
Portainer Documentation

Release 1.20.2

Portainer.io

Apr 04, 2019

1	Deployment	3
1.1	Quick start	3
1.2	Inside a Swarm cluster	3
1.3	Persist Portainer data	4
1.4	Advanced deployment	4
2	Configuration	9
2.1	Admin password	9
2.2	Hiding specific containers	10
2.3	Use your own logo	10
2.4	Use your own templates	10
2.5	Use an external endpoint source	11
2.6	Available flags	11
3	API	13
3.1	Documentation	13
4	Agent	15
4.1	Purpose	15
4.2	Deployment	15
4.3	Configuration	17
4.4	Usage	19
5	External endpoints	21
5.1	Endpoint definition format	21
5.2	Endpoint synchronization	23
6	Templates	25
6.1	Container template definition format	25
6.2	Stack template definition format	31
6.3	Build and host your own templates	34
7	Contribute	35
7.1	Build Portainer locally	35
7.2	Contribution guidelines	36
7.3	Contributing to the documentation	36

8	Limitations	37
8.1	Docker	37
8.2	Swarm	37
8.3	Supported platforms	37
9	FAQ	39
9.1	My host is using SELinux, can I use Portainer ?	39
9.2	How can I expose the Docker API over TCP so that Portainer can communicate with my environment?	39
9.3	How can I setup Portainer on Windows Server 2016 ?	39
9.4	How can I play with Portainer outside of the public demo?	39
9.5	How can I configure my reverse proxy to serve Portainer?	40
9.6	How can I configure my reverse proxy to serve Portainer using HAProxy?	40
9.7	Exposed ports in the container view redirects me to 0.0.0.0, what can I do?	41
9.8	I restarted Portainer and lost all my data, why?	42
9.9	I am getting the error “Your session has expired” on login and cannot login. What’s wrong?	42
9.10	How can I access the Docker API on port 2375 on Windows?	42
9.11	How can I use Portainer behind a proxy?	42
9.12	How can I upgrade my version of Portainer?	42
9.13	How can I manage a remote Dokku host with Portainer?	43
9.14	How can I enable LDAP authentication ?	43
9.15	Where can I find the source code of the Portainer agent ?	43

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.

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://downloads.portainer.io/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
```

1.3.1 Windows

Docker for Windows 10 supports running both Linux and Windows containers and you need use different start command depending which one you are using. Windows Server supports only native Windows containers.

Example for Linux containers:

```
$ docker run -d -p 9000:9000 --name portainer --restart always -v /var/run/docker.
↪sock:/var/run/docker.sock -v C:\ProgramData\Portainer:/data portainer/portainer
```

Example for native Windows containers:

```
$ docker run -d -p 9000:9000 --name portainer --restart always -v \\.\pipe\docker_
↪engine:\\.\pipe\docker_engine -v C:\ProgramData\Portainer:C:\data portainer/
↪portainer
```

1.3.2 Docker Swarm service

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. However, because Docker has issues with [symlinks](#), if you use Certbot, you will need to pass both the “live” and “archive” directories as volumes (shown below).

```
docker run -d -p 9000:9000 \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /root/portainer/data:/data \
  -v /etc/letsencrypt/live/<redacted>:/certs/live/<redacted>:ro \
  -v /etc/letsencrypt/archive/<redacted>:/certs/archive/<redacted>:ro \
  --name portainer \
  portainer/portainer:1.13.4 --ssl --sslcert /certs/live/<redacted>/cert.pem --
  ↪sslkey /certs/live/<redacted>/privkey.pem
```

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
    restart: always
    ports:
      - 9000:9000
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - portainer_data:/data

volumes:
  portainer_data:
```

[Click here](#) to download the Compose file.

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/1.20.2/portainer-1.20.
  ↪2-linux-amd64.tar.gz
$ tar xvpfz portainer-1.20.2-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 --template-file "${PWD}/templates.json"
```

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
```


Portainer can be easily tuned using CLI flags.

