



What does it take to deploy to production with confidence?

Ixchel Ruiz

KARAKUN

Ixchel Ruiz

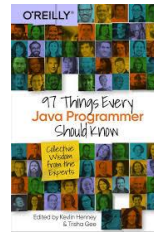


[linkedin.com/in/ixchelruiz](https://www.linkedin.com/in/ixchelruiz)

[@ixchelruiz@mastodon.social](mailto:ixchelruiz@mastodon.social)

[@ixchelruiz.bsky.social](https://bsky.social/ixchelruiz)

github.com/ixchelruiz



What does it take to
deploy to production
with **confidence**?

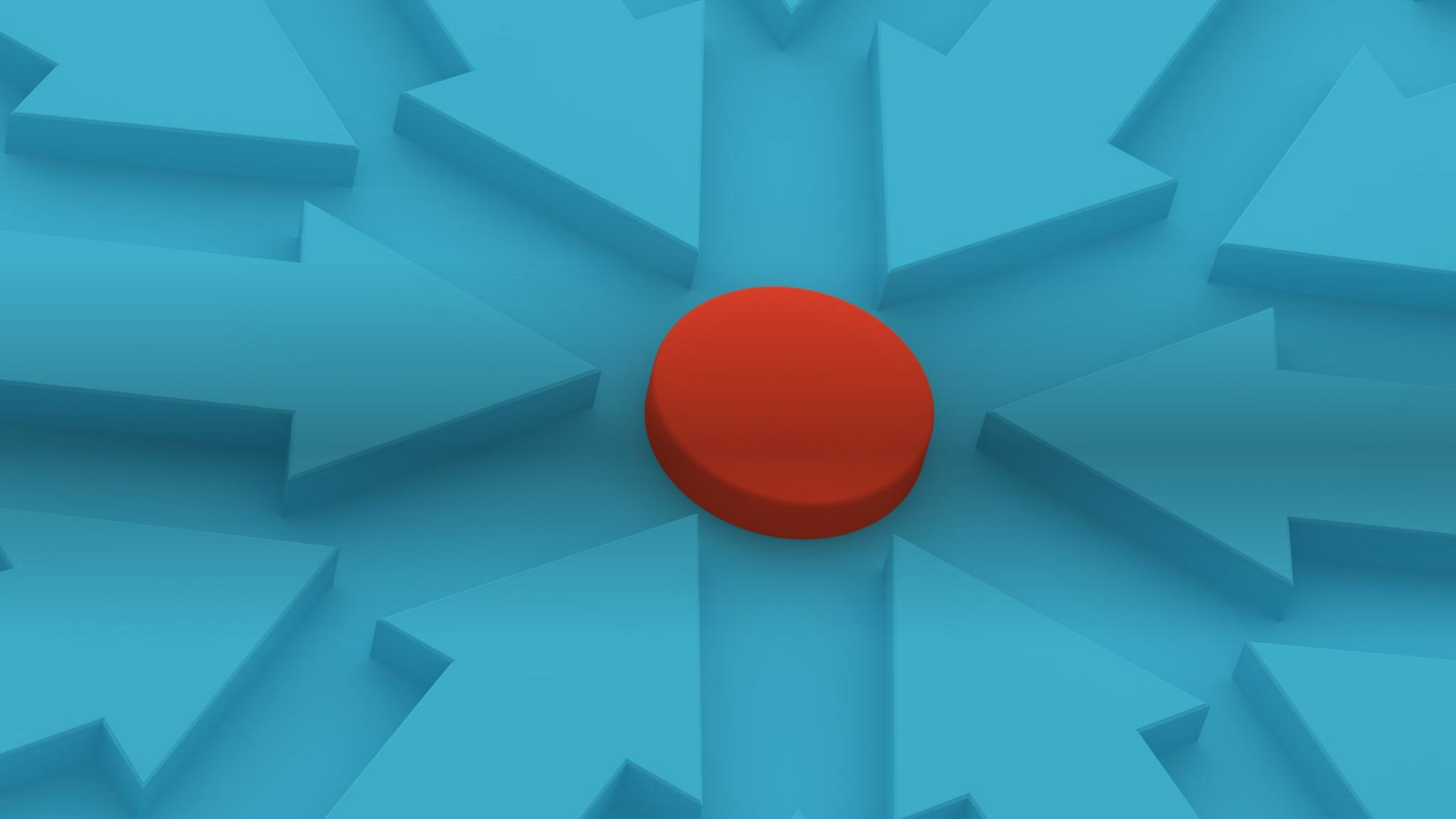


DEPLOY

Artifacts

Confidence





The image features a dense crowd of stylized human figures, rendered in a 3D, blocky aesthetic. Most figures are dark brown or black, while one figure in the center is a lighter, greyish-blue color. This central figure has its arms raised in a 'V' shape, standing out from the rest of the crowd. The background is a dark, textured blue-grey, and the overall lighting is dim, creating a somber yet hopeful atmosphere.

