YARN Service资源目录

YARN Service使用以下提供的JSON文件运行命令：

*$yarn app -launch my-sleeper sleeper.json*

*{*

*"name": "sleeper-service",*

*"version": "1.0.0",*

*"components" :*

*[{*

*"name": "sleeper",*

*"number\_of\_containers": 2,*

*"launch\_command": "sleep 900000",*

*"resource": {*

*"cpus": 1,*

*"memory": "256" } }]*

*}*

命令提交后，会在hdfs目录中生成对应的Service目录，如下所示：

*/user/hdfs/.yarn/services/my-sleeper //该Service的启动资源目录*

*$ tree -L 3 .*

*.*

├── *components*

│   └── *sleeper*

│   ├── *sleeper-0 //组件具体描述*

│   └── *sleeper-1*

├── *conf*

│   └── *yarnservice-log4j.properties //配置文件*

├── *lib //依赖jar包*

└── *my-sleeper2.json //Service定义文件*

# AM启动信息-ServiceMaster

ServiceMaster的启动命令行如下：

*/usr/jdk64/jdk1.8.0\_112/bin/java*

*-Djava.net.preferIPv4Stack=true*

*-Djava.awt.headless=true*

*-Dlog4j.configuration=yarnservice-log4j.properties*

*-DLOG\_DIR=.../container\_e09\_1535536235958\_0001\_02\_000001*

*org.apache.hadoop.yarn.service.ServiceMaster*

*-yarnfile hdfs://fys1.cmss.com:8020/user/hdfs/.yarn/services/my-sleeper/my-sleeper.json*

*-D hadoop.registry.zk.root=/registry*

*-D hadoop.registry.zk.quorum=fys1.cmss.com:2181*

其启动依赖的资源包括：

* service描述文件
* conf，配置文件
* lib，依赖jar包
* components，组件依赖

# **Component启动**

核心ContainerLaunchService#launchComponent

*Component compSpec = instance.getCompSpec();*

*ProviderService provider = ProviderFactory.getProviderService( compSpec.getArtifact());*

*AbstractLauncher launcher = new AbstractLauncher(context);*

*try {*

*provider.buildContainerLaunchContext(launcher, service,instance, fs, getConfig(), container);*

*instance.getComponent().getScheduler().getNmClient().startContainerAsync(container,*

*launcher.completeContainerLaunch());*

*}*

其中根据Artifact类型生成对应类型的ProviderService



目前支持三种，分别对应：

*DOCKER("DOCKER"),*

*TARBALL("TARBALL"),*

*SERVICE("SERVICE")*

在Artifact中配置，示例如下：

*"artifact": {*

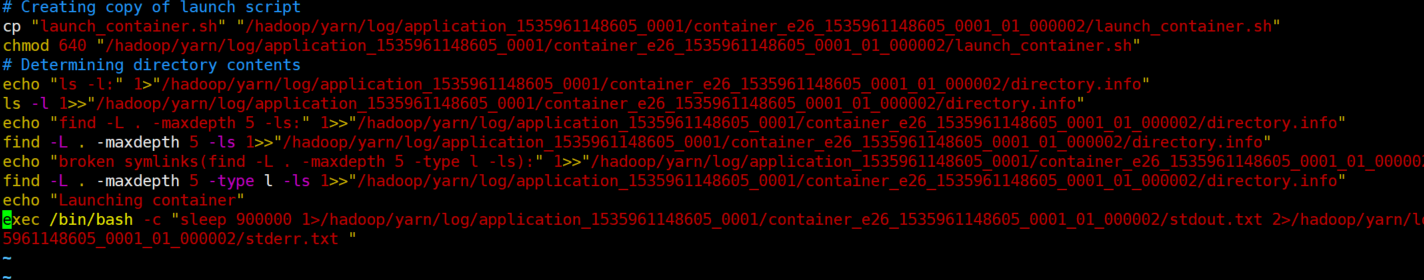
*"id": "centos/httpd-24-centos7:latest",*

*"type": "DOCKER"*

*}*

*switch (artifact.getType()) {  
 // TODO add handling for custom types?   
 // TODO handle service //还不支持Service类型  
 case DOCKER:  
 return DockerProviderFactory.getInstance();  
 case TARBALL:  
 return TarballProviderFactory.getInstance();  
 default:  
 throw new IllegalArgumentException(String.format("Resolution error, " +  
 "%s should not be passed to createServiceProviderFactory",  
 artifact.getType()));  
 }  
}*

1. 默认不配置，则没有artifact，直接启动脚本，如示例，直接执行command



直接指向launch\_container.sh。而且使用DefaultLinuxContainerRuntime

1. DOCKER("DOCKER")

*case DOCKER:  
 return DockerProviderFactory.getInstance();*

根据Docker的类型，设置环境变量

*public void processArtifact(AbstractLauncher launcher,ComponentInstance compInstance,*

