探索 Rust 和 C++ 在云原生下的制品管理

JFrog

王青
JFrog 中国技术总监

The Liquid Software Company

AGENDA

- 云原生环境下制品管理的挑战
- C++和Rust的包管理
- 使用Artifactory统一管理云原生制品

云原生环境下制品管理的挑战

1. 构建速度慢 -> 研发等待时间长
2. 依赖管理混乱 ->供应链攻击
3. 制品缺乏元数据信息->版本管理混乱

源码依赖的痛点

假设您的项目需要使用以下两个外部库：
Boost: Boost 是一个广泛使用的 C++ 库集合，提供了许多用于各种任务的工具和组件。
OpenCV: OpenCV 是一个开源计算机视觉库，用于处理图像和视频数据。

下载和解压缩：您需要前往 Boost 和 OpenCV 的官方网站，下载适用于您的操作系统的源代码压缩包，并手动解压缩。

配置和编译：对于每个库，需要手动设置编译选项等。可能需要为不同的平台和编译器配置不同的选项。

编译时间：编译大型库，可能需要很长时间。

依赖项处理：如果这些库本身还有依赖项，可能还需要手动下载和编译这些依赖项。

更新和维护：在将来，如果需要升级库的版本，您需要重新下载并手动执行上述步骤。

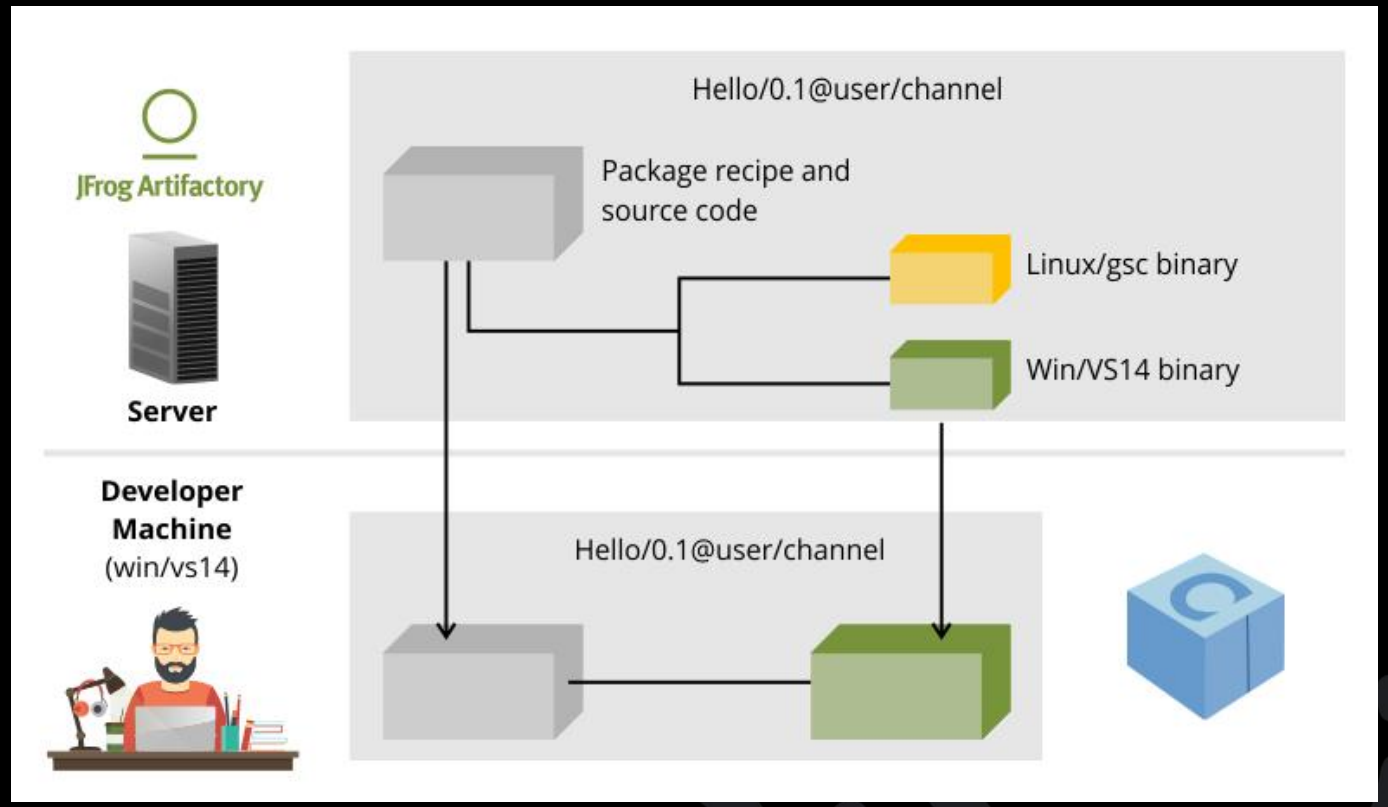
C++项目如何编译加速？ Conan.io

1.定义依赖
conanfile.txt
[requires]
zlib/1.2.11

[generators]
cmake

2. 执行Conan install
conan install . --output-folder=build

3. 构建
$ cd build
$ cmake .. -DCMAKE_TOOLCHAIN_FILE=conan_toolchain.cmake -DCMAKE_BUILD_TYPE=Release
$ cmake --build .



