Container

Summary

Sub-menu: `/container`
Packages required: `container`

A container is MikroTik's implementation of Linux containers, allowing users to run containerized environments within RouterOS. The container feature was added in RouterOS v7.4beta4.

Disclaimer

- you need physical access to the router to enable support for the container feature, it is disabled by default;
- once the container feature is enabled, containers can be added/configured/started/stopped/removed remotely!
- if the router is compromised, containers can be used to easily install malicious software in your router and over network;
- your router is as secure as anything you run in container;
- if you run container, there is no security guarantee of any kind;
- running a 3rd party container image on your router could open a security hole/attack vector/attack surface;
- an expert with knowledge how to build exploits will be able to jailbreak/elevate to root;

Security risks:

when a security expert publishes his exploit research - anyone can apply such an exploit;
someone will build a container image that will do the exploit AND provide a Linux root shell;
by using a root shell someone may leave a permanent backdoor/vulnerability in your RouterOS system even after the docker image is removed and the container feature disabled;
if a vulnerability is injected into the primary or secondary routerboot (or vendor pre-loader), then even netinstall may not be able to fix it;

Requirements

Container package is compatible with **arm arm64** and **x86** architectures. Using of remote-image (similar to docker pull) functionality requires a lot of free space in main memory, 16MB SPI flash boards may use pre-build images on USB or other disk media.

External disk is highly recommended

Container package needs to be installed

```
/container
```

Properties

Property	Description
cmd (string; Default:)	The main purpose of a CMD is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an ENTRYPOINT instruction as well.
comment (*string*; Default:)	Short description
dns (string; Default:)	
domain-name (string; Default:)	
entrypoint (string; Default: **)**	An ENTRYPOINT allows to specify executable to run when starting container. Example: /bin/sh
envlist (*string*; Default:)	list of environmental variables (configured under */container envs*) to be used with container
file (*string; Default:*)	container *tar.gz tarball if the container is imported from a file
hostname (*string; Default:*)	
interface (*string; Default:*)	veth interface to be used with the container
logging (*string; Default:*)	if set to yes, all container-generated output will be shown in the RouterOS log
mounts (*string; Default:*)	mounts from /container/mounts/ sub-menu to be used with this container
remote-image (*string; Default:*)	the container image name to be installed if an external registry is used (configured under /container/config set registry-url=...)
root-dir (*string; Default:*)	used to save container store outside main memory
stop-signal (*string ; Default:*)	
workdir (*string*; Default:)	the working directory for cmd entrypoint

Container configuration

```
/container/config/
```

Property	Description
registry-url	external registry url from where the container will be downloaded
tmpdir	container extraction directory
ram-high	RAM usage limit. (0 for unlimited)
username	Specifies the username for authentication (starting from ROS 7.8)
password	Specifies the password for authentication (starting from ROS 7.8)

Container use example

Prerequisites:

a. RouterOS device with RouterOS v7.4beta or later and **installed Container package**
b. Physical access to a device to enable container mode
c. Attached hard drive or USB drive for storage - formatted as ext3/ext4

Enable Container mode

Device-mode limits container use by default, before granting container mode access - make sure your device is fully secured.

enable container mode

```
/system/device-mode/update container=yes
```

You will need to confirm the device-mode with a press of the reset button, or a cold reboot, if using container on X86.

Create network

Add veth interface for the container:

```
/interface/veth/add name=veth1 address=172.17.0.2/24 gateway=172.17.0.1
```

Create a bridge for containers and add veth to it:

```
/interface/bridge/add name=containers
/ip/address/add address=172.17.0.1/24 interface=containers
/interface/bridge/port add bridge=containers interface=veth1
```

Setup NAT for outgoing traffic:

```
/ip/firewall/nat/add chain=srcnat action=masquerade src-address=172.17.0.0/24
```

Add environment variables and mounts (optional)

Create environment variables for container(optional):

```
/container/envs/add name=pihole_envs key=TZ value="Europe/Riga"
/container/envs/add name=pihole_envs key=WEBPASSWORD value="mysecurepassword"
/container/envs/add name=pihole_envs key=DNSMASQ_USER value="root"
```

Define mounts (optional):

```
/container/mounts/add name=etc_pihole src=disk1/etc dst=/etc/pihole
/container/mounts/add name=dnsmasq_pihole src=disk1/etc-dnsmasq.d dst=/etc/dnsmasq.d
```

⚠ `src=` points to RouterOS location (could also be `src=disk1/etc_pihole` if, for example, You decide to put configuration files on external USB media), `dst=` points to defined location (consult containers manual/wiki/github for information on where to point). If `src` directory does not exist on first time use then it will be populated with whatever container have in `dst` location.

