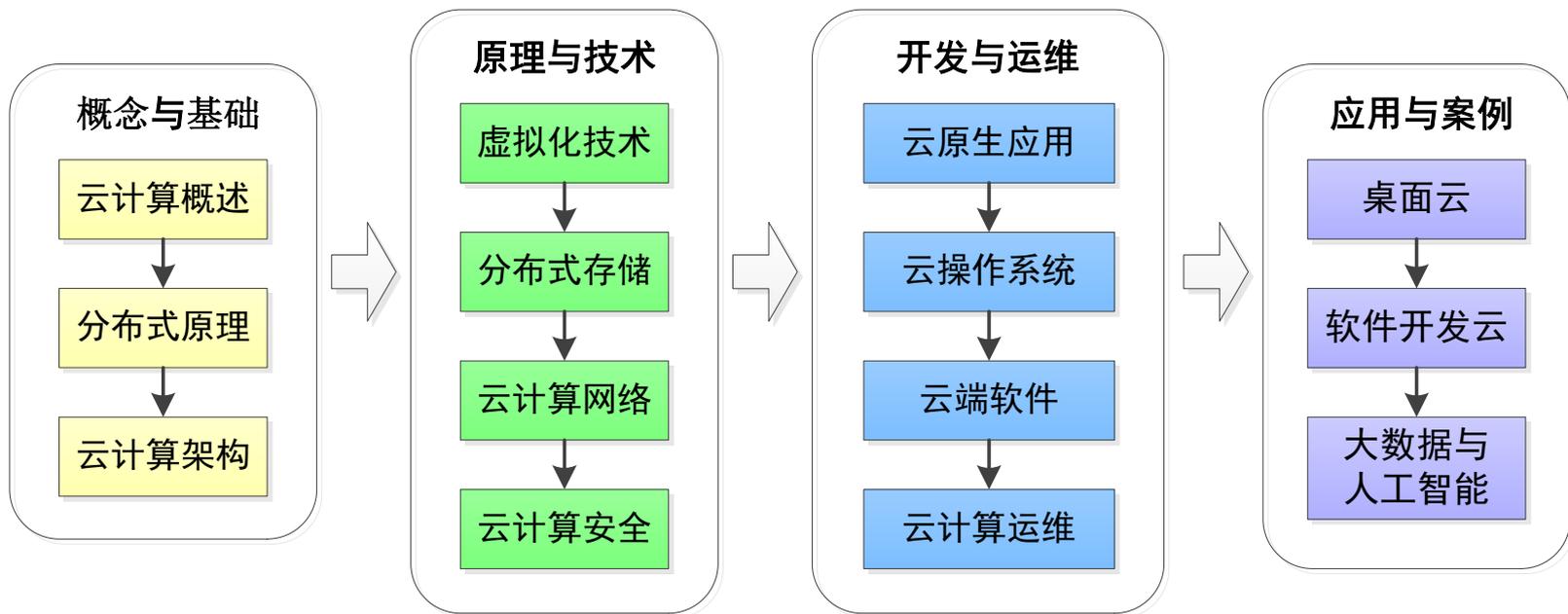


云计算原理与实践

Principles and Practice of Cloud Computing

《云计算原理与实践》课程总览



Outline

- 8.1 云原生的相关概念
- 8.2 云原生应用开发实践的12要素
- 8.3 云原生应用开发
- 8.4 实践：基于Node.js的云原生应用开发

8.1 初识云计算

8.1.1 云原生简介

8.1.2 云原生的内容

8.1.3 云原生应用的技术手段

8.1.1 云原生简介

- 2015年，Pivotal公司的马特·斯泰恩（Matt Stine）提出Cloud Native这一概念，并结合这个概念包装了自己的新产品Pivotal Web Service和Spring Cloud。
- 云原生的主旨是构建运行在云端的应用程序，致力于使应用程序能够最大限度地利用云计算技术特性的优势，提供更加优质的应用服务。云原生也是一种构建和运行应用程序的方法，它充分利用了云计算的优势，重点关注如何在云计算交付模式下创建和部署应用程序。
- 云原生应用应该具备以下几个关键词：**敏捷，可靠，高弹性，易扩展，故障隔离保护，不中断业务持续更新。**

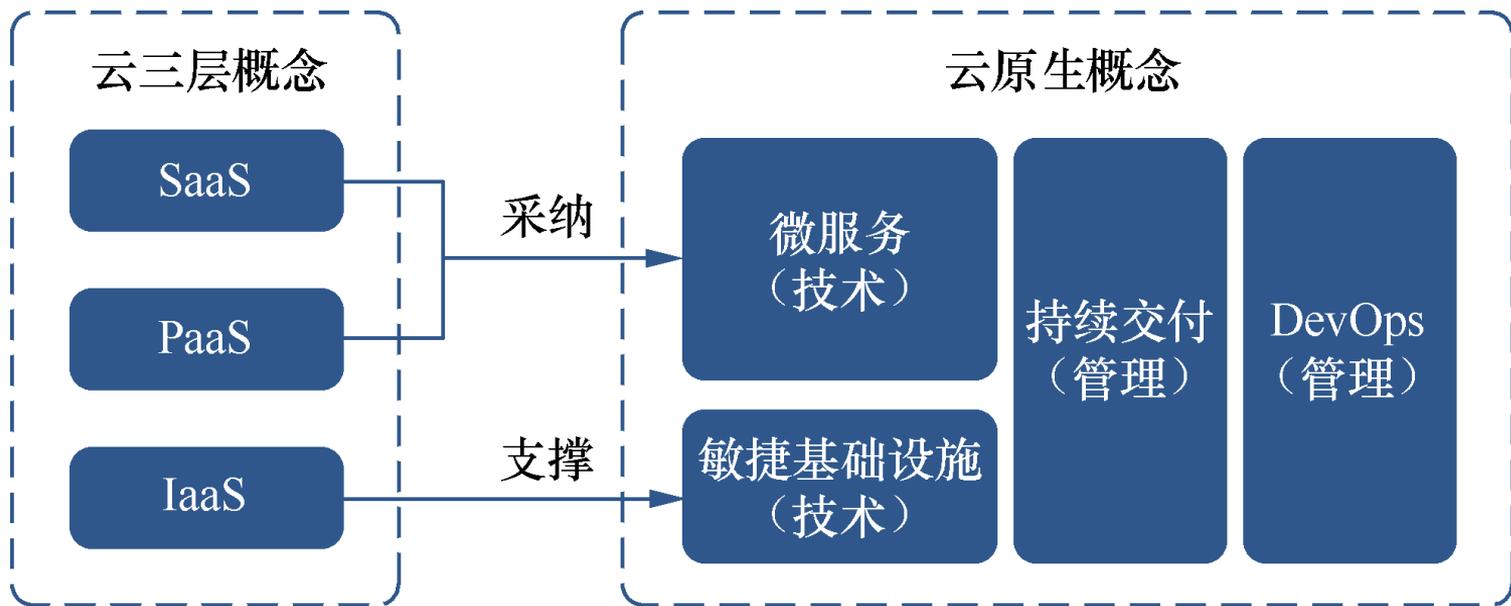
8.1.1 云原生简介

- 目前有许多不同类型的云服务可用于支持云原生应用的开发。以**基础设施即服务（IaaS）**为例，假如用户是一名开发或运维人员，用户可以在云端订阅提供虚拟机的服务，这个虚拟机的环境与用户在本地使用的虚拟或物理环境一模一样。
- 取而代之的是将用户当前的平台或软件栈移植到云服务中的**平台即服务（PaaS）**产品中。这种方式可以避免单独购买授权产品，省去烦琐的安装和维护过程。
- PaaS产品服务的目标是突破IaaS云服务所不能提供的一些平台级服务，这个目标或原则也最终被转化为**软件即服务（SaaS）**产品的使用。

8.1.2 云原生的内容

- 云原生是面向“云”设计的应用，因此技术部分依赖于传统云计算的三层概念，即基础设施即服务（IaaS）、平台即服务（PaaS）和软件即服务（SaaS）。
- 应用基于云服务进行架构设计，对技术人员的要求更高。除了对业务场景的考虑外，对隔离故障、容错、自动恢复等非功能需求会考虑更多。
- 借助云服务提供的能力也能实现更优雅的设计，例如弹性资源的需求、跨机房的高可用、11个9（99.999999999%）的数据可靠性等特性，基本是云计算服务本身就提供的能力，开发者直接选择对应的服务即可，一般不需要过多考虑本身机房的问题。

图8.1 云原生的内容



8.1.2 云原生的内容

1. 敏捷基础设施

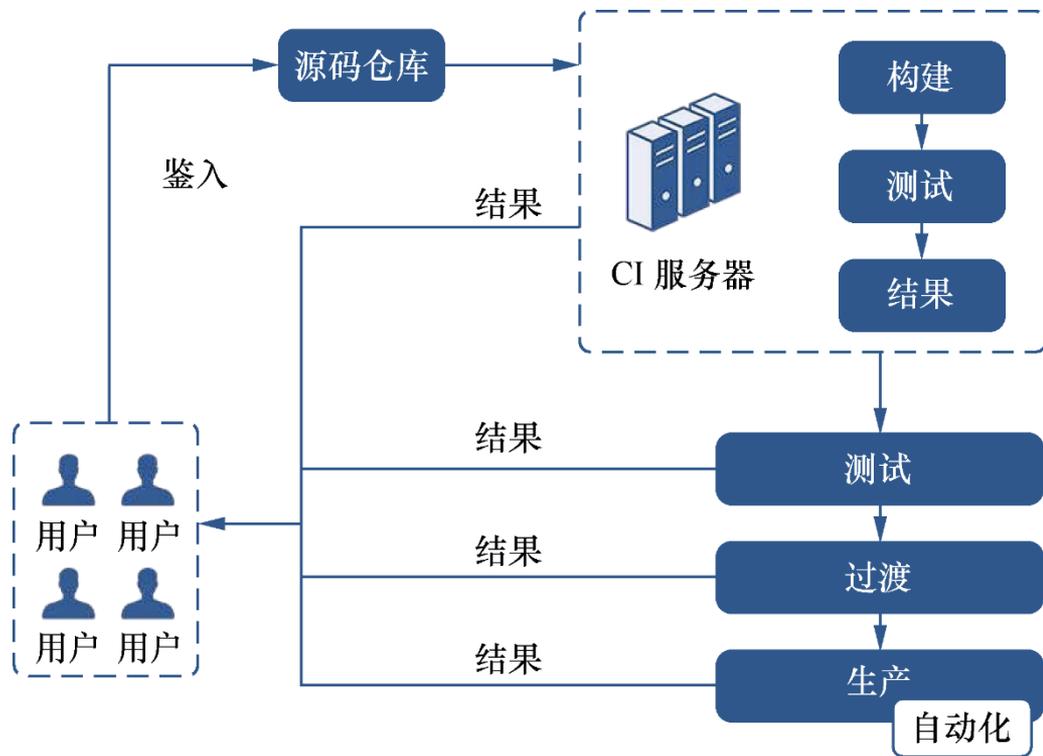
- 正如通过业务代码能够实现产品需求、通过版本化的管理能够保证业务的快速变更，基于云计算的开发模式也要考虑如何保证基础资源的提供能够根据代码自动实现需求，并实现记录变更，保证环境的一致性。
- 技术人员部署服务器、管理服务器模板、更新服务器和定义基础设施的模式都是通过代码来完成的，并且是自动化的，不能通过手工安装或克隆的方式来管理服务器资源。
- 此外，基础设施的范围也会更加广泛，不仅包括机器，还包括不同的机柜或交换机、同城多机房、异地多机房等。

