password-
↪file /tmp/portainer_password
```

This works well with Swarm & Docker secrets too:

```
# mypassword is plaintext here
$ echo -n mypassword | docker secret create portainer-pass -
$ docker service create \
  --name portainer \
  --secret portainer-pass \
  --publish 9000:9000 \
  --replicas=1 \
  --constraint 'node.role == manager' \
  --mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \
  portainer/portainer \
  --admin-password-file '/run/secrets/portainer-pass' \
  -H unix:///var/run/docker.sock
```

Note: This will automatically create an administrator account called **admin** with the specified password.

2.3 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 service=secret
```

2.4 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 (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"
```

2.5 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 http://my-host.my-domain/templates.json
```

For more information about hosting your own template definitions see *Templates*

2.6 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*

2.7 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>`
- `--admin-password-file`: Path to the file containing the password for the admin user

- `--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)
- `--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`)

CHAPTER 3

API

Portainer exposes an HTTP API that you can use to automate everything you do via the Portainer UI.

3.1 Documentation

The API documentation is available on Swaggerhub and you can also find some examples here.

CHAPTER 4

Agent

4.1 Purpose

The Portainer Agent is a workaround for a Docker API limitation when using the Docker API to manage a Docker environment. The user interactions with specific resources (containers, networks, volumes and images) are limited to those available on the node targeted by the Docker API request.

Docker Swarm mode introduces a concept which is the clustering of Docker nodes. It also adds services, tasks, configs and secrets which are cluster-aware resources. Cluster-aware means that you can query for a list of services or inspect a task inside any node on the cluster, as long as you're executing the Docker API request on a manager node.

Containers, networks, volumes and images are node specific resources, not cluster-aware. When you, for example, want to list all the volumes available on a node inside your cluster, you will need to send a query to that specific node.

The purpose of the agent aims to allows previously node specific resources to be cluster-aware, all while keeping the Docker API request format. As aforementioned, this means that you only need to execute one Docker API request to retrieve all these resources from every node inside the cluster. In all bringing a better Docker user experience when managing Swarm clusters.

4.2 Deployment

Here follow the instructions to deploy the Agent, and to connect it to Portainer.

4.2.1 Deploy it as a stack

Have a look at the deployment documentation *Inside a Swarm cluster* to quickly deploy the agent and a Portainer instance inside a Swarm cluster via `docker stack deploy`.

4.2.2 Manual deployment

Overall, the setup consists of the following steps:

- Step 1: Create a new overlay network in your Swarm cluster for the Agent.
- Step 2: Deploy the Agent as a global service in your cluster (connected to the overlay network).
- Step 3: Connect your Portainer instance to any of the agents by using the Agent's IP:PORT as an endpoint.

Note: This setup assumes that you are executing the following instructions on a Swarm manager node.

Step 1, creating a new overlay network in your Swarm cluster:

```
$ docker network create --driver overlay portainer_agent_network
```

Step 2, deploying the Agent as a *global* service in your cluster:

```
$ docker service create \
    --name portainer_agent \
    --network portainer_agent_network \
    -e AGENT_CLUSTER_ADDR=tasks.portainer_agent \
    --mode global \
    --mount type=bind,src=//var/run/docker.sock,dst=/var/run/docker.sock \
    portainer/agent
```

Step 3, deploying the Portainer instance as a service:

```
$ docker service create \
    --name portainer \
    --network portainer_agent_network \
    --publish 9000:9000 \
    --replicas=1 \
    --constraint 'node.role == manager' \
    portainer/portainer -H "tcp://tasks.portainer_agent:9001" --tlsskipverify
```

4.2.3 Connecting an existing Portainer instance to an agent

If you want to connect an existing Portainer instance to an agent, you can choose the **Agent** environment type when creating a new endpoint.

Ensure when deploying the agent, that you expose the Agent's port inside your Swarm cluster, and that the mode is set to **host** (default port is 9001):

```
$ docker service create \
    --name portainer_agent \
    --network portainer_agent_network \
    --publish mode=host,target=9001,published=9001 \
    -e AGENT_CLUSTER_ADDR=tasks.portainer_agent \
    --mode global \
    --mount type=bind,src=//var/run/docker.sock,dst=/var/run/docker.sock \
    portainer/agent
```

You can then use the address of any node in your cluster (with the agent port) inside the Agent URL field.

