Deployment configurations

- Docker Deploy Daemon
- Docker Deploy Daemon (HaRP)
  - Nextcloud and Docker on the same host - with Nextcloud bare metal
  - Nextcloud and Docker on the same host - with Nextcloud in Docker
  - Docker on a remote host - with HaRP container on the local host
  - Docker / Reverse Proxy / Nextcloud on 3 independent hosts - with HaRP container
- Docker Deploy Daemon (Docker Socket Proxy)
  - NC & Docker on the Same-Host
  - Docker on a remote host
  - NC & ExApps in the same Docker
  - Nextcloud in Docker AIO (all-in-one)
    - Default AIO Deploy Daemon (Docker Socket Proxy)
    - Docker Socket Proxy security
- Container log rotation
  - Docker
  - Podman
  - Kubernetes
- NC to ExApp Communication

Currently, two kinds of application deployments are supported:

- Docker Deploy Daemon (Docker Socket Proxy)
- Docker Deploy Daemon (HaRP)

Docker Deploy Daemon

Orchestrates the deployment of applications as Docker containers.

The administrator is responsible for the security actions taken to configure the Docker daemon connected to the Nextcloud instance.
These schemes are only examples of possible configurations.

For Docker Deploy Daemon (HaRP), AppAPI HaRP is required.
For Docker Deploy Daemon (Docker Socket Proxy), we recommend that you use the AppAPI Docker Socket Proxy or AIO Docker Socket Proxy container for Nextcloud AIO.

There are several Docker Daemon Deploy configurations (example schemes):

- Nextcloud and Docker on the **same host** (via socket, DockerSocketProxy, or HaRP)
- Nextcloud on the host and Docker on a **remote** host (via DockerSocketProxy with HTTPS, or HaRP)
- Nextcloud and **ExApps** in the **same Docker network** (via DockerSocketProxy, or HaRP)
- Nextcloud in AIO Docker and **ExApps** in the **same Docker network** (via AIO DockerSocketProxy)

Docker Deploy Daemon (HaRP)

With HaRP, the ExApps initiate the connection for tunneling to the Nextcloud instance and the HaRP container so there is no need to expose any ports or open any firewall rules.

See the diagrams of the respective configurations in the Docker Deploy Daemon (Docker Socket Proxy) section below.

A little introduction to the default ports of the HaRP container is given below. More about it can be found in the HaRP's readme.

- Port `8780` is the HTTP communication port used where Nextcloud connects to the HaRP container.
- Port `8781` is the HTTPS communication port when setup.
- Port `8782` is the FRP tunnel port used by ExApps to connect to the HaRP container.

In any of the cases, the following connections should succeed:

- Nextcloud -> HaRP container (on port 8780/8781)
- HaRP container -> Nextcloud (through proxy or directly as the NC_INSTANCE_URL env var dictates)
- ExApp -> HaRP container (on port 8782)
- ExApp -> Nextcloud (through proxy or directly as the `Nextcloud URL` in the daemon config dictates)

> “
>
> The examples below use `some_very_secure_password` as the HaRP shared key. This is only a placeholder: generate your own random key for every installation, for example with `openssl rand -hex 32`.
>
> Use the same key for `HP_SHARED_KEY` of the HaRP container, the **HaRP shared key** field of the deploy daemon and, for a remote Docker host, `metadatas.token` in `frpc.toml`. Anyone who knows the key can talk to HaRP the way Nextcloud does, including managing ExApp containers on the Docker host, so keep it secret.

