LOGGLY

The JournalD Reference Guide

by Jorgen Schäfer

Introduction

With the advent of systemd, journald has arrived. Most systems run this program, but not everyone is fully aware of what journald is, which problems it solves, and why it's valuable. After reading this white paper, both system administrators and programmers should know:

- **What journald is**
- **Why there is some controversy around it**
- **What it means for systems and programs running on it and what to expect from it**

What is journald?

Journald is a daemon included in the systemd distribution to handle log data. It is tightly integrated with systemd and provides an interface that is compatible with syslog, the standard logging interface on UNIX.

The tight integration with systemd makes it possible for all log messages to be handled with a single interface. Traditional syslog servers are started much like other services during the boot process and can miss log messages emitted during early boot. By integrating journald with systemd, even the earliest boot process messages are available to journald.

Another problem with traditional syslog processes is that log messages are treated as mostly unstructured text fields. Journald adds a key-value concept that is used, for example, to add audited and verified fields on which application a log message originates from, something that is easily faked in the traditional syslog protocol.

Finally, syslog servers rely on plain text files for storing log messages. While this is a simple approach, it also has drawbacks. Plain text files need to be "rotated" when they become too large: that is, renamed, compressed, archived, and eventually deleted. Also, text files do not offer any structure to improve searching, making it slow to find log messages, such as from specific services. Journald replaces the plain text files with a custom file format that addresses these problems.

The file format controversy

This move to a custom file format has been met with strong criticism. Log files created with syslog are simple plain text files that can be easily read and analyzed with standard UNIX tools. Contrary to this, the journald format requires a specific tool, `journalctl`, to read its files.

There is a fear that the binary format can lead to all sorts of issues. For example, with binary files, it is much easier for a programming error to lead to large amounts of corrupted and lost data. Data recovery with plain text files is much simpler.

These fears are fostered by the decision of the journald authors not to standardize the file format but to make it explicitly subject to change. This gives rise to the fear that tools that might work for analyzing today might not work tomorrow—or worse, that log files written today might not be readable anymore tomorrow.

Given these fears, some people argue that the drawbacks of traditional syslog servers are manageable and getting rid of them is problematic.

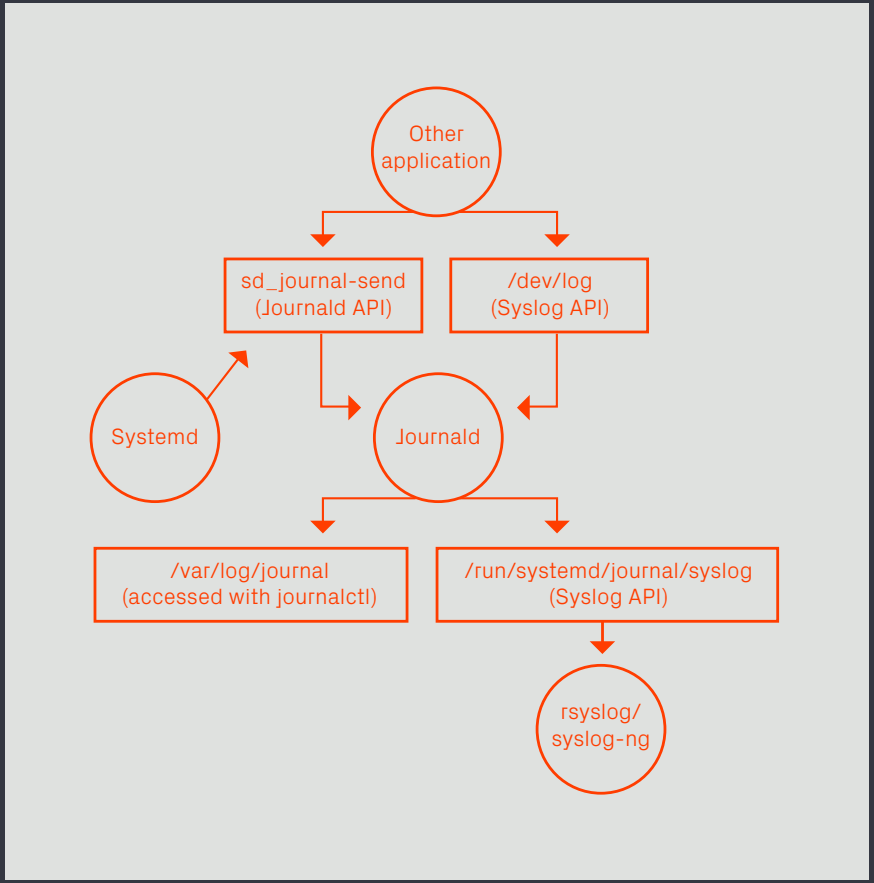
Preserving syslog compatibility

Journald follows a two-way syslog compatibility approach.

