LFS242 - Cloud Native Logging with Fluentd

Fluentd is a cross platform open-source data collector. It is written primarily in the Ruby programming language with several components developed in C for performance. Fluentd can be used to create a unified logging layer, allowing users to unify data collection and consumption across a wide array of systems and services.

Fluentd can be installed and configured for a variety of environments. Lab 1 is organized into three separate parts, each involves the installation and configuration steps necessary to get started with Fluentd in a different environment:

- A. Linux
- B. Docker
- C. Kubernetes

Lab 1-B – Running Fluentd in a Docker environment

In this lab you will get a chance to run Fluentd using the Docker container runtime. This lab can be completed on the same lab system used for Lab 1-A or on a new machine. Instructions for configuring a suitable lab machine can be found in that lab.

Docker is a platform for developers and sysadmins to develop, deploy, and run applications with containers. The use of Linux containers to deploy applications is called containerization. Containers can greatly simplify the process of application deployment, increasing deployment modularity and reliability. Containerized applications, like all applications, emit log data that needs to be processed and collated to allow developers and administrators to detect trends and research problems. In this lab we'll see how Fluentd can play a key role in the log management of containerized applications.

Objectives

- Learn how to install Docker CE.
- Learn how to run Fluentd in a Docker environment.
- Learn how to process logs from containerized applications with Fluentd.

1. Prepare the lab system

If you have not already, log back into your lab system. Before we begin working with containers, make sure that any Fluentd instances or other programs started in prior labs are stopped.

2. Install Docker

Docker is available in a commercial Enterprise Edition (Docker EE) and a free open source Community Edition (Docker CE), though the core docker container runtime functionality is the same in both. We will install Docker CE on our lab systems. The docker web site provides an easy to use installation script that we can run to install docker.

In a new shell on your lab system, run the get.docker.com script to install Docker CE:

```
ubuntu@labsys:~$ wget -O - https://get.docker.com | sh

--2022-06-15 16:58:21--  https://get.docker.com/
Resolving get.docker.com (get.docker.com)... 13.32.208.79, 13.32.208.39, 13.32.208.37, ...
Connecting to get.docker.com (get.docker.com)|13.32.208.79|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20009 (20K) [text/plain]
Saving to: ‘STDOUT’

-                                           100%
[=======================================================================================>]
19.54K  --.-KB/s    in 0s
```

```
2022-06-15 16:58:21 (56.9 MB/s) - written to stdout [20009/20009]

# Executing docker install script, commit: b2e29ef7a9a89840d2333637f7d1900a83e7153f
+ sudo -E sh -c apt-get update -qq >/dev/null
+ sudo -E sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq apt-transport-https ca-
certificates curl >/dev/null
+ sudo -E sh -c mkdir -p /etc/apt/keyrings && chmod -R 0755 /etc/apt/keyrings
+ sudo -E sh -c curl -fsSL "https://download.docker.com/linux/ubuntu/gpg" | gpg --dearmor --yes -o
/etc/apt/keyrings/docker.gpg
+ sudo -E sh -c chmod a+r /etc/apt/keyrings/docker.gpg
+ sudo -E sh -c echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu focal stable" > /etc/apt/sources.list.d/docker.list
+ sudo -E sh -c apt-get update -qq >/dev/null
+ sudo -E sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq --no-install-recommends
docker-ce docker-ce-cli containerd.io docker-compose-plugin docker-scan-plugin >/dev/null
+ version_gte 20.10
+ [ -z  ]
+ return 0
+ sudo -E sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq docker-ce-rootless-extras
>/dev/null
+ sudo -E sh -c docker version
Client: Docker Engine - Community
 Version:           20.10.17
 API version:       1.41
 Go version:        go1.17.11
 Git commit:        100c701
 Built:             Mon Jun  6 23:02:57 2022
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.17
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.17.11
  Git commit:       a89b842
  Built:            Mon Jun  6 23:01:03 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.6
  GitCommit:        10c12954828e7c7c9b6e0ea9b0c02b01407d3ae1
 runc:
  Version:          1.1.2
  GitCommit:        v1.1.2-0-ga916309
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0

================================================================================

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
```

```
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface'
         documentation for details: https://docs.docker.com/go/attack-surface/

================================================================================

ubuntu@labsys:~$
```

The command above sends the get.docker.com install script to a pipe (`|`) connected to a Bourne shell (`sh`). The Bourne shell runs the script, adding the Docker package repository and key to the local apt cache, then installing docker from packages.

