浅谈企业统一制品库的建设和管理

中金所技术公司 刘智勇

摘要：

本文首先介绍了制品库的概念和意义。通过阐述企业自身实践说明制品库几个重要特性。分别从“**多类型制品库**”，“**多来源制品库**”介绍了制品库的两大重要功能。在此基础上阐述了“**以制品库为枢纽中心的持续交付方法**”。最后从安全的角度重点介绍了网段隔离环境下的制品库运维和安全管控。

一、前言

制品是由源代码编译打包生成的二进制文件。一般来说，不同的开发语言对应着不同格式的二进制文件，这些二进制文件通常可以直接运行在机器上。**制品库**统一管理不同格式的软件制品。提供基本的制品托管、版本控制、访问控制、安全扫描、依赖分析等重要功能，是一种企业处理软件开发过程中产生的所有包类的标准化管控方式。

粗粒度的制品管理方式（FTP、NAS 或者 svn 等）会造成存储不清晰，制品获取困难，版本追踪混乱等问题，从而造成团队协作障碍，进而导致研发效率的低下。我司在引入统一制品库之前，正面临着以上种种困难。譬如：

- 开发环境包，开发工具包随意存储在 NAS 上，混乱且获取困难；
- 构建包和源代码混合打包到 svn 或者 fms 上，发布包巨大且管理困难；
- 缺乏便捷地获取三方包的途径，导致开发模式中难以引入 maven，npm，docker 等高效生产工具，java 项目采用传统的 lib 管理方式，效率极其低下。

以上种种问题，引入制品库之后都得到了很好的解决，受到了开发测试运维人员的好评。在此基础上，结合 devops 思想，逐步形成了以制品库为枢纽中心的持续交付方法。让研发团队真正做到“**deploy anytime anywhere**”。

二、多类型制品库

现代软件系统开发，面临着一个重要问题是二进制包种类变多。各种编程语言层出不穷。以我司举例，公司内部存在 C/C++，Java，python，C#，JavaScript，移动等多个开发团队。各个团队发布的二进制包千差万别，使用的构建流程涵盖 make，maven，gradle，ivy，pypi，nuget，npm，yarn 等多种构建工具。开发人员和运维员工面临着 Windows，Linux 的软件包安装困难的问题。特别是随着容器化的推动，镜像中心，helm-chart 存取又带来了新的挑战。开发运维人员基于自身实践独立搭建特定语言的制品库，解决了部分问题，但管理不善，而且带出来很多的烟囱系统。基于以上挑战，组织内对于建设企业多类型统一制品库都有着强烈的需求。图 1 阐述了我司多类型制品库的整体架构图。下面将对此逐一详细介绍。

