Rust server on Linux using Docker

16 MAY 2016

Introduction

Ever wanted to run a Rust Linux server, but figured it's more of a hassle than it's worth?

It's about to become much easier, as long as you follow along and consume as much of the information I'm about to bestow upon you.

This guide assumes general knowledge of Linux, such as basic commands and security (firewall, ssh).

The only reason this guide is quite lengthy, is because it's also introducing Docker and it's basic usage, but if you're already familiar with it, you can feel free to skip a lot of this.

What's Docker?

Docker is a system for running services, such as apps, in tiny containers.

When people hear the word *containers*, they usually think about virtual machines, but this is not the case with Docker.

Containers are lightweight, fast and secure, well, containers.

They run identically on all supported systems, which makes them ideal for running single services or apps (or a Rust server, in this case).

Docker containers, by default, don't persist any data when they're restarted or destroyed, but they do support something called *volumes*, which is a way of mounting directories on the host system, inside the container itself.

With the help of volumes, it's exceptionally easy to move Docker containers between servers, restore from backups or recover from hardware failures (as long as you have backups!).

We won't go in to much more depth about Docker here, but if you'd like to read more about it, you can do so here.

Pre-setup

NEW: You can skip the pre-setup and setting up Docker if you

just want to test this out.
You can do so by clicking this button:

Install on DigitalOcean

The two things you need are: a dedicated server with enough memory (preferably 4GB or more), and a somewhat recent operating system, such as Ubuntu 14/15/16, or Debian 7/8.

Technically the OS doesn't matter, as long as it can run Docker. The server itself is entirely independent of the operating system.

NOTE: If you're running Debian, please see this article for updating your kernel.

For the rest of this guide, we'll assume you're using Ubuntu 14.04.

Setup

Make sure curl is installed:

```
sudo apt-get install curl -y
```

Install Docker:

```
curl -fsSL https://get.docker.com/ | sh
```

That's it for the installation.

There's still a bit of configuration to do, but we'll go through that in an upcoming section.

First boot

At this point we're ready to start the Rust server for the first time. Finally, right?

Let's run our server with this simple command:

```
docker run --name rust-server didstopia/rust-server
```

What this does, is it first downloads the latest *rust-server* image from Docker Hub (a public repository of Docker images), then it continues to start a new *rust-server* container, giving it a name of *rust-server*.

IMPORTANT: The image name (`didstopia/rust-server`) needs to be the last argument.
Arguments should added before the image name, otherwise they'll be ignored.

The initial startup might take a while, since our image will download or update to the latest version of *steamcmd* and the official *Rust server*.
Once that's done, you should see the server starting up.

Now, to shutdown the server, you can either press *CTRL-C*, which will initiate the shutdown procedure for *rust-server*, or you can open another terminal window and type in the following command to stop the server:

```
docker stop rust-server
```

Docker commands are fairly straightforward and easy to guess, but you can look them all up by doing the following:

```
docker help
```

Firewall

Next we'll need to open a couple of ports to a) get the server communicating with Steam, and b) to get remote administration (RCON) working from outside the Docker container.

Using your favorite firewall utility (iptables, ufw etc.), open the following ports:

```
28015 (TCP & UDP)
28016 (TCP)
```

Here's an example of how we would set up *ufw* on a fresh

install of Ubuntu:

```
sudo apt-get install ufw -y
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow ssh
sudo ufw allow http
sudo ufw allow https
sudo ufw allow 28015
sudo ufw allow 28015/udp
sudo ufw allow 28016
sudo ufw allow 8080
sudo ufw enable
```

Configuration

We're nearly there! All that's left is to configure the server to your preferences, and to have the server automatically startup with the host system (in case the host system restarts or crashes).

First, create a file that will hold our server configuration (environment variables). For example, at `/rust.env`.

Inside the file, put the following and customize where necessary (don't use double quotes!):

```
RUST_SERVER_STARTUP_ARGUMENTS=-batchmode -load -logfile
/dev/stdout +server.secure 1
RUST_SERVER_IDENTITY=my_awesome_server
RUST_SERVER_SEED=12345
RUST_SERVER_NAME=My Awesome Server
RUST_SERVER_DESCRIPTION=This is my awesome server
RUST_RCON_PASSWORD=SuperSecurePassword
```

If you're already familiar with Rust servers, you can use the first variable (`RUST_SERVER_STARTUP_ARGUMENTS`) to modify things like save intervals, maximum amount of players and so on.

The rest of the variables should be self explanatory.

There are a lot more variables that you can set to customize the server to your liking. A list of all the variables (and much more) can be found here in the container repository.

Firing it up

We can now use Docker to load these variables when starting the container, keeping the configuration in one place, making it easy to modify later on if necessary.

Run the following command, making sure that you've configured everything correctly so far:

```
docker run --name rust-server -d -p 28015:28015 -p
28015:28015/udp -p 28016:28016 -p 8080:8080 -v
/rust:/steamcmd/rust --env-file /rust.env
didstopia/rust-server
```

Parts of this should look familiar at this point, as all they do is fetch the latest version of the server and assigns a name to it.
A keen eye will notice that we've added the `-d` flag, which will detach (or daemonize) the process, so this time you're not getting regular console output from the server.

The `-p` flag will actually bind our local (ie. host machine) ports to the container ports, which is what enables us to connect to the server from outside our network. If you look closely, you'll see port *8080*, which is actually used together with *webrcon* to provide a built-in web-server, which you can then use to access the server remotely using any browser. You can disable this by simply removing the port binding part for port *8080*.

