Uibot部署手册

修订内容有如下几种：

序号	增加（+增加说明）	修改（+修改说明）	使用产品	版本	日期
1	初次编辑		commander5.5 Mage 1.10	v1.0	2021-05-27
2	GPU显卡安装说明		docker1.19.03 k8s 1.18.12	v1.1	2021-08-27

[TOC]

第1章 简介

1.1 来也科技介绍

来也科技是中国乃至全球的 RPA+AI 行业领导者，为客户提供变革性的智能自动化解决方案，提升组织生产力和办公效率，释放员工潜力，助力政企实现智能时代的人机协同。

来也科技 RPA + AI 软件机器人已成功应用于银行、保险、电信、电力、制造、零售、政府、物流、地产、教育和医疗等行业，助力政企在财税、人力资源、法务、客服和营销等场景实现智能化转型。已

服务中国太平、中国移动、中国南方电网、美的、沃尔玛、耐克、北京市海淀园、中通、龙湖、好未来和罗氏等近百家世界 500 强、中国 500 强，数十个省市政府以及上千家中小企业客户

现拥有机器人流程自动化平台“来也UiBot”、智能对话机器人平台“吾来”、全球首个专为 RPA 机器人打造的 AI 能力平台“UiBot Mage”三大核心产品，为政企实现“端到端”的智能自动化。

1.2 依赖组件说明

- **Docker**

Docker 是一个开放源代码软件，是一个开放平台，用于开发应用、交付（shipping）应用、运行应用。 Docker允许用户将基础设施（Infrastructure）中的应用单独分割出来，形成更小的颗粒（容器），从而提高交付软件的速度。Dockers是有能力打包应用程序及其虚拟容器，可以在任何Linux服务器上运行的依赖性工具，这有助于实现灵活性和便携性，应用程序在任何地方都可以运行，无论是公有云，私有云还是单机等。Commander或Mage模块以Docker镜像方式部署并提供服务。

- **Kubernetes**

Kubernetes又简称为k8s，主要是用于自动部署、扩展和管理“容器化应用程序”的开源系统。 它旨在提供“跨主机集群的自动部署、扩展以及运行应用程序容器的平台”。 它支持一系列容器工具，包括Docker等。是一种可自动实施容器操作的开源平台。它可以帮助用户省去应用容器化过程的许多手动部署和扩展操作。也就是说，您可以将运行 Linux 容器的多组主机聚集在一起，由 Kubernetes 帮助您轻松高效地管理这些集群。k8s的目标就是简单并且高效部署容器化应用，可提供应用部署、规划、更新维护等机制。

- **Harbor**

Harbor是一个用于存储和分发Docker镜像的企业级Registry服务器，通过添加一些企业必需的功能特性，例如安全、标识和管理等，扩展了开源Docker Distribution。 作为一个企业级私有Registry服务器，Harbor提供了更好的性能和安全。Docker镜像存储在Harbor服务器上。

- **Istio**

Istio管理服务之间的流量，实施访问政策并汇总遥测数据，而不需要更改应用代码。Istio 以透明的方式对现有分布式应用进行分层，从而简化了部署复杂性。网络运营商可以通过一致的方式管理其所有服务的网络，而不会增加开发者开销。实现 Canary 版发布等最佳做法并深入了解应用，以便确定应该集中关注何处以提升性能

