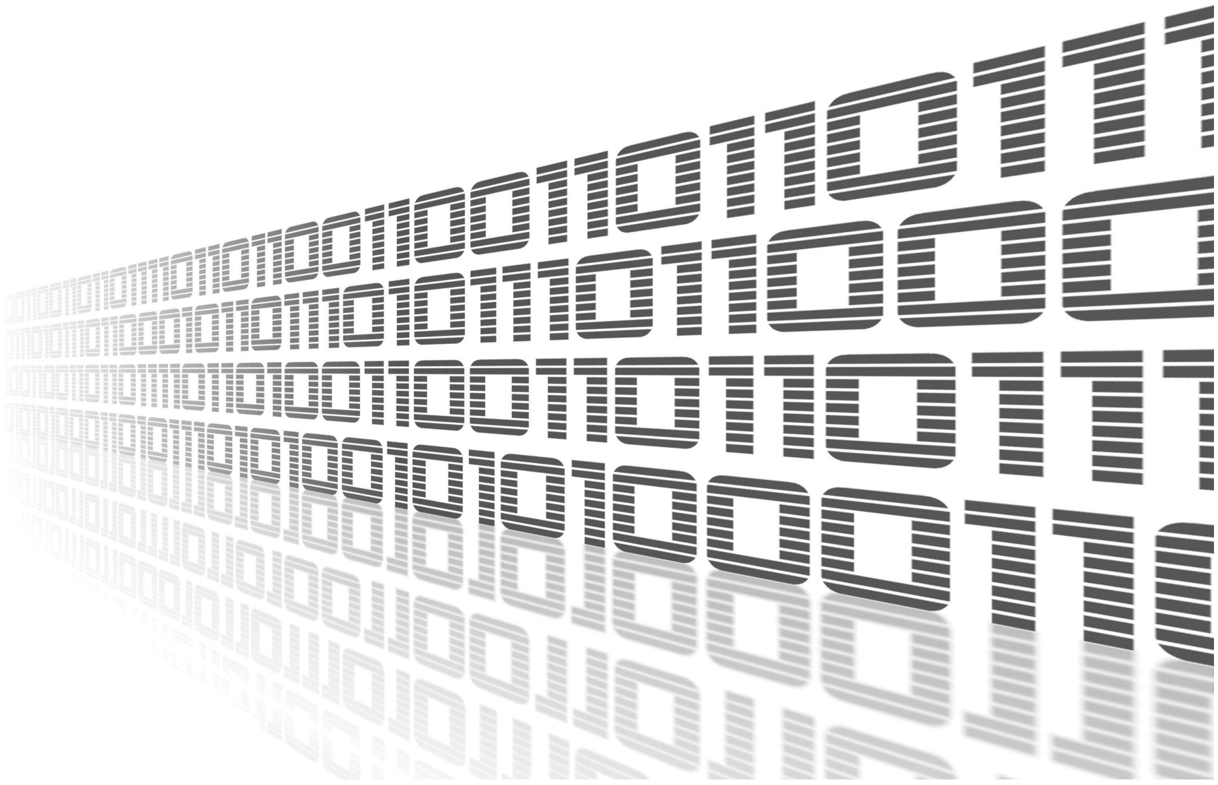
RouterApp

Docker

APPLICATION NOTE



ADVANTECH

Used symbols

Danger – Information regarding user safety or potential damage to the router.

Attention – Problems that may arise in specific situations.

Information or notice – Useful tips or information of special interest.

Example – Example of function, command or script.

Advantech Czech s.r.o., Sokolska 71, 562 04 Usti nad Orlici, Czech Republic
Document No. APP-0104-EN, revised on September 1, 2022. Released in the Czech Republic.

Contents

List of Figures

1. What is Docker?

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Figure 1: Docker logo

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

1.1 Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

Image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

1.2 Example *docker run* command

The following command runs an ubuntu container, attaches interactively to your local command-line session, and runs /bin/bash.

```
$ docker run -i -t ubuntu /bin/bash
```

When you run this command, the following happens (assuming you are using the default registry configuration):

1. If you do not have the ubuntu image locally, Docker pulls it from your configured registry, as though you had run docker pull ubuntu manually.
2. Docker creates a new container, as though you had run a docker container create command manually.
3. Docker allocates a read-write filesystem to the container, as its final layer. This allows a running container to create or modify files and directories in its local filesystem.
4. Docker creates a network interface to connect the container to the default network, since you did not specify any networking options. This includes assigning an IP address to the container. By default, containers can connect to external networks using the host machine's network connection.
5. Docker starts the container and executes /bin/bash. Because the container is running interactively and attached to your terminal (due to the -i and -t flags), you can provide input using your keyboard while the output is logged to your terminal.
6. When you type exit to terminate the /bin/bash command, the container stops but is not removed. You can start it again or remove it.

2. Docker Router App Description

Supported routers: **SmartStart SL305**, **ICR-3200 family**, and **ICR-4400 family**.

Please note that router's firmware of version 6.3.2 and above is required for the *Docker* router app to work properly.

When uploaded to the router, the router app is accessible in the *Customization* section in the *Router Apps* item of the router's web interface. Click on the title of the router app to see the router app menu as shown below:

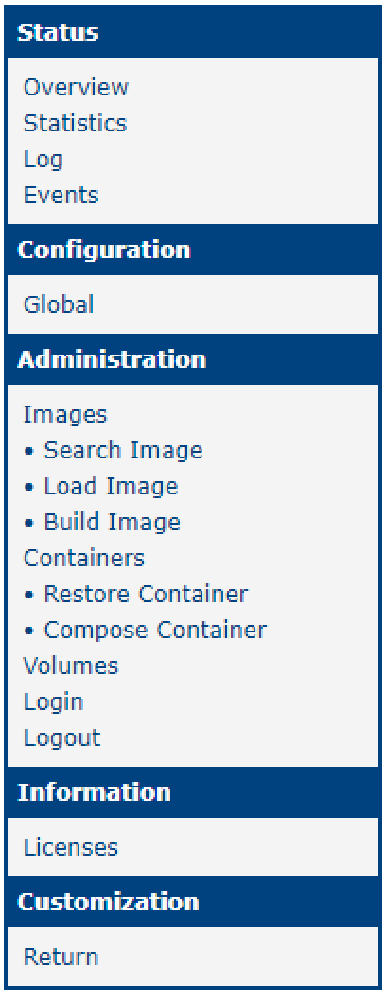


Figure 1: Docker Router App Menu

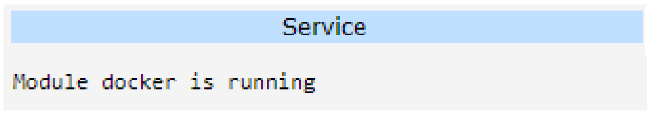
The *Status* section provides the *Overview*, *Statistics*, *Log*, *Events* pages. In the *Configuration* section we will find the *Global* page. *Administration* section contains pages which works with *Images*, *Containers*, *Volumes* and pages for *Login* and *Logout*. More about every page mentioned above will be described further in this application note. The *Information* section provides the *Licenses* page with the list of licenses used in this Router App. The *Return* item in the *Customization* section is to return to the higher menu of the router.

Menu item *Compose Container* is not available on ICR-3xxx routers.

2.1 Status

2.1.1 Overview

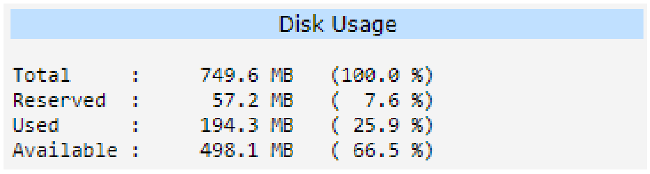
Overview section consists of four parts, first *Service* contains information about Router App status



```
Service

Module docker is running
```

Figure 2: Overview service info

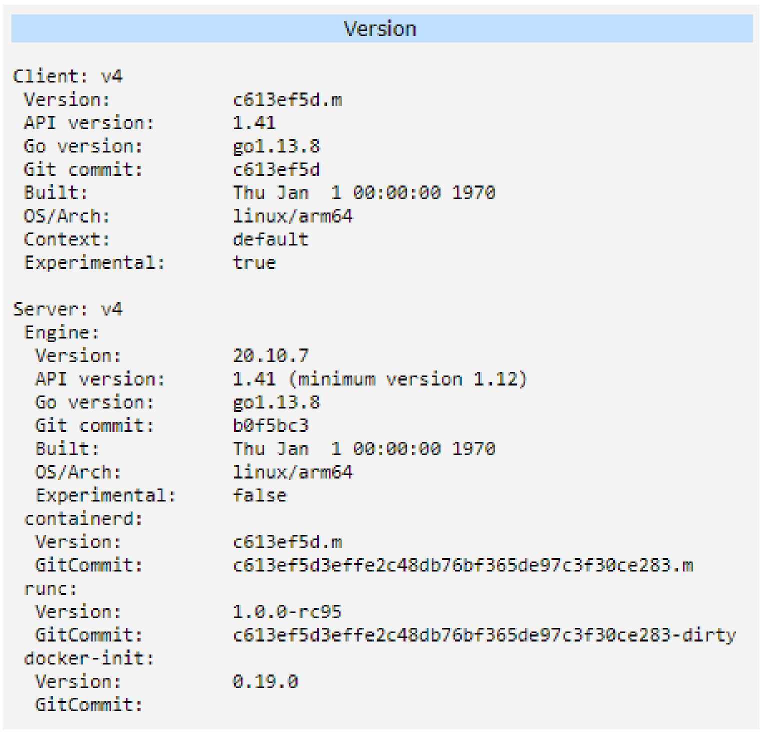
Second part provides information about disk usage



```
Disk Usage

Total     :    749.6 MB   (100.0 %)
Reserved  :     57.2 MB   (  7.6 %)
Used      :    194.3 MB   ( 25.9 %)
Available :    498.1 MB   ( 66.5 %)
```

Figure 3: Status overview disk usage

Third provides information about Docker Router App Version

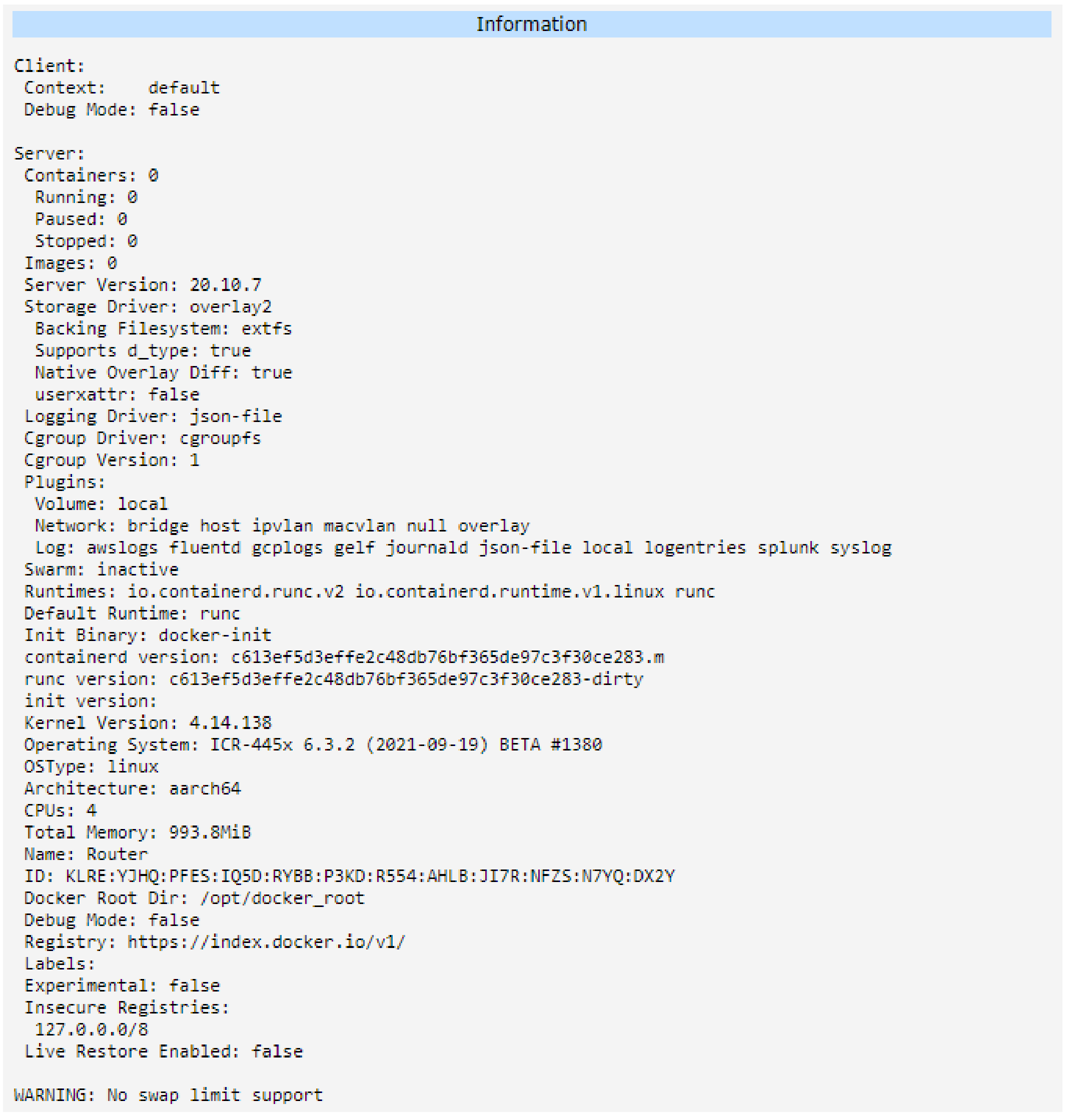


Version

```
Client: v4
 Version:           c613ef5d.m
 API version:       1.41
 Go version:        go1.13.8
 Git commit:        c613ef5d
 Built:             Thu Jan  1 00:00:00 1970
 OS/Arch:           linux/arm64
 Context:           default
 Experimental:      true

Server: v4
 Engine:
  Version:          20.10.7
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.13.8
  Git commit:       b0f5bc3
  Built:            Thu Jan  1 00:00:00 1970
  OS/Arch:          linux/arm64
  Experimental:     false
 containerd:
  Version:          c613ef5d.m
  GitCommit:        c613ef5d3effe2c48db76bf365de97c3f30ce283.m
 runc:
  Version:          1.0.0-rc95
  GitCommit:        c613ef5d3effe2c48db76bf365de97c3f30ce283-dirty
 docker-init:
  Version:          0.19.0
  GitCommit:
```

Figure 4: Status overview version

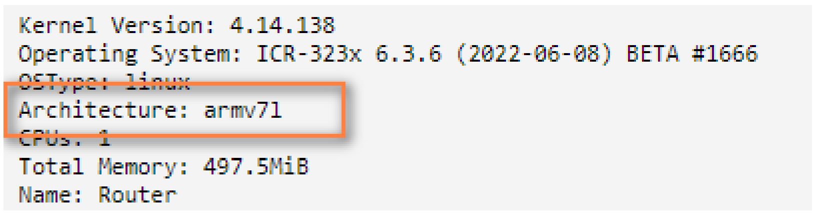
And last one contains all other useful information



```
Client:
 Context:    default
 Debug Mode: false

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 0
 Server Version: 20.10.7
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
  userxattr: false
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Cgroup Version: 1
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
 Swarm: inactive
 Runtimes: io.containerd.runc.v2 io.containerd.runtime.v1.linux runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: c613ef5d3effe2c48db76bf365de97c3f30ce283.m
 runc version: c613ef5d3effe2c48db76bf365de97c3f30ce283-dirty
 init version:
 Kernel Version: 4.14.138
 Operating System: ICR-445x 6.3.2 (2021-09-19) BETA #1380
 OSType: linux
 Architecture: aarch64
 CPUs: 4
 Total Memory: 993.8MiB
 Name: Router
 ID: KLRE:YJHQ:PFES:IQ5D:RYBB:P3KD:R554:AHLB:JI7R:NFZS:N7YQ:DX2Y
 Docker Root Dir: /opt/docker_root
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Live Restore Enabled: false

WARNING: No swap limit support
```