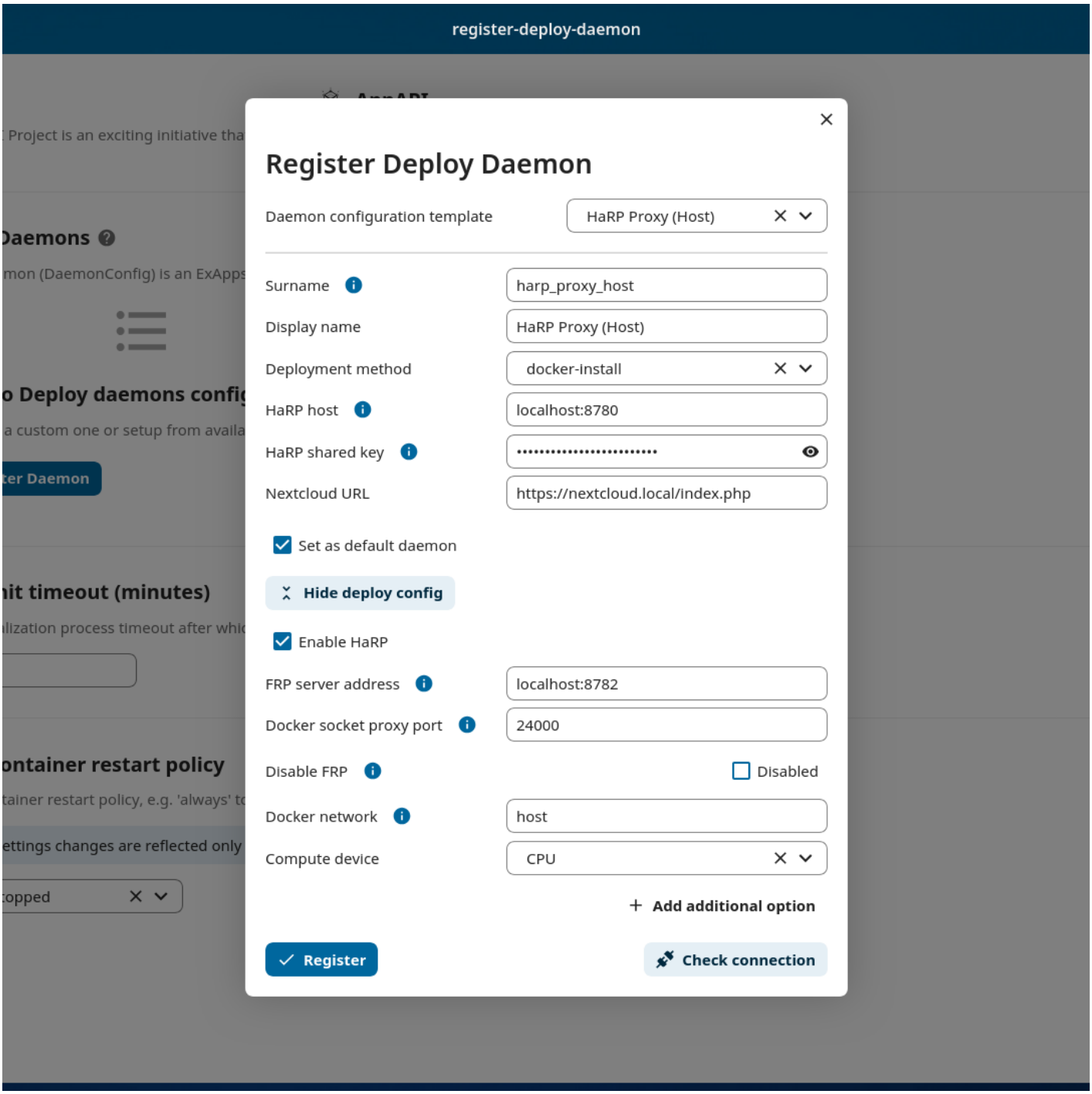
Nextcloud and Docker on the same host - with Nextcloud bare metal

The simplest configuration is when Nextcloud is installed on the host and docker is on the same host and applications are deployed to it.

Create a HaRP container with either `--network host` option or expose the ports `8780` and `8782` to the host.

```
docker run \
  -e HP_SHARED_KEY="some_very_secure_password" \
  -e NC_INSTANCE_URL="https://127.0.0.1:8080" \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v `pwd`/certs:/certs \
  --name appapi-harp -h appapi-harp \
  --restart unless-stopped \
  -p 8780:8780 \
  -p 8782:8782 \
  -d ghcr.io/nextcloud/nextcloud-appapi-harp:release
```

Go to AppAPI admin settings and register a `HaRP Proxy (Host)` daemon.



Finally, test the whole setup with "Test deploy" in the 3-dots menu of the deploy daemon.

Nextcloud and Docker on the same host - with Nextcloud in Docker

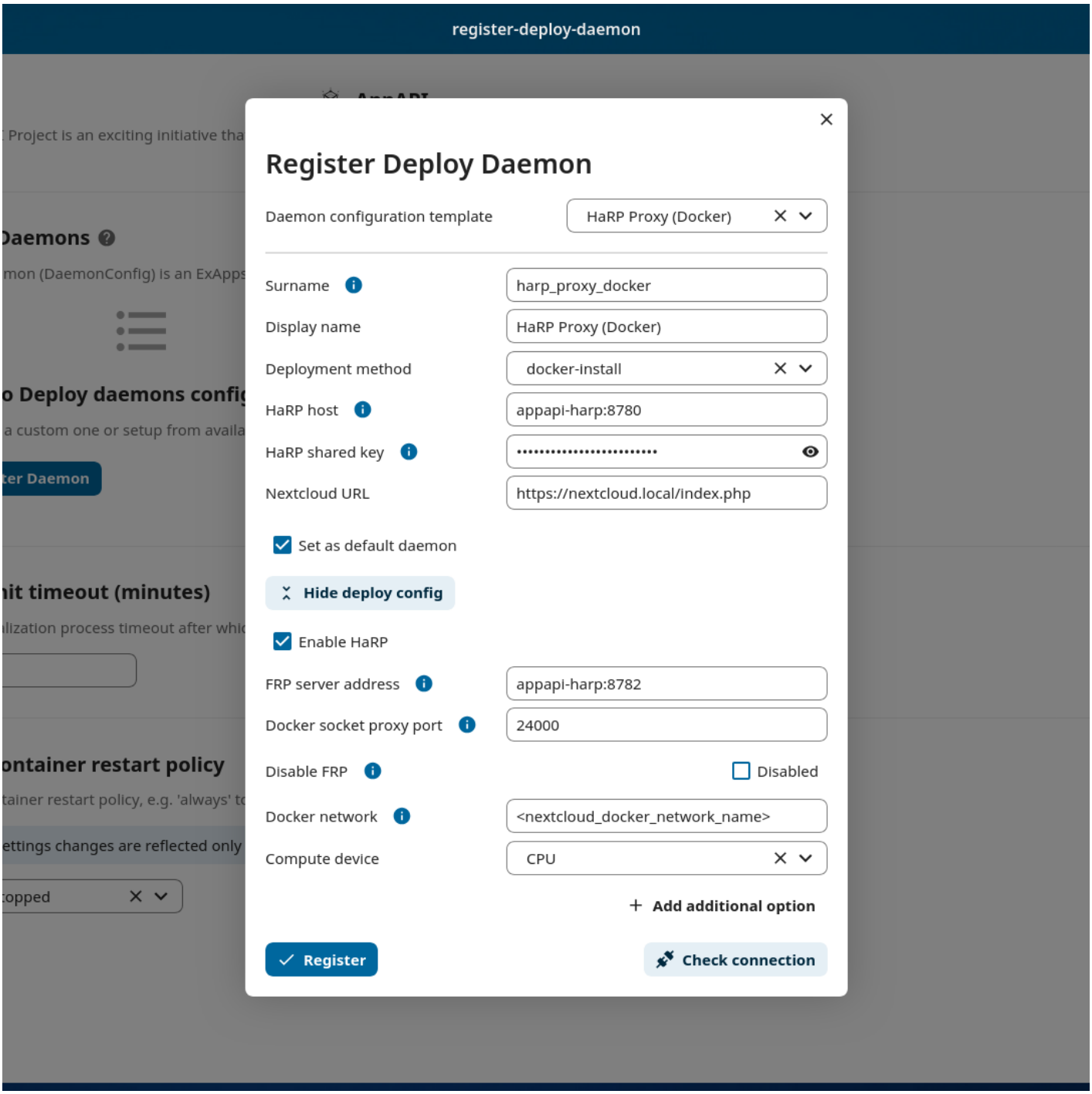
When Nextcloud is installed in Docker, the HaRP container can be created in the same docker network as the Nextcloud instance.

