UNIX® AND LINUX® SYSTEM ADMINISTRATION HANDBOOK

FIFTH EDITION

EVI NEMETH • GARTH SNYDER • TRENT R. HEIN
BEN WHALEY • DAN MACKIN
with James Garnett, Fabrizio Branca, and Adrian Mouat

UNIX® AND LINUX® SYSTEM ADMINISTRATION HANDBOOK

FIFTH EDITION

logrotate is normally run out of **cron** once a day. Its standard configuration file is **/etc/logrotate.conf**, but multiple configuration files (or directories containing configuration files) can appear on **logrotate**'s command line.

This feature is used by Linux distributions, which define the **/etc/logrotate.d** directory as a standard place for **logrotate** config files. **logrotate**-aware software packages (there are many) can drop in log management instructions as part of their installation procedure, thus greatly simplifying administration.

The `delaycompress` option is worthy of further explanation. Some applications continue to write to the previous log file for a bit after it has been rotated. Use `delaycompress` to defer compression for one additional rotation cycle. This option results in three types of log files lying around: the active log file, the previously rotated but not yet compressed file, and compressed, rotated files.

In addition to **logrotate**, Ubuntu has a simpler program called **savelog** that manages rotation for individual files. It's more straightforward than **logrotate** and doesn't use (or need) a config file. Some packages prefer to use their own **savelog** configurations rather than **logrotate**.

newsyslog: log management on FreeBSD

The misleadingly named **newsyslog**—so named because it was originally intended to rotate files managed by syslog—is the FreeBSD equivalent of **logrotate**. Its syntax and implementation are entirely different from those of **logrotate,** but aside from its peculiar date formatting, the syntax of a **newsyslog** configuration is actually somewhat simpler.

The primary configuration file is **/etc/newsyslog.conf**. See **man newsyslog** for the format and syntax. The default **/etc/newsyslog.conf** has examples for standard log files.

Like **logrotate**, **newsyslog** runs from **cron**. In a vanilla FreeBSD configuration, **/etc/crontab** includes a line that runs **newsyslog** once per hour.

10.6 MANAGEMENT OF LOGS AT SCALE

It's one thing to capture log messages, store them on disk, and forward them to a central server. It's another thing entirely to handle logging data from hundreds or thousands of servers. The message volumes are simply too high to be managed effectively without tools designed to function at this scale. Fortunately, multiple commercial and open source tools are available to address this need.

The ELK stack

The clear leader in the open source space—and indeed, one of the better software suites we've had the pleasure of working with—is the formidable "ELK" stack consisting of Elasticsearch, Logstash, and Kibana. This combination of tools helps you sort, search, analyze, and visualize large volumes of log data generated by a global

network of logging clients. ELK is built by Elastic (elastic.co), which also offers training, support, and enterprise add-ons for ELK.

Elasticsearch is a scalable database and search engine with a RESTful API for querying data. It's written in Java. Elasticsearch installations can range from a single node that handles a low volume of data to several dozen nodes in a cluster that indexes many thousands of events each second. Searching and analyzing log data is one of the most popular applications for Elasticsearch.

If Elasticsearch is the hero of the ELK stack, Logstash is its sidekick and trusted partner. Logstash accepts data from many sources, including queueing systems such as RabbitMQ and AWS SQS. It can also read data directly from TCP or UDP sockets and from the traditional logging stalwart, syslog. Logstash can parse messages to add additional structured fields and can filter out unwanted or nonconformant data. Once messages have been ingested, Logstash can write them to a wide variety of destinations, including, of course, Elasticsearch.

You can send log entries to Logstash in a variety of ways. You can configure a syslog input for Logstash and use the rsyslog `omfwd` output module, as described in *Rsyslog configuration* on page 305. You can also use a dedicated log shipper. Elastic's own version is called Filebeat and can ship logs either to Logstash or directly to Elasticsearch.

The final ELK component, Kibana, is a graphical front end for Elasticsearch. It gives you a search interface through which to find the entries you need among all the data that has been indexed by Elasticsearch. Kibana can create graphs and visualizations that help to generate new insights about your applications. It's possible, for example, to plot log events on a map to see geographically what's happening with your systems. Other plug-ins add alerting and system monitoring interfaces.

Of course, ELK doesn't come without operational burden. Building a large scale ELK stack with a custom configuration is no simple task, and managing it takes time and expertise. Most administrators we know (present company included!) have accidentally lost data because of bugs in the software or operational errors. If you choose to deploy ELK, be aware that you're signing up for substantial administrative overhead.