8.1.2 云原生的内容

2. 持续交付

- 为了满足业务需求频繁变动，通过快速迭代，产品能做到随时都能发布的能力，是一系列的开发实践方法。它分为**持续集成**、**持续部署**、**持续发布**等阶段，用来确保从需求的提出到设计开发和测试，再到让代码快速、安全地部署到产品环境中。
- 持续集成是指每当开发人员提交了一次改动，就立刻进行构建、自动化测试，确保业务应用和服务能符合预期。
- 持续交付是软件发布的能力，在持续集成完成之后，能够提供到预发布之类系统上，达到生产环境的条件。
- 持续部署是指使用完全的自动化过程来把每个变更自动提交到测试环境中。

图8.2 持续交付流程示例

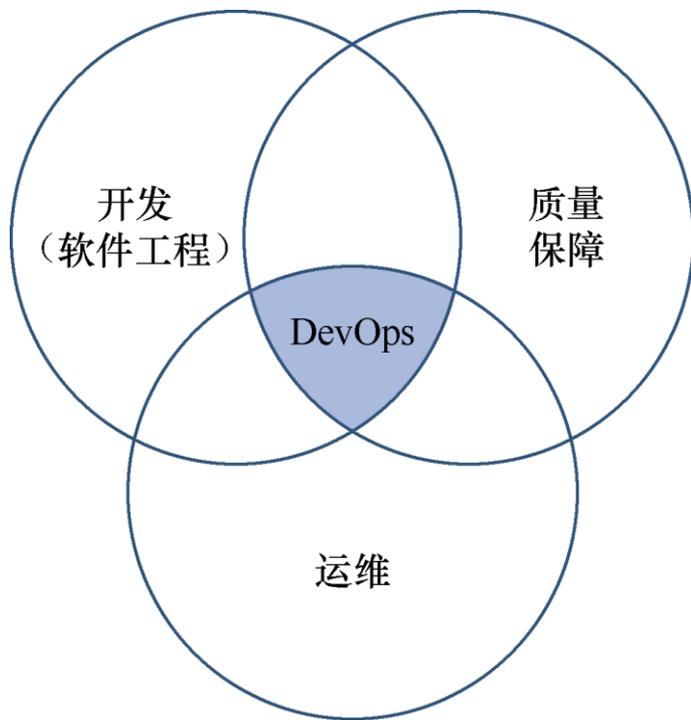


8.1.2 云原生的内容

3. DevOps

- DevOps从字面上理解只是Dev（开发人员）+ Ops（运维人员），实际，它是一组过程、方法与系统的统称。
- （1）组织架构、企业文化与理念等，需要自上而下设计，用于促进开发部门、运维部门和质量保障部门之间的沟通、协作与整合。
- （2）自动化是指所有的操作都不需要人工参与，全部依赖系统自动完成。
- （3）DevOps的出现是由于软件行业日益清晰地认识到，为了按时交付软件产品和服务，开发部门和运维部门必须紧密合作。

图8.3 DevOps强调组织的沟通与协作



8.1.2 云原生的内容

4. 微服务

- 随着企业的业务发展，传统业务架构面临着很多问题。
- ① 单体架构在需求越来越多的时候无法满足其变更要求，开发人员对大量代码的变更会越来越困难，同时也无法很好地评估风险，所以迭代速度慢。
- ② 系统经常会因为某处业务的瓶颈导致整个业务瘫痪，架构无法扩展，木桶效应严重，无法满足业务的可用性要求。
- ③ 整体组织效率低下，无法很好地利用资源，存在大量的浪费。

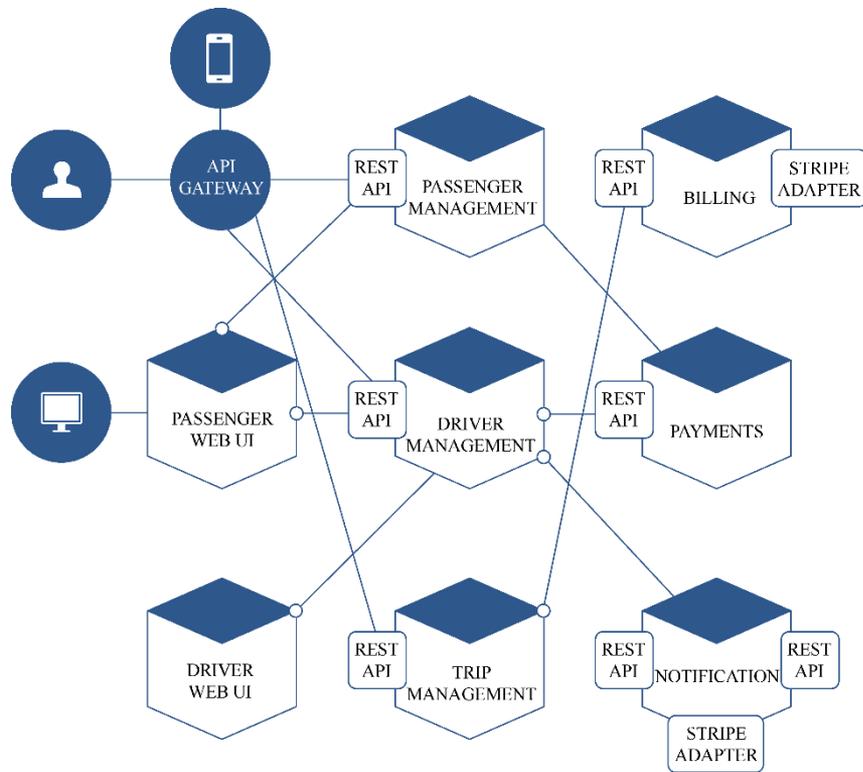


8.1.2 云原生的内容

4. 微服务

- 随着微服务化架构的优势展现和快速发展，2013年，马丁·福勒（Martin Flower）对微服务概念进行了系统的理论阐述，总结了其相关的技术特征。
- ①微服务是一种架构风格，也是一种服务；
- ②微服务的颗粒比较小，一个大型复杂软件应用由多个微服务组成，例如Netflix目前由500多个微服务组成；
- ③它采用UNIX设计的哲学——每种服务只做一件事，是一种松耦合的、能够被独立开发和部署的无状态化服务（独立扩展、升级和可替换）。

图8.5 微服务架构示例



8.1.3 云原生应用的技术手段

- 从宏观概念上讲，云原生是不同思想的集合，集目前各种热门技术之大成。
- 在实际的云原生开发过程中，团队需要一个构建和运行云原生应用程序的平台，这个平台需要具有高度自动化和集成化的特点。从具体的技术手段来说，它会涉及微服务、DevOps、持续集成（Continuous Integration, CI）与持续交付（Continuous Delivery, CD）、容器等技术。

1. 微服务技术

- 从宏观概念上讲，云原生是不同思想的集合，集目前各种热门技术之大成。
- 在实际的云原生开发过程中，团队需要一个构建和运行云原生应用程序的平台，这个平台需要具有高度**自动化**和**集成化**的特点。从具体的技术手段来说，它会涉及微服务、DevOps、持续集成（Continuous Integration, CI）与持续交付（Continuous Delivery, CD）、容器等技术。
- 值得一提的是，微服务领域有一个著名的“**康威定律**”：设计系统的组织、最终产生的设计等同于组织之内、之间的沟通结构。这意味着设计系统的企业生产的设计等同于企业内的沟通结构

图8.6 云原生应用的关键技术

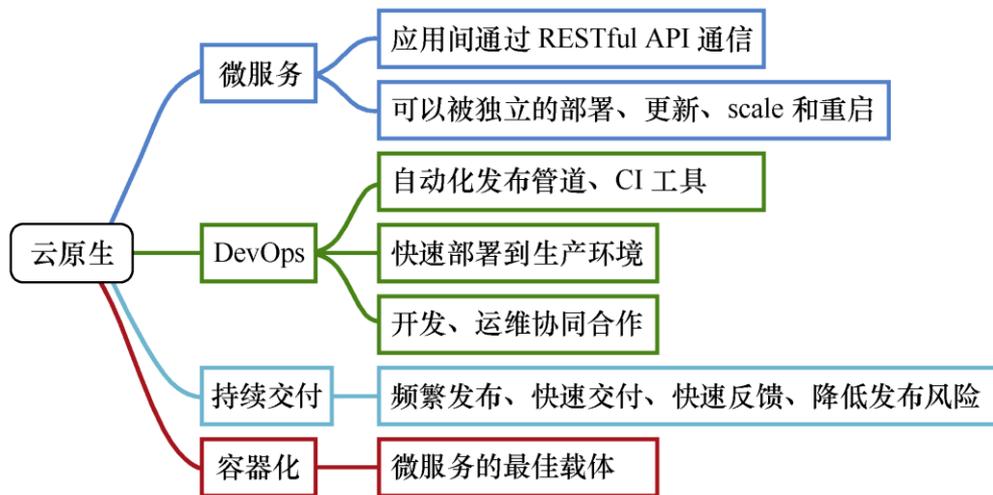
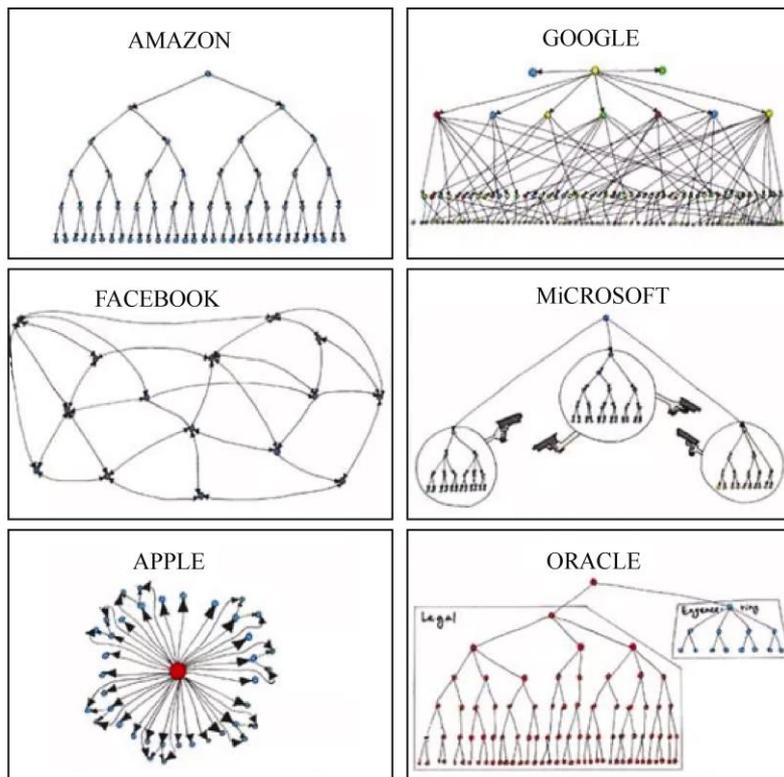