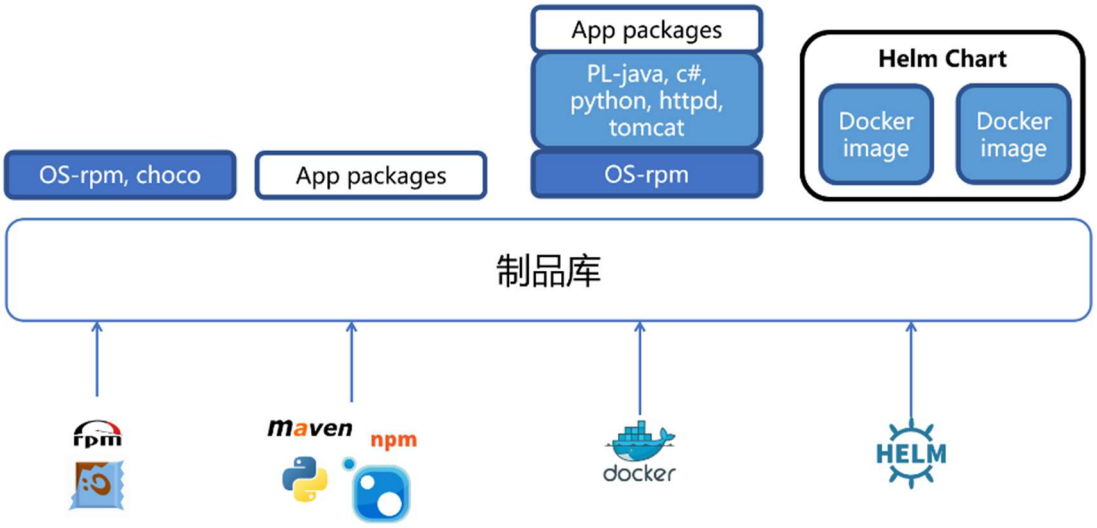

图 1. 多类型制品库

2.1 操作系统包

IaaS 平台或者说虚拟化平台（VMWare[1]或者 OpenStack[2]）解决了便捷获取各类操作系统问题。开发人员依托云管平台可以非常方便地虚拟出 CentOS，redhat，Windows7，windows Server 等一系列 OS 平台。然而如何便捷管理和获取操作系统的软件包并没有解决。制品库通过建立 yum-rpm[3]和 chocolatey[4]库分别解决了 Linux 和 Windows 下的包管理难题。

2.2 应用构建包

现代的软件开发愈来愈向共享共建的工厂化流水线模式演进。通过四面八方的小而美组件组装软件系统是大势所趋。譬如现代的 java 应用采用 maven[5]，gradle[6]来管理项目，javascript 应用采用 npm[7]，yarn[8]来管理项目，python 应用采用 pypi[9], c#应用采用 nuget[10]。基本每种语言都有自己的依赖构建管理工具，无论是 C/C++、C#、Java 类的传统语言，还是 R、Python、Go 这类型的新式语言。

企业统一制品库基本上包含了大多数语言的类型包管理。目前我司基本建设完成了 maven, npm, nuget, pypi 类型包管理机制。未来还会在此基础上进一步扩展。

2.3 容器与微服务应用包

容器化和微服务是当今软件行业的宠儿。愈来愈多的公司在其内部开始尝试容器化和微服务。采用 docker[11]技术来充当容器引擎是大多数公司的选择。在此背景下，制品库衍生出了镜像中心的概念，从而在镜像托管这一领域发挥了重大作用。

随着容器化的应用和微服务的实践，容器云集群应运而生。k8s[12]作为稳定高效的生产级别容器编排平台，也成为了大多数公司的容器云的集群的技术选择。如果管理 k8s 应用，业界给了 helm[13]的答案。因此，在 docker 镜像中心的基础上，制品库也承担了 helm-chart 包管理的功能。

2.4 其他类型包

上述三种类型包基本可以覆盖大多数的开发场景。但还要一些场景依然没有涉及到。插件机制是保证软件系统可扩展性的重要收到，管理特定软件系统的插件也是制品库的重要功能（譬如 jira 插件, confluence 插件, jenkins 插件, visual studio 插件, IntelliJ IDEA 插件等)。未来，随着软件系统的发展，还会有众多的二进制类型包，制品库需要保持一定的可扩展性。

三、多来源制品库

现代软件产品是组装出来的，其组装件既有来做第三方的组件，也有公司内部其他项目的组件，也有项目本身的组件。譬如一个 Java 开发的结算系统，会用到 gson，jdbc 这样的第三方组件，也会用到公司内部的微服务框架组件，还会自己开发一些期货结算组件。针对多来源组件，我们将来自第三方组件称为**三方库**，来自公司内部其他项目的称为**二方库**，来自自己项目的称为**一方库**。企业统一制品库需要区分清楚二方库和三方库的管理。图 2 是一个制品库模型设计，下面就此图所涉及内容做详细阐述。