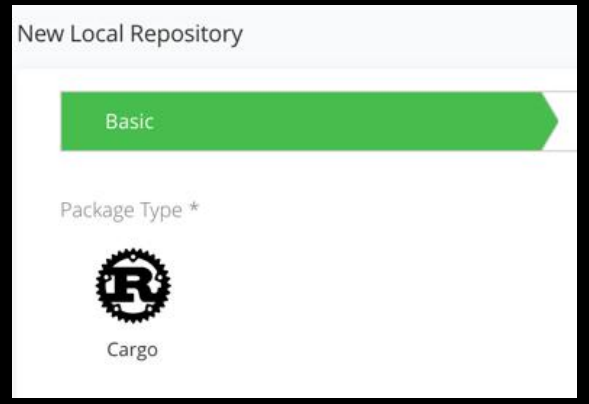
Rust 项目使用Cargo仓库拉取远程依赖

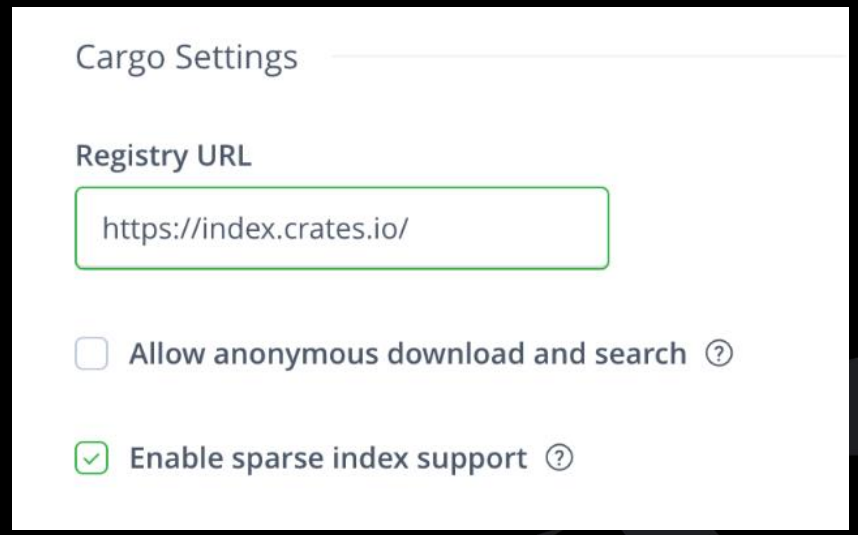
vi ~/.cargo/config.toml

```
# Makes artifactory the default registry and
saves passing --registry parameter
[registry]
default = "artifactory"

[registries.artifactory]
index =
"sparse+https://demo.jfrogchina.com/artifactor
y/api/cargo/cargo-local/index/"
```

cargo install <PACKAGE_NAME>





使用包管理工具带来的问题 - 供应链攻击

软件供应链攻击（Software Supply Chain Attack）是一种恶意行为，攻击者通过在软件开发、部署或分发过程中植入恶意代码或漏洞，以获取非法访问、窃取信息、传播恶意软件或实施其他恶意活动。