Confidence

Release

Deploy to *Production*





Artifacts



Deploy

Process

Process - (How we build) - Artifacts



CI/CD
→ CD Foundation



Reproducible
Builds



Security
→ SLSA



Transparency
→ SBOM





Continuous
Delivery
Foundation



Learn

[Where to Start?](#)[Team culture](#)[Version Control](#)[Continuous integration](#)**[Continuous deployment](#)**[Continuous testing](#)[Software Supply Chain](#)[Configuration management](#)[Assess your current state](#)[Domain-specific practices](#)

Best Practices in Action

Repeatability

The deployment process must be repeatable across all stages of the pipeline. To achieve repeatability, values that are specific to an environment should be separated from the deployment tasks. This allows the logic of the deployment to remain consistent, while the values change according to the endpoint.

Automation to Reduce One-Off Scripting

Continuous deployment requires the ability to scale quickly. This means that the reliance on deployment scripts can impede scaling of your release process. To avoid the reliance on scripts, the process should include a set of reusable tasks, components and functions that can define a templated approach to deployments.

Environment Modeling

A logical view of your endpoints, their use, ownership and capabilities is essential for defining your release landscape and creating a reference for automated deployments. Reporting on the Environment configurations is required for abstracting the differences between any two environments - a process required for debugging when a deployment does not perform as expected based on metrics defined in a previous environment.

[View page source](#)[Edit this page](#)[Create child page](#)[Create](#)[documentation issue](#)[Print entire section](#)[Repeatability](#)[Automation to Reduce](#)[One-Off Scripting](#)[Environment Modeling](#)[Approval and Approval Gates](#)[Release Coordination and Auditing](#)[Inventory Tracking](#)[Calendar and Scheduling](#)[Immutable Deployments](#)[Deployment Models](#)[Push Vs. Pull](#)[Policies](#)

Tag Cloud

[assess 1](#)[learn 1](#)

Categories

[Assess 1](#)[Learn 1](#)

Continuous Deployment

Repeatability

**Automation to
Reduce One-
Off Scripting**

**Environment
Modeling**

**Approval and
Approval
Gates**

**Release
Coordination
and Auditing**

**Inventory
Tracking**

**Immutable
Deployments**

**Deployment
Models**





Continuous Deployment



Repeatability

Deployment process repeatable across all stages of the pipeline



Automation to Reduce One-Off Scripting

Ability to scale quickly. Set of reusable tasks, components and functions in templates



Environment Modeling

Logical view of endpoints, use, ownership and capabilities



Approval and Approval Gates

Restricting access to lifecycle. Notification and approval that a new release



Release Coordination and Auditing

Audit log (who, when and where) an update occurred.



Inventory Tracking

Recording location of any artifact deployed to any location in an environment.



Immutable Deployments

All release metadata to and logic to be maintained in an immutable state



Deployment Models

Canary deployments, blue/green deployments and rolling blue/green deployments



Repeatable vs Reproducible

Artifacts (Reproducible Builds)



A build is **reproducible** if given the same source code, build environment and build instructions, any party can recreate bit-by-bit identical copies of all specified artifacts.



Reproducible builds are a set of software development practices that create an independently-verifiable path from source to binary code. ([more](#))



Reproducible Build Maven Plugin

[Reproducible Build Maven Plugin](#) / Introduction

```
$!18n.getString( "site-renderer", $locale, "template.lastpublished" ): $dateValue  
$!18n.getString( "site-renderer", $locale, "template.version" ): 0.16
```

OVERVIEW

Introduction

[Goals](#)[Usage](#)[FAQ](#)

PROJECT

DOCUMENTATION

Project Information

[Dependencies](#)[Dependency
Information](#)[Distribution
Management](#)

About

[Issue](#)[Management](#)[Licenses](#)[Plugin](#)[Management](#)[Plugins](#)[Source Code](#)[Management](#)[Summary](#)[Team](#)[Project Reports](#)

Reproducible Build Maven Plugin

Have you ever tried to compile twice the same sources with Maven and compared the hashes of the generated artifacts? They are not the same! Maven is not able to build an artifact in a real reproducible (i.e. byte-for-byte) way.

NOTE: Recent versions of the main Maven plugins have been modified to allow reproducible builds without the use of this plugin. See [Configuring for Reproducible Builds](#) for more details.

This Maven plugin tries to strip "non reproducible" data from the generated artifacts. It follows the same goals as [Debian's Reproducible Builds project](#) but at the modest scale of a Maven project. You can also have a look at my Devovx France 2016 talk "[Bit-for-bit reproducible builds with Maven](#)".

Using this plugin is a no-brainer: simply add it to your pom and it will try to "automagically" make the build byte-for-byte reproducible.

NOTE: This plugin requires Java 8 or later.

Goals Overview

There are several goals:

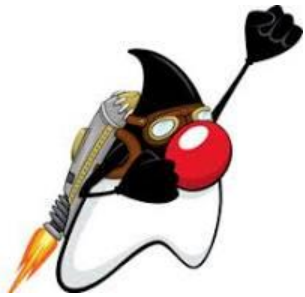
- The "strip-jar" goal processes all the ZIP/JAR/WAR/EAR files found in the target repository and does the following things:
 - sorts ZIP entries by name,
 - replaces file timestamps in ZIP entries with a fixed value,
 - removes timestamps, user names and tool versions in MANIFEST.MF,
 - removes comments in pom.properties file (some of them can contain time/date).

NOTE: As of version 0.5, the "strip-jar" goal also processes TAR/TAR.GZ/TAR.BZ2 files.

