Introduction

Traditionally, designing and implementing centralized logging is an after-thought. It is not until problems arise that priorities shift to a centralized logging solution to query, view, and analyze the logs so the root-cause of the problem can be found. However, in the container era, when designing a Containers-as-a-Service (CaaS) platform with Docker Enterprise, it is critical to prioritize centralized logging. As the number of micro-services deployed in containers increases, the amount of data produced by them in the form of logs (or events) exponentially increases.

What You Will Learn

This reference architecture provides an overview of how Docker logging works, explains the two main categories of Docker logs, and then discusses Docker logging best practices.

Understanding Docker Logging

Before diving into design considerations, it's important to start with the basics of Docker logging.

Docker supports different logging drivers used to store and/or stream container stdout and stderr logs of the main container process (pid 1). By default, Docker uses the json-file logging driver, but it can be configured to use many other drivers ([https://docs.docker.com/config/containers/logging/#supported-logging-drivers](https://docs.docker.com/config/containers/logging/#supported-logging-drivers)) by setting the value of log-driver in /etc/docker/daemon.json followed by restarting the Docker daemon to reload its configuration.

The logging driver settings apply to ALL containers launched after reconfiguring the daemon (restarting existing containers after reconfiguring the logging driver does not result in containers using the updated config). To override the default container logging driver run the container with --log-driver and --log-opt options. Swarm-mode services, on the other hand, can be updated to use a different logging driver on the go by using docker service update --log-driver <DRIVER_NAME> --log-opt <LIST OF OPTIONS> <SERVICE NAME>.

What about Docker Engine logs? These logs are typically handled by the default system manager logger. Most of the modern distros (CentOS 7, RHEL 7, Ubuntu 16, etc.) use systemd, which uses journal for logging and journalctl for accessing the logs. To access the Engine logs use journalctl -u docker.service.

Docker Logs Categories and Sources

Now that the basics of Docker logging have been covered, this section explains their categories and sources.

Docker logs typically fall into one of two categories: Infrastructure Management or Application logs. Most logs naturally fall into these categories based on the roles of who needs access to the logs.

- Operators are mostly concerned with the stability of the platform as well as the availability of the services.
- Developers are more concerned with their application code and how their service is performing.

In order to have a self-service platform, both operators and developers should have access to the logs they need in order to perform their role. DevOps practices suggest that there is an overall, shared responsibility when it comes to service availability and performance. However, everyone shouldn't need access to every log on the platform. For instance, developers should only need access to the logs for their services and the integration points. Operators are more concerned with Docker daemon logs, UCP and DTR availability, as well as service availability. There is a bit of overlap since developers and operators both should be aware of service availability. Having access to the logs that each role needs allows for simpler troubleshooting when an issue occurs and a decreased Mean Time To Resolve (MTTR).
Infrastructure Management Logs

The infrastructure management logs include the logs of the Docker Engine, containers running UCP or DTR, and any containerized infrastructure services that are deployed (think containerized monitoring agents).

**Docker Engine Logs**

As previously mentioned, Docker Engine logs are captured by the OS's system manager by default. These logs can be sent to a centralized logging server.

**UCP and DTR System Logs**

UCP and DTR are deployed as Docker containers. All their logs are captured in the containers' STDOUT/STDERR. The default logging driver for Docker Engine captures these logs.

UCP can be configured to use remote syslog logging. This can be done post-installation from the UCP UI for all of its containers.

Note: It is recommended that the Docker Engine default logging driver be configured before installing UCP and DTR so that their logs are captured by the chosen logging driver. This is due to the inability to change a container's logging driver once it had been created. The only exception to this is ucp-agent, which is a component of UCP that gets deployed as a Swarm service.

**Infrastructure Services**

Infrastructure operation teams deploy containerized infrastructure services used for various infrastructure operations such as monitoring, auditing, reporting, config deployment, etc. These services also produce important logs that need to be captured. Typically, their logs are limited to the STDOUT/STDERR of their containers, so they are also captured by the Docker Engine default logging driver. If not, they need to be handled separately.

**Application Logs**

Application-produced logs can be a combination of custom application logs and the STDOUT/STDERR logs of the main process of the application. As described earlier, the STDOUT/STDERR logs of all containers are captured by the Docker Engine default logging driver. So, no need to do any custom configuration to capture them. If the application has custom logging (e.g. writes logs to /var/log/myapp.log within the container), it's important to take that into consideration.

