阿里云

专有云 Enterprise 版

方舟

运维指南

产品版本：V3.3.0

文档版本：20180331

为了无法计算的价值 | 阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

目录

法律声明 I

目录 II

1 Butler 操作指南 1

1.1 简介 1

1.2 登录控制台 1

1.3 账号管理 2

1.3.1 创建账号 2

1.3.2 设置登录密码 3

1.3.3 获取安全凭证 4

1.3.4 删除账号 5

1.4 开启多因素认证 5

1.5 私有证书管理 8

1.5.1 安装证书 8

1.5.2 管理证书 9

1.5.3 查看审计记录 10

1.5.4 管理节点 11

1.5.5 管理抓取任务 11

1.6 系统概览 11

1.7 容器列表 12

1.8 告警列表 14

1.9 监控管理 14

1.9.1 监控接入 15
1.9.1.1 巡检接入.....15
1.9.1.2 指标接入.....17
1.10 系统管理 20
1.10.1 调度管理.....21
1.10.2 巡检管理.....21
1.11 指标管理 24
1.11.1 联系人管理 25
1.11.1.1 联系人管理 26
1.11.1.2 联系组管理 26
1.11.2 通知管理.....26
1.11.2.1 邮件设置.....26
1.11.2.2 短信设置.....27
2 巡检 28
2.1 DRDS 巡检 28
2.1.1 DRDS 监控与问题发现 28
2.1.1.1 登录 DRDS-manager 管理控制台.....28
2.1.1.2 监控实例.....28
2.1.1.3 监控节点.....29
2.1.1.4 监控数据库.....30
3 运维操作.....32
3.1 产品启动操作 32
3.1.1 启动 EDAS&DRDS 依赖的应用 32

3.1.2 启动 EDAS 37
3.1.3 启动 DRDS 39
3.1.4 启动 MQ 41
3.1.5 启动 GTS 43
3.1.6 应用停止 43
3.1.6.1 停止 vipclient 43
3.1.6.2 停止 EDAS 44
3.1.6.3 停止 DRDS 44
3.2 方舟命令行运维操作 44
3.2.1 命令行入口介绍 44
3.2.1.1 Swarm mode CLI 45
3.2.1.2 Docker Engine CLI 45
3.2.2 查找容器-ID or Name 45
3.2.3 管理容器 46
3.2.3.1 登录容器 46
3.2.3.2 不进入容器执行命令 46
3.2.3.3 删除容器 46
3.2.3.4 启动/停止/重启容器 47
3.2.4 查看容器日志 47
3.2.4.1 查看容器本身日志 47
3.2.4.2 查看容器内的应用程序日志 47
3.2.4.3 查看容器启动删除等日志 47
3.2.5 分析容器元信息 47
3.2.6 查看方舟集群整体情况 48

3.2.6.1 确定容器所在物理机.....48
3.2.6.2 查看集群节点情况.....48
3.2.7 网络相关运维.....48
3.2.7.1 判断网络状况.....48
3.2.7.2 网络初始化.....49
3.2.8 组件容器的健康检查.....49
4 故障处理.....51
4.1 方舟常见问题.....51
4.1.1 容器无法删除.....51
4.1.2 清理未使用镜像.....51
4.1.3 清理 ovs 日志.....51
4.1.4 容器被删除.....52
4.1.5 启动容器 ID 冲突.....53
4.1.6 创建本地镜像仓库.....53
4.1.7 容器重启出现 endpoint has attached in network.....54
4.2 产品常见问题.....54
4.2.1 EDAS.....54
4.2.1.1 EDAS 节点配置.....54
5 集群节点扩容指南.....55
5.1 添加节点到方舟底座集群.....55
5.2 为服务节点添加 Label 标签.....55

1 Butler 操作指南

1.1 简介

Butler 是中间件平台的轻量级运维平台，具备统一运维、组件与容器监控和告警通道对接功能。

Butler 可通过“巡检”和“指标”两种方式进行业务和系统层级的检测，在触发异常时，可通过短信、邮件方式进行告警。

- **巡检**：周期性地拨测 URL 或端口等来判断业务运转是否正常。目前支持的巡检方式有：HTTP、TCP、PING 和 JDBC。
- **指标**：通过 Tlog 从客户机搜集日志并按一定方式汇聚出关键指标，以此来衡量系统运行状况。Butler 可以监控 CPU、负载、存储等系统指标，也可以监控客户自定义的指标（前提是客户机能以一定格式输出日志，并在 Tlog 上配置响应的采集规则）。
- **告警**：在 Butler 上可以以产品维度配置告警联系人和告警方式。告警方式因网络不同而有所差异：
  - 能接入公网时，无需做任何额外配置，即可通过云鸽和中间件邮件服务器发送短信或邮件。
  - 不能接入公网时，可在 Butler 配置客户的短信网关接口或 SMTP 服务器。

Butler 可以通过 Region 来区分和管理不同的工作环境。如果需要 Butler 管理多套环境，只要保证采用不同的 Region 即可。

1.2 登录控制台

登录控制台是使用 Bulter 进行监控的前提。

1. 在浏览器（推荐使用 Chrome）的地址栏中输入 **Butler 控制台的 URL**，并单击**回车**。

   初次登录会跳转到 Dauth 控制台进行登录。

2. 在 Dauth 登录中心页面输入**用户名**、**密码**，然后单击**登录**。

   动态令牌是用户**开启多因素认证**（MFA）后的必填项，在未开启 MFA 时也忽略。如何开启 MFA，请参考 1.4 Butler 操作指南。

登录成功后，会返回 Butler 控制台首页。

1.3 账号管理

中间件各产品登录控制台所用的账号为 Dauth 账号，账号管理需要使用管理员账号登录 Dauth 控制台进行操作。

1.3.1 创建账号

1 使用管理员账号登录 Dauth 控制台。

管理员的默认用户名和密码为：用户名：admin；密码：admin。

注: 动态令牌在开启多因素认证（MFA）后为必填项，开启 MFA 请参考。

2 在菜单栏中选择**系统管理 > 账号管理**。

3 在账号管理页面单击**创建账号**。

4 在创建账号页面，设置**账号**、**登录名**，并设置是否**生成安全凭证**，然后单击**提交**。

注: 如果生成安全凭证选择了是，则会为该账号生成 APP_KEY 和 APP_SECRET。

1.3.2 设置登录密码

1 在账号管理页面单击登录信息。

2 在登录信息页面单击**更新**。

3 在更新登录信息页面设置登录密码及其它信息，设置完成后，单击**提交**。

登录信息说明：

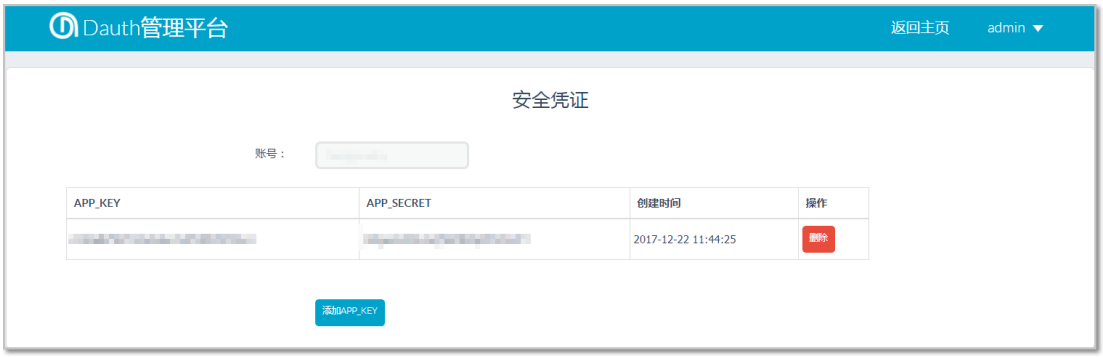
- **登录名**：在创建账号时已设置，此处可以修改。
- **密码**：登录名对应的登录密码。
- **开启 MFA**：默认为否，及不开启多因素认证（MFA），且不可设置。如果需要开启，请参考 1.4 开启多因素认证。

其它信息，如真实姓名、昵称、身份证号、手机、电子邮件、用户类型及附件信息，为可选信息。

1.3.3 获取安全凭证

1 在账号管理页面单击**安全凭证**。

2 在安全凭证页面单击**添加 APP_KEY**。



注: 如果在创建账号时，选择了生成安全安全凭证，账号创建完成后即会有系统分配的默认的 APP_KEY 和 APP_SECRET。原有 APP_KEY 有安全隐患，可以在添加新的 APP_KEY 之后，在此页面将原来的 APP_KEY 删除。

