20
Cloud Native Development, Containers, and Open Source Licensing

Richard Fontana

20.1 Overview of Linux Containers

A container is a Linux operating system feature that enables isolation of a user-space instance from the rest of the system. Containers provide a lightweight form of virtualisation in comparison to the hardware virtualisation enabled by the use of hypervisor technology.[1] The Docker project popularised an easy way of building and sharing containerised applications.[2] Under this approach, a container image format, featuring a set of immutable filesystem layers, is used for packaging and distributing software intended to be run in containers. Container images are delivered through a network service known as a registry. A container image may store hundreds of applications, utilities, and libraries, and typically includes a 'base layer' consisting of a stripped-down Linux distribution without the kernel. Various aspects of container technology are now undergoing standardisation by the Open Container Initiative.[3]

[1] 'OS-level virtualization', <https://en.wikipedia.org/wiki/OS-level_virtualization> accessed 2 August 2020.

[2] The Docker community project, launched in 2013, was partially rebranded as 'Moby' in 2017. See 'Moby Project', <https://github.com/moby/moby> accessed 2 August 2020. There are now alternative Open Source implementations of Docker-style container technology, such as the buildah, podman, and skopeo projects maintained by Red Hat. See 'Say "Hello" to Buildah, Podman, and Skopeo', <https://servicesblog.redhat.com/2019/10/09/say-hello-to-buildah-podman-and-skopeo/> accessed 2 August 2020.

[3] 'Open Container Initiative', <https://opencontainers.org/> accessed 2 August 2020.

Richard Fontana, *Cloud Native Development, Containers, and Open Source Licensing* In: *Open Source Law, Policy and Practice.* Edited by: Amanda Brock, Oxford University Press. DOI: 10.1093/oso/9780198862345.003.0020

Enterprise cloud computing has become closely associated with a so-called cloud native approach to building and running scalable applications in public, private, and hybrid clouds. Cloud native development centres around the use of container technology along with the adoption of DevOps practices. Cloud native applications are structured as a set of containerised microservices, each of which focuses on performing one service.[4] Container orchestration tools, the most widely adopted of which has been the Kubernetes project, are used to dynamically coordinate the multiple containers making up an application.[5]

Containers typically include a large amount of Open Source software, for the most part ultimately derived from upstream community projects, even in cases where the container is focused on providing a proprietary service. This is so for two overlapping reasons. First, much of the software included in a container image consists of packages maintained by the underlying Linux distribution the container is based on (including the packages in the base layer). Second, containerised applications are no different from contemporary non-containerised software in that application runtime environments consist largely of Open Source-licensed dependencies. The licence makeup of the open source part of a container application runtime invariably includes both copyleft and permissive (non-copyleft) licences. Copyleft licences in the General Public Licence (GPL) family, primarily the GNU GPL version 2 (GPLv2),[6] GNU GPL version 3 (GPLv3),[7] and GNU Library or Lesser GPL versions 2.0 and 2.1 (LGPLv2.0[8] and LGPLv2.1[9]), are particularly prevalent in the subset of the container runtime that consists of Linux distribution packages. This reflects the licence composition of the most fundamental user-space packages in mainstream Linux server distributions, which are largely GPL-licensed.[10]

Given the abundance of Open Source software in containers, issues of Open Source licence interpretation and compliance naturally arise in the container setting. One of the characteristics of the rapid rise of container technology adoption has been a perceived inattention to Open Source licence compliance by many

[4] 'CNCF Cloud Native Definition v1.0', <https://github.com/cncf/toc/blob/master/DEFINITION.md> accessed 2 August 2020.

[5] 'Kubernetes', <https://kubernetes.io/> accessed 2 August 2020.

[6] <https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html> accessed 18 April 2022.

[7] <https://www.gnu.org/licenses/gpl-3.0.html> accessed 18 April 2022.

[8] <https://www.gnu.org/licenses/old-licenses/lgpl-2.0.en.html> accessed 18 April 2022.

[9] <https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html> accessed 18 April 2022.

[10] See, e.g., Red Hat Universal Base Image 8, <https://catalog.redhat.com/software/containers/ubi8/ubi/5c359854d70cc534b3a3784e?container-tabs=packages> accessed 18 April 2022. For a good historically oriented discussion of containers, see Daniel Riek, 'A Greybeard's Worst Nightmare: How Kubernetes and Containers are re-defining the Linux OS', <https://www.redhat.com/files/summit/session-assets/2017/S104999-graybeard-rhsummit2017.pdf> accessed 18 April 2022.

container users.[11] To the extent this is a fair assessment, it may reflect the ease with which a user of container technology can assemble and distribute complex, Open Source-based software collections without awareness of their contents or associated licensing terms. At the same time, because containers may be new and unfamiliar to lawyers, they may be overly concerned about the impact of Open Source licence obligations arising out of the use of containers.[12]

20.2 Containers and the Scope of Copyleft

The 'scope of copyleft' issue in Open Source licensing centres on interpretation of the GPL (see an overview of licensing and copyright at Chapter 3). With the increasing adoption and awareness of containers, some have raised concerns that in the container setting the GPL's copyleft is broader in scope than it would be in counterpart non-containerised scenarios.[13] This section provides some background on the issue and explains that containers do not warrant any alteration of the general interpretation of GPL copyleft.

Three broad technical observations may be useful for understanding the GPL copyleft scope issue. First, software is typically developed through the composition of smaller, modular components. The additional components needed by an application at runtime are termed the runtime dependencies of the application. With the rise of Open Source, a growing portion of an application's runtime dependencies are drawn from Open Source project codebases, typically as packaged by a major Linux distribution or in a standard package repository for a particular programing language community.[14] This characteristic of modern software development is particularly evident in containers, since containers isolate an application along with its user-space stack.

