**亚信大数据技术框架及标准**

**版本号：**0.0.0

2015-10-30

技术委员会

目 录

[1 背景 4](#_Toc434349352)

[2 目标 4](#_Toc434349353)

[3 总体原则 5](#_Toc434349354)

[4 架构风格 5](#_Toc434349355)

[4.1 应用和后端服务分离 5](#_Toc434349356)

[4.2 应用内部的解耦 6](#_Toc434349357)

[4.3 代码和配置分离 7](#_Toc434349358)

[4.4 弹性扩展和高可用 8](#_Toc434349359)

[4.5 健壮性设计 8](#_Toc434349360)

[5 集成框架 9](#_Toc434349361)

[5.1 服务框架 9](#_Toc434349362)

[5.2 应用框架 10](#_Toc434349363)

[6 技术标准 11](#_Toc434349364)

[6.1 应用标准 11](#_Toc434349365)

[6.1.1 应用开发 11](#_Toc434349366)

[6.1.2 UI 11](#_Toc434349367)

[6.1.3 应用日志 12](#_Toc434349368)

[6.1.4 应用部署 12](#_Toc434349369)

[6.2 后端服务标准 13](#_Toc434349370)

[6.2.1 服务上架（编目） 14](#_Toc434349371)

[6.2.2 服务创建 14](#_Toc434349372)

[6.2.3 服务回收 14](#_Toc434349373)

[6.2.4 服务绑定 14](#_Toc434349374)

[6.2.5 服务解绑 14](#_Toc434349375)

[6.2.6 详细接口信息 15](#_Toc434349376)

[6.3 安全标准 15](#_Toc434349377)

[7 路线图 15](#_Toc434349378)

[7.1 近期试运行阶段 15](#_Toc434349379)

前 言

本文旨在指导亚信大数据各产品线高效地进行大数据技术产品的开发、测试、部署和运维。主要内容为明确亚信大数据的总体技术框架，明确架构风格、技术原则和接口原则，并给出实施路线。

# 背景



亚信大数据改变了以前以区域项目为主的经营模式，变革为产品线+行业线的矩阵式管理模式，希望能将积累的大数据能力通过产品的方式提供给客户，降低边际成本、提高利润率。

当前有以下问题亟待解决：

* 产品各自建立了自己的技术栈，基本上从底到上开发，存在组件重复
* 产品之间缺少集成框架，无论在开发过程还是部署时难以互相调用
* 产品开发过程未实现敏捷，未应用持续集成和持续部署，效率较低

# 目标

通过规划和实施亚信大数据技术框架，将现有各条产品线的大数据产品集成为“亚信大数据套件”，实现大数据应用的快速开发，实现在各个实施点的快速部署，建立数据和应用的生态，助力亚信大数据战略目标的实现。主要包括如下目标：

* 针对单一产品，制定技术架构风格应用最佳实践，制定部分标准要求
* 针对产品套件，制定集成框架和接口标准便于产品之间的相互调用和集成
* 针对产品开发过程，指导进行代码管理、开发过程管理、持续集成和持续部署