At the bottom of the output the script reports the docker version installed and provides instructions on running Docker in rootless mode. Normally the dockerd daemon runs as root and can run arbitrary containers as root on the host. This makes users who have access to docker very powerful. By default only root and members of the docker group can run docker commands.

Install Docker in rootless mode. First, install the `uidmap` package from apt:

```
ubuntu@labsys:~$ sudo apt install uidmap -y

...

ubuntu@labsys:~$
```

Next, run the Docker rootless mode installation script:

```
ubuntu@labsys:~$ dockerd-rootless-setuptool.sh install

[INFO] Creating /home/ubuntu/.config/systemd/user/docker.service
[INFO] starting systemd service docker.service
+ systemctl --user start docker.service
+ sleep 3
+ systemctl --user --no-pager --full status docker.service
● docker.service - Docker Application Container Engine (Rootless)
     Loaded: loaded (/home/ubuntu/.config/systemd/user/docker.service; disabled; vendor preset:
enabled)
     Active: active (running) since Wed 2022-06-15 17:06:48 UTC; 3s ago
       Docs: https://docs.docker.com/go/rootless/
   Main PID: 20459 (rootlesskit)
     CGroup: /user.slice/user-1000.slice/user@1000.service/docker.service
             ├─20459 rootlesskit --net=slirp4netns --mtu=65520 --slirp4netns-sandbox=auto --
slirp4netns-seccomp=auto --disable-host-loopback --port-driver=builtin --copy-up=/etc --copy-up=/run
--propagation=rslave /usr/bin/dockerd-rootless.sh
             ├─20470 /proc/self/exe --net=slirp4netns --mtu=65520 --slirp4netns-sandbox=auto --
slirp4netns-seccomp=auto --disable-host-loopback --port-driver=builtin --copy-up=/etc --copy-up=/run
--propagation=rslave /usr/bin/dockerd-rootless.sh
             ├─20486 slirp4netns --mtu 65520 -r 3 --disable-host-loopback --enable-sandbox --enable-
seccomp 20470 tap0
             ├─20493 dockerd
             └─20508 containerd --config /run/user/1000/docker/containerd/containerd.toml --log-
level info

Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.514638655Z"
level=warning msg="Your kernel does not support CPU realtime scheduler"
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.514674527Z"
level=warning msg="Your kernel does not support cgroup blkio weight"
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.514685253Z"
level=warning msg="Your kernel does not support cgroup blkio weight_device"
```

```
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.515295476Z"
level=info msg="Loading containers: start."
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.639161573Z"
level=info msg="Default bridge (docker0) is assigned with an IP address 172.17.0.0/16. Daemon option
--bip can be used to set a preferred IP address"
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.706986336Z"
level=info msg="Loading containers: done."
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.725144035Z"
level=warning msg="Not using native diff for overlay2, this may cause degraded performance for
building images: running in a user namespace" storage-driver=overlay2
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.725845647Z"
level=info msg="Docker daemon" commit=a89b842 graphdriver(s)=overlay2 version=20.10.17
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.726124312Z"
level=info msg="Daemon has completed initialization"
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.812766614Z"
level=info msg="API listen on /run/user/1000/docker.sock"
+ DOCKER_HOST=unix:///run/user/1000/docker.sock /usr/bin/docker version
Client: Docker Engine - Community
 Version:           20.10.17
 API version:       1.41
 Go version:        go1.17.11
 Git commit:        100c701
 Built:             Mon Jun  6 23:02:57 2022
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.17
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.17.11
  Git commit:       a89b842
  Built:            Mon Jun  6 23:01:03 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.6
  GitCommit:        10c12954828e7c7c9b6e0ea9b0c02b01407d3ae1
 runc:
  Version:          1.1.2
  GitCommit:        v1.1.2-0-ga916309
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
+ systemctl --user enable docker.service
Created symlink /home/ubuntu/.config/systemd/user/default.target.wants/docker.service →
/home/ubuntu/.config/systemd/user/docker.service.
[INFO] Installed docker.service successfully.
[INFO] To control docker.service, run: `systemctl --user (start|stop|restart) docker.service`
[INFO] To run docker.service on system startup, run: `sudo loginctl enable-linger ubuntu`

[INFO] Creating CLI context "rootless"
Successfully created context "rootless"

[INFO] Make sure the following environment variables are set (or add them to ~/.bashrc):

export PATH=/usr/bin:$PATH
export DOCKER_HOST=unix:///run/user/1000/docker.sock

ubuntu@labsys:~$
```