Create a HaRP container with `--network <nextcloud_docker_network_name>` option, where `<nextcloud_docker_network_name>` is the name of the Docker network in which Nextcloud is

accessible.

```
docker run \
  -e HP_SHARED_KEY="some_very_secure_password" \
  -e NC_INSTANCE_URL="https://nextcloud.tld" \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v `pwd`/certs:/certs \
  --name appapi-harp -h appapi-harp \
  --restart unless-stopped \
  --net <nextcloud_docker_network_name> \
  -d ghcr.io/nextcloud/nextcloud-appapi-harp:release
```

Go to AppAPI admin settings and register a `HaRP Proxy (Docker)` daemon. Take note of the `<nextcloud_docker_network_name>` value in the `Docker network` field.



Finally, test the whole setup with "Test deploy" in the 3-dots menu of the deploy daemon.

Docker on a remote host - with HaRP container on the local host

This configuration is suited for deployments that want to offload the heavy lifting of the ExApps to a remote host, especially when using GPUs as compute devices. There can be multiple deploy daemons that can be used to deploy ExApps on different remote hosts for different compute capabilities. Here the HaRP container is deployed on the local host and the remote host tunnels the remote host's docker socket to the local host over the

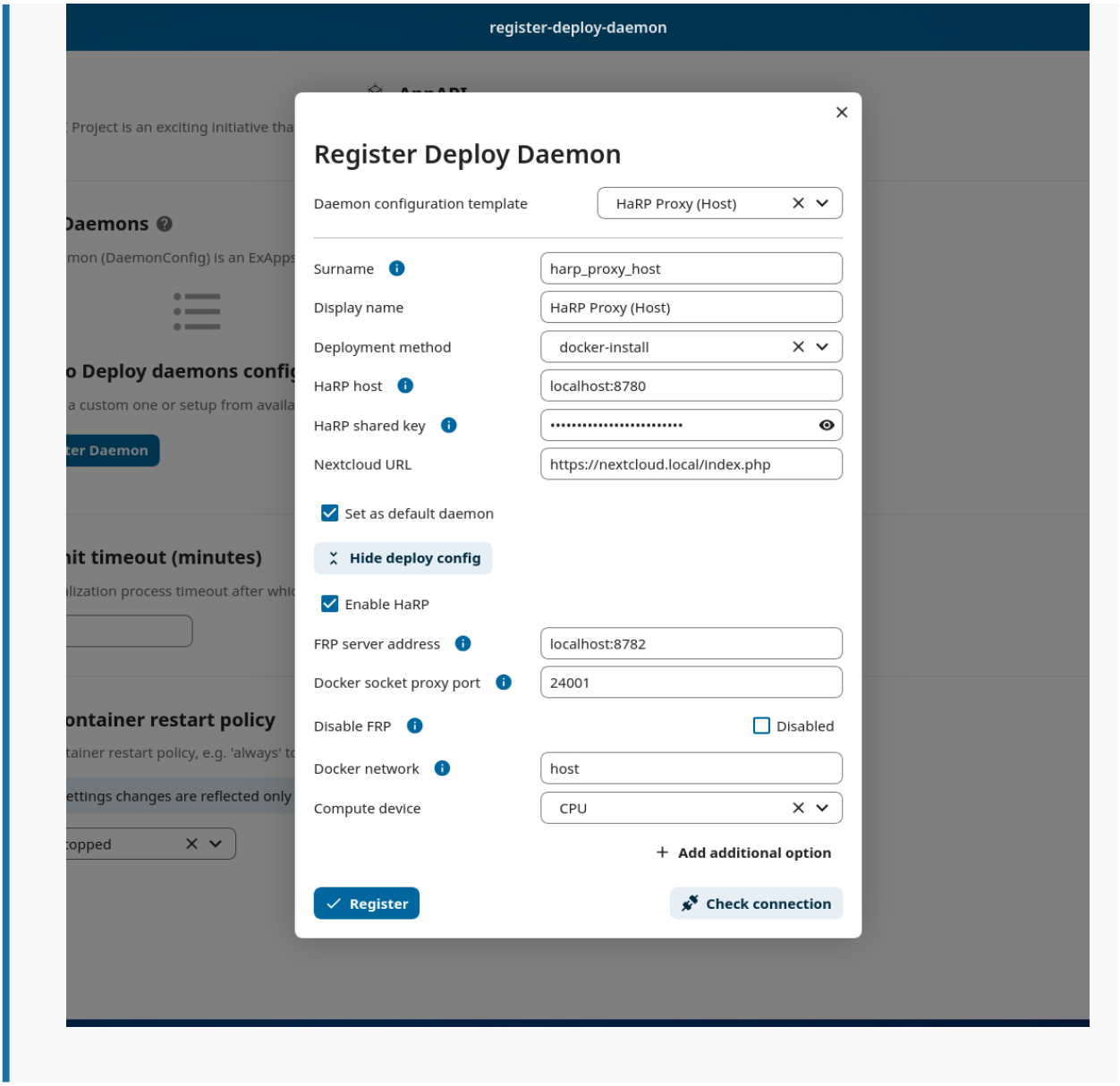
FRP secure tunnel. The ExApps are deployed on the remote host. A setup with the HaRP container itself on the remote is not supported.

