Facebook Open Source Software

Posted on March 22, 2019

Facebook began as a web application built substantially on open-source technologies and later developed many systems that it released for use by the wider engineering community. The original essay describes Facebook as a LAMP site using Linux, Apache, MySQL, and PHP. That description is historically useful but incomplete for a platform that expanded into a global family of services under Meta. The early LAMP foundation was modified, supplemented, and in some areas replaced by specialized databases, programming languages, runtimes, storage systems, networking software, mobile frameworks, machine-learning tools, and data-center hardware designed for operation at very large scale (Meta Open Source, “Projects”).

Open source refers to software whose source code is available under a license permitting specified forms of use, inspection, modification, and distribution. It does not mean that every system operated by Meta is public, that all user data are accessible, or that the software lacks ownership and licensing conditions. Meta combines open-source projects, third-party open-source dependencies, internally developed proprietary systems, commercial services, and confidential product code. Its open-source strategy is important because tools first created to solve internal engineering problems have influenced web development, mobile applications, databases, artificial intelligence, and infrastructure outside the company (Meta Open Source, “About”).

The Historical LAMP Foundation

LAMP is an acronym for Linux, Apache, MySQL, and PHP. In a conventional LAMP application, Linux provides the operating system, Apache serves web requests, MySQL stores persistent data, and PHP generates dynamic pages. Facebook’s early architecture used this general combination because the components were available, flexible, familiar, and supported rapid development.

Growth placed pressure on every layer. A simple architecture that works for a university network cannot automatically serve billions of accounts, photographs, messages, videos, advertisements, and real-time interactions. Facebook engineers optimized open-source components, created additional services, distributed data across many machines, and separated functions that a smaller application might keep together. Describing modern Facebook only as a LAMP site therefore hides the scale of that evolution (Meta Open Source, “Projects”).

Linux

Linux has long provided a foundation for server infrastructure because it can be inspected, configured, automated, and adapted to specialized hardware and network requirements. Large

operators can tune scheduling, memory, storage, networking, observability, and security while contributing some improvements back to upstream communities.

The original essay says Facebook optimized Linux particularly for network throughput. The broader point is correct: hyperscale services require operating-system and network behavior suited to very high traffic. However, internal modifications, upstream versions, and deployments change over time. An academic overview should explain the architectural role rather than claim one permanent configuration.

PHP, Hack, and HHVM

PHP helped early Facebook engineers develop web features quickly. Interpreted dynamic languages can support rapid iteration but may consume substantial computing resources at scale. Facebook initially created HipHop for PHP to transform PHP code into optimized C++, and later developed the HipHop Virtual Machine, or HHVM, as a high-performance runtime.

Facebook also created Hack, a language related to PHP that added gradual and static typing features intended to improve reliability and development at scale. Historical articles from Facebook Engineering describe the conversion of its codebase and the release of Hack and HHVM as open source. These projects show how a company can retain the productivity of a familiar language while developing tools to address performance and code-quality limits.

The original statement that Facebook simply compiles PHP into native code captures one stage of this history but should not be treated as a timeless description of every request. Architecture evolves, and a public social platform includes many services written in several languages.

MySQL and Distributed Data

MySQL has powered important Facebook workloads. Meta engineers have described extensive infrastructure for operating MySQL instances and the migration of major workloads to MySQL 8.0. At scale, the database is supported by replication, sharding, caching, schema management, custom tooling, storage engines, and application logic (Meta Engineering, “Migrating Facebook to MySQL 8.0”).

The original essay states that joins and logic were moved to web servers. Some early scaling approaches avoided expensive database operations and distributed work differently, but this should not be presented as a universal rule. Query design depends on data model, consistency, latency, and workload. Modern database systems can perform complex joins effectively in appropriate contexts, while distributed services sometimes denormalize data or combine results outside the database.

MyRocks and RocksDB

RocksDB is an open-source embeddable key-value store developed at Facebook and based on log-structured merge-tree concepts. It is designed for high-performance storage on fast local devices and has been adopted in many systems beyond Meta. MyRocks integrates RocksDB technology with MySQL to improve storage efficiency for selected workloads (Meta Open Source, “Projects”).

These projects demonstrate an important form of open-source innovation: a company may extend a widely used database rather than build every interface from nothing. Users can benefit from MySQL compatibility while applying a different storage engine. The approach also creates maintenance and migration complexity, as custom features must remain compatible with new upstream versions.

Haystack and Photo Storage

Haystack was developed to store and retrieve the enormous number of photographs uploaded to Facebook efficiently. Conventional file systems can incur excessive metadata operations when billions of small files are stored individually. Haystack combined many photos into larger storage objects and used indexes to locate them with fewer disk operations.

The original essay says Haystack makes sure photographs remain safe until the account owner deletes them. That claim is too strong and confuses storage architecture with retention, privacy, backup, and account policy. Haystack addressed efficient photo storage and access. Data durability and deletion depend on replication, recovery, product policy, legal obligations, and later infrastructure. No storage system can guarantee safety in absolute terms.