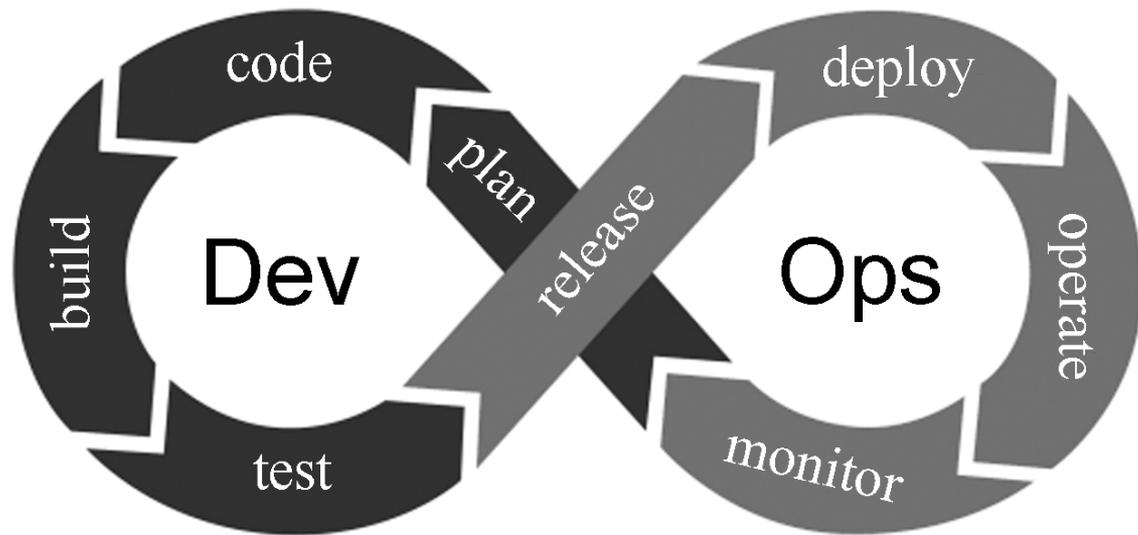
图8.7 康威定律的形象说明



2. DevOps

- DevOps技术通过自动化软件交付和架构变更的流程，使得构建、测试、发布软件能够更加地快捷、频繁和可靠
- 可以把DevOps看作**开发（软件工程）、技术运营和质量保障（QA）**三者的交集。传统的软件组织将开发、IT运营和质量保障（QA）设为各自独立的部门，在这种环境下如何采用新的开发方法（例如敏捷软件开发）是一个重要的课题。
- DevOps考虑的还不只是软件部署，它是一套针对这几个部门间沟通与协作问题的流程和方法。需要频繁交付的企业可能更需要了解DevOps。

图8.8 DevOps流程



2. DevOps

- 以下几方面因素可能促使一个组织引入DevOps：
 - 使用敏捷或其他软件开发过程与方法；
 - 业务负责人要求加快产品交付的速度；
 - 虚拟化和云计算基础设施（可能来自内部或外部供应商）日益普遍；
 - 数据中心自动化技术和配置管理工具的普及。

2. DevOps

- DevOps的落地实现需要通过一套集成的工具链，具体包括以下目标：
- 开发、交付和运维工具之间的实时协作；
- 实现从需求获取和需求评审到设计和代码分析的持续规划；
- 落实测试策略以实施持续测试；
- 当成功完成代码签入后，通过自动触发构建持续集成；
- 测试自动化脚本可以按照作业计划执行，实现持续交付；
- 通过报告和仪表盘持续监测程序发布质量；
- 可通过自动化的缺陷识别和解决方案，帮助用户快速响应变更；
- 可以提供基于关键绩效指标（KPI）的有价值的报告，以使用户快速做出决策；
- 通过跟踪发布流水线实现持续交付。

3. 持续集成与持续交付技术

- **持续集成**是一种软件开发的实践方法，它要求团队成员经常整合他们的工作成果（通常是程序代码）。通常情况下，团队成员中的每人每天至少提交一次自己的代码到代码仓库做集成构建，这样对于整个项目而言，每天就会有多次集成构建。
- **持续交付**是一种以可持续的方式安全快速地将所有类型的软件变更（包括新功能开发、配置更改、Bug修复等）转化为生产环节下的工作产品交付给用户直接使用的软件过程控制方法，它的最终目标是将变更直接部署到生产环境。

4. 容器技术

- 容器技术与虚拟机技术相比，拥有更高的资源使用效率，因为它并不需要为每个应用分配单独的操作系统，所以实例规模更小、创建和迁移速度也更快。相对于虚拟机，单个操作系统能够承载更多的容器。
- 容器化最大的好处是保持运行环境的一致性，只要应用可以打包成容器镜像（通常使用Docker容器），就可以一次编译后，在各处运行。
- 同时，容器也可以作为应用运行的最小组件来部署，且更适合作为无状态应用运行。结合容器编排工具（如Kubernetes）将大大增强系统的扩展性和自愈能力，轻松应对大流量下的高并发场景，加快业务的迭代速度。Kubernetes作为CNCF（云原生计算基金会）成员的核心，本身就是与云原生应用的理念紧密结合的产物。

云原生归纳

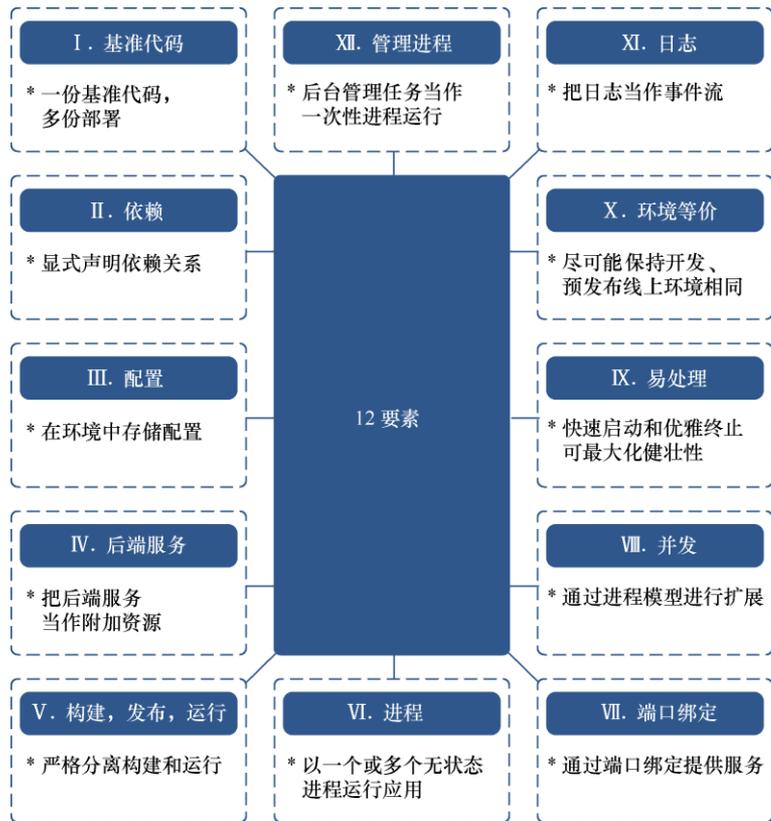
- 充分利用云计算技术的优势：采用**云端优先策略**，从云服务中获取最大价值；
- 实现**快速、敏捷、频繁的交付模式**；
- 通过技术创新更多地**扩展云计算技术的边界**。

- 云原生中包含的不同思想，与其所解释的云上应用架构应该具备的特性几乎是一一对应的：
 - DevOps、持续交付对应更快的上线速度，即敏捷性；
 - 微服务对应可扩展性及故障可恢复性；
 - 敏捷基础设施实现了扩展能力的资源层支持；
 - **康威定律**在组织结构和流程上确保架构特性能够快速实施。

8.2 云原生应用开发实践的12要素

1. 一份代码库与多份部署
2. 显式声明依赖关系
3. 在环境中存储配置
4. 把后端服务当作附加资源
5. 严格分离构建和运行
6. 以一个或多个无状态进程运行应用
7. 通过端口绑定提供服务
8. 通过进程模型进行扩展
9. 快速启动和优雅终止可最大化健壮性
10. 尽可能保持开发与预发布线上环境相同
11. 把日志当作事件流
12. 后台管理任务当作一次性进程运行

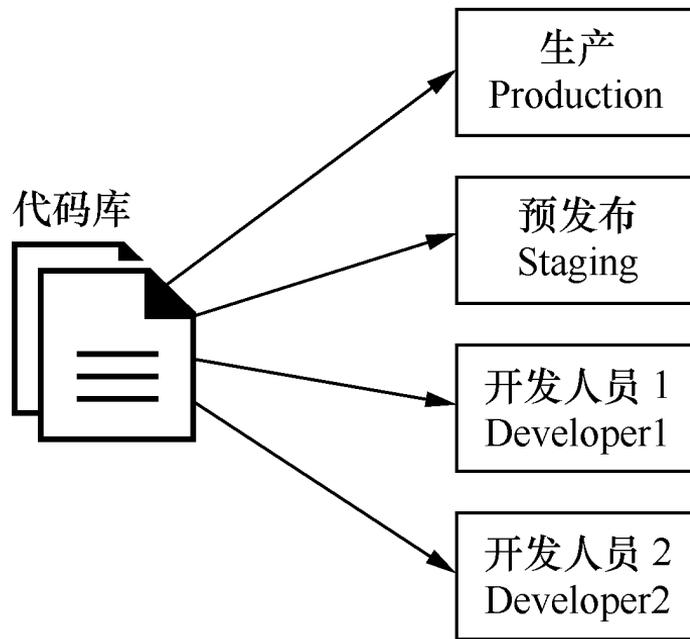
图8.9 “12要素”的内容



1. 一份代码库与多份部署

- 12要素应用通常会使用版本控制系统加以管理，如Git、Mercurial、Subversion。一个用来跟踪代码所有修订版本的数据库被称作**代码库**（code repository、code repo、repo）。
- 代码库和应用之间总是保持一一对应的关系：
- 一旦有多个代码库，就不能称为一个应用，而是一个分布式系统。分布式系统中的每一个组件都是一个应用，每一个应用可以分别使用12要素进行开发。
- 多个应用共享一个代码库是有悖12要素原则的。解决方案是将共享的代码拆分为独立的类库，然后使用依赖管理策略去加载它们。
- 尽管每个应用只对应一个代码库，但可以同时存在多份部署。所有部署的代码库相同，但每份部署可以使用其不同的版本。

