Running Icinga2 on the Raspberry Pi 3

by Benedict Reuschling

In my day job, I am responsible for a 40-node, big data cluster used in education and research at the Darmstadt University of Applied Sciences, Darmstadt, Germany. One of the tasks of a sysadmin is checking whether the systems you are responsible for are healthy and available on the network.

This is typically done using periodic checks from a monitoring instance that collects data from remote machines and uses the metrics to trigger alarms when certain thresholds are reached. We monitor the cluster nodes from the central file server, which can spare a few cycles when it is not serving files to clients (ZFS-based NFSv4 exports for home directories and other shared data).

Another sysadmin job is thinking about the "what-could-go-wrong" scenario. That scenario might never happen, but when and if it does, you are glad to have already implemented a "better-safe-than-sorry" scenario. The situation here is "who is watching the watcher?"—what will happen if the central monitoring instance goes down? It could very well be that the outage is not recognized for a while, which would also not inform us of any other failures occurring in the meantime. When that happens, we not only have to deal with one but two problems: the unavailable monitoring system and the problem that the monitoring should have told us about. We quickly came to the conclusion that a second system needs to monitor the main monitoring server and alert us if that system disappears from the network without running other checks itself (which is optional). Ideally, this should be done from a separate network. If both systems were on the same subnet, we would not get any information from our two monitoring systems if that whole network became unreachable. Of course, a second system just for the sake of monitoring is expensive (both the initial investment and running costs like electricity), so we needed to find a low-cost solution.

The Raspberry Pi is the perfect tool for long-term, 24/7 monitoring. With the small form factor, low power use, and a powerful enough CPU for periodic checks of remote systems, it can solve a variety of tasks. With a small initial cost (the board itself, a compact flash card, and some cables) it is easy to get started, although some tasks like processing package updates may take a bit longer than on an amd64 system. Compiling things locally on the RPi3 takes much longer, hence we rely on packages for our third-party software needs.

First, we download the FreeBSD-12.0-RELEASE-arm64-aarch64-RPi3.img.xz from the SD Card Images section in the download area of freebsd.org. After un-xz-ing the archive, it is time to insert the SD card (we use one with 64 GB; smaller ones also work perfectly fine) into the reader on the machine that was downloading the image. The following command line will copy the image to the SD card (change device names to match; take care not to overwrite a different partition):

```
# dd if=/FreeBSD-12.0-RELEASE-arm64-aarch64-RPI3.img of=/dev/disk2\
bs=1m conv=sync
```

Once dd(1) has finished writing the image, unmount it and insert it into the RPi3's SD card slot. Connect all the other peripherals like monitor, keyboard, network cable, and power. The Pi should start booting once the power is connected. The first boot will take a bit of time to expand the image according to the size of the SD card. Be patient—this will only be done once and subsequent boots will run faster. There are two users that you can log in with by default: root and freebsd, with the same password as the username, respectively. This is the first thing you should change after logging into the Pi for the first time. Then it is time to do some initial system configuration based on your needs and environment.

You may have heard that writing to the SD card constantly will reduce the lifetime as there are only so many rewrite cycles that the cells that make up the storage media can take. To avoid an early death, we attached a cheap 32-GB SSD to it and create a ZFS pool on it. Running ZFS on an embedded device is generally discouraged and UFS is a perfectly fine filesystem for those kinds of devices. Surprisingly enough, we've been running this setup for months now without any problems. Granted, the zpool is not redundant and the writes on the pool are not as heavy as on our big data file server. Your mileage may vary, and additional tuning may be required. Here is the ZFS list output to show which datasets were created:

```
ssd                22.4G       6.42G       22.5M        /ssd
ssd/swap           1.03G       6.45G       1.00G        -
ssd/tmp            41.6M       6.42G       41.6M        /tmp
ssd/usr            1.44G       6.42G       1.44G        /usr
ssd/usr/ports      1.08G       6.42G       1.08G        /usr/ports
ssd/var            10.8G       6.42G       10.5G        /var
ssd/var/cache       860K       6.42G        860K        /var/cache
ssd/var/db          252M       6.42G        252M        /var/db
ssd/var/log        34.8M       6.42G       34.8M        /var/log
```

We are still booting from UFS, but the main, write-heavy filesystems like `/usr` and `/var` reside on the SSD pool. This is as simple as creating the datasets (activating `lz4` compression in the process) and copying the directory contents from UFS *before* setting a mountpoint. Take extra precautions when mounting `/usr` as this will mount the ZFS dataset /usr over your current system, which will likely cause some interruptions. To prevent that and let the Pi boot properly, I have the following lines in my `/etc/rc.local`:

```
/sbin/zpool import -f ssd
/sbin/zfs mount -a
swapon /dev/zvol/ssd/swap
/usr/sbin/ntpdate -b de.pool.ntp.org
```