- The "strip-jaxb" goal normalizes ObjectFactory.java files generated by the JAXB xjc tool (before JAXB 2.2.11, xjc generates ObjectFactory.java files where the methods are put in a non-predictable order, which produces non-reproducible class files. Cf. issue [JAXB-598](#)).

If you find other interesting sources of "non reproducible" data that this plugin could remove, please open a ticket in the bug tracker or send a pull request.

Please note that you must use the same compiler (and possibly Maven) version to get the same results. You can use the maven-enforcer-plugin for that, or use a tool like [moot](#) to download and use the JDK and Maven versions people need to use for the build.



Maven™



Reproducible Builds for Maven Central Repository

This project is the [Java part](#) of the [Reproducible Builds](#) effort:



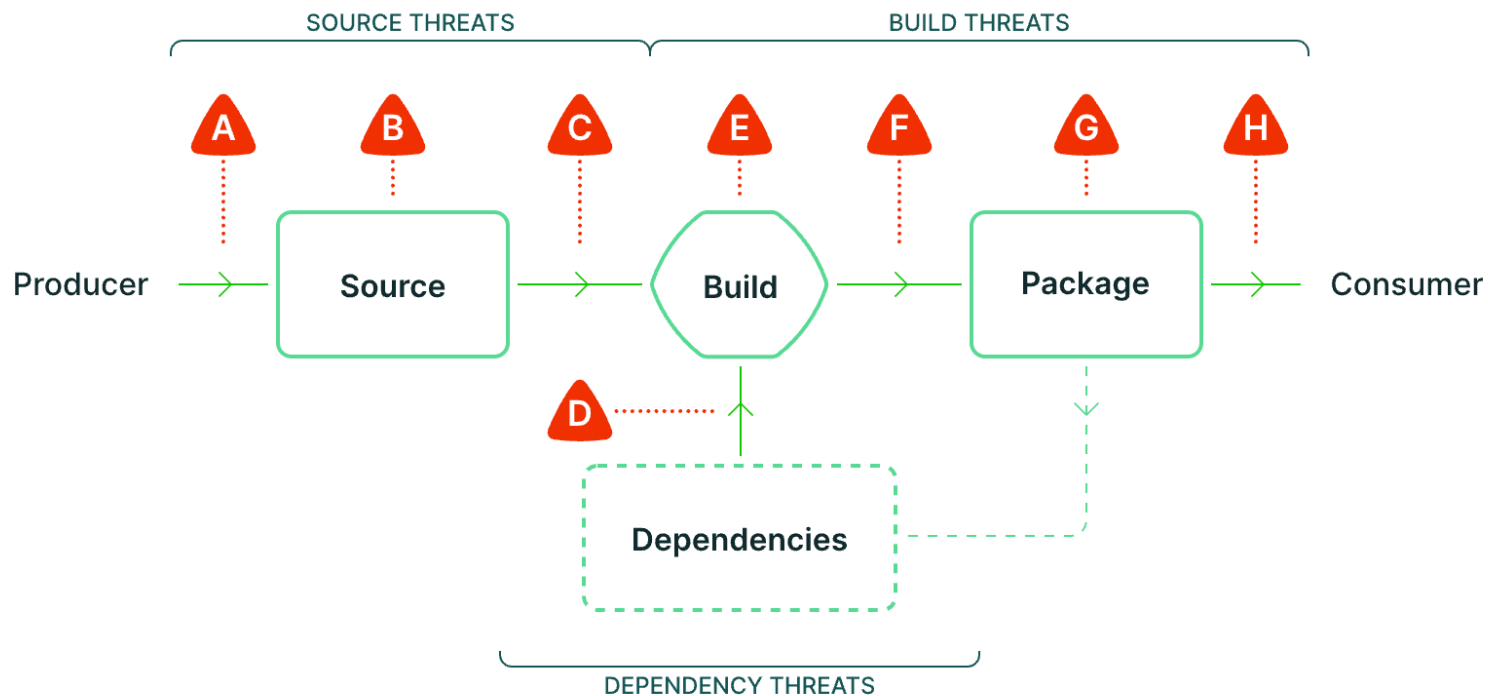
Its objectives are to provide:

- [Tools and methods allowing to verify that Java builds are reproducible](#)
- [A list of reproducible releases published to Maven Central](#)
rebuilding **4042** releases of **680** projects:
 - 3217 releases are confirmed **fully reproducible** (100% reproducible artifacts ✓),
 - 825 releases are only partially reproducible (contain some unreproducible artifacts ⚠)
 - on 680 projects, 579 have at least one fully reproducible release, 101 have none

Rebuild Detailed Results

Central Repository groupId	artifactId(s)	versions	re: reproducible
biz.aQute.bnd	bnd-plugin-parent	9	9 ✓
ch.qos.logback	logback-parent	23	16 ✓ /
ch.qos.reload4j	reload4j	8	1 ✓ / 7
ch.qos.logback.db	logback-parent-db	1	1 ⚠
ch.vorburger.mariaDB4j	mariaDB4j	1	1 ✓
com.flowlogic	flowlogic	14	10 ✓ /





SOURCE THREATS

- A** Submit unauthorized change
- B** Compromise source repo
- C** Build from modified source

DEPENDENCY THREATS

- D** Use compromised dependency

BUILD THREATS

- E** Compromise build process
- F** Upload modified package
- G** Compromise package registry
- H** Use compromised package



Safeguarding artifact integrity across any software supply chain



Supply-chain Levels for Software Artifacts

It's a security framework, a checklist of standards and controls to prevent tampering, improve integrity, and secure packages and infrastructure. It's how you get from "safe enough" to being as resilient as possible, at any link in the chain.