3 在添加 APP_KEY 页面单击**提交**。

注: APP_KEY 和 APP_SECRET 不建议自己设置，以保证复杂程度符合安全要求。

1.3.4 删除账号

在账号管理页面的操作列单击删除，在弹出的确认对话框中确认要删除的账号，然后单击提交。

1.4 开启多因素认证

开启多因素认证需要给您的手机发送短信验证码，所以需要先在手机上安装谷歌身份认证器。

谷歌身份认证器是一种支持 TOTP(RFC 6238)协议的动态口令生成工具，已经被广泛应用于多因素身份认证场景中。

根据您的手机的操作系统扫描相应的二维码进行下载：

- iOS 系统

- Android 系统

1 在浏览器（推荐使用 Chrome）的地址栏中输入 **Dauth 控制台的 URL**，并单击**回车。**

2 在 Dauth 登录中心页面输入**用户名**、**密码**，然后单击**登录。**

注: 未开启多因素认证前，无需动态令牌。

3 在登录首页单击**自助服务**。

4 在**自助服务**页面，单击**管理多因素认证**。

5 在**更改 MFA 配置**页面，选择**是**。

6 打开手机上安装的身份认证器，扫描页面中的二维码或者手动输入 MFA 密钥，添加账号。

账号添加完成后，会发送该账号登录的动态令牌。

7 重新登录 Dauth 控制台，输入**用户名**、**密码**，再输入**动态令牌**，然后单击**登录**。

如果您的手机遗失了，无法获取动态令牌，请联系管理员帮您关闭多因素认证功能。

1.5 私有证书管理

在专有云环境中，SSL 证书由 Private Certificate Authority（PCA）统一管理，用于提高中间件产品的安全级别（信息安全等级保护四级）。

1.5.1 安装证书

Dauth 是中间件的鉴权系统，而 Dauth 依赖于 PCA。所以在使用中间件产品前，需要先安装证书。

- 请确认您的实际使用环境中的访问域名。并在登录 PCA 控制台（pca.example.com）时替换 example.com。
- 请确认您使用的浏览器。证书下载、安装是基于不同浏览器的。

  注: 本文档以 Chrome 为例进行说明。如果您使用其它浏览器，可以根据控制台的提示文案进行安装。

1 打开 Chrome 浏览器，在地址栏输入 **PCA 控制台 URL（pca.example.com）**，并**回车**。

说明:

- 请根据实际环境替换访问域名 example.com。
- 下载根证书无需登录。

2 单击**下载根证书**，将根证书下载到本地。

3 参考不同浏览器的安装教程，安装证书。

不同浏览器的安装步骤有所不同，本文档以 Chrome 为例进行说明。

a. 单击**菜单**，在下拉菜单中单击**设置**。

b. 在**设置**页面单击**高级**，在**高级设置**区域单击**管理证书**。

c. 在**证书**对话框中选择受信任的根证书颁发机构页签，然后单击**导入**。

d. 在**证书导入向导**对话框中，按页面提示操作，将下载的根证书导入。

注:

- 根证书为.pem 文件，在选择证书文件时默认没有此类型文件，需要将文件类型设置为所有文件。
- 在证书存储页面，需要选择受信任的根证书颁发机构。

1.5.2 管理证书

您在登录 PCA 后，还可以管理相关证书，包括添加证书、编辑证书和删除证书。

通过 Dauth 的管理员账号登录后，才能够管理证书。

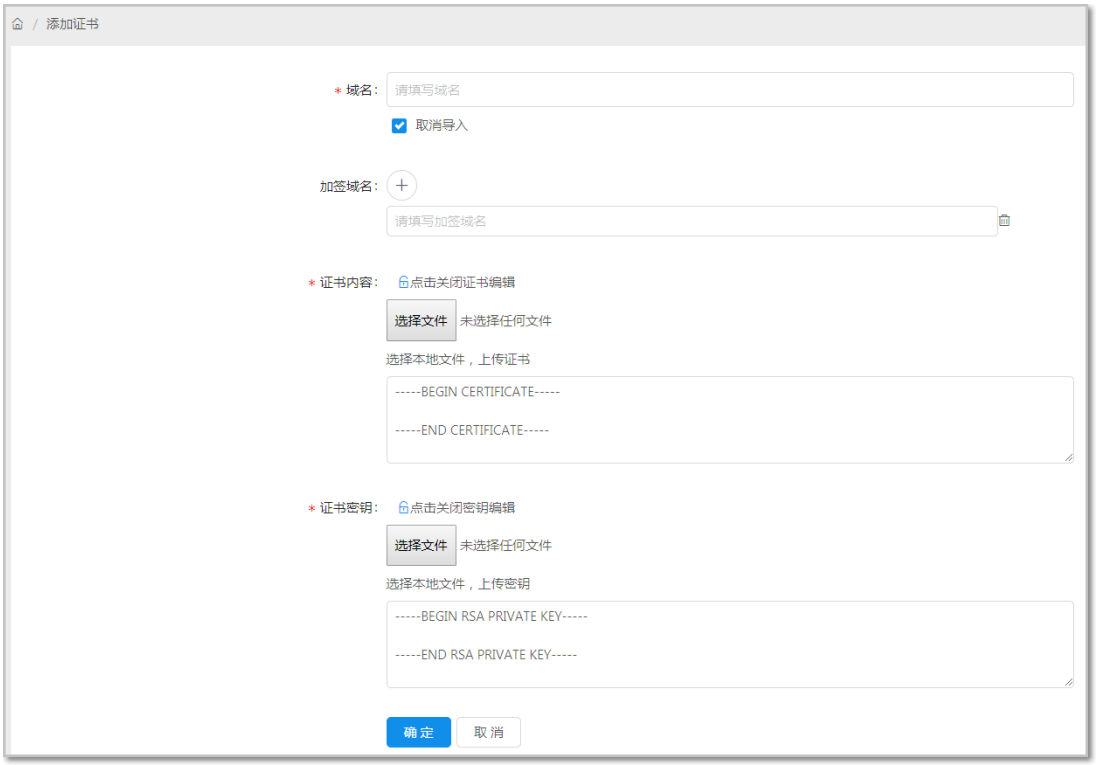
1 在 PCA 控制台右上角单击**登录**。

2 跳转到 Dauth 登录中心，输入 Dauth 的管理员**用户名**和**密码**，单击**登录**。

3 在 PCA 控制台左侧导航栏中单击**添加证书**。

4 输入您所购买的证书对应的域名，然后勾选**导入新证书**。

5 设置证书参数，然后单击**确定**。



- **域名**：证书对应的域名。
- **加签域名**：如果您购买的证书支持多个域名，请单击+，然后在输入。
- **证书内容**：可以选择证书文件或在文本框内输入证书内容。
- **证书密钥**：可以选择密钥文件或在文本框中输入密钥。

证书添加完成后，在控制台左侧的导航栏中单击证书列表，可以在证书列表页面编辑或删除证书。编辑证书的操作和添加证书类似、删除证书操作简单，不再赘述。

审计日志服务

审计日志服务（Audit Log Service，简称 ALS）为中间件产品提供审计日志服务，包括自动采集日志、日志内容展示与查询。ALS 还提供了日志采集节点的配置界面和抓取任务的触发与查询界面，用户可以自助完成采集节点的配置，以及数据补抓任务的触发。

1.5.3 查看审计记录

中间件产品通过 Dauth 登录控制台后，在控制台进行的所有操作都会上报给 ALS。在 ALS 控制台上可以查看这些操作记录。

1 登录 ALS 控制台（als.example.com）。

说明: example.com 需要用实际访问域名替换。

2 在 ALS 控制台左侧的导航栏中单击**产品/组件**。

3 在**监控信息**页面设置审计参数。

审计参数中，各产品会根据自身的特点定义参数，包括操作结果、匹配规则、操作类型、操作资源、原因、等。

1.5.4 管理节点

在 ALS 控制台，可以基于产品对节点进行管理。

1 登录 ALS 控制台（als.example.com）。

说明: example.com 需要用实际访问域名替换。

2 在 ALS 控制台左侧的导航栏中单击**产品/组件**。

3 在监控信息页面中单击**产品管理**页签。

4 单击**节点管理**，然后单击**添加节点**。

5 在**添加节点**对话框中输入**节点 IP**，单击**提交**。

添加的节点会显示在节点列表里。

1.5.5 管理抓取任务

在 ALS 控制台，可以基于产品对抓取任务进行管理。

1 登录 ALS 控制台（als.example.com）。

说明: example.com 需要用实际访问域名替换。

2 在 ALS 控制台左侧的导航栏中单击**产品/组件**。

3 在监控信息页面中单击**产品管理**页签。

4 单击**节点管理**，然后单击**抓取任务**。

5 单击**创建任务**，在创建任务对话框中设置**任务执行时间段**和**任务执行节点**，单击**创建**。

任务创建完成后，在抓取任务页面可以查看创建的任务及状态。

1.6 系统概览

登录 Butler 控制台后，首先会进入系统概览页面，可以查看容器、组件的整体状态、告警等信息。

1 在 Butler 控制台左侧的导航栏中单击**系统概览**。

2 在**系统概览**页面查看各 Region 内的容器、组件的整体状态及具体告警。

   系统概览页面在每个 Region 下包括三个部分：容器监控、未恢复的告警和快捷链接。

   - **容器监控：** 展示当前 Region 的容器个数及状态。容器状态分为三种：正常、异常和未知。容器的健康状况是由执行容器的 HealthCheck 脚本的结果决定的。如果容器没有配置健康检测脚本时，检测结果就会是未知。
   - **未恢复告警：** 展示当前 Region 内各组件还未恢复的告警，更多告警详情可点击链接“查看更多详情”查看。
   - **快捷链接：** 展示该 Region 管理的产品，以及产品的登录方式。