We are aware of at least one service, logz.io, that offers production-grade ELK-as-a-service. You can send log messages from your network over encrypted channels to an endpoint that logz.io provides. There, the messages are ingested, indexed, and made available through Kibana. This is not a low-cost solution, but it's worth evaluating. As with many cloud services, you may find that it's ultimately more expensive to replicate the service locally.

Graylog

Graylog is the spunky underdog to ELK's pack leader. It resembles the ELK stack in several ways: it keeps data in Elasticsearch, and it can accept log messages either

directly or through Logstash, just as in the ELK stack. The real differentiator is the Graylog UI, which many users proclaim to be superior and easier to use.

See pages 85 and 580 for more information about RBAC and LDAP.

Some of the enterprise (read: paid) features of ELK are included in the Graylog open source product, including support for role-based access control and LDAP integration. Graylog is certainly worthy of inclusion in a bake-off when you're choosing a new logging infrastructure.

Logging as a service

Several commercial log management offerings are available. Splunk is the most mature and trusted; both hosted and on-premises versions are available. Some of the largest corporate networks rely on Splunk, not only as a log manager but also as a business analytics system. But if you choose Splunk, be prepared to pay dearly for the privilege.

Alternative SaaS options include Sumo Logic, Loggly, and Papertrail, all of which have native syslog integration and a reasonable search interface. If you use AWS, Amazon's CloudWatch Logs service can collect log data both from AWS services and from your own applications.

10.7 LOGGING POLICIES

Over the years, log management has emerged from the realm of system administration minutiae to become a formidable enterprise management challenge in its own right. IT standards, legislative edicts, and provisions for security-incident handling may all impose requirements on the handling of log data. A majority of sites will eventually need to adopt a holistic and structured approach to the management of this data.

Log data from a single system has a relatively inconsequential effect on storage, but a centralized event register that covers hundreds of servers and dozens of applications is a different story entirely. Thanks in large part to the mission-critical nature of web services, application and daemon logs have become as important as those generated by the operating system.

Keep these questions in mind when designing your logging strategy:

- How many systems and applications will be included?
- What type of storage infrastructure is available?
- How long must logs be retained?
- What types of events are important?

The answers to these questions depend on business requirements and on any applicable standards or regulations. For example, one standard from the Payment Card Industry Security Standards Council requires that logs be retained on easy-access media (e.g., a locally mounted hard disk) for three months and archived to long-term

storage for at least one year. The same standard also includes guidance about the types of data that must be included.

Of course, as one of our reviewers mentioned, you can't be subpoenaed for log data you do not possess. Some sites do not collect (or intentionally destroy) sensitive log data for this reason. You might or might not be able get away with this kind of approach, depending on the compliance requirements that apply to you.

However you answer the questions above, be sure to gather input from your information security and compliance departments if your organization has them.

For most applications, consider capturing at least the following information:

- Username or user ID
- Event success or failure
- Source address for network events
- Date and time (from an authoritative source, such as NTP)
- Sensitive data added, altered, or removed
- Event details

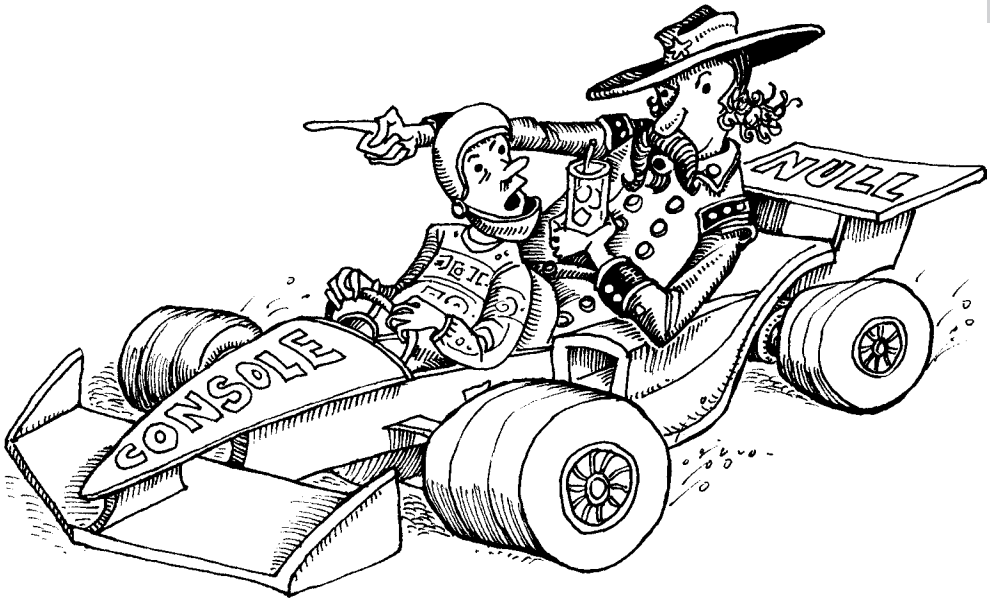
See page 753 for information about RAID.