图8.10 一份代码库 (Codebase) 与多份部署 (deploy)



2. 显式声明依赖关系

- 大多数编程语言都会提供一个打包系统，为各个类库提供打包服务，就像 Perl 的CPAN或是 Ruby的Rubygems。通过打包系统安装类库可以是系统级的（称之为“site packages”），或仅供某个应用程序使用，部署在相应的目录中（称之为“vendoring”或“bundling”）。
- 12要素原则下的应用程序不会隐式依赖系统级的类库。它一定通过“**依赖清单**”，确切地声明所有依赖项。
- 显式声明依赖的优点之一是为新进开发者简化了环境配置流程。新进开发者可以找出应用程序的代码库，安装编程语言环境和它对应的依赖管理工具，只需通过一个“构建命令”就能安装所有的依赖项开始工作。

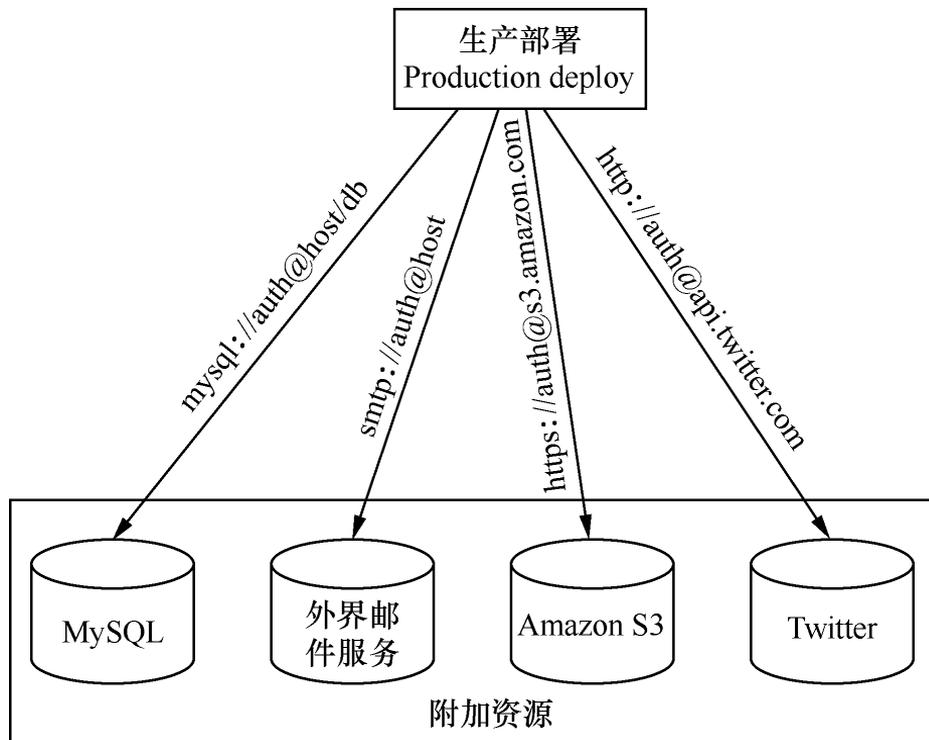
3. 在环境中存储配置

- 通常，应用的配置在不同部署（预发布、生产环境、开发环境等）间会有很大差异。有些应用在代码中使用常量保存配置，这与12要素所要求的代码和配置严格分离显然不符。配置文件在各部署间存在很大差异，代码却完全一致。
- 判断一个应用是否正确地将配置排除在代码之外，一个简单的方法是看该应用的代码库是否可以立刻开源，而不用担心会暴露任何敏感的信息。另外一个解决方法是使用配置文件，但不把它们纳入版本控制系统，就像Rails的config/ database.yml。
- 12要素推荐**将应用的配置存储于环境变量中**（env vars, env）。环境变量可以非常方便地在不同的部署间做修改，却不用改一行代码；与配置文件不同，不小心把它们签入代码库的概率微乎其微；与一些传统的解决配置问题的机制（例如Java的属性配置文件）相比，环境变量与语言和系

4. 把后端服务当作附加资源

- 后端服务是指程序运行所需要的通过网络调用的各种服务，如数据库（MySQL, CouchDB）、消息/队列系统（RabbitMQ, Beanstalkd）、SMTP邮件发送服务（Postfix），以及缓存系统（Memcached）。
- 类似数据库的后端服务，通常由部署应用程序的系统管理员一起管理。
- 12要素应用不会区别对待本地或第三方服务。对应用程序而言，两种都是附加资源，都可以通过一个URL或是其他存储在配置中的服务定位/服务证书来获取数据。
- **每个不同的后端服务是一个资源。** 部署可以按需加载或卸载资源。

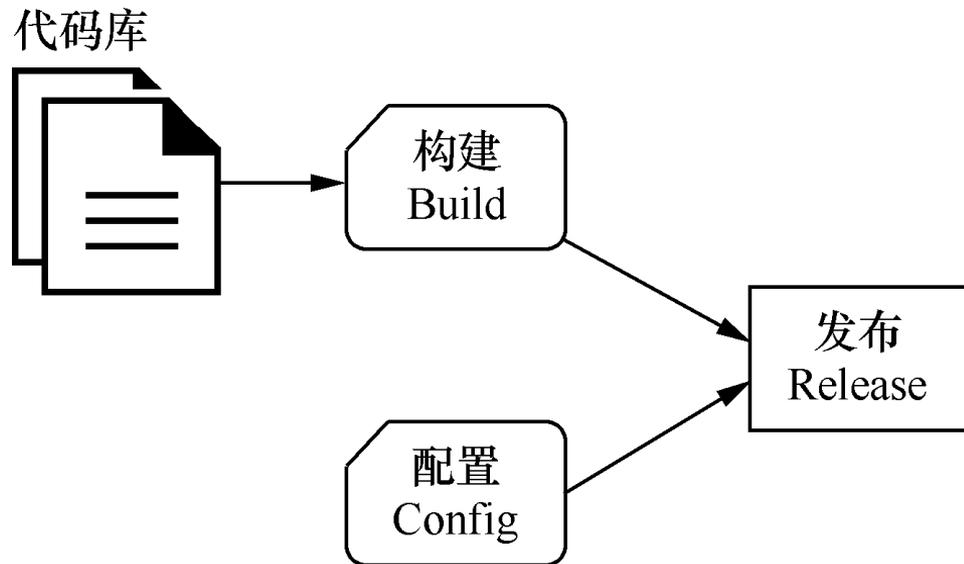
4. 把后端服务当作附加资源



5. 严格分离构建和运行

- 代码库转化为一份部署（非开发环境）需要以下三个阶段
- （1）**构建阶段**是指将代码仓库转化为可执行包的过程。构建时会使用指定版本的代码，获取和打包依赖项，编译成二进制文件和资源文件。
- （2）**发布阶段**会将构建的结果和当前部署所需配置相结合，并能够立刻在运行环境中投入使用。
- （3）**运行阶段**（或者说“运行时”）是指针对选定的发布版本，在执行环境中启动一系列应用程序进程。
- 12要素应用严格区分构建、发布、运行三个步骤。每一个发布版本必须对应一个唯一的发布ID，例如可以使用发布时的时间戳（2011-04-06- 20:32:17）或是一个增长的数字（v100）。新的代码在部署之前，需要开发人员触发构建操作。但是，运行阶段不一定需要人为触发，而是可以自动进行。

5. 严格分离构建和运行



6. 以一个或多个无状态进程运行应用

- 运行环境中，应用程序通常是以一个或多个进程运行的。
- 12要素应用的进程必须无状态且无共享。任何需要持久化的数据都要存储在后端服务内，例如数据库。
- **源文件打包工具**（Jammit、django-compressor）使用文件系统来缓存编译过的源文件。12要素应用更倾向于在构建步骤执行此操作（如Rails资源管道），而不是在运行阶段。
- 一些互联网系统依赖于“黏性Session”，是指将用户Session中的数据缓存至某进程的内存中，并将同一用户的后续请求路由到同一个进程。**黏性Session是12要素极力反对的**。Session中的数据应该保存在诸如Memcached或Redis这样的带有过期时间的缓存中。

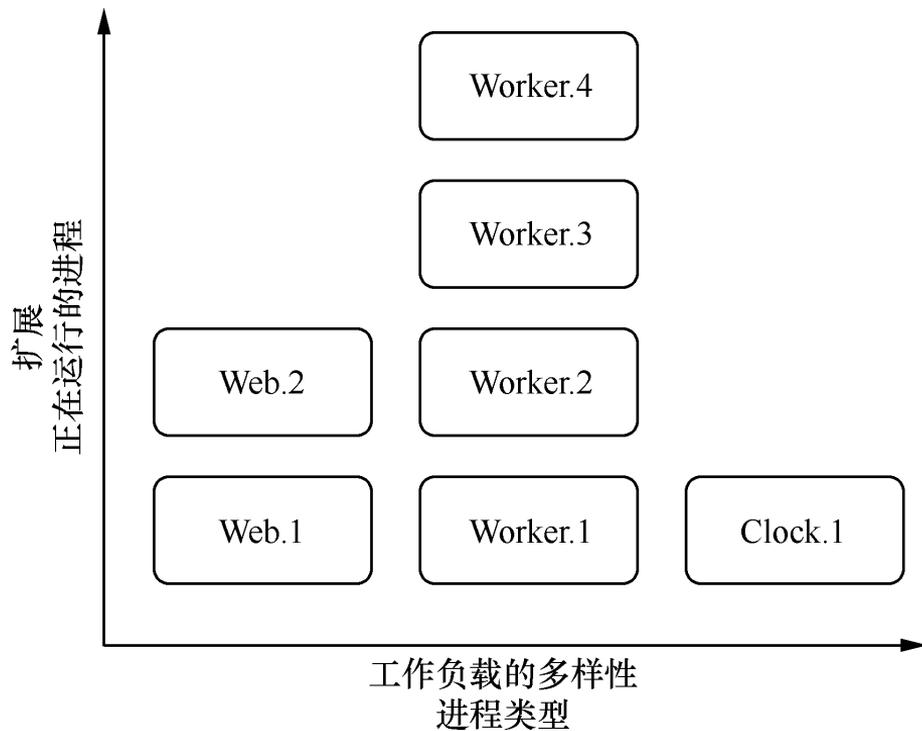
7. 通过端口绑定提供服务

- 互联网应用有时会运行于服务器的容器之中。
- 12要素应用**完全自我加载**而不依赖于任何网络服务器就可以创建一个面向网络的服务。互联网应用通过端口绑定来提供服务，并监听发送至该端口的请求。
- HTTP并不是唯一可以由端口绑定提供的服务。几乎所有服务器软件都可以通过进程绑定端口来等待请求。例如，使用XMPP的ejabberd，以及使用Redis协议的Redis。
- 还要指出的是，端口绑定这种方式意味着一个应用可以成为另外一个应用的后端服务，调用方将服务方提供的相应URL当作资源存入配置以备将来调用。