这种类型的攻击通常利用了软件开发和分发过程中的弱点，以获得攻击目标的控制权。

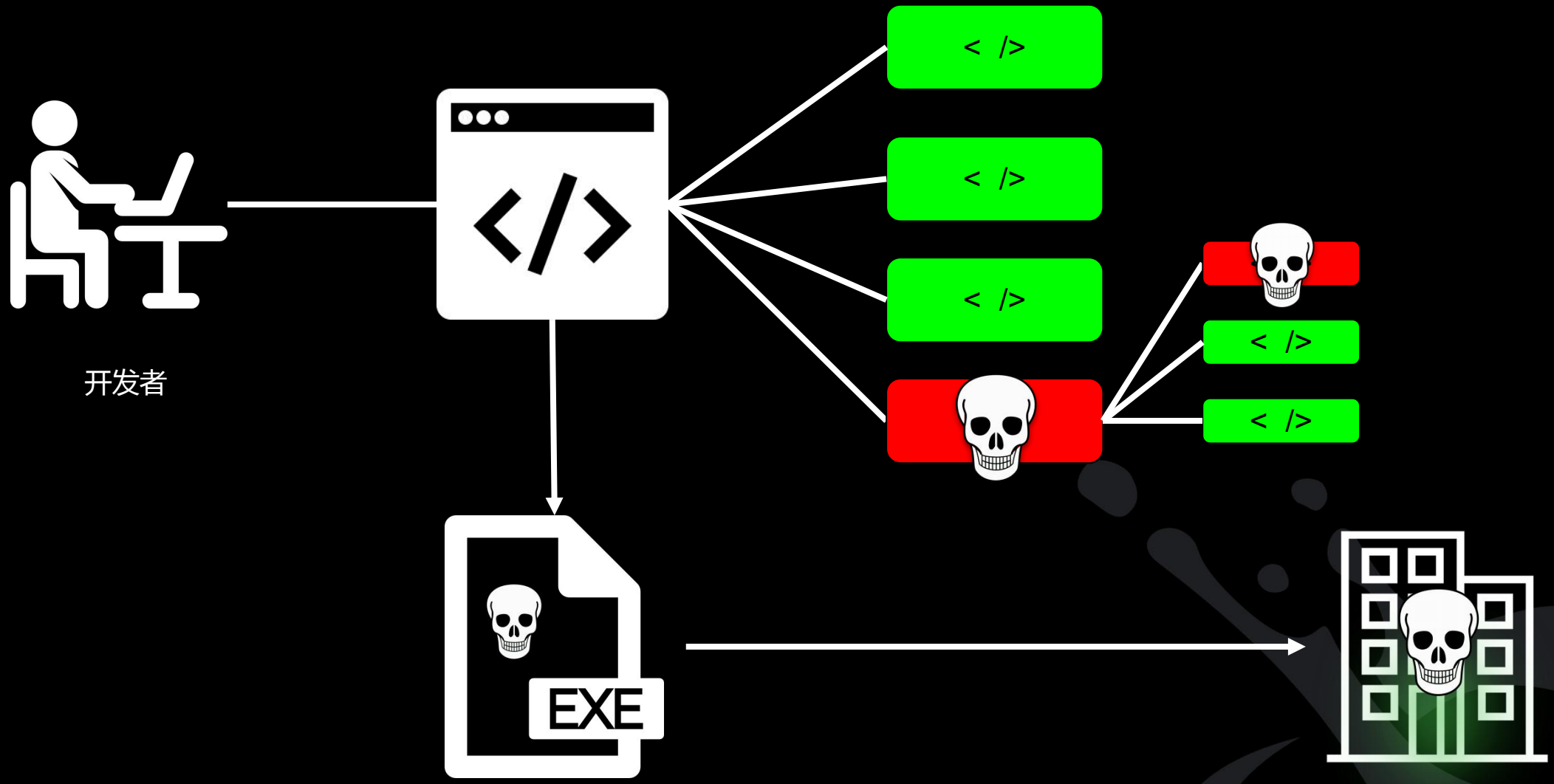
软件供应链攻击

为什么会有这种攻击方式？

1. 成本低，范围广
2. 伪装在授信的软件背后
3. 攻击者的代码潜伏在开源社区的代码或者或伪装成开源社区软件的版本

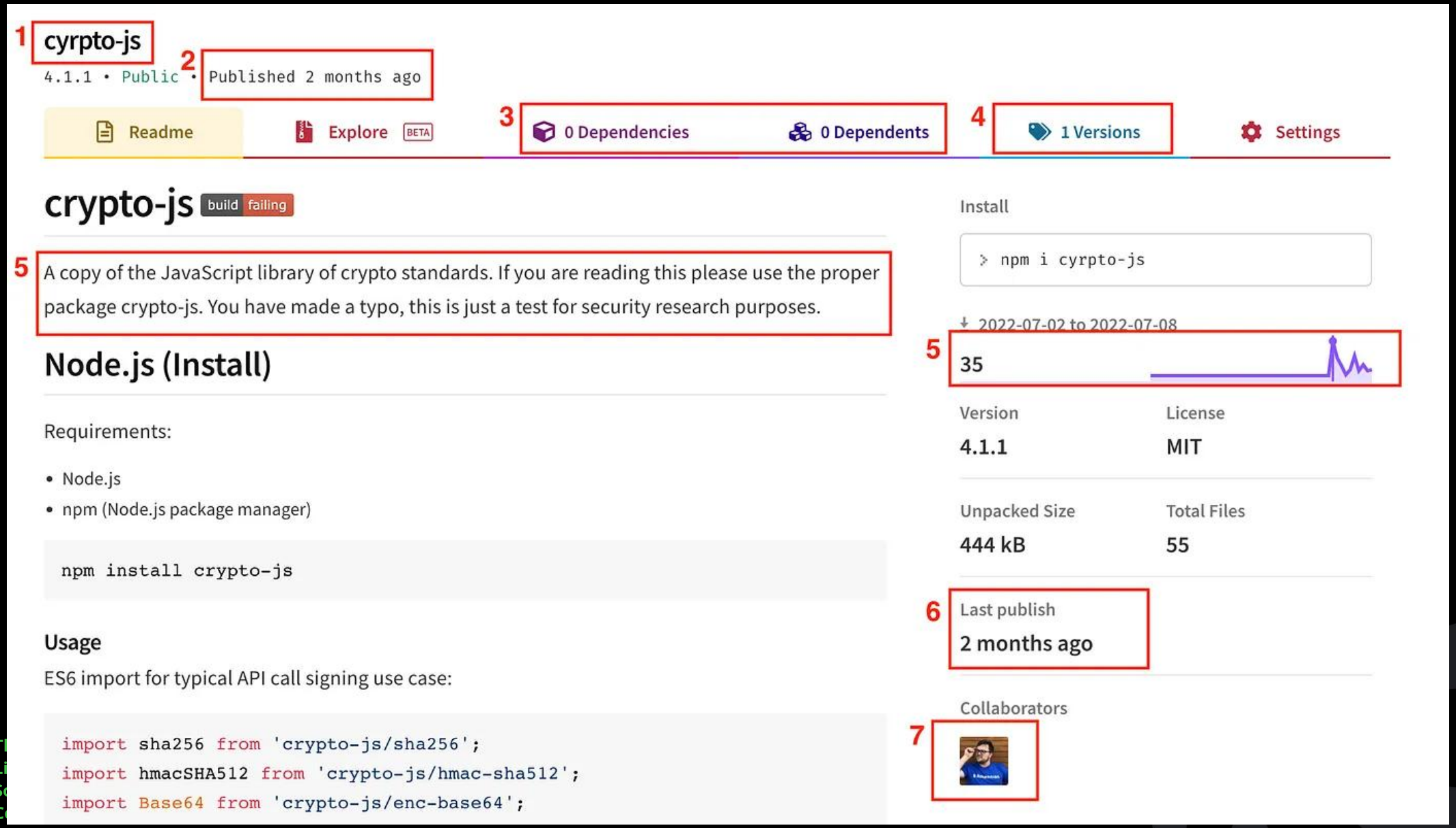
Malicious code

攻击是如何发生的？



The Liquid Software Company

DEPENDENCY TYPOSQUATTING 依赖误植



DEPENDENCY CONFUSION - 依赖混淆攻击

NPM依赖包的语法： ^3.0.0 := >=3.0.0 <4.0.0

