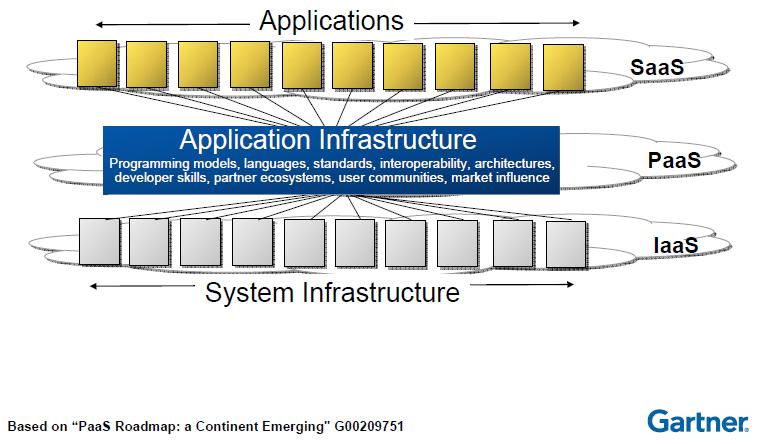
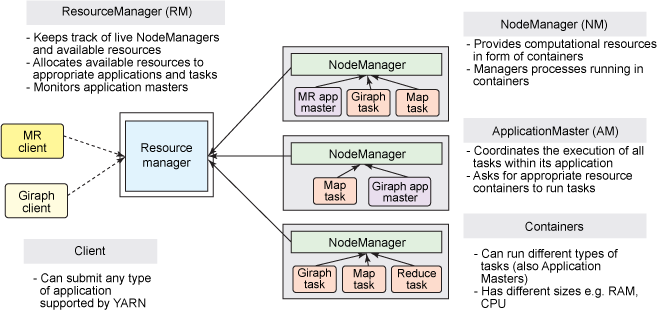
YARN Registry简介

# 1、简介

PASS，平台即服务，把服务器平台作为一种服务提供的商业模式，将软件研发的平台作为服务，以SAAS的模式提交给用户，可以理解成中间件服务，如下图所示：



为了解决MR可伸缩性瓶颈，高效的使用Hadoop集群的资源，引入了YARN组件，将资源管理和任务的协调分开，YARN的框架如下图所示：



RM、NM和容器都不关心应用程序或任务的类型，所有特定于应用程序的代码都转移到AM，以便YARN支持任何分布式框架。目前在YARN中已经可以运行MR、TEZ、Storm、Spark、HBASE、Kafka等多种计算框架，未来的趋势是各类系统或者服务都可以运行在YARN集群中。将框架运行在YARN中，具有简化应用程序的部署，资源的隔离和共享容易实现、提供监控和权限等优点。

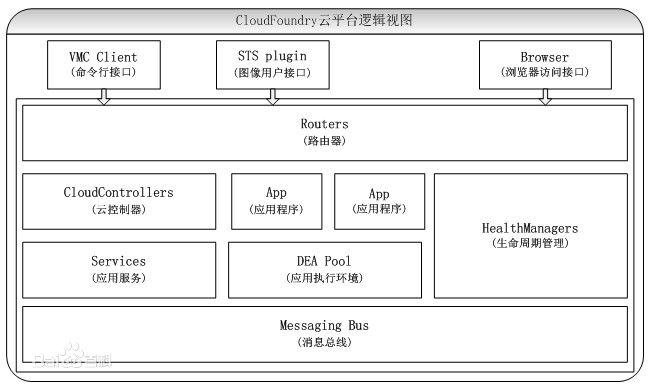
随着各种框架在YARN中的应用，YARN作为PASS的核心框架有可行性，而且YARN已经具有作为PASS具有的特性：

1）Hadoop在大规模集群中的应用推广，很多公司已经部署了Hadoop集群，可同时作为大规模数据处理和PASS。

2）YARN的三层框架，Client/RM/NM和PASS比较贴近，而且YARN的监控和资源管理能力，在PASS中比较有前途

# 2、PAAS ON YARN框架

Cloud Foundry是VMWare推出的业界第一个开源PASS云平台，支持多种框架、语言、运行时环境、云平台即应用服务，是开发人员能走在几秒进行应用程序的部署和扩展，其无需担心任务基础架构的问题。Cloud Foundry是由相对独立的多个模块构成的分布式系统，每个模块单独存在和运行，各模式之间通过消息机制进行通信，整体逻辑组成如下图所示：