Scribe and Logging

Scribe was created to collect large volumes of log messages from distributed servers. Facebook released it as open source in 2008 after using it for many logging cases and very high message volumes. Logging supports performance analysis, debugging, security, product measurement, and operational monitoring (Meta Engineering, “Scribe: Transporting Petabytes per Hour”).

Meta later explained that the open-source Scribe project was archived as its internal infrastructure became more complex and specialized. This history is important because open-source projects have lifecycles. Release does not guarantee permanent maintenance. Organizations adopting a project should examine current activity, governance, community, and migration options (Meta Engineering, “Scribe: Transporting Petabytes per Hour”).

React

React is an open-source JavaScript library for building user interfaces. Meta's official open-source description emphasizes declarative views and reusable components. React allowed developers to represent interface state through components and update the necessary parts of a page as data changed (Meta Open Source, "React").

React became widely influential beyond Facebook because it addressed a common problem in complex interactive applications. Its ecosystem includes independent contributors, companies, frameworks, and tools. Adoption brings flexibility and a large community, but organizations still need architecture, testing, accessibility, security, and long-term maintenance practices.

React Native

React Native applies React concepts to native mobile application development. Developers can use JavaScript or related tooling while rendering platform-native interface components. It can allow teams to share concepts and portions of code across iOS and Android while retaining access to native capabilities (Meta Open Source, "Projects").

Cross-platform development involves trade-offs. Shared code may accelerate delivery, but performance, platform conventions, debugging, upgrades, and native integrations require expertise. Open source provides access and community contribution; it does not remove engineering decisions.

GraphQL

Facebook developed GraphQL as a query language and runtime approach for APIs and later released it openly. Clients specify the data structure they need, and servers resolve the requested fields. This can reduce over-fetching and support rapidly changing user interfaces (Meta Open Source, "Projects").

GraphQL is not a database and does not eliminate authorization, caching, monitoring, or query-cost controls. Poorly governed queries can create performance or security problems. Its influence illustrates how an internal interface pattern can become a broader standard through documentation and community adoption.

PyTorch

PyTorch is an open-source machine-learning framework that supports research prototyping and production deployment. Meta's open-source project page highlights tensor computation, GPU

acceleration, distributed training, and an ecosystem for computer vision, natural-language processing, and other fields. PyTorch became widely used in universities, research laboratories, startups, and major technology companies (Meta Open Source, “PyTorch”).

Machine-learning frameworks show that open source can accelerate research by allowing scientists to reproduce methods, inspect implementation, and build on shared tools. The availability of a framework does not make every model, dataset, or production system open. Training data, model weights, safety methods, and deployed services may have different licenses and access.

Open Compute Project

Facebook also helped launch the Open Compute Project to share data-center hardware designs and related infrastructure ideas. Open source can therefore extend beyond software into servers, racks, power systems, networking, and storage hardware. Sharing designs can promote standardization, supplier competition, and efficiency (Meta Open Source, “Projects”).

Hardware openness differs from downloading a software library. Manufacturing, certification, supply chains, facilities, and capital are required. The broader strategic principle is that collaboration on infrastructure can allow companies to focus competition on services and innovation above the shared layer.

Why Meta Releases Open-Source Software

Open sourcing can attract external contributors, improve recruitment, encourage standards, build ecosystems, and reduce duplication across the industry. When many organizations use a project, they may identify bugs, add integrations, and strengthen documentation. A widely adopted framework can also influence developer skills and technology choices in ways beneficial to its originator (Meta Open Source, “About”).

Release can increase credibility and allow researchers to evaluate methods, but it is not purely charitable. Companies select what to open according to strategy, legal risk, competitive advantage, and maintenance capacity. Open source can benefit both the originating company and the public.

Control

The original essay identifies control as an advantage. Access to source code allows an organization to inspect behavior, modify features, audit dependencies, and avoid relying entirely on one vendor’s roadmap. It may be possible to maintain a fork if the original project changes direction.

This control is not free. A custom fork creates responsibility for security updates, compatibility,

testing, and documentation. The more an organization diverges from the community version, the harder future upgrades may become. Effective control often means participating upstream rather than customizing privately without limit.

Cost

Open-source software may have no license fee for basic use, but total cost includes engineering, hosting, integration, security, support, training, compliance, and migration. A proprietary product may be cheaper for an organization lacking internal expertise, while open source may be more economical at scale or where customization matters.

Cost evaluation should include exit options and data portability. “Free” software can create expensive operational dependence if the organization cannot maintain it. Conversely, commercial support is available for many open-source technologies.

Security

The original essay claims that open source is more secure because users can inspect and correct code. Visibility can support review, rapid patching, and independent research, but it does not guarantee that anyone has examined a particular component. Popular projects can contain vulnerabilities, compromised dependencies, or delayed maintenance.