1. 黑客上传版本com-bank-app:3.99.99到npm registry 仓库npmjs.com(任何人均可注册，或恶意抢注)
2. 用户请求： com-bank-app:^3.0.0 (私有版本)
3. 在本地仓库中找到版本为 3.2.4。
4. 在 npm-registry 代理远程仓库中找到版本为 3.99.99。
5. 来自 npm registry 的伪造 com-bank-app:3.99.99 获胜，供应链被劫持。

Docker镜像的安全问题

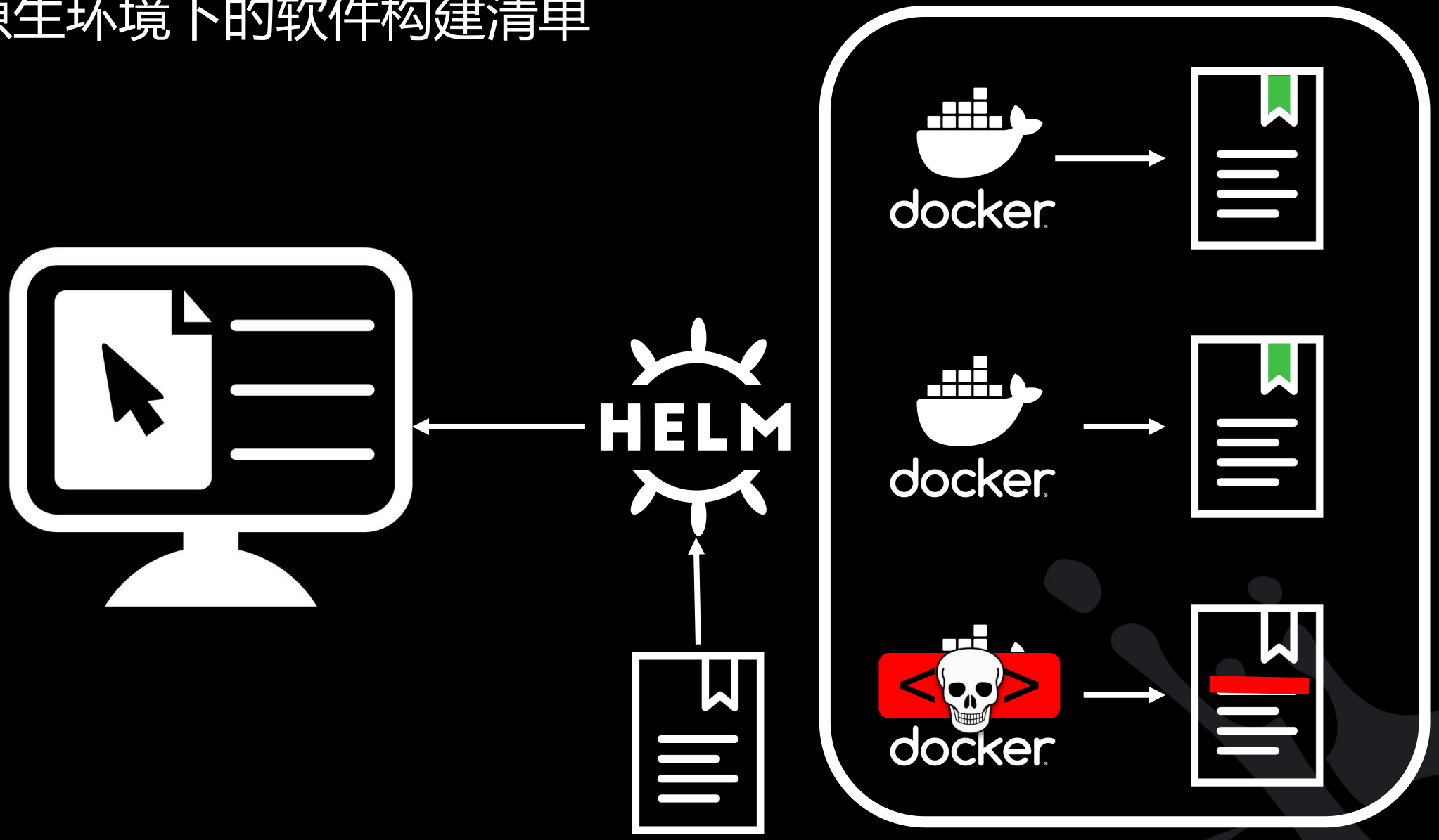
不安全的基础镜像： 使用来自不受信任或不安全来源的基础镜像可能会引入潜在的漏洞和恶意软件。建议使用官方和受信任的基础镜像，避免从未经验证的源拉取镜像。