Second, an application's dependencies may be divided into two categories, library dependencies and system dependencies,[15] though the distinction is not

[11] See Jake Edge, 'An update on compliance in containers' (16 April 2019), <https://lwn.net/Articles/786066/> accessed 18 April 2022; Armijn Hemel, 'Docker containers for legal professionals', <https://www.linuxfoundation.org/publications/2020/04/docker-containers-for-legal-professionals/> accessed 18 April 2022.

[12] One common use case for containers involves mere internal consumption, which should not justify concerns about Open Source licence compliance, given that, for all practical purposes, the obligations of Open Source licences are triggered by distribution.

[13] See Richard Fontana, 'Containers, the GPL, and copyleft: No reason for concern', 24 January 2018, <https://opensource.com/article/18/1/containers-gpl-and-copyleft>; Dirk Riehle, 'The GNU Public License v2 in the land of microservices', 17 June 2020, <https://dirkriehle.com/2020/06/17/the-gnu-public-license-v2-in-the-land-of-microservices/> both accessed 8 August 2020.

[14] Some important examples of the latter type of package repository are Maven Central, <https://repo1.maven.org/maven2/> (Java); PyPI, <https://pypi.org/> (Python); and npm, <https://www.npmjs.com/> (Node.js), all accessed 18 April 2022.

[15] This terminology is not widely used but captures a useful distinction. It was suggested to the author by Van Lindberg in a mailing list discussion.

always clear-cut. Libraries are collections of modular, reusable software functions that are called by a program to access their functionality.[16] The libraries used by an application are typically written in the same programing language as the application, and the application and the libraries it invokes will typically run in the same operating system process (i.e. an executing instance of a program). An application's system dependencies will generally be run in separate processes and often involve a different programing language runtime.

Third, as a consequence of various technical and commercial developments in programing languages and operating systems over the past few decades, software development can be said to have become increasingly 'loosely coupled' in style. The earliest GPL-licensed programs in the late 1980s were typically written in C, a relatively low-level systems language, and compiled programs typically consisted of a single executable object code file including all libraries statically linked into the application.

Today, lower-level user-space programs (those that interact more directly with real or virtual hardware) more typically use dynamically linked libraries shared among processes, and enterprise application development has increasingly favoured higher-level programing languages (such as Java, JavaScript, and Python), which rely on a virtual machine or interpreter and dynamic loading of library modules.[17]

The GPL has a number of interrelated requirements associated with the term *copyleft*, which may be summarised as follows:

1. Derivative works of a GPL-licensed work, if distributed, must be licensed 'as a whole' under the GPL.[18]
2. Binary versions of a GPL-licensed work, if distributed, must be accompanied by complete corresponding source code in one of a number of specified ways.[19]

[16] For a good judicial discussion of the related topic of APIs, see *Oracle America, Inc. v Google, Inc.*, 872 F. Supp. 2d 974 (N.D. Cal. 2012), *rev'd*, 750 F.3d 1339 (Fed. Cir. 2014), *cert. denied*, 135 S. Ct. 2887 (2015).

[17] See John R Levine, *Linkers and Loaders* (San Francisco: Morgan Kaufmann Publishers 2000); see also Mark Radcliffe, 'Top 10 FOSS issues of 2012', 11 January 2013, <https://opensource.com/law/13/1/top-ten-foss-2012> accessed 8 August 2020, (noting 'the rise of "loosely coupled" programming techniques').

[18] GPLv2 § 2; GPLv3 § 5. Note that, unlike GPLv2, GPLv3 does not use the US copyright statutory term 'derivative work', instead relying on a definition of 'modified version' that references permissions under background copyright law. GPLv3 § 0 ('To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a 'modified version' of the earlier work or a work "based on" the earlier work.'). Despite the change in terminology in GPLv3, informal discussions of GPL copyleft interpretation have generally continued to refer to derivative works even in contexts where GPLv3 is the relevant licence. GPLv3 is not intended to expand or narrow the general scope of copyleft as it was interpreted under GPLv2, though in certain specific ways it relaxes the strictures of GPLv2 copyleft (particularly with respect to compatibility of other licences with the GPL).

[19] GPLv2 § 3; GPLv3 § 6. Compliance with this requirement in the container setting is discussed in the following section.

3. Distributors may not impose 'further restrictions' on recipients' exercise of permissions under the GPL.[20]

As to the derivative work requirement, the copyleft provisions of GPLv2 more precisely use the term 'work based on the Program', which in turn is defined by reference to derivative works. The term 'the Program' refers to the work originally licensed under the GPL. GPLv2 defines 'work based on the Program' as 'either the Program or any derivative work under copyright law; that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language'.[21] Some lawyers have seen this as a sweeping partial redefinition of the statutory term 'derivative work' (see also Chapter 3). However, GPLv2 also has a 'mere aggregation' clause that makes clear that the scope of copyleft, and in particular the breadth of what can be considered a 'work based on the Program', is limited: '[M]ere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.'[22]

From the earliest years of the GPL in the late 1980s, there has been a debate in the technical community over the interpretation of 'derivative work' as used in GPLv2. A preoccupation with the derivative work issue later developed among lawyers as they became more familiar with Open Source licensing. A thorough exploration of this topic is beyond the scope of this chapter but is dealt with in Chapter 3. It is sufficient to note that there is a long-standing view, widely held among GPL licensors and Open Source community members generally, that under the GPL derivative works encompass notional combinations of a GPL-licensed work with other software having a library dependency relationship with the GPL-licensed software.

The view that a derivative work of GPL-licensed software extends across (library) dependency boundaries in at least some circumstances is largely the result of the influence of the Free Software Foundation (FSF), the drafter and

[20] GPLv2 § 6; GPLv3 § 10.

[21] GPLv2 § 0. See also GPLv2, final paragraph ('This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.').

[22] GPLv2 § 2, last para. The more complex counterpart provision of GPLv3 states:

> 'A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.'