2.1 Admin password

2.1.1 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$05$qFHA1NAH0A.6oCDe1/4W.ueCWC/  
↳iTfBMXIHBI97QYfMWlMCJ7N.a6'
```

2.1.2 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 (continues on next page)
```

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.

2.2 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
```

2.3 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"
```

2.4 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.

Note: at the moment, templates are only loaded once at first Portainer startup. If you already deployed a Portainer instance and want to use your own templates after this, you'll need to clear any existing templates (default templates) via the HTTP API.

There are two ways to specify your own templates:

2.4.1 Bind-mount your own templates

Using the `--template-file` flag you can specify the path to your own template file on the file-system. By default, it points to `/templates.json` on both Linux and Windows hosts.

For example, you can mount your own template file inside the container:

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

Or using the `--template-file` to specify a specific path to the templates file:

```
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock -v /path/to/
↳template/folder:/templates portainer/portainer --template-file /templates/templates.
↳json
```

2.4.2 Host your template file

Using the `--templates` flag you can specify an URL where the template file can be accessed via HTTP.

```
$ 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](#)

2.5 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](#)

2.6 Available flags

The following CLI flags are available:

- `--admin-password`: Admin password in the form `admin:<hashed_password>`

- `--admin-password-file`: Path to the file containing the password for the admin user
- `--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)
- `--external-endpoints`: Enable external endpoint management by specifying the path to a JSON endpoint source in a file
- `--hide-label, -l`: Hide containers with a specific label in the UI
- `--host, -H`: Docker daemon endpoint
- `--logo`: URL to a picture to be displayed as a logo in the UI, use Portainer logo if not specified
- `--no-analytics`: Disable analytics (default: `false`)
- `--no-snapshot`: Disable periodic endpoint snapshot (default: `false`)
- `--snapshot-interval`: Time interval between two endpoint snapshot jobs expressed as a string, e.g. `30s`, `5m`, `1h...` as supported by the [time.ParseDuration method](#) (default: `5m`)
- `--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)
- `--sync-interval`: Time interval between two endpoint synchronization requests expressed as a string, e.g. `30s`, `5m`, `1h...` as supported by the [time.ParseDuration method](#) (default: `60s`)
- `--templates, -t`: URL to templates (apps) definitions
- `--template-file`: Path on disk to templates (apps) definitions (default: `/templates.json`)
- `--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)
- `--tlsverify`: TLS support (default: `false`)

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](#).

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 allow 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

Instructions on how to deploy the Agent and how 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 --attachable 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 \  
  --constraint 'node.platform.os == linux' \  
  --mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \  
  --mount type=bind,src=/var/lib/docker/volumes,dst=/var/lib/docker/volumes \  
  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
```

Step 4, deploying the Agent for all Windows Server nodes

Because of Docker limitation you need to deploy the Agent to all Windows Server nodes by running following command on each of them.

```
$ docker run -d --name portainer_agent --restart always --network portainer_agent_\  
↪network -e AGENT_CLUSTER_ADDR=tasks.portainer_agent --mount type=npipe,source=\  
↪\pipe\docker_engine,target=\\.pipe\docker_engine portainer/agent:windows1803-amd64
```

Note: If you're using Windows server 1803, you might need to open up DNS ports to support the DNS resolution of *tasks.portainer_agent*. See: <https://success.docker.com/article/swarm-internal-dns-is-inaccessible-on-windows-server-1803>