过时的镜像和组件： 使用过时的镜像和组件可能会包含已知的漏洞和安全问题。定期更新镜像和依赖组件，以确保应用程序使用的是最新的和安全的版本。

不必要的软件和依赖项： 镜像中包含不必要的软件和依赖项可能会增加攻击面和漏洞风险。最小化镜像的大小，仅包含应用所需的组件和依赖项。

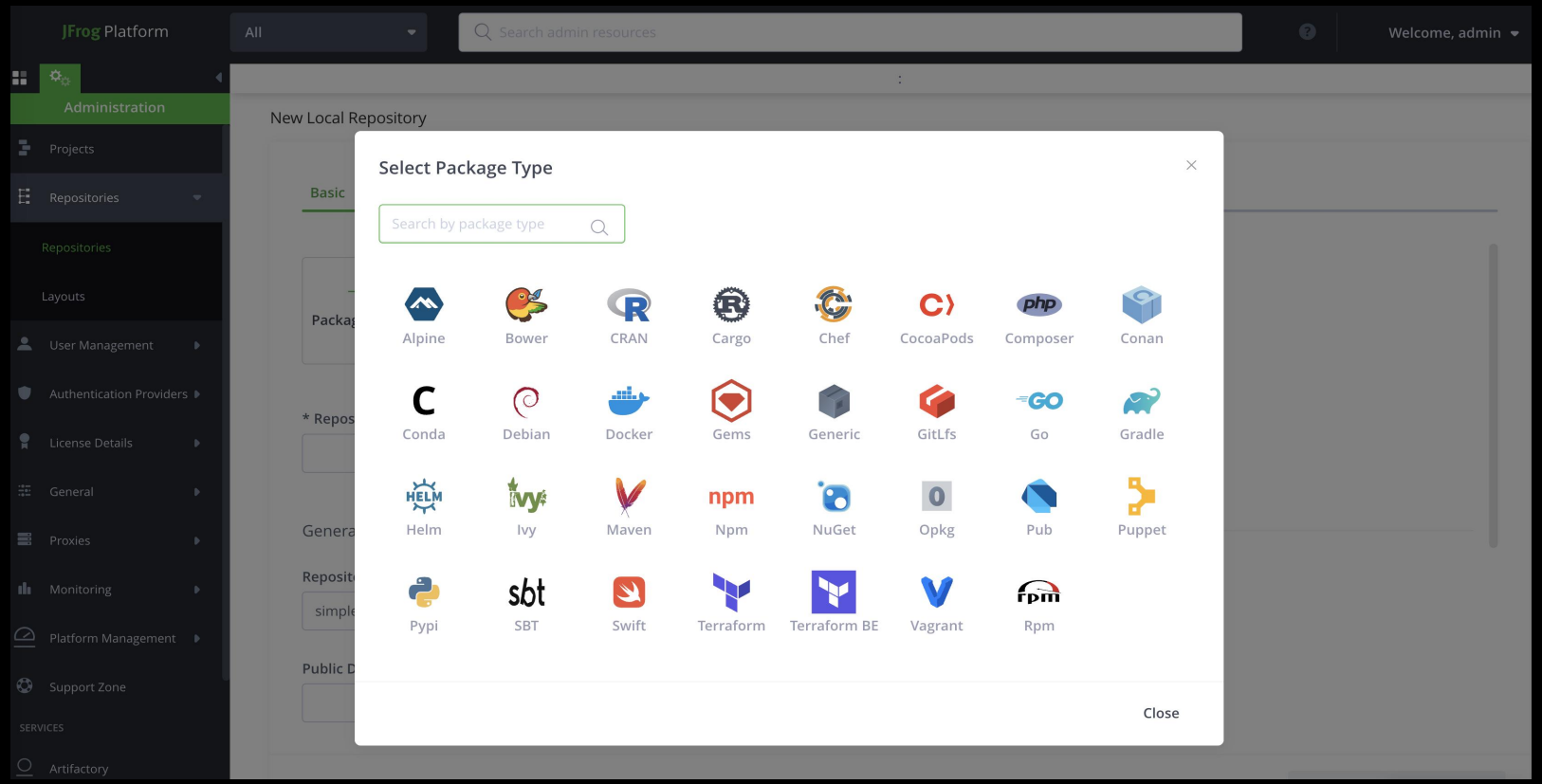
未加密的敏感数据： 在镜像中存储未加密的敏感数据可能会被攻击者访问。确保敏感数据在存储和传输过程中进行加密。

