Slider Docker源码解析

# 1.简介

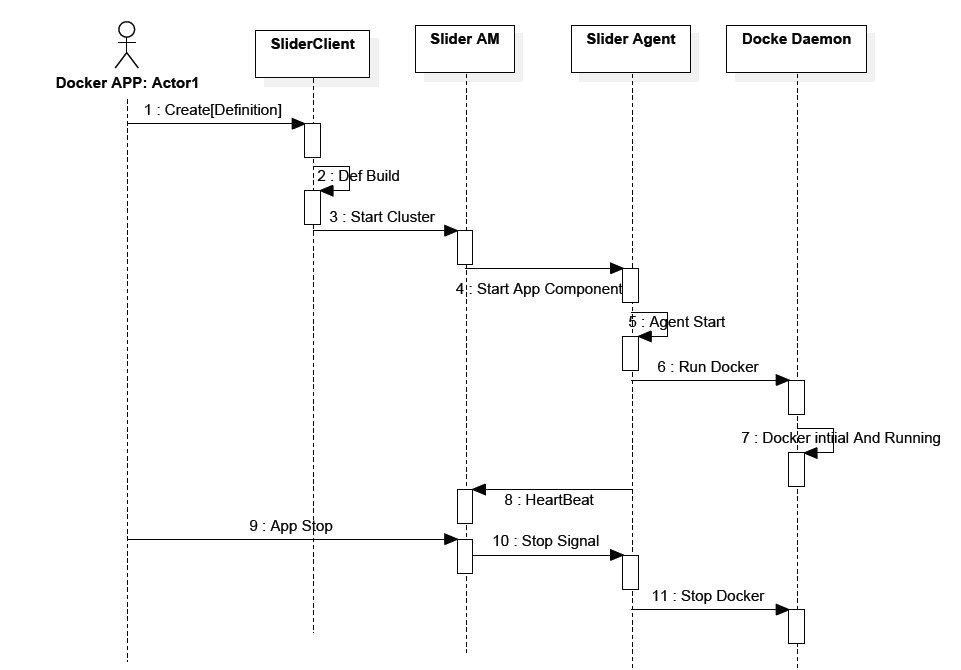
当前使用Apache Slider可以将HBase、Storm这些常驻服务运行在YARN上，但是这些常驻服务的启动与维护都有一定的难度；Docker是一个开源的容器引擎，可以将应用及其依赖打包到一个可移植的容器中，然后发布到Linux之上。

在YARN中启动Docker有两种形式，使用DockerContainerExecutor将YARN Containers发布到Docker Container中，用户需要根据他们的需求定义Docker images，将用户程序所依赖的软件环境打包到镜像中，但是在。但是使用DockerContainerExecutor时，需要NM的启动Container时以root来执行，另外一种方式是通过YARN-3611中的新特性，Support Docker Containers in LinuxContainerExecutor，使用LinuxContainerExecutor来启动Docker，这两种Docker的执行方式不在这里介绍。

不管是通过哪种形式来执行Docker集群，都需要用户根据需求提前生成image，然后实现应用程序的运行逻辑，这个过程工作量大而且过程比较复杂。Apache Slider在JIRA SLIDER-780和SLIDER-906中实现了两个docker集群的执行方式，前一种的执行方式是通过Slider Agent中直接调用docker命令来启动Docker Container，第二种是基于YARN的DockerLinuxContainerRuntime(即Support Docker Container In LinuxContainerExecutor)来实现Docker的启动和管理，其执行的总体框架基本相同（细节在以后会进行分析），如下图所示：



大致流程为通过SliderClient端启动程序的执行，在YARN中启动SliderAppMaster，Slider AM与YARN进行通信获取所需要的资源，在Container在启动Slider Agent后，Slider AM向Agent发送执行命令，启动与管理Docker，时序如下所示：



# 2. 运行实例

通过Apache Slider来运行和管理Docker集群，简化了开发者的工作，开发者仅需要在应用的定义文件中描述其执行逻辑即可，下面是一个例子，通过该例子介绍其定义及执行过程，Docker集群的执行命令和其他应用程序的执行略有不同，如下：

$slider create [app-name] --template appConfig.json --metainfo metainfo.json --resources resources.json

在命令中要提供三个定义文件：

* metainfo.json，docker集群的描述文件，定义了集群包含的docker component类型及默认使用的一些配置
* appConfig.json，docker在运行过程中配置的文件，可以包括mount,inputfile及端口等信息，这些信息会覆盖metainfo.json中的属性
* resource.json，docker集群启动的docker数目及资源等