1.7 容器列表

容器列表用户对 Region 下的容器进行监控及升级。

因为 Butler 的部署信息要与 Panama 保持严格一致，所以想要通过 Butler 监控容器，必须要从 Panama 生成同步部署任务包，并同步到 Butler 上。

1 在 Butler 控制台左侧的导航栏中单击**容器列表**。

2 将从 Panama 下载的部署信息同步文件（文件名格式为 task_xxxx.tar.gz）拖入到上传 Panama 部署信息同步文件区域。

   右侧会显示同步文件名、大小、进度及状态。

3 单击**上传**，将文件上传到 Bulter 控制台。

   上传完成后，刷新容器列表，就可看到组件和容器的相关信息了。

4 在 Butler 控制台左侧的导航栏中单击**容器列表**。

容器列表页面包含三个部分：容器列表、上传 Panama 部署信息同步文件和上传镜像 diff 文件。

5 查看各 Region 中的容器及其信息。

容器列表按照产品、服务、组件三个层次展示各个组件的运行状况。一个组件可能又包含有多个运行实例，每个实例对应一个容器。

a. 在容器列表中查看组件的信息及状态。

组件列表展示的字段有：状态、产品、服务、组件、镜像 ID 和版本号。

通过绿红两种颜色来标识组件或容器的状态。

- 绿色：代表该组件下所有容器的运行状态正常。
- 红色：代表该组件下有容器的运行状态异常。

容器列表中提供切换版本按钮，单击可进行版本切换。

b. 在容器列表的展开列单击下拉按钮，查看该组件下的容器信息及状态。

容器列表展示的字段有：状态、容器 ID、IP、启动状态、启动时间和操作。

- 绿色：healthcheck 检测通过，表示容器正常。

- 红色：healthcheck 脚本不存在或者检测不通过，表示容器异常。

容器列表的操作列包含控制台、日志和重启按钮，单击可进入相应的页面或进行对应的操作。

当用户使用的是本地镜像仓库并且需要升级时，在本地生成镜像增量包并在拖入容器列表页面上传，即可把增量包并合到客户本地镜像仓库。

1.8 告警列表

在 Butler 控制台的告警列表中，可以查看各 Region 中产品组件当前的告警。

1 在 Butler 控制台左侧的导航栏中单击**告警列表**。

2 选择 **Region**，查看该 Region 中所有产品组件当前未恢复的告警。

告警信息包括发生时间、产品、组件、目标、告警内容。其中目标一般是异常节点的 IP，也可能是用户自定义的条件组合。

说明: 只有当异常节点是容器时才会有操作链接，其与容器列表部分的操作相同。

1.9 监控管理

Butler 可通过**巡检**和**指标**两种方式进行业务和系统层级的检测，在触发异常时，可通过**短信**、**邮件**或**钉钉**方式进行告警。

监控的产品包括：

- EDAS
- Butler
- CSB
- MQ
- DRDS

- GTS
- SchedulerX

监控管理本身不设定规则和方案，而是利用巡检管理和指标管理中的规则进行监控，然后在 Butler 控制台中已健康状态列表和图表的方式呈现。

- 健康状态列表：基于产品组件下的各容器展示基本信息及状态。具体包括：
  - **系统**：即组件名称
  - **类别**：包括巡检和指标
  - **目标**：监控容器的 IP
  - **状态**：根据巡检规则和指标规则检测得到容器状态，如果有异常，触发了告警，会显示告警信息。
  - **最近检测时间**：上一次检测的时间。
- 图表：展示容器的机器性能及相关监控信息。具体监控项取决于各产品配置的巡检及指标规则。

1.9.1 监控接入

Butler 的监控接入包括两部分：巡检接入和指标接入。而 Butler 除了在控制台配置巡检、指标规则外，还支持通过 SDK/API 的方式接入。本文档介绍如何通过 SDK/API 的方式将监控接入 Butler。

在专有云环境中，所有的组件，都是按照产品、服务和组件三层结构来表示的。所以在往 Butler 注册 CMDB 信息时，请保持这些信息与 docker-config 中的配置保持一致。

1.9.1.1 巡检接入

介绍如何在 Butler 中接入产品的巡检。

- 在目标机器（不论是容器还是虚机）上，请确保装有 **Tsar** 和 **StarAgent**。
- StarAgent 的白名单确保为 1~255，以保证有抓取日志的权限。
- 如果需要接入 JVM 监控，请在 Tomcat 应用中引入 eagleyeye-core 这个 JAR 包（一个机器只能配置一个 JVM，因为其日志固定输出到 /home/admin/logs/stat-eagleeye-jvm.log）。

1 在工程中添加 SDK 依赖。

```
<dependency>
    upId>com.alibaba</groupId>
    tifactId>butler-api-sdk</artifactId>
    rsion>2.0.1-SNAPSHOT</version>
</dependency>
```

2 在要接入 Butler 的产品系统代码中添加巡检规则。

```
public void registryDemo() throws Exception {
OpenAPIClient client = openAPIClient.getInstance("http://butler.console.butlertest.com
");

String region = "testRegion"; //所在区域

String product = "butler"; // 产品，请与 docker-config 产品目录名保持一致

String service = "butlerService"; // 服务，请与 docker-config 服务目录名保持一致

String component = "butler-all"; // 组件，请与 docker-config 组件目录名保持一致

String productName = "专有云监控运维平台"; //产品名字

String cname = "Butler 控制台"; // 组件名称

String version = "2.0.1"; //产品版本，如果是专有云环境，请与 docker-config 产品目录下

baseline-info.yml 中“版本”字段保持一致

String ip = "192.168.1.100"; // 组件 IP 地址

int port = 8080; // 组件对外提供的端口

String type = "DOCKER"; // 组件类型，如果是容器，值为 DOCKER；如果是虚机，值为

VM；如果是物理机，值为 PM。

// 注册产品

client.registryProduct(region, product, productName, version);

// 注册组件

client.registryComponent(region, product, service, component, cname, ip, port, type);

// 保存巡检规则，前提是产品和组件必须已注册过，否则会保存失败

client.savePatrol(new File("d:\\tmp\\initialized-patrol.yaml"));
```

```
}
```

initialized-patrol.yaml 代码示例

```
---
common: &common
checkInterval: 600
creator: admin
region: butlertest
available: true
rules:
- ruleName: deploy-broker(HTTP)检查
<<: *common
product: mq
service: broker
component: deploy-broker
callType: HTTP
errorDesc: "deploy-broker 服务异常"
target:
targetType: ALL_HOSTS
callScope: SINGLE
ruleDetail:
checkInfo: !!com.alibaba.butler.patrol.sdk.bean.HttpCheckInfoBean
method: GET
uri: /butler-cmdb/index.jsp
port: 8080
connTimeout: 3000
readTimeout: 8000
retryTime: 2
retryInterval: 1000
checkItems:
- itemType: HTTP_CODE
conditions:
- errorLevel: FATAL
priority: P0
compartor: "!="
right: 200
```

1.9.1.2 指标接入

介绍如何在 Butler 中接入产品的性能指标。

- 在目标机器（不论是容器还是虚机）上，请确保装有 **Tsar** 和 **StarAgent**。
- StarAgent 的白名单确保为 1~255，以保证有抓取日志的权限。

- 如果需要接入 JVM 监控，请在 Tomcat 应用中引入 eagleyeye-core 这个 JAR 包（一个机器只能配置一个 JVM，因为其日志固定输出到 /home/admin/logs/stat-eagleeye-jvm.log）。

