Red Hat Reference Architecture Series

Accelerating Red Hat Enterprise Linux 7-based Linux Containers with Solarflare OpenOnload

Jeremy Eder, Principal Software Engineer

Version 1.0

April, 2015

redhat.

100 East Davie Street
Raleigh NC 27601 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588
Research Triangle Park NC 27709 USA

The GPG fingerprint of the security@redhat.com key is:
CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

Comments and Feedback

In the spirit of open source, we invite anyone to provide feedback and comments on any reference architectures. Although we review our papers internally, sometimes issues or typographical errors are encountered. Feedback allows us to not only improve the quality of the papers we produce, but allows the reader to provide their thoughts on potential improvements and topic expansion to the papers.

Feedback on the papers can be provided by emailing refarch-feedback@redhat.com. Please refer to the title within the email.

Staying In Touch

Join us on some of the popular social media sites where we keep our audience informed on new reference architectures as well as offer related information on things we find interesting.

Like us on Facebook:

https://www.facebook.com/rhrefarch

Follow us on Twitter:

https://twitter.com/RedHatRefArch

Plus us on Google+:

https://plus.google.com/114152126783830728030/

Table of Contents

1 Executive Summary

With the June 2014 release of Red Hat Enterprise Linux 7, Red Hat debuted support for Docker and Docker-formatted containers. As of February 2015, Solarflare added support for Docker-formatted Linux Containers to their OpenOnload product.

Linux Containers combine well-established Linux kernel technologies such as namespaces, SELinux, cgroups and iptables with incredible ease of use and exceptional performance.

This paper provides installation, configuration and tuning guidance for Docker containers running on Red Hat Enterprise Linux with Solarflare OpenOnload network acceleration. It introduces container-related enhancements specific to Red Hat's distribution, such as the atomic utility, the implementation of "super-privileged containers" (SPC) and heavily builds upon the Low Latency Performance Tuning Guide for Red Hat Enterprise Linux 7. Finally, it provides a comparison of OpenOnload-accelerated bare metal network latency tests versus OpenOnload-accelerated Linux containers.

2 Hardware and Software Configuration

The following hardware and software configuration was used for these tests:

Server	(2) Dell R620, Intel Xeon E5-2690
Operating System	Red Hat Enterprise Linux 7.1 kernel 3.10.0-229
Linux Container Engine	docker-1.5.0-27
Network Cards	(2) Solarflare SFN7122F-R1 OpenOnload 201502-u1 Firmware 4.4.2.1011 Driver 4.4.1.1021

3 Install and Configure Red Hat Enterprise Linux

Best practice low latency, BIOS, and operating system tuning was implemented:

- The BIOS was configured according to Dell's Optimal BIOS settings for HPC with Dell PowerEdge 12th generation servers.
- Red Hat Enterprise Linux 7.1 was installed on the two bare metal servers.
- Recommended tuning from the Low Latency Performance Tuning Guide for Red Hat Enterprise Linux 7 was applied to the test environment. This includes:
  - The network-latency tuned profile was applied.

- The following entries were appended to the kernel command line:

```
isolcpus=1,3,5,7,9,11,13,15 nosoftlockup mce=ignore_ce
```

- netperf was bound to CPU cores PCI-local to the network adapter. More information can be found in this Red Hat Knowledgebase article.
- Subscriptions were attached to the system, and Docker installed according to the Get Started with Docker Formatted Container Images on Red Hat Systems guide.

The OpenOnload source tarball includes an openonload.spec file that was used to build RPM packages. This results in two RPM files: openonload-201502-1_u1.el7.x86_64.rpm and openonload-kmod-3.10.0-229.el7-201502_u1-1.el7.x86_64.rpm.

4 Install and Configure Solarflare OpenOnload

OpenOnload's support of Docker containers on RHEL7 relies on:

- Latest version of Docker provided by Red Hat (as of this writing, docker-1.5.0)
- OpenOnload version 201502-u1 (version on host must match the version in containers)
- Passing /dev/onload and /dev/onload_epoll devices into a container
- Docker's --net=host support to disable the use of network namespaces for this container.

As mentioned previously, there were 2 RPMs generated from the openonload.spec file included with the OpenOnload distribution. The OpenOnload kernel modules must be installed on the host. See the Onload User Guide, section 10.6 for more information.