As you can see, I'm also running the swap from the SSD and sync my time upon reboot from a timeserver near me. For the sake of completeness, these are my entries in `/boot/loader.conf`:

```
# Configure USB OTG; see usb_template(4).
hw.usb.template=3
#umodem_load="YES"
# Multiple console (serial+efi gop) enabled.
boot_multicons="YES"
boot_serial="YES"
# Disable the beastie menu and color
beastie_disable="YES"
loader_color="NO"
geom_label_load="YES"
verbose_loading="YES"
autoboot_delay="2"
zfs_load="YES"
```

The `/etc/rc.conf` is fairly minimalistic and only has a few changes in it:

```
hostname="rpi3.mydomain.local"
ifconfig_DEFAULT="DHCP"
sshd_enable="YES"
sendmail_enable="NONE"
```

Code continues

Continued

```
sendmail_submit_enable="NO"
sendmail_outbound_enable="NO"
sendmail_msp_queue_enable="NO"
growfs_enable="YES"
keymap="de.kbd"
hostid_enable="YES"
fsck_y_enable="NO"
background_fsck="no"
mixer_enable="no"
```

Reboot to see that the Pi will come back up to the login prompt without any errors and that ssh logins are possible. This is an important step to running the Pi headless without a monitor attached to it.

Since we want the RPi3 to send us notifications if our monitoring system detects anything unusual, we configure SSMTP. This provides us with a simple, send-only mail delivery system. For this article, we use Google mail to deliver email. Be aware of the sensitive nature of the emails you are sending, so Gmail might not be the best solution for all environments. Replace it with your own mail delivery that you know and trust to ship messages about any system problems you might encounter.

First, we need to install the ssmtp package:

```
# pkg install ssmtp
```

The configuration file is located in `/usr/local/etc/ssmtp/` and we need to modify two files here (create them if they do not exist yet or copy the .sample files of the same name and modify them): `ssmtp.conf` and `revaliases`. In `ssmtp.conf`, we need only to have the following lines present:

```
root=yourusername@gmail.com
mailhub=smtp.gmail.com:587
AuthUser=yourusername@gmail.com
AuthPass=thepasswordforthegmailaccount
UseSTARTTLS=YES
rewriteDomain=gmail.com
hostname=rpi3.mydomain.local
FromLineOverride=YES
UseTLS=YES
```

The `revaliases` file has just two lines in it:

```
root:yourusername@gmail.com:smtp.gmail.com:587
otheruser:yourotherusername@gmail.com:smtp.gmail.com:587
```

This will configure the local mapping to determine which local user should get mail delivered. At least root and an unprivileged account should be configured here. Periodic system mail is sent to root by default and the email address after defines where that should be delivered. The same is true for personal mail to that unprivileged user in the line below it. The `mailer.conf` in `/etc` should have the following lines in it (likely done by the port/package):

```
#sendmail      /usr/libexec/sendmail/sendmail
#send-mail     /usr/libexec/sendmail/sendmail
#mailq         /usr/libexec/sendmail/sendmail
#newaliases    /usr/libexec/sendmail/sendmail
#hoststat      /usr/libexec/sendmail/sendmail
#purgestat     /usr/libexec/sendmail/sendmail
sendmail       /usr/local/sbin/ssmtp
send-mail      /usr/local/sbin/ssmtp
mailq          /usr/local/sbin/ssmtp
newaliases     /usr/local/sbin/ssmtp
hoststat       /usr/bin/true
purgestat      /usr/bin/true
```

Make sure that ssmtp owns the directory var/spool/clientmqueue using chown:

```
# chown -R smmsp:smmsp /var/spool/clientmqueue
# chmod -R 0775 /var/spool/clientmqueue
```

Test the mail setup by sending yourself a test message as the root user and the unprivileged user. You may need to allow unsecure applications in your Google account settings to enable mail transport. When mail is arriving, the configuration for Icinga2, the monitoring system we are using, can commence.