1 在工程中添加 SDK 依赖。

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>butler-api-sdk</artifactId>
    <version>2.0.1-SNAPSHOT</version>
</dependency>
```

2 在要接入 Butler 的产品系统代码中添加巡检规则。

```
public void registryDemo() throws Exception {
OpenAPIClient client = openAPIClient.getInstance("http://butler.console.butlertest.com
");
String region = "testRegion"; //所在区域

String product = "butler"; // 产品，请与 docker-config 产品目录名保持一致

String service = "butlerService"; // 服务，请与 docker-config 服务目录名保持一致

String component = "butler-all"; // 组件，请与 docker-config 组件目录名保持一致

String productName = "专有云监控运维平台"; //产品名字

String cname = "Butler 控制台"; // 组件名称

String version = "2.0.1"; //产品版本，如果是专有云环境，请与 docker-config 产品目录下

baseline-info.yml 中“版本”字段保持一致

String ip = "192.168.1.100"; // 组件 IP 地址

int port = 8080; // 组件对外提供的端口

String type = "DOCKER"; // 组件类型，如果是容器，值为 DOCKER；如果是虚机，值为

VM；如果是物理机，值为 PM。

// 注册产品
client.registryProduct(region, product, productName, version);
// 注册组件
```

```
client.registryComponent(region, product, service, component, cname, ip, port, type);

// 保存巡检规则，前提是产品和组件必须已注册过，否则会保存失败

client.saveMetric(new File("d:\\tmp\\initialized-metric.yaml"));
}
```

代码示例

- initialized-metric.yaml

```
---
components:
  configserver:
    region: butlertest
    product: mq
    service: broker
    component: deploy-broker

    ruleName: deploy-broker 指标

    targets:
      - targetType: PREDEFINED
        targetPre: CONNECTION_COUNT

        targetName: 连接数

        alertEnabled: true
        alertInfo:
          strategy: CURRENT
          conditions:
            - operator: GTE
              value: 3000
              priority: P1
              level: WARN
```

- 添加默认系统指标规则代码

```
  public void registryDemo() throws Exception {

        OpenAPIClient client = OpenAPIClient.getInstance("http://butler.console.butlertest.com");

        String region = "testRegion"; //所在区域

        String product = "butler"; // 产品，请与docker-config 产品目录名保持一致

        String service = "butlerService"; // 服务，请与docker-config 服务目录名保持一致

        String component = "butler-all"; // 组件，请与docker-config 组件目录名保持一致
```

```
    String productName = "专有云监控运维平台"; //产品名字

    String cname = "Butler 控制台"; // 组件名称

    String version = "2.0.1"; //产品版本，如果是专有云环境，请与docker-config 产品目录下baseline-info.yml 中"版本"字段保持一致

    String ip = "192.168.1.100"; // 组件 ip 地址

    int port = 8080; // 组件对外提供的端口

    String type = "DOCKER"; // 组件类型，如果是容器值为DOCKER，如果是虚机值为VM，如果是物理机值为PM


    // 注册产品
    client.registryProduct(region, product, productName, version);
    // 注册组件
    client.registryComponent(region, product, service, component, cname, ip, port, type);
    // 保存指标规则，前提是产品和组件必须已注册过，否则会保存失败
    client.addDefaultInfrastructureMetric(region, product, service, component);
  }
```

- 添加默认 JVM 指标
  - 在 Java 应用中引用 eagleeye-core 生成 JVM 相关日志

```
<dependency>
    <groupId>com.taobao.eagleeye</groupId>
    <artifactId>eagleeye-core</artifactId>
    <version>1.4.8</version>
</dependency>
```

  - 在 Java 应用中拉起 eagleyeye，自动生成 /home/admin/logs/stat-eagleeye-jvm.log