Finally, modify the `PATH` and `DOCKER_HOST` variables as suggested at the end of the script:

```
ubuntu@labsys:~$ export PATH=/usr/bin:$PATH

ubuntu@labsys:~$ export DOCKER_HOST=unix:///run/user/1000/docker.sock

ubuntu@labsys:~$
```

2.a. Verify the Docker installation

Check the docker version to insure the docker command line client in properly installed:

```
ubuntu@labsys:~$ docker --version

Docker version 20.10.17, build 100c701

ubuntu@labsys:~$
```

You can get command line help with the docker help switch. Try it:

```
ubuntu@labsys:~$ docker --help

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/home/ubuntu/.docker")
  -c, --context string     Name of the context to use to connect to the daemon (overrides
DOCKER_HOST env var and default context set with "docker context use")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default
"info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default
"/home/ubuntu/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default "/home/ubuntu/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/home/ubuntu/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

...

ubuntu@labsys:~$
```

Both of the above commands invoke the docker command line program but neither connect to the docker daemon. To test the full docker installation run the docker info command:

```
ubuntu@labsys:~$ docker info

Client:
 Context:    default
 Debug Mode: false
 Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  buildx: Docker Buildx (Docker Inc., v0.8.2-docker)
```

```
   compose: Docker Compose (Docker Inc., v2.6.0)
   scan: Docker Scan (Docker Inc., v0.17.0)

 Server:
  Containers: 0
   Running: 0
   Paused: 0
   Stopped: 0
  Images: 0
  Server Version: 20.10.17
  Storage Driver: overlay2
   Backing Filesystem: extfs
   Supports d_type: true
   Native Overlay Diff: false
   userxattr: true
  Logging Driver: json-file
  Cgroup Driver: none
  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: 10c12954828e7c7c9b6e0ea9b0c02b01407d3ae1
  runc version: v1.1.2-0-ga916309
  init version: de40ad0
  Security Options:
   seccomp
    Profile: default
   rootless
  Kernel Version: 5.13.0-1022-aws
  Operating System: Ubuntu 20.04.4 LTS
  OSType: linux
  Architecture: x86_64
  CPUs: 2
  Total Memory: 3.775GiB
  Name: ip-172-31-9-135
  ID: 26TG:DAZN:RLQL:OL2A:WPMF:WTOX:5P2N:YMST:T46U:WOQ2:ROH4:PBYN
  Docker Root Dir: /home/ubuntu/.local/share/docker
  Debug Mode: false
  Registry: https://index.docker.io/v1/
  Labels:
  Experimental: false
  Insecure Registries:
   127.0.0.0/8
  Live Restore Enabled: false

 WARNING: Running in rootless-mode without cgroups. To enable cgroups in rootless-mode, you need to
 boot the system in cgroup v2 mode.

 ubuntu@labsys:~$
```

> N.B. You can safely ignore any warnings regarding cgroups - this lab will not be using any container options related to them.