The `-v` flag is where the real action happens, as the *-v* flag translates to *volume*, meaning it will mount a volume on the local filesystem inside the container. The first part is the host path (*/rust* in this case), and the second part is the container path (*/steamcmd/rust*, which is always the same).

Since Docker containers aren't persistent (they can't be permanently modified, changes aren't saved and so on), using a volume will now store both the Rust installation data, as

well as the server data (level, users, blueprints, logfiles). Storing the Rust installation data on the host system means that when the container is restarted, it won't have to download the entire server again, but it can still update it if necessary.

As you might've guessed, the `--env-file` flag will simply load our variables from the file we set up before.

At this point you might be wondering how we would access the server console. We can do so with the following command:

```
docker logs -f rust-server
```

If we wanted to stop the server, we would run the following command:

```
docker stop rust-server
```

To update to the latest image, pulling it from Docker Hub if necessary, we would run the following:

```
docker pull didstopia/rust-server
```

Finally, if we got an error saying that a server with the name *rust-server* already exists, we could forcibly remove the container by running:

```
docker rm -f rust-server
```

We should now have a fully functioning, fully configured server setup. All that's left now is to make sure it's automatically restarted in case it crashes, or the host system is restarted.

Automatic startup

Ubuntu 14.04 uses a system called *upstart*, which we'll use in this example to allow our Rust server to start automatically.

Create a new file in `/etc/init/rust-server.conf` and paste the following in it:

```
description "Rust server container"
author "Dids"
start on filesystem and started docker
stop on runlevel [!2345]
respawn
script
    /usr/bin/docker rm -f rust-server 2>/dev/null ||
true
    /usr/bin/docker pull didstopia/rust-server
    /usr/bin/docker run -p 28015:28015 -p
28015:28015/udp -p 28016:28016 -p 8080:8080 --name
rust-server --env-file /rust.env -v
```

```
/rust:/steamcmd/rust --rm didstopia/rust-server
end script
pre-stop script
    /usr/bin/docker stop rust-server
    /usr/bin/docker rm -f rust-server 2>/dev/null ||
true
end script
```

Note that this particular startup script uses the `--rm` flag, which will destroy the container once it's stopped. This allows for clean starting/stopping of the server, as the container name will always be available this way. We're also forcibly destroying the container both when stopping and starting the server, just in case anything goes wrong.

For *upstart*, the log file exists at `/var/log/upstart/rust-server.log`, and you should check it for errors if the server doesn't seem to be starting. This is usually just a naming conflict, which can be resolved by destroying the container using `docker rm -f rust-server` and restarting.

You should now be able to start and stop the server, as well as check the status of it, by running any of the following commands:

```
service rust-server start
service rust-server stop
service rust-server status
```

Docker provides a nice list of examples on how to do this on

other process managers.

Administering the server

Since the Docker image we're using is running a built-in web-server (if you're using webrcon, that is), we can access it at *http://server-ip-or-hostname:8080*. The password is the one we setup in the *rust.env* file.

You can also use remote RCON tools, such as RustAdmin.

Rust's developers tend to release updates every Thursday, but because of how magical Docker is, you can simply run `service rust-server restart` and the container will gracefully shutdown, restart and check for updates, installing them as necessary.

From now on, maintaining a Rust server should be a breeze. :)

Sending commands to the server

Sending commands to a linux version of a Rust server is a tricky subject, because this can only be done using RCON.

I've managed to create a small application inside the Docker

image, called `rcon`, which you can use to both send and receive data from the server.

To use it, simply execute a Docker command like `docker exec rust-server rcon fps`, with *rust-server* being your container name.

As I mentioned, it also tries it's best to return the response from the server, so in this case it would return `263 FPS` or something similar.

You can hook this up to your bash scripts, or call it manually, but as long as your server is setup with *webrcon*, it should "just work".

Tips and tricks

Fixing memory issues

Rust (or more accurately Unity) is known to crash when using more than 16GB memory. Docker can solve this issue easily, in case your system has more than 16GB available.

All you need to do, is add `-m 16g` to the startup arguments for the `docker run` command. This will limit the maximum amount of memory the single container is allowed to consume.

Remember, all Docker arguments need to be added *before* the

image name (`didstopia/rust-server` in this case), otherwise they'll get ignored!

I'll keep updating this section with new tips and tricks as they come up.

Setting the branch

You can actually use the environment variables `RUST_BRANCH` to set your server to use a specific branch.

This alters the installation parameters, so for example to use the prerelease branch, you would use `RUST_BRANCH="-beta prerelease"`, note the `-beta` there for this particular branch.

Enabling fully automatic updates

There is now a way for your server to periodically check for updates, notify players of the update, restart in 60 seconds, install the update and come back up, all completely automatically.

You can enable this functionality by using `RUST_UPDATE_CHECKING=1`, which enables checking for updates.

Additionally, you can set a specific update branch (default is called "public") by using `RUST_UPDATE_BRANCH="staging"`. Do note that you should NOT specify any arguments, such as "-beta", as this differs slightly from setting the branch above.

Final notes

This guide is a living thing, as is the server image we used here, so both of them will change and be updated with time.

I'll try to keep this guide up to date when it's necessary, and add small update notes to what has changed, to avoid confusion.

If you have any feedback on the guide, or want to ask something, get help with your Docker setup or anything of the sorts, please feel free to contact me on the *Rust Server Owners* Slack Community (I'm *dids* there), or on Twitter (@Dids).

YOU MIGHT ENJOY

New website

So, the new website is finally here! We'll try our best to communicate things such as the server rules,...