1. Install the OpenOnload kernel modules, sfutils package and netperf benchmark:

```
# yum install -y openonload-kmod-3.10.0-229.el7-201502_u1-1.el7.x86_64.rpm \
sfutils-4.5.0.1009-1.x86_64.rpm \
netperf-2.6.0-1.el7.x86_64.rpm
```

2. Following section 10.4 of the Onload User Guide, ensure that the adapter firmware is fully updated and the firmware is set to "full-feature" mode. Full-feature mode is required to partition the adapter for an upcoming example.

```
# sfupdate --write

# sfboot pf-count=2 switch-mode=partitioning firmware-variant=full-feature
```

3. Reboot to apply the firmware change and kernel cmdline options. This must be a cold reboot/power cycle.

5 Install Container Images

The RHEL7 base image is a minimal RHEL7-based userspace environment designed to be the foundation upon which application containers are built. Red Hat currently distributes several containers that utilize the rhel7 base image: rsyslog, rhel-tools, sadc. Additional images are forthcoming.

Red Hat has also announced a Container Development Kit; a collection of tools and resources that enable developers to easily build and maintain containerized applications based on Docker for the Red Hat ecosystem.

Download RHEL7-based container images from Red Hat's image registry. The rhel7 base image is used to build a custom container image to host OpenOnload and the benchmark netperf. Attach Red Hat subscriptions to the test server (see the Get Started with Docker Formatted Container Images on Red Hat Systems guide).

1. Download (pull) the rhel7 base container:

```
-bash-4.2# docker pull rhel7
Pulling repository registry.access.redhat.com/rhel7
10acc31def5d: Download complete
Status: Downloaded newer image for registry.access.redhat.com/rhel7:latest
```

2. List the locally available images:

```
-bash-4.2# docker images
REPOSITORY                         TAG     IMAGE ID
registry.access.redhat.com/rhel7   7.1-4   10acc31def5d
registry.access.redhat.com/rhel7   latest  10acc31def5d
```

3. To make it easier to reference these images, optionally tag them with a shorter name:

```
-bash-4.2# docker tag registry.access.redhat.com/rhel7 rhel7
-bash-4.2# docker images
REPOSITORY                          TAG     IMAGE ID
rhel7                               latest  10acc31def5d
registry.access.redhat.com/rhel7    7.1-4   10acc31def5d
registry.access.redhat.com/rhel7    latest  10acc31def5d
```

4. Verify the Docker environment is functional by running a rhel7 container and displaying the content of /etc/hostname within the container:

```
# docker run rhel7 cat /etc/hostname
957b394734be
```

If this command is successful, the environment and rhel7 container image are functional.

6 Create an OpenOnload Container

Build a new Docker container image using the rhel7 image as the base, and install OpenOnload in the container, and (optionally) use the new LABEL Dockerfile command to ease use of this container when coupled with Red Hat's 'atomic' utility for super privileged containers.

1. Create a working directory for the container build, and copy the OpenOnload and netperf software into the directory:

```
# pwd
/root/openonload_container
```

2. The contents of the directory should be:

```
# ls -1
Dockerfile
netperf-2.6.0-1.el7.x86_64.rpm
openonload-201502_u1-1.el7.x86_64.rpm
openonload-kmod-3.10.0-229.el7-201502_u1-1.el7.x86_64.rpm
```

3. Create a Dockerfile similar to the following:

```
# cat Dockerfile
FROM registry.access.redhat.com/rhel7
MAINTAINER user@domain.com
COPY openonload-201502_u1-1.el7.x86_64.rpm /root/
COPY netperf-2.6.0-1.el7.x86_64.rpm /root/
RUN yum -y -q update ; yum localinstall -y -q /root/openonload*rpm \
/root/netperf*rpm ; yum clean all
LABEL RUN="docker run -d --device=/dev/onload –device=/dev/onload_epoll
--name NAME --net=host -e NAME=NAME -e IMAGE=IMAGE IMAGE"
CMD onload --profile=latency /usr/bin/netserver -D
```

4. Build the new image:

```
# docker build -t rhel7_openonload .
```

5. Upon successful build, the OpenOnload container is ready for use:

```
# docker images
REPOSITORY                      TAG                 IMAGE ID
rhel7_openonload                latest              db6e2ae1e1d9
```