Figure 5: Status overview information

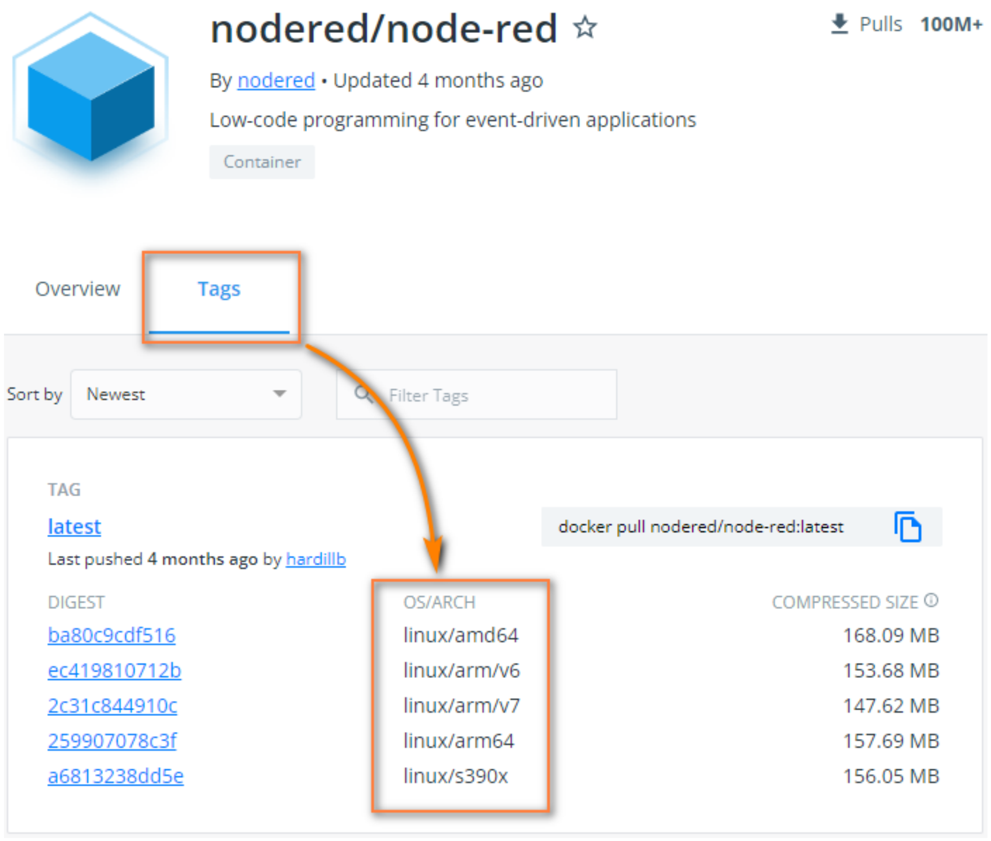
In the *Information* section is one item worth mentioning and it is *Architecture*. This information could be useful when you are not sure if your routers architecture support certain image.



```
Kernel Version: 4.14.138
Operating System: ICR-323x 6.3.6 (2022-06-08) BETA #1666
OSType: linux
Architecture: armv7l
CPUs: 1
Total Memory: 497.5MiB
Name: Router
```

Figure 6: Architecture

You can just go on Docker Hub[1], search for Image you desire and in the *Tags* tab are supported architectures listed.



nodered/node-red ☆ Pulls 100M+
By nodered • Updated 4 months ago
Low-code programming for event-driven applications
Container

TAG
latest
Last pushed 4 months ago by hardillb

docker pull nodered/node-red:latest

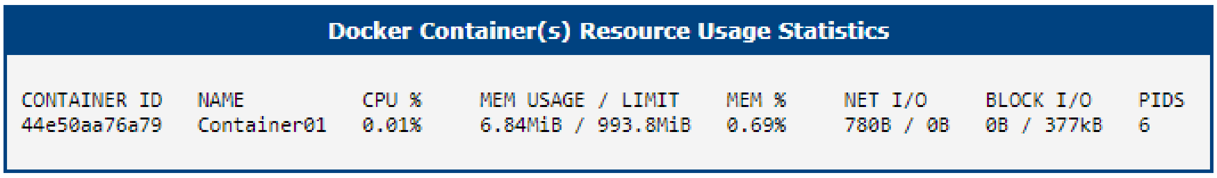
DIGEST	OS/ARCH	COMPRESSED SIZE
ba80c9cdf516	linux/amd64	168.09 MB
ec419810712b	linux/arm/v6	153.68 MB
2c31c844910c	linux/arm/v7	147.62 MB
259907078c3f	linux/arm64	157.69 MB
a6813238dd5e	linux/s390x	156.05 MB

Figure 7: Architectures supported by the image

[1] https:\hub.docker.com

2.1.2 Statistics

In this section we could see statistics of running containers



Docker Container(s) Resource Usage Statistics

```
CONTAINER ID   NAME          CPU %     MEM USAGE / LIMIT     MEM %     NET I/O      BLOCK I/O    PIDS
44e50aa76a79   Container01   0.01%     6.84MiB / 993.8MiB    0.69%     780B / 0B    0B / 377kB   6
```

Figure 8: Statistics

Column	Description
CONTAINER ID	Id of the container
NAME	Name of the container
CPU %	Actual CPU usage
MEM USAGE / LIMIT	Actual memory used / Limit of the memory used
MEM %	Actual memory used from limit in percent
NET I/O	Actual network operations
BLOCK I/O	Actual drive operations
PIDS	Process IDs

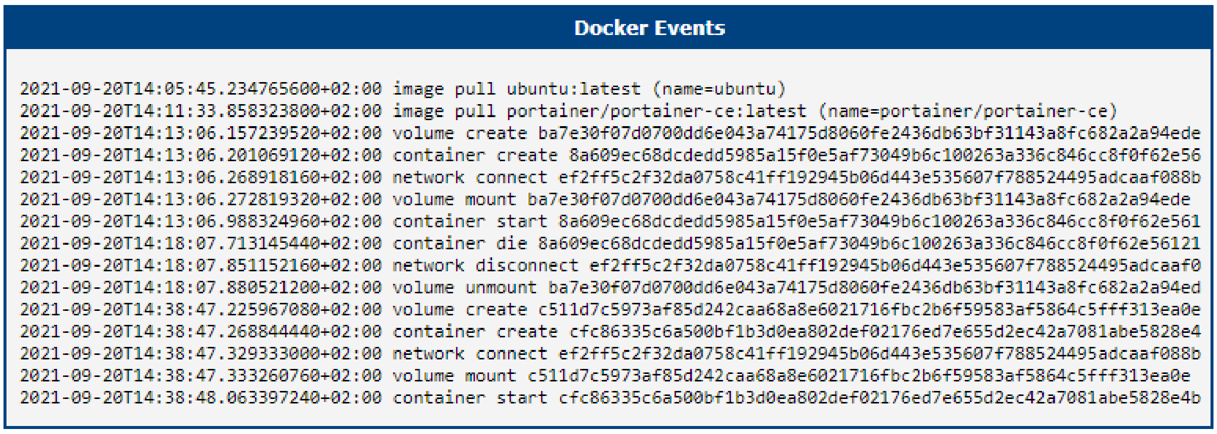
Table 1: Statistics columns

2.1.3 Log

This section contains detail log messages of Docker Router App divided into 2 parts *Containerd Log Messages* and *Dockerd Log Messages*

2.1.4 Events

In this section we could find list of Docker event raised by our usage.



Docker Events

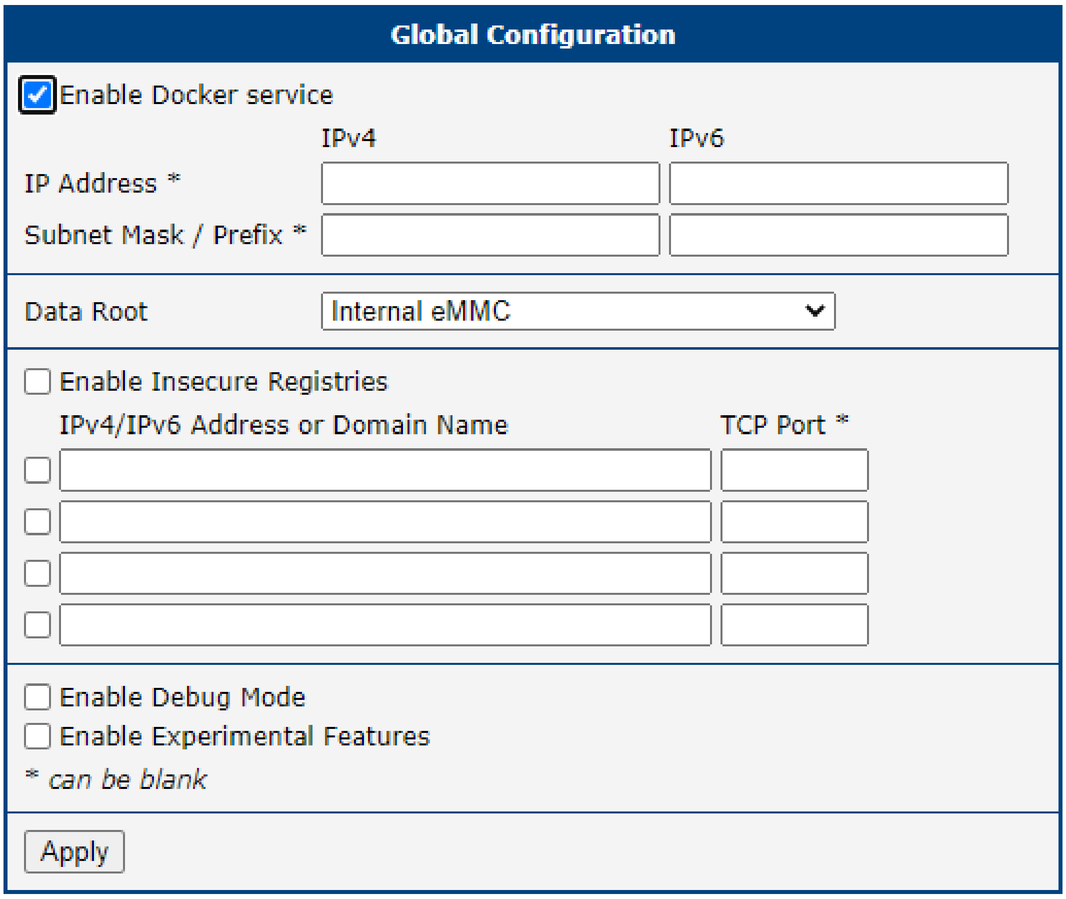
```
2021-09-20T14:05:45.234765600+02:00 image pull ubuntu:latest (name=ubuntu)
2021-09-20T14:11:33.858323800+02:00 image pull portainer/portainer-ce:latest (name=portainer/portainer-ce)
2021-09-20T14:13:06.157239520+02:00 volume create ba7e30f07d0700dd6e043a74175d8060fe2436db63bf31143a8fc682a2a94ede
2021-09-20T14:13:06.201069120+02:00 container create 8a609ec68dcdedd5985a15f0e5af73049b6c100263a336c846cc8f0f62e56
2021-09-20T14:13:06.268918160+02:00 network connect ef2ff5c2f32da0758c41ff192945b06d443e535607f788524495adcaaf088b
2021-09-20T14:13:06.272819320+02:00 volume mount ba7e30f07d0700dd6e043a74175d8060fe2436db63bf31143a8fc682a2a94ede
2021-09-20T14:13:06.988324960+02:00 container start 8a609ec68dcdedd5985a15f0e5af73049b6c100263a336c846cc8f0f62e561
2021-09-20T14:18:07.713145440+02:00 container die 8a609ec68dcdedd5985a15f0e5af73049b6c100263a336c846cc8f0f62e56121
2021-09-20T14:18:07.851152160+02:00 network disconnect ef2ff5c2f32da0758c41ff192945b06d443e535607f788524495adcaaf0
2021-09-20T14:18:07.880521200+02:00 volume unmount ba7e30f07d0700dd6e043a74175d8060fe2436db63bf31143a8fc682a2a94ed
2021-09-20T14:38:47.225967080+02:00 volume create c511d7c5973af85d242caa68a8e6021716fbc2b6f59583af5864c5fff313ea0e
2021-09-20T14:38:47.268844440+02:00 container create cfc86335c6a500bf1b3d0ea802def02176ed7e655d2ec42a7081abe5828e4
2021-09-20T14:38:47.329333000+02:00 network connect ef2ff5c2f32da0758c41ff192945b06d443e535607f788524495adcaaf088b
2021-09-20T14:38:47.333260760+02:00 volume mount c511d7c5973af85d242caa68a8e6021716fbc2b6f59583af5864c5fff313ea0e
2021-09-20T14:38:48.063397240+02:00 container start cfc86335c6a500bf1b3d0ea802def02176ed7e655d2ec42a7081abe5828e4b
```

Figure 9: Docker events

2.2 Configuration

2.2.1 Global

Docker Router App configuration is placed in *Global* section.



Global Configuration

- [✓] Enable Docker service

	IPv4	IPv6
IP Address *		
Subnet Mask / Prefix *		

Data Root: Internal eMMC

- [ ] Enable Insecure Registries

	IPv4/IPv6 Address or Domain Name	TCP Port *
[ ]		
[ ]		
[ ]		
[ ]		

- [ ] Enable Debug Mode
- [ ] Enable Experimental Features

* *can be blank*

Apply

Figure 10: Global configuration

Item	Description
Enable Docker Service	Enables Docker functionality. This option alone suffice for using Docker Router App

Continued on the next page

Continued from previous page

Item	Description
IP Address	Docker itself makes up his own IPv4 address, if it does suit your need for any reason, you can fill IP address you want. For IPv6 its little different - Docker won't make a IPv6 address and if you want one, you have to add one here in IPv6 field.
Subnet Mask / Prefix	Similar to IP Address above - in case you want some specific Subnet Mask / Prefix, you should specify it here.
Data Root	Dynamically generated list of options where is possible to create data root for Docker Router App. This list could differ on different routers. All possible options are: • Internal eMMC • SD Card • USB Flashdrive • SATA harddrive (*only on v4 routers*) **There has to be supported filesystem on selected Data Root (ext2, ext3, ext4)!**
Enable Insecure Registries	Enable if you have your own custom reigistry without https access. You can add up to 4 registries.
Enables Debug Mode	When enabled, extra data will be written out to logs.
Enables Experimental Features	Possibility to enable experimenta features.

Table 2: Configuration items description

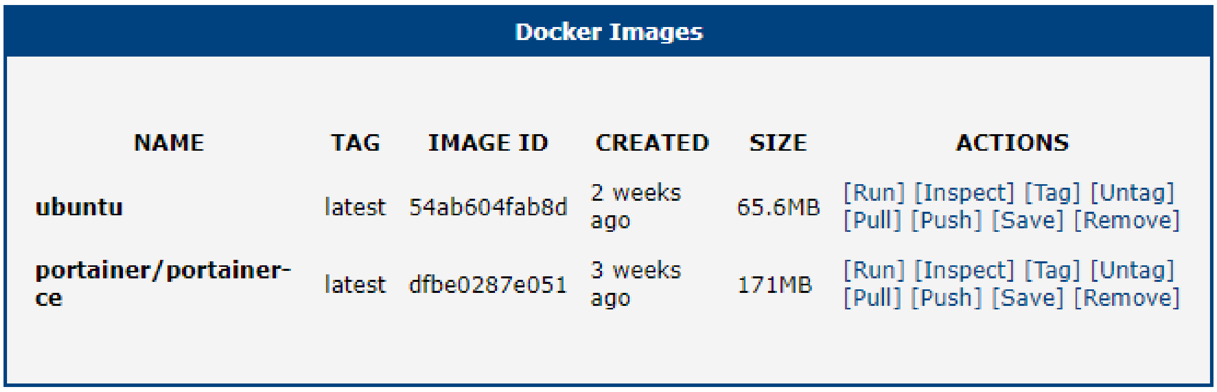
2.3 Administration

This section allows you administer your Images, Containers, Volumes and lets you Login and Logout to and from your preferred registry.