Packages for Icinga2 need to be installed first, including PostgreSQL and the necessary PHP libraries for nginx, which we'll use as the webserver to run the Icingaweb2 front-end.

```
# pkg install icinga2 icingaweb2 postgresql95-server nginx
ImageMagick6-nox11 php72-pecl-imagick
```

We enable the icinga service and postgresql to run upon reboot. Do not start these services just yet as they are still unconfigured:

```
# sysrc icinga2_enable=yes
# sysrc postgresql_enable=yes
```

Configuring PostgreSQL as a backend database is the next task. Since we are running on the SSD, the database can benefit from some tuning for ZFS. Note that this is not strictly needed and postgres will run just fine on UFS.

A dataset called pgdata is created, some properties are set, and ownership of the mountpoint is changed to the pgsql user:

```
# zfs create -o mountpoint=/usr/local/pgsql/data ssd/pgdata
# zfs set recordsize=8k ssd/pgdata
# zfs set logbias=throughput ssd/pgdata
```

Code continues

Continued

```
# zfs set redundant_metadata=most ssd/pgdata
# zfs set primarycache=metadata ssd/pgdata
# chown pgsql:pgsql /usr/local/pgsql/data
```

The next steps are done as the `pgsql` user, either by switching to it using `su(1)` or (when configured) via `sudo -u pgsql`.

```
pgsql$ cd
pgsql$ initdb --no-locale --encoding=utf-8 --lc-collate=C -E UTF8 -D ./data
psql$ pg_ctl start -D ./data
```

This will initialize the database cluster and start the database using pg_ctl. UTF-8 is set as the encoding for the database as Icinga requires it. Once the postgresql service is running, a user and database called icinga is created:

```
pgsql$ createuser -dPrs icinga
pgsql$ createdb -O icinga -E UTF8 icinga
```

To allow this user to access the database, the following lines need to be added to `/usr/local/pgsql/data/pg_hba.conf`:

```
local    icinga     icinga                        md5
host     icinga     icinga     127.0.0.1/32       md5
host     icinga     icinga     ::1/128            md5
```

Icinga is using its own database schema to store tables for hosts, services, users, notifications, etc. These need to exist before Icinga can start monitoring. An SQL script is provided to set up the database using psql:

```
pgsql$ psql -U icinga -d icinga < /usr/local/share/icinga2-ido-pgsql/
schema/pgsql.sql
```

Subsequent updates of the Icinga package/port will require updates to the database schema as well. These updates are located under `/usr/local/share/icinga2-ido-pgsql/schema/upgrades` and must be applied in order. Check the update instructions on the Icinga website for details. After the schema has been set up in the database, log out of the `pgsql` user.

Like nginx, Icinga has a plugin-like system to activate certain features. Our setup requires the activation of at least the ido-pgsql (since we are using that as the backing database) and command (to, i.e., schedule service checks through Icingaweb2) modules. The api feature will replace the command module in the future, but for now, we activate it to be on the safe side.

```
# icinga2 feature enable ido-pgsql api command
```

To tell Icinga about the postgresql database details, the `ido-pgsql.conf` file located in `/usr/local/etc/icinga2/features-enabled/must` contain the following lines:

```
object IdoPgsqlConnection "ido-pgsql" {
  user = "icinga"
  password = "thepasswordyousetwhencreatingthedatabaseabove"
  host = "localhost"
  database = "icinga"
}
```

Run the command `icinga api setup` and modify `/usr/local/etc/icinga2/conf.d/api-users.conf` to set up a separate API user for icingaweb2:

```
object ApiUser "icingaweb2" {
  password = "somerandompasswordstrongerthanthisone"
  permissions = [ "status/query", "actions/*", "objects/modify/*",
"objects/query/*" ]
}
```

The permissions allow certain actions to be performed by that user. Other API users are possible for different actions that may have fewer privileges. This is beyond the scope of this article, but details can be found in the Icinga2 documentation (https://icinga.com/docs/).

Time to restart our database and bring up icinga2:

```
# service postgresql restart
# service icinga2 start
```

Configure nginx as the webserver for icingaweb2. The following steps are required for that:

```
# sysrc php_fpm_enable=yes
# sysrc nginx_enable=yes
# sed -i '' "s/listen\ =\ 127.0.0.1:9000/listen\ =\ \/var\/run\/php5-
fpm.sock/" /usr/local/etc/php-fpm.d/www.conf
# sed -i '' "s/;listen.owner/listen.owner/" /usr/local/etc/php-
fpm.d/www.conf
# sed -i '' "s/;listen.group/listen.group/" /usr/local/etc/php-
fpm.d/www.conf
# sed -i '' "s/;listen.mode/listen.mode/" /usr/local/etc/php-fpm.d/www.conf
```