# 总体原则

**提高效率**：技术框架的首要目的是要将大数据发挥价值的速度加快

**降低成本**：其次的目的是低成本的实现

**鼓励协作**：我们应当鼓励并推动协作，包括产品间的协作、产品和外部的合作，这要求保持足够的开放性

**拥抱开源**：没有必要重复造轮子，开源的生态已经摧毁了私有软件方案，我们应当主动拥抱开源，才能打造生态

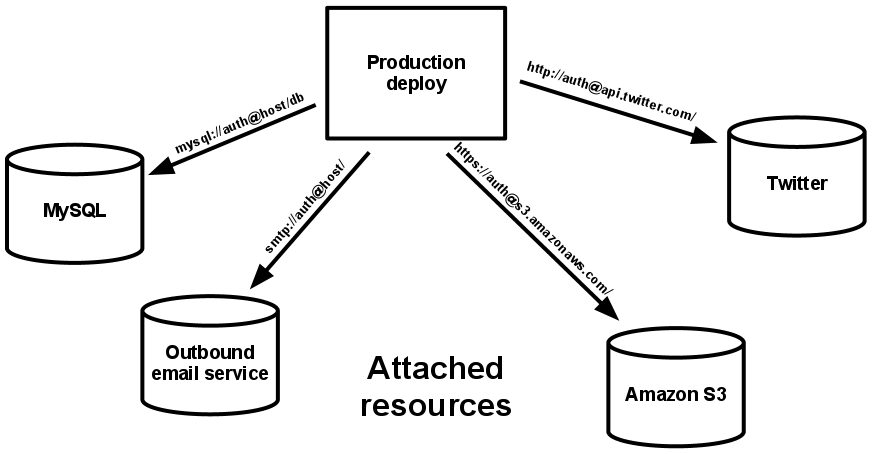
# 架构风格

要达到上述目标，亚信大数据的单个产品架构必须按照“高内聚、松耦合”的原则构建，具体包括如下。

## 应用和后端服务分离

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），以及缓存系统（Memcached）。

因为后端服务通常是共用的，且每个地方的资源有所区别，所以未来要实现和本地定制化内容的解耦，必须**将应用逻辑和后端服务分离**。**无论是本地后端服务还是远程后端服务（例如互联网OPEN API），我们都可以将其视为资源**，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，这样应用可以部署在多个地方而不需要进行任何代码改动，同时也可以方便地将本地 MySQL 数据库换成第三方服务（例如 Amazon RDS）。

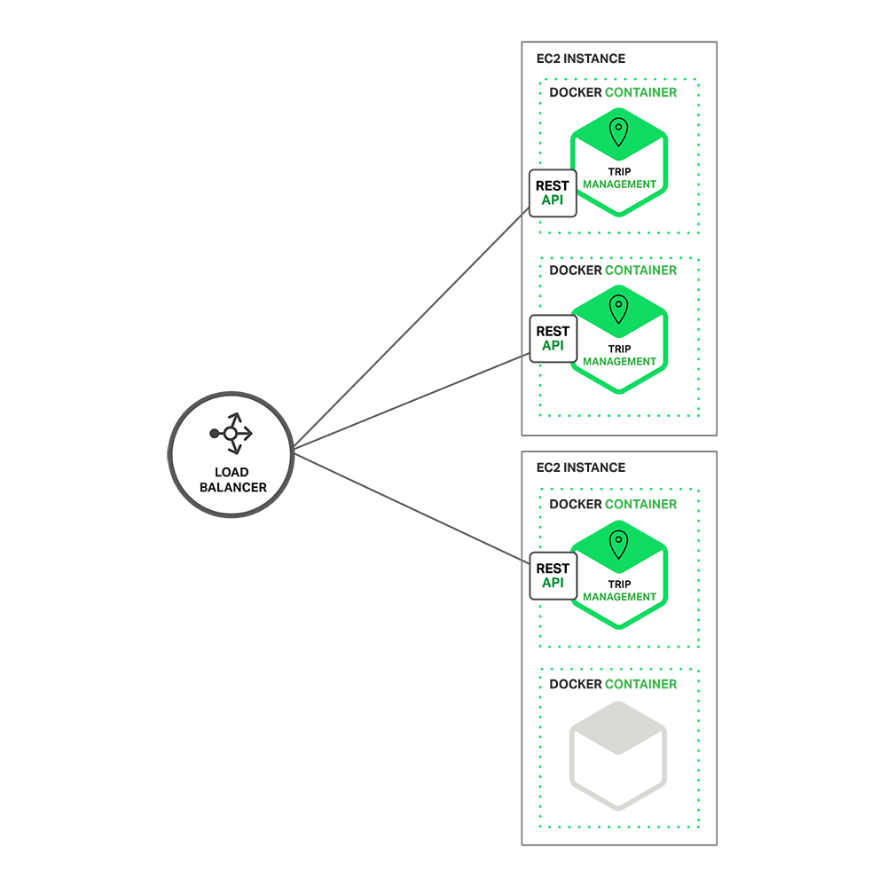


## 应用内部的解耦

为了实现应用的松耦合、支持快速迭代，应用本身的规模也需要进行拆分。拆分的原则可能包括：技术栈不一样、迭代周期不一样和扩展规模的不一样。将应用按照“微服务”的方式进行解耦，并通过服务发现的方式进行通信，能享受以下好处：