All actions and operation from *Administration* section are available only when *Docker* is enabled and running.

2.3.1 Images

Here you will find the list with your available Images.



Docker Images

NAME	TAG	IMAGE ID	CREATED	SIZE	ACTIONS
ubuntu	latest	54ab604fab8d	2 weeks ago	65.6MB	[Run] [Inspect] [Tag] [Untag] [Pull] [Push] [Save] [Remove]
portainer/portainer-ce	latest	dfbe0287e051	3 weeks ago	171MB	[Run] [Inspect] [Tag] [Untag] [Pull] [Push] [Save] [Remove]

Figure 11: Images

Column	Description
Name	Name of the container
Tag	Tag of the container
Image ID	ID of the image
Created	Date of creation of image
Size	Size of the image
Actions	Possible action with image

Table 3: Docker Images columns

Images actions

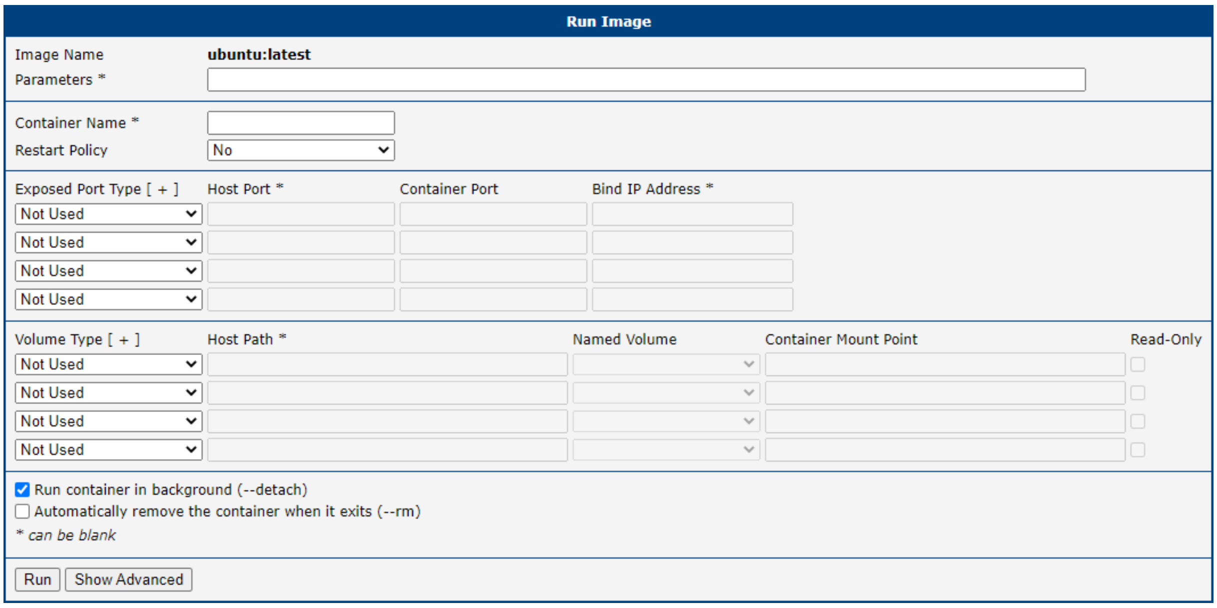
Each Image has 8 actions available.

Item	Description
Run	Runs the image and creates the container
Inspect	Display detailed information of image
Tag	Creates a tag
Untag	Removes a tag, removing last tag will at the same time remove whole image
Pull	Pull an image from a registry
Push	Push an image to a registry
Save	Saves image to a tar archive (streamed to STDOUT by default)
Remove	Remove image. Image can't be removed when it has multiple tags or container from this image already exists.

Table 4: Image actions

Run Image

The run action is used to mention that we want to create an instance of an image, which is then called a container. After clicking on action *Run* the options of this action will show. There are 2 levels of options available *Basic* and *Advanced*. Basic options are displayed everytime, but Advanced are shown only after clicking on the button *Show Advanced*. Run Image is the key part of Docker Router App. We have Image and we need to run the Image and create the container. In simplest case we click on Run and *something* happens - either it will run, or it won't and we get some kind of error which helps with running the image. Worst case scenario is that it won't run. Some images needs parameters to be filled. The way, how to run an image typically can not be guessed. Author of the image have to give you some rules how to run the image corectly.



Run Image

Image Name	**ubuntu:latest**			
Parameters *				
Container Name *				
Restart Policy	No			
Exposed Port Type [+]	Host Port *	Container Port	Bind IP Address *	
Not Used				
Not Used				
Not Used				
Not Used				
Volume Type [+]	Host Path *	Named Volume	Container Mount Point	Read-Only
Not Used				☐
Not Used				☐
Not Used				☐
Not Used				☐

☑ Run container in background (--detach)

☐ Automatically remove the container when it exits (--rm)

** can be blank*

Figure 12: Image Run

Item	Description
Parameters	Parameters for the container or the service that run inside.
Container Name	Name of the container, if left blank, random name will be given.

Continued on the next page

Continued from previous page

Item	Description
Restart Policy	Specify a restart policy for how a container should or should not be restarted on exit. Options are: • **no** - Do not automatically restart the container when it exits. This is the default. • **on-failure** - Restart only if the container exits with a non-zero exit status. Optionally, limit the number of restart retries the Docker daemon attempts. • **unless-stopped** - Always restart the container regardless of the exit status. When you specify always, the Docker daemon will try to restart the container indefinitely. The container will also always start on daemon startup, regardless of the current state of the container. • **always** - Always restart the container regardless of the exit status, including on daemon startup, except if the container was put into a stopped state before the Docker daemon was stopped.
Exposed Port Type	Map or bind ports where the container should run od listen. Container port is most important, its sufficient to specify only where the container should run, if its the same on routers it does not need to be specified, if you want some other, just fill it in. If you want to specify to not listening everywhere, but only on some interface, fill the address of the interface where it should listen.
Volume Type	Data storage for the container. For example database need to store data somewhere or you need to take some files from the router to the container, this is the section where to map it. There are two options • Host path - path is specified • Named Volume - volume is selected from the available volumes

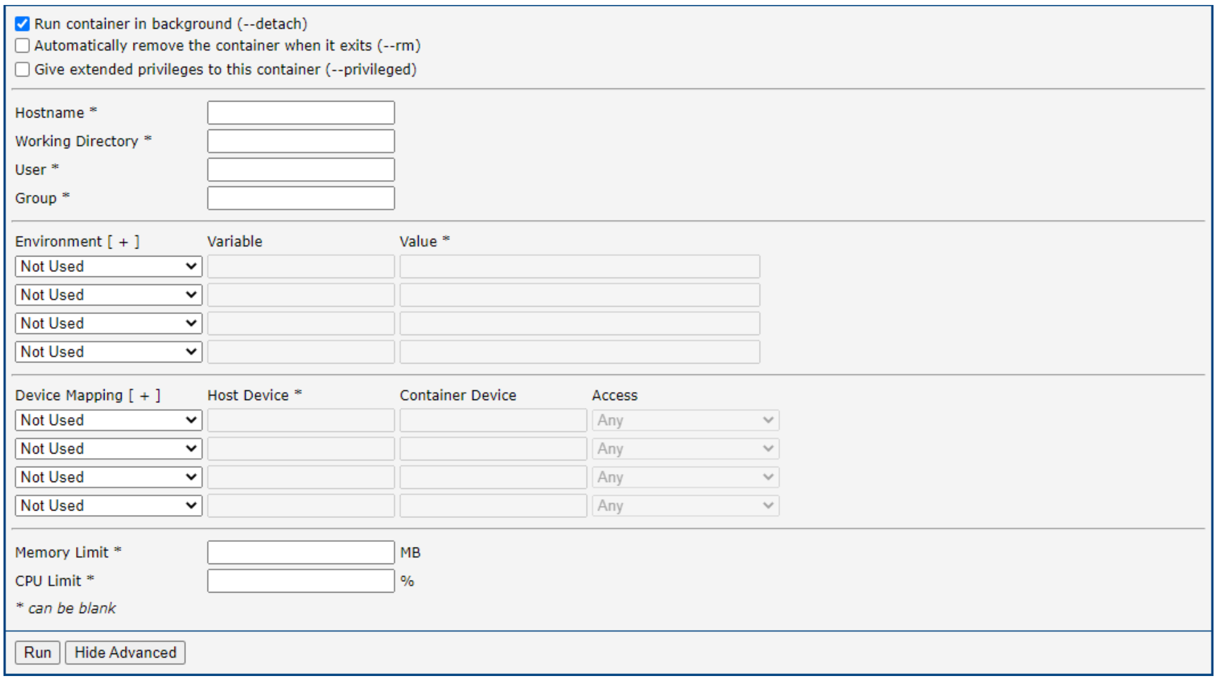
Continued on the next page

Continued from previous page

Item	Description
Run Container in background	If enabled, the service runs on background. By default is this option enabled, in most cases we want to services to be ready to fulfil its purpose.
Automatically remove the container when it exists	When container ends, it stays in the list and theoretically can be started again, but there are cases, where you do some one-time operation and there no need to container stays in list to be removed manually so you can check this option even before it starts and save some time.

Table 5: Run Image Basic Options

Options from the Advanced part are mostly optional, therefore are hidden at first to make the Run action clearer.



- [✓] Run container in background (--detach)
- [ ] Automatically remove the container when it exits (--rm)
- [ ] Give extended privileges to this container (--privileged)

Hostname *	
Working Directory *	
User *	
Group *	

Environment [+]	Variable	Value *
Not Used		
Not Used		
Not Used		
Not Used		

Device Mapping [+]	Host Device *	Container Device	Access
Not Used			Any
Not Used			Any
Not Used			Any
Not Used			Any

Memory Limit *		MB
CPU Limit *		%

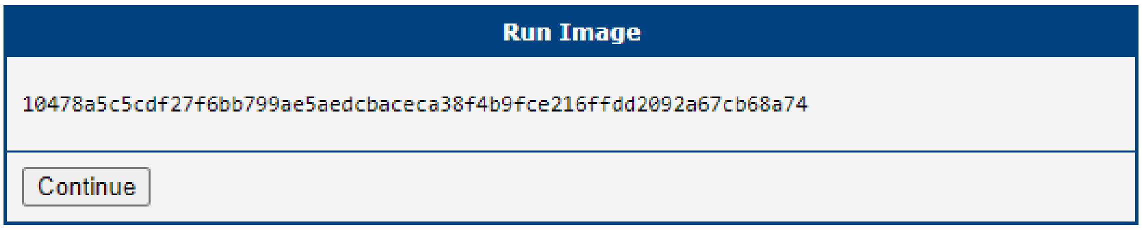
* *can be blank*

Figure 13: Image Run Advanced

Item	Description
Give extended privileges to this container	The –privileged flag gives all capabilities to the container, and it also lifts all the limitations enforced by the device cgroup controller. In other words, the container can then do almost everything that the host can do. This flag exists to allow special use-cases, like running Docker within Docker.
Hostname	Enter the hostname of the container.
Working Directory	It is possible to select working directory, where container is running e.g. if you want the container to start in /tmp/ directory to handle some temporary files, it could be done with this option.
User	Specify the user under which the container will run.
Group	Specify the group under which the container will run.
Environment	Set simple (non-array) environment variables in the container you're running, or overwrite variables that are defined in the Dockerfile of the image you're running.
Device Mapping	It is often necessary to directly expose devices to a container. This option option enables that. For example, a specific block storage device or loop device or audio device can be added to an otherwise unprivileged container (without the –privileged flag) and have the application directly access it.
Memory Limit	Memory limit given to the container. Can be seen in Container list. Default value is maximum memory usage and that could be undesirable, therefore you can specify the amount needed.
CPU Limit	Specify how much CPU power you want to give to the container. Its in percent and 1 core means 100%, so in case 4-core v4 router is 400% absolute maximum and 200% means that container could use 2 cores.

Table 6: Run Image Advanced Options

As a result of running an image we get a Container ID. More about Containers will be described below in *Containers* section.



```
10478a5c5cdf27f6bb799ae5aedcbaceca38f4b9fce216ffdd2092a67cb68a74
```

Figure 14: Image run result

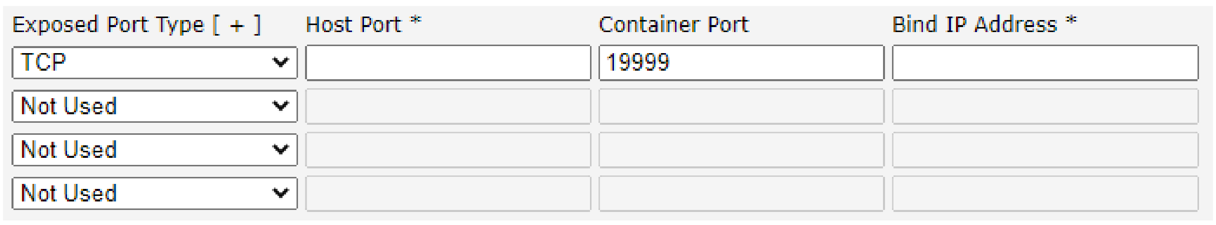
Transforming Docker Run Command into Run Image action

Let's say we want to run Image of for example *netdata*. So we search for it and pull the first one with most ratings. As we stated earlier, we have to find some rules how to run this Image. So we search the Docker Hub [2], find this Image and go to the netdata official site and here we have the example we were looking for:

```
docker run -d --name=netdata \
-p 19999:19999 \
-v netdataconfig:/etc/netdata \
-v netdatalib:/var/lib/netdata \
-v netdatacache:/var/cache/netdata \
-v /etc/passwd:/host/etc/passwd:ro \
-v /etc/group:/host/etc/group:ro \
-v /proc:/host/proc:ro \
-v /sys:/host/sys:ro \
-v /etc/os-release:/host/etc/os-release:ro \
--restart unless-stopped \
--cap-add SYS_PTRACE \
--security-opt apparmor=unconfined \
netdata/netdata
```

First line alone will run, but it won't do what we need it to do and that is the reason why there are those extra parameters, which tells what to do/from where take data/how it should behave/what it should do if it crashes/other necessities and thats what we need se set in our Router App in Run Image action.

Second line *-p 19999:19999* is port, the first 19999 defines on which port on our router it will be visible and the second defines on which port the service is running in container. In this case ports are the same, but you can remap those ports as you wish. As mentioned in table above, when the ports are identical it is sufficient to fill only *Container Port*. Ports are TCP as default, when not stated otherwise. So for our example - this is how the ports section should look like.