开发任务通过VMC命令行工具或STS插件方便的部分应用程序到Cloud Foundry云平台上，最终用户可以通过浏览器访问在Cloud Foundry云平台上的应用。所有的访问请求都通过Router进行转发，分别由云控制器Cloud Controller和应用运行代理DEA模块进行请求响应。

以YARN作为PASS云平台的核心框架，其整体逻辑组成可如下所示：



开发人员可以通过客户端或BDOC申请服务。所有的应用请求都通过Router进行转发，分别由RM和NM进行请求响应，RM也负责监控和管理整个应用在平台上的正常运行。云平台上的各种服务由Service模块提供，该模块可采用Slider，Service可以灵活扩展。应用的实际执行由NM中的Container来执行。PAAS云控制器及生命周期管理，均有RM来完成，

各模块的协作如下图所示：



# 3、YARN Registry

随着YARN总体符合PAAS的需要，但是目前还缺少一些PAAS组件，或者说不完善。其中YARN Registry是其中一部分。各种服务运行在YARN中，运行的节点及使用的端口是不可控的，因此YARN作为PAAS需要解决客户端如何和YARN部署的服务交互，简化Service的注册和查找过程这两个问题。YARN中运行批处理作业或者查询，可以通过API进行管理，但是YARN中部署Tomcat Web Service或者HBASE这些长时间执行的服务，其运行的Host及端口动态分配，因此很难注册到DNS或者配置文件中。目前YARN的基本注册运行AM注册Web URL及IPC address，但是这些服务的endpoint都不能注册，例如Tomcat的访问端口。更难的是，YARN app启动后的Instance ID会变化，通过映射来完成endpoint的解析，很可能会映射到不存在的服务实例。

在YARN中，Registry相关的概念如下：

1. Service，YARN中部署的应用，如HBASE、HCATLOG、STORM，这些Service长时间或者短期执行
2. Service Class，Service Type的名称，在Zookeeper中注册的路径，该名称符合DNS的命名规则，例如org-apache-hbase及org-apache-hcatlog。DNS（域名系统）是域名及IP地址的分布式数据库，通过域名获取IP地址。
3. Component，Service的分布式组件，例如HBase HMASTER、REGIONS及REST Servers
4. Service Instance，多个相同Service的Cluster可以同时运行在YARN集群中，构建的Cluster都可以作为Service Instance
5. Endpoint，Service Instance或者Component Instance的绑定信息（IPC、JMX、REST API等），例如HBASE绑定的Zookeeper端口，RegionServer的JMX端口，HBASE Master的Web UI等。Endpoint可以是Internal，用于内部Component交互，External Endpoint用于提供给Client或者第三方软件使用
6. ServiceRecord，Service 或者Component instance在Registry中的注册记录，包括Endpoint列表

## 3.1 Registry Use Case

Hadoop中拥有权限即可以将服务注册到/services路径下，包括Core Hadoop Service，Hadoop集群可以访问的系统外Service，YARN中部署的服务属于每个用户。下面的服务都可以注册到YARN中：

1）Hadoop Core Service，例如HDFS，虽然没有运行在YARN中，但是可以通过服务本身或者管理工具将其注册，注册后第三方应用可以通过查询及使用

2）Long-lived 应用，在YARN中运行，并通过YARN Registry将信息注册到服务中。这些信息的生命周期是Service Instance的存在时间，AppMaster停止并不会删除注册信息。

3）Component Instance，将自身信息（Internal bind Info）发布到Registry中，例如JMX端口。

4）YARN-deployed应用可以动态或者静态绑定到独立的服务，例如Tomcat Web pool绑定到动态的HBASE Service实例(/users/joe/service/hbase/demo1)。

5）Component Instance，从App Master注册的Internal Endpoint中获取AM的信息，然后发送心跳从而进行交互。

但是Yarn Registry不支持Short-lived应用，应用的container在任务结束后马上注销，如果支持Short-lived应用，会给Zookeeper服务带来很大开销，所有的应用都可以设置成自动注册，但是不建议注册short-lived Containers。应用的Endpoint信息注册到Zookeeper后，有各种的使用方式，如下所示：

1）客户端应用，根据Service Instance的user，service class及instance name，例如/users/joe/services/hbase/demo1，查询连接到服务的信息