```
Class.forName("com.taobao.eagleeye.EagleEye");
```

1.10 系统管理

系统管理是 Butler 的核心管控模块，可对所有 Region 进行系统层面的管控。

系统管理包括五部分：调度管理、巡检管理、指标管理、联系人管理和通知管理。

1.10.1 调度管理

调度管理部分负责调度巡检和指标监控，以及触发的告警，每一条调度规则对应一个巡检或指标规则。

1 在 Butler 控制台左侧的导航栏中选择**系统管理 > 调度管理**。

2 在调度管理页面，查看调度规则。

   调度任务的信息包含任务类型、注册参数、任务名称、任务分组、调度周期、调度状态、下次触发时间、上次触发时间、调度开始时间和操作。

   其中，调度周期为 Quartz 的 Cron 表达式。

3 在调度列表的操作列中单击**立即触发**，立即触发一次拨测，并及时返回拨测结果。

4 启动调度规则。

   启动调度规则包括一键启动和启动两种方式。

   - **一键启动**：启动所有调度规则。
   - **启动**：在调度列表的操作列中单击启动，启用该调度规则，并且随着时间推移，下次出发时间和上次触发时间会根据调度周期而发生变更。

5 停止调度规则。

   停止调度规则包括一键停止和停止两种方式。

   - **一键停止**：停止所有调度规则。
   - **停止**：在调度列表的操作列中单击停止，停止该调度规则。并且下次触发时间、上次触发时间、调度开始时间会被清空。

6 在调度列表的操作列中单击**删除**，删除该规则，与该规则对应的巡检或指标规则将不会被再次调度。

1.10.2 巡检管理

巡检管理用于创建巡检规则并执行巡检。

1 在 Butler 控制台左侧的导航栏中选择**系统管理 > 巡检管理**。

2 在**巡检配置管理**页面单击**创建巡检配置**。

3 在**配置巡检对象**页面选择**模板**，单击**巡检类型（HTTP）**。

- **模板**：包括全部模板、系统模板和自定义模板。
- **巡检类型**。目前支持 HTTP、PING、TCP 和 DB 四种巡检方式。本文档以 HTTP 巡检规则为例进行说明。

4 在**配置巡检对象**页面设置巡检对象参数。包括**租户**（Region）、**产品**、**服务**、**组件及 IP**，然后单击**下一步**。

说明: IP 即组件中容器的 IP 地址，可以选择全部机器和部分机器，如果选择全部机器，无需设置 IP；如果选择部分机器，在容器 IP 列表中选择需要巡检的容器的 IP。

5 在**配置巡检方案**页面设置巡检规则参数，然后单击**创建巡检配置**。

巡检规则（方案）有普通配置和高级配置两种，参数说明如下：

名称	高级配置	含义
规则名称	否	巡检规则的名字，建议填写简明易懂的名称
HTTP 端口	否	HTTP 的端口号
HTTP URI	否	因 IP 在第二步已经被选中，所以这里只需填写拨测的 URI 即可
检测周期	否	巡检的检测周期，包括 5 分钟、10 分钟、15 分钟、30 分钟和 60 分钟。

名称	高级配置	含义
出错描述	否	在巡检出错时，发出的告警内容。其中有一些关键字可通过表达式来获取。比如通过${url}获取拨测的完整URL。
状态码检测	否	这里可配置多条状态码匹配规则。规则按从上到下的顺序匹配，只要有任一规则匹配成功后就会触发告警。**目前在控制台只支持状态码检测，通过 OpenAPI 或者 SDK 还可做响应内容和响应时间检测。**
HTTP 请求类型	是	默认为 GET 请求，还可配置 POST、HEAD 和 DELETE 请求。
拨测范围	是	拨测范围分两种：单节点拨测和多节点拨测，默认是单节点拨测。 单节点拨测中各个拨测对象结果互相独立 多节点拨测会把多个拨测对象结果做比对，如果返回不同则认为异常
读超时时间	是	从建立起 HTTP 链接到读取第一个字节的超时时长，单位 ms。
连接超时时间	是	从发起请求到建立起 HTTP 连接的超时时长，单位 ms。
AccessKey	是	在发起阿里云签名时的公钥
SecretKey	是	在发起阿里云签名时的密钥

创建完成后，在巡检管理页面就可以看到刚创建的巡检规则了。

在创建巡检规则后，可以通过操作列的按钮进行如下操作：

- **立即拨测**： 会立即触发一次拨测，并及时返回拨测结果。这在刚创建好巡检规则时验证规则有效性时比较常用。
- **编辑规则**： 可编辑已创建的巡检规则。编辑步骤与创建巡检规则流程类似，不再累述。
- **删除规则**： 可删除已创建的巡检规则，同时也会把调度管理里对应的调度规则删掉。

1.11 指标管理

指标管理用于创建、管理监控指标及规则。

指标规则是基于日志分析汇聚后生成的合成指标而创建，常用来监控系统指标和业务自定义指标。

1 在 Butler 控制台左侧的导航栏中选择**系统管理** > **指标管理**。

2 在**指标配置管理**页面单击**创建指标规则配置**。

3 在创建规则页面设置规则参数，包括**规则名称**、**租户（Region）**、**产品**、**服务**、**组件**、**服务器**、监控**周期**、**有效时间**和**触发条件**。

   服务器即组件下的容器，可以选择全部（该组件下的所有容器，包括后续新增的容器），也可以选择部分容器，对这些容器进行逐台检测。

   触发条件：“任一条件”是指任何一个监控对象满足告警条件就触发告警；“全部条件”是所有监控对象都满足告警条件时才触发告警

   说明: 规则创建完成后，不能保存，必须创建监控对象。

4 单击**新增监控对象**。

5 在**监控对象**区域设置对象参数。

   监控对象包括内置和自定义两种类型，相应的参数说明如下：

名称	对象类型	含义
对象名称	All	监控对象名称，方便在告警时定位到告警对象
对象描述	All	对象名称的描述信息
指标类型	自定义	包含直接指标和简介指标： “直接指标”是直接通过数据生成的指标 “间接指标”是依赖其他指标生成新的指标
查询标识	自定义	StageId 是 Tlog 任务的 StageId BizId 是 Tlog 任务中的查询 ID dateId 是 Tlog 任务中时间戳字段
查询维度	自定义	**名称**：Tlog 任务的查询 id 下的聚合字段 **值**：可以取值为通配符*，也可以配置确切的值

名称	对象类型	含义
直接指标	自定义	**运算**表示直接指标的生成方式，包括 A、A+B、A-B、A/B 和 A/(A+B)五种方式，列 A 列 B 是查询维度中的**名称**字段
自定义乘积	自定义	默认是 1，上述直接指标乘以该值作为最终指标值
小数位	自定义	保留小数位个数
指标单位	自定义	指标单位名称
开启报警	自定义	选择是则规则生效，选择否则规则不生效
取值策略	自定义	取值策略有当前值、环比、上周同比等 7 种策略
比较方式	自定义	可设定多条比较规则，按照从上到下顺序匹配，只要满足任一规则则触发报警

6 设置完成后，单击**保存**。

创建完成后，在指标管理页面就可以看到刚创建的指标规则了。

- **立即触发规则**： 会立即触发一次拨测，并及时返回拨测结果。
- **编辑规则**： 可编辑已创建的指标规则。编辑步骤与创建指标规则流程类似，不再累述。
- **删除规则**： 可删除已创建的指标规则，同时也会把调度管理里对应的调度规则删掉。

1.11.1 联系人管理

联系人管理包括联系人管理和联系组管理，用户报警通知。

一条联系人记录包括联系人名称、手机号码和邮箱；一个联系人组可包括一条或多条联系人。

一个联系人组对应一个产品，即该产品的所有告警信息都会发给绑定的联系人分组。

1.11.1.1 联系人管理

1 在 Butler 控制台左侧的导航栏中选择**系统管理** > **联系人管理**。

2 在**联系人管理列表**页面中单击**联系人**页签，然后单击**新建联系人**。

3 在**新建联系人**对话框中输入**账号**、**手机号码**和**邮箱**，然后单击**确定**。

   新建联系人完成后，即可在联系人页签中查看联系人信息。

4 管理联系人。

   在联系人列表中的操作列，单击编辑、删除，可以编辑联系人或删除联系人。

1.11.1.2 联系组管理

1 在联系人管理列表页面中单击**联系人组**页签，然后单击**新建联系组**。

2 在新建联系组对话框中设置联系组参数，包括**组名**、**租户**（Region）、**产品名**，并添加**联系人**，然后单击**确定**。

   新建联系人完成后，即可在联系人组页签中查看联系组信息。

3 管理联系组。

   在联系组列表的操作列，单击编辑、删除，可以编辑或删除联系组。单击下拉箭头可以查看该组中的联系人，单击移除，可以将联系人从该组中删除。

1.11.2 通知管理

通知管理用于配置用户的短信网关和邮件服务器，以便接收告警的短信或邮件，主要在封闭网络环境中使用。

通知管理包括邮件设置和短信设置两部分。

- **邮件设置**用来配置客户的邮件服务器地址、登录账户密码和是否开启 SSL。
- **短信设置**需要配置客户的短信网关接口，该接口需要能够正确接收并解析 Butler 提供的告警数据。目前 Butler 的告警数据以 JSON 格式输出，例如：{"mobiles":["1381111111"],"content":"this is a test sms"}。

在 Butler 控制台左侧的导航栏中选择**系统管理** > **通知管理**。

1.11.2.1 邮件设置

1 在**邮件设置**区域中单击**编辑**。

2 在更新邮件设置对话框中输入 **SMTP 服务器的 URL**、登录 SMTP 服务器的**用户名**和**密码**、并选择是否为 SSL 以及是否启动，然后单击**确定**。

1.11.2.2 短信设置

1 在**短信设置**区域中单击**编辑**。

2 在更新短信设置区域设置参数，然后单击**确定**。

短信参数说明

- **示例 JSON**：按照 JSON 的格式编写短信模板，如 {"mobiles":["1381111111"],"content":"this is a test sms"}。
- **短信服务**：设置短信网关的 URL。
- **启用否**：是否启用。

2 巡检

本文档中仅介绍如何通过 Butler 控制台或产品运维控制台对方舟平台的容器/组件及产品进行巡检。命令行巡检请参考具体产品的《运维指南》中的相关内容。

所以，本文档及介绍如何通过 DRDS Manager 对 DRDS 进行巡检。

2.1 DRDS 巡检

2.1.1 DRDS 监控与问题发现

2.1.1.1 登录 DRDS-manager 管理控制台

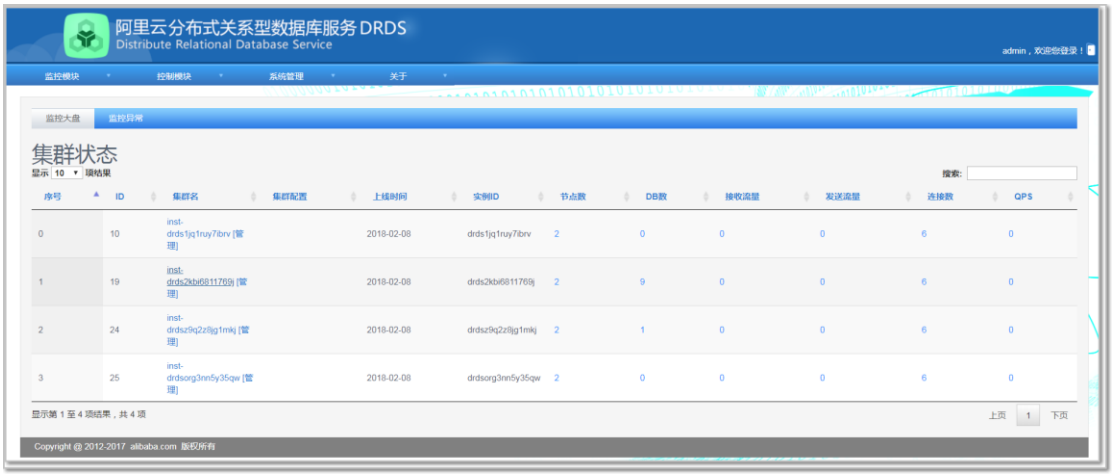
1 打开浏览器（推荐使用 Chrome），在地址栏中输入 Butler 控制台的 URL。如 http://butler.console.example.com/。

使用时，需要根据具体环境，将 example.com 替换为实际的访问域名。

注意：在登录 Butler 控制台前，可能需要先绑定 host 或 DNS。

2 在 Dauth 登录中心页面中，输入登录 DRDS-manager 的用户名和密码，单击**登录**。

3 在系统概览页面快速链接区域中单击 **DRDS-manager** 图标。



登录控制台后，默认进入监控大盘页面。在监控大盘页面，可以看到所有集群的信息及状态。

2.1.1.2 监控实例

1 在菜单栏中选择 **监控模块** > **实例管理**。

2 在实例管理页面，查看实例信息。

3 单击 **监控**，进入集群信息页面。

该页面中主要包括：实例基本信息、统计数据和节点信息。

2.1.1.3 监控节点

1 在集群信息页面单击 **节点信息**。

2 在节点信息页签查看该实例中节点的信息和一些即时状态。

3 在节点列表中单击节点名称。

4 在节点信息页面中监控该节点相关状态。

在该页面中，可以监控该节点的内存使用情况、总内存、接收/发送流量、连接数、QPS、处理器信息、线程池、SQL 统计、连接数、连接池、数据库、流量及报警。

2.1.1.4 监控数据库

1 控制台菜单栏中选择 **监控模块** > **监控大盘**。

2 在监控大盘页面单击具体集群名。

3 在集群信息页面监控该实例（集群）下数据库的状态。

4 单击具体 Database，监控该数据库的详细信息及状态。

3 运维操作

3.1 产品启动操作

方舟是基于 docker 进行虚拟化部署的，所以对产品容器的操作都是基于 docker 相关命令的。

3.1.1 启动 EDAS&DRDS 依赖的应用

启动 dauth-cai

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep dauth-cai|awk '{print $1}') 重启：docker restart $(docker ps -a|grep dauth-cai|awk '{print $1}')	

启动 dauth-diamond

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep dauth-diamond|awk '{print $1}') 重启：docker restart $(docker ps -a|grep dauth-diamond|awk '{print $1}')	

启动 dauth-server

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep dauth-server|awk '{print $1}') 重启：docker restart $(docker ps -a|grep dauth-server|awk '{print $1}')	

启动 zk_middleWare

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	

主机名	PMNYDRD05	IP 地址	client 机器 IP
命令		启动：docker start $(docker ps -a|grep zk_middleWare|awk '{print $1}') 重启：docker restart $(docker ps -a|grep zk_middleWare|awk '{print $1}')	

启动 histore_middleWare

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep histore_middleWare|awk '{print $1}') 重启：docker restart $(docker ps -a|grep histore_middleWare|awk '{print $1}')	

启动 cproxy

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep cproxy|awk '{print $1}') 重启：docker restart $(docker ps -a|grep cproxy|awk '{print $1}')	

启动 redis_middleWare

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep redis_middleWare|awk '{print $1}') 重启：docker restart $(docker ps -a|grep redis_middleWare|awk '{print $1}')	

启动 cai-address_middleWare

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep cai-address_middleWare|awk '{print $1}') 重启：docker restart $(docker ps -a|grep cai-address_middleWare|awk '{print $1}')	

启动 vipserver-service_middleWare

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep vipserver-service_middleWare|awk '{print $1}') 重启：docker restart $(docker ps -a|grep vipserver-service_middleWare|awk '{print $1}')	

启动 tlog-hbase

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep tlog-hbase|awk '{print $1}') 重启：docker restart $(docker ps -a|grep tlog-hbase|awk '{print $1}')	

启动 tlog-console

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep tlog-console|awk '{print $1}') 重启：docker restart $(docker ps -a|grep tlog-console|awk '{print $1}')	

启动 jstorm_tlog

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	

主机名	PMNYDRD05	IP 地址	client 机器 IP