1. Create a HaRP container in the local host following the above examples but without the docker socket mount.

   > ```
   > docker run \
   >   -e HP_SHARED_KEY="some_very_secure_password" \
   >   -e NC_INSTANCE_URL="https://127.0.0.1:8080" \
   >   -v `pwd`/certs:/certs \
   >   --name appapi-harp -h appapi-harp \
   >   --restart unless-stopped \
   >   -p 8780:8780 \
   >   -p 8782:8782 \
   >   -d ghcr.io/nextcloud/nextcloud-appapi-harp:release
   > ```

2. Create a matching deploy daemon with `Docker socket proxy port` set to `24001`.

   >



3. The FRP generated client certificates should be present in the `certs` folder locally. Copy the files `client.crt`, `client.key` and `ca.crt` inside the `certs` folder to the remote host.
4. Create a folder structure on the remote host: `mkdir -p certs/frp` and copy the files `client.crt`, `client.key` and `ca.crt` to the `certs/frp` folder.
5. Create a new file `frpc.toml` with the following contents.

> ```
> # frpc.toml
> serverAddr = "your.harp.server.address"          # Replace with
> your HP_FRP_ADDRESS host
> serverPort = 8782                                # Default port for
> FRP or the port your reverse proxy listens on
> loginFailExit = false                            # If the FRP
> ```

```
(HaRP) server is unavailable, continue trying to log in.

transport.tls.certFile = "certs/frp/client.crt"
transport.tls.keyFile = "certs/frp/client.key"
transport.tls.trustedCaFile = "certs/frp/ca.crt"
transport.tls.serverName = "harp.nc"             # DO NOT CHANGE
THIS VALUE

metadatas.token = "some_very_secure_password"    # HP_SHARED_KEY in
quotes

[[proxies]]
remotePort = 24001                               # Unique
remotePort for each Docker Engine (range: 24001-24099)
name = "deploy-daemon-1"                         # Unique name for
each Docker Engine
type = "tcp"
[proxies.plugin]
type = "unix_domain_socket"
unixPath = "/var/run/docker.sock"
```

Make sure to replace the `your.harp.server.address` with the actual address of the local host where the HaRP container is running.
You might want to open the port `8782` on the local host firewall to allow the remote host to connect to it,
or use a reverse proxy to forward the requests to the HaRP container. An example with nginx is given below. Feel free to adjust the port you want to listen on. The FRP client will connect to this port exposed port.
With the reverse proxy config below, the whole setup would only need the main Nextcloud proxy to be exposed and reachable from the outside world, simplifying the network setup.

```
stream {
    server {
        listen 8782;  # Replace with the port you want to listen on
        proxy_pass 127.0.0.1:8782;
        proxy_protocol off;
        proxy_connect_timeout 10s;
```

```
        proxy_timeout 300s;
    }
}
```

6. Download a release of the FRP client from the official releases or our snapshot from here.
7. Extract and copy the `frpc` binary to an appropriate location on the remote host, e.g. `/usr/local/bin`.
8. Make it executable: `chmod +x /usr/local/bin/frpc`.
9. Start the FRP client with the command: `frpc -c /path/to/frpc.toml`.
10. Finally, test the whole setup with "Test deploy" in the 3-dots menu of the deploy daemon.