2）客户端应用，查询静态部署的Hadoop Service，例如/services/hdfs

3）App Master，可以从注册信息中获取所有的Component实例，包括JMX端口，根据这些信息初始化Web UI，并提供这些endpoints的链接

4）用户，访问private HBase Service，/users/joe/services/hbase/demo1

5）用户，访问部署在Hadoop中的HBase Service，例如/services/hbase

6）用户，访问remote Hadoop集群的文件系统，/net/cluster4/services/hdfs，该信息注册到本集群的zookeeper中，注册的信息包括remote文件系统的URL，例如webhdfs://

7）用户，列出HBase Service instance

8）用户，查找集群中所有的HBase Service

9）以后可能支持DNS服务，

注：根据上面列举的注册及查询的例子，可以看出registry的推荐用法是为App提供注册服务endpoint的方法，客户端及第三方软件可以根据这些注册信息定位应用并访问。

## 3.2 Service Registry的关键性质

Registry使用的几个关键点如下：

1）Service Instance动态注册到Zookeeper中，使用前提是YARN deployed Service能够注册binding信息，客户端都够从注册服务中获取这些信息；Core Hadoop 服务能够注册他们的Service Endpoint；binding信息随着服务的moves及HA切换动态更新。Service Instance能够发布服务的不同Endpoint，包括Web UI、REST及其他；此外，certificates及其他Sercurity信息可以作为binding的一部分注册到服务中。

2）Registry Service必须具有几个性质，Registry高可用性；由于服务及客户端很多，限制了每个服务可发布的信息；不管是物理、虚拟及在云中的YARN Cluster都可用，因此Registry Service要有普适性；必须支持分层次的命名空间及命名，命名必须符合DNS的规模，以后项目可能会使用DNS协议；Registry API语言及协议，Registry Service要跨语言；提供注册的REST API。

3）访问控制，所有的信息都可以可读；防止非法操作，要限制写；可以通过Apache Knox提供远程访问的访问控制。

4）Registry Sevice为了实现Components实例之间的binding信息的分享，不需要考虑Liveness监控、Leader-eletection及一致性等操作。Registry并不是为了存储应用的运行状态或者为了发布配置信息，这两种使用会使Registry负载过重，Registry更多的是为Service及Component提供Endpoing等binding信息。

## 3.3 Registry Architecture

Apache Hadoop提供的Registry服务，注册的信息形式为<name:String,records>，这些信息描述了Service及Component Instance。当前一个推荐的解决方案是使用ZK作为基本框架，将ZK NameSpace作为Service注册的根目录，例如使用/registry。Registry Service的一个基本实现必须提供Registry Service API及Yarn使用的naming Convenvions，虽然可以使用ZK，但是还是推荐使用Registry API来访问Registry信息。下面是实现的几个框架：

1）Service注册路径是分层的，以/为根目录，注册时为服务path绑定ServiceRecord

2）ServiceRecord，作为Znodes，持久化存储到ZK中。这保证了服务是否运行Record都可用，这就使Client访问变化的Service称为可能。

3）ServiceRecord有多种protocol的endpoint，每个Protocol必须包含: protocal Name，例如Web、REST、IPC及zookeeper；address，用户定位endpoint的详细信息；addressType，绑定string的格式（格式可以使URL，ZK path，hostname:port对），例如设置addressType=REST则绑定的type为URL，设置为IPC则绑定的type为host/port；api，endpoint提供的API，和应用相关，例如org.apache.hadoop.namenode、org.apache.hadoop

.webhdfs。

4）EndPoint，分成External(外部程序使用)及Internal(Service内部Component交互)，在ServiceRecord中存储在不同部分中。

5）注册信息的命名规则:Core Service使用下面的规则进行注册: /services/{servicename}，例如/services/hdfs；Yarn Service注册的规则为/users/{username}/{serviceclass}/{instancename}

;Component Instance注册在/users/{username}/{serviceclass}/{instancename}/components/

{componentname}。

注意：User's Service的Service Class名字必须唯一，Service Instance中Component Instance的名字唯一，建议以container ID作为Component的Instance。

## 3.4 Registry Model

### 3.4.1 Path Elements

使用RFC1123中定义的HostName Path格式（lower-case Entry），常用的表达式:

([a-z0-9]|([a-z0-9][a-z0-9\-]\*[a-z0-9]))

这个规则保证DNS服务使用Registry时，不会出现异常。在用户名中出现空格、high unicode及非常规字符，需要将hostName换成有效的DNS HostName Entries，增加DNS的处理难度，