Build Level 0 (No guarantees)

- This level doesn't have any requirements nor provides any guarantees.
- This level indicates a lack of SLSA.
 - Intended for software development or test builds



Build Level 1 (Provenance exists)

- Organizations are required to maintain a reproducible and consistent build environment at Build L1.
- Tracking environment modifications and documenting the build environment are necessary.
- The build environment and inputs must also be identical for the build process to be replicated.



Build Level 2 (Hosted build platform)

- Requires tamper protection, which includes the use of version control and a hosted build service to generate provenance.
- The build platform runs on a dedicated infrastructure, not an individual's workstation, and the provenance is tied to that infrastructure through a digital signature.



Build Level 3 (Hardened builds)

- Build and source platforms meet auditing standards and maintain the integrity of the provenance.



SBOM

- SBOMs (software bill of materials) are structured documents that inventory the software components
- May communicate information about software licenses, vulnerabilities, and other metadata relevant to software transparency.



Table 1: SBOM Type Definition and Composition

SBOM Type	Definition	Data Description
Design	SBOM of intended, planned software project or product with included components (some of which may not yet exist) for a new software artifact.	Typically derived from a design specification, RFP, or initial concept.
Source	SBOM created directly from the development environment, source files, and included dependencies used to build an product artifact.	Typically generated from software composition analysis (SCA) tooling, with manual clarifications.
Build	SBOM generated as part of the process of building the software to create a releasable artifact (e.g., executable or package) from data such as source files, dependencies, built components, build process ephemeral data, and other SBOMs.	Typically generated as part of a build process. May consist of integrated intermediate Build and Source SBOMs for a final release artifact SBOM.
Analyzed	SBOM generated through analysis of artifacts (e.g., executables, packages, containers, and virtual machine images) after its build. Such analysis generally requires a variety of heuristics. In some contexts, this may also be referred to as a "3rd party" SBOM.	Typically generated through analysis of artifacts by 3rd party tooling.
Deployed	SBOM provides an inventory of software that is present on a system. This may be an assembly of other SBOMs that combines analysis of configuration options, and examination of execution behavior in a (potentially simulated) deployment environment.	Typically generated by recording the SBOMs and configuration information of artifacts that have been installed on systems.
Runtime	SBOM generated through instrumenting the system running the software, to capture only components present in the system, as well as external call-outs or dynamically loaded components. In some contexts, this may also be referred to as an "Instrumented" or "Dynamic" SBOM.	Typically generated from tooling interacting with a system to record the artifacts present in a running environment and/or that have been executed.



**CYBERSECURITY &
INFRASTRUCTURE
SECURITY AGENCY**





ISSN 1831-9424

Cyber Resilience Act Requirements Standards Mapping

Joint Research Centre & ENISA Joint Analysis



Cyber Resilience Act

2024-04-04

- Manufacturers of the products with digital elements shall:
 - Identify and document vulnerabilities and components contained in the product
 - **Software Bill Of Materials**

All Systems Operational

Documentation

Search

Getting started

Introduction

Sonatype OSSRH (OSS Repository Hosting) uses [Sonatype Nexus Repository Manager](#) to provide repository hosting service for open source project binaries - be sure to review the full [terms of service](#). OSSRH uses the Maven repository format and allows you to:

- deploy development version binaries (snapshots)
- stage release binaries
- promote release binaries and sync them to the Central Repository

Once released/published, you will not be able to remove/update/modify that artifact

We provide the option to publish artifacts using the `-SNAPSHOT` suffix in case that you need to do any test on your publishing process, but once it is released there is no possibility to change it. Please check the [Can I change \(modify, delete, update\) a component on Central?](#) FAQ for more details.

The initial setup for your OSSRH repository requires some manual steps and human review ([see why](#)), after which your deployment process is typically modified to get components into OSSRH. **These are all one time steps.**


After the initial setup, publishing from OSSRH to Central is a trivial action which can be done via a browser or programmatically.

Initial Setup

Review Requirements

Table of contents

- Introduction
- Initial Setup
 - Review Requirements
 - Deployment
- Releasing to Central
- OSSRH Usage Notes
 - Accessing Repositories



Ask Central



[← BACK TO BLOG](#)

How to publish a Java library to Maven Central - Complete Guide

November 7, 2022

[JAVA](#)[MAVEN](#)[JRELEASER](#)[GUIDE](#)

This is an opinionated **step-by-step guide** on how to **publish Java library to Maven Central** repository.