云原生环境下的软件构建清单

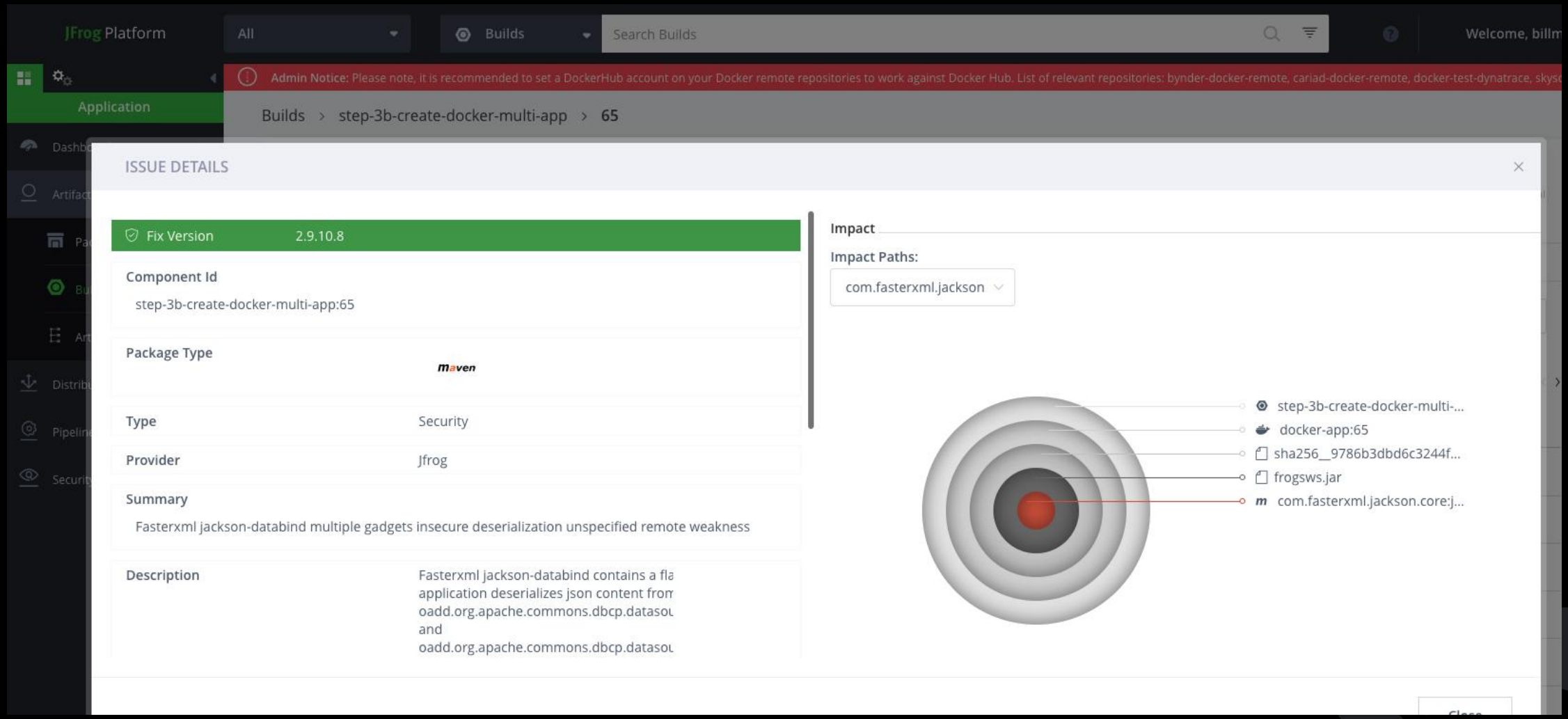


Artifactory管理全语言制品

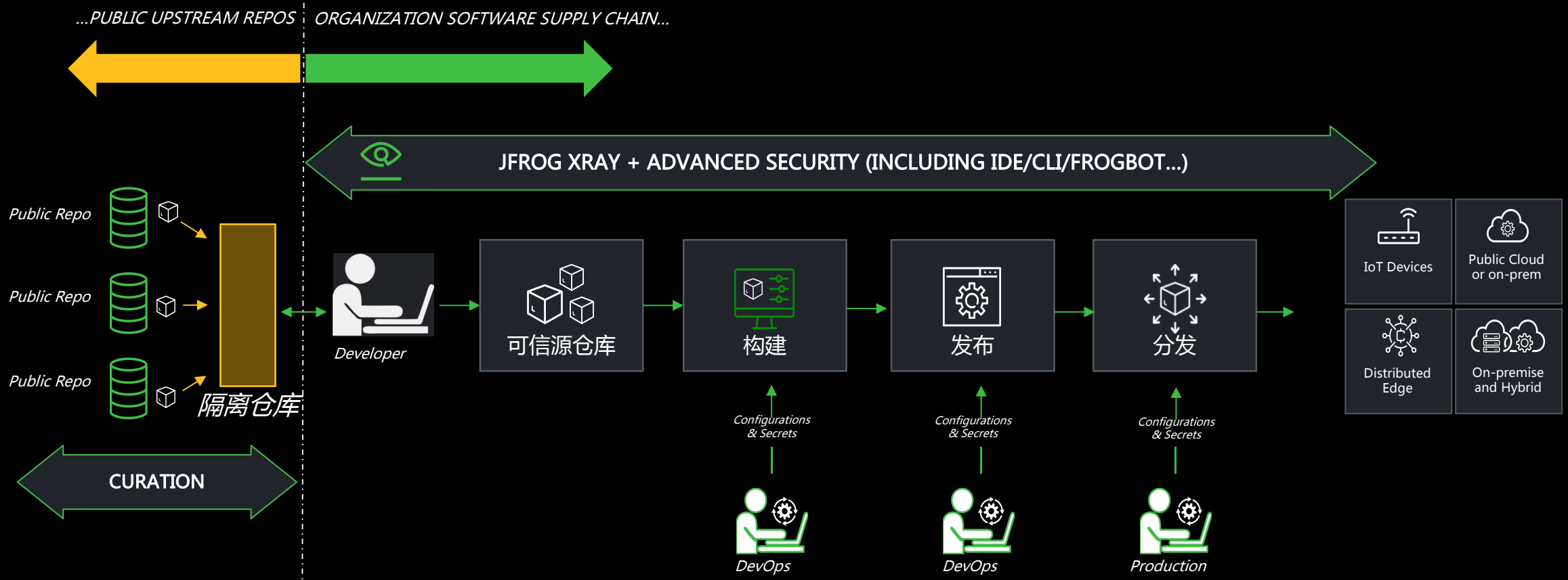
- 统一管理所有软件包
  - 三方库
  - 二方库
  - 版本库
- 开源组件漏洞扫描