Alternatively, you can deploy the agent using the following stack:

```
version: '3.2'

services:
  agent:
    image: portainer/agent
    environment:
      AGENT_CLUSTER_ADDR: tasks.agent
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - target: 9001
        published: 9001
        protocol: tcp
        mode: host
    networks:
      - portainer_agent
    deploy:
      mode: global

networks:
  portainer_agent:
    driver: overlay
```

4.2.4 Configuration

You can change the configuration of the agent by using environment variables.

The following environment variables can be tuned:

- AGENT_PORT: Agent port (default: `9001`)
- LOG_LEVEL: Agent log level (default: `INFO`)

CHAPTER 5

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,
    "TLSSkipVerify": true,
    "TLSCACert": "/tmp/ca.pem",
    "TLSCert": "/tmp/cert.pem",
    "TLSKey": "/tmp/key.pem"
  }
]
```

5.1 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",
```

(continues on next page)

(continued from previous page)

```
  "TLSKey": "/tmp/key.pem"
}
```

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

5.1.1 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**.

5.1.2 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**.

5.1.3 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**.

5.1.4 TLSSkipVerify

Specify this field to true if you want to skip server verification. Defaults to `false`.

This field is **optional**.

5.1.5 TLSCACert

Path to the CA used to connect to the endpoint.

This field is **optional**.

5.1.6 TLSCert

Path to the certificate used to connect to the endpoint.

This field is **optional**.

5.1.7 `TLSKey`

Path to the key used to connect to the endpoint.

This field is **optional**.

5.2 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 find in the file using the enpoint 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 will 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.

CHAPTER 6

Templates

Template definitions are written in JSON.

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

6.1 Container template definition format

A template element must be a valid JSON object.

Example of a container template:

```
{
  "type": "container",
  "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 optionals.

6.1.1 `type`

Template type, either *container* or *stack*.

This field is **mandatory**.

6.1.2 title

Title of the template.

This field is **mandatory**.

6.1.3 description

Description of the template.

This field is **mandatory**.

6.1.4 image

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

This field is **mandatory**.

6.1.5 name

Default name to use for this template in the UI.

This field is **optional**.

6.1.6 logo

URL of the template's logo.

This field is **optional**.

6.1.7 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**.

6.1.8 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"
}
```

6.1.9 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"
    }
  ]
}
```

6.1.10 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"
}
```

6.1.11 volumes

A JSON array describing the associated volumes of the template. Each element in the array must be a valid JSON object that has a required container property.

For each element in the array, a Docker volume will be created and associated when starting the container. If a bind property is defined it will be used as the source of a bind mount.

This field is **optional**.

Example:

```
{
  "volumes": [
    {
      "container": "/etc/nginx"
    },
    {
      "container": "/usr/share/nginx/html",
      "bind": "/var/www"
    }
  ]
}
```

6.1.12 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"]
}
```

6.1.13 labels

A JSON array describing the labels associated to the template. Each element in the array must be a valid JSON object with two properties `name` and `value`.

This field is **optional**.

Example:

```
{
  "labels": [
    { "name": "com.example.vendor", "value": "Acme" },
    { "name": "com.example.license", "value": "GPL" },
    { "name": "com.example.version", "value": "1.0" }
  ]
}
```

6.1.14 privileged

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

This field is **optional**.

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

6.1.15 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
}
```

6.1.16 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"
}
```

6.1.17 hostname

Set the hostname of the container.

This field is **optional**. Will use Docker default if not specified.

```
{
  "hostname": "mycontainername"
}
```

6.1.18 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>."
}
```

6.1.19 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"
}
```

6.1.20 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"]
}
```

6.2 Stack template definition format

A template element must be a valid JSON object.

Stack templates can only be deployed inside Swarm clusters via `docker stack deploy`. Portainer is not compatible with `docker-compose` at the moment.

Example of a stack template:

```
{
  "type": "stack",
  "title": "CockroachDB",
  "description": "CockroachDB cluster",
  "note": "Deploys an insecure CockroachDB cluster, please refer to <a href=\"https://
↪www.cockroachlabs.com/docs/stable/orchestrate-cockroachdb-with-docker-swarm.html\"␣
↪target=\"_blank\">CockroachDB documentation</a> for production deployments.",
  "categories": ["database"],
  "platform": "linux",
  "logo": "https://cloudinovasi.id/assets/img/logos/cockroachdb.png",
  "repository": {
    "url": "https://github.com/portainer/templates",
    "stackfile": "stacks/cockroachdb/docker-stack.yml"
  }
}
```

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

6.2.1 type

Template type, either *container* or *stack*.

This field is **mandatory**.

6.2.2 title

Title of the template.