8. 通过进程模型进行扩展

- 任何计算机程序一旦启动，就会生成一个或多个进程。互联网应用采用多进程运行方式。
- 在12要素应用中，进程是一等公民。12要素应用的进程主要借鉴于UNIX守护进程模型。开发人员可以运用这个模型去设计应用架构，将不同的工作分配给不同类型的进程。
- 这其中并不包括个别较为特殊的进程，例如通过虚拟机的线程处理并发的内部运算，或是使用诸如EventMachine、Twisted、Node.js的异步事件触发模型。
- 12要素应用的进程所具备的**无共享、水平分区**的特性意味着添加并发应用会变得简单而稳妥。12要素应用的进程不需要守护进程或是写入PID文件。相反的，应该借助操作系统的进程管理器来管理输出流

图8.13 通过进程模型进行扩展



9. 快速启动和优雅终止可最大化健壮性

- 12要素应用的进程是**易处理（Disposable）**的，意思是它们可以瞬间开启或停止。这有利于快速、弹性地伸缩应用，迅速部署变化的代码或配置，稳健地部署应用。
- 进程应当追求最少启动时间。理想状态下，进程从输入命令到真正启动并等待请求的时间应该很短。
- 进程一旦接收到终止信号（Sigterm）就会优雅终止。就网络进程而言，优雅终止是指停止监听服务的端口，即拒绝所有新的请求，并继续执行当前已接收的请求，然后退出。
- 进程还应当面对突然死亡时保持健壮，例如底层硬件故障。
- 无论如何，12要素应用都应该可以设计能够应对意外的、不优雅的终结。**Crash-only design**将这种概念转化为合乎逻辑的理论。

10. 尽可能保持环境相同

- 从以往的经验来看，开发环境（即开发人员的本地部署）和线上环境（外部用户访问的真实部署）之间存在着很多差异。这些差异表现在以下三个方面。
- **时间差异**：开发人员正在编写的代码可能需要几天、几周，甚至几个月才会上线。
- **人员差异**：开发人员编写代码，运维人员部署代码。
- **工具差异**：开发人员使用Nginx、SQLite、OS X，而线上环境使用Apache、MySQL及Linux。

- 12要素应用要想做到持续部署就必须缩小本地与线上差异。缩小时间差异：开发人员可以几小时，甚至几分钟就部署完代码。
- 缩小人员差异：开发人员不只是编写代码，更应该密切参与部署过程以及关注代码在线上的表现。
- 缩小工具差异：尽量保证开发环境以及线上环境的一致性。

11. 把日志当作事件流

- 日志使应用程序的运行变得透明。日志应该是事件流的汇总，即将所有运行中进程和后端服务的输出流按照时间顺序收集起来。
- 12要素应用本身并不考虑存储自己的输出流，因此不应该试图去写或者管理日志文件。相反，每一个运行的进程都会对应直接的**标准输出（Stdout）事件流**。这些事件流可以输出至文件，或者在终端实时观察。最重要的，输出流可以发送到Splunk这样的日志索引及分析系统，或Hadoop/Hive这样的通用数据存储系统。这些系统为查看应用的历史活动提供了强大而灵活的功能。

12. 后台管理任务当作一次性进程运行

- **进程构成 (Process Formation)** 是指用来处理应用的常规业务（例如处理Web请求）的一组进程。与常规业务不同，开发人员经常希望执行一些管理或维护应用的一次性任务。
- 一次性管理进程应该和正常的常驻进程一样使用同样的环境，和任何其他进程一样使用相同的代码和配置，基于某个发布版本运行。后台管理代码应该随其他应用程序代码一起发布，从而避免同步问题。
- 所有进程类型应该使用同样的依赖隔离技术。
- 12要素应用尤其青睐那些提供了REPL shell的语言，因为这会让运行一次性脚本变简单。在本地部署中，开发人员直接在命令行使用shell命令调用一次性管理进程。

8.3 云原生应用开发

8.3.1 云原生应用开发的原则

8.3.2 云原生的落地：Kubernetes

8.3.1 云原生应用开发的原则

- 云原生的开发范式是软件开发演进的一种新型范式，它不仅仅是将应用程序迁移和移植到云平台上运行，更加关注如何利用云计算并最大限度地发挥其优势。为了实现这一目标，在生产和开发过程中，软件开发相关的部门都需要认真关注如何使用云服务，进而关注并实践如何构建云原生应用。综合前面章节的内容，可以归纳云原生应用开发的几项原则。

1. 原则1：云服务优先策略

- 原则：**云服务优先策略（Cloud-First）**。
- 描述：在评估技术解决方案中的服务或组件时，首先要考察目前市场上是否有可用的云服务功能，并优先考虑使用最适合用户需求的云服务。
- 理由：将需要自己负责全新开发的软件模块数量降到最低、最合理水平。
- 参考建议：
 - 云原生的服务应该部署在云端，除非受限于一一些特殊的环境因素，如安全、合规问题，或者受限于一特殊的网络、集成需求问题。
 - SaaS适用于一些大中型应用功能，同时也支持自定义和个性化设置，这一点相对于版权许可软件来说更具灵活性。
 - 必须权衡考虑版权许可软件和开源软件。

2. 原则2：基础设施即代码

- 原则：**基础设施即代码**（Infrastructure as Code, IaC）。
- 描述：以处理应用程序代码相同的方式来管理基础设施配置以及工作流的定义。
- 理由：通过API的方式来构建环境，提供管理和执行运行环境工作流的工具，这使得环境配置可以视为软件功能的一部分。
- 参考建议：
 - 需要使用支持IaC的工具；
 - 需要为应用软件及其运行环境编写相应的测试脚本；
 - 环境的准备和配置不可以通过手动操作的方式进行。

3. 原则3：敏捷交付

- 原则：**敏捷交付（Agile Delivery）**。
- 描述：在交付过程的各个阶段争取敏捷，包括开发前的项目启动和计划阶段，以及开发后发布管理和运维管理阶段。
- 基本原理：敏捷软件开发过程通常能使产品更快地投入生产，但如果开发过程控制过于死板，项目开发就无法敏捷，只有力争各个阶段保持敏捷，才可以最大限度地提高效益。
- 理由：
 - 前期开发规划应充分考虑项目迭代周期与开发交付周期之间的呼应关系，使软件的开发过程适应敏捷开发的过程控制方式。
 - 必须设定一个初始交付目标，这个交付物必须是可以运行的工作成果。
 - 随着业务目标的调整，对于开发过程中的需求变更应该抱有开放的态度，拥抱变化。
 - 开发团队和运维团队紧密合作，力求做到频繁发布，充分采用DevOps的开发理念。
 - 快速试错，避免冗长的QA测试环节，最大程度地降低交付风险。

4. 原则4：自动化交付原则

- 原则：自动化交付原则（Delivery Automation）。
- 描述：力求在开发运维过程中做到从构建到发布的全自动化。
- 理由：实现软件构建、环境准备、测试和部署的自动化能力可以使得产品在加速市场化的过程中占据绝对的优势。
- 参考建议：
 - 这个原则建立在前面的基础设施即代码的原则之上。
 - 自动化测试工具是必需的。
 - “快速试错”的方法是为了加快部署和自动化生产。
 - 应该设计一个监控系统 and 回滚计划，以便快速检测和回退有问题的版本，而不用等待错误修复。

5. 原则5：基于服务架构

- 原则：**基于服务架构**（Service-Based Architecture）。
- 描述：必须按照既定的项目目标和期望的特点来遵循各种形式的基于服务的体系结构（SBA）。
- 理由：所有形式的基于服务的体系结构都有其优点，应该加以利用。
- 虽然在选择一种具体的形式时需要权衡，但应该考虑和评估各种服务形式，为给定的解决方案确定最合适的架构方法。
- 参考建议：
 - 为了确定基于服务的体系结构最适合的应用，在软件开发生命周期的早期就需要进行一些分析。
 - 所有形式的SBA都要求按API的规范化开发。
 - 应采用API优先开发战略。
 - 需要考虑API的接口风格的标准化的。
 - 需要考虑API的接口的安全性，并采取相应的措施保障API不暴露给不安全或不受信的网络。

6. 原则6：12要素应用

- 原则：12要素应用（Twelve-Factor Applications）。
- 描述：遵循最佳实践（如12要素应用原则），开发云原生应用程序。
- 理由：一些组织多年来一直致力于开发云原生应用程序，并开始记录最佳实践，需要吸取别人的教训，并在适当的时候采取最佳作法。
- 参考建议：
 - 构建过程，发布过程和配置管理实践可能受某些最佳实践的影响。
 - 一些最佳实践会影响应用程序的部署和管理方式，因此可能有必要查看运营团队成员的最佳实践。