GPLv3 § 5, last para. Cf. OSD para 9, <https://opensource.org/osd/> ('The licence must not place restrictions on other software that is distributed along with the licensed software') accessed 21 July 2022.

maintainer of the GPL, on community interpretation of the GPL, particularly given the general absence of case law interpreting the licence as well as the lack of helpful case law on the meaning of 'derivative work' in the software context.[23] It is likely that the FSF's views on the policy issue—how far should the GPL copyleft requirements extend to downstream combinations of GPL and non-GPL components—were influenced by the shift towards more 'loosely coupled' programing following the release of GPLv2 in 1991. In some of its interpretive materials, the FSF has associated derivative work scope with the notion of a 'single program'. Library dependencies, under this view, were historically statically linked into the same executable file as the rest of the program, and the GPL should not be interpreted to have a weaker copyleft scope in a 'loosely coupled' counterpart program.[24]

On the other hand, the FSF-influenced copyleft interpretation would generally not view system dependencies as falling within GPL copyleft scope. Rather, an application and its system dependencies would be 'mere aggregation'. The FSF provides a technical rationale:

> Where's the line between two separate programs, and one program with two parts? This is a legal question, which ultimately judges will decide. We believe that a proper criterion depends both on the mechanism of communication (exec, pipes, rpc, function calls within a shared address space, etc.) and the semantics of the communication (what kinds of information are interchanged).
>
> If the modules are included in the same executable file, they are definitely combined in one program. If modules are designed to run linked together in a shared address space, that almost surely means combining them into one program.
>
> By contrast, pipes, sockets and command-line arguments are communication mechanisms normally used between two separate programs. So when they are used for communication, the modules normally are separate programs. But if the semantics of the communication are intimate enough, exchanging complex internal data structures, that too could be a basis to consider the two parts as combined into a larger program.[25]

Using examples familiar to technical Unix and Linux users, this explanation relies on the significance of the process boundary in distinguishing 'mere aggregation' from derivative work.

[23] See Dan Ravicher, 'Software Derivative Work: A Jurisdiction Dependent Determination', 13 November 2002, <https://www.linux.com/news/software-derivative-work-jurisdiction-dependent-determination/> accessed 8 August 2020.

[24] Cf. GPLv3 § 5, last para (definition of 'Aggregate' suggests that a work based on the Program does not include a compilation of works 'which are not combined … such as to form a larger program').

[25] 'Frequently Asked Questions about the GNU Licenses', <https://www.gnu.org/licenses/gpl-faq.en.html> accessed 9 August 2020.

The limited scope of GPL copyleft is consistent with long-established practice and assumptions among distributors and users of traditional (non-containerised) Linux. For example, there has never been a serious contention that the GPL extends to the entirety of Linux user-space merely because many user-space packages are licensed under the GPL. Indeed, many packages in Linux distributions are under GPL-incompatible licences. Moreover, proprietary software is commonly distributed for execution on, along with, or as part of Linux distribution products. Linux-based virtual machine images, which are somewhat analogous to container images, often include proprietary software in addition to GPL- licensed and GPL-incompatible Open Source software.

We can now address the specific topic of copyleft scope in the container context. There is anecdotal evidence of a belief that all the code in a container, and perhaps even all the code in a multi-container implementation of an application, necessarily becomes subject to the GPL's copyleft requirements because the container causes it all to be a derivative work of one or more of the container's GPL-licensed components. Such views are likely to reflect a fear of unfamiliar technology, along with a misimpression caused by the use of the term 'container'. To those not technically knowledgeable about containers, the term might seem to suggest a container is analogous to a modular package or a 'single program', to use the FSF's phrasing. However, containers are so called because of the isolation they achieve, and not because they 'contain' anything.

A company concerned about the potential effects of distributing GPL-licensed code along with its own proprietary software might attempt to prohibit its developers from adding any third-party GPL code to a container image it plans to push to a registry or distribute by way of a registry. To the extent the goal is to avoid distributing GPL-licensed code, this is a dubious strategy. As noted earlier, the base layer of any normal container image will include many GPLv2- and GPLv3-licensed packages. While the company might be able to prevent its developers from including GPL-licensed software in the container image layers they create, it generally cannot guarantee that when it pushes a container image to a registry it will never push the base layer or other GPL-code-containing third-party-created layers that its container image is building on.

In any case, the concern about having both proprietary software and GPL-licensed software running in the same container, or distributed in the same container image, is misplaced, because the two components are no more likely to have a copyleft-significant relationship in the containerised case than in the well-understood non-containerised case. In general, unless there are particular facts suggesting that the two components are not 'separate and independent', the simultaneous inclusion of a proprietary component and a GPL-licensed component in a container image will be 'mere aggregation', compliant with and contemplated by the GPL, with no impact on the licensing of either component. While in a given situation the relationship between two components

may not be 'mere aggregation', the same is true of software running in non-containerised user-space on Linux, or any other mainstream operating system. That is, there is nothing in the technical makeup of containers or container images that suggests a need to apply a different, expanded form of copyleft scope analysis.

It follows that when examining the relationship between code running in a container and code running on the same Linux system outside a container, the 'separate and independent' criterion suggested by the GPL text and FSF interpretive guidance is almost certainly met. The two components in this case will necessarily run as separate processes, and the whole technical point of using containers is isolation from other software running on the system.

A more practical scenario would involve separate but potentially interacting containers, as part of a containerised application with a microservices architecture. Each container will undoubtedly have GPLv2 and GPLv3-licensed software, in addition to code under various other Open Source licences. Suppose one of the interacting containers also has some proprietary code. In the absence of very unusual facts, copyleft scope should be expected not to extend across multiple containers. Separate containers run in separate processes. Communication between containers by way of network interfaces is analogous to the operating system mechanisms cited by the FSF as examples of mere aggregation (e.g. pipes and sockets). Moreover, a multi-container microservices scenario would seem to preclude what the FSF calls 'intimate' communication, by definition. While the composition of an application using multiple containers may not be dispositive of the GPL scope issue, it makes the technical boundaries between the components more apparent and provides a strong basis for arguing separateness. Here too there is no technical characteristic of containers or container images that suggests application of a different and stricter approach to copyleft scope analysis.

