sonatype | LEARN

Visit us at learn.sonatype.com

Quick Reference Guide

Course: InnerSource Insight

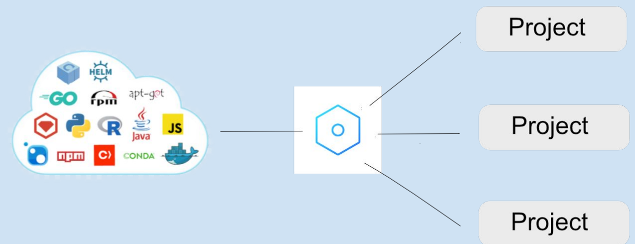
Introduction to InnerSource

DEFINING INNERSOURCE

“InnerSource” refers to bringing the core principles of open source and collaboration entirely **within the walls of your organization** - focused on your own **intellectual property**, technology, and teams.

Innersource is not a product or service that you buy and install on your network.

It is instead a term that refers to the overall workflow, methodology, community, and culture that optimizes an organization for open source style collaboration.



WHAT ARE INNERSOURCE COMPONENTS

Both InnerSource and open source components are software components that are used to build software. The **key difference** is that InnerSource components are **internally developed components that are shared with other internal applications.**

Lifecycle identifies transitive dependencies pulled in by an InnerSource component. This helps reduce the amount of time it takes to identify transitive dependency violations as reported in a **Software Bill of Materials SBOM**.

To learn more, check out our *Insider’s guide to federal secure development mandates and compliance* blog.

VIOLATIONS - FOCUS ON WHAT IS ACTIONABLE

When **fixing policy violations** it's all about focusing on what you as the developer can do. One of the easiest ways of doing this is to upgrade to a newer version that does not have any policy violations.

Understanding the component relationships allows you to **focus on what is actionable**. You need to know which dependencies are direct and which ones are transitive.

Note: To upgrade a transitive dependency, the direct dependency needs to no longer use it.

sonatype | LEARN

Visit us at learn.sonatype.com

Quick Reference Guide

Course: InnerSource Insight

Introduction to InnerSource

WITH INNERSOURCE, IT'S ALL ABOUT THE TRANSITIVES

We say "it's all about the transitives" because any **violation** associated with InnerSource components are referring to the open source **transitive dependencies** that the InnerSource component relies on.

> **#TechTalk**: To state this another way, the "child" transitive dependencies will be the source of the policy violation, but the fix will be with the "parent" InnerSource component.

Why Is All of This So Important?

The difficulty with remediating these specific policy violations is understanding they are **NOT direct dependencies**, but in fact, ***transitive dependencies of the InnerSource component***. Historically, teams have struggled with identifying how those open source vulnerabilities align to their internally developed components.

USING MAVEN & npm WITH INNERSOURCE INSIGHT

Maven	**npm**
Lifecycle has the information to link the **consumer** and **producer**. • Consumers are linked to producers using the metadata provided by **Maven**. • A **pom.xml** declares what artifacts are built and lays out the dependencies. • With this information, Lifecycle links a producer to a consumer via **the artifact and dependency linkage**.	In order to include dependency type information (i.e. Direct vs Transitive), a **package.json** file must also exist and be readable as a sibling file to the target **lock file** (i.e. one of the listed below). This file is typically auto-generated and managed by a package manager. • Yarn.lock • Pnpm-lock.yaml • Package-lock.json • Npm-shrinkwrap.json
Check out the Sonatype CLM for Maven documentation and the Gradle Plugin to learn more.	Check out the npm Application Analysis documentation to learn more about supported npm manifests.