- **其它**

如Mysql，Redis，Rabbitmq，Nginx，Minio等均集成部署在服务器中

1.3 名词解释

Commander：指挥官、流程管理系统。管理运行机器人，分配运行任务。

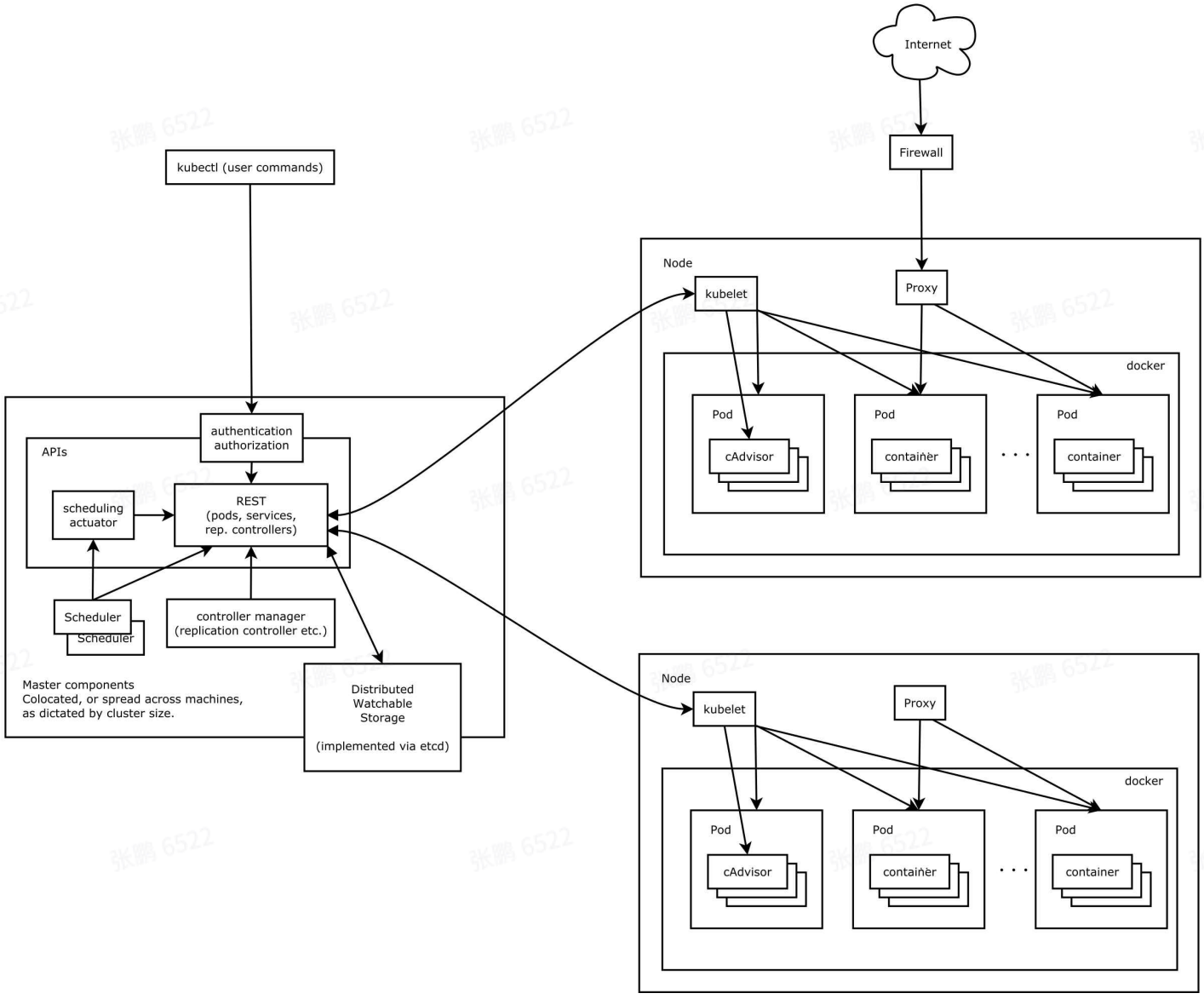
Mage：魔法师，可给Worker提供Ai能力

Creator：又称为设计器，主要作用是设计、运行、控制发布流程包

Worker：又称为运行器，主要作用是执行任务

1.4 架构图