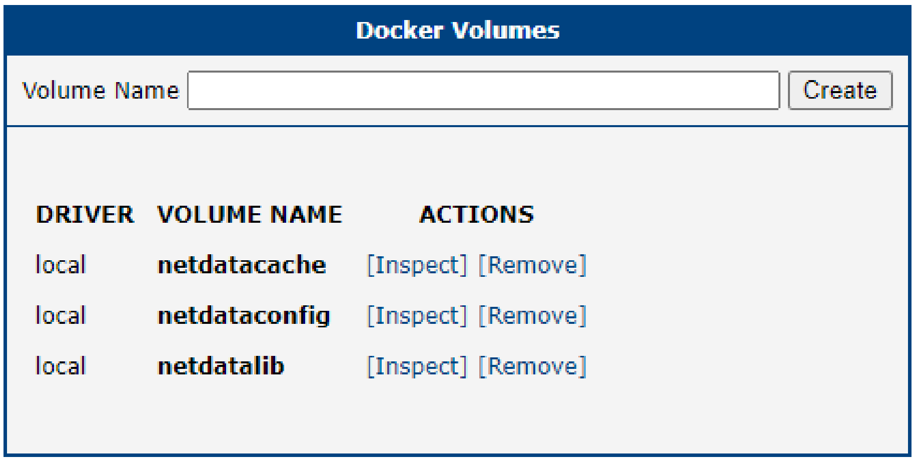
Exposed Port Type [+]	Host Port *	Container Port	Bind IP Address *
TCP		19999	
Not Used			
Not Used			
Not Used			

Figure 15: Image run ports

Lines 3-10 are about volumes. When the argument of the parameter *-v* begin with slash then its *Host Path* Volume Type, otherwise it is *Named Volume*. When selecting one of those

[2] https:\hub.docker.com

options in Volume Type section of Run Image section, some fields grays up to indicate how the field should be filled in. On how to create *Named Volume* consult the section 2.3.3. When all Volumes needed for netdata image are done, the *Docker Volumes* section should look like this:



Docker Volumes

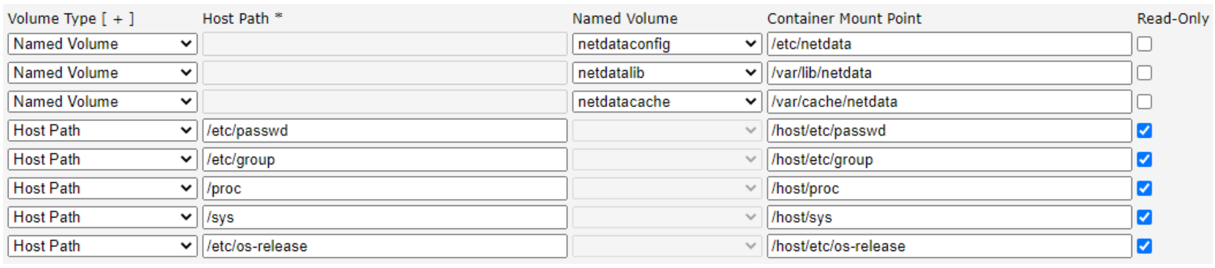
Volume Name [ ] Create

DRIVER	VOLUME NAME	ACTIONS
local	netdatacache	[Inspect] [Remove]
local	netdataconfig	[Inspect] [Remove]
local	netdatalib	[Inspect] [Remove]

Figure 16: Image run volumes

Lines 3-5 are Named Volumes, so we select this option and choose right one from the now active dropdown menu. The *Container Mount Point* is the second part of the argument, right after colon.

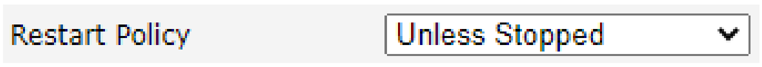
Lines 6-10 are *Host Path* volumes. So we select this option and similarly to *Named Volumes* we split the arguments into *Host Path* and *Container Mount Point* inputs. We can see, that on the end of each line is another colon with *ro* - this is *Read-Only* so we check the checkbox for those lines. The result should look like this:



Volume Type [+]	Host Path *	Named Volume	Container Mount Point	Read-Only
Named Volume		netdataconfig	/etc/netdata	☐
Named Volume		netdatalib	/var/lib/netdata	☐
Named Volume		netdatacache	/var/cache/netdata	☐
Host Path	/etc/passwd		/host/etc/passwd	☑
Host Path	/etc/group		/host/etc/group	☑
Host Path	/proc		/host/proc	☑
Host Path	/sys		/host/sys	☑
Host Path	/etc/os-release		/host/etc/os-release	☑

Figure 17: Image run netdata volumes

Last thing worth mentioning is the line 11. According to this line we can set the restart policy like that:



Restart Policy: Unless Stopped

Figure 18: Image run volumes

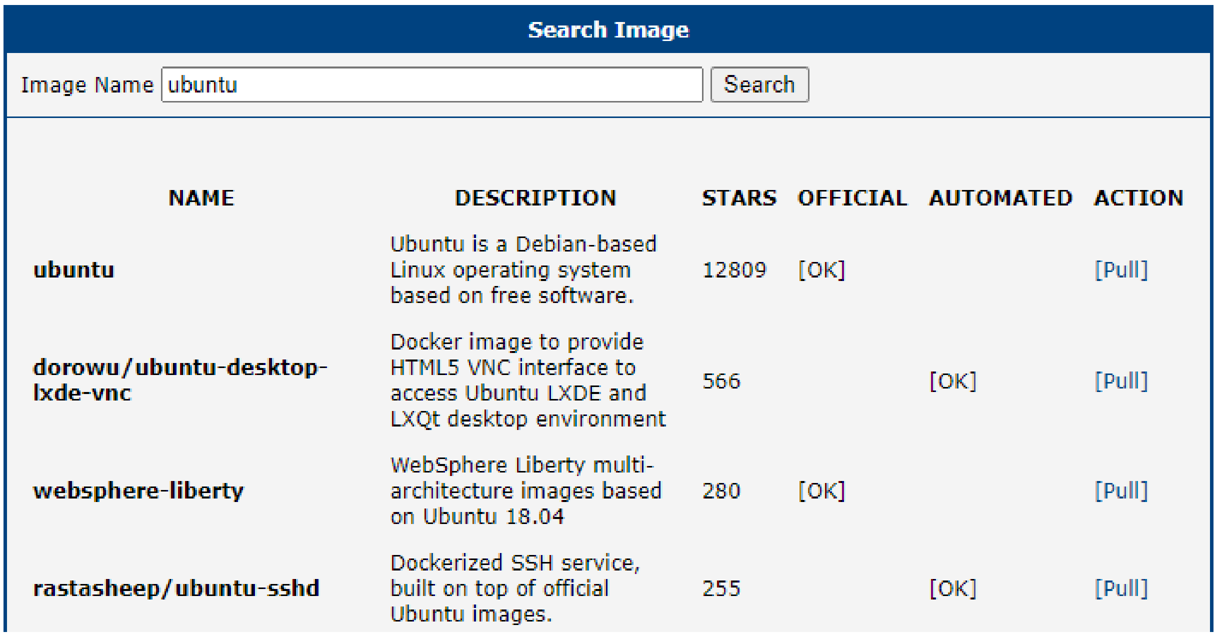
Lines 12 and 13 we can skip. We don't have items for them in our *Run Image* dialog window as they are not essential and the Image will run just fine without them.

And thats it. We can now hit the *Run* button and Run our netdata image and create container.

Search Image

Working connection to the internet and to the router is needed for use of the *Search Image* function

There are three ways how to get Images to our router. First is via *Search Image*. In this case you are able to search images directly in oficial repository, just type in the image name and browse the results. In this case we are searching for images *ubuntu*, you can see results of our search below:



NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED	ACTION
ubuntu	Ubuntu is a Debian-based Linux operating system based on free software.	12809	[OK]		[Pull]
dorowu/ubuntu-desktop-lxde-vnc	Docker image to provide HTML5 VNC interface to access Ubuntu LXDE and LXQt desktop environment	566		[OK]	[Pull]
websphere-liberty	WebSphere Liberty multi-architecture images based on Ubuntu 18.04	280	[OK]		[Pull]
rastasheep/ubuntu-sshd	Dockerized SSH service, built on top of official Ubuntu images.	255		[OK]	[Pull]

Figure 19: Image search

Official repository and the base source of images is Docker Hub - `https://hub.docker.com/`

Column	Description
Name	Name of the image
Description	Description of the image
Stars	Rating of the image
Official	Official image. These images have clear documentation, promote best practices, and are designed for the most common use cases.

Continued on the next page

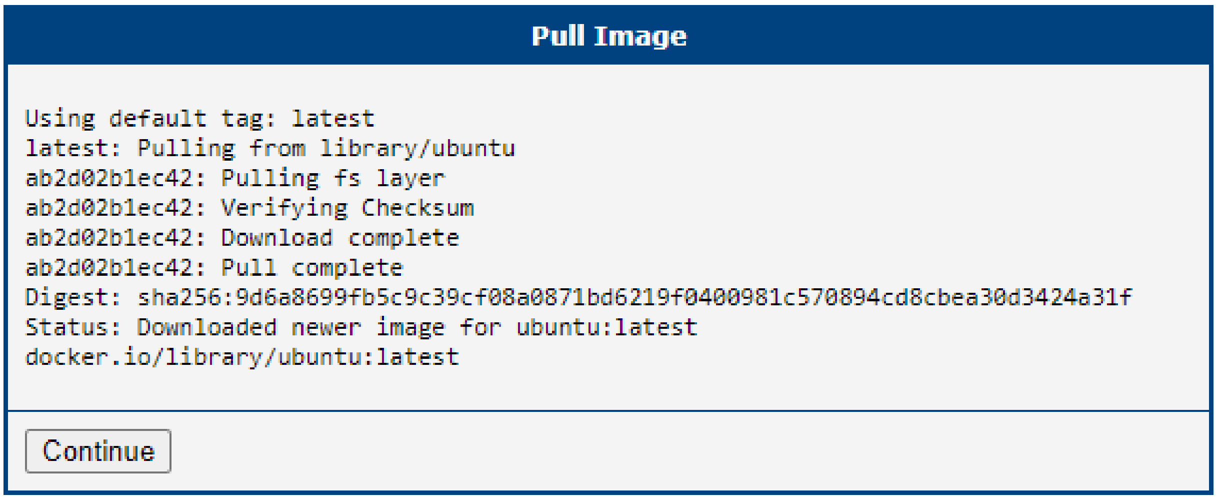
Continued from previous page

Column	Description
Automated	Docker Hub can automatically build images from source code in an external repository and automatically push the built image to your Docker repositories.
Action	Actions available to the image

Table 7: Search image columns

When we choose the image we want to use, we just click on the *Pull* action on the right side of the result view and the image gets pulled to our router

Depending on the Image size, pull could take up to few minutes to download.

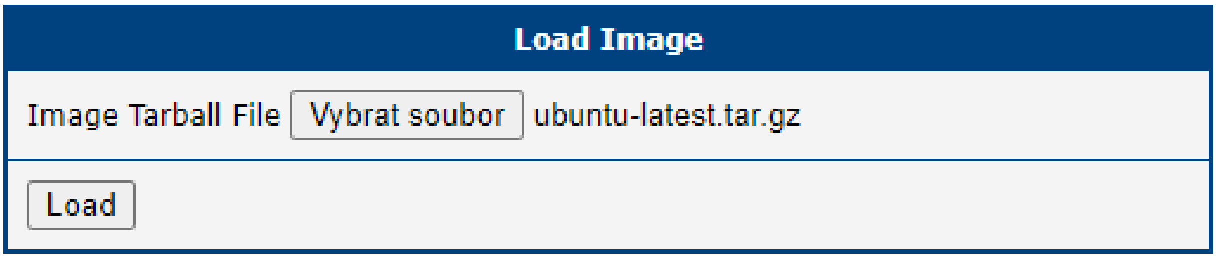


```
Using default tag: latest
latest: Pulling from library/ubuntu
ab2d02b1ec42: Pulling fs layer
ab2d02b1ec42: Verifying Checksum
ab2d02b1ec42: Download complete
ab2d02b1ec42: Pull complete
Digest: sha256:9d6a8699fb5c9c39cf08a0871bd6219f0400981c570894cd8cbea30d3424a31f
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

Figure 20: Pull image

Load Image

Another way how to obtain images is to Load them directly from our file system. You just need the tar archive of the image and you are ready to go. This way it is possible to load image saved by the action *Save* mentioned in section 2.3.1.



Load Image

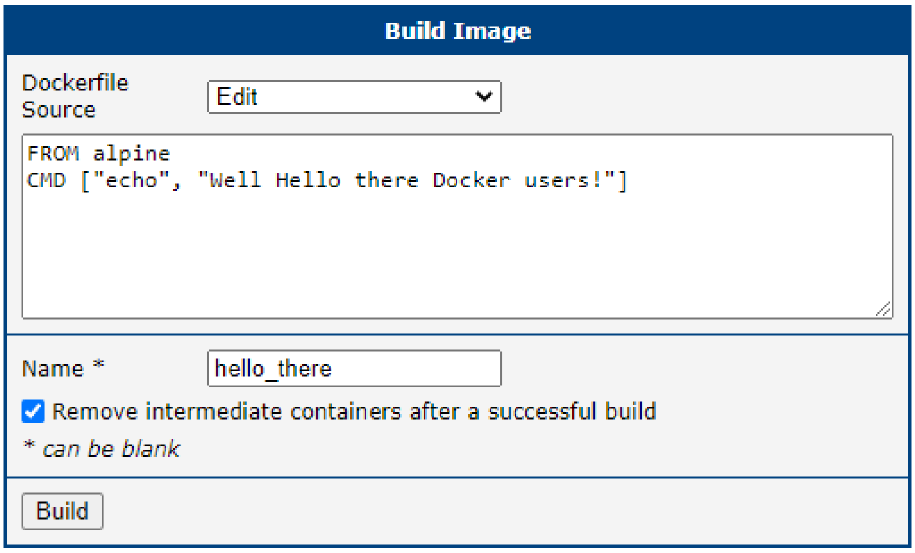
Image Tarball File: Vybrat soubor ubuntu-latest.tar.gz

Figure 21: Load image

Build Image

Third way ho to get Docker Image into our router is via *Build Image* option. There are 2 option from which you can choose while building image - first one is *Edit*, where you can write the Docker Image definition i.e. *dockerfile* by yourself. Many projects e.g. on GitHub offers already prepared dockerfiles.

Lets build a simple Image that only greets the users in the log.



Build Image

Dockerfile Source: Edit

```
FROM alpine
CMD ["echo", "Well Hello there Docker users!"]
```

Name *: hello_there

☑ Remove intermediate containers after a successful build

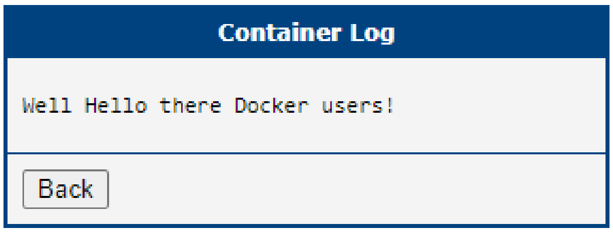
** can be blank*

Figure 22: Images Build Edit

Item	Description
Name	Name of the image which will be displayed in the image list
Remove intermediate containers after succesfull build	Support container can be created during the build, in most cases those are not useful and theres no need to keep those so this option allows to remove then automatically.

Table 8: Search image columns

After building the image, running the image and inspecting log of freshly created container we get this heartwarming greeting

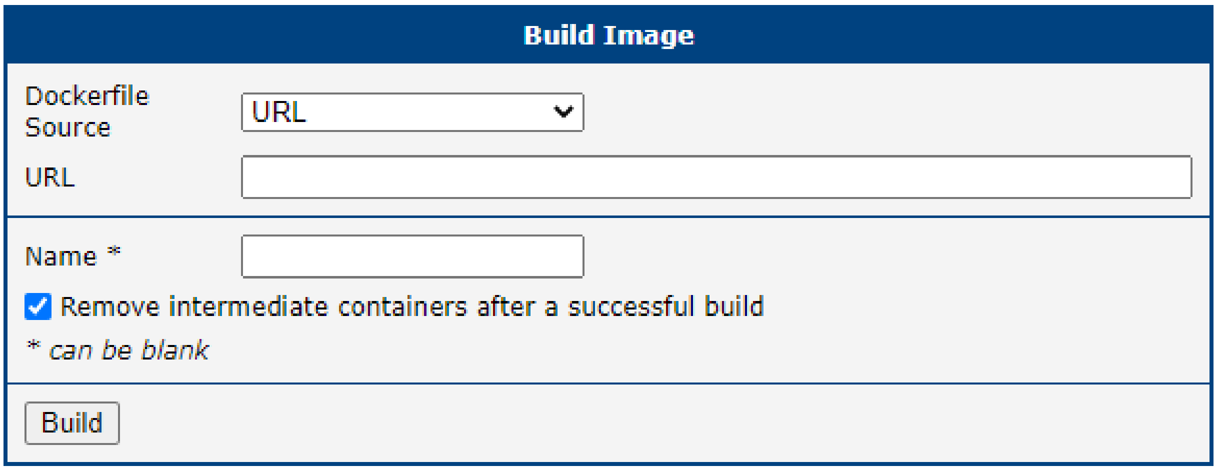