It assumes that:

- the project is built with **Maven** (small modifications would be needed for Gradle)
- the project code is hosted on **GitHub** and **GitHub Actions** are used to trigger the release

All sample Maven invocations use [Maven Wrapper](#) (`./mvnw`) - either generate wrapper files with `mvn wrapper:wrapper` or just remember to use `mvn` instead of `./mvnw` when following this tutorial

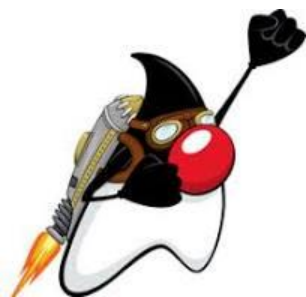
It uses [JReleaser](#) - I believe this is the simplest and the most straightforward way of signing and uploading artifacts.

This guide is based on [Official guide from Sonatype](#) but thanks JReleaser some steps are either heavily simplified or completely eliminated.

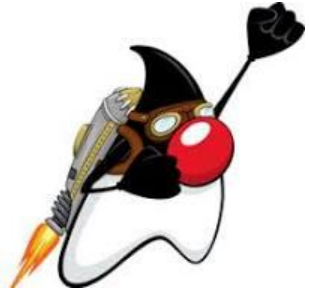
1. [1. Create an account in Sonatype JIRA](#)
2. [2. Create a "New Project" ticket](#)
 1. [If a custom domain is used as a group id](#)
 2. [If GitHub is used as a group id](#)

On this page

1. Create an account in Sonatyp...
- Create a "New Project" ticket
 - If a custom domain is used a...
 - If GitHub is used as a group id
- Set ticket to "Open"
3. Create GPG keys
 - 3.1 Export key to a key server
4. Export public and secret key ...
 - 4.1. Create GitHub secrets with UI
 - 4.2. Create secrets with GitHub...
5. Adjust pom.xml
 - 5.1. Generate javadocs and sour...
 - 5.2 Configure JReleaser Maven ...
6. Create a GitHub action
7. Get familiar with Sonatype Ne...
8. When is the library actually a...
- Conclusion



Maven™



This guide is based on [Official guide from Sonatype](#) but thanks JReleaser some steps are either heavily simplified or completely eliminated.

1. [1. Create an account in Sonatype JIRA](#)
2. [2. Create a "New Project" ticket](#)
 1. [1. If a custom domain is used as a group id](#)
 2. [2. If GitHub is used as a group id](#)
 3. [3. Set ticket to "Open"](#)
3. [3. Create GPG keys](#)
4. [3.1 Export key to a key server](#)
5. [4. Export public and secret key to GitHub secrets](#)
6. [4.1. Create GitHub secrets with UI](#)
7. [4.2. Create secrets with GitHub CLI](#)
8. [5. Adjust pom.xml](#)
9. [5.1. Generate javadocs and sources JARs](#)
10. [5.2 Configure JReleaser Maven Plugin](#)
11. [6. Create a GitHub action](#)
12. [7. Get familiar with Sonatype Nexus UI](#)
13. [8. When is the library actually available to use?](#)
14. [Conclusion](#)

1. Create an account in Sonatype JIRA

[Sign up in Sonatype JIRA.](#)