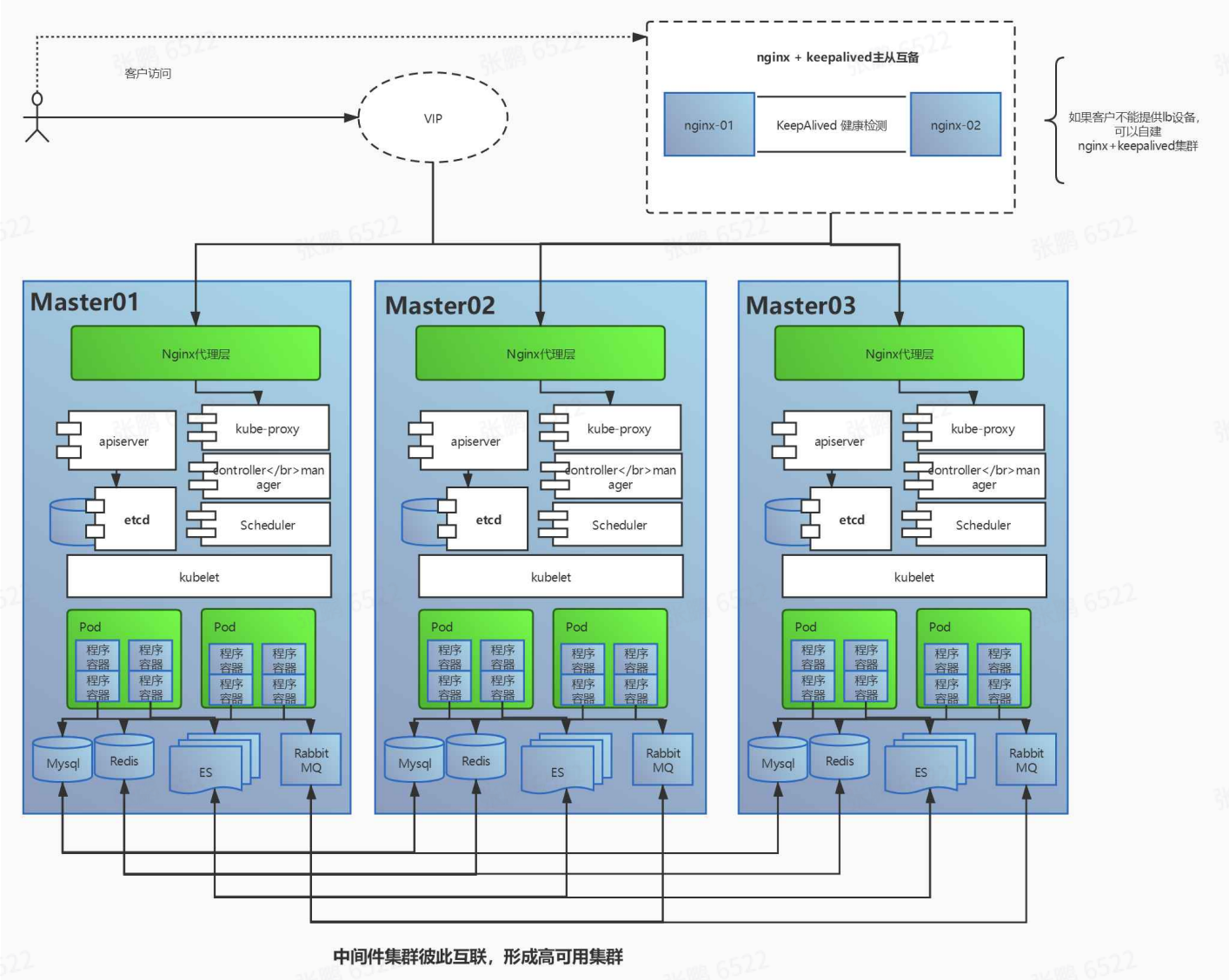
📌 kubernetes架构图

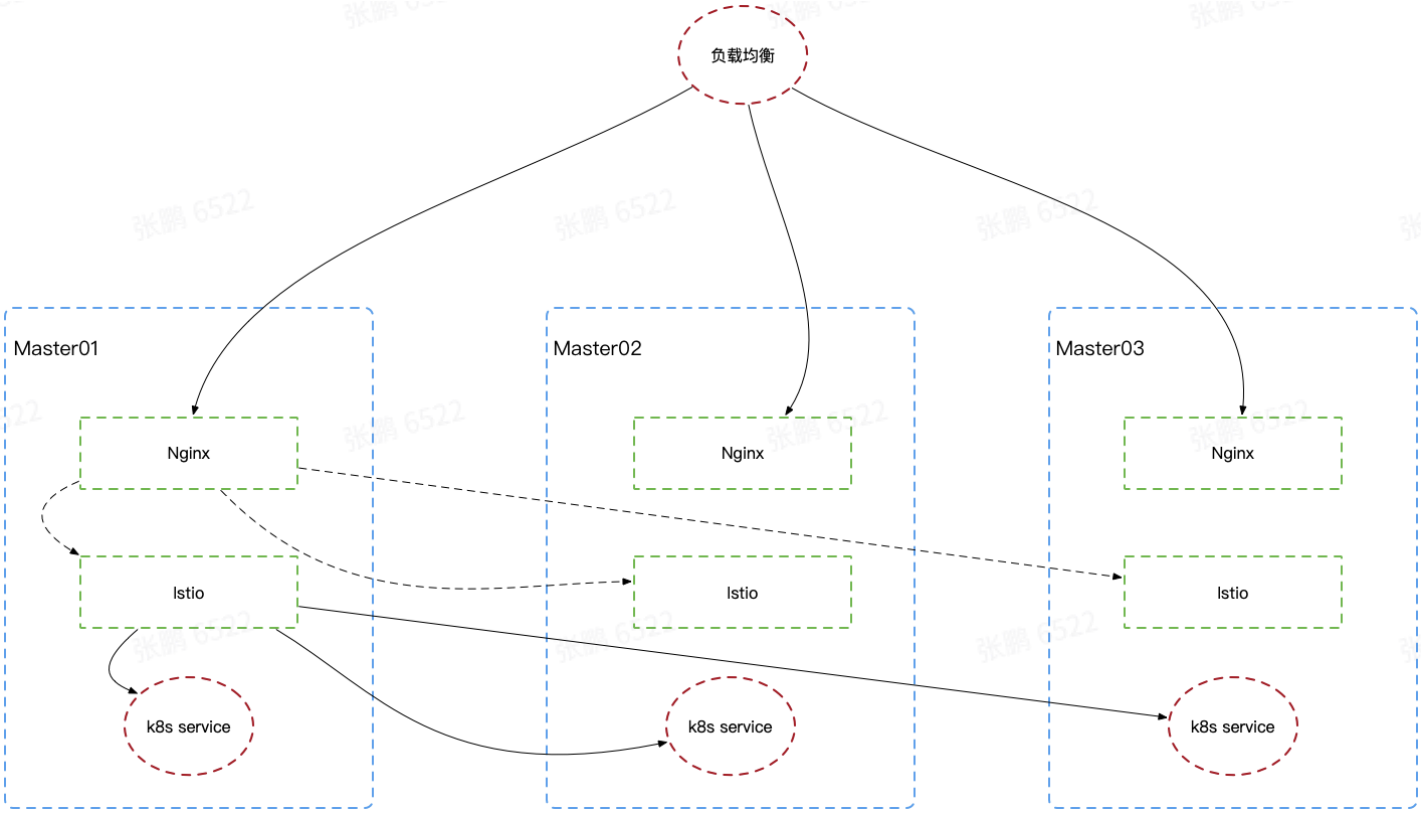


📌 单机架构图

📌 多机架构图

私有部署底层使用kubernetes的架构，主要通过部署三节点master的方案来实现一个高可用的kubernetes集群。在通过k8s内部的可用性机制来保证在集群内部运行服务的高可用。关于kubernetes高可用的文档，可以参考文档https://kubernetes.io/zh/docs/tasks/administer-cluster/highly-available-master/ 在私有部署环境中，中间件服务会部署在机器角色为Master的服务器上。私有部署架构则使用官方推荐的架构。具体细节如下（VIP地址指的是负载均衡，负载均衡后端服务器为三台master机器。如果客户无法提供负载均衡，需要客户提供2台2c2g40G的机器和一个vip ip地址，我们利用keepalived+nginx帮助搭建一套软负载。）