First of all, journald provides the standard syslog message API using the /dev/log endpoint. This means that applications that use the syslog protocol to write their log messages work exactly the same when journald is installed, with or without a traditional syslog daemon.

At the same time, journald also has the ability to forward all messages it receives to a traditional syslog daemon using the same protocol but a different endpoint. This means that it is possible to use syslog daemons alongside journald transparently, and use the features of both at the same time. And this setup is indeed the case for almost all GNU/Linux distributions. While most of them (excluding Gentoo and Slackware) have switched to systemd by default and enable journald, only Fedora does not install a traditional syslog by default next to journald.

So on most distributions, syslog works as it has before both for applications and for administrators, thanks to the full compatibility layer provided by journald. Additionally, very few programs target journald specifically. Most use the



traditional syslog API without any of the additional features of structured logging provided by journald.

For those who want even less of an impact by journald, it can also be configured to store its journal log data only transiently or even not at all. This means that journald only acts as a logging gateway to syslog for systemd, and all log data is available only through syslog. As systemd explicitly and exclusively uses journald for its own log messages, it's not possible to remove it entirely, though.

Pros and cons of custom file formats

Pros

- Easier and faster to search for relevant information
- Structured information easier to analyze and aggregate by machines
- Structured information allows data with different authenticity
- Multi-line and binary data possible
- No extra services required to limit disk space usage

Cons

- Raw output harder for humans to read—requires a specific tool
- Data corruption more likely—data recovery more difficult
- Future changes to standards could make current logs more difficult to read

Fears mostly unfounded

The danger of file corruption is addressed directly by journald. Files are almost exclusively append-only. Journald also checks whether it can parse a file on every write, and as soon as it detects a problem, whether it's corruption due to a programming error or even hardware problems, journald automatically rotates the file and starts a new one. This significantly limits the amount of potential damage due to corruption.

While the format, in theory, could change a lot between versions, so far all changes have been backwards-compatible. So parsing logs from a different system is quite possible even on a system with older software, and certainly on a system with newer versions installed.

Administrators who seriously worry about data loss due to corruption should not rely on text versus binary file formats, but rather ship their logs to a redundant centralized logging system. For reading logs directly on a system, journald is an excellent choice.

