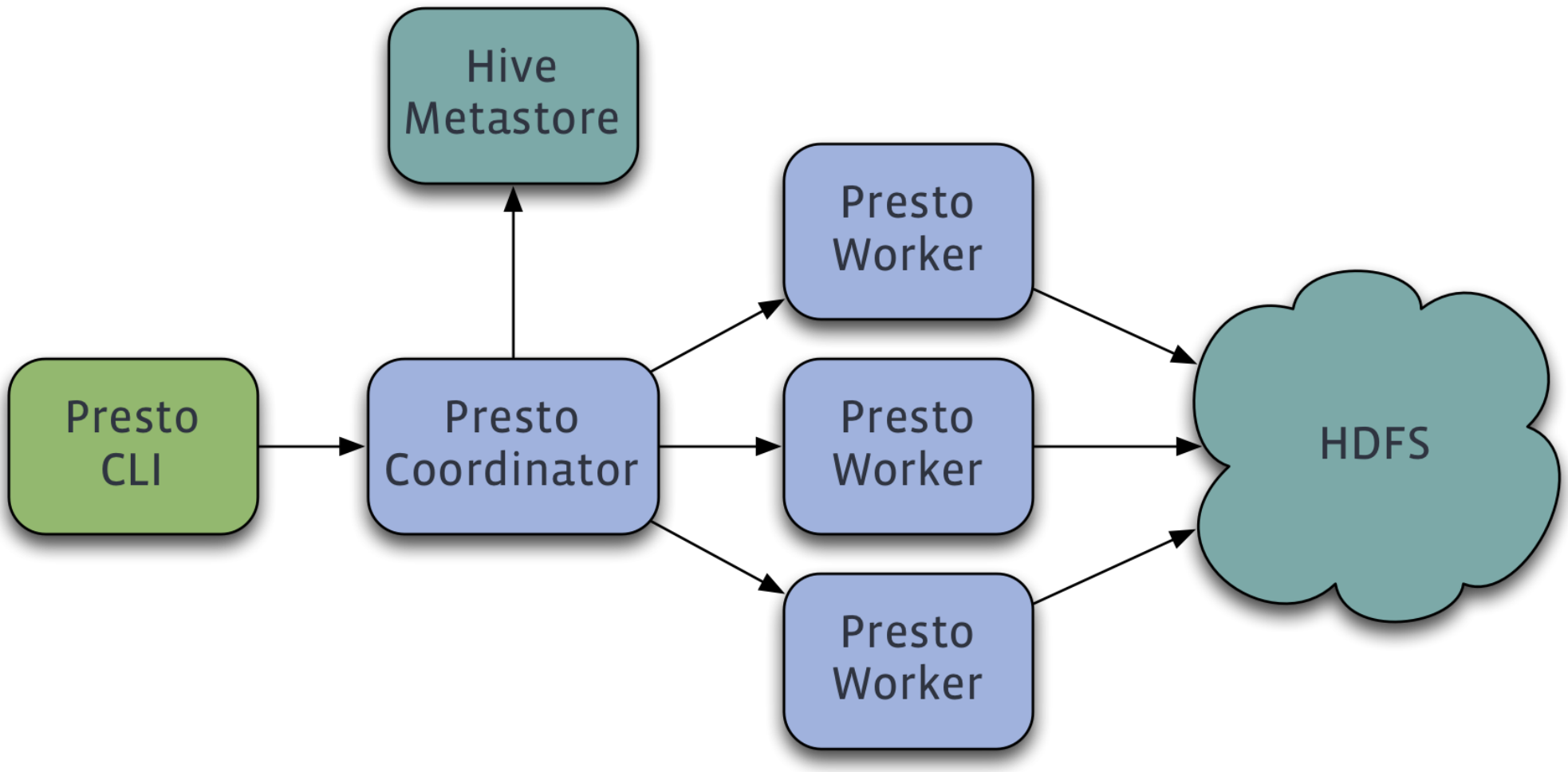
Presto独立服务发现(Discovery Service)