命令		启动：docker start $(docker ps -a|grep jstorm_tlog|awk '{print $1}') 重启：docker restart $(docker ps -a|grep jstorm_tlog|awk '{print $1}')	

启动 tlogst

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep tlogst|awk '{print $1}') 重启：docker restart $(docker ps -a|grep tlogst|awk '{print $1}')	

启动 diamond-service_middleWare

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep diamond-service_middleWare|awk '{print $1}') 重启：docker restart $(docker ps -a|grep diamond-service_middleWare|awk '{print $1}')	

启动 dbpaas-api-server

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep dbpaas-api-server|awk '{print $1}') 重启：docker restart $(docker ps -a|grep dbpaas-api-server|awk '{print $1}')	

启动 dbpaas-dbfree

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep dbpaas-dbfree|awk '{print $1}') 重启：docker restart $(docker ps -a|grep dbpaas-dbfree|awk '{print $1}')	

启动 dbpaas-fstack

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep dbpaas-fstack|awk '{print $1}') 重启：docker restart $(docker ps -a|grep dbpaas-fstack|awk '{print $1}')	

启动 dbpaas-hitsdb

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep dbpaas-hitsdb|awk '{print $1}') 重启：docker restart $(docker ps -a|grep dbpaas-hitsdb|awk '{print $1}')	

启动 dbpaas-jstorm

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep dbpaas-jstorm|awk '{print $1}') 重启：docker restart $(docker ps -a|grep dbpaas-jstorm|awk '{print $1}')	

启动 dbpaas-kafka

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep dbpaas-kafka|awk '{print $1}') 重启：docker restart $(docker ps -a|grep dbpaas-kafka|awk '{print $1}')	

启动 dbpaas-metadb

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep dbpaas-metadb|awk '{print $1}') 重启：docker restart $(docker ps -a|grep dbpaas-metadb|awk '{print $1}')	

启动 dbpaas-saltstack

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep dbpaas-saltstack|awk '{print $1}') 重启：docker restart $(docker ps -a|grep dbpaas-saltstack|awk '{print $1}')	

启动 dbpaas-adha-console

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep dbpaas-adha-console|awk '{print $1}') 重启：docker restart $(docker ps -a|grep dbpaas-adha-console|awk '{print $1}')	

3.1.2 启动 EDAS

启动 cai-address

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep cai-address_edas|awk '{print $1}') 重启：docker restart $(docker ps -a|grep cai-address _edas|awk '{print $1}')	

启动 configserver

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep configserver|awk '{print $1}') 重启：docker restart $(docker ps -a|grep configserver|awk '{print $1}')	

启动 edas-server

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep edas-server|awk '{print $1}') 重启：docker restart $(docker ps -a|grep edas-server|awk '{print $1}')	

启动 edas-console

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep edas-console|awk '{print $1}') 重启：docker restart $(docker ps -a|grep edas- console|awk '{print $1}')	

启动 edas-admin

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep edas-admin|awk '{print $1}') 重启：docker restart $(docker ps -a|grep edas- admin|awk '{print $1}')	

启动 cai-fs

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep cai-fs_edas|awk '{print $1}') 重启：docker restart $(docker ps -a|grep cai-fs_edas|awk '{print $1}')	

3.1.3 启动 DRDS

启动 rtools

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep rtools_drds|awk '{print $1}') 重启：docker restart $(docker ps -a|grep rtools_drds|awk '{print $1}')	

启动 mc

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep mc_drds|awk '{print $1}') 重启：docker restart $(docker ps -a|grep mc_drds|awk '{print $1}')	

启动 drds-console

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep drds-console|awk '{print $1}') 重启：docker restart $(docker ps -a|grep drds-console|awk '{print $1}')	

启动 drds-dayu

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep drds-dayu|awk '{print $1}') 重启：docker restart $(docker ps -a|grep drds-dayu|awk '{print $1}')	

启动 drds-manager

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep drds-manager|awk '{print $1}') 重启：docker restart $(docker ps -a|grep drds-manager|awk '{print $1}')	

启动 drds-jingwei-console

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep drds-jingwei-console|awk '{print $1}') 重启：docker restart $(docker ps -a|grep drds-jingwei-console|awk '{print $1}')	

启动 drds-server

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep drds-server|awk '{print $1}') 重启：docker restart $(docker ps -a|grep drds-server|awk '{print $1}')	

启动 jingwei-worker

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep jingwei-worker|awk '{print $1}') 重启：docker restart $(docker ps -a|grep jingwei-worker|awk '{print $1}')	

启动 yugong

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep yugong_drds|awk '{print $1}') 重启：docker restart $(docker ps -a|grep yugong_drds|awk '{print $1}')	

3.1.4 启动 MQ

启动 nameserver

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep mq-namesrv|awk '{print $1}') 重启：docker restart $(docker ps -a|grep mq-namesrv|awk '{print $1}')	

启动 mq-console

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep mq-console|awk '{print $1}') 重启：docker restart $(docker ps -a|grep mq-console|awk '{print $1}')	

启动 mq-monitor

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep mq-monitor|awk '{print $1}') 重启：docker restart $(docker ps -a|grep mq-monitor|awk '{print $1}')	

启动 broker

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep deploy-broker|awk '{print $1}') 重启：docker restart $(docker ps -a|grep deploy-broker|awk '{print $1}')	

启动 mq-controller

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep mq-controller|awk '{print $1}') 重启：docker restart $(docker ps -a|grep mq-controller|awk '{print $1}')	

启动 mq-cai

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	

主机名	PMNYDRD05	IP 地址	client 机器 IP
命令		启动：docker start $(docker ps -a|grep mq-cai|awk '{print $1}') 重启：docker restart $(docker ps -a|grep mq-cai|awk '{print $1}')	

启动 mq-diamond

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep mq-diamond|awk '{print $1}') 重启：docker restart $(docker ps -a|grep mq-diamond|awk '{print $1}')	

3.1.5 启动 GTS

启动 txc-server

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		启动：docker start $(docker ps -a|grep txc-server|awk '{print $1}') 重启：docker restart $(docker ps -a|grep mq-namesrv|awk '{print $1}')	

3.1.6 应用停止

3.1.6.1 停止 vipclient

主机名	PNCMCMS01/PNCMCMS02 PNCMWEB1/PNCMWEB3 PNMNYAPP03/PNMNYAPP04 PNMNYAPP05-PNMNYAPP12	IP 地址	192.168.51.5 / 192.168.51.6 192.168.52.7 / 192.168.52.9 192.168.52.175 / 192.168.52.176 192.168.51.151 - 192.168.51.158
用户		root	
命令		/home/xxxcompany/vipsrv-all/vipsrv-dns-0.3.5-202/bin/stop.sh	

主机名	PMNYCMS1/PMNYCMS2	IP 地址	192.168.51.2 / 192.168.51.4
用户		root	
命令		/home/bea/vipsrv-all/vipsrv-dns-0.3.5-202/bin/stop.sh	

3.1.6.2 停止 EDAS

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		docker stop $(docker ps -a|grep edas-server|awk '{print $1}')	

3.1.6.3 停止 DRDS

主机名	PMNYDRD05	IP 地址	client 机器 IP
用户		xxxcompany	
命令		docker stop $(docker ps -a|grep drds-server|awk '{print $1}')	

3.2 方舟命令行运维操作

3.2.1 命令行入口介绍

方舟命令行运维会用到两种命令行操作。

Swarm mode CLI 和 Docker Engine CLI

3.2.1.1 Swarm mode CLI

- 操作范围作用到整个集群。
- 默认在方舟部署的 client 客户端生效，只有客户端和集群 Swarm Manager 正常工作时才能正常工作。
- 对应的 DOCKER_HOST 是某台 Swarm Manager 服务的监听端口 3376。
- docker info 在 client 客户端上运行这个命令可以显示整个集群中运行的所有节点。
- docker ps 在 client 客户端上运行这个命令可以显示整个集群中运行的所有容器。
- 因为是集群操作，谨慎执行如 docker pull 这种操作。

3.2.1.2 Docker Engine CLI

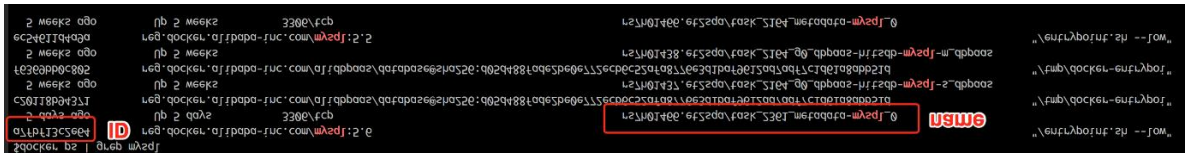
- 操作范围作用到某台主机的单个 Docker Engine。
- 在每个部署了 Docker Engine 的机器上生效。
- 对应的 DOCKER_HOST 是本台主机上 Docker Engine 监听端口 2376。
- sudo docker info 在任意主机上运行这个命令可以显示当前主机 Docker Engine 的信息。
- sudo docker ps 在任意主机上运行这个命令可以显示当前主机上运行的容器。
- sudo docker pull 在任意主机上运行这个命令只会拉取该主机需要的镜像。

3.2.2 查找容器-ID or Name

swarm-cli 下：