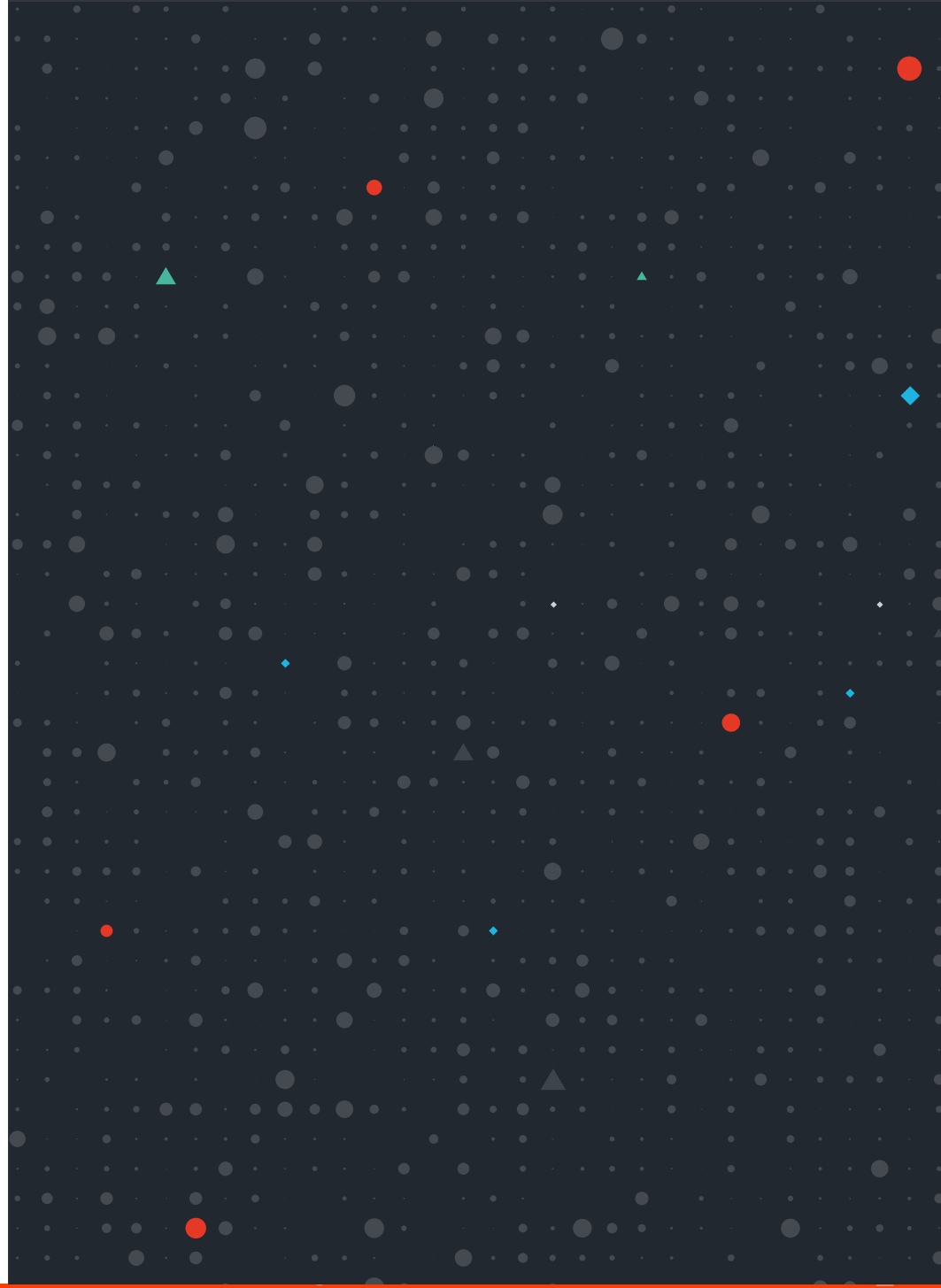
How journald works with Loggly

While journald has some support for centralized logging, it is quite rudimentary. Sending logs through standard syslog channels is robust and supported well by different log aggregators.

As journald can send all messages to syslog, and does so by default, this means that standard setups, for example with Loggly, simply work.

Using syslog as the transport layer means that some of the features of journald are lost. Most notably, structured fields and especially the audited log information cannot be transferred through syslog. This is unfortunate, but until journald gets robust direct support for centralized logging systems, it's the best option. Of course, the advantages of the file format are also irrelevant when log messages are sent to a central system.

Not all advantages are lost, though. Journald still receives messages from earlier in the boot process than any syslog daemon could, and these are also forwarded to the centralized logging system.