```
Well Hello there Docker users!
```

Figure 23: Container Log

For tips and hints how to write dockerfiles see `https://docs.docker.com/develop/develop-images/dockerfile_best-practices/`

And second is *URL*, where you can insert URL of the *dockerfile*.



Dockerfile Source	URL
URL	
Name *	

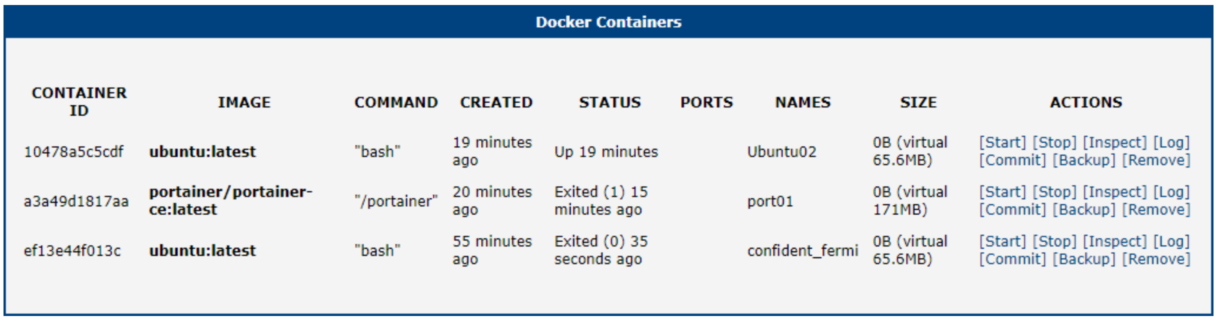
[✓] Remove intermediate containers after a successful build

** can be blank*

Figure 24: Images Build Url

2.3.2 Containers

This section contains list of available Containers



Docker Containers

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES	SIZE	ACTIONS
10478a5c5cdf	**ubuntu:latest**	"bash"	19 minutes ago	Up 19 minutes		Ubuntu02	0B (virtual 65.6MB)	[Start] [Stop] [Inspect] [Log] [Commit] [Backup] [Remove]
a3a49d1817aa	**portainer/portainer-ce:latest**	"/portainer"	20 minutes ago	Exited (1) 15 minutes ago		port01	0B (virtual 171MB)	[Start] [Stop] [Inspect] [Log] [Commit] [Backup] [Remove]
ef13e44f013c	**ubuntu:latest**	"bash"	55 minutes ago	Exited (0) 35 seconds ago		confident_fermi	0B (virtual 65.6MB)	[Start] [Stop] [Inspect] [Log] [Commit] [Backup] [Remove]

Figure 25: Available containers

Column	Description
Container ID	ID of the container
Image	Source image
Command	Command created after container is built. We can after use this command to get to the functionality of container.
Created	Container creation date.
Status	Actual status of the container
Ports	Ports declared in the Run Image action
Names	Name of the container
Size	Actual size of the container
Actions	Actions available with the container

Table 9: Search image columns

Containers Actions

Each Container has 7 actions available.

Item	Description
Start	Start stopped container
Stop	Stop running container
Inspect	Display detailed information on container
Log	Fetch the logs of a container
Commit	Create a new image from a container's changes
Backup	Saves a container and all its data as a tar file
Remove	Remove container

Table 10: Container actions

Restore Container

Restores a backed up container. After using Backup action on container, you'll get a tar file containing a backup of this container. In this section you can restore the container and use it afterwards.

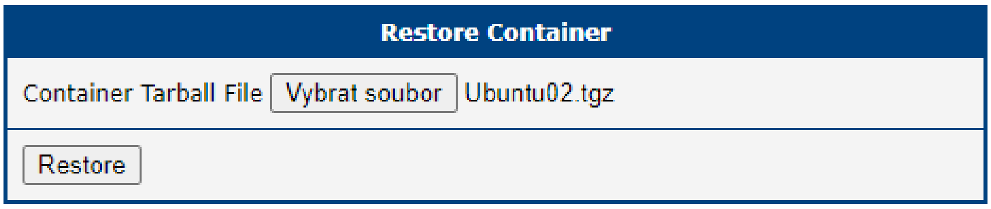
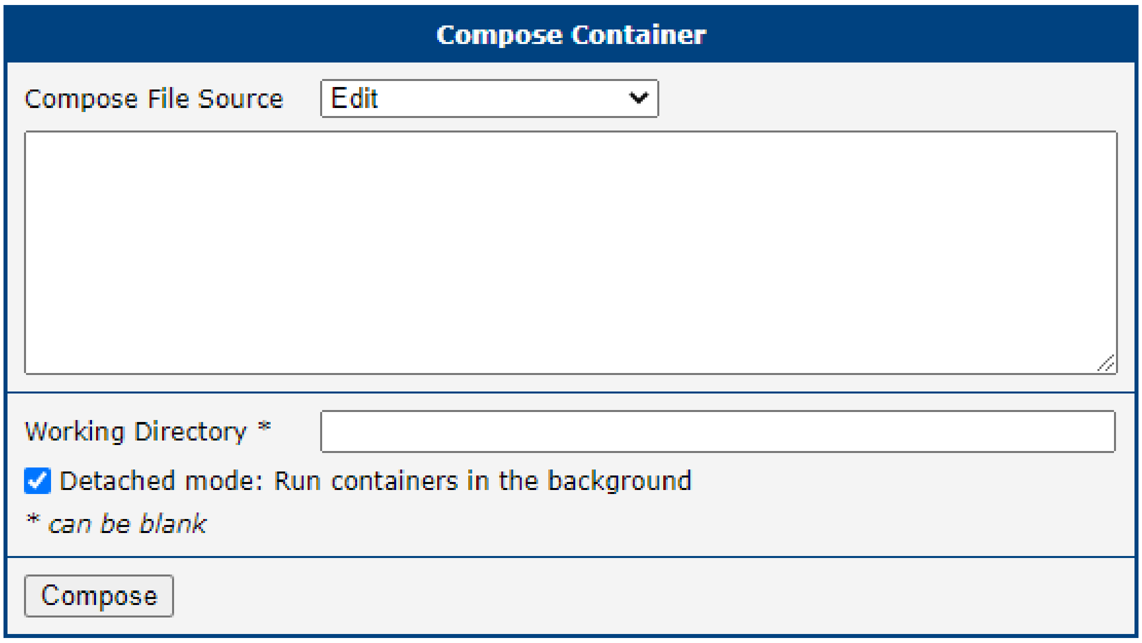


Figure 26: Restore container

Compose Container

This option is similar as the *Build Image* - you can write a definition of container and use it. Similarily as in case of Image there are 2 option from which you can choose while compose container - first one is *Edit*, where you can write the Docker Image definition by yourself.


Compose Container

Compose File Source: Edit

Working Directory *

[✓] Detached mode: Run containers in the background

* can be blank

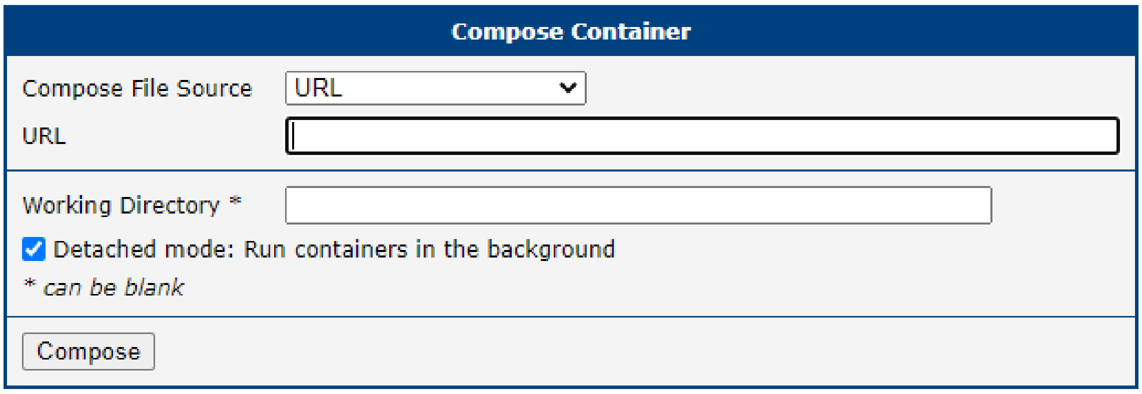
Figure 27: Compose container

Item	Description
Working directory	Directory, where the container will run. Typically needed when getting data from the router, the container expect certain directory to work in.
Detached mode: Run containers in the background	Background/foreground container run switch.

Table 11: Compose container items

For tips and hints how to write composefiles see `https://docs.docker.com/compose/gettingstarted/`

And second is *URL*, where you can insert URL of Docker Container definition.



Compose File Source	URL
URL	
Working Directory *	

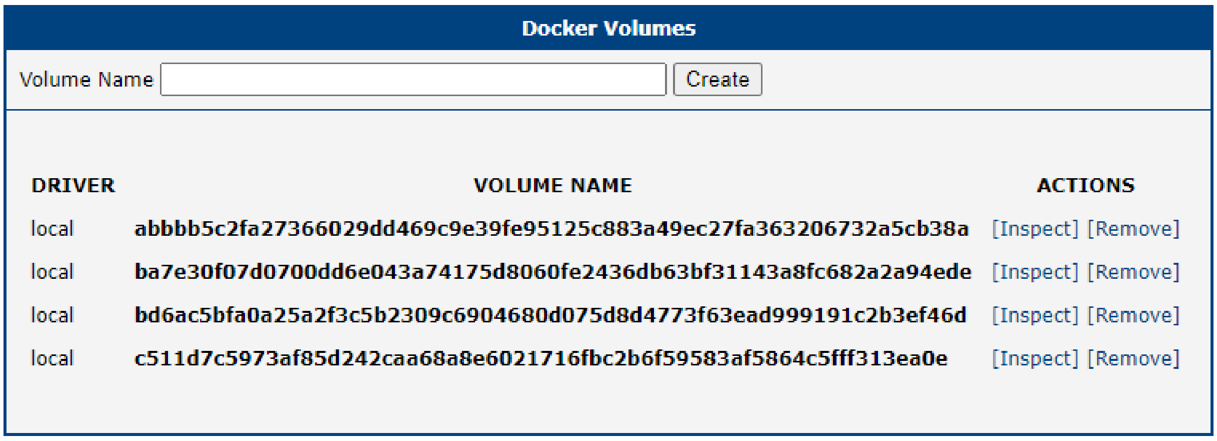
☑ Detached mode: Run containers in the background

** can be blank*

Figure 28: Compose container url

2.3.3 Volumes

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. In addition, volumes are often a better choice than persisting data in a container's writable layer, because a volume does not increase the size of the containers using it, and the volume's contents exist outside the lifecycle of a given container.



DRIVER	VOLUME NAME	ACTIONS
local	abbbb5c2fa27366029dd469c9e39fe95125c883a49ec27fa363206732a5cb38a	[Inspect] [Remove]
local	ba7e30f07d0700dd6e043a74175d8060fe2436db63bf31143a8fc682a2a94ede	[Inspect] [Remove]
local	bd6ac5bfa0a25a2f3c5b2309c6904680d075d8d4773f63ead999191c2b3ef46d	[Inspect] [Remove]
local	c511d7c5973af85d242caa68a8e6021716fbc2b6f59583af5864c5fff313ea0e	[Inspect] [Remove]

Figure 29: Volumes

To create a volume, simply enter volume name and hit the create button. After that, this volume will be available to be selected in *Run Image* Volume Type dropdown.

Volume actions

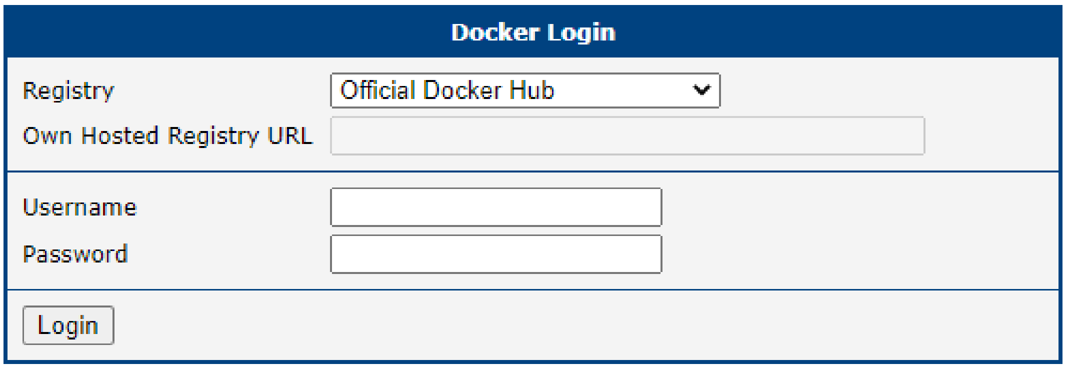
Each Volume has 2 actions available.

Item	Description
Inspect	Display detailed information on volume
Remove	Remove volume

Table 12: Volume actions

2.3.4 Login

There is possibility to Login to *Official Docker Hub* or your *Own Hosted* hub. In case of Official Docker Hub just fill in Username and Password and login. In case of Own Hosted hub you need to fill in the *Own Hosted Registry URL*. Login is persistent even after router restart. Only way how to remove *Login* is *Logout*.



Registry	Official Docker Hub
Own Hosted Registry URL	
Username	
Password	

Figure 30: Login

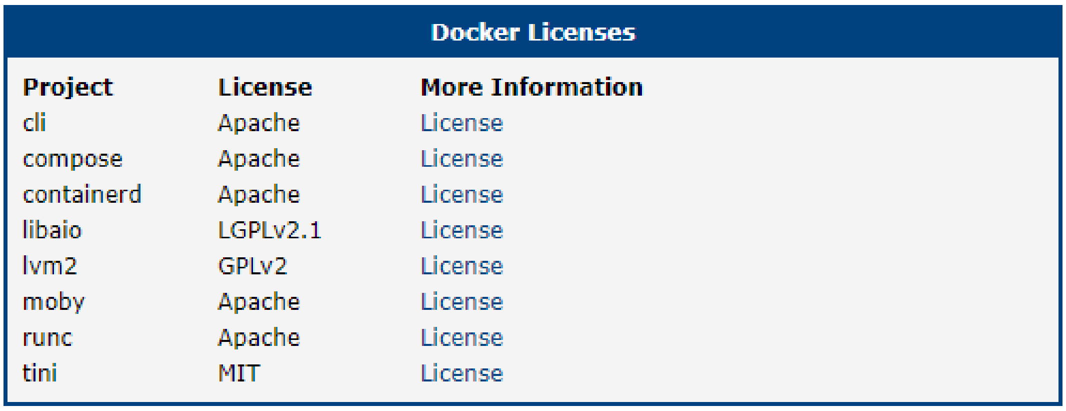
2.3.5 Logout

In this section you can Logout from previously logged in Registries.

2.4 Information

2.4.1 Licenses

Licenses used in Docker Router App are listed below.



Docker Licenses

Project	License	More Information
cli	Apache	License
compose	Apache	License
containerd	Apache	License
libaio	LGPLv2.1	License
lvm2	GPLv2	License
moby	Apache	License
runc	Apache	License
tini	MIT	License

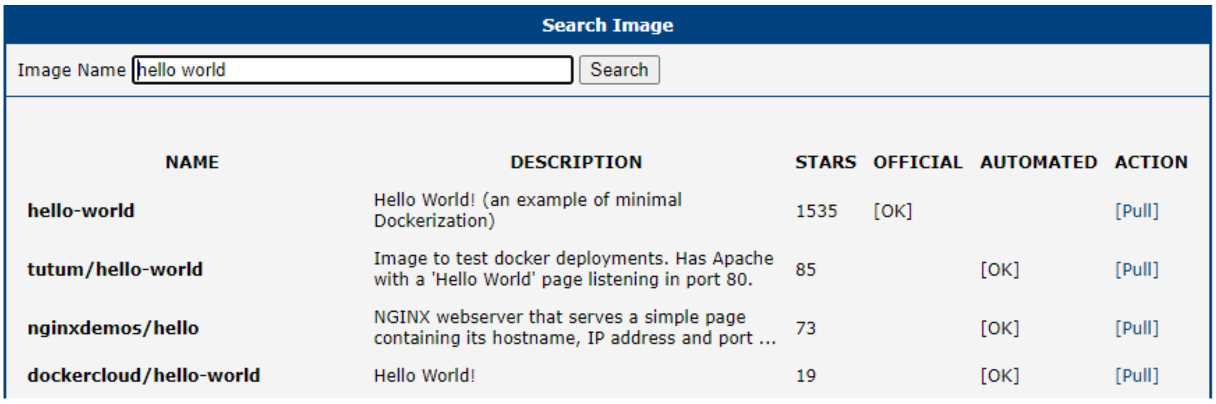
Figure 31: Licenses

3. Examples

In this chapter we focus on few examples on how we use Docker Router App. Those examples are step by step description on how to get the image, how to run it and what the result should be. Lets start with absolute basics - Hello World!

3.1 Example 1: Hello World

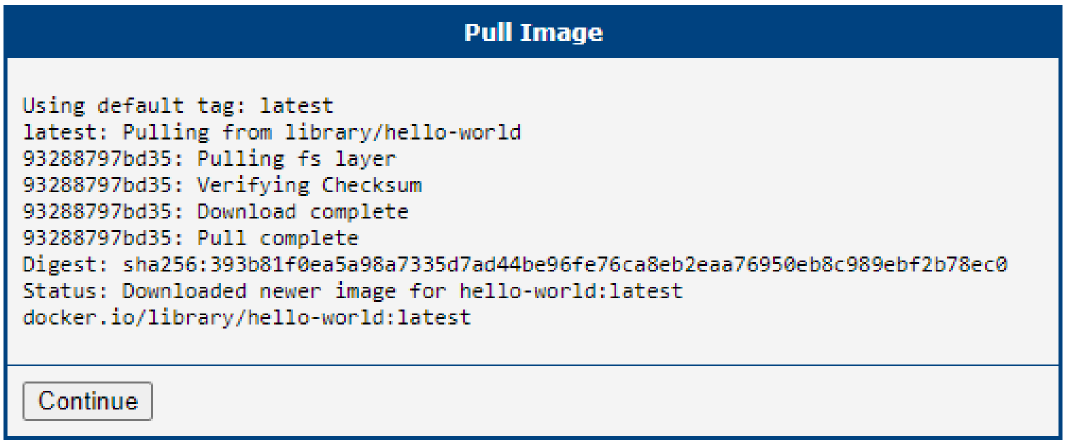
First step should always be finding the right Image - so lets head to the *Search Image* section and search for *Hello World*.



NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED	ACTION
hello-world	Hello World! (an example of minimal Dockerization)	1535	[OK]		[Pull]
tutum/hello-world	Image to test docker deployments. Has Apache with a 'Hello World' page listening in port 80.	85		[OK]	[Pull]
nginxdemos/hello	NGINX webserver that serves a simple page containing its hostname, IP address and port ...	73		[OK]	[Pull]
dockercloud/hello-world	Hello World!	19		[OK]	[Pull]

Figure 32: Hello World Image Search

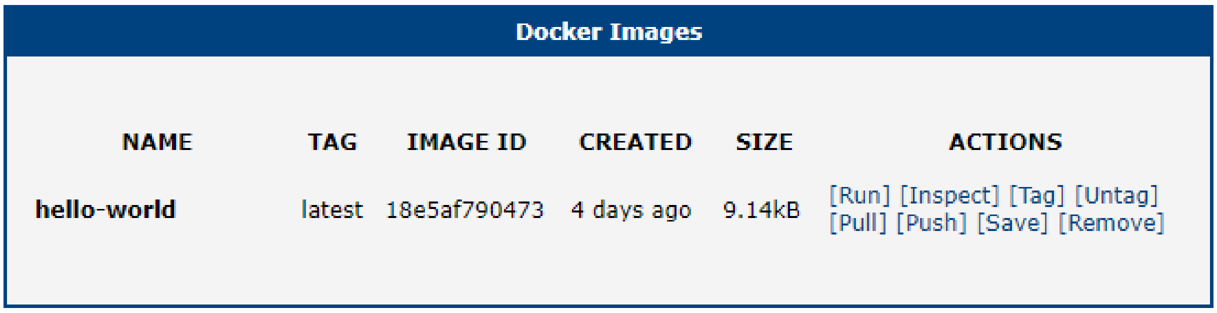
First one looks perfect, it has amount of start exceeding any other else, its from Official source, its what we want. So lets Pull it!