### 3.4.2 ServiceRecord

ServiceRecord中的基本信息Internal及External Endpoint都是可选的，基本信息如下所示：

|  |  |
| --- | --- |
| Name | Description |
| type: String | Always: "JSONServiceRecord" |
| description: String | Human-readable description. |
| external: List[Endpoint] | A list of service endpoints for external callers. |
| internal: List[Endpoint] | A list of service endpoints for internal use within the service instance. |

type必须设置成JSONServiceRecord，如果缺少这个字段，不能通过JSON parser进行处理

### 3.4.3 YARN Persistence Policies

Registry信息在YARN中的存储规则，当application、attempt及container结束时，RM根据设置的Persistence Policies执行清除操作，例如设置Persistence Policies为permanent时，ServiceRecord会持久存在。当配置hadoop.registry.rm.enabled为true时，RM才会根据设置进行ServiceRecord的操作，disabled时不提供registry support。

Persistence共有两个字段：yarn:id及yarn:persistence。yarn:id定义了ServiceRecord的类别，yarn:persistence定义record cleanup的trigger，可以隐式定义yarn:id。之所以用yarn:作为前缀是为了提示这个操作是依赖于YARN Layer进行的操作。如果Registry是standalone的运行模式，则默认采用persistent，所有的record持久化。yarn:persistence定义的具体的规则如下：

|  |  |  |
| --- | --- | --- |
| Name | Description | contents of `yarn:id` field |
| permanent | The record persists until removed manually. | (unused) |
| application | Remove when the YARN application defined in the id field terminates. | application ID |
| application-attempt | Remove when the current YARN application attempt finishes. | application attempt ID |
| container | Remove when the YARN container in the ID field finishes | container ID |

yarn:id根据具体信息进行配置。在进行操作时，根据yarn:id匹配正确的application,attempt及container。如果yarn:id设置错误，那么就不会执行cleanup操作，如果设置成了不同的application或者container，那么根据application的生命周期进行操作。

### 3.4.4 EndPoint

Endpoint是ServiceRecord对外暴露的主要信息，第三方程序或内部Component根据Endpoint进行通信及操作。下面是EndPoint的几个字段：

|  |  |
| --- | --- |
| Name | Description |
| api: URI as String | API implemented at the end of the binding |
| protocol: String | Protocol. Examples: `http`, `https`, `hadoop-rpc`, `zookeeper`, `web`, `REST`, `SOAP`, ... |
| addressType: String | format of the binding |
| addresses: List[Map[String, String]] | a list of address maps |

所有的String fields都有大小的限制，而且最好不要将复杂的Json Structure放到text description中。下面介绍每个字段具体的含义：

addressType: Address Type，定义了entries的字符串格式，下面的表格是当前可以处理的格式：

|  |  |
| --- | --- |
| format | binding format |
| uri | uri:URI of endpoint |
| hostname | hostname: service host |
| inetaddress | hostname: service host, port: service port |
| path | path: generic unix filesystem path |
| zookeeper | hostname: service host, port: service port, path: ZK path |

在zookeeper binding中，每个entry都代表了quorum中的单个节点，hostname和port 定义了zk instance是的hostname及监听的端口。Path field列举了app使用的Zookeeper path。例如，HBASE可能会使用HBase Cluster的包含信息的znode。

api: API，API字段，必须包含一个URI，用于标识endpoint的API的详细信息，API必须唯一。下面是为了保证唯一性默认使用的规则：

1. WSDL定义的服务，使用SOAP/WS-\*的URL
2. svn/git的URL管理了REST API定义的文档
3. classpath schema后面要跟随app package的路径
4. uuid，为产生的UUID

但是推荐将一些Standard API URIS定义成common APIs，在文档中有两个非常规的APIs，分别是http://, 根据人的使用习惯; classpath:javax.management.jxm，支持JMX管理协议的Endpoint。

## 3.5 ServiceRecord Example

在Registry Service中，完整的Registry信息包括type、description、yarn:persistence及yarn:id，endpoint(external及internal)。

例子如下：



在Slider中，Python agent（运行在containers）中，通过registry获取用和AM交互的internal endpoints。container创建过程中，该record会被缓存在agent中。当交互失败后，会重新获取当前record，根据新endpoint进行重连。交互失败，可能会是low level socket/io或者https 认证操作。在Slider中，agents使用two-way HTTPS认证，如果AM故障，另外一个应用监听该AM端口，那么会触发认证失败，并重新读取record。