* 降低了复杂度：由于解耦，单个服务开发难度降低，因此系统整体复杂度降低了。
* 使用不同的技术栈：每个开发团队可能有自己最熟悉的技能，每个模块可能有最适合的后端服务，如果解耦则可以灵活选择。
* 可扩展性强：除了每个“服务”均可以自行扩展到相应规模以外，由于解耦，使得某个服务的迭代不会影响其他服务。
* 可以更好支持前端：对于前端JS框架、H5的移动页面、原生移动客户端，微服务均可以统一支撑。

微服务架构示例如下：



Docker是很好的微服务封装的标准。

解耦的各个微服务之间通过REST风格的API进行互相通信，使用与后端服务同样的服务发现框架进行服务的注册、查找和路由。

## 代码和配置分离

通常，应用的配置在不同部署(预发布、生产环境、开发环境等等)间会有很大差异。这些配置可能包括：

* 数据库，Memcached，以及其他后端服务的配置
* 第三方服务的证书，如 Amazon S3、Twitter等
* 每份部署特有的配置，如域名等

有些应用在代码中使用常量保存配置，这就使得应用的部署需要修改代码。为了实现一份基准代码、多这个部署，**我们需要实现在代码库中代码文件和配置的分离。**

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基准代码是否可以立刻开源，而不用担心会暴露任何敏感的信息。

**我们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，却不动一行代码；与配置文件不同，不小心把它们提交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（比如 Java 的属性配置文件）相比，环境变量与语言和系统无关。

**将配置存储在环境变量中，还提供了一个简单的机制以便应用框架为应用传递信息**。

## 弹性扩展和高可用

避免任何单点故障，任何组件均可以运行一到多个实例，其他调用通过服务发现来访问这些实例，实例可以分别承担负载。

要实现这个需求，要求组件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，比如数据库。

如果组件使用内存区域或磁盘空间作为缓存，应不能考虑这些缓存的内容是不是可以保留给之后的请求来使用，这是因为将来的请求多半会由其他组件来服务。即使在只有一个组件，先前保存的数据（内存或文件系统中）也会因为重启（如代码部署、配置更改、或运行环境将进程调度至另一个物理区域执行）而丢失。

同样，不应依赖 “粘性 session”( 这是指将用户 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)。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。

新的组件实例启动后应当主动向服务发现功能进行注册，注册的内容包括IP地址和端口等。组件实例失效后应当及时从服务发现的列表中移除，可以通过组件向zookeeper或etcd等定期更新带TTL的节点，并由confd等进程watch并修改服务发现的配置文件实现。

## 健壮性设计

应用的进程应该可以瞬间开启或停止。 这有利于快速、弹性的伸缩应用，迅速部署变化的代码或配置 ，稳健的部署应用。

进程应当追求最小启动时间。理想状态下，进程从敲下命令到真正启动并等待请求的时间应该只需很短的时间。更少的启动时间提供了更敏捷的发布以及扩展过程，此外还增加了健壮性，因为进程管理器可以在授权情形下容易的将进程搬到新的物理机器上。

**进程一旦接收终止信号（SIGTERM）就会优雅的终止**。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。此类型的进程所隐含的要求是HTTP请求大多都很短(不会超过几秒钟)，而在长时间轮询中，客户端在丢失连接后应该马上尝试重连。