```
Using default tag: latest
latest: Pulling from library/hello-world
93288797bd35: Pulling fs layer
93288797bd35: Verifying Checksum
93288797bd35: Download complete
93288797bd35: Pull complete
Digest: sha256:393b81f0ea5a98a7335d7ad44be96fe76ca8eb2eaa76950eb8c989ebf2b78ec0
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
```

Figure 33: Hello World Image Pull

Image is pulled and ready to run.

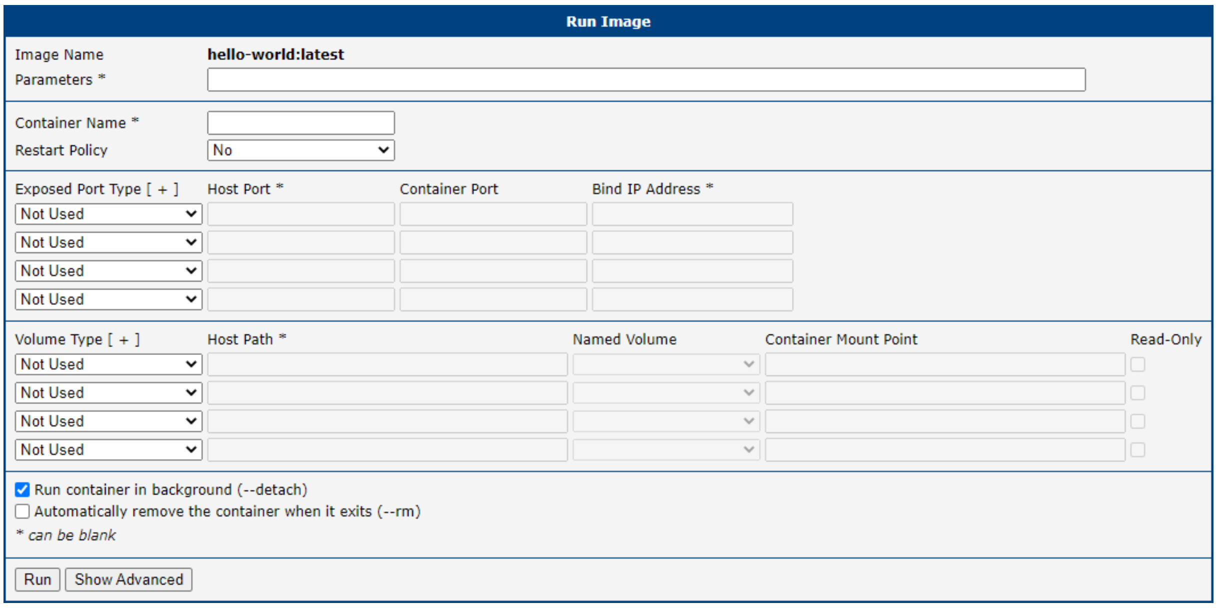


Docker Images

NAME	TAG	IMAGE ID	CREATED	SIZE	ACTIONS
hello-world	latest	18e5af790473	4 days ago	9.14kB	[Run] [Inspect] [Tag] [Untag] [Pull] [Push] [Save] [Remove]

Figure 34: Hello World Image

This Image is so simple, that we don't need to fill anything into Run Image options.



Run Image

Image Name	hello-world:latest
Parameters *	
Container Name *	
Restart Policy	No

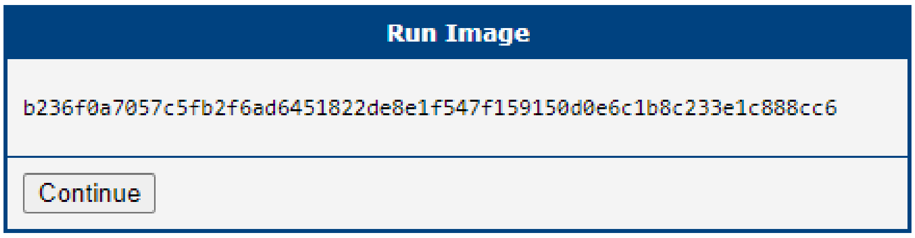
Exposed Port Type [+]	Host Port *	Container Port	Bind IP Address *
Not Used			
Not Used			
Not Used			
Not Used			

Volume Type [+]	Host Path *	Named Volume	Container Mount Point	Read-Only
Not Used				☐
Not Used				☐
Not Used				☐
Not Used				☐

☑ Run container in background (--detach)
☐ Automatically remove the container when it exits (--rm)
** can be blank*

Figure 35: Hello World Container Log

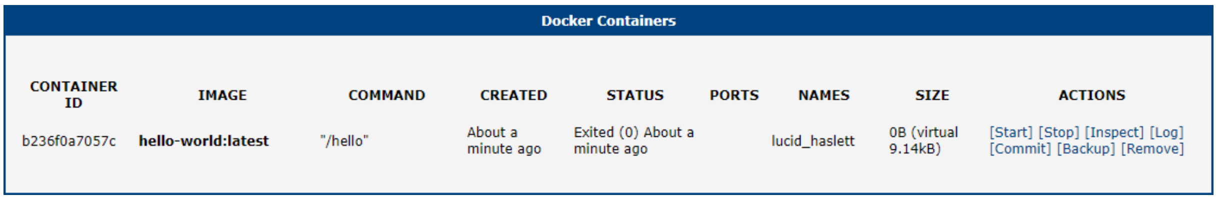
After running, we can see screen with full Container ID.



```
b236f0a7057c5fb2f6ad6451822de8e1f547f159150d0e6c1b8c233e1c888cc6
```

Figure 36: Hello World Image Run Done

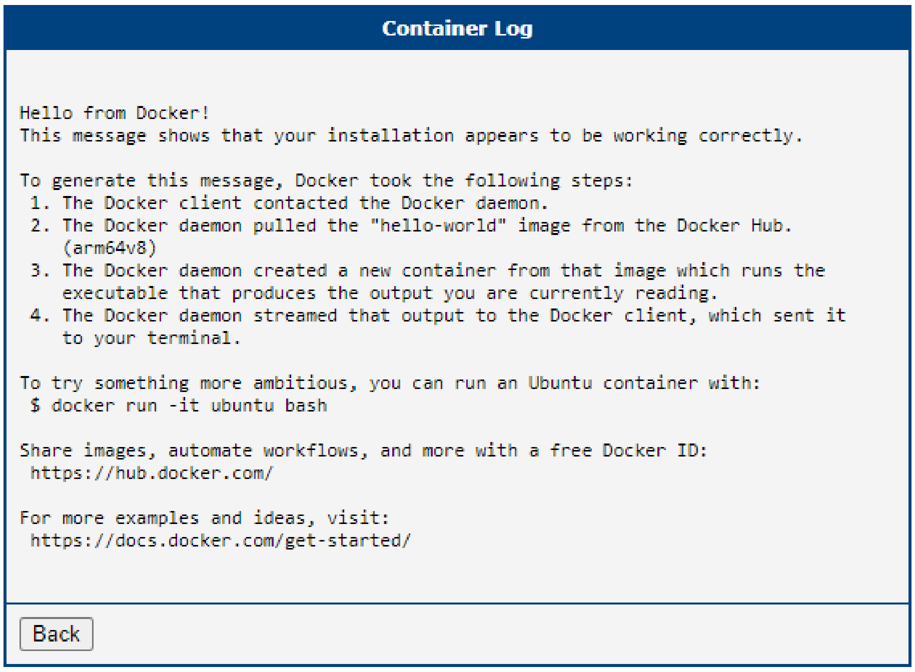
Now lets head to the Container list and the container is really here.



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES	SIZE	ACTIONS
b236f0a7057c	**hello-world:latest**	"/hello"	About a minute ago	Exited (0) About a minute ago		lucid_haslett	0B (virtual 9.14kB)	[Start] [Stop] [Inspect] [Log] [Commit] [Backup] [Remove]

Figure 37: Hello World Container

Now we could read the Container log to see that the Hello World ran successfully.



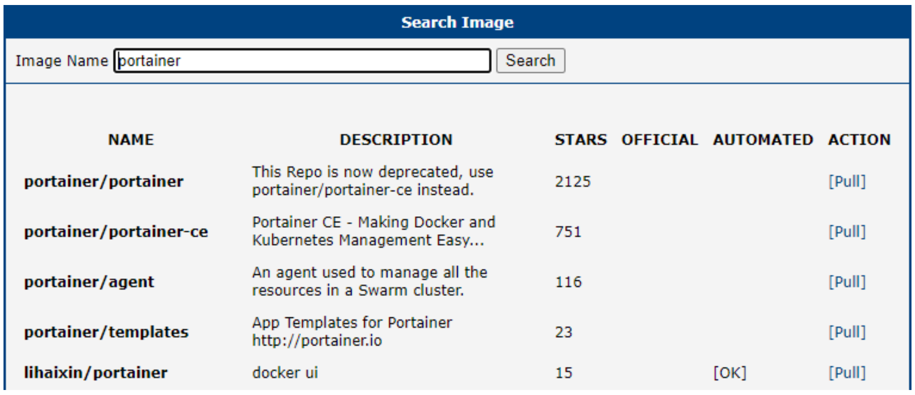
```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figure 38: Hello World Image Run

And thats it!

3.2 Example 2: Portainer

Now lets take a look on something more challenging - the Portainer. We start the same way like in first example and that is searching for the right image.