6月 1, 2017 ► **原创文章**

Presto的运行机制

Presto的运行机制如下：



不管是coordinator还是worker配置项中都有一项 `discovery.uri` ,这个是一个比较核心的东西，简单来说就是服务发现的地址。

coordinator和worker都会将自身注册到这个服务发现地址上，供彼此发现对方，coordinator可以通过个发现服务知道有多少worker节点，而worker节点可以通过这个发现服务知道coordinator是谁，这样做的好处是coordinator和worker做到了完全的解耦，彼此都不需要在启动时配置对方，而是通过第三方服务来发现对方。

Presto On Yarn的问题

在默认的情况下这个发现服务是内嵌在coordinator中的，也就是coordinator在启动的时候会启动一个内嵌的发现服务，在这种情况下，coordinator将自身注册给自身的发现服务，而worker则将发现服务的地址配置成coordinator的发现服务地址，此时coordinator同时充当presto协调者和服务发现的提供者。

以上这种情况在一般的情况下可以良好的运行，但是当我们将presto服务迁移到Presto On Yarn时就会遇到一些问题：

- presto on yarn是一种动态的运行策略，在yarn上面，哪个节点运行presto的coordinator和worker是不确定的，这会给外部调用presto的程序带来困扰

  外部的程序和presto的交流一般是通过presto提供的客户端来调用，而它的客户端需要事先知道presto的coordinator地址，在presto on yarn的情况下，coordinator的地址是不确定的，有可能会发生变化。

  这种情况下的处理方案是：**将presto的服务发现方案外置**，将presto的服务发现服务独立于presto的coordinator运行，将presto的coordinator和worker中的 `discovery.uri` 配置成外部独立的发现服务地址，在外部提供具有HA的服务发现，提供稳定的发现服务。

  Presto的服务发现是基于airlift的服务发现做的实现，airlift的服务发现可以在这里查看实现和源码，不过它基本是处于无文档的状态，所以理解要多花些功夫。

  airlift的服务发现的总体思路是基于http提供一个提供服务发现的HA集群，集群之间通过http通信，通过数据同步方式，提供最终一致性的保证。

这里我们就来说说airlift的服务发现服务的HA安装。

Airlift Discovery安装

安装步骤

- 下载源码
  git clone https://github.com/airlift/discovery.git
- 编译源码
  mvn clean package -DskipTests=true
- 环境安装
  将target目录下的discovery-server--SNAPSHOT.tar.gz安装包copy至安装机器上进行解压安装

环境配置

- 解压后在解压目录新建etc目录，并在etc目录下新建以下配置文件
  - config.properties
  - jvm.config
  - log.properties
  - node.properties
  - service-inventory.json
- 配置文件
  config.properties文件为主配置文件，主要配置该discovery服务的主要配置信息，如运行环境，服务端口，节点id等信息，配置信息一般情况如下：

```
node.environment=test
http-server.http.port=8411
node.id=597A741E-9968-40E2-BB4D-7AF26DE18689
service-inventory.uri=file://<installation-location-of-your-discovery-service>/etc/service-inventory.json
```

node.environment指定运行环境
http-server.http.port指定服务运行的端口
node.id指定该节点的id
service-inventory.uri指定了该集群拥有的所有节点信息

jvm.config文件主要配置服务jvm的配置信息，该配置和presto的配置文件的jvm配置类似，一般情况按如下信息自行进行调整：

```
-server
-Xmx2G
-XX:+UseG1GC
-XX:G1HeapRegionSize=32M
-XX:+UseGCOverheadLimit
-XX:+ExplicitGCInvokesConcurrent
-XX:+HeapDumpOnOutOfMemoryError
-XX:OnOutOfMemoryError=kill -9 %p
```