Summary

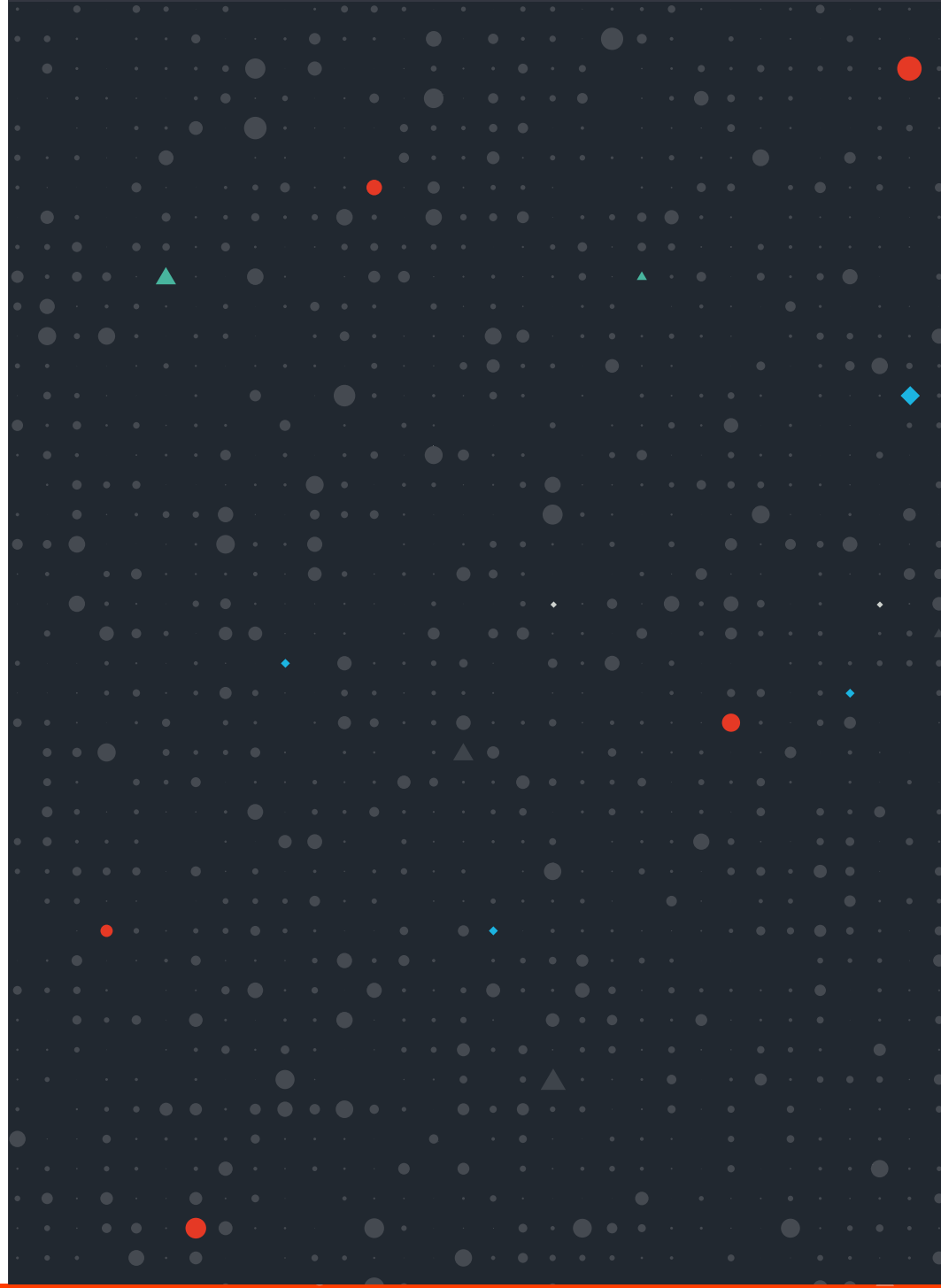
So what does journald mean to administrators and programmers?

For administrators, this is quite simple. Journald is here, and it is here to stay. It's part of and required by systemd, and almost all GNU/Linux distributions have converted to systemd. Chances are, you are already using journald.

When working on a system directly, the journald log file format provides some tremendous advantages. The utility of being able to quickly find log messages associated with a specific service alone cannot be overemphasized. Seeing the last few log messages when checking the status of a service, as is the default with systemd, is exceptionally useful.

So the question is not so much whether you should use journald, but whether you still need a syslog daemon. Unless you are using Fedora, your distribution will by default install a syslog daemon for you, so you have the customary log files where they have always been. While it can be useful to get used to `journalctl` to read logs, it's by no means mandatory. And if you use centralized logging for your systems, and there are very few reasons why you should not, syslog daemons still serve as important a role as the transport layer.

For programmers, the question is a bit more nuanced. Journald, even without a syslog daemon installed, provides the same API for sending log messages as syslog daemons do. For maximum compatibility and also ease of use, sticking with that is the simplest approach. Journald provides its own API with extended capabilities, in particular the ability to emit structured log messages and even include binary data, so when this is useful to the program, simply using the journald API is preferable to trying to implement something yourself. Journald handles all the compatibility with syslog where necessary. But until journald addresses centralized logging more thoroughly, there is no need to go out of your way to support it directly.



About Loggly

Loggly is the world's most popular cloud-based, enterprise-class log management service, serving more than 10,000 customers including one-third of the Fortune 500. The Loggly service integrates into the engineering processes of teams employing continuous deployment and DevOps practices to reduce MTTR, improve service quality, accelerate innovation, and make better use of valuable development resources. We offer an alternative to traditional, search-based log analysis by structuring and summarizing your log data before you ask it to. With Loggly, your logs reveal what matters through real-time metrics and dashboards. Founded in 2009 and based in San Francisco, the company is backed by True Ventures, Matrix Partners, Cisco, Trinity Ventures, Harmony Partners, Data Collective Venture Capital, and others. Loggly is an AWS Advanced Technology Partner and a Docker Ecosystem Technology Partner. Visit us at www.loggly.com or follow @loggly on Twitter.

START FREE TRIAL