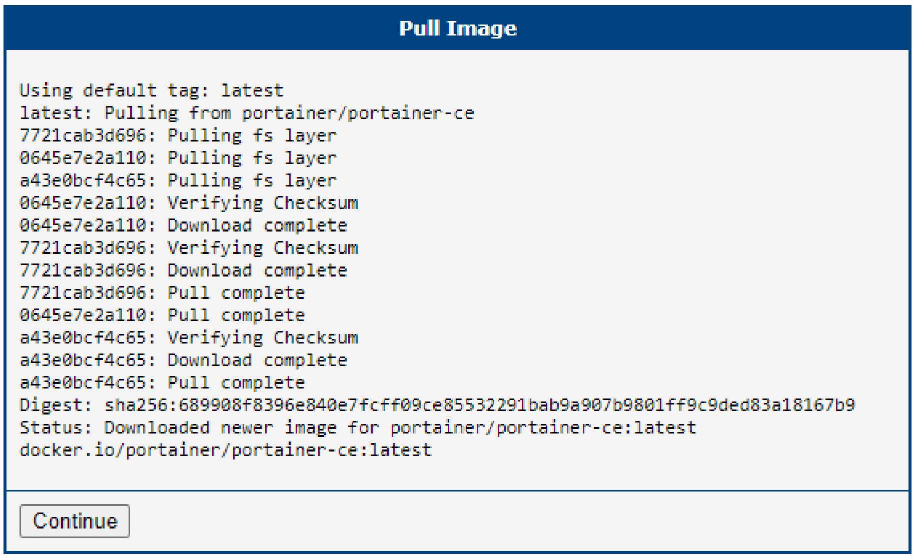
Search Image

Image Name: portainer [Search]

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED	ACTION
portainer/portainer	This Repo is now deprecated, use portainer/portainer-ce instead.	2125			[Pull]
portainer/portainer-ce	Portainer CE - Making Docker and Kubernetes Management Easy...	751			[Pull]
portainer/agent	An agent used to manage all the resources in a Swarm cluster.	116			[Pull]
portainer/templates	App Templates for Portainer http://portainer.io	23			[Pull]
lihaixin/portainer	docker ui	15		[OK]	[Pull]

Figure 39: Portainer Image Search

Ok, results are in and we can see that the first and most starred image has in description *This Repo is now deprecated, use portainer/portainer-ce instead.*, ok, let's go with the second one and pull it.

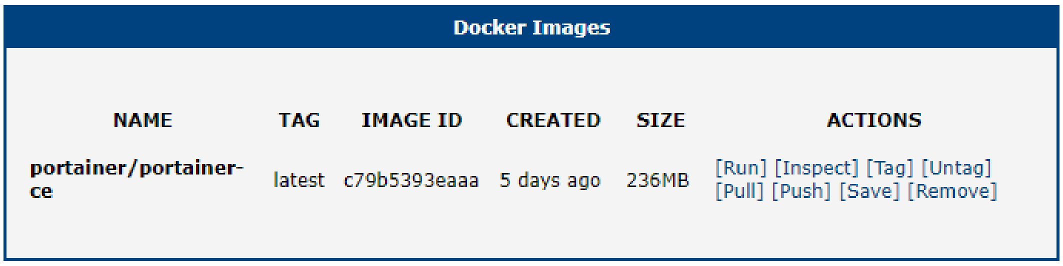


Pull Image

```
Using default tag: latest
latest: Pulling from portainer/portainer-ce
7721cab3d696: Pulling fs layer
0645e7e2a110: Pulling fs layer
a43e0bcf4c65: Pulling fs layer
0645e7e2a110: Verifying Checksum
0645e7e2a110: Download complete
7721cab3d696: Verifying Checksum
7721cab3d696: Download complete
7721cab3d696: Pull complete
0645e7e2a110: Pull complete
a43e0bcf4c65: Verifying Checksum
a43e0bcf4c65: Download complete
a43e0bcf4c65: Pull complete
Digest: sha256:689908f8396e840e7fcff09ce85532291bab9a907b9801ff9c9ded83a18167b9
Status: Downloaded newer image for portainer/portainer-ce:latest
docker.io/portainer/portainer-ce:latest
```

Figure 40: Portainer Image Pull

Image is pulled and ready.



Docker Images

NAME	TAG	IMAGE ID	CREATED	SIZE	ACTIONS
portainer/portainer-ce	latest	c79b5393eaaa	5 days ago	236MB	[Run] [Inspect] [Tag] [Untag] [Pull] [Push] [Save] [Remove]

Figure 41: Portainer Image

But how to run it? This is not as simple as Hello World, we actually need to know how to run it from Portainer developers, we can't just guess it. So lets head to Docker Hub [1] and search for this image here.

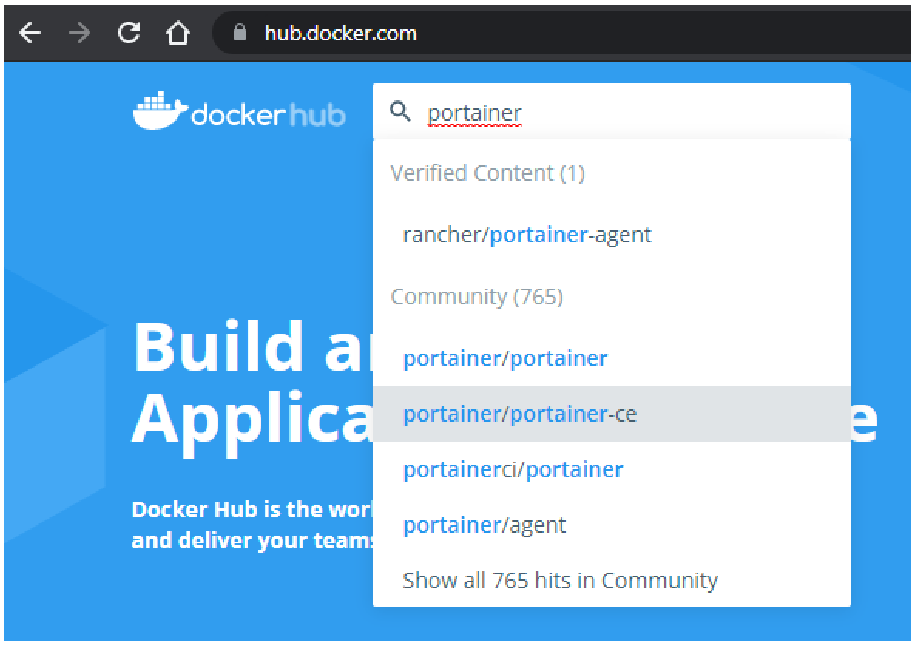
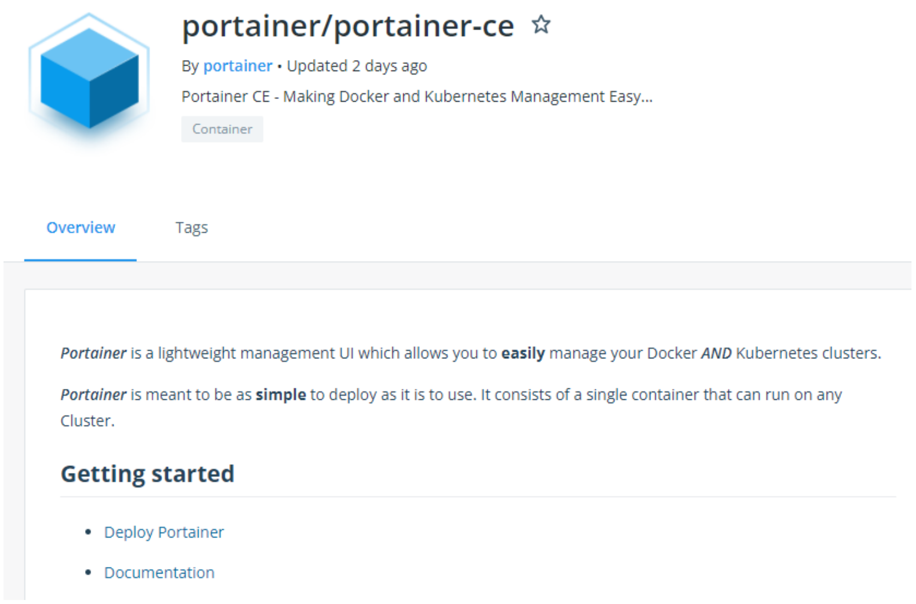


Figure 42: Portainer Search on Docker Hub

[1]https://hub.docker.com

The image is found, but we need example how to run this image and we definitely will find that on the official pages. We can find the link to those on Docker Hub.



portainer/portainer-ce ☆

By portainer • Updated 2 days ago

Portainer CE - Making Docker and Kubernetes Management Easy...

Container

Overview Tags

Portainer is a lightweight management UI which allows you to **easily** manage your Docker ***AND*** Kubernetes clusters.

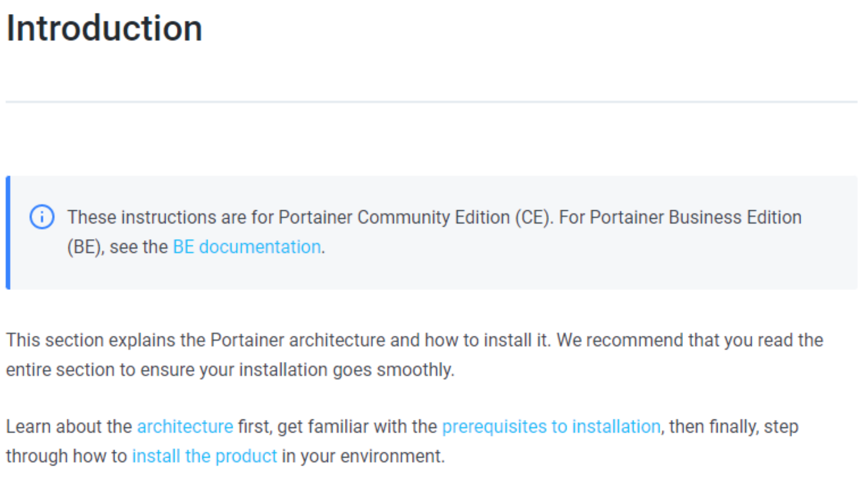
Portainer is meant to be as **simple** to deploy as it is to use. It consists of a single container that can run on any Cluster.

Getting started

- Deploy Portainer
- Documentation

Figure 43: Portainer Search detail on Docker Hub

Lets click on the *Deploy Portainer* item. And now we are redirected to the official pages for portainer `https://docs.portainer.io/v/ce-2.9/start/intro`



Introduction

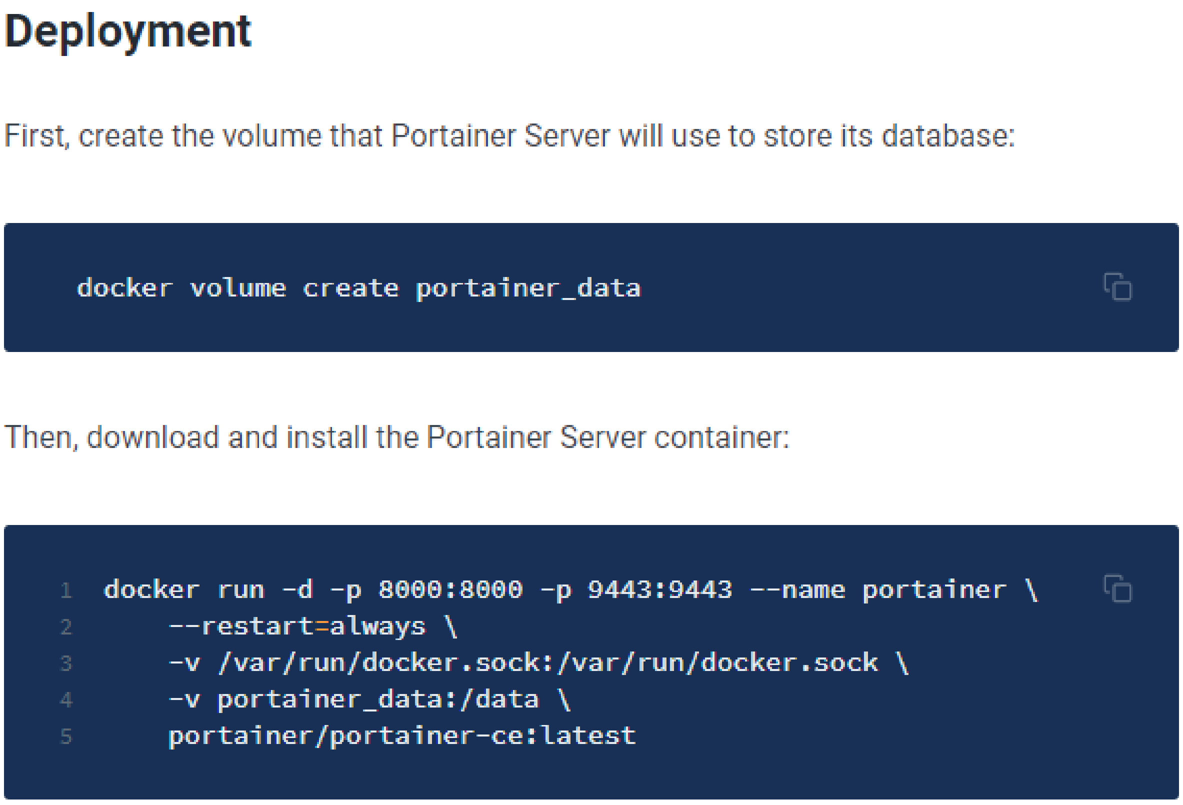
These instructions are for Portainer Community Edition (CE). For Portainer Business Edition (BE), see the BE documentation.

This section explains the Portainer architecture and how to install it. We recommend that you read the entire section to ensure your installation goes smoothly.

Learn about the architecture first, get familiar with the prerequisites to installation, then finally, step through how to install the product in your environment.

Figure 44: Portainer Official Pages

Now we navigate through their site - click on the *Install the product*, then *Set up a new Portainer Server installatiob* then select *Docker Standalone* and finally click on *Install Portainer with Docker on WSL / Docker Desktop* a we are where we need to be - behold, the Docker run example we were looking for:



Deployment

First, create the volume that Portainer Server will use to store its database:

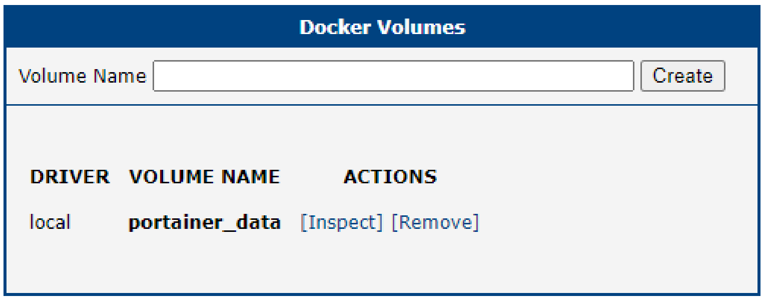
```
docker volume create portainer_data
```

Then, download and install the Portainer Server container:

```
docker run -d -p 8000:8000 -p 9443:9443 --name portainer \
    --restart=always \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v portainer_data:/data \
    portainer/portainer-ce:latest
```

Figure 45: Portainer Deployment

Lets use the knowledge we gained in section 2.3.1 *Transforming Docker Run Command into Run Image action* and create needed volume



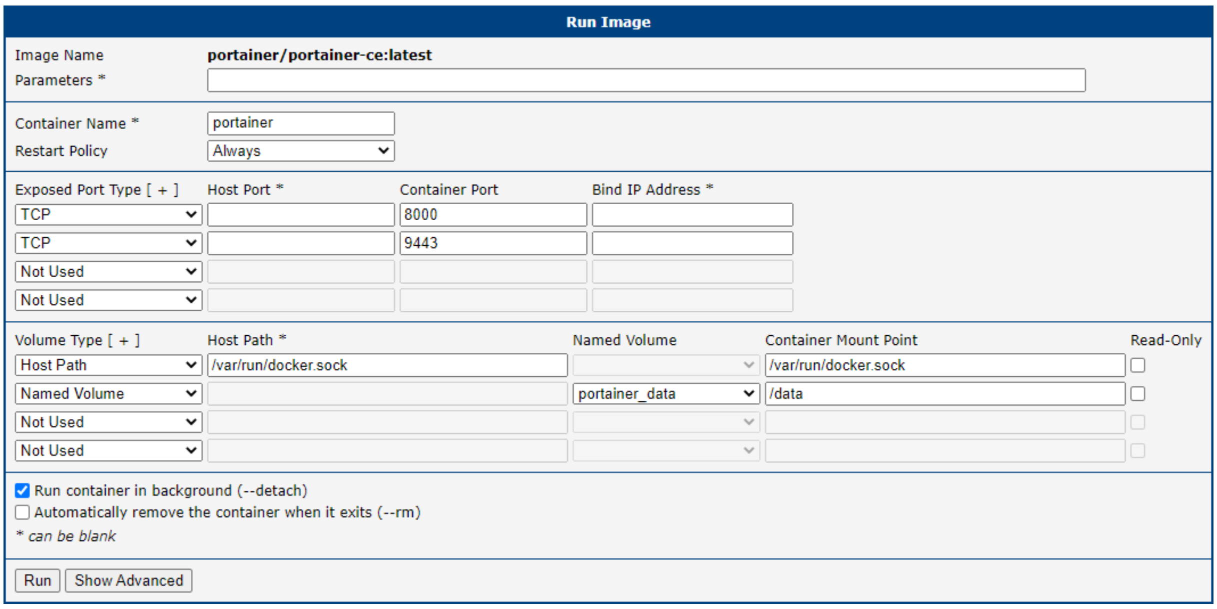
Docker Volumes

Volume Name [ ] Create

DRIVER	VOLUME NAME	ACTIONS
local	**portainer_data**	[Inspect] [Remove]

Figure 46: Portainer Volume

and then fill out the Run Image action for Portainer image like this:



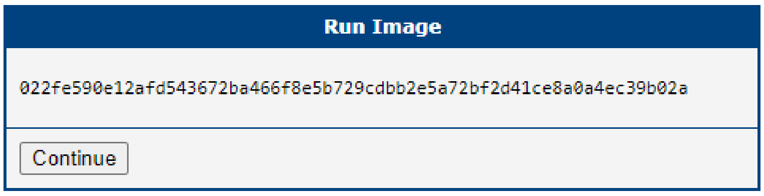
Run Image

Image Name	**portainer/portainer-ce:latest**			
Parameters *				
Container Name *	portainer			
Restart Policy	Always			
Exposed Port Type [+]	Host Port *	Container Port	Bind IP Address *	
TCP		8000		
TCP		9443		
Not Used				
Not Used				
Volume Type [+]	Host Path *	Named Volume	Container Mount Point	Read-Only
Host Path	/var/run/docker.sock		/var/run/docker.sock	☐
Named Volume		portainer_data	/data	☐
Not Used				☐
Not Used				☐

☑ Run container in background (--detach)
☐ Automatically remove the container when it exits (--rm)
** can be blank*

Figure 47: Portainer Image Run

And now we can run the Image.