1.5 kubernetes组件详细说明

在部署环境中，我们使用了其他工具为快速部署提供技术上的实现方式，主要有helm和harbor。helm是kubernetes环境中的一个包管理工具，主要用于维护一个服务在kubernetes环境中使用到的配置，并通过替换变量的方式来适配不同的部署环境。例如通过替换镜像的tag来完成服务的升级、通过替换namespace来部署不同环境的服务。 harbor是docker的镜像仓库。在私有部署过程中，我们会将提前准备好的服务镜像数据打包，并导入到harbor仓库中，以便在不同的服务器上，都可以通过harbor来获取需要的镜像。保证在扩容、服务宕机自动迁移时，都可以快速启动。避免了人工上传、下载镜像的过程。

作为私有部署环境，可扩展性是一个独有的特点。我们可以通过扩容kubernetes的node节点来实现对环境的快速扩容。 适用于扩容的场景：

- 用户量、访问量的增长
- 增加新的部署模块
- 当前服务器配置不满足现有的需求

扩容流程：

- 1,新建服务器，并部署好docker、kubelet
- 2,在现有的master节点，通过kubeadm命令生成加入集群的命令
- 3,在新建的服务器上执行命令加入集群
- 4,驱逐现有的pod到新服务器或扩容现有的pod

etcd

etcd 是兼具一致性和高可用性的键值数据库，用于存储K8S中资源对象，如集群中Pod,Node等状态以及配置数据等。生产中需要对etcd进行高可用部署，通常为三个节点到七个节点之间的奇数节点，并且需要定期进行数据备份。etcd各个节点使用raft共识算法进行数据复制。当节点数量过多会导致仲裁和数据复制的效率降低！

在kubernetes集群中，仅apiserver会和etcd集群进行通信。在k8s集群中，etcd的性能至关重要！

master组件

master和node是两个逻辑上节点，当服务器资源充足时，可以将其分开在不同的机器上部署，当服务器资源不足时，也可以放到同一台机器上部署。

apiserver

主节点上负责提供 Kubernetes API 服务的组件；它是 Kubernetes 控制面的前端。是整个集群中资源操作的唯一入口，并提供认证、授权、访问控制、API注册和发现等机制。

apiserver提供了集群管理的restful api接口(鉴权、数据校验、集群变更等)，负责和其它模块之间进行数据交互，承担了通信枢纽功能。

apiserver是无状态的，可以横向扩展，当存在多个apiserver时，前端会采用LB作为负载均衡，如Nginx,LVS,ELB等

controller manager

controller manager 译为“控制器管理器”，k8s内部有很多资源控制器，比如：Node Controller、Replication Controller、Deployment Controller、Job Controller、Endpoints Controller等等，为了降低复杂度，将这些控制切都编译成了一个可执行文件，并且在同一个进程中运行。controller manager 负责维护集群的状态，比如故障检测、自动扩展、滚动更新等。

controller-manager 虽然允许同时运行多个实例，但是只有一个能成为leader，并处理控制器的请求。controller-manager 锁对象存储在 kube-system 名称空间下的 kube-controller-manager endpoints对象中。

scheduler

调度器组件监视那些新创建的未指定运行节点的 Pod，并选择节点让 Pod 在上面运行。调度决策考虑的因素包括单个 Pod 和 Pod 集合的资源需求、硬件/软件/策略约束、亲和性和反亲和性规范、数据位置、工作负载间的干扰和最后时限。

当前各个node节点资源会通过kubelet汇总到etcd中，当用户提交创建pod请求时，controller manager监听apiserver的pod事件并处理。此时scheduler通过etcd中存储的node信息进行预选(predict)，将符合要求的node选出来。再根据优选(priorities)策略，最后执行Pod与Node绑定。

scheduler并不是每次都对所有的node进行遍历筛选，而是根据参数 `percentOfNodeToScore` 的值确定需要检查多少个node状态，0表示未设置。如果节点小于50，该参数不生效；超过100个节点集群，按一定的百分比选取参与调度的node节点。

scheduler 与 controller-manager 一致，也是主备模式，只有一个leader处理调度。

node组件

kubelet

一个在集群中每个节点上运行的代理，kubelet 接收一组通过各类机制提供给它的 PodSpecs，确保这些 PodSpecs 中描述的容器处于运行状态且健康。kubelet 不会管理非 Kubernetes 创建的容器。

