activestate

Scalable Dependency vendoring

A COST-EFFECTIVE PAT H TO DEPENDENCY MANAGEMENT

Executive Summary

DevOps teams that have experimented with different dependency management strategies often settle for self-vendoring as "the least worst" way to manage their open source dependencies, since it solves many common issues despite the complexities it introduces.

However, for software development managers concerned with maximizing the velocity of their teams, dedicating developers to non-differentiating tasks like dependency management is a poor use of valuable resources. Instead, the time and resources spent on dependency management tasks are often better spent on creating features and functionality that differentiate the organization's offerings from competitors.

For this reason, DevOps teams typically automate dependency management tasks wherever possible to reduce developer burden. However, automation tools can often create larger workload challenges than they solve. For example, automating the identification of vulnerabilities is merely the tip of the iceberg, since it does nothing to help with the lengthy, manual remediation process.

Dependency automation tactics rarely cost-effectively scale across any enterprise that is characterized by multiple development teams working with diverse technology stacks. While better automation tools are always welcome, software development managers may want to explore alternative approaches, such as outsourcing dependency vendoring to a trusted service provider.

Introduction

The software industry's widespread adoption of open source software has resulted in the continuous reuse of open source libraries to gain specific functionality. These libraries, written by members of the open source community, become dependencies of the commercial software. While employing open source saves time and promotes innovation, it has also created the need for developers to continuously track and manage third-party dependencies to ensure that their project builds successfully without introducing known vulnerabilities. All of which dramatically increase operational costs and time to market.

Effective dependency management helps reduce process variability and increase predictability. But in practice, software vendors continue to struggle with choosing, using and maintaining open source dependencies effectively. For this reason, multiple dependency management solutions have been proposed and tried over the past decades, including:

Relying on a language's **package manager** to install dependencies on demand from a public repository.

- **Pros:** easy to set up; community supported, and allows a highly automatable "infrastructure as code" approach.
- **Cons:** tooling proliferates, often requiring one or more tools per operating system and per language. The availability of prebuilt packages can vary by OS, or disappear altogether such as when dependencies become corrupted or deleted from public repositories.

Relying on an **artifact repository** to cache a standard, shared set of approved dependencies for all development teams to use.

- **Pros:** creates team and environment consistency; allows for provenance tracking, and provides a central location for dependency import, distribution, management and auditing.
- **Cons:** Poor support for native libraries; inconsistent states are still possible with manually-driven installations; expensive.

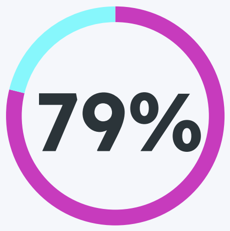
Relying on **Virtual Machines (VMs) or containers** to provide consistent environments.

- **Pros:** Better support for consistent use of native packages; faster deployment speeds, and less tooling requirements by users.
- **Cons:** build complexity now requires multiple tools per OS, per language, and per image. Additional complexity arises with versioned image distribution, updating, and creating backwards compatibility.

Most DevOps teams start by relying on package managers and may migrate to other solutions as limitations are uncovered. More commonly, rather than address the limitations of dependency management best practices, a surprising number of DevOps teams simply ignore their dependencies once they've included them in their codebase, as evidenced by numerous reports and surveys:

Open source libraries are constantly evolving; what appears secure today may not be tomorrow.

Despite this dynamic landscape, 79 percent of the time, developers never update third-party libraries after including them in a codebase.



Source: Veracode's State of Software Security v12

The amount of work associated with updating dependencies, the opportunity costs they engender, and the risk of breaking the build all serve as drags on update frequency. It's this sharp distinction between "risk" and "work" associated with updating dependencies where most DevOps teams end up inflicting self-harm, including:

- Lagging performance as outdated dependencies forego speed and functionality enhancements.
- Instability, which inevitably occurs when an update to a critical vulnerability is applied, resulting in a rabbit hole of fixes that break the build, requiring more fixes, and so on.
- "Works on my machine" issues that arise from the update being inconsistently applied across all teams/ environments.

Alternatively, critical vulnerabilities go unpatched, exposing the organization to cyberattack. To minimize these risks and costs, some enterprises have turned to self-vendoring despite the complexities it engenders.