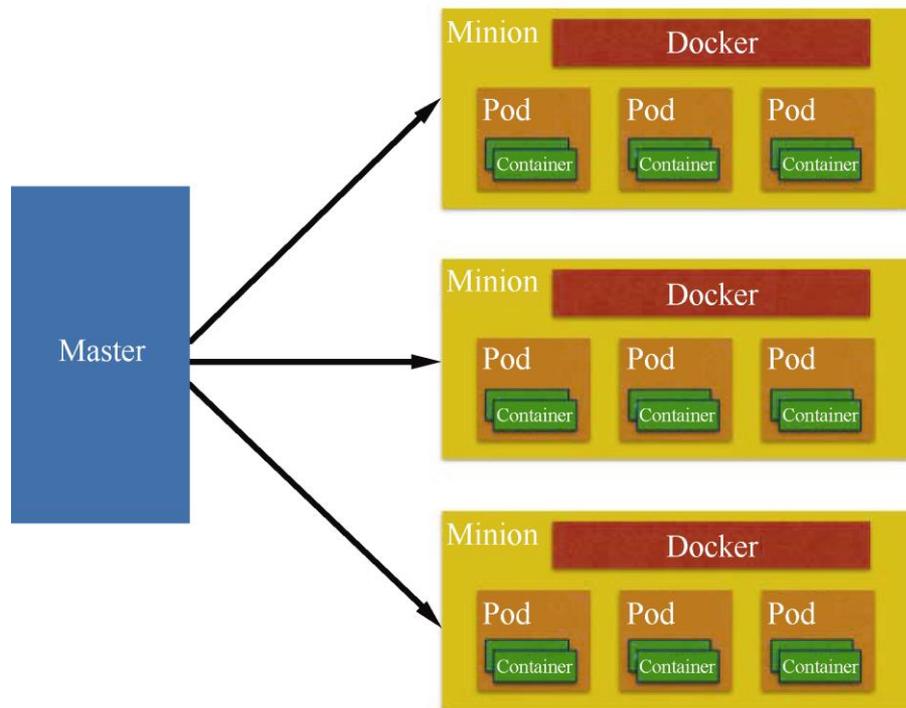
8.3.2 云原生的落地：Kubernetes

- Kubernetes是Google基于其内部使用的Borg改造的一个通用容器编排调度器，于2014年开源，并于2015年捐赠给Linux基金会下属的云原生计算基金会（CNCF）；同时它也是GIFEE（Google Infrastructure For Everyone Else）中的一员，该组织还包括了HDFS、HBase和ZooKeeper等项目。
- Kubernetes的架构做得足够开放，通过一系列的接口，如CRI（Container Runtime Interface）作为Kubelet与容器之间的通信接口，CNI（Container Networking）管理网络服务，持久化存储通过各种Volume Plugin来实现。

8.3.2 云原生的落地：Kubernetes

- Kubernetes的基本概念如下。
- Cluster: Kubernetes维护一个集群，Docker的容器运行于其上。这个集群可以运维在任何云和Bare Metal物理机上。
- Master: Master节点包含apiserver、controller-manager、sheduler等核心组件（常常也将etcd部署于其中）。
- Node: Kubernetes采用Master-Slaves方式部署，单独一台Slave机器称为一个Node（以前叫Minion）。
- Pod: Kubernetes的最小管理单位，用于控制创建、重启、伸缩一组功能相近、共享磁盘的Docker容器。虽然Pod可以单独创建使用，但是推荐通过Replication Controller管理。
- Replication Controller（RC）：管理其下控制的Pod的生命周期，保证指定数量（replicas）的Pod正常运行。
- Service: 可用作服务发现，类似于LoadBalancer，通过Selectors为一组Pod提供对外的接口。
- Label: K/V键值对，用来标记Kubernetes组件的类别关系（例如标记一组Pod是frontServices，另一组是backServices）。Label对于Kubernet的伸缩调度非常重要。

图8.14 Kubernetes的整体架构



8.3.2 云原生的落地：Kubernetes

- Kubernetes目前已经成为容器编排调度的实际标准，Docker官方和Mesos都已经支持了Kubernetes。
- Google的GKE、微软的Azure ACS、AWS的Fargate和2018年推出的EKS、Rancher联合Ubuntu推出的RKE，以及华为云、腾讯云、阿里云等都已推出了公有云上的Kubernetes服务，Kubernetes已经成为公有云的容器部署的标配，私有云领域也有众多厂商在做基于Kubernetes的PaaS平台。
- Kubernetes是云原生哲学的体现，通过容器技术和抽象的IaaS接口，屏蔽了底层基础设施的细节和差异，可实现多环境部署并在多环境之间灵活迁移。

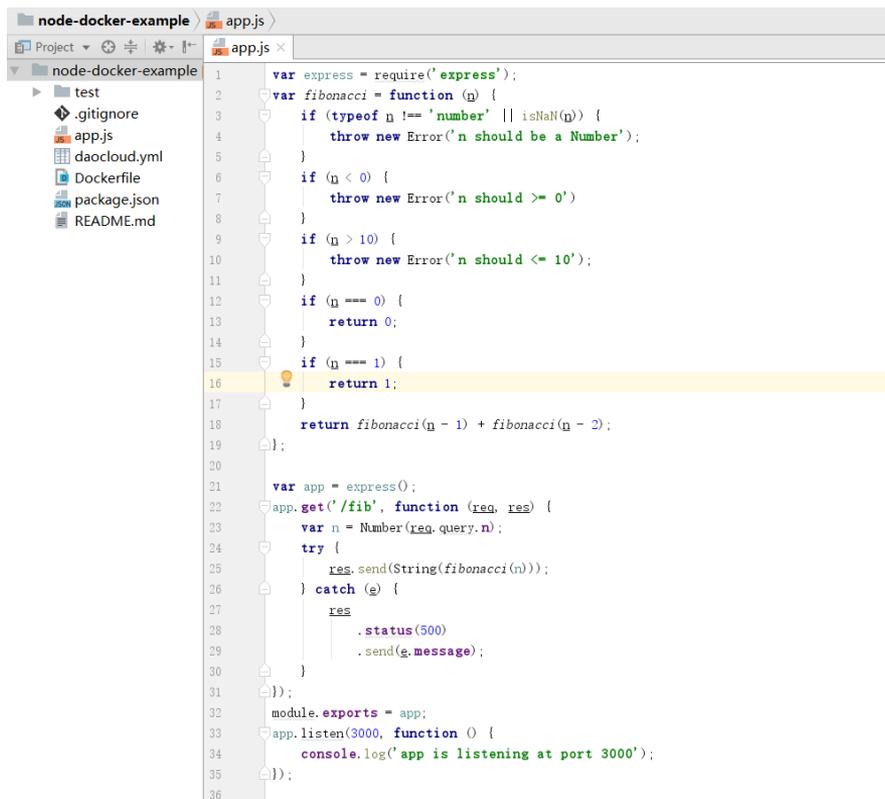
8.4 实践：Node.js云原生应用开发

- 随着云计算的火热发展，云原生应用的开发实践也成为一种趋势，目前已有很多厂商都提供了支持云原生应用开发的基础设施，本节的示例将会使用DaoCloud的服务加以说明，以一个Node.js的Hello World的简单程序来讲述实践云原生的应用开发过程。

8.4 实践：Node.js云原生应用开发

- 1. 第一步：准备Node.js程序
- 与之前的Node.js开发流程一样，首先要准备好Node.js应用的配置。
- `$ mkdir -p node-docker-example`
- `$ cd node-docker-example`
- 开始编写主要的程序。
- `// app.js`
- 2. 第二步：添加配置文件
- 我们可以给项目添加Node.js Package配置文件。
- `$ npm init -y`
- 3. 第三步：编写Dockerfile
- Dockerfile是一个Docker镜像的核心部件，所有的构建、运行入口、容器配置都依赖它。从DockerHub拉取一个Node.js的官方Docker镜像，作为基础镜像。

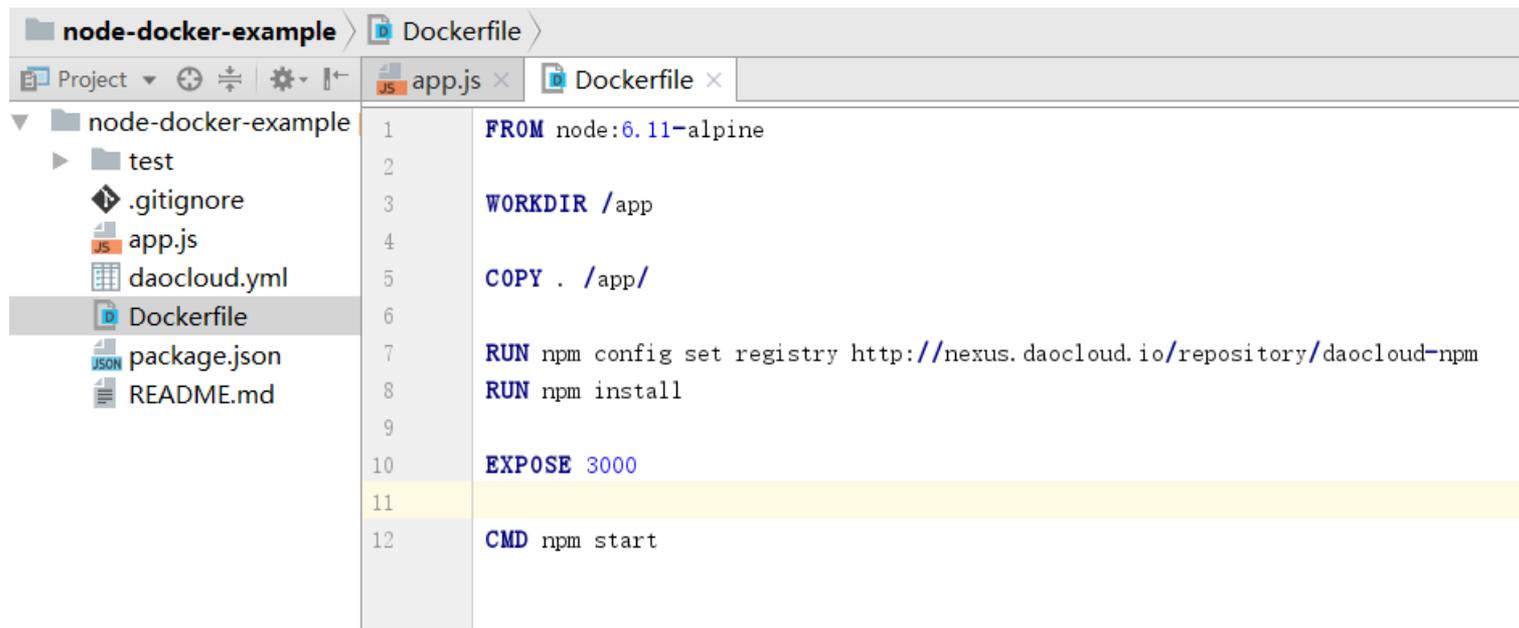
图8.15 演示程序代码结构



```
node-docker-example app.js
Project
node-docker-example
├── test
├── .gitignore
├── app.js
├── daocloud.yml
├── Dockerfile
├── package.json
└── README.md

1  var express = require('express');
2  var fibonacci = function (n) {
3    if (typeof n !== 'number' || isNaN(n)) {
4      throw new Error('n should be a Number');
5    }
6    if (n < 0) {
7      throw new Error('n should >= 0');
8    }
9    if (n > 10) {
10     throw new Error('n should <= 10');
11   }
12   if (n === 0) {
13     return 0;
14   }
15   if (n === 1) {
16     return 1;
17   }
18   return fibonacci(n - 1) + fibonacci(n - 2);
19 }
20
21 var app = express();
22 app.get('/fib', function (req, res) {
23   var n = Number(req.query.n);
24   try {
25     res.send(String(fibonacci(n)));
26   } catch (e) {
27     res
28       .status(500)
29       .send(e.message);
30   }
31 });
32 module.exports = app;
33 app.listen(3000, function () {
34   console.log('app is listening at port 3000');
35 });
36
```