This command causes the docker client to connect to dockerd and output various pieces of information associated with the dockerd daemon.

Look through the output of the info command and answer these questions:

- What logging driver is dockerd using?
- A registry, in docker terms, is a network server from which container images can be downloaded, which "Registry" is docker using as a default?
- The docker daemon (dockerd) is sometimes referred to as the docker server, what is the docker server version?
- In docker, plugins extend the functionality of the basic dockerd services, what log plugins are available?

3. Running Fluentd in a container

In this step we'll run the latest public containerized release of Fluentd from the docker hub registry. Fluentd is a Cloud Native Computing Foundation open source project with source code available on GitHub under the Fluent organization: https://github.com/fluent. The Fluentd project community also hosts prebuilt Fluentd docker images on docker hub under the Fluent organization: https://hub.docker.com/u/fluent.

As we have seen in previous labs, using Fluentd in most settings will involve running it with a specific configuration file and the necessary third party plugins (if any). By default, containerized applications can only read and write files from within their isolated container filesystem. Thus two applications running in separate containers on the same host will not see each other's files. In particular, /fluent/etc/fluent.conf in one container is a completely separate file from /fluent/etc/fluent.conf in another container and both are sperate from /fluent/etc/fluent.conf on the host.

We can, however, run a stock Fluentd container with a custom configuration file by mapping the configuration file from a host path into the container using a container "volume". "Volumes" (in the vocabulary of containers) refer to storage mapped into the container from the outside world. Such storage can be as simple as a directory on the host computer's hard disk, or as complex as a dynamically mounted iSCSI, network attached, block storage device. For our Fluentd container purposes, mapping in a host path containing our Fluentd configuration file will work perfectly.

Let's start by creating a simple Fluentd configuration file on the host that we can later map into our Fluentd container with a volume. Create the following docker.conf Fluentd configuration in the fluent/ folder:

```
ubuntu@labsys:~$ mkdir ~/fluent

ubuntu@labsys:~$ nano ~/fluent/docker.conf && $_

<source>
  @type http
  port 24220
  bind 0.0.0.0
</source>

<source>
  @type forward
  port 24224
  bind 0.0.0.0
</source>

<match **>
  @type stdout
</match>

ubuntu@labsys:~$
```

- What type of input plugins are being used in this configuration?
- What ports will Fluentd be listening on?
- What type of output plugins are being used in this configuration?

Now run the fluent organization Fluentd stable release container with our ~/fluent directory mapped into the container /fluent/etc

directory:

```
ubuntu@labsys:~$ docker run -d -p 24220:24220 -p 24224:24224 \
-v $HOME/fluent:/fluentd/etc -e FLUENTD_CONF=docker.conf --name fluentd fluent/fluentd:v1.14-1

Unable to find image 'fluent/fluentd:v1.14-1' locally
v1.14-1: Pulling from fluent/fluentd
6097bfa160c1: Pull complete
6f3790a59bdb: Pull complete
10979cec8705: Pull complete
f8fd29633127: Pull complete
4847fc0f8623: Pull complete
Digest: sha256:3749047d00b40bd5854df8dd8dd036aaf4ea3e12c2d4ed3c3ee2b329abfd6ddf
Status: Downloaded newer image for fluent/fluentd:v1.14-1
120249ab5531c4d94c3e2970519329567540b2bc00c7b2396597b4a8c5e04da4

ubuntu@labsys:~$
```

Let's review the parts of the docker command we used to launch our Fluentd instance:

- `docker` - this is the docker command line program
- `run` - the docker run subcommand runs a new container, downloading the container image if it is not found locally
- `-p 24220:24220` - the port switch (-p) maps a port from the host to the container
- `-p 24224:24224` - the port switch can be used multiple times to create multiple port mappings
- `-v $HOME/fluent:/fluentd/etc` - the volume switch (-v) maps a path on the host to a path in the container
- `-e FLUENTD_CONF=docker.conf` - the environment switch (-e) creates an environment variable inside the container
- `--name fluentd` - provides a user-defined name for the image, making it easy to refer to this container
- `fluent/fluentd:<tag>` - this is the image to run in organization/image:tag format (tags are often used to pull different versions of the same image)

Our configuration file asks Fluentd to listen on ports 24220 and 24224 which we have mapped to the host in this example so that clients outside of the Fluentd container can reach Fluentd. We have also mapped our fluent directory on the host into the container's /fluent/etc directory. This is the default directory that Fluentd will inspect for configuration files. By setting the FLUENTD_CONF environment variable to docker.conf we have effectively asked the containerized Fluentd to use the /fluent/etc/docker.conf configuration file, which maps to ~/fluent/docker.conf on the host.

Since the docker `-d` flag is daemonizing the container, so you need to use `docker logs` to view its output as it runs in the background.

```
ubuntu@labsys:~$ docker logs fluentd

2021-07-22 18:14:32 +0000 [info]: parsing config file is succeeded path="/fluentd/etc/docker.conf"
2021-07-22 18:14:32 +0000 [info]: gem 'fluentd' version '1.13.2'
2021-07-22 18:14:32 +0000 [warn]: define <match fluent.**> to capture fluentd logs in top level is
deprecated. Use <label @FLUENT_LOG> instead
2021-07-22 18:14:32 +0000 [info]: using configuration file: <ROOT>
  <source>
    @type http
    port 24220
    bind "0.0.0.0"
  </source>
  <source>
    @type forward
    port 24224
    bind "0.0.0.0"
  </source>
  <match **>
```

```
    @type stdout
  </match>
</ROOT>
2021-07-22 18:14:32 +0000 [info]: starting fluentd-1.13.2 pid=8 ruby="2.7.3"
2021-07-22 18:14:32 +0000 [info]: spawn command to main:  cmdline=["/usr/bin/ruby", "-Eascii-
8bit:ascii-8bit", "/usr/bin/fluentd", "-c", "/fluentd/etc/docker.conf", "-p", "/fluentd/plugins", "-
-under-supervisor"]
2021-07-22 18:14:33 +0000 [info]: adding match pattern="**" type="stdout"
2021-07-22 18:14:33 +0000 [info]: adding source type="http"
2021-07-22 18:14:33 +0000 [info]: adding source type="forward"
2021-07-22 18:14:33 +0000 [warn]: #0 define <match fluent.**> to capture fluentd logs in top level
is deprecated. Use <label @FLUENT_LOG> instead
2021-07-22 18:14:33 +0000 [info]: #0 starting fluentd worker pid=17 ppid=8 worker=0
2021-07-22 18:14:33 +0000 [info]: #0 listening port port=24224 bind="0.0.0.0"
2021-07-22 18:14:33 +0000 [info]: #0 fluentd worker is now running worker=0
2021-07-22 18:14:33.128930680 +0000 fluent.info: {"pid":17,"ppid":8,"worker":0,"message":"starting
fluentd worker pid=17 ppid=8 worker=0"}
2021-07-22 18:14:33.129215627 +0000 fluent.info: {"port":24224,"bind":"0.0.0.0","message":"listening
port port=24224 bind=\"0.0.0.0\""}
2021-07-22 18:14:33.130459979 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}

ubuntu@labsys:~$
```

Examine the containerized Fluentd output.

- Identify the log lines that report docker downloading the Fluentd container image
- Locate the log output that reports the Fluentd configuration file in use
- What version of Fluentd is running in the container?