The preceding discussion suggests that a company concerned about the impact of GPL code distribution on simultaneously distributed proprietary code might wish to embrace containerisation as a strategy for minimising copyleft scope concerns, by isolating GPL and proprietary code from one another. This may not always be practical (e.g. if the GPL code is a system dependency of the proprietary code, it may be necessary for it to run in the same container). It would be more appropriate for such basic architectural decisions to be driven by technical considerations, rather than often-unfounded or overblown legal concerns about the impact of the GPL that fail to account for mere aggregation. While in a non-containerised setting the relationship between two interacting components will often be mere aggregation, the evidence of separateness that containers provide may be comforting to those who worry (however needlessly) about GPL copyleft scope.

20.3 Container Images and Source Code Compliance

Licences classified as copyleft typically impose some requirement on distributors of binaries to provide source code. As noted earlier, this is true of the GPL with its relatively elaborate 'complete corresponding source code' requirement.[26] The source code disclosure requirements in other, less commonly used copyleft licences are generally worded in a less detailed manner and are often assumed to be generally satisfied through attention to GPL-style source compliance, although it is not clear that this is entirely correct.[27] In contrast to source code requirements, nearly all Open Source licences include requirements of various sorts to preserve legal notices such as copyright statements, licence notices, licence texts, and in some cases author attributions.[28]

Making source code available simultaneously with distribution of binaries is an efficient way of complying with both types of basic Open Source licence requirements, since source code will generally contain all legal notices for which licences mandate preservation, though this does not appear to be widely recognised by vendors engaged in Open Source licence compliance efforts.[29] Distribution of container images, for example distribution by way of a registry, will, of course, give rise to all the distribution-triggered Open Source licence requirements that would be triggered in a non-containerised setting.

GPLv2 section 3 gives two options for satisfying the corresponding source code requirement in the case of commercial distribution of object code:[30]

[26] GPLv2 defines complete corresponding source code: 'For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable.' GPLv2 § 3. GPLv3 defines the term 'Corresponding Source':

> 'The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities …. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.' GPLv3 § 1.

[27] See, e.g., Eclipse Public Licence 2.0 § 3.1 (if distributed in object code, 'the Program must also be made available as Source Code …, and the Contributor must accompany the Program with a statement that the Source Code for the Program is available under this Agreement, and informs Recipients how to obtain it in a reasonable manner on or through a medium customarily used for software exchange'); Mozilla Public Licence 2.0 § 3.2 ('If You distribute Covered Software in Executable Form then … such Covered Software must also be made available in Source Code Form …, and You must inform recipients of the Executable Form how they can obtain a copy of such Source Code Form by reasonable means in a timely manner, at a charge no more than the cost of distribution to the recipient').

[28] See, e.g., Apache Licence 2.0 § 4.

[29] Jeffrey Robert Kaufman, 'An economically efficient model for Open Source software license compliance', 1 September 2017, <https://opensource.com/article/17/9/economically-efficient-model>; Heather Meeker, 'Is the container half empty or half full?', <https://heathermeeker.com/2020/07/29/is-the-container-half-empty-or-half-full/> both accessed 9 August 2020.

[30] For non-commercial distribution, GPLv2 also allows the distributor to '[a]ccompany [the object code] with the information you received as to the offer to distribute corresponding source code'.

a) Accompany [the object code] with the complete corresponding machine-readable source code, which must be distributed … on a medium customarily used for software interchange; or,
b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed … on a medium customarily used for software interchange ….

In addition, the last paragraph of section 3 states:

> If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

Though it is not immediately obvious from the licence text, this is generally understood as a clarification that the first option (simultaneous source availability) is available when object code is distributed over a computer network, which of course today is the normal manner in which object code is commercially distributed. It should be remembered that when this language was added to GPLv2 in 1991, it was still common for free software to be distributed using physical tape media, and large-scale network distribution of software was still in its infancy.[31]

The source availability options in GPLv3 for commercial distribution of object code are similar, though with some notable changes:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost

GPLv2 § 3(c). GPLv3 has substantially the same option but limits it to being used only 'occasionally'. GPLv3 § 6(c).

[31] Cf. GPLv1 (1989) § 3 (containing no counterpart to the 'equivalent access' clause of GPLv2).

of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

…

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.[32]

Under GPLv3 the written offer option appears no longer to be available when distribution ('conveying') of object code is not done by way of a hardware device or physical storage medium. The simultaneous-source option, which now explicitly includes the network binary distribution case, is liberalised somewhat to codify FSF interpretation of GPLv2.[33]

The written source offer option is a possible form of source compliance for network binary distribution under GPLv2, and is one commonly chosen by vendors in that context, perhaps because it can seemingly provide a quick solution for last-minute GPL compliance problems. However, written offers have a number of drawbacks.[34]

A benefit of the simultaneous source availability approach is that compliance is fully achieved at the time of object code distribution, whereas with the written offer option there is an ongoing and somewhat unpredictable source compliance obligation based on the required term of the offer. Moreover, in addition to its narrowed availability under GPLv3, it is not clear that the written offer approach would comply with the source availability requirements of some non-GPL copyleft licences, nor would it seem to take care of notice preservation compliance adequately under Open Source licences generally. For these reasons, the written offer

[32] GPLv3 § 6.

[33] See 'Frequently Asked Questions about version 2 of the GNU GPL', <https://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.en.html> accessed 9 August 2020.