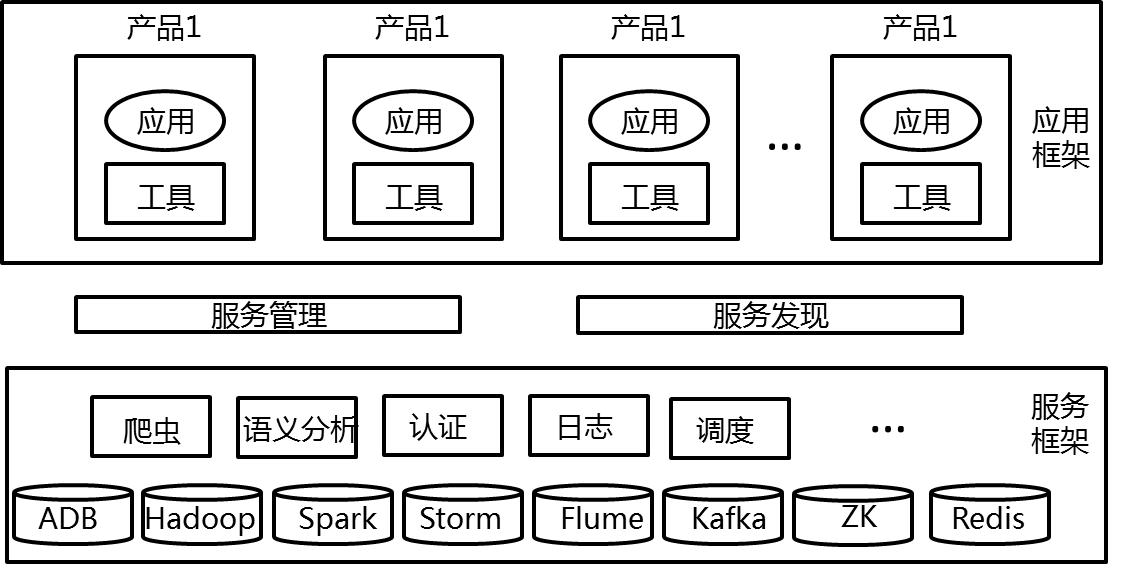
对于 worker 进程来说，优雅终止是指将当前任务退回队列。例如，RabbitMQ 中，worker 可以发送一个NACK信号。 Beanstalkd 中，任务终止并退回队列会在worker断开时自动触发。有锁机制的系统诸如 Delayed Job 则需要确定释放了系统资源。此类型的进程所隐含的要求是，**任务都应该可重复执行**， 这主要由将结果包装进事务或是使重复操作**幂等**来实现。

进程还应当在面对突然死亡时保持健壮，例如底层硬件故障。虽然这种情况比起优雅终止来说少之又少，但终究有可能发生。一种推荐的方式是使用一个健壮的后端队列，例如 Beanstalkd ，它可以在客户端断开或超时后自动退回任务。无论如何，应用都应该可以设计能够应对意外的、不优雅的终结。

# 集成框架

站在企业的角度，将现有产品或应用进行上述改造还不够，应用和应用之间需要交互，而其中共性的后端服务可以统一提供。此外应用从开发、部署到运维有一些通用的模块需要统一起来，或者是为了提高开发者的效率，或者是为了给最终使用用户一个一致的体验。

因此需要一个企业级的集成框架来解决上述问题，即一个服务于开发者的平台，一个在开发、部署和运维应用需要的“操作系统”。



## 服务框架

服务框架实现对“通用服务”和“共性工具”等各产品中的标准件的统一供给，从而使产品解耦并关注自身应用逻辑，提高应用型产品的开发迭代速度。

服务框架主要提供：

* 服务管理：服务生成、回收等生命周期管理，服务的配额和计费，服务的绑定和解除绑定（用于传递服务入口和鉴权参数）
* 服务发现：服务发现及路由能力（本身作为一个特殊服务存在，应用访问服务不一定通过它，可以直达，其也能用于应用之间的服务发现），用于实现服务的高可用、负载均衡等
* 框架能力：服务上架（MarketPlace）、服务编排（含模板）、资源管理与部署（对于内部服务）、安全认证（与应用）、日志、健康管理、配置管理等用于支持服务的能力

服务是像插件一样插在服务框架之中，服务本身可有两种实现方式：外部服务集成，通过Service Broker的模式将PaaS平台外部的服务集成到PaaS平台之中来； 内部服务模式，根据模板和镜像通过PaaS平台预定义的脚本或编排文件自动部署服务。

无论是外部服务还是内部服务模式，仅需要为服务框架提供如下接口:

* 服务上架。将服务入口、SLA、价格计划注册到服务市场（Marketplace）。
* 服务生命周期管理。编目列举（SLA和价格）、生成、回收
* 服务管理。日志、管理配置界面、健康检查等接口，最简单可以就是一个带认证Token的URL

## 应用框架

应用框架提供了应用型产品的开发、测试和运行服务，通过预定义的runtime支持实现Code2Cloud，通过内嵌的代码库、CI\CD工具实现快速迭代和DevOps。

应用框架主要提供：

* runtime：各种预定义的runtime统一了企业内应用交付环境，将各种语言编译、运行环境、各种库和依赖包进行统一提供
* 框架能力：动态路由、自动扩展、高可用、应用编排（含模板）、资源管理与部署（应用容器部署） 、配置管理、日志、监控、计费、安全认证（与用户）
* 敏捷开发：代码库管理、开发流程管理、代码审查、持续集成（单元、集成、压力测试）
* 持续交付：持续交付、灰度发布、版本管理（镜像管理）

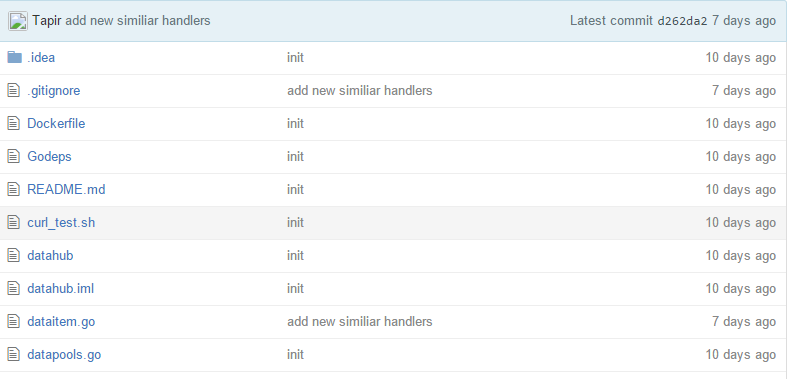
# 技术标准

## 应用标准

### 应用开发

应用应按照微服务的方式进行解耦，应用需要是无状态、可以弹性扩展和高可用的，参考第4章。

应用均通过Docker的方式进行交付，应用代码的根目录下放置Dockerfile用于代码的打包。



应用对后端服务的访问不通过配置文件等方式，而从环境变量中读出，以便灵活部署到其他平台,示例如下：

Java：



GOlang：



### 应用UI

**TBD**

### 应用日志

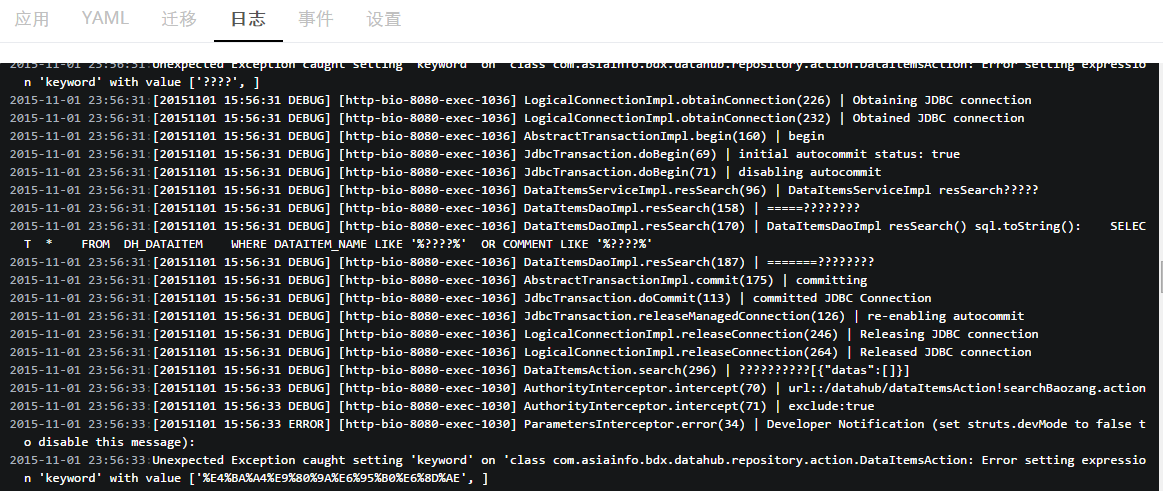
为了实现解耦和灵活部署的目的，应当避免将日志写入本地硬盘，甚至不应该写入持久化的后端服务。应该将日志视为事件流，按照时间的顺序输出到标准输出STDOUD和标准错误输出STDERR。