Docker / Reverse Proxy / Nextcloud on 3 independent hosts - with HaRP container

This is the related infrastructure

```
stateDiagram-v2
    classDef docker fill: #1f97ee, color: white, font-size: 34px, stroke: #364c53, stroke-width: 1px, background: url(https://docs.cocobook.com/uploads/images/gallery/2026-10/docker.png) no-repeat center center / contain
    classDef nextcloud fill: #006aa3, color: white, font-size: 34px, stroke: #045987, stroke-width: 1px, background: url(https://docs.cocobook.com/uploads/images/gallery/2026-10/nextcloud-svg.png) no-repeat center center / contain
    classDef python fill: #1e415f, color: white, stroke: #364c53, stroke-width: 1px
    Direction LR
        Host1 --> Host2 : by port
        Host3 --> Host1 : by port
        Host3 --> Host2 : by port
    state Host1 {
        Nextcloud
    }
    state Host2 {
        [*] --> HaRP : by port
        Daemon --> Containers
        state Containers {
            [*] --> HaRP : /var/run/docker.sock
            HaRP --> ExApp1
            HaRP --> ExApp2
```

```
        HaRP --> ExApp3
    }
}
state Host3 {
    Apache Reverse Proxy
}
class Nextcloud nextcloud
class Daemon docker
class ExApp1 python
class ExApp2 python
class ExApp3 python
```

Please see below the steps I follow All of the following steps are based on a Almalinux Distro. Please customize for your distribution.

1. On the Host2 Docker

1.1. Creation of Cert folder (if necessary)

> ```
> mkdir -p /some/path/certs
> ```

1.2. Open ports

> ```
> firewall-cmd --permanent --zone=public --add-port=8780/tcp
> firewall-cmd --permanent --zone=public --add-port=8782/tcp
> firewall-cmd --reload
> ```

1.3. Deploy of the HaRP Container

> ```
> docker run \
>   -e HP_SHARED_KEY="some_very_secure_password" \
>   -e NC_INSTANCE_URL="https://cloud.acme.com" \
>   -e HP_TRUSTED_PROXY_IPS="192.168.0.0/24" \
>   -v /var/run/docker.sock:/var/run/docker.sock \
>   -v /some/path/certs:/certs \
>   -p 8780:8780 \
> ```

```
-p 8782:8782 \
--name appapi-harp -h appapi-harp \
--restart unless-stopped \
-d ghcr.io/nextcloud/nextcloud-appapi-harp:release
```

Replace `192.168.0.0/24` with the IP address or subnet of your reverse proxy (Host3).

2. On the Host3 Apache Reverse Proxy - Reverse proxy redirections

On the virtual Host "cloud.acme.com" of the apache conf file Add the following lines (before the existing configuration)

```
#  AppAPI Configuration
ProxyPass /exapps/ http://<IP_host2_docker>:8780/exapps/
ProxyPassReverse /exapps/ http://<IP_host2_docker>:8780/exapps/
```

3. On the Nextcloud Web Interface - Daemon Register

Add the following configuration :

- Daemon configuration template : `HaRP Proxy (HOST)`
- Surname : `appapi-harp`
- Display name : `appapi-harp`
- Deployment method : `docker-install`
- HaRP host : `<IP_host2_docker>:8780`
- HaRP shared key : `some_very_secure_password`
- Nextcloud URL : `https://cloud.acme.com`
- FRP server address : `<IP_host2_docker>:8782`
- Docker network : `bridge`

Finally, test the whole setup with “Test deploy” in the 3-dots menu of the deploy daemon.

4. Additional tests from the network of your hosts

> □□
>
> The `docker-engine-port` header tells HaRP which internal virtual port to route to the Docker engine. When HaRP has the Docker socket mounted directly (as in

this setup), it uses `24000` as the default virtual port for that local socket. This value does not require any additional firewall or port configuration.

curl -fsS -H "harp-shared-key: some_very_secure_password" -H "docker-engine-port: 24000" http://<IP_host2_docker>:8780/exapps/app_api/v1.41/ping

curl -fsS -H "harp-shared-key: some_very_secure_password" -H "docker-engine-port: 24000" https://cloud.acme.com/exapps/app_api/v1.41/_ping

Docker Deploy Daemon (Docker Socket Proxy)

NC & Docker on the Same-Host

The simplest configuration is when Nextcloud is installed on the host and Docker is on the same host and applications are deployed to it.

```
stateDiagram-v2
    classDef docker fill: #1f97ee, color: white, font-size: 34px, stroke: #364c53, stroke-width: 1px, background: url(https://docs.cocobook.com/uploads/images/gallery/2026-10/docker.png) no-repeat center center / contain
    classDef nextcloud fill: #006aa3, color: white, font-size: 34px, stroke: #045987, stroke-width: 1px, background: url(https://docs.cocobook.com/uploads/images/gallery/2026-10/nextcloud-svg.png) no-repeat center center / contain
    classDef python fill: #1e415f, color: white, stroke: #364c53, stroke-width: 1px
    Host
    state Host {
        Nextcloud --> Daemon : /var/run/docker.sock
        Daemon --> Containers
        state Containers {
            ExApp1
            --
            ExApp2
```

```
            --
            ExApp3
        }
    }
    class Nextcloud nextcloud
    class Daemon docker
    class ExApp1 python
    class ExApp2 python
    class ExApp3 python
```

Suggested config values(template *Custom default*):

1. Daemon host: `/var/run/docker.sock`
2. HTTPS checkbox: *not supported using docker socket*
3. Network: `host`
4. HaProxy password: **not supported using raw docker socket, should be empty**