```
docker ps | grep ${容器特征值}
```

此时可以获得容器 id 或者容器 name。



3.2.3 管理容器

```
docker exec
Usage:      docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
Run a command in a running container
  -d, --detach              Detached mode: run command in the background
  --detach-keys             Override the key sequence for detaching a container
  --help                    Print usage
  -i, --interactive         Keep STDIN open even if not attached
  --privileged              Give extended privileges to the command
  -t, --tty                 Allocate a pseudo-TTY
  -u, --user                Username or UID (format: <name|uid>[:<group|gid>])
```

3.2.3.1 登录容器

- 进入容器的方式有若干种，推荐使用 docker exec -it 登录。
- 不建议使用的的登录容器的方式：ssh

  一般容器内不会启动 sshd

  在容器是以 —net=host 启动时，容器的 IP 和宿主机 IP 相同。此时通过 ssh 登录很可能会登录到宿主机而非容器内。
- docker exec -it ${容器 id/name} bash 可以登录指定容器。
- 如果遇到报错：

  ```
  rpc error: code = 2 desc = “oci runtime error: exec failed: exec: “bash”: executable file not found in $PATH”
  ```

  表示该容器内没有配置 bash 命令，换用：

  ```
  docker exec -it ${容器 id/name} sh
  ```

3.2.3.2 不进入容器执行命令

```
docker exec -it ${容器 id/name} ${cmd} $[Args]
```

例如：docker exec -it 59615bd706bc uname -a

3.2.3.3 删除容器

- docker kill 容器 ID //kill 容器
- docker rm 容器 ID //执行完 kill 再执行 rm 后才会删除

- docker rm -f 容器 ID //强制删除容器
- docker rm -fv 容器 ID //强制删除容器和与其关联的数据存储卷 Volumes

例如：docker rm -fv 59615bd706bc

3.2.3.4 启动/停止/重启容器

- docker start 容器 id
- docker stop 容器 id
- docker restart 容器 id

例如：docker start 59615bd706bc

3.2.4 查看容器日志

3.2.4.1 查看容器本身日志

```
docker inspect –format=´{{.LogPath}}´ ${容器 id/name}
```

得出的结果是容器所在宿主机的路径。如果是在 client 端执行该命令，还需要跳到容器的宿主机上查看，容器本身的日志一般对产品定位问题意义不大。

3.2.4.2 查看容器内的应用程序日志

一般是进入容器，定位寻找，常见的位置 /home/admin/logs。

docker-config 中的 log_path，这是日志路径配置参数。

3.2.4.3 查看容器启动删除等日志

```
sudo docker events --since="1505122429" --until="1506677629"
```

3.2.5 分析容器元信息

```
docker inspect

https://docs.docker.com/engine/reference/commandline/inspect/
Usage:	docker inspect [OPTIONS] CONTAINER|IMAGE [CONTAINER|IMAGE...]
Return low-level information on a container or image
  -f, --format	Format the output using the given go template
  --help	Print usage
  -s, --size	Display total file sizes if the type is container
  --type	Return JSON for specified type, (e.g image or container)
```

```
-f, –format
```

go 语言的模板

使用 go 语言模版可以准确地格式化输出容器地属性，这种操作实际上更适合程序化的操作。在命令行交互使用中可以用 grep 替代这种操作（较简单）。

3.2.6 查看方舟集群整体情况

3.2.6.1 确定容器所在物理机

- 物理机模式

```
docker inspect ${容器 id/name} | grep ACS_DOCKER_HOST
```

- 虚拟机模式

```
docker inspect ${容器 id/name} | grep ark.ip
```

3.2.6.2 查看集群节点情况

在 client 客户端机器上执行 docker info，返回集群所有节点的使用情况。

3.2.7 网络相关运维

3.2.7.1 判断网络状况

- 查看网络插件容器是否正常，每个主机节点上的容器与外界的网络通信依赖于主机的网络和该主机节点上运行的两个容器 ovs 和 vlan。
  - sudo docker ps -a | grep ovs 或者 sudo docker ps -a | grep openvswitch 用来查看 ovs 容器
  - sudo docker ps -a | grep vlan 或者 sudo docker ps -a | grep networkplugins 用来查看 vlan 容器
- 查看 ovs 容器日志

```
docker exec -it ovs 容器 Id sh //进入 ovs 容器

cd /tmp/supervisor/ //进入目录查看日志
```

```
ovs-vsctl show //查看一些基本信息
```

- 查看网络信息

```
docker network ls //物理机部署使用的网络为 vlan

docker network inspect vlan700 //查看此 vlan 下的各个容器网络信息
```

- 抓包判断是 docker 网络插件的问题，还是网络配置问题导致网络不通：
  - 通过 docker exec –it 容器 id bash，进入容器内部。
  - ping 物理机的网关。
  - 在物理机 tcpdump-I bond0 –netvv host 容器 ip。
    - bond0 是物理机网卡。
    - 如果在容器内 ping 物理机网关，tcpdump 命令可以看到数据包 则说明 docker 网络插件正常需要排查网络配置；否则可尝试 重启 docker 网络插件。

3.2.7.2 网络初始化

1 创建/usr/local/src/docker_rc.sh，并拷贝如下内容：

```
#!/bin/bash
sudo modprobe openvswitch
sudo service docker restart
sudo docker restart ovs
sudo docker restart vlan # Change the vlan to vxlan in vxlan mode
sudo docker ps | grep task | awk '{print $1}' | xargs
sudo docker restart echo "`date`" > /root/result.log
```

2 为/usr/local/src/docker_rc.sh 添加执行权限。