简单来说主要是三个功能：

- 接收pod的期望状态(副本数、镜像、网络等)，并调用容器运行环境(container runtime)来实现预期状态，目前container runtime基本都是docker ce。需要注意的是，**pod网络是由kubelet管理的，而不是kube-proxy**。
- 定时汇报节点的状态给 apiserver，作为scheduler调度的基础
- 对镜像和容器的清理工作，避免不必要的文件资源占用磁盘空间

当Pod被调度到该节点后：

- Kubelet会将申请的volume挂载到当前节点上，
- 创建沙箱容器(podsand box，pause容器)，将自己阻塞(pause系统调用)
- 基于pause的网络名称空间，创建业务容器

kube-proxy

kube-proxy 是集群中每个节点上运行的网络代理，是实现service资源功能组件之一。kube-proxy 建立了pod网络和集群网络之间的关系，即 cluster ip 和 pod ip 中间的关系。不同node上的service流量转发规则会通过kube-proxy进行更新，其实是调用apiserver访问etcd进行规则更新。

service流量调度方式有三种方式: userspace(废弃，性能很差)、iptables(性能差，复杂)、ipvs(性能好，转发方式清晰)。

container runtime

即容器运行环境，Kubernetes 支持多个容器运行环境: Docker、 containerd、cri-o、 rktlet 以及任何实现 Kubernetes CRI (容器运行环境接口)。常用的是Docker CE

核心附件

4.1 CNI网络插件

Kubernetes设计了网络模型，但是却将具体实现方式交给了网络插件来实现，CNI网络插件实现的是pod之间跨宿主机进行通信。默认情况下安装好node节点后，无法跨node进行宿主机通信。可以参考: https://www.yuque.com/duduniao/ww8pmw/tr3hch#GtBRc

CNI网络插件有很多，比如：Flannel、Calico、Canal、Contiv、OpenContrail等，其中最常用的是Flannel和Calico两种。Flannel在node节点比较少，不跨网段时性能比较好，也是市场占有率最高的网络插件