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 \
  --mount type=bind,src=/var/lib/docker/volumes,dst=/var/lib/docker/volumes \
  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
      - /var/lib/docker/volumes:/var/lib/docker/volumes
    ports:
      - target: 9001
        published: 9001
        protocol: tcp
        mode: host
    networks:
      - portainer_agent
    deploy:
      mode: global
      placement:
        constraints: [node.platform.os == linux]

networks:
  portainer_agent:
    driver: overlay
    attachable: true
```

4.3 Configuration

You can use variant agent configurations to achieve different setups or enable specific features.

4.3.1 Shared secret

By default, the agent will register the first Portainer instance that connects to it and prevent connections from any other instance after that.

To bypass this security mechanism, Portainer and the agent can be configured at deployment time to use a shared secret. This configuration allows multiple Portainer instances to connect to the same agent endpoint.

The `AGENT_SECRET` environment variable can be used to define the shared secret.

When deploying the agent as a service:

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

Via a stack file:

```
version: '3.2'  
  
services:  
  agent:  
    image: portainer/agent  
    environment:  
      AGENT_CLUSTER_ADDR: tasks.agent  
      AGENT_SECRET: mysecrettoken  
    volumes:  
      - /var/run/docker.sock:/var/run/docker.sock  
      - /var/lib/docker/volumes:/var/lib/docker/volumes  
    ports:  
      - target: 9001  
        published: 9001  
        protocol: tcp  
        mode: host  
    networks:  
      - portainer_agent  
    deploy:  
      mode: global  
      placement:  
        constraints: [node.platform.os == linux]  
  
networks:  
  portainer_agent:  
    driver: overlay  
    attachable: true
```

The AGENT_SECRET must be specified when deploying Portainer as well:

```
$ docker run -d -p 9000:9000 --name portainer --restart always -e AGENT_  
↪SECRET=mysecrettoken -v /var/run/docker.sock:/var/run/docker.sock -v portainer_  
↪data:/data portainer/portainer
```

4.3.2 Enable host management features

The following features are disabled by default for security reasons:

- Ability to manage the filesystem of the host where the agent is running
- Ability to retrieve hardware information about the host where the agent is running (PCI devices/disks)

In order to enable these features, the agent must be configured properly by:

- Enabling the host management features via the CAP_HOST_MANAGEMENT environment variable

- Bind-mounting the root of the host in the agent container (must be bind-mounted in /host)

Example when deploying the agent via a stack file:

```
version: '3.2'

services:
  agent:
    image: portainer/agent
    environment:
      AGENT_CLUSTER_ADDR: tasks.agent
      CAP_HOST_MANAGEMENT: 1
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /var/lib/docker/volumes:/var/lib/docker/volumes
      - /:/host
    ports:
      - target: 9001
        published: 9001
        protocol: tcp
        mode: host
    networks:
      - portainer_agent
    deploy:
      mode: global
      placement:
        constraints: [node.platform.os == linux]

networks:
  portainer_agent:
    driver: overlay
    attachable: true
```

4.3.3 Available options

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)
- AGENT_CLUSTER_ADDR: Address used by each agent to form a cluster. It is recommended to set this value to `tasks.<AGENT_SERVICE_NAME>` when deploying the agent inside a Swarm cluster.
- AGENT_SECRET: Shared secret used to authorize Portainer instances to connect to the agent
- CAP_HOST_MANAGEMENT: Enable host management features by setting the value to 1

4.4 Usage

4.4.1 API

If you want to use the Portainer API to query containers running on a specific node inside a Swarm cluster and when using the Portainer agent setup, you can specify the `X-PortainerAgent-Target` header in the HTTP request to

target a specific node in the cluster. The value must be set to the name of a specific node that can be retrieved via the `nodeName` property when querying cluster resources (containers, volumes...).

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 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.

6.1 Container template definition format

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

Example of a container template:

```
{
  "type": 1,
  "title": "Nginx",
  "description": "High performance web server",
  "logo": "https://cloudinovasi.id/assets/img/logos/nginx.png",
  "image": "nginx:latest",
  "ports": [
    "8080:80/tcp",
    "443/tcp"
  ]
}
```

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

6.1.1 type

Template type, valid values are: 1 (container), 2 (Swarm stack) or 3 (Compose 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 administrator_only

Should the template be available to administrator users only.

This field is **optional**.

Example:

```
{  
  "administrator_only": true  
}
```

6.1.6 name

Default name to use for this template in the UI.

This field is **optional**.

6.1.7 logo

URL of the template's logo.

This field is **optional**.

6.1.8 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**.

6.1.9 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"
}
```

6.1.10 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)",
  "default": "default value associated to the variable (optional)",
  "preset": "boolean. If set to true, the UI will not generate an input (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",
      "default": "default_value",
      "preset": true
    },
    {
      "name": "ENV_VAR_WITH_SELECT_VALUE",
      "label": "An environment variable",
      "description": "A description for this env var",
      "select": [
        {
          "text": "Yes, I agree",
          "value": "Y",

```

(continues on next page)

(continued from previous page)

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

6.1.11 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"
}
```

6.1.12 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. If a readonly property is defined and true, the volume will be mounted in read-only mode.

This field is **optional**.

Example:

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


6.1.13 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.

It can be optionally prefixed with the port that must be mapped on the host in the `port :` form.

If the host port is not specified, the Docker host will automatically assign one when starting the container.

This field is **optional**.

Example:

```
{
  "ports": ["8080:80/tcp", "443/tcp"]
}
```

6.1.14 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" }
  ]
}
```

6.1.15 privileged

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

This field is **optional**.

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

6.1.16 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
}
```

6.1.17 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"
}
```

6.1.18 hostname

Set the hostname of the container.

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

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

6.1.19 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>."
}
```

6.1.20 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"
}
```


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 administrator_only

Should the template be available to administrator users only.

This field is **optional**.

Example:

```
{
  "administrator_only": true
}
```

6.2.6 name

Default name to use for this template in the UI.

This field is **optional**.

6.2.7 logo

URL of the template's logo.

This field is **optional**.

6.2.8 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)",
  "default": "default value associated to the variable (optional)",
  "preset": "boolean. If set to true, the UI will not generate an input (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",
      "default": "default_value",
      "preset": true
    },
    {
      "name": "ENV_VAR_WITH_SELECT_VALUE",
      "label": "An environment variable",
      "description": "A description for this env var",
      "select": [
        {
          "text": "Yes, I agree",
          "value": "Y",
          "default": true
        },
        {
          "text": "No, I disagree",
          "value": "N"
        },
        {
          "text": "Maybe",
          "value": "YN"
        }
      ]
    },
    "description": "Some environment variable."
  ]
}
```

6.2.9 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>."
}
```

6.2.10 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"
}
```

6.2.11 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

The simplest way to use your own templates is to bind mount your own template file directly into the Portainer container, see *Configuration*.

You can also 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.

7.1 Build Portainer locally

7.1.1 Requirements

Ensure you have Docker, Node.js ≥ 6 , yarn and Golang (≥ 1.11) installed on your system.

7.1.2 Build

Checkout the project, set up the project inside your $\$GOPATH$ and go inside the root directory:

```
$ git clone https://github.com/portainer/portainer.git
$ mkdir -p ${GOPATH}/src/github.com/portainer
$ ln -s ${PWD}/portainer/api ${GOPATH}/src/github.com/portainer/portainer
$ cd portainer
```

Install dependencies with yarn:

```
$ yarn
```

Build and run the project:

```
$ yarn start
```

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.

7.2 Contribution guidelines

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

7.3 Contributing to the documentation

Checkout the project and go inside the root directory:

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

Update the documentation and trigger a local build:

```
$ docker run --rm -v ${PWD}:/docs:/src portainer/docbuilder:latest make html
```

This will create a local folder `docs/build/html` where you will find the generated static files for the documentation.

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 \geq 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

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  1d
    timeout server  1d
    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.

More details about automatic user creation and team assignment in [this article](#).

9.15 Where can I find the source code of the Portainer agent ?

The Portainer agent is a closed source software.

Release notes for the Portainer agent are available via [this repository](#).