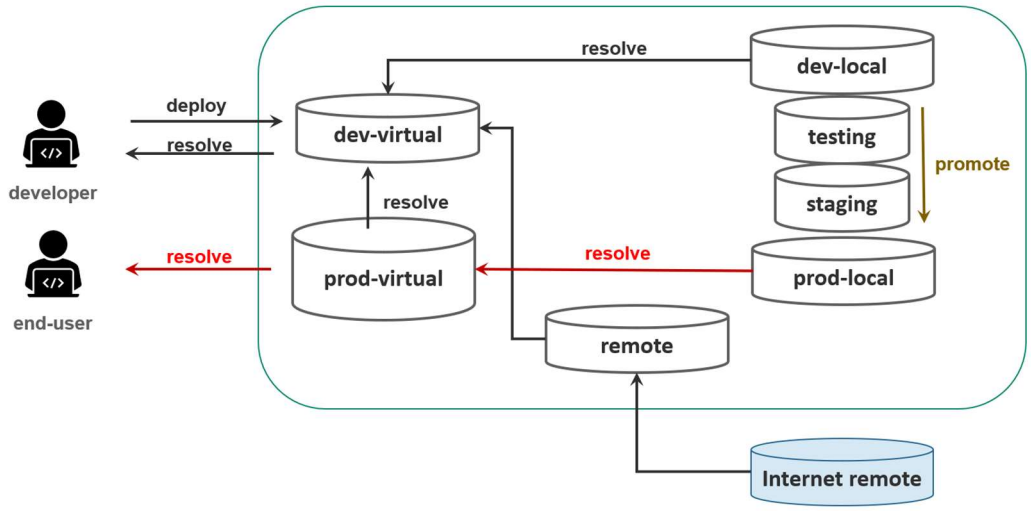

图 2. 库模型

如上图所示，在制品库中，有三种库类型：local 库，remote 库，virtual 库。顾名思义，local 库是存储本地制品，remote 库代理远端仓库，virtual 库可以代理 local 库和 remote 库。通过三种库类型的巧妙应用，可以变换出符合组织流程的库模型设计。

如上图，remote 库代理众多互联网库。local 库依据软件的生产流程，分为开发库，测试库，仿真库，生产库等。制品通过在库中的升级流转体现出软件产品的稳定迭代。而 virtual 库是制品库的对外接口。dev-virtual 接口可以上传和获取开发构建在制品和三方制品，prod-virtual 接口可以获取稳定发布制品和三方制品。

3.1 三方库建设

三方库的建设比较简单，需要特别注意以下几点。

（1）互联网第三方库的入库流程

简单的 remote 库代理即可实现自动入库，不过需要安全校验以排除有安全问题的三方制品，该流程详见 6.2 制品安全一节。

（2）管理许可证以避免法律风险

大部分的三方制品都不太强限制使用方式。不过如果是商业发行版的产品，一定要清晰其使用的依赖制品，以免遭遇法律风险。

（3）与付费软件的采购流程的结合

付费软件属于公司的无线资产，制品库可以充当无线资产制品的托管平台，与采购流程结合可以完善无线资产管理体系。

3.2 二方库建设

3.2.1 升级

一个成熟的软件制品需要开发，测试，安全，运维的多部门协作，多次的修改返工才可以达到稳定。通过制品库作为枢纽可以有力地将这个过程串联起来。制品库可以切分为开发制品库，测试制品库，生产制品库。只有通过核验的制品才可以升级到下一个阶段。制品不合格需要开发团队重新发布版本。配合语义化版本等标准化管理手段，可以非常清晰地管理，追踪和展现整个制品生产流水线。整个升级数据可以作为软件生产效率评测和度量，从而指导整个生产过程优化和提升。