Dependency Vendoring Pros & Cons

Dependency vendoring is a dependency management strategy that requires the inclusion of third-party source code in a product's codebase. In practical terms, that means checking a single version of each open source dependency into the source control system rather than relying on a package manager to install dependencies into environments on demand.

For those who adopt it, self-vendoring represents a viable way to solve the limitations of other dependency management strategies, such as:

- **Dependency Conflicts -** Including a specific version of a dependency in a codebase ensures compatibility with all the other components in that codebase.
- **Broken Builds -** Package managers (unless explicitly instructed otherwise) will import newer versions of dependencies and/or transitive dependencies as they become available, potentially breaking the build.
- **Inconsistencies -** Self-vendoring ensures successful builds work reliably and consistently on every team member's machine.
- **Deficiencies -** When bugs, vulnerabilities or requests for functionality are slow to be implemented by third-party authors and maintainers, DevOps teams can customize, patch and/or add new functionality themselves.

However, self-vendoring is not a panacea, because it can often create as many problems as it solves, including:

- **Complex Builds -** Building dependencies and all of their transitive dependencies from source code for all major operating systems requires extensive language and OS expertise that many teams may lack.
- **Unsecure Applications -** Once checked in to a code repository, dependencies are rarely updated for fear of breaking the build. This leads to buggy codebases riddled with security holes.
- **Increased Complexity -** Dependency and transitive dependency source code inflates the source tree, making tasks like code review, license audits, or even just checking out the codebase overly difficult.
- **Developer Stress -** DevOps teams that standardize on a set of dependencies for every project can create internal friction as teams make demands on each other to fix, update, patch, or otherwise manage the dependency they checked in.

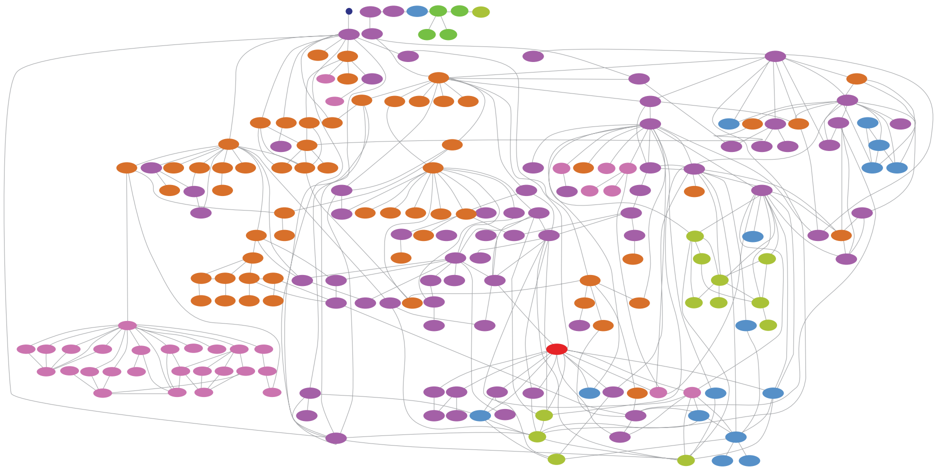
Experienced DevOps teams that have tried managing their dependencies with native package managers, artifact repositories and special-built containers/VMs already understand the tradeoffs between each strategy, and have settled on self-vendoring because it best fits their software development processes despite its drawbacks. Following best practices can help counter the weaknesses.

Dependency Vendoring Best Practices

Dependency vendoring makes third-party code the responsibility of internal development teams who already have their own code to manage. This results in a tradeoff between time and resources spent on low-value work associated with managing dependencies, versus high-value coding of differentiating features and functionality. Best practices can help minimize non-differentiating work.

CREATE DEPENDENCY GRAPHS/SBOMS

A dependency graph is a data structure that shows the inter-relational dependencies for each entity in the software product. Essentially, it's a diagram to visualize a product's interdependencies, and by extension, its complexity and maintainability.



Source: dependency graph for TinyTag explorer

A Software Bill Of Materials (SBOM) is an alternative way of enumerating all of the components in a software product, along with each component's name, version, license, and supplier. Both methods can help software vendors understand the risk posed by the proliferation of dependencies.