**Docker Logging Design Considerations**

Understanding the types of Docker logs is important. It is also important to define which entities are best suited to consume and own them.

**Categorizing the Docker Logs**

Mainly, there are two categories: infrastructure logs and application logs.

**Defining the Organizational Ownership**
Based on the organization's structure and policies, decide if these categories have a direct mapping to existing teams. If they do not, then it is important to define the right organization or team responsible for these log categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>System and Management Logs</td>
<td>Infrastructure Operations</td>
</tr>
<tr>
<td>Application Logs</td>
<td>Application Operations</td>
</tr>
</tbody>
</table>

If the organization is part of a larger organization, these categories may be too broad. Sub-divide them into more specific ownership teams:

<table>
<thead>
<tr>
<th>Category</th>
<th>Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker Engine Logs</td>
<td>Infrastructure Operations</td>
</tr>
<tr>
<td>Infrastructure Services</td>
<td>Infrastructure Operations</td>
</tr>
<tr>
<td>UCP and DTR Logs</td>
<td>UCP/DTR Operations</td>
</tr>
<tr>
<td>Application A Logs</td>
<td>Application A Operations</td>
</tr>
<tr>
<td>Application B Logs</td>
<td>Application B Operations</td>
</tr>
</tbody>
</table>

Some organizations don't distinguish between infrastructure and application operations, so they might combine the two categories and have a single operations team own them:

<table>
<thead>
<tr>
<th>Category</th>
<th>Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>System and Management Logs</td>
<td>Infrastructure Operations</td>
</tr>
<tr>
<td>Application Logs</td>
<td>Infrastructure Operations</td>
</tr>
</tbody>
</table>

Pick the right model to clearly define the appropriate ownership for each type of log, resulting in decreased mean time to resolve (MTTR). Once organizational ownership has been determined for the type of logs, it is time to start investigating the right logging solution for deployment.

### Picking a Logging Infrastructure

Docker can easily integrate with existing logging tools and solutions. Most of the major logging utilities in the logging ecosystem have developed Docker logging or provided proper documentation to integrate with Docker.

Pick the logging solution that:

1. Allows for the implementation of the organizational ownership model defined in the previous section. For example, some organizations may choose to send all logs to a single logging infrastructure and then provide the right level of access to the functional teams.

2. The organization is most familiar with. Docker can integrate with most of the popular logging providers. Please refer to your logging provider's documentation for additional information.

3. Has Docker integration: pre-configured dashboards, stable Docker plugin, proper documentation, etc.

### Application Log Drivers
Docker has several available logging drivers that can be used for the management of application logs. Check the Docker docs (https://docs.docker.com/config/containers/logging/#supported-logging-drivers) for the complete list as well as detailed information on how to use them. Many logging vendors have agents that can be used to collect and ship the logs, please refer to their official documentation on how to configure those agents with Docker Enterprise.

As a general rule, if you already have logging infrastructure in place, then you should use the logging driver for that existing infrastructure. Below is a list of the logging drivers built-in to the Docker engine.