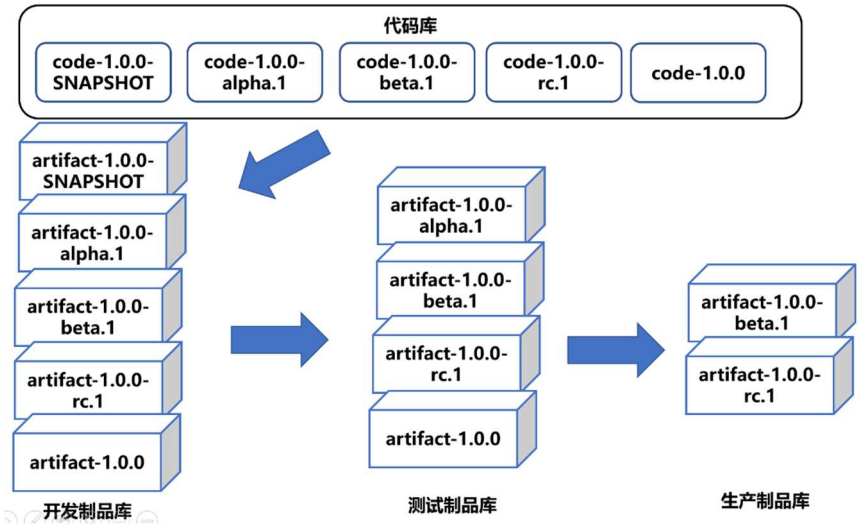

图 3. 制品升级

3.2.2 发布

不同的软件制品发布的含义是不同。对于非部署产品（组件，插件等），发布意味将该制品上传到制品库，并置为用户可获取状态。而部署类产品的发布意味该产品部署到生产环境，并置为用户可访问状态。前者的发布完全依赖制品库即可完成，而后者的发布需要部署工程的参与。

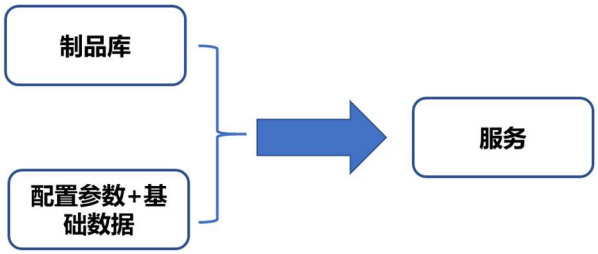

图 4. 部署工程

由上图可以看到，制品库是部署工程的发起点，需要配合配置参数和基础数据，整个部署工程才可以完整运转起来。

四、以制品库为中心的持续交付方法

前两章分别阐述了制品库的两个重要特征：多类型和多来源。这两个特征的建设完成，制品库的基本功能也即展现出来了。开发人员可以方便自由地获取制品，编译软件系统，发布制品。组织内部构建工具地推行实施也流畅很大。然后，面对持续交付地挑战，必须将制品库和流水线平台结合起来，才可以将我们地研发效率做更大提升。下面我们将讨论该内容。

4.1 元数据建设

制品元数据标识了制品出生，使用，消亡等信息，是制品的身份证。典型的单个制品元数据需要包含如下内容，见图 5。

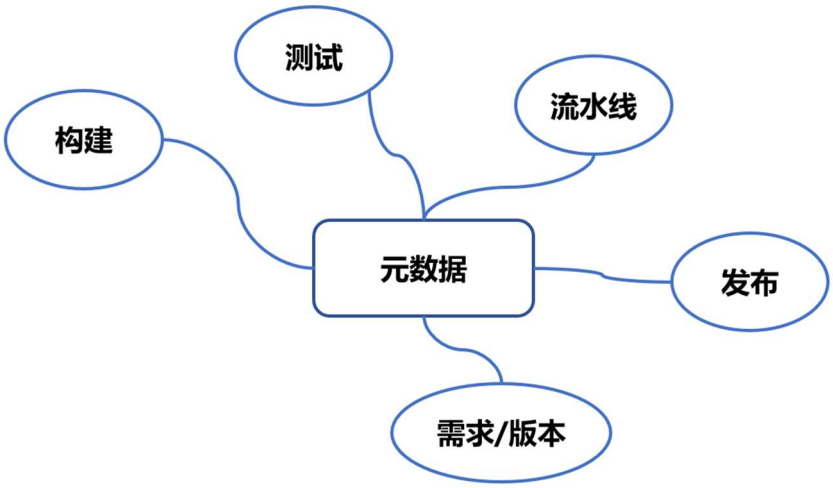

图 5. 单个制品元数据

如图 5 所示，元数据的数据项有：

1. 构建
   - 构建工具版本
   - 依赖树
2. 测试
   - 测试项
   - 测试方法

3. 流水线
    - 流水线项目
    - 流水线引擎
4. 发布
    - 发布环境
    - 许可证
5. 需求/版本
    - 版本号
    - git-commit
    - jira-issue-id