Add container image

If You wish to see container output in log - add `logging=yes` when creating a container, root-dir should point to an external drive formatted in ext3 or ext4. It's not recommended to use internal storage for containers.
There are multiple ways to add containers:

a) get an image from an external library

Set registry-url (for downloading containers from Docker registry) and set extract directory (tmpdir) to attached USB media:

```
/container/config/set registry-url=https://registry-1.docker.io tmpdir=disk1/pull
```

pull image:

```
 /container/add remote-image=pihole/pihole:latest interface=veth1 root-dir=disk1/pihole mounts=dnsmasq_pihole,
etc_pihole envlist=pihole_envs
```

The image will be automatically pulled and extracted to root-dir, status can be checked by using

```
/container/print
```

b) import image from PC

These links are `latest` as of 16th of June, 2022. Please make sure to download the right version that matches Your RouterOS device's architecture. Update sha256 sum from docker hub to get the latest image files

```
arm64:
        docker pull pihole/pihole:latest@sha256:4cef8a7b32d318ba218c080a3673b56f396d2e2c74d375bef537ff5e41fc4638
    docker save pihole/pihole > pihole.tar

arm
        docker pull pihole/pihole:latest@sha256:684c59c7c057b2829d19d08179265c79a9ddabf03145c1e2fad2fae3d9c36a94
    docker save pihole/pihole > pihole.tar

amd64
        docker pull pihole/pihole:latest@sha256:f56885979dcffeb902d2ca51828c92118199222ffb8f6644505e7881e11eeb85
    docker save pihole/pihole > pihole.tar
```

After the file has been downloaded and extracted - upload it to Your RouterOS device. Create a container from tar image

```
/container/add file=pihole.tar interface=veth1 envlist=pihole_envs root-dir=disk1/pihole mounts=dnsmasq_pihole,
etc_pihole hostname=PiHole
```

c) build an image on PC

Steps for Linux systems

To use Dockerfile and make your own docker package - docker needs to be installed as well as buildx or other builder toolkit.

Easiest way is to download and install Docker Engine:
https://docs.docker.com/engine/install/

After install check if extra architectures are available:

```
docker buildx ls
```

should return:

```
NAME/NODE DRIVER/ENDPOINT STATUS  PLATFORMS
default * docker
  default default         running linux/amd64, linux/arm64, linux/riscv64, linux/ppc64le, linux/s390x, linux
/386, linux/arm/v7, linux/arm/v6
```

If not - install extra architectures:

```
docker run --privileged --rm tonistiigi/binfmt --install all
```

pull or create your project with Dockerfile included and build, extract image (adjust --platform if needed):

```
git clone https://github.com/pi-hole/docker-pi-hole.git
cd docker-pi-hole
docker buildx build  --no-cache --platform arm64 --output=type=docker -t pihole .
docker save pihole > pihole.tar
```

Upload *pihole.tar* to Your RouterOS device.

Images and objects on the Linux system can be `pruned`

Create a container from the tar image

```
/container/add file=pihole.tar interface=veth1 envlist=pihole_envs mounts=dnsmasq_pihole,etc_pihole
hostname=PiHole
```

Start container

Make sure container has been added and `status=stopped` by using `/container/print`

```
/container/start 0
```

You should be able to access the PiHole web panel by navigating to `http://172.17.0.2` in your web browser.

Forward ports to internal container

Ports can be forwarded using dst-nat (where 192.168.88.1 routers IP address):

```
/ip firewall nat
add action=dst-nat chain=dstnat dst-address=192.168.88.1 dst-port=80 protocol=tcp to-addresses=172.17.0.2 to-
ports=80
```

For Pihole container - set DNS server to containers veth interface IP address -

```
/ip dns set servers=172.17.0.2
```

or change DHCP servers settings to serve Pihole DNS

Tips and tricks

- Containers use up a lot of disk space, USB/SATA,NVMe attached media is highly recommended. For devices with USB ports - USB to SATA adapters can be used with 2.5" drives - for extra storage and faster file operations.
- RAM usage can be limited by using:

```
/container/config/set ram-high=200M
```

this will soft limit RAM usage - if a RAM usage goes over the high boundary, the processes of the cgroup are throttled and put under heavy reclaim pressure.

- For starting containers after router reboot use start-on-boot option (starting from 7.6beta6)

```
/container/print
 0 name="2e679415-2edd-4300-8fab-a779ec267058" tag="test_arm64:latest" os="linux" arch="arm"
interface=veth2
   root-dir=disk1/alpine mounts="" dns="" logging=yes start-on-boot=yes status=running

/container/set 0 start-on-boot=yes
```

- It is possible to get to **running** container shell:

```
/container/shell 0
```

- Enable logging to get output from container:

```
/container/set 0 logging=yes
```

- Starting from 7.11beta5 version multiple addresses and ipv6 addresses can be added:

```
interface/veth add address=172.17.0.3/16,fd8d:5ad2:24:2::2/64 gateway=172.17.0.1 gateway6=fd8d:5ad2:24:
2::1
```