6. Inspect the new image:

```
# atomic info rhel7_openonload
Vendor       : Red Hat, Inc.
Name         : rhel-server-docker
Build_Host   : rcm-img04.build.eng.bos.redhat.com
Version      : 7.1
Architecture : x86_64
Release      : 4
RUN          : docker run -it --name NAME --net=host -e NAME=NAME -e
IMAGE=IMAGE IMAGE
```

Since “FROM registry.access.redhat.com/rhel7” was used, the new image includes version information from the parent rhel7 base image (7.1-4). The RUN command comes from the rhel7_openonload Dockerfile.

7. Verify OpenOnload is installed inside the image:

```
# docker run -it rhel7_openonload rpm -q openonload netperf
openonload-201502_u1-1.el7.x86_64
netperf-2.6.0-1.el7.x86_64
```

7 Launch OpenOnload-accelerated containers

Use the atomic run command to create a new container that runs the default CMD as specified in the Dockerfile:

```
onload --profile=latency /usr/bin/netserver -D
```

This means by default, netserver uses the OpenOnload userspace network stack.

Run the rhel7_openonload container using the atomic command:

```
# atomic run rhel7_openonload
docker run -d --device=/dev/onload --device=/dev/onload_epoll --name
rhel7_openonload --net=host -e NAME=rhel7_openonload -e
IMAGE=rhel7_openonload rhel7_openonload
46100d94976b792419314af5073b04a7a21886cff76d9ce0627adb3c9efe2515
```

Notice the docker run command above uses the docker -d flag. This means the container runs in the background, has no tty associated with it, and works if the containerized application has a foreground mode. Another common example is netserver -D. httpd -DFOREGROUND.

At the time of this writing, Solarflare recommends setting --privileged mode for OpenOnload-accelerated containers. This disables kernel capability checks, and SElinux enforcement. Work is ongoing to remove the --privileged requirement.

The netserver process is visible on the host. This is the running, containerized benchmark. By default, the host’s process table is not visible from inside containers unless the --pid=host

option is used.

```
# ps aux|grep netserver
root     14550  0.0  0.0   9704   704 ?        Ss   12:07   0:00
/usr/bin/netserver -D
```

Because the –net=host has been used, the new container shares a hostname with the host itself, and has full permission over network devices in the host's network stack. The tradeoff of using --net=host is that network resources such as devices, ports, firewall rules are shared with the host.

For example, since sshd is already running in the host, a containerized ssh server would have a port conflict on port 22. sshd could be configured to listen on an alternate port (i.e., port 2222). This workaround is not suitable for applications such as httpd, where browsers expect a well-known port.

To deal with this, Solarflare provides the ability to partition a single physical NIC into up to 16 PCIe "physical functions", or PFs. This option helps run multiple containers with applications listening on the same port because each container uses a dedicated PF/interface. Section 10.4 of the Onload User Guide shows how to partition a Solarflare network adapter. For this document, 2 PFs are added:

```
# sfboot pf-count=2 switch-mode=partitioning firmware-variant=full-feature
```

A reboot is required to complete this configuration.

PFs are allocated per physical NIC port. This means that on dual-port adapters such as the SFN7122F-R1 used in this test, setting pf-count=2 allocates one additional PF on each port, for a total of 4 ports. New PFs are created in round-robin fashion. Adding two PFs created ports p1p3 (attached to p1p1) and p1p4 (attached to p1p2):

```
# for i in `find /sys/class/net/*/device/physical_port` ; \
do echo -n "$i " ; cat $i ; done
/sys/class/net/p1p1/device/physical_port 0
/sys/class/net/p1p2/device/physical_port 1
/sys/class/net/p1p3/device/physical_port 0
/sys/class/net/p1p4/device/physical_port 1
```

Further examples use interfaces p1p1 and p1p3.