[34] See Eben Moglen and Mishi Choudhary, *Software Freedom Law Center Guide to GPL Compliance*, 2nd edn, (2014) 52–54, available at <http://softwarefreedom.org/resources/2014/SFLC-Guide_to_GPL_Compliance_2d_ed.pdf> accessed 9 August 2020; *Copyleft and the GNU General Public License: A Comprehensive Tutorial and Guide*, part II, § 15.1.2, available at <https://copyleft.org/guide/comprehensive-gpl-guidech16.html#x21-13100015> accessed 9 August 2020.

option should be avoided in the network binary distribution context, including distribution of container images, other than perhaps as a 'belt and braces' mechanism. An additional problem with the written offer approach in the container setting is that there is no straightforward way to present the offer to the recipient of the container image.

Because containers invariably include a large amount of Open Source-licensed software, distribution of container images gives rise to the same source availability requirements that would be triggered in a corresponding non-containerised distribution setting. However, in some important respects container images differ from non-containerised software, in ways that may make Open Source licence compliance more challenging.

In the past, prior to the rise of containers, modular software components were commonly obtained separately. For example, a Linux user seeking to install some additional packaged software (corresponding generally to a particular upstream Open Source project) would install just that package along with any missing dependencies; the resulting distribution event would typically involve a relatively small number of packages. In contrast, even though containers are often focused on a particular application or service, a container image will include a heterogeneous collection of software—often hundreds of components, more analogous to the installation of an entire (non-containerised) working Linux distribution. This represents a shift in the focus of software delivery from one driven by how the software is built to one driven by how the software is used.[35] A typical creator or consumer of a container image will not be aware of the scale and complexity of the contents of the image, including with respect to licensing.[36]

In the shift to containers, one compliance-related challenge concerns the diminished role of package management. Prior to the rise of containers, package maintainers and package management tools played a key role in facilitating source availability and thus Open Source licence compliance. The focused nature of a package, the role of a package maintainer in Linux distributions and upstream language-specific package repositories, and the tooling that has been built to support package management systems has resulted in an expectation that the package maintainer will take responsibility for ensuring that source code is available. In Linux distributions, at least, tools that build binaries also collect the corresponding source code into an archive that can be delivered along with the binaries. The result is that most users of these systems have not needed to be concerned with source

[35] Scott K Peterson, 'Making compliance scalable in a container world' (9 July 2020),available at <https://opensource.com/article/20/7/compliance-containers>; Scott K Peterson, 'What's in a container image: Meeting the legal challenges' (31 July 2018), available at <https://opensource.com/article/18/7/whats-container-image-meeting-legal-challenges> both accessed 9 August 2020).

[36] See Jake Edge, 'Containers and license compliance' (2 May 2018), available at <https://lwn.net/Articles/752982/> accessed 9 August 2020.

availability. Source code is routinely available in the same format as the delivery of the executable software, using the same distribution mechanism.

For example, many Linux distributions use the RPM Package Manager (RPM) package management technology, in which binary packages are delivered in RPM format, and corresponding source code is normally available in a corresponding source RPM. In contrast, there is no convention for making available the source code corresponding to the executable software in a container image.

Another compliance-related challenge associated with the adoption of container technology is an increase in the number of distributors of Open Source software inexperienced with the requirements of Open Source licences. Companies that are beginning to offer their proprietary applications as container images may be facing GPL compliance obligations for the first time (see Chapter 5).

Determining what the corresponding source code is for a set of binaries is more challenging for container images than it is for conventional modular packages. With traditional server and desktop-oriented Linux distributions, source code is normally available to the person building the package. In contrast, binaries in container images are typically built using components previously compiled by third parties other than the container creator. When the container image is built, the source code for those components may or may not be readily at hand, depending on how the binaries were acquired and how they were built.

Recently Scott Peterson of Red Hat has described a way of solving these challenges by implementing a 'registry-native' approach to source code availability: delivering source code itself as a container image, through container registries. An Open Container Initiative (OCI)-conformant container image includes an image manifest which identifies the other elements of the image, including the various filesystem layers (each of which is a tar archive) making up the image. In the registry-native approach, a list of corresponding source artifacts is arranged as an image manifest, enabling the registry to serve those source artifacts in the same way it serves other container image parts. The layers of the source image are those source artifacts. Tools for moving container images can be directly applied to move the source artifacts.

There are a number of compliance-related benefits to adoption of the registry-native approach. It clearly satisfies the 'equivalent access' criterion of the GPL.[37] Compliance is addressed in an efficient manner, at the time of creation of a container image, avoiding the need to maintain additional, external processes and

[37] Patrick McHardy, a former Netfilter contributor who has gained notoriety through a series of controversial GPL compliance litigation actions in Germany, has reportedly raised GPL non-compliance claims based on failure to provide equivalent access. Greg Kroah-Hartman, 'Linux Kernel Community Enforcement Statement' (16 October 2017), available at <http://kroah.com/log/blog/2017/10/16/linux-kernel-community-enforcement-statement/> accessed 9 August 2020. While those claims may be without foundation, they have had the side effect of calling general attention to compliance with the equivalent access requirement.

mechanisms for making source code available. Compliance is also made portable: when an image is moved from one registry to another, it remains in compliance. The same tooling that moves executable container images can be used to move, store, and serve the source images on all hosting registries. Finally, the registry-native approach can take advantage of the deduplication capability that is inherent in the layered container image format, by storing separate source artifacts for each of the hundred or more software components from which the container image is built.[38]

20.4 Identifying the Licence of a Container

In recent years the term 'Open Source compliance' has come to be applied to activities that do not focus on substantive compliance with provisions of Open Source licences, but which instead centre upon compiling lists of information about the Open Source contents of software products.[39] The focus is on the discovery of what Open Source-derived software is in a product and how such software is licensed, as well as on how best to maintain, present, and share that information (see Chapter 6 on OpenChain and Chapter 7 on SPDX).