Build Tools



MavenTM



JReleaser

Git Services



Release

Packagers



Package

Announce

Announce Channels



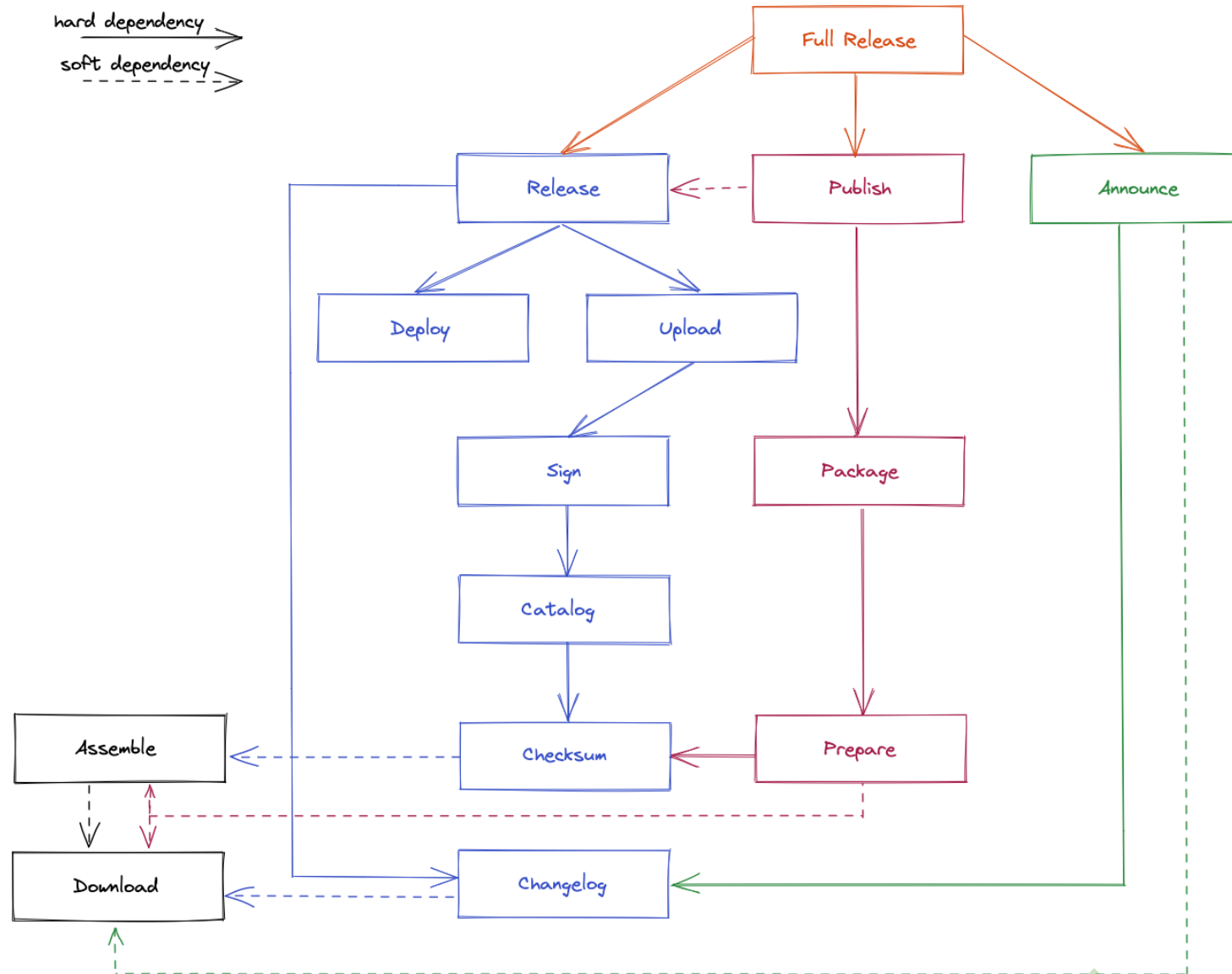
JReleaser

Release automation tool.

Simplify creating releases and publishing artifacts to multiple package managers.

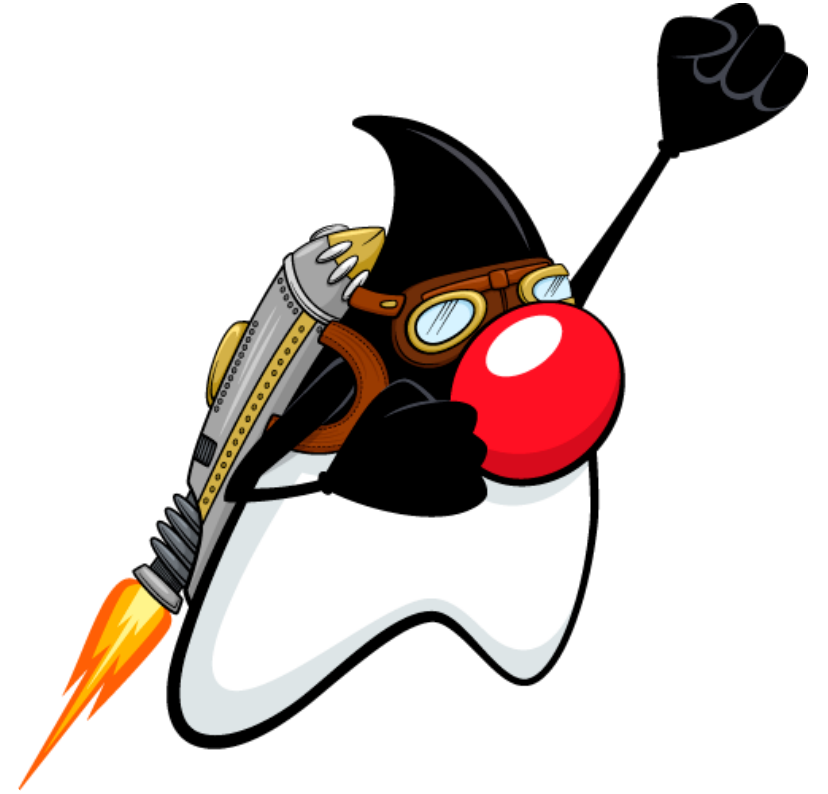


hard dependency →
soft dependency - - ->



JReleaser

- **Create** Git releases (tag, changelog, assets),
- **Announce** releases,
- **Assemble** additional binaries and files
- **Publish** to several package managers





latest ▾

Download

Files may be downloaded from the following services:

- [Ftp](#)
- [Generic HTTP/HTTPS](#)
- [Scp](#)
- [Sftp](#)

Assemble

Distributions may be assembled using your build tool of choice, also with any of the following assemblers:

- [Archive](#)
- [Java Archive](#)
- [Jlink](#)
- [Jpackage](#)
- [Native Image](#)

Release

Releases may be posted to the following services:

- [GitHub](#)
- [GitLab](#)
- [Gitea](#)



latest ▾

- [Codeberg](#)
- [Generic git](#)

! IMPORTANT

The generic releaser does not support all features.