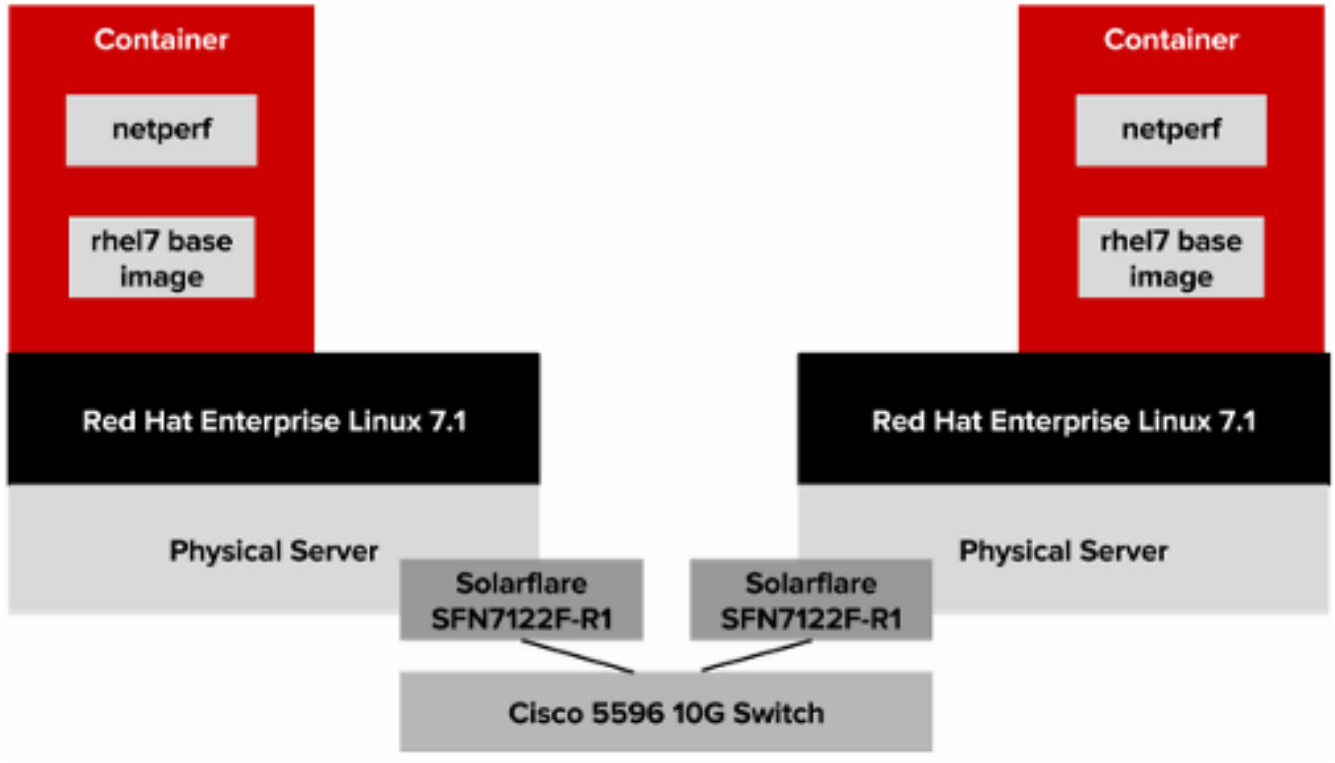
Configure IP addresses for p1p1 and p1p3 using standard methods. These interfaces can be used by a single container or two separate containers, each having their own OpenOnload stack:

```
# ip a l p1p1|grep p1p1
6: p1p1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    inet 172.16.1.1/24 brd 172.16.1.255 scope global p1p1
```

```
# ip a l p1p3|grep p1p3
9: p1p3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    inet 172.16.3.1/24 brd 172.16.3.255 scope global p1p3
```