图8.16 Dockerfile内容



```
node-docker-example > Dockerfile
Project  app.js  Dockerfile
node-docker-example
├── test
├── .gitignore
├── app.js
├── daocloud.yml
├── Dockerfile
├── package.json
└── README.md

1 FROM node:6.11-alpine
2
3 WORKDIR /app
4
5 COPY . /app/
6
7 RUN npm config set registry http://nexus.daocloud.io/repository/daocloud-npm
8 RUN npm install
9
10 EXPOSE 3000
11
12 CMD npm start
```

图8.17 新建DaoCloud Service项目



图8.18 关联GitHub项目

The screenshot displays the 'DaoCloud Services' interface. On the left is a dark sidebar with navigation options: '项目' (Projects), '镜像仓库' (Image Repositories), '收藏夹' (Favorites), '发现镜像' (Discover Images), '自有主机' (Self-hosted), and '用户中心' (User Center). The main content area is titled '项目名称' (Project Name) and shows 'nodejs-demo' in a text input field. Below it, the '项目镜像' (Project Image) is 'daocloud.io/jasper_chiu/nodejs-demo'. The '设置代码源' (Set Code Source) section includes a description: '代码源指定了您项目的代码仓库位置, 关联代码源后, 您对代码源的操作会自动触发项目的相应动作比如持续集成和镜像构建。' (Code source specifies the location of your project's code repository. After associating the code source, your operations on the code source will automatically trigger the corresponding actions of the project, such as continuous integration and image building.) Below this is a tabbed interface with 'Github', 'Bitbucket', 'Coding', 'GitLab', and 'Git 地址'. The 'Github' tab is active, showing a grid of repository cards. The 'nodejs-sample' card by user 'chunchill' is highlighted with a red border. Other visible cards include 'ry-bpms' by JohnGriffithLondon, 'nodeinpractice' by NatureFeng, 'SVG' by PYDebug, 'BA' by Super-Warrior, and 'jiashuhui.github.com' by jiashuhui.

图8.19 关联之前新建的GitHub项目

The screenshot shows the DaoCloud Services interface. On the left is a dark sidebar with navigation items: DevOps, 项目, 交付中心, 镜像仓库, 收藏夹, 发现镜像, 应用平台, 自有主机, 设置, and 用户中心. The main area displays a grid of project cards. The 'nodejs-sample' card is highlighted with a blue border. Below the grid, there is a section titled '发布应用镜像' with a description and three radio button options: '镜像仓库' (selected), 'Docker Hub', and '私有 Registry'. At the bottom, there is a large green button labeled '开始创建'.

DaoCloud Services

DevOps
项目
交付中心
镜像仓库
收藏夹
发现镜像
应用平台
自有主机
设置
用户中心

JohnGriffithLondon
ry-bpms

NatureFeng
nodeinpractice

PVDebug
SVG

Super-Warrior
BA

chunchill
nodejs-sample

jiashuhui
jiashuhui.github.com

xujihui1985
nodeinpractice

发布应用镜像

应用镜像是打通应用开发和业务运维之间通路的关键交付件，也是应用统一发布的重要一环。为了方便您管理这一核心资产，应用构建成功后，系统会自动把应用镜像发布在 DaoCloud 镜像仓库（您的私有空间）中，同时也支持将应用镜像发布到第三方镜像仓库中。

镜像仓库 Docker Hub 私有 Registry

开始创建

图8.20 定义CI流程

The screenshot displays the 'nodejs-demo' project configuration in DaoCloud Services. The interface is divided into a left sidebar, a main configuration area, and a right-hand settings panel.

- Left Sidebar (Navigation):**
 - DaoCloud Services
 - DevOps
 - 项目
 - 交付中心
 - 镜像仓库
 - 收藏夹
 - 发现镜像
 - 应用平台
 - 自有主机
 - 设置
 - 用户中心
- Main Configuration Area:**
 - Repository: nodejs-demo (镜像: nodejs-demo, 代码源: chunchill/nodejs-sample)
 - Actions: push 代码触发或, 手动触发
 - Workflow Definition: 测试阶段 (test) and 构建阶段 (build)
 - Buttons: 添加并行任务
- Right Panel (Settings):**
 - 全局镜像
 - 环境变量
 - 使用 submodule
 - 构建集群 IP
 - 通过 yaml 快捷编辑
 - 切换至本地 yaml

图8.21 daocloud.yml内容

DaoCloud Services

DevOps

- 项目
- 文件中心
- 镜像仓库
- 收藏夹
- 发现镜像
- 应用平台
- 自有主机
- 设置
- 用户中心

nodejs-demo

镜像: nodejs-demo 代码源: chunchill/nodejs-sample

push 代码触发或 手动触发

执行记录 流程定义 设置

```
version: 3
stages:
- 测试阶段
- 构建阶段
默认构建任务:
  label: build
  stage: 构建阶段
  job_type: image_build
  only:
    branches:
    - "*"
    tags:
    - "*"
  build_dir: /
  cache: true
  dockerfile_path: /Dockerfile
默认测试任务:
  label: test
  stage: 测试阶段
  job_type: test
  only:
    branches:
    - "*"
    tags:
    - "*"
  image: node:6.11.0-wheezy
  script:
  - npm install
```

使用 submodule
构建集群 IP

切换至云端定义

图8.22 手动触发流水线



图8.23 流水线执行过程中

The screenshot displays the DaoCloud Services DevOps interface. On the left is a dark sidebar with navigation options: DaoCloud Services, DevOps, 项目 (Projects), 交付中心 (Delivery Center), 镜像仓库 (Image Repository), 收藏夹 (Favorites), 发现镜像 (Discover Images), 应用平台 (Application Platform), 自有主机 (Self-hosted), 设置 (Settings), and 用户中心 (User Center). The main content area shows a table of projects with columns for '项目名称' (Project Name), '最近更新' (Last Updated), '代码仓库' (Code Repository), and '执行状态' (Execution Status). The 'nodejs-demo' project is highlighted with a red box around its status '正在执行' (Running). Below the table are pagination controls: '< 上一页' (Previous Page), '下一页 >' (Next Page), and '第 1 页, 共 1 页' (Page 1 of 1). The top right corner shows a search bar and a user profile for 'jasper_chiu'.

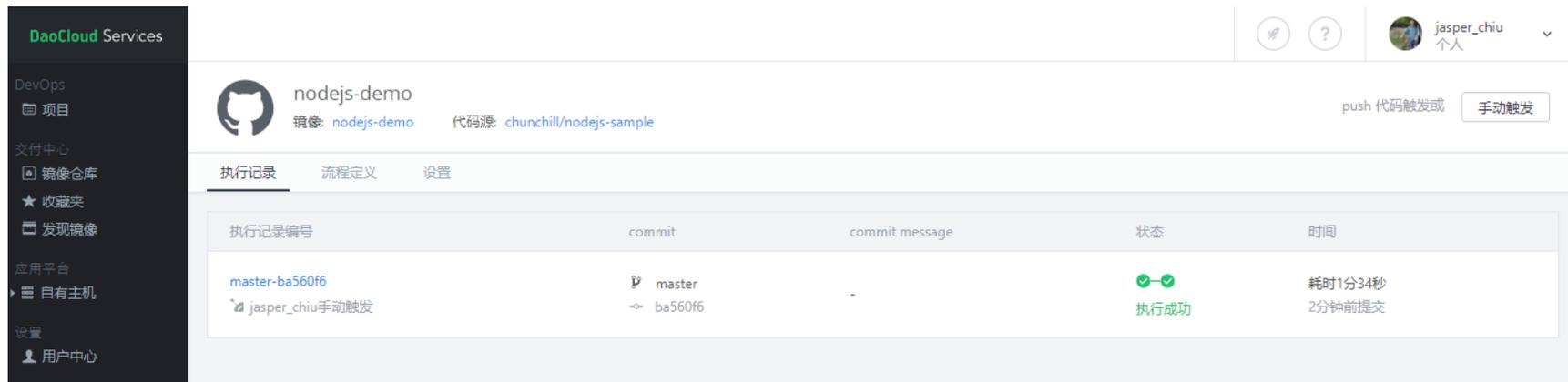
项目名称	最近更新	代码仓库	执行状态
nodejs-demo master-ba560f6	大约1分钟前	chunchill/nodejs-sample	正在执行
python-mysql-sample master-07f3d43	9月前	Super-Warrior/python-mysql-sample	执行成功
example master-a2c564e	11月前	chunchill/dao-2048	执行成功

图8.24 流水线执行成功

The screenshot displays the DaoCloud Services interface. On the left is a dark sidebar with navigation options: DevOps, 项目, 交付中心, 镜像仓库, 收藏夹, 发现镜像, 应用平台, 自有主机, 设置, and 用户中心. The main content area shows a list of pipeline jobs. At the top, there is a '创建新项目' button and '共 3 个项目'. A search bar is also present. The job list has columns for '项目名称', '最近更新', '代码仓库', and '执行状态'. The first job, 'nodejs-demo', has its '执行成功' status highlighted with a red box. Below the list are pagination controls: '< 上一页', '下一页 >', and '第 1 页, 共 1 页'.

项目名称	最近更新	代码仓库	执行状态
nodejs-demo master-ba560f6	2分钟前	chunchill/nodejs-sample	执行成功
python-mysql-sample master-07f3d43	9月前	Super-Warrior/python-mysql-sample	执行成功
example master-a2c564e	11月前	chunchill/dao-2048	执行成功

图8.25 查看构建状态



The screenshot displays the DaoCloud Services interface for a project named 'nodejs-demo'. The interface includes a sidebar with navigation options like '项目', '镜像仓库', and '发现镜像'. The main content area shows the project details, including the repository name 'nodejs-demo', its mirror 'nodejs-demo', and the code source 'chunchill/nodejs-sample'. Below this, there are tabs for '执行记录', '流程定义', and '设置'. The '执行记录' tab is active, showing a table of build records. The table has columns for '执行记录编号', 'commit', 'commit message', '状态', and '时间'. A single record is shown with a successful status and a duration of 1 minute and 34 seconds.