log.properties主要记录的日志级别调整，这里不再叙述
node.properties主要记录的是节点相关的配置，类似于config.properties配置，但是不同点在于config.properties强调集群共有的特性，而node.properites强调节点间相同配置项的不同配置值区别
service-inventory.json这是一个比较重要的文件，里面记录了整个集群的信息，discovery集群利用这个配置文件获取集群的所有信息，知道集群中所有部署的情况及如何与其它节点进行通信。它的配置如下：

```
{
"environment": "test",
"services": [
    {
        "id": "C8A9EE64-0476-452C-8638-8E72F3EE3CA6",
        "nodeId": "597A741E-9968-40E2-BB4D-7AF26DE18689",
        "type": "discovery",
        "pool": "general",
        "location": "/172.17.31.245",
```

```
            "state": "RUNNING",
            "properties": {
                "http": "http://172.17.31.245:8411"
            }
        },
        {
            "id": "370AF416-5F44-47D3-BFB6-D93A92676D49",
            "nodeId": "0BA42FDB-5DBA-4A2C-BE26-9596B7B4368E",
            "type": "discovery",
            "pool": "general",
            "location": "/172.17.31.246",
            "state": "RUNNING",
            "properties": {
                "http": "http://172.17.31.246:8411"
            }
        }
    ]
}

```

以上面的配置中，集群中有两个节点，并指出了两个节点的节点id信息，以及他们的通信地址 `properties.http` 等信息，有了这份信息，集群中的各节点就知道如何同其它节点进行数据交互与同步了。

- 运行集群

  在集群每个节点的安装目录下bin目录中运行: ./launcher start进行服务的启动， ./launcher stop 进行服务的停止 ./launcher restart 进行服务的重启

- 验证服务

  当服务运行成功后，可以通过浏览器进行访问，若配置的端口为8411，则访问发现服务的地址为：http://localhost:8411/v1/service
  这个地址将返回所有注册到这个发现服务的服务的列表

- 高可用

  因多台机器共同组成了发现服务，发现服务有最终一致性保障，所以只需要访问其中一台就可以，但是为了高可用，可以在发现服务前端加入NGINX作流量分担与负载解决高可用的问题

Presto节点信息注册到发现服务

将Presto的节点信息注册到发现服务非常简单，上面也说过了，Presto节点之前是通过自身位集群的coordinator节点充当服务发现者提供服务的，现在只需要将 `discovery.uri` 的配置换成外置的airlift服务发现服务地址就可以了用了。在这个示例中我将配置值修改成了'172.17.31.245:8411',因为是测试环境，不需要过于要求的HA场景，所以我只配置了服务发现集群中的一个节点。

Presto的客户端集成

Presto独立服务发现(Discovery Service)

因Presto的客户端调用需要知道coordinator，而现在Presot On Yarn上了过后，coordinator的地址是不定的，且是注册到服务发现上的，对于Presto客户端想知道明确的coordinator地址需要做一些改变：将调用presto客户端前要先得到coordinator，而要得到coordinator可以通过服务发现获取，看了下airlift这个框架，它自身提供了服务发现的客户端的功能，但是看了有点晕眩，大致思路是实现一个http接口去定期轮询服务发现地址，得到服务地址(coordinator)就可以了，于是我自己实现了一个简易版本的，通过一个服务发现的网关地址，应用启动后通过后台线程每隔10s去轮询一次该服务发现网关，得到更新的coordirnator地址，更新本报的缓存，所有获取coordinator地址都从本地缓存中获取，避免每次的服务发现网关轮询。

目前运行情况

目前运行情况良好，充分解决了Presto On Yarn后的coordinator随时可变的情况，应用能够根据coordinator的变化随时适应变化（10s延时）及时调整，避免因coordinator的变化导致的查询应用不可用问题。

参考文档：

- https://prestodb.io
- https://prestodb.io/presto-yarn
- https://github.com/airlift/airlift
- https://github.com/airlift/discovery