A log server should have a carefully considered storage strategy. For example, a cloud-based system might offer immediate access to 90 days of data, with a year of older data being rolled over to an object storage service and three additional years being saved to an archival storage solution. Storage requirements evolve over time, so a successful implementation must adapt easily to changing conditions.

Limit shell access to centralized log servers to trusted system administrators and personnel involved in addressing compliance and security issues. These log warehouse systems have no real role in the organization's daily business beyond satisfying auditability requirements, so application administrators, end users, and the help desk have no business accessing them. Access to log files on the central servers should itself be logged.

Centralization takes work, and at smaller sites it may not represent a net benefit. We suggest twenty servers as a reasonable threshold for considering centralization. Below that size, just ensure that logs are rotated properly and are archived frequently enough to avoid filling up a disk. Include log files in a monitoring solution that alerts you if a log file stops growing.

11 Drivers and the Kernel



The kernel is the central government of a UNIX or Linux system. It's responsible for enforcing rules, sharing resources, and providing the core services that user processes rely on.

We don't usually think too much about what the kernel is doing. That's fortunate, because even a simple command such as **cat /etc/passwd** entails a complex series of underlying actions. If the system were an airliner, we'd want to think in terms of commands such as "increase altitude to 35,000 feet" rather than having to worry about the thousands of tiny internal steps that were needed to manage the airplane's control surfaces.

The kernel hides the details of the system's hardware underneath an abstract, high-level interface. It's akin to an API for application programmers: a well-defined interface that provides useful facilities for interacting with the system. This interface provides five basic features:

- Management and abstraction of hardware devices
- Processes and threads (and ways to communicate among them)
- Management of memory (virtual memory and memory-space protection)
- I/O facilities (filesystems, network interfaces, serial interfaces, etc.)
- Housekeeping functions (startup, shutdown, timers, multitasking, etc.)

Only device drivers are aware of the specific capabilities and communication protocols of the system's hardware. User programs and the rest of the kernel are largely independent of that knowledge. For example, a filesystem on disk is very different from a network filesystem, but the kernel's VFS layer makes them look the same to user processes and to other parts of the kernel. You don't need to know whether the data you're writing is headed to block 3,829 of disk device #8 or whether it's headed for Ethernet interface e1000e wrapped in a TCP packet. All you need to know is that it will go to the file descriptor you specified.

Processes (and threads, their lightweight cousins) are the mechanisms through which the kernel implements CPU time sharing and memory protection. The kernel fluidly switches among the system's processes, giving each runnable thread a small slice of time in which to get work done. The kernel prevents processes from reading and writing each other's memory spaces unless they have explicit permission to do so.

The memory management system defines an address space for each process and creates the illusion that the process owns an essentially unlimited region of contiguous memory. In reality, different processes' memory pages are jumbled together in the system's physical memory. Only the kernel's bookkeeping and memory protection schemes keep them sorted out.

Layered on top of the hardware device drivers, but below most other parts of the kernel, are the I/O facilities. These consist of filesystem services, the networking subsystem, and various other services that are used to get data into and out from the system.

11.1 Kernel chores for system administrators

Nearly all of the kernel's multilayered functionality is written in C, with a few dabs of assembly language code thrown in to give access to CPU features that are not accessible through C compiler directives (e.g., the atomic read-modify-write instructions defined by many CPUs). Fortunately, you can be a perfectly effective system administrator without being a C programmer and without ever touching kernel code.

That said, it's inevitable that at some point you'll need to make some tweaks. These can take several forms.

Many of the kernel's behaviors (such as network-packet forwarding) are controlled or influenced by tuning parameters that are accessible from user space. Setting these values appropriately for your environment and workload is a common administrative task.

Another common kernel-related task is the installation of new device drivers. New models and types of hardware (video cards, wireless devices, specialized audio cards, etc.) appear on the market constantly, and vendor-distributed kernels aren't always equipped to take advantage of them.