4.2 流水线建设

结合 3.2 二方库建设的阐述，以及持续交付流水线建设的相关经验，我司设计了图 6 的三级跃迁流水线模型。区分开发，测试，生产制品库。分三个跃迁建设流水线，具体的跃迁内容结合组织自身实践酌情增减。

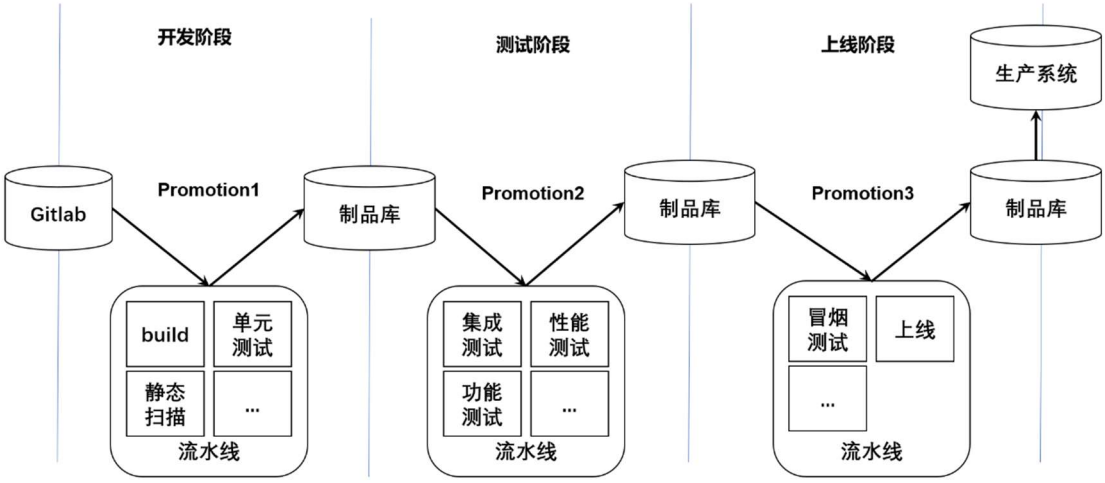

图 6. 三级跃迁流水线模型

如图 6 所示，流水线建设的内容有：

1. 一级跃迁
    - 代码扫描
    - 白盒测试
    - 构建
2. 二级跃迁
    - 黑盒测试
3. 三级跃迁
    - 上线核验
    - 安全校验
    - 上线