<table>
<thead>
<tr>
<th>Driver</th>
<th>Advantages Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Ultra-secure, since nothing gets logged</td>
</tr>
<tr>
<td>local (<a href="https://docs.docker.com/config/containers/logging/local/">https://docs.docker.com/config/containers/logging/local/</a>)</td>
<td>Optimized for performance and disk use. Limits on log size by default.</td>
</tr>
<tr>
<td>json-file (<a href="https://docs.docker.com/engine/admin/logging/json-file/">https://docs.docker.com/engine/admin/logging/json-file/</a>)</td>
<td>The default, supports tags</td>
</tr>
<tr>
<td>syslog (<a href="https://docs.docker.com/engine/admin/logging/syslog/">https://docs.docker.com/engine/admin/logging/syslog/</a>)</td>
<td>Most machines come with syslog, supports TLS for encrypted log shipping, supports tags.</td>
</tr>
<tr>
<td>journald (<a href="https://docs.docker.com/engine/admin/logging/journald/">https://docs.docker.com/engine/admin/logging/journald/</a>)</td>
<td>Log aggregator can be down without impact by spooling locally, this also collects Docker daemon logs</td>
</tr>
<tr>
<td>gelf (<a href="https://docs.docker.com/engine/admin/logging/gelf/">https://docs.docker.com/engine/admin/logging/gelf/</a>)</td>
<td>Provides indexable fields by defaults (container id, host, container name, etc.), tag support.</td>
</tr>
<tr>
<td>fluentd (<a href="https://docs.docker.com/engine/admin/logging/fluentd/">https://docs.docker.com/engine/admin/logging/fluentd/</a>)</td>
<td>Provides container_name and container_id fields by default, fluentd supports multiple outputs.</td>
</tr>
<tr>
<td>awslogs</td>
<td>Easy integration</td>
</tr>
<tr>
<td>Logging Service</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>AWS Logs</strong> (<a href="https://docs.docker.com/engine/admin/logging/awslogs/">https://docs.docker.com/engine/admin/logging/awslogs/</a>)</td>
<td>When using Amazon Web Services, less infrastructure to maintain, tag support, centralized view of logs.</td>
</tr>
<tr>
<td><strong>Splunk</strong> (<a href="https://docs.docker.com/engine/admin/logging/splunk/">https://docs.docker.com/engine/admin/logging/splunk/</a>)</td>
<td>Easy integration with Splunk, TLS support, highly configurable, tag support, additional metrics. Works on Windows.</td>
</tr>
<tr>
<td><strong>Etwlogs</strong> (<a href="https://docs.docker.com/engine/admin/logging/etwlogs/">https://docs.docker.com/engine/admin/logging/etwlogs/</a>)</td>
<td>Common framework for logging on Windows, default indexable values</td>
</tr>
<tr>
<td><strong>GCP Logs</strong> (<a href="https://docs.docker.com/engine/admin/logging/gcplogs/">https://docs.docker.com/engine/admin/logging/gcplogs/</a>)</td>
<td>Simple integration with Google Compute, less infrastructure to maintain, tag support, centralized view of logs.</td>
</tr>
<tr>
<td><strong>Logentries</strong> (<a href="https://docs.docker.com/config/containers/logging/logentries/">https://docs.docker.com/config/containers/logging/logentries/</a>)</td>
<td>Less to manage, SaaS based log aggregation and analytics. Supports TLS.</td>
</tr>
</tbody>
</table>

**Collecting Logs**

There's a few different ways to perform cluster-level logging with Docker Enterprise.

- At the node level using a logging driver
- Using a logging agent deployed either as a global service with Swarm or as a DaemonSet with Kubernetes
- Have applications themselves send logs to your logging infrastructure

**Node Level Logging**

To implement node level logging, simply create an entry in `/etc/docker/daemon.json` specifying your log driver on Linux machines. The default Docker daemon configuration file location is `%programdata%\docker\config\daemon.json` on Windows machines.

Logging at the node level can also be accomplished by using the default `json-file` or `journald` log driver and then using a logging agent to ship these logs.
**Note:** With no specific logging driver set in the `daemon.json`, be default the `json-file` log driver is used. By default this comes with no auto-rotate setting. To ensure your disk doesn't fill up with logs it is recommended to at least change to an auto-rotate configuration before installing Docker Enterprise, as seen here:

```json
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  }
}
```

Users of Docker Enterprise can make use of “dual logging”, which enables you to use the `docker logs` command for any logging driver. Please refer to the Docker documentation ([https://docs.docker.com/config/containers/logging/dual-logging/](https://docs.docker.com/config/containers/logging/dual-logging/)) for more details on this Docker Enterprise feature.

### Windows Logging

The ETW logging driver is supported for Windows. ETW stands for Event Tracing in Windows, and is the common framework for tracing applications in Windows. Each ETW event contains a message with both the log and its context information. A client can then create an ETW listener to listen to these events.