Catalog

Assembled artifacts, distribution artifacts, and files may be cataloged:

SBOM

- [CycloneDX](#)
- [Syft](#)

Provenance

- [Github Attestation](#)
- [SLSA](#)

Tracking

- [SWID](#)

Deploy

Staged artifacts may be deployed to the following services:

Maven

- [Artifactory](#)
- [Azure](#)





latest ▾

- [Gitea](#)
- [Gitlab](#)
- [Github](#)
- [MavenCentral](#)
- [Nexus2](#)

Upload

Artifacts, checksums, signatures may be uploaded to the following services:

- [Artifactory](#)
- [Ftp](#)
- [Gitea](#)
- [GitLab](#)
- [Generic HTTP/HTTPS](#)
- [AWS S3](#)
- [Scp](#)
- [Sftp](#)

Package & Publish

Distributions may be packaged and published with the following tools:

- [ApplImage](#)
- [Asdf](#)



latest ▾

- [Asdf](#)
- [Chocolatey](#)
- [Docker](#)
- [Flatpak](#)
- [Homebrew](#)
- [JBang](#)
- [Jib](#)
- [Macports](#)
- [Scoop](#)
- [Sdkman](#)
- [Snap](#)
- [Spec](#)
- [Winget](#)

Announce

Releases may be announced using the following tools and communication channels:

- [Article](#)
- [Bluesky](#)
- [Discord](#)
- [Discourse](#)
- [GitHub Discussions](#)
- [Gitter](#)






Announce


Releases may be announced using the following tools and communication channels:

- [Article](#)
- [Bluesky](#)
- [Discord](#)
- [Discourse](#)
- [GitHub Discussions](#)
- [Gitter](#)
- [GoogleChat](#)
- [Linkedin](#)
- [Http](#)
- [Mastodon](#)
- [Mattermost](#)
- [OpenCollective](#)
- [Sdkman](#)
- [Slack](#)
- [Sntp](#)
- [Teams](#)
- [Telegram](#)
- [Twitter](#)
- [Webhooks](#)
- [Zulip](#)















 sigstore

[Overview](#) [Community](#) [How sigstore works](#) [Trust and security](#) [Blog](#) [Docs](#) 

sign. verify. protect.

Making sure your software is what it claims to be.

In collaboration with

 OpenSSF  Chainguard  CISCO   Google  Hewlett Packard Enterprise  PURDUE UNIVERSITY  Red Hat  stacklok  vmware



We've combined a few technologies that can be used independently, or as one single process. It's a way for software developers to sign off on what they build, without needing to jump through hoops or know tricky security protocols. And it's a way for anyone using those releases to verify the signatures against a tamper-proof log.



Sign

Easy authentication and smart cryptography work in the background. Just push your code, sigstore can handle the rest.

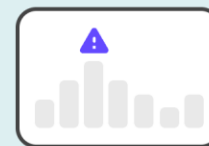
[Learn more](#)



Verify

Transparency logs store unique identification like who created it and where it was built, so you know it hasn't been changed when you verify.

[Learn more](#)



Monitor

Data stored in the logs is readily auditable, a foundation for future monitors and integrations to build into your security workflow.

[Learn more](#)



What's behind the scenes?

Cosign

For signing and verification of artifacts and containers, with storage in an Open Container Initiative (OCI) registry, making signatures and in-toto/SLSA attestations invisible infrastructure.

[View the repo](#) 

Policy Controller

Policy Controller is used to enforce policy on a cluster on verifiable supply-chain metadata from Cosign.

[View the repo](#) 

Fulcio

Code-signing certificate authority, issuing short-lived certificates to an authenticated identity and publishing them to a certificate transparency log.

[View the repo](#) 

Rekor

Append-only, auditable transparency log service, Rekor records signed metadata to a ledger that can be queried, but can't be tampered with.

[View the repo](#) 


OpenID Connect

An identity layer that checks if you're who you say you are. It lets clients request and receive information about authenticated sessions and users.

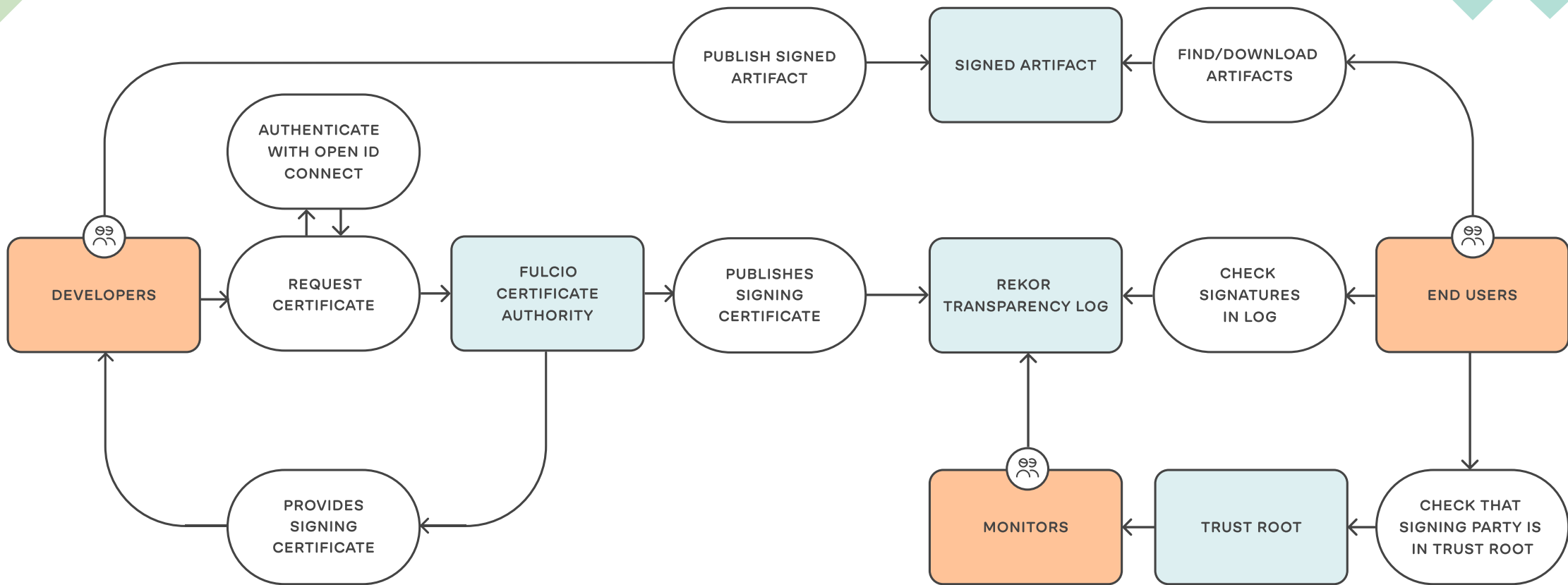
[Learn more](#) 