依赖对应用的深度分析



JFrog Curation & Advanced Security



The Liquid Software Company

JFrog扫描了160万+开源组件

NEW PACKAGES
65K

NEW VERSIONS
500K

NEW PACKAGES
20K

NEW VERSIONS
100K

NEW PACKAGES
95K

NEW VERSIONS
750K

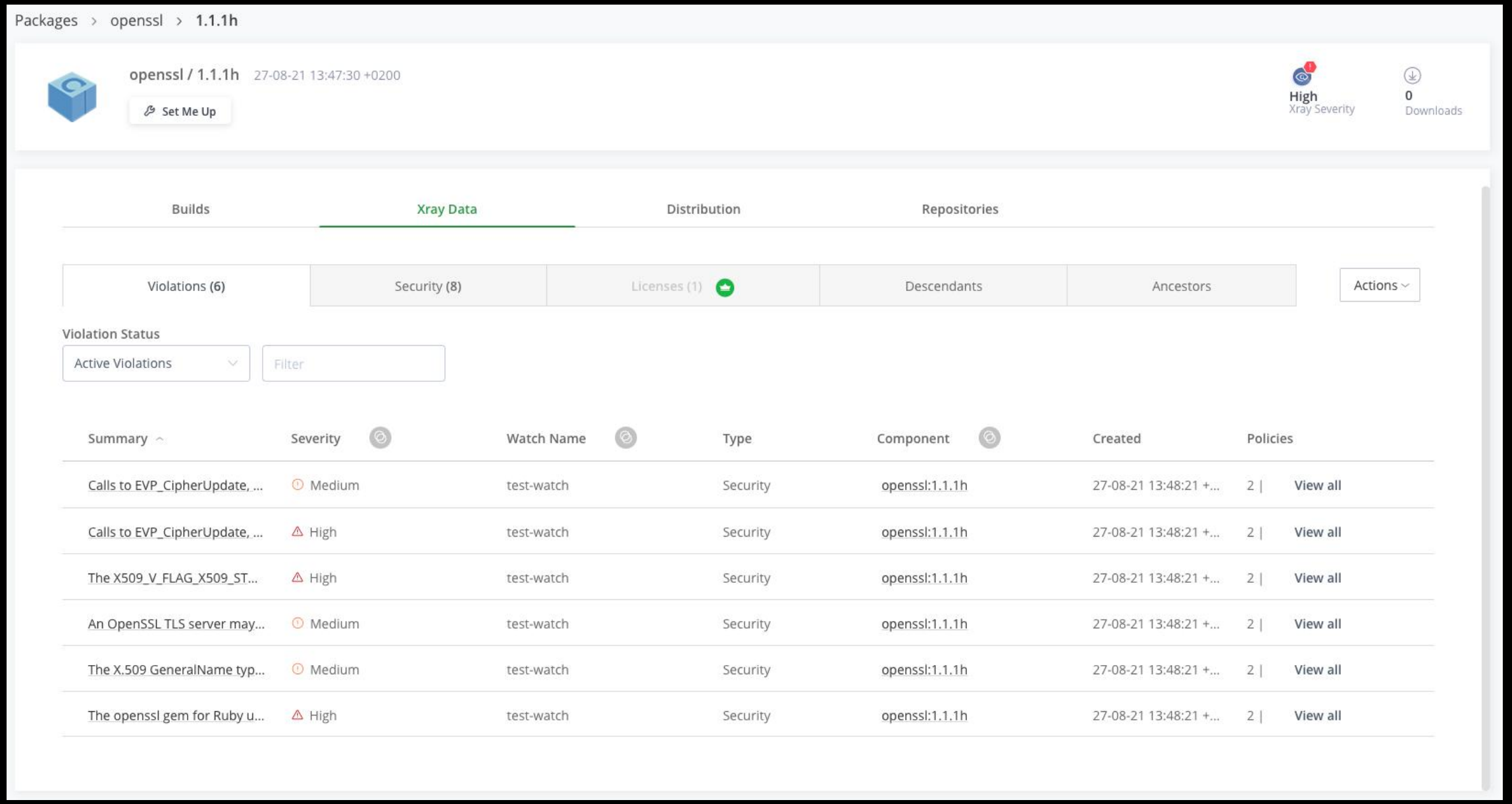
NEW PACKAGES
15K

NEW VERSIONS
125K

NEW PACKAGES
5K

NEW VERSIONS
200K

Openssl Conan包的漏洞



制品版本缺乏元数据的问题：

难以识别版本： 缺乏元数据可能使得很难识别制品的版本，增加了管理和追踪制品的复杂性。

无法验证来源： 元数据可以包含制品的来源、作者和数字签名等信息，来验证制品的可信度和完整性。缺乏这些元数据可能会导致不确定制品的真实性，可能会引入潜在的安全风险。