This sort of 'compliance' orientation can be expected to be increasingly directed towards containers as adoption of containers continues to grow. It has already occasionally manifested in discussions and activities in the community and among companies relating to how to describe or identify the 'licence' of a container. These activities sometimes reveal a failure to appreciate the licensing complexity of container images, even among those having technical familiarity with containers.

For example, in the OCI Image Format Specification,[40] one of the predefined annotation keys is 'org.opencontainers.image.licenses', which is described as 'License(s) under which contained software is distributed as an SPDX License Expression'.[41] But a typical container image is built from hundreds of components. An SPDX licence expression is generally used to convey licensing information for a single source file, or perhaps a simplified view of the licensing of one package. Such expressions are designed to allow composite expressions, such as 'GPL-2.0-or-later OR BSD-3-Clause'.[42] The SPDX licence's (see Chapter 7) expression format is not designed to represent the complex licensing information details that would

[38] For further information on the registry-native approach, including current and anticipated future implementation details, see Peterson, 'Making compliance scalable in a container world, see note 35.

[39] See, e.g., <https://compliance.linuxfoundation.org/about/> (Linux Foundation Open Compliance Program), accessed 10 August 2020.

[40] <https://github.com/opencontainers/image-spec> accessed 10 August 2020.

[41] <https://github.com/opencontainers/image-spec/blob/master/annotations.md#pre-defined-annotation-keys> accessed 10 August 2020.

[42] See SPDX Specification v2.2.0, Appendix IV: SPDX License Expressions, available at <https://spdx.github.io/spdx-spec/appendix-IV-SPDX-license-expressions/> accessed 10 August 2020.

accurately describe a typical container image, which would require an extremely lengthy conjunctive expression of standardised and ad hoc licence identifiers of little communicative value to those concerned about identifying issues of substantive Open Source licence compliance. There is some evidence that the practice of using Dockerfiles has led to confusion over how to describe the licence of a container image accurately. A Dockerfile is a text document that contains a set of commands for assembling a container image.[43] Although Dockerfiles tend to be fairly trivial and non-expressive, Dockerfiles are sometimes placed under an Open Source licence through application of a licence notice at the top of the file. In those cases, it sometimes happens that the nominal Dockerfile licence is mistakenly assumed to be the licence of the set of container images that can be generated from the Dockerfile. In actuality, a Dockerfile might be licensed under the MIT licence, while the associated container image will generally have some complex composite licence that will include GPLv2 and GPLv3.[44]

Another misconception is that the licence of a container image can be determined through reverse engineering of the image to attempt to extract licence texts. This is not a reliable approach because there is no reason to assume that the image has compliantly included all licence texts and of course reverse engineering is not legally permissible in all jurisdictions, though reverse engineering is implicitly allowed under Open Source licences and therefore is allowed in cases where a binary file is actually governed by an Open Source licence. Licence files have sometimes been deliberately removed from container images in an effort to make them smaller.

The best way to discover accurate information about the licensing terms applicable to a container image is to access the corresponding source code of the components of the image. This is another argument in favor of the registry-native approach discussed in the previous section. With complete source code, scanning tools like ScanCode Toolkit[45] can be used to extract the kinds of legal information that the user considers important.[46]

20.5 Containers and Network Services Copyleft

For the most part, copyleft licence requirements, including source code obligations and restrictions on licensing of derivative works, are triggered by

[43] 'Best practices for writing Dockerfiles', <https://docs.docker.com/develop/develop-images/dockerfile_best-practices/> accessed 10 August 2020.

[44] Hemel, see note 11.

[45] <https://github.com/nexB/scancode-toolkit> accessed 10 August 2020.

[46] Peterson, 'Making compliance scalable in a container world', see note 35; see also Peterson, 'The source code is the license' (29 December 2017), <https://opensource.com/article/17/12/source-code-license> accessed 10 August 2020.

distribution of copies to third parties. This is true of the GPL and GNU LGPL in the GPL family, as well as less widely adopted copyleft licences like the Eclipse Public Licence and Mozilla Public Licence. The growth of web services-based applications in the early 2000s gave rise to a concern that the GPL and other existing copyleft licences had a web services 'loophole'.[47] Then as now, web services were commonly based in part on upstream Open Source software, and improvements to that upstream code were not compelled by traditional copyleft licences to be shared as Open Source-licensed source code because such improvements were generally not distributed to third parties in source or binary form.[48]

This concern, along with an almost entirely distinct interest among some vendors in experimentation with copyleft/proprietary dual-licensing business models, led to development of a class of copyleft licences that attempted to extend the trigger for copyleft licence compliance to non-distribution scenarios involving provision of network services to third parties. Most of these licences effectively attempted to redefine 'distribution' to capture the network services case;[49] these licences are little used today, if not altogether obsolete. The one network services copyleft licence that has survived this earlier period of experimentation is the GNU Affero General Public Licence version 3 (AGPLv3), an FSF-maintained variant of GPLv3 first released in 2008.[50]

Instead of defining distribution or 'conveying', AGPLv3 implements the extension of copyleft to the deployment of network services using the defined term 'modified version' (which in GPLv3 is a synonym for 'work based on the Program'):[51]

> Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software.[52]

[47] See Tim O'Reilly, 'The GPL and Software as a Service' (12 July 2007), <http://radar.oreilly.com/2007/07/the-gpl-and-software-as-a-serv.html> accessed 10 August 2020.

[48] See Richard Fontana, 'The AGPL, Network Copyleft and the Cloud', <https://www.youtube.com/watch?v=ncXa0LFeQY0> accessed 21 August 2022 (presentation given for OpenUK Future Leaders session).