Now that we have Fluentd running, let's test it! We'll use `curl` to send a simple HTTP message to the containerized Fluentd instance. Recall that we configured Fluentd to listen for HTTP traffic on interface 0.0.0.0 (all interfaces) and port 24220. We can use the curl command to POST a simple JSON message to locahost:24220. Run the following command in a new shell:

```
ubuntu@labsys:~$ curl -X POST -d 'json={"hi":"mom"}' http://localhost:24220/test

ubuntu@labsys:~$
```

In the Fluentd terminal, note the stdout output of the posted message:

```
ubuntu@labsys:~$ docker logs fluentd | tail -5

2022-06-15 17:38:58 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-15 17:38:58.258981881 +0000 fluent.info: {"pid":16,"ppid":7,"worker":0,"message":"starting
fluentd worker pid=16 ppid=7 worker=0"}
2022-06-15 17:38:58.259342336 +0000 fluent.info: {"port":24224,"bind":"0.0.0.0","message":"listening
port port=24224 bind=\"0.0.0.0\""}
2022-06-15 17:38:58.260795935 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}
2022-06-15 17:39:41.169533020 +0000 test: {"hi":"mom"}

ubuntu@labsys:~$
```

- How did the route in the curl command (`/test`) manifest in the container output?
- Try this curl command: `curl -X POST -d 'json={"hi":"mom"}' http://localhost:24220/test/ing`
- Try this curl command:
  `curl -X POST -d 'json={"hi":"mom", "severity":"superbad"}' http://localhost:24220/test/ing`

- Experiment with some other curl commands of your own

4. Using Fluentd to process container log data

Now that we have Fluentd up and running in a container, imagine we need to run multiple containers on this machine and that we'd like all of their log output to be collected and processed by the Fluentd instance we are already running.

Recall that in the configuration we used in the previous step we configured the following source:

```
<source>
  @type forward
  port 24224
  bind 0.0.0.0
</source>
```

This source forwards traffic inbound on port 24224. Docker's own integrated Fluentd log driver plugin forwards traffic to localhost:24224 by default. This means that as long as we have some Fluentd instance listening on localhost:24224 (which we do), any docker container using the Fluentd log drive plugin will send log output to that container. In the world of containers, any text written to stdout or stderr is piped to docker and delivered to the terminal (if any) and the configured log driver. By default Docker logs container output to files in /var/lib/docker, however we can select a different default log driver and/or specify a specific log driver for each container we run.

The docker run `--log-driver=fluentd` flag tells Docker to send output from that container to Fluentd. Try it by running an nginx container set to log using the Fluentd docker plugin:

```
ubuntu@labsys:~$ docker run -P -d --log-driver=fluentd --name nginx nginx

Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
33847f680f63: Pull complete
dbb907d5159d: Pull complete
8a268f30c42a: Pull complete
b10cf527a02d: Pull complete
c90b090c213b: Pull complete
1f41b2f2bf94: Pull complete
Digest: sha256:8f335768880da6baf72b70c701002b45f4932acae8d574dedfddaf967fc3ac90
Status: Downloaded newer image for nginx:latest
0f21b957ab5a0e58a19b08d0fe6a21294b4590142a67903adf5b9d04dfbedfa3

ubuntu@labsys:~$
```

Let's examine the docker command used:

- `docker` - the docker command line program
- `run` - creates a new container, downloading the image if it is not present
- `-P` - maps all of the container's exposed ports to the host (80 and 443 in this case)
- `-d` - run the container detached, in the background
- `--log-driver=fluentd` - sets the log driver plugin to Fluentd
- `--name nginx` - to name the nginx container with something easy to reference
- `nginx` - specifies the container image to run, in this case using the default registry (hub.docker.com), the default organization (none, aka official), the nginx repository and the default tag ("latest")

List the currently running docker containers with `docker container ls`:

```
ubuntu@labsys:~$ docker container ls
```

```
CONTAINER ID   IMAGE                    COMMAND                  CREATED         STATUS
PORTS
NAMES
663b62907d68   nginx                    "/docker-entrypoint.…"   7 minutes ago   Up 7 minutes
0.0.0.0:49153->80/tcp, :::49153->80/tcp
nginx
120249ab5531   fluent/fluentd:v1.14-1   "tini -- /bin/entryp…"   9 minutes ago   Up 9 minutes
0.0.0.0:24220->24220/tcp, :::24220->24220/tcp, 5140/tcp, 0.0.0.0:24224->24224/tcp, :::24224-
>24224/tcp   fluentd

ubuntu@labsys:~$
```

The container running the `nginx` image has been assigned the localhost port `49153`, which maps to port 80 in that container. This means that the container should be accessible through `curl` on localhost:49153.