难以追踪变更历史： 缺少元数据可能使得很难追踪制品的变更历史，包括什么时间进行了哪些更改。这可能会阻碍了故障排查、版本回退和问题分析等活动。

难以自动化处理： 元数据对于自动化构建、部署和运维流程非常重要。缺乏元数据可能会阻碍自动化流程的执行，从而增加了手动干预的需求。

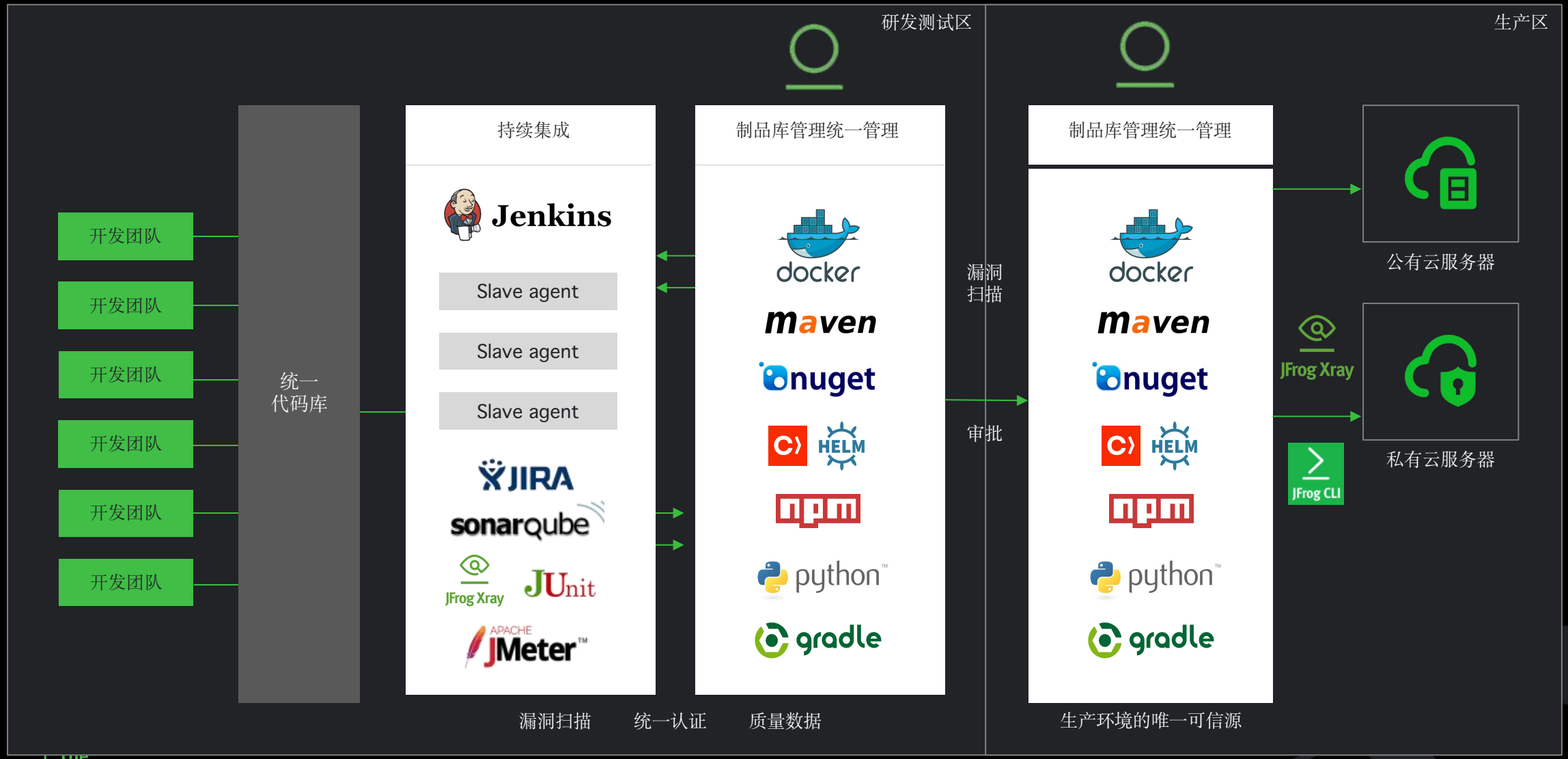
某金融企业制品晋级案例

The Liquid Software Company

制品元数据规范

元数据 \ 阶段	元数据名称	所属仓库	内容说明	创建日期	元数据key
元数据分类					
租户	租户简称			2021/11/17	tenant.abbr
	租户名称			2021/12/5	tenant.name
投产	实际投产类型			2021/11/17	release.type
	实际投产类型编码			2021/11/17	release.typeCode
	计划投产类型			2021/11/17	release.planType
	计划投产类型编码			2021/11/17	release.planTypeCode
	计划投产日			2021/11/17	release.planDate
	实际投产日			2021/11/17	release.date
	制品投产状态			2021/11/17	release.tag
	制品投产状态编码			2021/11/17	release.tagCode
	投产文件名			2021/11/17	release.fileName
物理子系统	物理子系统编码			2021/11/17	subSys.code
	物理子系统名称			2021/12/5	subSys.name
	物理子系统简称			2021/11/17	subSys.abbr

Artifactory - 企业软件唯一可信源



JFrog

The Liquid Software Company

王青

北京 朝阳

扫一扫上面的二维码图案，加我为朋友。

添加好友获得更多JFrog制品库使用案例