五、网段隔离与制品库运维

大部分的证券期货行业公司的内网系统都是在网段隔离的环境下运行。本章节重点介绍在网段隔离环境下的部署模型设计。

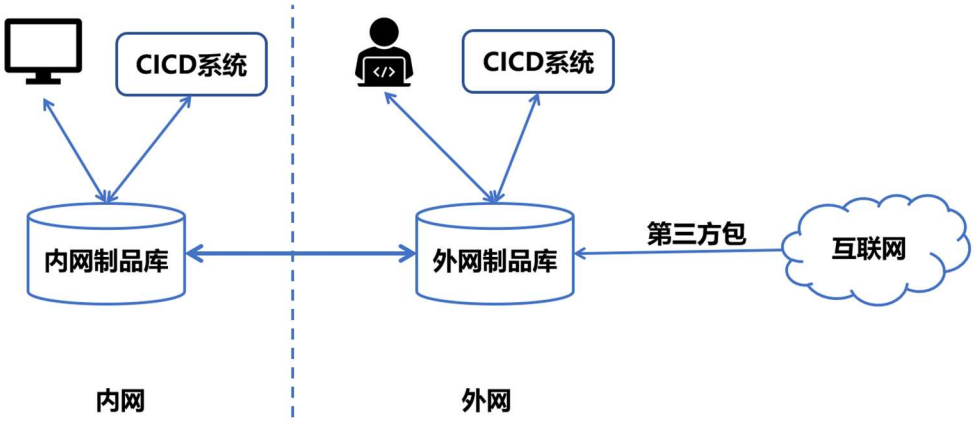

图 7. 网段隔离环境下的部署模型

图 7 展示了在内外网隔离条件下，如何建设一个方便可用的制品库平台。在可访问互联网的网段部署外网制品库。在安全访问受限的内网部署内网制品库。两个制品库实例在安全链路做数据交换。在部署设计上，两个制品库实例库模型完全一致，只是库内数据有差别。原则上，三方库制品以外网为基准，内网三方库是外网的子集，二方库制品以内网为基准，外网二方库是内网的子集。

实际使用中，基本是会产生以下三类场景：

场景一：内网三方库按需同步外网三方库。这是最多的场景，带宽占用最多的场景。

场景二：内网二方库同步外网二方库。需要使用外网二方制品的场景较少，可用采用定时同步外网二方库到内网以满足“**外网二方库是内网的子集**”的原则。再加一个按需同步以应对使用需求。

场景三：二方库外放（内网二方库按需同步外网二方库），此场景实际上相当于把二方制品发布到组织外部，如果是商业版的制品发布就依照组织的产品外放发布流程即可（可能需要安全审核和商务事宜）。

六、安全管控

6.1 权限设计

制品库的权限设计需要和 4.3 节的流水线模型以及第五章节的部署模型配合起来。综合制品库的使用情况。设计权限角色表如表 1。角色和人员的对应需要各组织结合特定的组织架构配置完成。

action	anonymous	internal-users	developer	QA	operator	master	admin
读三方库	√	√	√	√	√	√	√
读二方库		√	√	√	√	√	√
写开发二方库			√			√	√
写测试二方库				√		√	√
写生产二方库					√	√	√
写三方库							√

表 1. 权限角色表

如表 1 所示，划分 7 个角色，具体含义如下：

（1）anonymos：匿名用户，等同于任何人，某些系统也称为 Guests；

（2）internal-users：组织内员工，技术上映射为组织用户服务器中的所有用户；

（3）developer：开发部门员工

（4）QA：测试部门员工

（5）operator：运维部门员工

（6）master：库管理员

（7）admin：平台管理员

6.2 制品安全

6.2.1 安全扫描

第三方的制品为我们创造软件系统提供了极大的便利，但也带来普遍的安全性上的问题。社区驱动和商业研发的产品即使再有保障手段，其引入恶意代码的风险也是存在。过去几年，我们注意到在 npm 社区就有这样的案例。（譬如一个周下载量过 200 万的 npm 包被注入恶意代码，导致 vue，Node 项目受影响；阿里的一个 npm 组件由于开发人员乱加彩蛋导致页面出错等）。由此可见，对三方库制品的定期安全扫描是必要的。对于注重安全的金融行业，要特别注意在这方面的投入和研发。

二方库制品的安全扫描主要是分析其依赖的制品安全状况。其制品本身的安全状况结果要前期的代码安全校验得出，然后传输到制品库的元数据中。对于制品的依赖分析不仅有助于安全分析，也可以分析制品的使用情况和依赖关系，其分析数据对于制品采购、制品引入以及 bug 溯源等都有很好的借鉴意义。

6.2.2 容灾高可用和制品清理

1. 容灾高可用

作为一个数据托管系统，制品的容灾高可用体系也是非常重要的一块内容。我司目前的仅对制品库做了简单的备份容灾。后续需要以同机备份，异机备份，磁带备份的三级备份体系对容灾体系做进一步优化和提升。高可用体系在目前的使用场景下还不太紧迫，但基本的架构体系和实施方案也在推进中。