实例example，启动一个memached server, 如下：

metainfo.json

{

"schemaVersion":"1.0",

"application":{

"name":"MEMCACHED",

"components":[

{

"name":"MEMCACHED",

"type":"docker",

"dockerContainers":[

{

"name":"memcached",

"commandPath:":"/usr/bin/docker",

"image":"memcached"

}

]

}

]

}

}

appConfig.json

{

"schema":"http://example.org/specification/v2.0.0",

"metadata":{},

"global":{},

"components":{

"MEMCACHED":{

"memcached.commandPath":"/usr/bin/docker"

}

}

}

resource.json

{

"schema":"http://example.org/specification/v2.0.0",

"metadata":{},

"global":{},

"components":{

"slider-appmaster":{},

"MEMCACHED":{

"yarn.role.priority":"1",

"yarn.component.instances":"1",

"yarn.memory":"512"

}

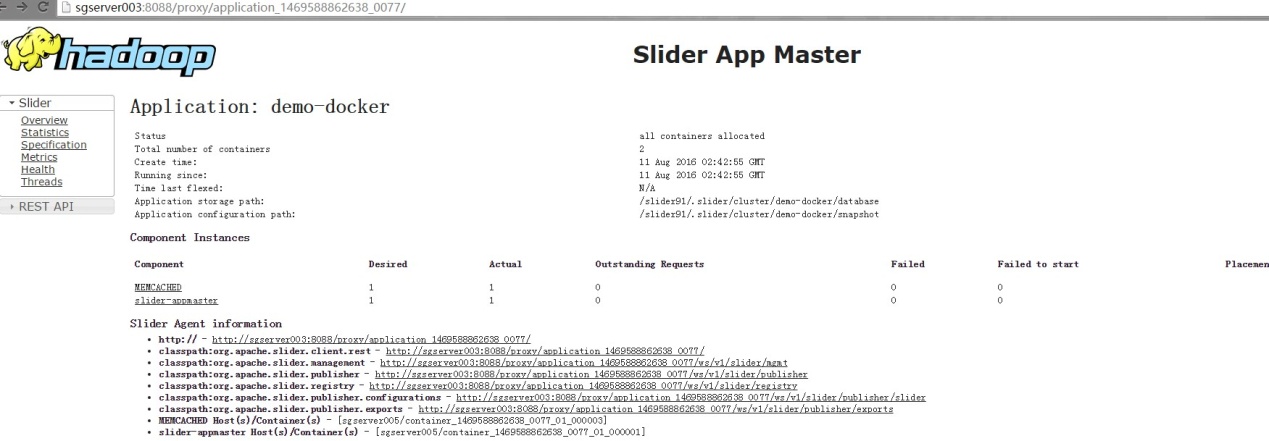
}

}

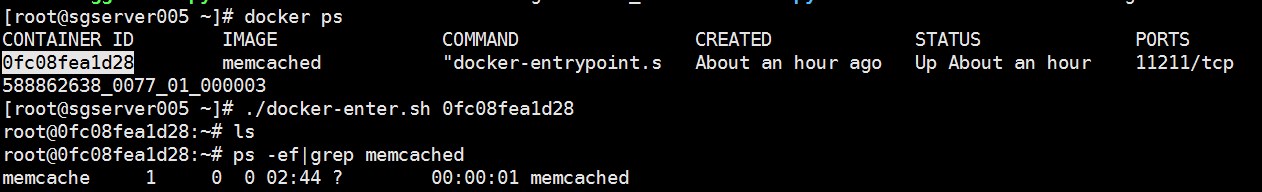
创建命令：

slider create demo-docker --template appConfig.json --resources resources.json --metainfo metainfo.jso

启动后，通过YARN AM Web查看的结果如下：



找到具体的执行host，查看docker container如下：



# 3. 定义描述

在Metainfo.json中定义了image如何运行，其字段和非Docker application被非全部不一样，仅其在dockerContainers描述中不同，包括以下部分：

1）name，container的名称，不影响发布的应用，但是在appConfig.json中标识覆盖的属性

2）image，docker image的全称

3）additionalParam，在启动Docker container中时将参数传递到命令中

4）commandPath，docker命令的路径

5） statusCommand，用于检查运行应用的健康状况，返回0时为healthy，非0为unhealthy。如果不定义，则slider执行docker top ${container\_ID}|grep来获取运行状况