```
chmod +x /usr/local/src/docker_rc.sh
```

3 在/etc/rc.d/rc.local 中添加如下语句：

```
su - root -c "/usr/local/src/docker_rc.sh"
```

3.2.8 组件容器的健康检查

对容器运行状态进行健康检查：

- 获取健康检查命令

```
cd /home/chinamoney/ark/tasks/installing/task_xxx
cat docker-run-all.sh | grep check | sed  's/$PWD/./g' |
sed 's/$INTERVAL/10/g'|sed 's/$COUNT/10/g'
```

- 在上一步命令获取到所有组件的健康检查语句后，可以进一步按照组件挑选需要的语句。

  如获得：

  ```
  sh ./checkHealth.sh url task_607_g0_yugong_drds_2 "python /checkHealth.py" 10 10
  ```

  直接执行即可，如果返回 SUCCESS，说明业务状态正常。

4 故障处理

4.1 方舟常见问题

4.1.1 容器无法删除

常见 docker restart 异常信息：

```
docker service endpoint with name already exists

docker network disconnect -f vlanID 容器 ID
```

4.1.2 清理未使用镜像

```
#/bin/bash
cd `dirname $0`
PWD=`pwd`
TMP=images_used.txt
sudo docker ps -a   | awk '{print $2}' | awk -F':' '{print $1}' | awk -F'@sha256' '{print $1}' | sort|
uniq > $TMP
LINES=`sudo docker images   | awk '{printf"%s:%s\n",$1,$3}' | grep -v "REPOSITORY" | sort`
for LINE in $LINES
do
    IMAGEID=${LINE#*:}
    NAME=${LINE%:*}
    grep $NAME $TMP | grep -v grep > /dev/null
    if [ $? -ne 0 ]; then
        echo "start remove image $NAME, imageid=$IMAGEID..."
        sudo docker rmi $IMAGEID
        echo "end remove image $NAME, imageid=$IMAGEID."
    fi
done
```

4.1.3 清理 ovs 日志

1 创建脚本/usr/local/src/ovsCleaner.sh，脚本内容如下：

```
#!/bin/bash
CONID=`sudo docker ps -a | grep ovs | awk -F' ' '{print $1}'`
READ=`sudo docker inspect --format={{.Id}} ${CONID}`
FILE=/data/docker/containers/${READ}/${READ}-json.log
echo "sudo sh -c \"echo '' > $FILE\""
sudo sh -c "echo '' > $FILE"
```

2 添加脚本执行权限：

```
chmod +x /usr/local/src/ovsCleaner.sh
```

3 在 root 用户下添加 crontab 任务。

执行 crontab -e，并添加如下任务

```
/10 * * * * sh /usr/local/src/ovsCleaner.sh
```

4.1.4 容器被删除

大部分原因可以在系统的 messages 中找到，其中一类是由于宿主机的内存不够导致删除了。Docker 计算容器所需的 mem 是根据 free 的资源计算的，在很多非标输出场景，free 所剩的 mem 是不够的，但 available 是够的，需要强制将 buf/cache 里的数据刷出才行。

运行 sync 将 dirty 的内容写回硬盘

```
$sync
```

通过修改 proc 系统的 drop_caches 清理 free 的 cache

```
$echo 3 > /proc/sys/vm/drop_caches
```

或者

```
sudo su -c "echo 3 > /proc/sys/vm/drop_caches"
```

drop_caches 的详细文档如下：

```
Writing to this will cause the kernel to drop clean caches, dentries and inodes from memory,
causing that memory to become free.

To free pagecache:

    * echo 1 > /proc/sys/vm/drop_caches

To free dentries and inodes:

    * echo 2 > /proc/sys/vm/drop_caches

To free pagecache, dentries and inodes:

    * echo 3 > /proc/sys/vm/drop_caches
```

4.1.5 启动容器 ID 冲突

容器启动的时候报重名，但是 docker ps 看不到。

如下步骤一个一个尝试：

- 在所有机器上查找有没有同名容器，如果可以找到手动处理。

```
sh ark/tools/batch.sh "sudo sh -c \"docker ps -a | grep 容器名\""
```

- 尝试断开找不到容器的 vlan 网络。

```
sh ark/tools/batch.sh "sudo docker network disconnect -f vlanxx 容器名"
```

- 尝试重启 swarm（**慎重使用**）。

```
sh ark/tools/batch.sh "sudo docker restart swarm-agent"
```

4.1.6 创建本地镜像仓库

在客户环境不能连公网的情况下，需要在客户机上搭建镜像仓库，搭建步骤如下：

1 创建镜像仓库根目录。

该目录由 ark.properties 的变量 registry_dir 指定，比如该值为：/var/images

2 上传镜像到镜像仓库根目录。

把镜像压缩包上传到/var/images 目录下并解压。

比如镜像包为：base.tar.gz，解压后其目录结构如下所示：

```
data/
data/docker/
data/docker/registry/
data/docker/registry/v2/
data/docker/registry/v2/blobs/
data/docker/registry/v2/blobs/sha256/
data/docker/registry/v2/blobs/sha256/90/
data/docker/registry/v2/blobs/sha256/90/9050ace8ca9ce3f5625cffa78d94f0a5c059
data/docker/registry/v2/blobs/sha256/90/9050ace8ca9ce3f5625cffa78d94f0a5c059
data/docker/registry/v2/blobs/sha256/b7/
```

3 调整镜像仓库目录。

正确的镜像仓库目录结构为：镜像仓库根目录/docker/registry/v2/...

以此处为例，正确的目录结构为：/var/images/docker/registry/v2/...

所以，需要把 data 目录下的内容全部移到其平级目录，并把 data 目录删除。如果有多个压缩包加压，执行相同操作即可，如果有同名文件强制覆盖即可。

4 验证镜像仓库。

在客户机执行如下命令：

```
curl –vL http://127.0.0.1:80/v2/_catalog
```

如果能返回镜像列表，即表示镜像仓库安装正常

4.1.7 容器重启出现 endpoint has attached in network

宿主机重启后，或者一些异常下，出现“endpoint has attached in network”。

需要将此容器重网络中 disconnect 掉，然后使用 connect 把原来的 ip 赋给此容器。

4.2 产品常见问题

4.2.1 EDAS

4.2.1.1 EDAS 节点配置

EDAS 节点需要对机器的网络和 IP 有要求，需要通过 hostname –i 命令取到节点的真实服务 IP，以下配置仅供参考。

1 在/etc/hosts 里添加节点名称和 IP 映射。

```
XXX.XXX.XXX.XXX     edas_node1
```

2 在/etc/sysconfig/network 里添加主机名称。

```
HOSTNAME=edas_node1
```

3 重启节点或者临时生效 sysctl kernel.hostname=edas_node1。

5 集群节点扩容指南

5.1 添加节点到方舟底座集群

1 准备好需要添加的节点，保证目标节点满足中间件安装的前提条件。

2 登录方舟客户端，在客户端上运行扩展节点程序，添加节点

```
cd ark/tools
sh expension.sh <ip_list> # 例如 10.1.2.1,10.1.2.3
```

3 扩展脚本运行完后，检查节点已经加入集群，在方舟客户端检查

```
docker ps
```

4 在新添加的节点上检查

```
sudo docker ps
```

5.2 为服务节点添加 Label 标签

方舟集群中管理的节点可以用于部署管控容器或中间件服务容器，例如 drds-server, jingwei-worker 和 mq 服务节点等。添加标签的目的在与通过向节点 docker Engine 的配置文件/etc/docker/daemon.json 添加标注，标注当前节点专用于某一类服务

标签用途	打标签后的/etc/docker/daemon.json 文件示例
管控节点	如维护操作可能造成磁盘丢失，需备份客户端安装程序和镜像仓库
DRDS 的 drds-server 服务节点	{ “labels” :[“nodetype=drds-server” , “ark.ip=10.1.1.1”]}
DRDS 的 jingwei-worker 服务节点	{ “labels” :[“nodetype=jingwei-worker” , “ark.ip=10.1.1.2”]}
MQ 的 broker 服务节点	{ “labels” :[“nodetype=mq” , “ark.ip=10.1.1.2”]}

步骤

1 创建或编辑/etc/docker/daemon.json，并加上 label，示例如下

```
{ “labels” :[ “nodetype=drds-server” , “ark.ip=10.1.1.1” ]}
```

2 重新刷新 docker 服务配置并重启 docker 服务使之生效

```
sudo systemctl daemon-reload
sudo systemctl restart docker.service
```