2. 制品清理

软件开发有时候是一个杂乱的过程。制品库在管理过程中，需要依需制定自己的保留策略。下面阐述几种制品清理技术：

- 限制保留多少 SNAPSHOT
- 定期清除超大缓存
- 定期删除未使用的制品
- 定期归档未使用的二方库制品

6.4 误判和白名单

安全扫描在保证安全性的同时，也有一些副作用。一个典型的问题是问题误判。误判产生的原因多种多样。主要有漏洞数据有错，漏洞不影响应用，临时上线需要放开校验等。白名单机制可以在安全扫描的基础上有效处理这些情况。

6.5 总结

综合上述分析，加入安全管控后，其部署模型将变成图 8 所示。主要增加点如下：

（1）建立 DMZ 仓库代理公网仓库，制品仓库通过 DMZ 仓库拉取第三方包，实现网络环境隔离；

（2）以漏洞扫描结果为基础，建立信任规则，原则上有漏洞的包不允许被制品仓库下载。

（3）建立白名单机制，特殊需求提交审批备案，通过后允许用户申请下载非安全包。

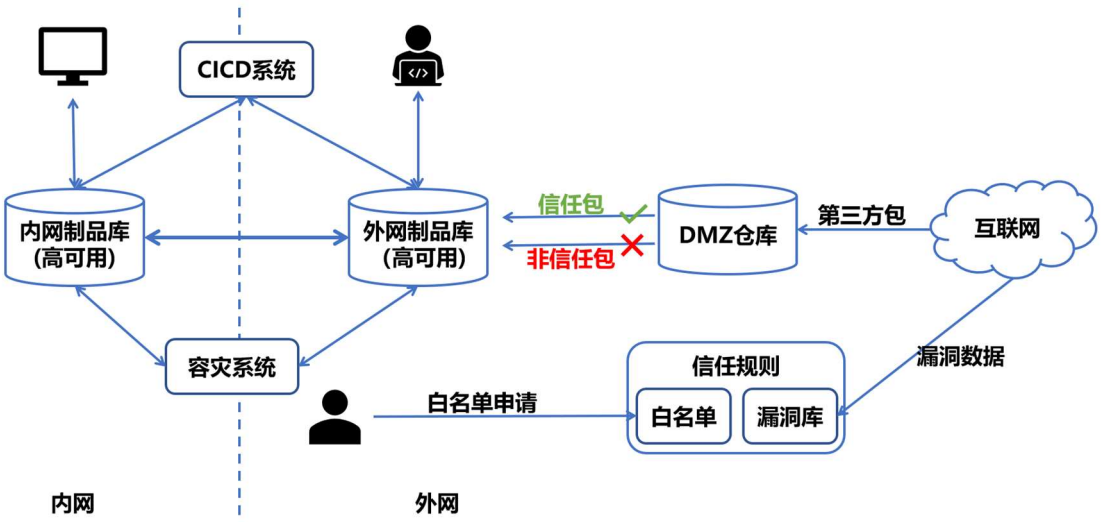

图 8. 安全管控部署模型

七、展望和下一步的工作

通过以上六个章节，基本上可以建立起一套功能完备，管控安全，配置合理，使用方便的制品库平台。接下来，关注点将会集中在以制品库为枢纽的持续交付体系建设中，制品库将成为该体系中的重要基础设施。本文 4.3 简略地介绍了一个基于制品库的三级跃迁流水线

模型。但是，软件开发过程是复杂多变，流水线模型也也应该基于现实情况应时而变。除此之外，安全管控的对象也不仅仅是制品库，建设**安全的持续交付流水线**，**安全左移**到代码的撰写过程中也都是很重要的课题。我司也将在这些方面持续努力，为开发者提效保质贡献更有力的产品。

参考文献：

[1]. https://www.vmware.com
[2]. https://docs.openstack.org
[3]. https://www.yum.com
[4]. https://chocolatey.org
[5]. https://maven.apache.org
[6]. https://gradle.org
[7]. https://www.npmjs.com
[8]. https://yarnpkg.com
[9]. https://pypi.org
[10]. https://www.nuget.org
[11]. https://www.docker.com
[12]. https://kubernetes.io
[13]. https://helm.sh