This field is **mandatory**.

6.2.3 description

Description of the template.

This field is **mandatory**.

6.2.4 repository

A JSON object describing the public git repository from where the stack template will be loaded. It indicates the URL of the git repository as well as the path to the Compose file inside the repository.

Element format:

```
{
  "url": "URL of the public git repository (mandatory)",
  "stackfile": "Path to the Compose file inside the repository (mandatory)",
}
```

Example:

```
{
  "url": "https://github.com/portainer/templates",
  "stackfile": "stacks/cockroachdb/docker-stack.yml"
}
```

This field is **mandatory**.

6.2.5 name

Default name to use for this template in the UI.

This field is **optional**.

6.2.6 logo

URL of the template's logo.

This field is **optional**.

6.2.7 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 object properties, different types of inputs can be generated (text input, select).

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 unless set is present)",
  "description": "a short description for this input, will be available as a tooltip␣
↪in the UI (optional)",
  "set": "pre-defined value for the variable, will not generate an input in the UI␣
↪(optional)",
  "select": "an array of possible values, will generate a select input (optional)"
}
```

Example:

```
{
  "env": [
    {
      "name": "MYSQL_ROOT_PASSWORD",
      "label": "Root password",
      "description": "Password used by the root user."
    },
    {
      "name": "ENV_VAR_WITH_DEFAULT_VALUE",
      "set": "some_value"
    },
    {
      "name": "ENV_VAR_WITH_SELECT_VALUE",
      "label": "An environment variable",
      "select": [
        {
          "text": "Yes, I agree",
```

(continues on next page)

(continued from previous page)

```
            "value": "Y"
          },
          {
            "text": "No, I disagree",
            "value": "N"
          },
          {
            "text": "Maybe",
            "value": "YN"
          }
        ],
        "description": "Some environment variable."
      }
    ]
}
```

6.2.8 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>."
}
```

6.2.9 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"
}
```

6.2.10 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"]
}
```

6.3 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
```

CHAPTER 7

Contribute

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

7.1 Build Portainer locally

7.1.1 Requirements

Ensure you have Docker, Node.js >= 6, and yarn.

7.1.2 Build

Checkout the project and go inside the root directory:

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

Install dependencies with yarn:

```
$ yarn
```

Build the app locally:

```
$ yarn grunt build
```

Start a watched build process:

```
$ yarn grunt run-dev
```

Access Portainer at http://localhost:9000

Tip: The frontend application will be updated when you save your changes to any of the sources (`app/**/*.js`, `assets/css/app.css` or `index.html`). Just refresh the browser.

Important: Do not forget to lint your code:

```
$ yarn grunt lint
```

7.2 Contribution guidelines

Please follow the contribution guidelines on the repository.

CHAPTER 8

Limitations

Information about supported platforms and Docker versions.

8.1 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

8.2 Swarm

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

- Docker Swarm >= 1.2.3

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

8.3 Supported platforms

Portainer can be deployed on the following platforms:

- Linux amd64
- Linux arm

- Linux arm64
- Linux ppc64le
- Linux s390x
- Windows amd64
- Darwin amd64

CHAPTER 9

FAQ

9.1 My host is using SELinux, can I use Portainer ?

If you want to manage a local Docker environment with **SELinux** enabled, you'll need to pass the `--privileged` flag to the Docker run command when deploying Portainer:

```
$ docker run -d --privileged -p 9000:9000 --name portainer --restart always -v /var/
↪run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer
```

You can also have a look at this helper: https://github.com/dpw/selinux-dockersock.

9.2 How can I expose the Docker API over TCP so that Portainer can communicate with my environment?

To manage a remote Docker environment, Portainer must be able to communicate with the Docker API over the network (usually on TCP 2375, 2376 with TLS).

You have to take into account the **security issues depending on your network environment**.

Please refer to Daemon socket option in the Docker Reference and to Docker Engine on Windows.

9.3 How can I setup Portainer on Windows Server 2016 ?

Have a look at the Airdesk blog post for instructions.

9.4 How can I play with Portainer outside of the public demo?

You can deploy Portainer as a stack in Play-with-Docker.

9.5 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.

9.6 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
```

(continues on next page)

(continued from previous page)

```
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

9.7 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.

9.8 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*

9.9 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.

9.10 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.

9.11 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
```

9.12 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.

9.13 How can I manage a remote Dokku host with Portainer?

Have a look at this gist for instructions.

9.14 How can I enable LDAP authentication ?

Have a look at this post for detailed instructions.