这样在开发环境中，开发人员可以通过这些数据流，实时在终端看到应用的活动。在预发布或线上部署中，每个应用进程的输出流由应用框架截获，并送到统一的日志采集和展示服务中，这样可以将一系列的微服务的日志统一展示或追朔分析。同时不同的运行环境也不会造成更改代码来记录日志的事情发生。日志保存的位置、周期等信息对应用来说不可见也不可配置。

日志输出的格式需要规范，需要包括如下要素，以TAB字符分割

* 时间戳
* 日志级别：DEBUG、INFO、WARNGING、ERROR、FATAL
* 应用名
* 实例名
* 日志编号
* 日志详细信息

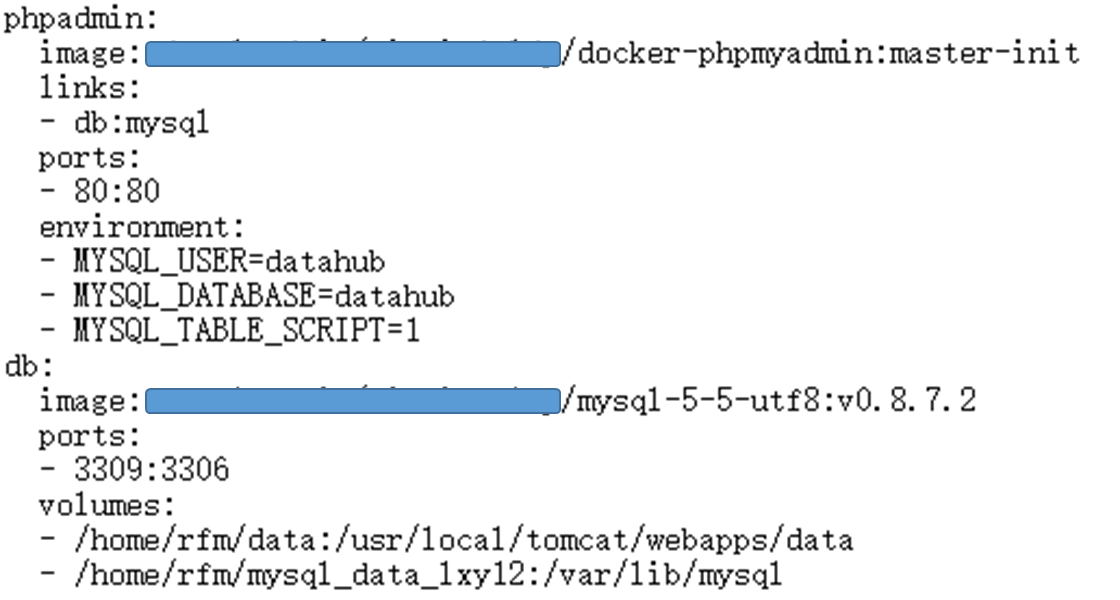
PaaS平台输出日志详情：



### 应用部署

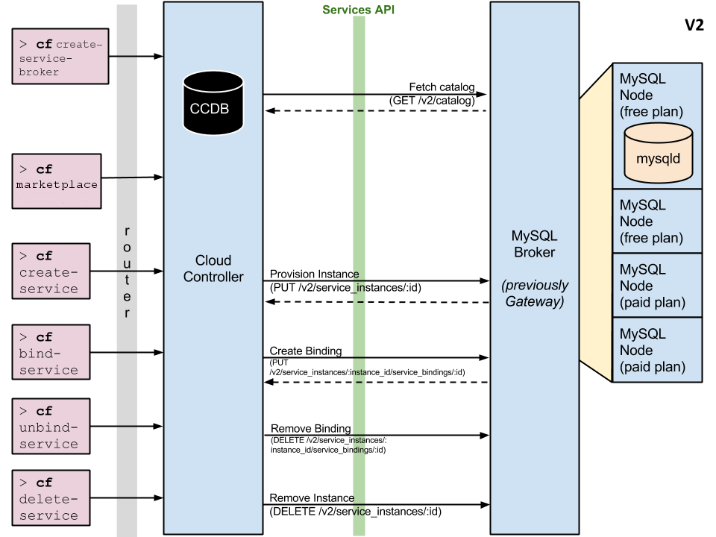
为了实现自动部署和持续交付，应用部署需要通过定义部署文件完成，在部署文件中需要定义所部署应用的镜像或镜像的组合，定义部署应用需要用到的环境变量以及必要的初始化信息。PaaS平台通过对部署文件的解析和执行完成应用镜像的拉取，服务的串接编排，应用的初始化等工作

部署文件定义示例



## 后端服务标准

PaaS平台提供了五类接口帮助开发者与后端服务平台建立连接，后端服务接入平台包括5方面的操作，服务上架、服务生成、服务回收、服务绑定、服务解绑。



### 服务上架（编目）

服务上架的过程也即服务注册到PaaS平台的过程，通过服务上架可以让开发者了解服务所提供资源类型，例如存储、计算、消息队列等，同时让开发者了解服务提供的资源粒度及需开发支付的成本。

### 服务创建

主要是PaaS平台将开发者订购服务的信息传递到服务平台进行服务资源分配的过程，当PaaS收到开发者订购服务的请求后，会将请求消息传递到服务平台，服务平台根据开发者订购服务的信息，例如存储、CPU、内存等信息，为开发者在服务平台上分配相应资源，并将资源分配的结果反馈到PaaS平台。

服务的创建要求服务平台支持多租户或多用户能力，为了支撑开发者通过PaaS平台订购后端服务，要求后端服务具备自动化的用户创建、资源分配、日志信息、高可用管理等能力。