6）port,containerPort和hostPort，container的port和hostPort字段相绑定，当docker run命令时通过-p hostPort:containerPort来转换

7）mount,containerMount和hostMount，在启动container时通过-v hostMount:container

Mount将主机的目录挂载到Container中

8）options，定义多个附加的docker run命令。在启动应用时，传到docker un中，如果没有定义，会使用-d

# 4. Slider Docker源码

**Input Data**

metainfo.json,appConfig.json和resources.json

其中metainfo.json和appConfig.json决定了docker container的所有信息，resoucres.json的作用是仅决定了YARN Containers的启动数目和设置内存及cpu。具体的定义方式在其他文档中介绍

**流程描述**

Agent main.py程序启动之前的流程是相同的，Agent和SliderAppMaster连通后进行心跳交互。

1）判断是否是docker

AgentProviderService

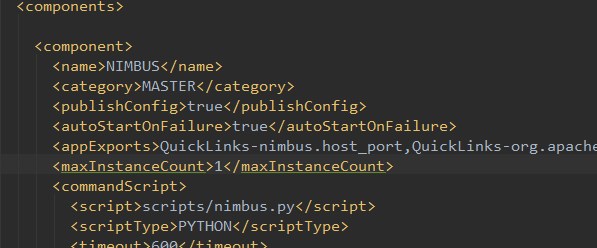
#isDockerContainer //判断component为docker类型

#isYarnDockerContainer //判断 component为yarn\_docker类型

从metainfo.xml中获取组件的type，如下所示：



对于非Docker的组件type为STANDARD：



type默认为TYPE\_STANDARD [STANDARD]，可设置类型：STANDARD|DOCKER|

PYTHON及YARN\_DOCKER

2）AgentProviderServier向Agent发送INSTALL命令

Slider AM根据Agent发过来的心跳信息，生成Install CommandAgentProviderService

*#addInstallDockerCommand*

cmd初始化信息如下：

#taskId，和command ID <= taskID+"-1"

#clusterName

#roleCommand <= INSTALL

#serviceName

#componentName

#roleName

#hostLevelParams

package\_list <= osSpecifics，[tarball,xxx.tar.gz]

containerId

#configurations

在appConfig.json中的global配置

#componentCommand

name <= INSTALL

exec <= DEFAULT

cmdParams: service\_package\_folder,cmd,schema\_version,command\_timeout,

script\_type,record\_config

#hostName

#addContainerDetails //docker执行相关信息

image <= metaContainer.getImage

network

networkScripts

name

options

additionalParams

commandPath

statusCommand

startCommand

mounts

ports

inputFile

configFiles

# dockerConfig 放入configurations中

yarnDocker，//do Nothing

docker，command\_path <= docker bin //来自于appConfig.json中的配置

docker.image\_name <= image

注意：cmd以HeartbeatResponse的形式放到给Slider Agent，

response.setRestartEanbled <= autoStartOnFailure是否为true

3）Agent端接收Response（先Register然后再Heatbeat）

Controller.py#heatbeatWithServer

response <= sendRequest(heartbeatUrl,data)

addToQueue <= executionCommands { response['executionCommands']}

updateStateBasedOnCommand ||在不同的状态会有不同的statusCommand

*将executionCommands传入Queue中等待调度*

ActionQueue#process\_command

#execute\_command [ExecutionCommands]

//根据命令的字段，决定使用哪个脚本执行命令

--------------- yarnDockerMode ---------------------

YarnDockerManager.py#exectue\_command

#获取config

#执行程序

INSTALL => install\_container

#configFiles 生成配置文件，=> workroot, InlineTemplate

START => start\_container

#outFile,errFile

#execute\_command\_on\_linux

startCommand <= extract\_config\_from\_command

与DockerManager的不同在于执行命令的组合方式不同，而且其中有获取container的执行Ip和Hostname命令

------------------- dockerMode -------------------------

DockerManager.py#execute\_command

1）获取config

2）获取命令名

INSTALL => pull\_image docker pull {imageName}

START => start\_container

#port

#mount

#inputFile

#memory

#image

#param

------------------------ regular ------------------

CustomServiceOrchestrator.runCommand

#获取脚本，并执行

4）Agent返回心跳（命令执行结果）

#commandResult <= 命令执行结束

#commandStatuses <= command,roleResult,reportResult ActionQueue

#Controller data<= Heatbeat.build() ActionQueue#commandStatus

#Controller.sendRequest[url,data]