Add the following section to nginx configuration file located in `/usr/local/etc/nginx/nginx.conf` before the `location / { ... }` part:

```
location ~ ^/icingaweb2/index\.php(.*)$ {
  fastcgi_pass unix:/var/run/php5-fpm.sock;
  fastcgi_index index.php;
  include fastcgi_params;
  fastcgi_param SCRIPT_FILENAME /usr/local/www/icingaweb2/public/index.php;
```

Code continues

Continued

```
  fastcgi_param ICINGAWEB_CONFIGDIR /usr/local/etc/icingaweb2;
  fastcgi_param REMOTE_USER $remote_user;
}

location ~ ^/icingaweb2(.+)? {
  alias /usr/local/www/icingaweb2/public;
  index index.php;
  try_files $1 $uri $uri/ /icingaweb2/index.php$is_args$args;
}
```

Note that setting up SSL is beyond the scope of this article but is fairly straightforward with services such a letsencrypt.

Create a **php.ini** from the provided php.ini-production file that came with the package:

```
# cp /usr/local/etc/php.ini-production /usr/local/etc/php.ini
```

Edit the file and set the **date.timezone** line to the time zone of the server. The webserver and php-fpm can start now:

```
# service nginx start
# service php-fpm start
```

Take a browser and point it to the IP/DNS name of your RPi3 and add **/icingaweb2/setup** at the end. The icingaweb setup is protected from drive-by attackers by a token that needs to be created from the command line like this:

```
# icingacli setup token create —config=/usr/local/etc/icingaweb2
# chown -R www:www /usr/local/etc/icingaweb2
```

Follow the setup steps and provide the postgresql database information (username, password) when prompted. Lars Engels, who maintains the icinga2 port has a blog post that will walk you through the setup (http://lme.postach.io/post/installing-icinga-web-2-with-apache-2-4-icinga-2-and-mysql-on-freebsd).

Configure hosts and services to monitor:

Icinga checks are divided into active and passive checks. The active checks are executed on the monitored hosts themselves, and the results are sent back to the icinga server for further processing. Passive checks are done by the icinga server from outside of the hosts like ping. We'll only cover passive checks; however, active checks are also working well enough on the Raspberry Pi. The configuration file for the hosts to monitor is located under **/usr/local/etc/icinga2/conf.d/hosts.conf** and my default already contains the icinga server itself. Simply add a new host using the following template:

```
object Host "myhost" {
  import "generic-host"
  address = "ip.address.of.myhost"
```

Code continues

Continued

```
    vars.notification["mail"] = {
      groups = [ "icingaadmins" ]
    }
}
```

Before restarting icinga, make sure to let it check the configuration file before restarting the service:

```
# icinga2 daemon -C && service icinga2 restart
```

The host should now appear in the icingaweb2 GUI. The Raspberry Pi will continue to contact the host periodically and send alerts if a host or service is changing its state from UP to DOWN or vice versa.

Icinga offers a lot of functionality. Depending on the number of hosts and services to check, the RPi3 might need some time to process those. For a small number of hosts/services, this should not become problematic and is a cost-effective solution with a lot of customization options. •

Benedict Reuschling joined the FreeBSD Project in 2009. After receiving his full documentation commit bit in 2010, he actively began mentoring other people to become FreeBSD committers. He joined the FreeBSD Foundation in 2015, where he is currently serving as vice president. Benedict has a Master of Science degree in Computer Science and is teaching a UNIX for software developers class at the Darmstadt University of Applied Sciences, Darmstadt, Germany. Together with Allan Jude, he is host of the weekly BSDNow.tv (http://BSDNow.tv) podcast.

Help Create the Future.
Join the FreeBSD Project!

FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system.

Not only is FreeBSD easy to install, but it runs a huge number of applications, offers powerful solutions, and cutting edge features. The best part? It's FREE of charge and comes with full source code.

Did you know that working with a mature, open source project is an excellent way to gain new skills, network with other professionals, and differentiate yourself in a competitive job market? Don't miss this opportunity to work with a diverse and committed community bringing about a better world powered by FreeBSD.

The FreeBSD Community is proudly supported by

FreeBSD FOUNDATION

FreeBSD®

The FreeBSD Project is looking for

- Programmers
- Testers
- Researchers
- Tech writers
- Anyone who wants to get involved

Find out more by

Checking out our website
freebsd.org/projects/newbies.html

Downloading the Software
freebsd.org/where.html

We're a welcoming community looking for people like you to help continue developing this robust operating system. Join us!

Already involved?

Don't forget to check out the latest grant opportunities at **freebsdfoundation.org**