Alternatively, if Splunk is available in your organization, Splunk can be used to collect Windows container logs. In order for this to function properly the [HTTP Collector](https://docs.splunk.com/Documentation/Splunk/latest/Data/UsetheHTTPEventCollector) needs to be configured on the Splunk server side. Below is an example `daemon.json` for sending container logs to Splunk on Windows:

```json
{
  "data-root": "d:\docker",
  "labels": ["os=windows"],
  "log-driver": "splunk",
  "log-opts": {
    "splunk-token": "AAAAAAAA-BBBB-CCCC-DDDD-EEEEEEEEEEEE",
    "splunk-url": "https://splunk.example.com",
    "splunk-format": "json",
    "splunk-index": "main",
    "splunk-insecureSkipVerify": "true",
    "splunk-verify-connection": "false",
    "tag": "{{.ImageName}} | {{.Name}} | {{.ID}}"
  }
}
```

### Node Level Swarm Logging Example

To implement system-wide logging, creating an entry in `/etc/docker/daemon.json`. For example, use the following to enable the gelf output plugin:
And then restart the Docker daemon. All of the logging drivers can be configured in a similar way, by using the /etc/docker/daemon.json file. In the previous example using the gelf log driver, the tag field sets additional data that can be searched and indexed when logs are collected. Please refer to the documentation for each of the logging drivers to see what additional fields can be set from the log driver.

Setting logs using the /etc/docker/daemon.json file will set the default logging behavior on a per-node basis. This can be overwritten on a per-service or a per-container level. Overwriting the default logging behavior can be useful for troubleshooting so that the logs can be viewed in real-time.

If a service is created on a system where the daemon.json file is configured to use the gelf log driver, then all container logs running on that host will go to where the gelf-address config is set.

If a different logging driver is preferred, for instance to view a log stream from the stdout of the container, then it's possible to override the default logging behavior ad-hoc.

```
docker service create -
--log-driver json-file --log-opt max-size=10m
nginx:alpine
```

This can then be coupled with Docker service logs to more readily identify issues with the service.

### Docker Swarm Service Logs

docker service logs provides a multiplexed stream of logs when a service has multiple replica tasks. By entering in docker service logs <service_id>, the logs show the originating task name in the first column and then real-time logs of each replica in the right column. For example:

```
$ docker service create -d --name ping --replicas=3 alpine:latest ping 8.8.8.8
5x3enwyyr1re3hglu2nog540z

$ docker service logs ping
ping.2.n0bg40kksu08e@m00 | 64 bytes from 8.8.8.8: seq=43 ttl=43 time=24.791 ms
ping.3.pofxdol20p51@w01 | 64 bytes from 8.8.8.8: seq=44 ttl=43 time=34.161 ms
ping.1.o07dxxfx2ou2@w00 | 64 bytes from 8.8.8.8: seq=44 ttl=43 time=30.111 ms
ping.2.n0bg40kksu08e@m00 | 64 bytes from 8.8.8.8: seq=45 ttl=43 time=25.276 ms
ping.3.pofxdol20p51@w01 | 64 bytes from 8.8.8.8: seq=45 ttl=43 time=24.239 ms
ping.1.o07dxxfx2ou2@w00 | 64 bytes from 8.8.8.8: seq=45 ttl=43 time=26.403 ms
```

This command is useful when trying to view the log output of a service that contains multiple replicas. Viewing the logs in real time, streamed across multiple replicas allows for instant understanding and troubleshooting of service issues across the entire cluster.

### Deploying a Logging Agent

Many logging providers have their own logging agents. Please refer to their respective documentation for detailed instructions on using their respective tooling.
Generally speaking, those agents will either be deployed as a global [Swarm service](https://docs.docker.com/engine/reference/commandline/service_create/#set-service-mode---mode) or as a Kubernetes [DaemonSet](https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/).

**Brownfield Application Logs**

Sometimes, especially when dealing with brownfield (existing) applications not all logs will be written to `stdout`. In this case it can be useful to deploy a sidecar container to ensure that logs that are written to disk are also collected. Please refer to the Kubernetes documentation [here](https://kubernetes.io/docs/concepts/cluster-administration/logging/#using-a-sidecar-container-with-the-logging-agent) for an example of using fluentd with a sidecar container to collect these additional logs.

** Logging Infrastructure**

It's recommended that logging infrastructure be placed in an environment separate from where you deploy your applications. Troubleshooting cluster and application issues will become much more complicated when your logging infrastructure is unavailable. Creating a utility cluster to collect metrics and logs is a best practice with Docker Enterprise.

**Conclusion**

Docker provides many options when it comes to logging, and it's helpful to have a logging strategy when adopting the platform. For most systems, leaving the log data on the host isn't adequate. Being able to index, search, and have a self-service platform allows for a smoother experience for both operators and developers.