执行记录编号	commit	commit message	状态	时间
master-ba560f6 jasper_chiu手动触发	master ba560f6	-	执行成功	耗时1分34秒 2分钟前提交

图8.26 部署构建好的版本

The screenshot displays the DaoCloud Services interface. On the left is a dark sidebar with navigation options: DevOps, 项目, 镜像仓库 (highlighted with a red box), 收藏夹, 发现镜像, 应用平台, 自有主机, 集群管理, 应用, Stack, 设置, and 用户中心. The main content area shows a list of images under the '本地 Push 镜像' tab, with a search bar and a '部署最新版本' button circled in red for the 'nodejs-demo' image.

镜像名称	镜像地址	版本	操作
nodejs-demo 更新于 2分钟前	镜像: daocloud.io/jasper_chiu/nodejs-demo 版本: master-ba560f6		部署最新版本
python-mysql-sample 更新于 9月前	镜像: daocloud.io/jasper_chiu/python-mysql-sample 版本: master-07f3d43		部署最新版本
example 更新于 11月前	镜像: daocloud.io/jasper_chiu/example 版本: master-a2c564e		部署最新版本

图8.27 对接部署服务器

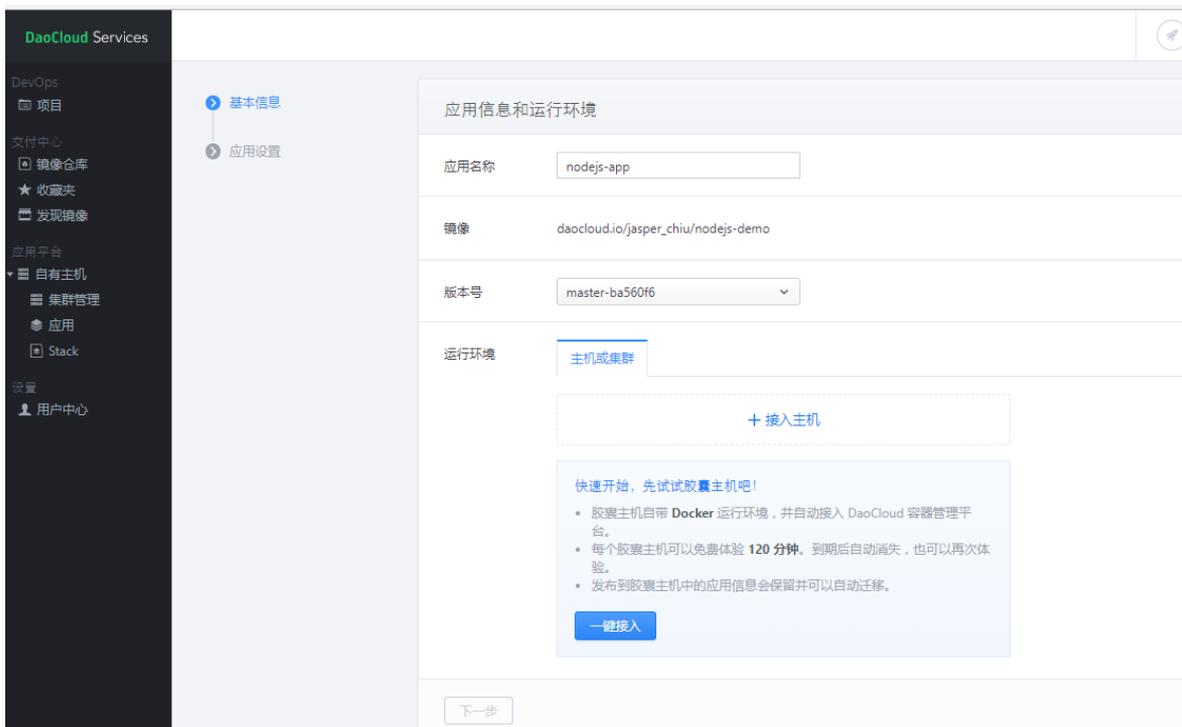


图8.28 对接DaoCloud胶囊主机

运行环境

主机或集群

+ 接入主机

快速开始，先试试胶囊主机吧！

- 胶囊主机自带 **Docker** 运行环境，并自动接入 DaoCloud 容器管理平台。
- 每个胶囊主机可以免费体验 **120 分钟**。到期后自动消失，也可以再次体验。
- 发布到胶囊主机中的应用信息会保留并可以自动迁移。

⚙ 正在接入 预计还剩 44 s

图8.29 随机生成DaoCloud胶囊主机

应用信息和运行环境

应用名称

镜像 daocloud.io/jasper_chiu/nodejs-demo

版本号

运行环境

主机 IP: 10.23.223.147,172.17.0.1

图8.30 配置映射端口

DaoCloud Services

jasper_chiu 个人

DevOps

项目

交付中心

镜像仓库

收藏夹

发现镜像

应用平台

自有主机

集群管理

应用

Stack

设置

用户中心

基本信息

应用设置

应用设置

端口

容器端口	协议	主机端口	是否发布
3000	TCP	动态端口	是

[添加容器端口](#)

容器端口会映射到主机端口上。

Volumes

容器路径	主机路径	是否可写
您还没有添加 Volume		

[添加 Volume](#)

自定义环境变量

[手动输入](#) [导入 YAML](#)

键 - 值

[显示高级设置](#)

[立即部署](#) [返回上一步](#)

图8.31 运行成功结果

The screenshot displays the DaoCloud Services interface for a deployment named 'nodejs-app'. The status is '运行中' (Running), indicated by a green dot and the text '创建于1分钟之内' (Created within 1 minute). The deployment details include the image 'daocloud.io/jasper_chiu/nodejs-demo' and environment 'Try_DaoCloud_1'. The logs show the following sequence of events:

```
2018-01-21 21:35:50 eb03949b4ac3: Extracting [=====>] 6.226 MB/15.44 MB
2018-01-21 21:35:51 eb03949b4ac3: Extracting [=====>] 14.58 MB/15.44 MB
2018-01-21 21:35:51 eb03949b4ac3: Extracting [=====>] 15.07 MB/15.44 MB
2018-01-21 21:35:51 eb03949b4ac3: Extracting [=====>] 15.44 MB/15.44 MB
2018-01-21 21:35:51 eb03949b4ac3: Pull complete
2018-01-21 21:35:51 7cb05f3b6429: Extracting [=>] 32.77 kB/1.016 MB
2018-01-21 21:35:52 663b3d1fcb84: Extracting [=====>] 92 B/92 B
2018-01-21 21:35:53 dd197113ec57: Extracting [=====>] 1.697 kB/1.697 kB
2018-01-21 21:35:53 dd197113ec57: Pull complete
2018-01-21 21:35:53 014895872457: Extracting [=====>] 852 kB/2.813 MB
2018-01-21 21:35:53 014895872457: Extracting [=====>] 2.596 MB/2.813 MB
2018-01-21 21:35:53 Status: Downloaded newer image for daocloud.io/jasper_chiu/nodejs-demo:master-ba560f6
2018-01-21 21:35:55 Creating dao_nodejs-app_1...
2018-01-21 21:35:57.
2018-01-21 21:35:57 > nodejs-sample@0.0.1 start /app
2018-01-21 21:35:57 > node app.js
2018-01-21 21:35:57.
2018-01-21 21:35:57 app is listening at port 3000
```

图8.32 运行成功后状态

The screenshot displays the DaoCloud Services dashboard. On the left is a dark sidebar with navigation options: DevOps, 项目, 交付中心, 镜像仓库, 收藏夹, 发现镜像, 应用平台, 自有主机, 集群管理, 应用, Stack, 设置, and 用户中心. The main content area shows the details for a service named 'Try_DaoCloud_1'. It features a red Ubuntu logo and a '还能试用: 01:56:07' timer. The status is '正常' (Normal). Technical details include IP addresses (10.23.223.147, 172.17.0.1, 52.80.161.165), hostname (ip-10-23-223-147), Docker version (1.13.0), connection info (ssh ubuntu@52.80.161.165), OS (Ubuntu 14.04.2 LTS), and cluster (默认). Below this, a tabbed interface shows '容器' (Containers) selected, displaying a table with one container: 'dao_nodesjs-app_1' (ID: 9f77198efaa1) running the image 'daocloud.io/jasper_chiu/nodejs-demo:master-ba560f6' with command '/bin/sh -c 'npm start''. The container is 'Up' and has port '32768 -> 3000'. A search bar and '查看详情' (View Details) button are also present.

DaoCloud Services

DevOps

- 项目
- 交付中心
- 镜像仓库
- 收藏夹
- 发现镜像
- 应用平台
- 自有主机
 - 集群管理
 - 应用
 - Stack
- 设置
- 用户中心

Try_DaoCloud_1

还能试用: 01:56:07

状态: 正常

IP: 10.23.223.147, 172.17.0.1 外网: 52.80.161.165

主机名: ip-10-23-223-147

Docker 1.13.0: 运行正常

连接主机: ssh ubuntu@52.80.161.165 密码: 61c4a9055d58a31e

操作系统: Ubuntu 14.04.2 LTS

集群: 默认

容器 镜像 网络 Volume 监控 设置

共 1 个容器, 已选中 0 个容器

容器	镜像	端口	状态
dao_nodesjs-app_1 9f77198efaa1	daocloud.io/jasper_chiu/nodejs-demo:master-ba560f6 /bin/sh -c 'npm start'	32768 -> 3000	Up Less than a second

查看详情

小结



summary

- 云原生的相关概念
- 云原生应用开发实践的12要素
- 云原生应用开发
- 实践：基于Node.js的云原生应用开发

课内复习

1. 什么是云原生？
2. 云原生包括哪几个方面的内容？
3. 什么是持续集成与持续交付？
4. 云原生的12要素是什么？

课外思考

1. 相对于传统云应用，云原生应用的优势是什么？
2. 为什么Docker和Kubernetes技术成为云原生落地的最佳实践之一？

动手实践

Node.js是一个基于Chrome V8引擎的JavaScript运行环境。Node.js使用了一个事件驱动、非阻塞式I/O的模型，既轻量又高效。Node.js的包管理器npm是全球最大的开源库生态系统。

- 任务：通过Node.js的项目网站（<https://github.com/nodejs>）了解并使用Node.js进行编程。
- 任务：在Docker环境中部署Node.js，并开发或部署一个简单的网站系统。

论文研习

- 阅读“论文阅读”部分的论文[40]，全面了解云端Web应用的自动扩展机制。
- 阅读“论文阅读”部分的论文[42]，理解面向云计算软件工程的挑战与未来方向。