Try to `curl` nginx on `localhost`:

> N.B. Make sure you check the ports assignments on your terminal, the port your container receives may be different!

```
ubuntu@labsys:~$ curl http://localhost:49153

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

ubuntu@labsys:~$
```

The `curl` should return the default nginx landing page. The nginx instance inside the container logs all web hits to stdout. Take a look at the Fluentd container's terminal:

```
ubuntu@labsys:~$ docker logs fluentd | tail -5

2022-06-15 17:40:37.000000000 +0000 663b62907d68: {"log":"2022/06/15 17:40:37 [notice] 1#1:
getrlimit(RLIMIT_NOFILE):
1048576:1048576","container_id":"663b62907d687a931831aeabc65c6591da35acf540f9c300341fc82603f4b5cc","
container_name":"/nginx","source":"stderr"}
```

```
2022-06-15 17:40:37.000000000 +0000 663b62907d68: {"source":"stderr","log":"2022/06/15 17:40:37
[notice] 1#1: start worker
processes","container_id":"663b62907d687a931831aeabc65c6591da35acf540f9c300341fc82603f4b5cc","contai
ner_name":"/nginx"}
2022-06-15 17:40:37.000000000 +0000 663b62907d68:
{"container_id":"663b62907d687a931831aeabc65c6591da35acf540f9c300341fc82603f4b5cc","container_name":
"/nginx","source":"stderr","log":"2022/06/15 17:40:37 [notice] 1#1: start worker process 32"}
2022-06-15 17:40:37.000000000 +0000 663b62907d68: {"source":"stderr","log":"2022/06/15 17:40:37
[notice] 1#1: start worker process
33","container_id":"663b62907d687a931831aeabc65c6591da35acf540f9c300341fc82603f4b5cc","container_nam
e":"/nginx"}
2022-06-15 17:48:24.000000000 +0000 663b62907d68:
{"container_id":"663b62907d687a931831aeabc65c6591da35acf540f9c300341fc82603f4b5cc","container_name":
"/nginx","source":"stdout","log":"172.17.0.1 - - [15/Jun/2022:17:48:24 +0000] \"GET / HTTP/1.1\" 200
615 \"-\" \"curl/7.68.0\" \"-\""}

ubuntu@labsys:~$
```

Fluentd received all logged events (including application-level events reported by NGINX) from the NGINX container via Docker. The Fluentd forward plugin requires log input in a particular format. Fortunately the built-in Docker Fluentd plugin automatically formats container output for Fluentd while enriching it with container metadata, like the container ID and name.

Using information from the lab vm, answer the following questions:

- Where did the logged event come from?
- What is the actual event text?

5. Cleanup

As a final step we'll shutdown and delete all of the containers we started in this lab. We can send the container ids to the `docker container rm` command to remove them, but first you need to stop the containers:

```
ubuntu@labsys:~$ docker container stop fluentd nginx

fluentd
nginx

ubuntu@labsys:~$
```

Now remove the containers using Docker. This removes them entirely, freeing up your system for later:

```
ubuntu@labsys:~$ docker container rm fluentd nginx

fluentd
nginx

ubuntu@labsys:~$
```

Verify that all of the containers have been removed:

```
ubuntu@labsys:~$ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

ubuntu@labsys:~$
```

Perfect!

In this lab we ran Fluentd in a Docker container and used it to manage log output from other containers. We'll learn a lot more about Fluentd in the labs ahead.

Congratulations, you have completed the Lab.