Suggested way to communicate with Docker via Docker Socket Proxy container.

```
stateDiagram-v2
    classDef docker fill: #1f97ee, color: white, font-size: 34px, stroke: #364c53, stroke-width: 1px, background: url(https://docs.cocobook.com/uploads/images/gallery/2026-10/docker.png) no-repeat center center / contain
    classDef nextcloud fill: #006aa3, color: white, font-size: 34px, stroke: #045987, stroke-width: 1px, background: url(https://docs.cocobook.com/uploads/images/gallery/2026-10/nextcloud-svg.png) no-repeat center center / contain
    classDef python fill: #1e415f, color: white, stroke: #364c53, stroke-width: 1px
    Host
    state Host {
        Nextcloud --> DockerSocketProxy: by port
        Docker --> Containers
        Docker --> DockerSocketProxy : /var/run/docker.sock
        state Containers {
            DockerSocketProxy --> ExApp1
            DockerSocketProxy --> ExApp2
            DockerSocketProxy --> ExApp3
        }
    }
    class Nextcloud nextcloud
    class Docker docker
```

```
    class ExApp1 python
    class ExApp2 python
    class ExApp3 python
```

Suggested config values(template *Docker Socket Proxy*):

1. Daemon host: `localhost:2375`
   Choose **A** or **B** option:
   1. Docker Socket Proxy should be deployed with `network=host` and `BIND_ADDRESS=127.0.0.1`
   2. Docker Socket Proxy should be deployed with `network=bridge` and it's port should be published to host's 127.0.0.1(e.g. **-p 127.0.0.1:2375:2375**)
2. HTTPS checkbox: **disabled**
3. Network: `host`
4. HaProxy password: **should not be empty**

> “
>
> Be careful with option `A`, by default **Docker Socket Proxy** binds to `*` if `BIND_ADDRESS` is not specified during container creation. Check opened ports after finishing configuration.

Docker on a remote host

Distributed configuration occurs when Nextcloud is installed on one host and Docker is located on a remote host, resulting in the deployment of applications on the remote host.

Benefit: no performance impact on Nextcloud host.

In this case, the AppAPI uses a Docker Socket Proxy deployed on remote host to access docker socket and ExApps.

```
stateDiagram-v2
    classDef docker fill: #1f97ee, color: white, font-size: 34px, stroke: #364c53, stroke-width: 1px, background: url(https://docs.cocobook.com/uploads/images/gallery/2026-10/docker.png) no-repeat center center / contain
    classDef nextcloud fill: #006aa3, color: white, font-size: 34px, stroke: #045987, stroke-width: 1px, background: url(https://docs.cocobook.com/uploads/images/gallery/2026-10/nextcloud-svg.png) no-repeat center center / contain
    classDef python fill: #1e415f, color: white, stroke: #364c53, stroke-width: 1px
    Direction LR
```

```
        Host1 --> Host2 : by port
    state Host1 {
        Nextcloud
    }
    state Host2 {
        [*] --> DockerSocketProxy : by port
        Daemon --> Containers
        state Containers {
            [*] --> DockerSocketProxy : /var/run/docker.sock
            DockerSocketProxy --> ExApp1
            DockerSocketProxy --> ExApp2
            DockerSocketProxy --> ExApp3
        }
    }
    class Nextcloud nextcloud
    class Daemon docker
    class ExApp1 python
    class ExApp2 python
    class ExApp3 python
```

Suggested config values(template *Docker Socket Proxy*):

1. Daemon host: ADDRESS_OF_REMOTE_MACHINE (e.g. **server_name.com:2375**)
2. HTTPS checkbox: `enabled`
3. Network: `host`
4. HaProxy password: **should not be empty**

NC & ExApps in the same Docker

Applications are deployed in the same Docker where Nextcloud resides.

Suggested way to communicate with Docker: via `docker-socket-proxy`.

```
stateDiagram-v2
    classDef docker fill: #1f97ee, color: white, font-size: 34px, stroke: #364c53, stroke-width: 1px, background: url(https://docs.cocobook.com/uploads/images/gallery/2026-10/docker.png) no-repeat center center / contain
    classDef nextcloud fill: #006aa3, color: white, font-size: 34px, stroke: #045987, stroke-width: 1px, background: url(https://docs.cocobook.com/uploads/images/gallery/2026-10/nextcloud-svg.png) no-repeat center center / contain
    classDef python fill: #1e415f, color: white, stroke: #364c53, stroke-width: 1px
```

```
Host
state Host {
    Daemon --> Containers
    state Containers {
        [*] --> DockerSocketProxy : /var/run/docker.sock
        Nextcloud --> DockerSocketProxy: by port
        --
        DockerSocketProxy --> ExApp1
        DockerSocketProxy --> ExApp2
    }
}
class Nextcloud nextcloud
class Daemon docker
class ExApp1 python
class ExApp2 python
class ExApp3 python
```

Suggested config values(template *Docker Socket Proxy*):

1. Daemon host: nextcloud-appapi-dsp:2375
2. HTTPS checkbox: `disabled`
3. Network: user defined network
4. HaProxy password: **should not be empty**

> “ □□
>
> Network **should not be the default docker's bridge** as it does not support DNS resolving by container names.
>
> This means that **Docker Socket Proxy**, **Nextcloud** and **ExApps** containers should all be in the same docker network, different from the default **bridge**.

Nextcloud in Docker AIO (all-in-one)

In the case of AppAPI in Docker AIO setup (installed in Nextcloud container).