Security depends on governance, contributor practices, release signing, dependency management, code review, testing, vulnerability response, and responsible deployment. Proprietary and open-source software can both be secure or insecure. The relevant question is whether the project and adopting organization manage risk competently.

Stability and Longevity

Open source can support longevity because the code remains available even if one vendor changes strategy. A broad community may maintain a project across organizations. Open standards can improve interoperability and reduce lock-in.

Scribe’s archival demonstrates the limitation: code availability does not guarantee active maintenance. A project may lose contributors or become obsolete. Adopters should examine release frequency, issue response, governance, documentation, dependencies, and the number of organizations capable of maintaining it (Meta Engineering, “Scribe: Transporting Petabytes per Hour”).

Innovation and Learning

Open repositories allow developers to study production-quality code, submit changes, report issues, and learn modern practices. Universities and small companies can use tools developed at a scale they could not finance independently. Shared infrastructure speeds experimentation and prevents every team from recreating the same foundation (Meta Open Source, “About”).

Participation also requires respectful community governance. Codes of conduct, transparent review, documentation, mentorship, and clear decision rights influence whether contributors can participate meaningfully. A public repository without responsive maintainers is not a healthy community.

Licensing

Open-source licenses define legal permissions and obligations. Permissive licenses may allow broad use with attribution, while copyleft licenses can require distribution of source under specified conditions when derivative software is shared. Organizations must track licenses, notices, patents, trademarks, and dependency obligations.

Source availability is not always the same as open source under recognized definitions. Some models or datasets impose restrictions that limit fields of use. Legal and engineering teams should verify each asset rather than rely on the label “open.”

Governance and Foundations

Some projects remain led by the originating company, while others move to independent foundations or shared governance. Governance affects roadmaps, trademark control, contributor trust, and the ability of competitors to participate. A project used across the industry may benefit from decision structures that do not depend entirely on one company.

Corporate leadership is not automatically harmful; dedicated engineers can provide direction and resources. The key is transparency about authority and contribution.

Operational Risks

Using open source introduces supply-chain risk through dependencies, build systems, package registries, and contributor accounts. Organizations need inventories, version control, vulnerability scanning, reproducible builds where feasible, and processes for urgent patches. Abandoned dependencies may need replacement.

Meta’s ability to operate open-source software at hyperscale does not mean another organization can

copy the architecture directly. Systems are designed for specific workloads, teams, and infrastructure. Adoption should begin with requirements and testing rather than admiration for the company that created the tool.

Open Source and User Privacy

Open-source components do not make Facebook's data practices automatically transparent. A public user-interface library reveals little about advertising systems, ranking decisions, data retention, or account governance. Software openness and platform accountability are related but distinct issues.

Open code can support auditing where the deployed version and configuration are known, but most users cannot verify a large service end to end. Privacy depends on product design, law, policy, security, and organizational conduct in addition to software licensing.

Updating the Original Architecture Description

The original essay's references to Linux, PHP, MySQL, Haystack, and Scribe capture important stages in Facebook engineering history. They should be presented historically rather than as a complete current stack. Meta's official project directory now covers a wide range of areas including React, PyTorch, databases, compilers, mobile development, artificial intelligence, privacy, hardware, and infrastructure (Meta Open Source, "Projects").

A modern platform is not one monolithic website. It contains thousands of services, clients, data systems, networks, and development tools. Architectural descriptions quickly become outdated, so reliable analysis should cite dates and official engineering sources.

Conclusion

Facebook's early use of the LAMP stack demonstrates how open-source software can support rapid creation and later scale through modification. Linux, PHP, and MySQL provided foundations, while systems such as Haystack and Scribe addressed specialized photo-storage and logging problems. HHVM and Hack responded to language performance and reliability needs.

Meta's broader open-source influence includes React, React Native, GraphQL, RocksDB, PyTorch, data infrastructure, developer tools, and hardware collaboration. Some projects remain active and widely adopted, while others such as the public Scribe project reached the end of active maintenance. Open source is therefore an ecosystem and lifecycle, not a permanent guarantee (Meta Open Source, "Projects"; Meta Engineering, "Scribe: Transporting Petabytes per Hour").

The benefits include inspection, customization, learning, ecosystem growth, interoperability, and

reduced vendor dependence. The risks include maintenance burden, vulnerable dependencies, licensing complexity, project abandonment, and false assumptions about security. Open source helped Facebook scale and allowed its engineering solutions to influence the wider industry, but success depends on governance, operational expertise, and responsible adoption rather than source-code availability alone (Meta Open Source, “About”).

Works Cited

Meta Open Source. “About: Empowering Communities through Open Source Technology.”

Meta Open Source. “Projects.”

Meta Open Source. “React.”

Meta Open Source. “PyTorch.”

Meta Engineering. “Migrating Facebook to MySQL 8.0.” 2021.

Meta Engineering. “Scribe: Transporting Petabytes per Hour.” 2019.