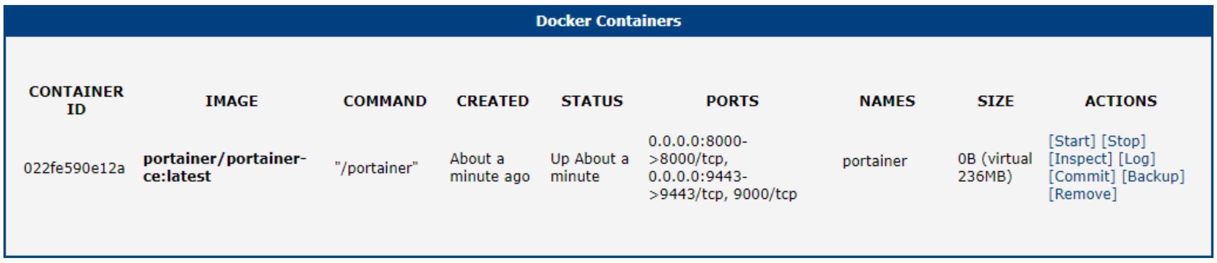


Run Image

```
022fe590e12afd543672ba466f8e5b729cdbb2e5a72bf2d41ce8a0a4ec39b02a
```

Figure 48: Portainer Image Run Done

After Image run is done, we could finally see Portainer Container in container list

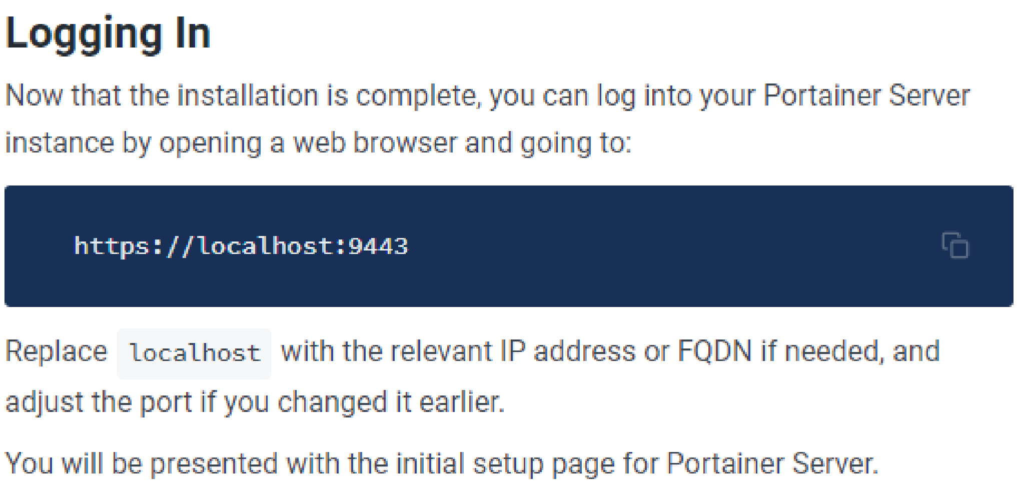


Docker Containers

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES	SIZE	ACTIONS
022fe590e12a	**portainer/portainer-ce:latest**	"/portainer"	About a minute ago	Up About a minute	0.0.0.0:8000->8000/tcp, 0.0.0.0:9443->9443/tcp, 9000/tcp	portainer	0B (virtual 236MB)	[Start] [Stop] [Inspect] [Log] [Commit] [Backup] [Remove]

Figure 49: Portainer Container

Whats next? Lets take a look what oficial guide says



Logging In

Now that the installation is complete, you can log into your Portainer Server instance by opening a web browser and going to:

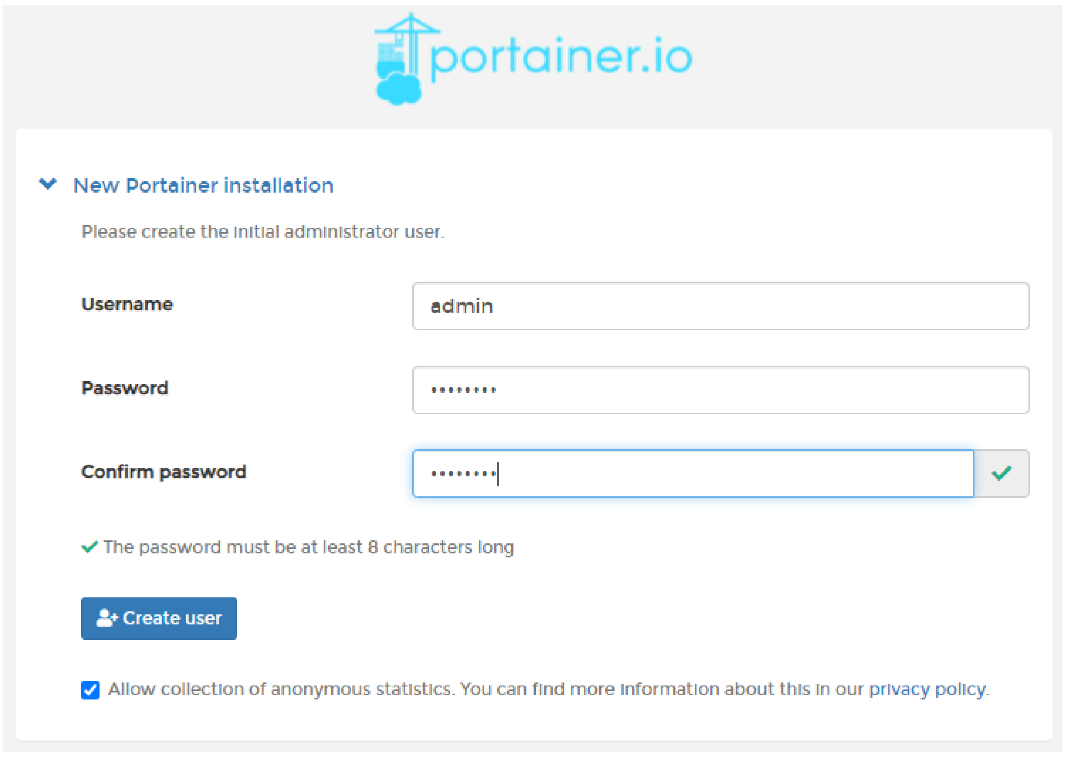
```
https://localhost:9443
```

Replace `localhost` with the relevant IP address or FQDN if needed, and adjust the port if you changed it earlier.

You will be presented with the initial setup page for Portainer Server.

Figure 50: Portainer How to log in

So lets do it and again, they know what they say, so we can see login screen to portaner



portainer.io

New Portainer installation

Please create the initial administrator user.

Username	admin
Password	•••••••••
Confirm password	•••••••••

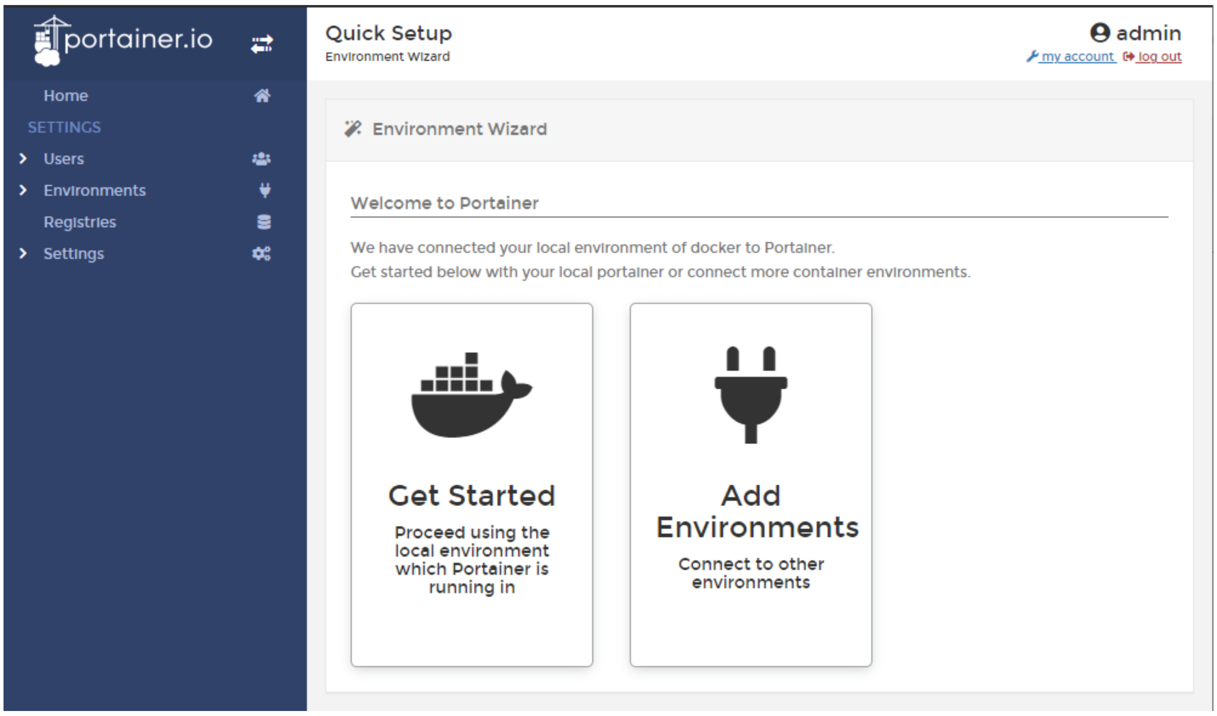
✓ The password must be at least 8 characters long

Create user

[✓] Allow collection of anonymous statistics. You can find more information about this in our privacy policy.

Figure 51: Portainer Logging In

and after creating an admin user account we can finally see working Portainer



We have connected your local environment of docker to Portainer.
Get started below with your local portainer or connect more container environments.

Figure 52: Portainer

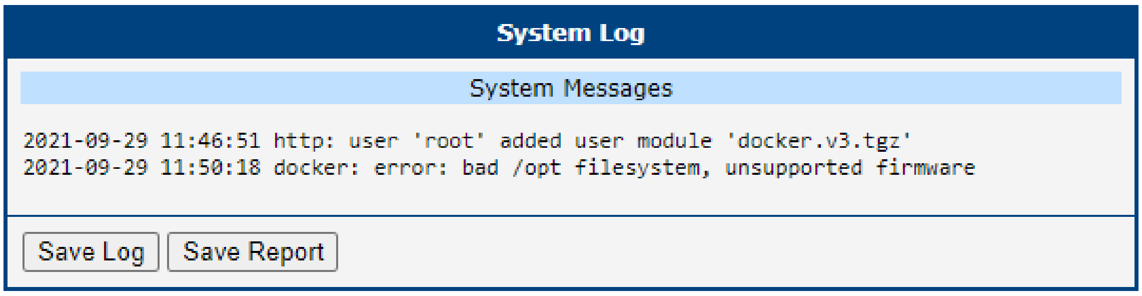
4. Troubleshooting

Now there are 2 kinds of error we could get when working with Docker Router App. You can find those errors in routers *System Log*.

4.1 Unsupported firmware

This error typically arises on for example v3 routers, which doesn't have eMMC memory. Simply said, the router isn't capable to run Docker Router App.

eMMC memory is vital for running Docker Router App. SD Card alone is not sufficient.

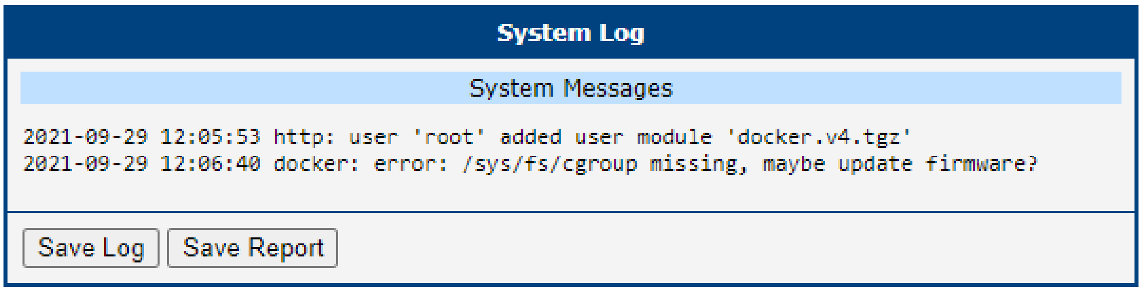


```
2021-09-29 11:46:51 http: user 'root' added user module 'docker.v3.tgz'
2021-09-29 11:50:18 docker: error: bad /opt filesystem, unsupported firmware
```

Figure 53: Unsupported firmware

4.2 Update firmware

This error gives us a hope compared to the previous one - it means, that the router has older firmware, than it is required to run. Firmware update to at least version 6.3.2 should do the trick and Docker Router App should work just fine after.



```
2021-09-29 12:05:53 http: user 'root' added user module 'docker.v4.tgz'
2021-09-29 12:06:40 docker: error: /sys/fs/cgroup missing, maybe update firmware?
```

Figure 54: Update firmware

5. Related Documents

[1] Docker Docs: **https://docs.docker.com/**
[2] Docker Hub: **https://hub.docker.com/**

You can obtain product-related documents on *Engineering Portal* at *icr.advantech.cz* address.

To get your router's *Quick Start Guide*, *User Manual*, *Configuration Manual*, or *Firmware* go to the *Router Models* page, find the required model, and switch to the *Manuals* or *Firmware* tab, respectively.

The *Router Apps* installation packages and manuals are available on the *Router Apps* page.

For the *Development Documents*, go to the *DevZone* page.