> “ □□

> AIO Docker Socket Proxy container must be enabled.

```
stateDiagram-v2
    classDef docker fill: #1f97ee, color: white, font-size: 34px, stroke: #364c53, stroke-width: 1px, background: url(https://docs.cocobook.com/uploads/images/gallery/2026-10/docker.png) no-repeat center center / contain
    classDef docker2 fill: #1f97ee, color: white, font-size: 20px, stroke: #364c53, stroke-width: 1px, background: url(https://docs.cocobook.com/uploads/images/gallery/2026-10/docker.png) no-repeat center center / contain
    classDef nextcloud fill: #006aa3, color: white, font-size: 34px, stroke: #045987, stroke-width: 1px, background: url(https://docs.cocobook.com/uploads/images/gallery/2026-10/nextcloud-svg.png) no-repeat center center / contain
    classDef python fill: #1e415f, color: white, stroke: #364c53, stroke-width: 1px
    Host
    state Host {
        Daemon --> Containers
        state Containers {
            [*] --> NextcloudAIOMasterContainer : /var/run/docker.sock
            [*] --> DockerSocketProxy : /var/run/docker.sock
            NextcloudAIOMasterContainer --> Nextcloud
            AppAPI --> Nextcloud : installed in
            Nextcloud --> DockerSocketProxy
            DockerSocketProxy --> ExApp1
            DockerSocketProxy --> ExApp2
            DockerSocketProxy --> ExApp3
        }
    }
    class Nextcloud nextcloud
    class Daemon docker
    class Daemon2 docker2
    class ExApp1 python
    class ExApp2 python
    class ExApp3 python
```

AppAPI will automatically create the default DaemonConfig for AIO Docker Socket Proxy in order to use it as an orchestrator to create ExApp containers.

> “□□

> Default DaemonConfig will be created only if the default DaemonConfig is not already registered.

Default AIO Deploy Daemon (Docker Socket Proxy)

Nextcloud AIO has a specifically created Docker Socket Proxy container to be used as the Deploy Daemon in AppAPI. It has fixed parameters):

- Name: `docker_aio`
- Display name: `AIO Docker Socket Proxy`
- Accepts Deploy ID: `docker-install`
- Protocol: `http`
- Host: `nextcloud-aio-docker-socket-proxy:2375`
- Compute device: `CPU`
- Network: `nextcloud-aio`
- Nextcloud URL (passed to ExApps): `https://$NC_DOMAIN`

Docker Socket Proxy security

AIO Docker Socket Proxy has strictly limited access to the Docker APIs described in HAProxy configuration.

Container log rotation

AppAPI deploys each ExApp as a container but does not set a logging configuration on it, so each ExApp container uses the logging settings of the deploy daemon that runs it. If those logs have no size limit they are never rotated and can grow until they fill the host disk. Configure rotation on the daemon, where it applies to every container the daemon runs.

Docker

Docker's default `json-file` driver sets no size limit. Add the following to the Docker daemon configuration file (`/etc/docker/daemon.json` on most Linux hosts) and restart the daemon:

```
{
    "log-driver": "json-file",
    "log-opts": {
        "max-size": "10m",
        "max-file": "3"
```

```
    }
}
```

```
sudo systemctl restart docker
```

Each container then keeps at most `max-file` log files of `max-size` each (here three files of 10 MB, so up to 30 MB per container). Adjust the values to suit your hosts.

> “□□
>
> A container keeps the logging configuration it was created with, and disabling and enabling an ExApp only stops and starts the existing container. Remove and install the affected ExApps again (or update them) so their containers are recreated with the new settings.

> “□□
>
> Keep the `json-file` or `journald` driver: AppAPI can only download container logs for these two drivers, so a different driver disables log downloads for ExApp containers.

If you deliberately use a different driver (`syslog`, `fluentd`, and so on) to forward logs elsewhere, keep your existing setup: rotation is then handled by that driver and the steps above do not apply.

Podman

When the deploy daemon is Podman, set a global default in `/etc/containers/containers.conf` (or the per-user file) and re-deploy the affected ExApps:

```
[containers]
log_driver = "json-file"
log_size_max = 10485760
```

Kubernetes

For the Kubernetes deploy daemon, container log rotation is handled by the kubelet, not by AppAPI. Set `containerLogMaxSize` and `containerLogMaxFiles` in the kubelet configuration on each node and

restart the kubelet for the change to take effect (it loads its configuration only at startup); see the Kubernetes logging architecture documentation.

NC to ExApp Communication

Communications between Nextcloud and ExApps are done via the AppAPI. With Docker Socket Proxy, the requests are sent to the ExApp container directly. For HaRP, the communication goes through the main Nextcloud proxy and the HaRP container.

Each type of DeployDaemon necessarily implements the `resolveExAppUrl` function.

It has the prototype:

```
public function resolveExAppUrl(
    string $appId, string $protocol, string $host, array $deployConfig, int $port, array
&$auth
) {}
```

where:

- **protocol** is daemon protocol value
- **host** is daemon host value, *can be DNS:port or IP:PORT or even path to docker socket*.
- **port** is an integer with ExApp port
- **deployConfig** can be custom for each Daemon type
- **auth** is an optional array, with *Basic Authentication* data if needed to access ExApp

> “□□
>
> Applies only to Docker Socket Proxy.
>
> The optional additional parameter *OVERRIDE_APP_HOST* can be used to override the host that will be used for ExApp binding.
>
> It can be `0.0.0.0` in some specific configurations, when VPN is used or both Nextcloud instance and ExApps are one the same physical machine but different virtual environments.
>
> Also you can specify something like `10.10.2.5` and in this case `ExApp` will try to bind to that address and AppAPI will try to send request s directly to this address assuming that ExApp itself bound on it.