[49] See, e.g., Open Software Licence 3.0, available at <https://opensource.org/licenses/OSL-3.0> accessed 10 August 2020 (defining 'External Deployment' as 'the use, distribution, or communication of the Original Work or Derivative Works in any way such that the Original Work or Derivative Works may be used by anyone other than You, whether those works are distributed or communicated to those persons or made available as an application intended for use over a network" and requiring that "You must treat any External Deployment by You of the Original Work or a Derivative Work as a distribution"').

[50] <https://www.gnu.org/licenses/agpl-3.0.en.html> accessed 10 August 2020.

[51] See note 18.

[52] AGPLv3 § 13.

AGPLv3 is sometimes described as a licence designed for 'cloud', but this illustrates the ambiguity surrounding that technology industry term. Particularly in popular discussion of consumer IT issues, 'cloud' is often used as a synonym for software-as-a-service (SaaS), which itself can be seen as equivalent to providing software functionality remotely through network services. AGPLv3 is indeed designed to capture the SaaS case. In enterprise computing, however, 'cloud computing' tends to refer to running workloads within 'clouds', which are understood to mean public, private, and hybrid IT environments that abstract, pool, and share scalable resources across a network.[53] The two notions of 'cloud' overlap, because SaaS, including familiar large-scale consumer web services, is often implemented today using cloud computing techniques, increasingly through the use of containers, container orchestration, and other methods of cloud native development. It is however entirely possible to implement SaaS without the use of cloud computing in this specialised sense.

While AGPLv3 is the one notable network services licence to have survived the earlier period of copyleft licence experimentation in the 2000s, it has never been widely adopted for Open Source projects. Nevertheless, it is conceivable that AGPLv3-licensed packages will be included in a container. This may give rise to a concern that running a containerised application that includes an AGPLv3 package in production to provide a public-facing web service—a common use case for containers—will give rise to copyleft obligations, including a requirement to make source code available to service users.

If the AGPLv3 software is used without modification, which will be typical for containers since containers largely reuse pre-built upstream packages without change, then the network services copyleft requirements of AGPLv3 section 13 are not triggered.[54] In the rarer case in which the container image creator has made modifications to the AGPLv3-licensed software, there may be a requirement to provide the 'Corresponding Source' of that software to users, if indeed users are 'interacting with it remotely', which will not inherently be the case merely because a container including such AGPLv3-licensed software is used in providing the service. However, the scope of copyleft in such cases will not extend to 'mere aggregation' within a container and across multiple containers, for all the same reasons that were given in the discussion of the container image distribution case earlier.

20.6 The Rise of 'Source-Available' Licences Targeting Cloud Service Providers

Modern cloud computing, including its current container-driven cloud native phase, has been enabled by a rich ecosystem of Open Source projects and thus

[53] 'What is cloud computing?', <https://www.redhat.com/en/topics/cloud> accessed 10 August 2020.

[54] See Jeffrey Robert Kaufman, 'Do I need to provide access to source code under the AGPLv3 license?' (18 January 2017), <https://opensource.com/article/17/1/providing-corresponding-source-agplv3-license> accessed 10 August 2020.

by Open Source licensing. In the context of projects providing libraries and tools around web and cloud technology, it has long been observed that Open Source is increasingly dominated by non-copyleft licensing.[55] The evolution of Open Source licensing itself does not seem to have been significantly influenced by the industry shift to cloud. AGPLv3, a product of the pre-cloud era targeting the 'web services loophole', has not been widely adopted. In the years since the introduction of AGPLv3, only one Open Source network services copyleft licence of note has appeared: the Cryptographic Autonomy Licence 1.0 (CAL), which was approved by the Open Source Initiative (OSI) in 2020.[56] The novel features of CAL are its 'User Autonomy' provisions, rather than its mechanism for extending copyleft to network services; in any case, CAL has not yet seen notable adoption beyond its initial use case in the Holochain project.[57]

In recent years, much attention has been given to the phenomenon of large third-party cloud service providers monetising popular Open Source projects through offering proprietary 'wrapper' services that benefit from operational advantages.[58] Companies maintaining such projects (which in many cases involve database technology),[59] generally using some variety of 'Open Core' business model, have decried what they regard as 'strip mining' of their projects by cloud providers. Several of these companies have reacted to this development by migrating their projects to 'source-available' licences, so called because they are applied to published source code and may resemble Open Source licences in certain respects, but they do not conform to the software freedom norms underlying the OSI's Open Source Definition (OSD).[60] These source-available licences are designed more or less explicitly to limit or prevent cloud providers from offering competing services based on the same projects, something which is definitionally allowed under any Open Source licence. Some commentators have described these licences as 'hybrid' or 'non-compete' licences.[61]

The source-available non-compete licences deviate from Open Source licensing norms in various ways. The Commons Clause,[62] which Redis Labs used for

[55] See Christine Lemmer-Webber, 'A Field Guide to Copyleft Perspectives' <http://dustycloud.org/blog/field-guide-to-copyleft/> accessed 18 April 2022.

[56] <https://opensource.org/licenses/CAL-1.0> accessed 18 April 2022.

[57] <https://github.com/holochain/holochain> accessed 18 April 2022.

[58] See Stephen O'Grady, 'Tragedy of the Commons Clause' <https://redmonk.com/sogrady/2018/09/10/tragedy-of-the-commons-clause/> accessed 18 April 2022.

[59] See Josh Berkus, 'Why database projects can't leave licenses alone', <https://www.youtube.com/watch?v=uoKcueUkCX4&feature=youtu.be> accessed 18 April 2022 (presentation given at State of the Source Summit 2020).

[60] <https://opensource.org/osd> accessed 18 April 2022 (including requirements that licences not discriminate against persons or groups and fields of endeavour); see also 'What is free software?', <https://www.gnu.org/philosophy/free-sw.en.html> accessed 18 April 2022 (FSF's Free Software Definition).

[61] Stephen O'Grady, 'What does Open Source mean in the era of cloud APIs?', <https://redmonk.com/sogrady/2019/01/25/Open Source-cloud-apis/> accessed 18 April 2022.