*SliderFileSystem fileSystem, Service service) throws IOException{  
 launcher.setYarnDockerMode(true);  
 launcher.setDockerImage(compInstance.getCompSpec().getArtifact().getId());  
 .....*

*launcher.setRunPrivilegedContainer(  
 compInstance.getCompSpec().getRunPrivilegedContainer());  
}*

环境变量在ContainerLaunchContext中对应的参数列表如下：

|  |  |  |
| --- | --- | --- |
| 参数名 | 描述 | Env-launchContext |
| yarnDockerMode | 是否使用Docker | YARN\_CONTAINER\_RUNTIME\_TYPE |
| dockerImage | Artifact的docker镜像 | YARN\_CONTAINER\_RUNTIME\_DOCKER\_IMAGE |
| dockerNetwork | Docker网络  <=docker.network | ARN\_CONTAINER\_RUNTIME\_DOCKER\_CONTAINER\_NETWORK |
| dockerHostname | {compInstance}.{service}.{user} | YARN\_CONTAINER\_RUNTIME\_DOCKER\_CONTAINER\_HOSTNAME |
| runPrivilegedContainer | run\_privileged\_container | YARN\_CONTAINER\_RUNTIME\_DOCKER\_RUN\_PRIVILEGED\_CONTAINER |
|  | configResource | YARN\_CONTAINER\_RUNTIME\_DOCKER\_LOCAL\_RESOURCE\_MOUNTS |

使用DockerLinuxContainerRuntime来启动Container，生成的DockerRunCommand如下：

*DockerRunCommand runCommand = new DockerRunCommand(containerIdStr,*

*dockerRunAsUser, imageName)*

*.detachOnRun()*

*.setContainerWorkDir(containerWorkDir.toString())*

*.setNetworkType(network);*

*.....*

*Path launchDst =new Path(containerWorkDir, ContainerLaunch.CONTAINER\_SCRIPT);  
overrideCommands.add("bash");  
overrideCommands.add(launchDst.toUri().getPath());  
runCommand.setOverrideCommandWithArgs(overrideCommands);*

*//launch-command <= launch\_container.sh*

*//将命令写到commandFile中*

*String commandFile = dockerClient.writeCommandToTempFile(runCommand, containerIdStr);*

*PrivilegedOperation launchOp = buildLaunchOp(ctx,commandFile, runCommand);*

*PrivilegedOperation launchOp = buildLaunchOp(ctx,commandFile, runCommand);*

*//最终的执行*

*privilegedOperationExecutor.executePrivilegedOperation(null,launchOp, null, null, false, false);*

Container的调用OperationExecutor的初始化如下：

*PrivilegedOperation launchOp = new PrivilegedOperation(  
 PrivilegedOperation.OperationType.LAUNCH\_DOCKER\_CONTAINER);  
launchOp.appendArgs(runAsUser, ctx.getExecutionAttribute(USER),  
 Integer.toString(PrivilegedOperation  
 .RunAsUserCommand.LAUNCH\_DOCKER\_CONTAINER.getValue()),  
 ctx.getExecutionAttribute(APPID),  
 containerIdStr,  
 containerWorkDir.toString(),  
 nmPrivateContainerScriptPath.toUri().getPath(),  
 ctx.getExecutionAttribute(NM\_PRIVATE\_TOKENS\_PATH).toUri().getPath(),  
 ctx.getExecutionAttribute(PID\_FILE\_PATH).toString(),  
 StringUtils.join(PrivilegedOperation.LINUX\_FILE\_PATH\_SEPARATOR,  
 localDirs),  
 StringUtils.join(PrivilegedOperation.LINUX\_FILE\_PATH\_SEPARATOR,  
 logDirs),  
 commandFile,  
 resourcesOpts);*

*Docker Container的执行格式如下：后续补充*

1. TARBALL("TARBALL")

使用TarballProviderService，Artifact的配置格式如下：

将Artifact中配置的tarball路径配置在LocalResource中，如下：

*Path artifact = new Path(instance.getCompSpec().getArtifact().getId());  
LocalResourceType type = LocalResourceType.ARCHIVE;  
LocalResource packageResource = fileSystem.createAmResource(artifact, type);  
launcher.addLocalResource(APP\_LIB\_DIR, packageResource);*

启动过程与默认的Default相同，仅多了Tarball的处理过程。

|  |  |  |
| --- | --- | --- |
| 参数 | 描述 | 使用 |
| yarn.service.framework.path |  | ServiceClient.addJarResource |
| yarn.service.am.java.opts | AM使用的物理内存，-Xms |  |
| --enableFastLaunch |  |  |
|  |  |  |