The simplest implementation is in the **Manual-Install** deploy type:

```
public function resolveExAppUrl(
    string $appId, string $protocol, string $host, array $deployConfig, int $port, array
&$auth
): string {
    if (boolval($deployConfig['harp'] ?? false)) {
        $url = rtrim($deployConfig['nextcloud_url'], '/');
        if (str_ends_with($url, '/index.php')) {
            $url = substr($url, 0, -10);
        }
        return sprintf('%s/exapps/%s', $url, $appId);
    }

    $auth = [];
    if (isset($deployConfig['additional_options']['OVERRIDE_APP_HOST']) &&
        $deployConfig['additional_options']['OVERRIDE_APP_HOST'] !== ''
    ) {
        $wideNetworkAddresses = ['0.0.0.0', '127.0.0.1', '::', '::1'];
        if (!in_array($deployConfig['additional_options']['OVERRIDE_APP_HOST'],
$wideNetworkAddresses)) {
            $host = $deployConfig['additional_options']['OVERRIDE_APP_HOST'];
        }
    }
    return sprintf('%s://%s:%s', $protocol, $host, $port);
}
```

Here we see that AppAPI sends requests to the **host**:**port** specified during daemon creation for manual-install without HaRP.
But it exclusively uses the `http(s)://nextcloud.example.tld/exapps/` route for manual deployments using the HaRP proxy. `http(s)://nextcloud.example.tld` is the Nextcloud URL specified in the daemon config. Take care to configure the `/exapps/` route in your reverse proxy accordingly if your Nextcloud instance is on a subpath `https://nextcloud.example.tld/nextcloud`. See Configuring Your Reverse Proxy in the HaRP readme for examples.

Now, let's take a look at the Docker Daemon implementation of `resolveExAppUrl`:

```
public function resolveExAppUrl(
    string $appId, string $protocol, string $host, array $deployConfig, int $port, array
&$auth
): string {
```

```
    if (boolval($deployConfig['harp'] ?? false)) {
        $url = rtrim($deployConfig['nextcloud_url'], '/');
        if (str_ends_with($url, '/index.php')) {
            $url = substr($url, 0, -10);
        }
        return sprintf('%s/exapps/%s', $url, $appId);
    }

    $auth = [];
    if (isset($deployConfig['additional_options']['OVERRIDE_APP_HOST']) &&
        $deployConfig['additional_options']['OVERRIDE_APP_HOST'] !== ''
    ) {
        $wideNetworkAddresses = ['0.0.0.0', '127.0.0.1', '::', '::1'];
        if (!in_array($deployConfig['additional_options']['OVERRIDE_APP_HOST'],
$wideNetworkAddresses)) {
            return sprintf(
                '%s://%s:%s', $protocol,
$deployConfig['additional_options']['OVERRIDE_APP_HOST'], $port
            );
        }
    }
    $host = explode(':', $host)[0];
    if ($protocol == 'https') {
        $exAppHost = $host;
    } elseif (isset($deployConfig['net']) && $deployConfig['net'] === 'host') {
        $exAppHost = 'localhost';
    } else {
        $exAppHost = $appId;
    }
    if ($protocol == 'https' && isset($deployConfig['haproxy_password']) &&
$deployConfig['haproxy_password'] !== '') {
        // we only set haproxy auth for remote installations, when all requests come through
HaProxy.
        $haproxyPass = $this->crypto->decrypt($deployConfig['haproxy_password']);
        $auth = [self::APP_API_HAPROXY_USER, $haproxyPass];
    }
    return sprintf('%s://%s:%s', $protocol, $exAppHost, $port);
}
```

The route for HaRP setups remain the same here as in the previous example. All the requests are sent to the Nextcloud URL with the `/exapps/` route.

For Docker Socket Proxy, however, we have much more complex algorithm of detecting to where requests should be send.

First of all, if the protocol is set to `https`, AppAPI always sends requests to the daemon host, and in this case, it is a HaProxy that will forward requests to ExApps that will be listening on `localhost`.

Briefly, it will look like this (*haproxy_host==daemon host value*):

NC --> *https* --> `haproxy_host:ex_app_port` --> *http* --> `localhost:ex_app_port`

When the protocol is not `https` but `http`, then what will be the endpoint where to send requests is determined by `$deployConfig['net']` value.

If `net` is defined and equal to `host`, then AppAPI assumes that ExApp is installed somewhere in the current host network and will be available on `localhost` loop-back adapter.

NC --> *http* --> `localhost:ex_app_port`

In all other cases, the ExApp should be available by it's name: e.g. when using docker **custom bridge** network all containers available by DNS.

NC --> *http* --> `app_container_name:ex_app_port`

These three different types of communication cover most popular configurations.

□□□ #4
□ cocobook-sync □□□ 2024-10-22 05:36:14 UTC
□ cocobook-sync □□□ 2026-10-27 19:48:39 UTC