Trust root

The foundation for trust underpinning Sigstore utilizes TUF. This repository describes this process, our keyholders, and how the root keys are protected.

[Sigstore's trust root](#) 



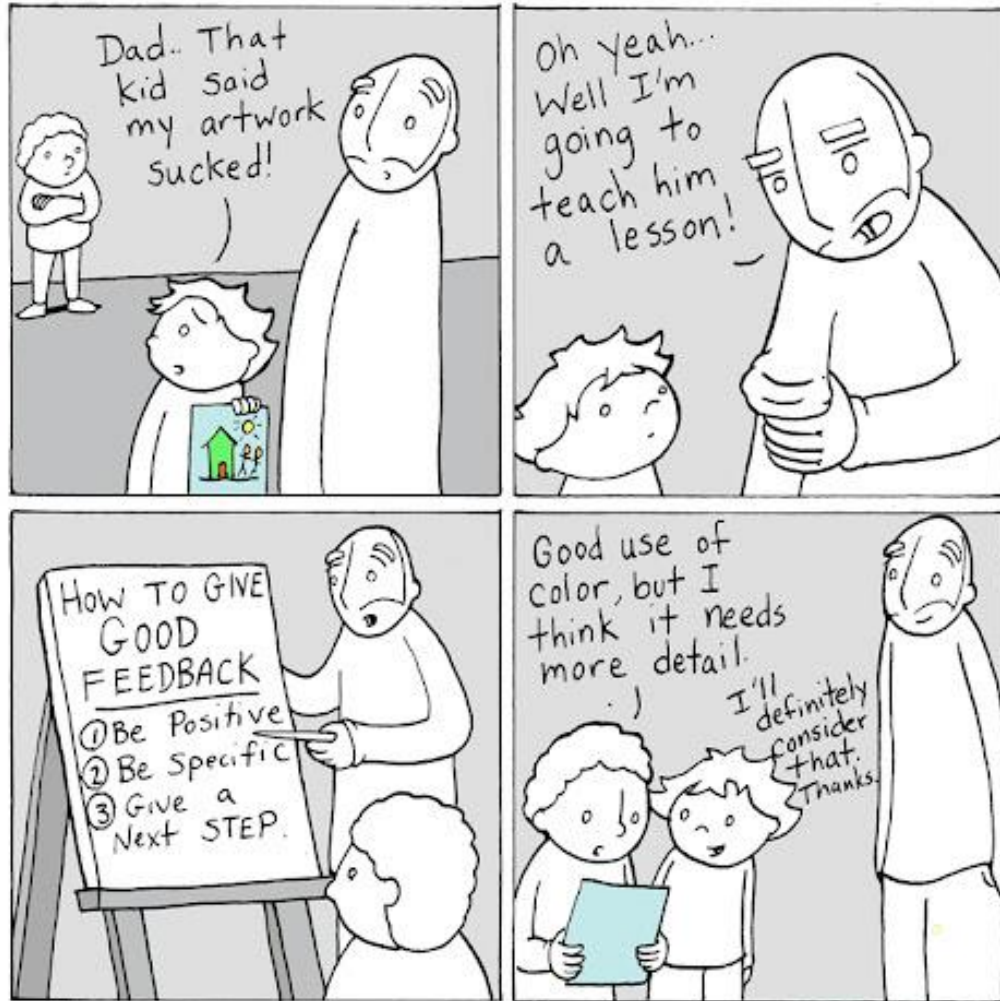


Artifacts (What has changed)

Changelog

- Keep the changelog up to date
- Show version and date in release header
- Group similar changes into categories
- Communicate relevant changes
- Highlight critical changes
 - Breaking changes
 - Binary incompatibility
 - Deprecations

→ Conventional Commits



www.lunarbaboon.com



[linkedin.com/in/ixchelruiz](https://www.linkedin.com/in/ixchelruiz)

[@ixchelruiz@mastodon.social](mailto:ixchelruiz@mastodon.social)

[@ixchelruiz.bsky.social](https://bsky.app/profile/ixchelruiz)

github.com/ixchelruiz