MINIMIZE THE NUMBER OF DEPENDENCIES

Prior to the availability of dependency managers, creating a dependency for an eight-line code library was a rare event, because it created too much overhead for minimal benefit. Package managers have greatly simplified this process. But even vendoring an eight-line dependency into a codebase is a non-trivial act since it still requires vetting for:

- Code quality
- Maintainability
- Security/vulnerabilities
- Licensing
- Transitive dependency quality, maintainability and security

As a result, DevOps should weigh the cost effectiveness of rewriting a dependency as native code, instead.

ABSTRACT DEPENDENCIES

Open source dependencies are written by third parties that are under no contractual obligation to update, maintain, or improve their code. In some cases, newer versions of a dependency may take the existing functionality in unexpected (and undesired) directions, introduce a fatal bug or vulnerability, or be abandoned altogether. For these reasons, code must be written in a way to ensure that existing dependencies can be replaced, substituted, or re-written with minimal effort by abstracting the way they're implemented.

AUTOMATE DEPENDENCY MANAGEMENT

Trying to manually manage all of the tasks associated with dependency vendoring is a non-starter. Instead, use dependency automation tools to help manage tasks such as:

- **Security -** Third-party services like GitHub's dependabot or Software Composition Analysis (SCA) tools automatically identify dependency vulnerabilities in a timely manner, and may also suggest remediation options.
- **Compliance -** The same SCA tools can also be used to identify the open source licenses in dependencies, and help ensure compliance with the organization's licensing guidelines.
- **Building from Source -** Third-party CI/CD solutions are typically used to build dependencies from source code and package them for deployment on one or more target platforms.
- **Control -** Making a set of pre-vetted, approved dependencies available for use by an organization's teams (and managing them over time) is easier if they're located in a central artifact repository.

It should be noted that, despite its many benefits, dependency management automation falls short of delivering a complete solution. For starters, automating all of these tasks requires DevOps to cobble together multiple tools and processes, since no existing solution automates all of them out of the box. Additionally, it's one thing to be notified of a vulnerability and a newer version of the dependency that resolves it, but quite another to build that new version, ensure that it works within existing environments, and then redeploy/update development, test and production instances. In practice, this kind of self-serve automation requires developers to dedicate a large chunk of their time to continuously managing dependencies rather than writing code, which increases operational costs and time to market.

As an alternative, consider employing an outsourcing service that can manage dependencies on your behalf. For example, ActiveState's "managed distribution" service securely builds, monitors, maintains, remediates and packages open source dependencies into dev, test and production runtime environments on behalf of development teams, no matter their deployment platform. As a result, DevOps can minimize the time and resources needed to ensure that dependencies are up to date, vulnerability free, and haven't suffered from configuration drift, thereby keeping the environment in sync with the development, test and production environments.

Conclusions

Dependencies are now an established foundation stone in the software development process, but the way dependencies are managed continues to evolve. Software development managers have implemented a number of tools based on current best practices, creating processes that evaluate, track and attempt to reduce dependency risk - from the original adoption decision, all the way through to production.

However, enterprises are beginning to realize the high opportunity cost of dependency management. While dependency managers themselves have essentially eliminated the cost of downloading and installing a dependency, the cost of updating, remediating, rebuilding and redeploying a dependency continues to grow. In fact, building and managing dependencies requires language and operating system expertise, which often requires dedicating some of the organization's most experienced and valuable resources.

Dependency automation tactics rarely cost-effectively scale across any enterprise characterized by multiple development teams working with diverse technology stacks. DevOps teams concerned with the security and compliance risks posed by the use of third-party dependencies, as well as time to market for their software offerings, should look to emerging technologies that can minimize the time and resources allocated to dependency management. Alternatively, explore outsourcing dependency vendoring to a trusted service provider.

About ActiveState

ActiveState enables DevOps, InfoSec, and Development teams to improve their security posture while simultaneously increasing productivity and innovation to deliver secure applications faster.

With a single platform that tames open source complexity, teams get a continuously secure software supply chain, unparalleled observability, robust vulnerability management, continuous upgrades, and governance support that enhance collaboration across the organization.

All from the trusted partner that pioneered and continues to lead enterprise adoption and use of open source software.

Start An Enterprise Trial

www.activestate.com • Toll-free in NA: 1-866.631.4581 • solutions@activestate.com