8 OpenOnload Performance Results: bare metal vs container

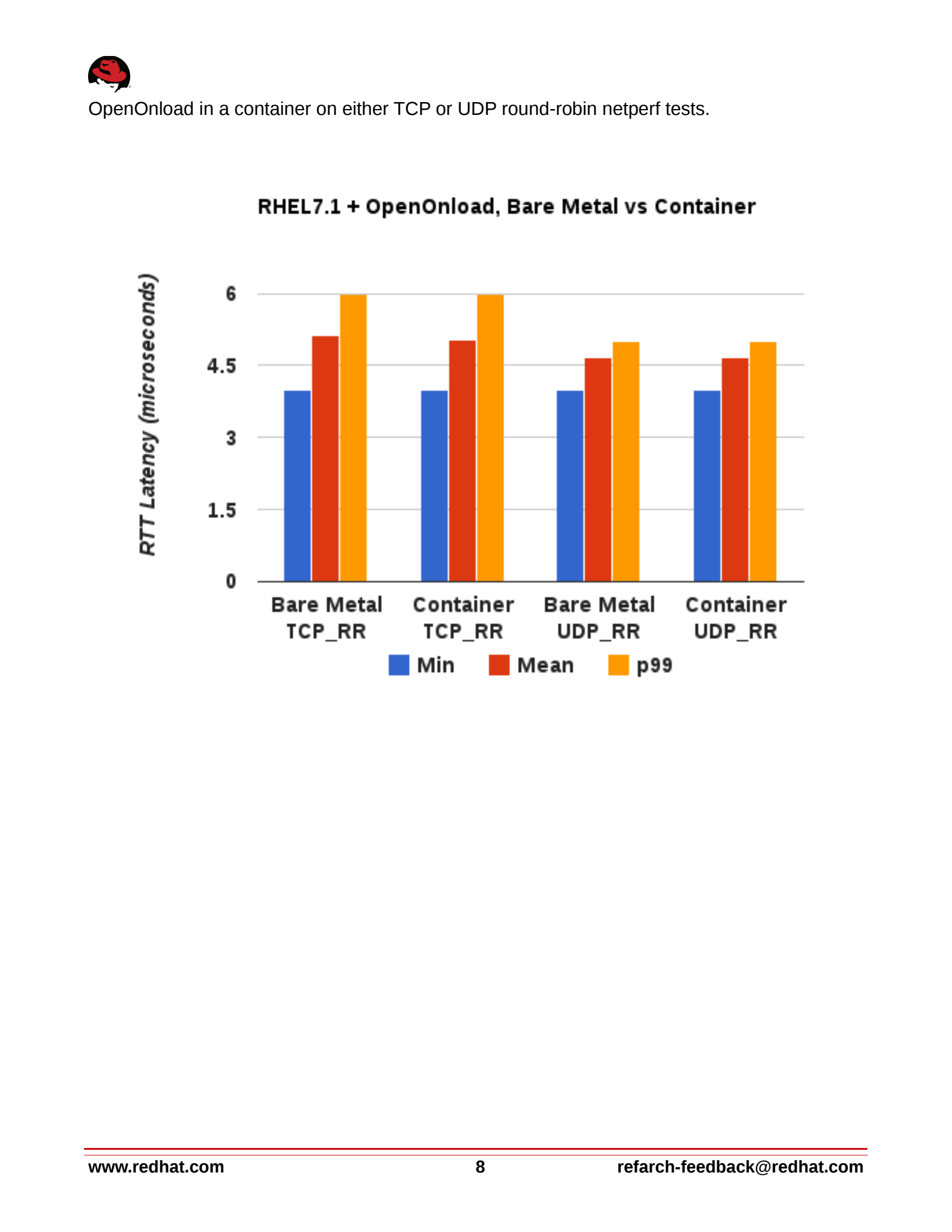
The industry standard netperf benchmark was used to determine both TCP and UDP round-trip latency between 2 physical servers and between 2 containers hosted on each server. The same physical servers were used in all tests. The servers were connected via fiber through a Cisco 5596 10G switch:



Example netperf command used:

```
# onload --profile=latency netperf -H 172.16.1.2 -D1,1 -T1,1 -t TCP_RR -I -i
5,5 -l 120 -- -O min_latency,mean_latency,p99_latency,stddev_latency
```

Data was collected for 5 runs, each two minutes in duration. The 5 samples were averaged to arrive at the results below, which indicate no perceived overhead when utilizing



OpenOnload in a container on either TCP or UDP round-robin netperf tests.

9 Conclusions

As the above test results indicate, performance of Solarflare OpenOnload-accelerated containerized application performance is equivalent to bare metal.

An added benefit of container technology from a performance perspective is that since applications run directly on the host kernel with direct access to physical resources, performance overhead is near zero for most workloads (especially network, compute and volume storage), and there is typically very little (often no) container-specific tuning. For example, numactl, taskset, hugepages and kernel task scheduler tuning can all be accomplished from within a container of sufficient privilege.

Appendix A: Revision History

Revision 1.0	04/03/15	Jeremy Eder
Initial Release		