[62] <https://commonsclause.com/> accessed 18 April 2022.

previously AGPLv3-licensed Redis modules before more recently switching them to the Redis Source Available Licence,[63] supplements a standard Open Source licence with a refusal to grant 'the right to Sell the Software'. The Redis Source Available Licence grants permission to 'use' the software 'only as part of Your Application, but not in connection with any Database Product that is distributed or otherwise made available by any third party'. CockroachDB uses a variant of the Business Source Licence that prohibits the licensee from providing CockroachDB as a service.[64] The Timescale Licence, which covers certain parts of Timescale Community Edition, prohibits using the software to provide a database as a service.[65] The Fair Source Licence, created by Sourcegraph, requires organisational licensees to pay for a licence once they exceed a specified threshold of users.[66] The Confluent Community Licence prohibits 'making available any software-as-a-service, platform-as-a-service, infrastructure-as-a-service or other similar online service that competes with Confluent products or services that provide the Software'.[67]

The Server Side Public Licence version 1[68] (SSPL) deserves closer attention. SSPL differs from other source-available non-compete licences in two respects: it is based on an Open Source licence text and its licence steward sought to have the licence legitimised as an Open Source licence. SSPL was created in 2018 by MongoDB, Inc. for its MongoDB Community Server, which was previously licensed under AGPLv3. In 2021 it was adopted by Elastic for its Elasticsearch and Kibana projects as a replacement for the Apache Licence 2.0.[69]

Like AGPLv3, SSPL largely reuses the GPLv3 text with few changes. However, in place of AGPLv3 section 13, SSPL has the following provision:

> 13. Offering the Program as a Service.
>
> If you make the functionality of the Program or a modified version available to third parties as a service, you must make the Service Source Code available via network download to everyone at no charge, under the terms of this License. Making the functionality of the Program or modified version available to third parties as a service includes, without limitation, enabling third parties to interact

[63] <https://redislabs.com/wp-content/uploads/2019/09/redis-source-available-license.pdf > accessed 18 April 2022.

[64] Peter Mattis, Ben Darnell, and Spencer Kimball, 'Why we're relicensing CockroachDB', <https://www.cockroachlabs.com/blog/oss-relicensing-cockroachdb/> accessed 18 April 2022.

[65] <https://www.timescale.com/legal/licenses> accessed 18 April 2022.

[66] <https://fair.io/?a> accessed 18 April 2022.

[67] <https://www.confluent.io/confluent-community-license> accessed 18 April 2022.

[68] <https://www.mongodb.com/licensing/server-side-public-license> accessed 18 April 2022.

[69] Shay Banon, 'Doubling down on open, Part II', <https://elastic.co/blog/licensing-change> accessed 18 April 2022.

> with the functionality of the Program or modified version remotely through a computer network, offering a service the value of which entirely or primarily derives from the value of the Program or modified version, or offering a service that accomplishes for users the primary purpose of the Program or modified version.
>
> 'Service Source Code' means the Corresponding Source for the Program or the modified version, *and the Corresponding Source for all programs that you use to make the Program or modified version available as a service, including, without limitation, management software, user interfaces, application program interfaces, automation software, monitoring software, backup software, storage software and hosting software, all such that a user could run an instance of the service using the Service Source Code you make available.*[70]

The key difference between SSPL section 13 and AGPLv3 section 13 is that the AGPLv3 Corresponding Source requirement covers only the 'modified version' as that is defined in GPLv3 and AGPLv3. The source code that must be offered to interacting users is equivalent to what would be required under GPLv3 if an object code version were being distributed. In SSPL, the required source code extends to the entire stack of programs used to implement the service, including programs that would be considered 'mere aggregation' of separate and independent programs under the GPL. Moreover, the entire set of Corresponding Source, including the source code of all those independent programs, must be provided under SSPL. In practice this requirement is not possible to comply with (without avoiding providing a service entirely), because it is not practically possible to form an entire services stack out of SSPL-licensed components without implementing much of that stack from scratch. For this reason, the SSPL section 13 requirement can be regarded as a prohibition on providing the software as a service, disguised as a modification of the GPLv3/AGPLv3 Corresponding Source requirement.

Unlike the licence stewards of other source-available non-compete licences, MongoDB submitted SSPL for approval by the OSI in 2018. The licence was discussed over the course of several months on the OSI's license-review mailing list, during which MongoDB submitted a revised draft; reception was largely hostile. MongoDB withdrew the licence from consideration, likely to avoid formal rejection by the OSI.[71] Objections to the licence from an OSD conformance perspective generally focused on the anti-discrimination provisions of the OSD as well as the spirit, if not the letter, of OSD 9 ('The license must not place restrictions on other software that is distributed along with the licensed software. For example, the licence must not insist that all other programs distributed on the same medium must be Open Source software.').

[70] Server Side Public Licence § 13 (emphasis added).

[71] OSI Board of Directors, 'The SSPL is not an Open Source license' (19 January 2021) <https://opensource.org/node/1099> accessed 18 April 2022.

The use of source-available licences has been the subject of much recent controversy in the Open Source technical community, not just because these licences violate Open Source definitional norms and have been applied to widely used projects, but also because they are seen as creating confusion around the meaning of Open Source as an identifying label. Some have complained that the 'Open Source companies' at issue have used Open Source licensing as a means of gaining adoption and thus causing widespread dependency on the projects, raising the costs to users of switching to alternative Open Source technologies. At the same time, the effectiveness of the source-available licence strategy as a protection against competition from cloud providers is doubtful, given the ability of well-resourced cloud providers to reimplement, fork, or create application programing interface- (API) compatible alternatives to such projects.[72]

[72] Stephen O'Grady, 'Cockroach and the source available future', <https://redmonk.com/sogrady/2019/06/21/cockroach-source-available> accessed 18 April 2022.