### 服务回收

当开发者不再需要使用某项服务后，可以通过PaaS平台释放服务，服务释放后可以终止计费等操作。

### 服务绑定

服务绑定是用户把服务资源应用到自己开发的应用或服务的过程，这时PaaS平台会想服务平台请求获取以分配服务资源的连接鉴权信息，例如JDBC连接串、hdfs集群namenode节点信息等，开发者可以凭借连接信息使用已订购的服务。服务平台通过必要的鉴权手段保证合法使用平台服务资源。

服务绑定后开发者就正式开始使用服务资源，因此要求服务平台具备稳定的服务质量，一方面能够及时响应服务接入类的接口，另一方面需要能够保证用户的连续、高效实用。

### 服务解绑

服务绑定的反过程，当某个应用或服务不再使用某项服务后，可以通过PaaS平台进行应用层面的资源释放。

### 详细接口信息

http://docs.cloudfoundry.org/services/api.html

## 安全标准

* Oauth2认证技术，PaaS平台广泛使用Oauth2协议进行应用与服务、服务与服务间的鉴权认证，因此PaaS平台需要提供Oauth2认证客户端和服务端能力；
* TBD

# 路线图

考虑到现有产品和服务的提供，将分两个阶段演进到新技术框架和标准。

* 近期试运行阶段，至2015年12月底。选取部分后端服务和前端应用进行改造和迁移。
* 正式运行阶段，至2016年6月底。所有适用产品全面向新架构进行改造和迁移。

以上两个阶段均仅对亚信内部和受邀的企业客户提供PaaS服务，2016年6月后发布大数据PaaS公有云服务。

## 近期试运行阶段

**后端服务实施计划**：爬虫服务、文本分词服务、MYSQL DB、PG DB、MongoDB、Redis、HDFS、Spark、ceph、ActiveMQ、Kafka、Zookeeper、etcd。

**应用实施计划**：DataHub Server端、DataHub 数据托管区、DataHub 自有产品生成、自助分析(讲述数据故事、Zepplin Notebook和R)、人流监控。