私有化部署产品自带的集成CNI插件为flannel。

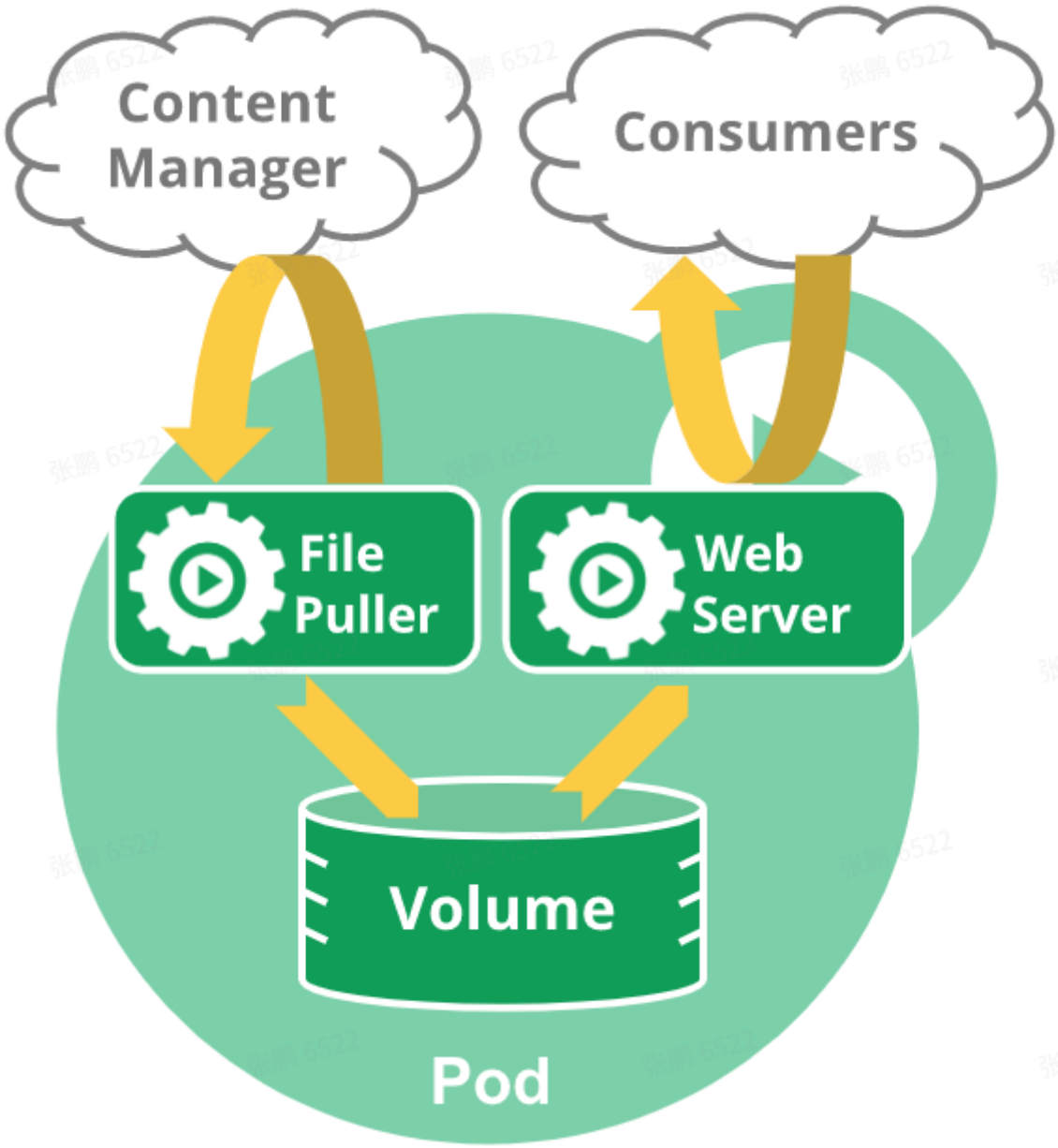
Plain Text

```
[root@pvt-test-commander-01 ~]# kubectl get po -A
NAMESPACE      NAME                                    READY   STATUS
RESTARTS   AGE
istio-system   istio-ingressgateway-766cd89bbb-8vxvw   1/1     Running   2
2d5h
istio-system   istio-ingressgateway-766cd89bbb-qhsr2   1/1     Running   2
2d5h
istio-system   istiod-9f76c5d9c-ghj4q                  1/1     Running   3
2d5h
istio-system   istiod-9f76c5d9c-kd42v                  1/1     Running   2
2d5h
kube-system    coredns-7d9985555d-95fsn                1/1     Running   2
2d5h
kube-system    coredns-7d9985555d-klsqz                1/1     Running   2
2d5h
kube-system    etcd-master-01                          1/1     Running   2
2d5h
kube-system    kube-apiserver-master-01                1/1     Running   2
2d5h
kube-system    kube-controller-manager-master-01       1/1     Running   3
2d5h
kube-system    kube-flannel-ds-ns2r5                   1/1     Running   2
2d5h
kube-system    kube-proxy-vfc97                        1/1     Running   2
2d5h
kube-system    kube-scheduler-master-01                1/1     Running   3
2d5h
```

1.6 模块服务详细说明

Pod 是 Kubernetes 的基本构建块，它是 Kubernetes 对象模型中创建或部署的最小和最简单的单元。Pod 表示集群上正在运行的进程。Pod 封装了应用程序容器（或者在某些情况下封装多个容器）、存储资源、唯一网络 IP 以及控制容器应该如何运行的选项。 Pod 表示部署单元：*Kubernetes 中应用程序的单个实例*，它可能由单个容器或少量紧密耦合并共享资源的容器组成。一个pod内部一般仅运行一个pod，也可以运行多个pod，如果存在多个pod时，其中一个为主容器，其它作为辅助容器，也被

称为边车模式。同一个pod共享一个网络名称空间和外部存储卷。**私有化部署使用K8S管理Commander和Mage各模块，会在k8s集群中创建多个namespac**e。具体功能如下：



✅ **namespace说明**

namespace	含义
rpa	Commander服务的namespace
mage	Mage服务的namespace
kube-system	Kubernetes系统的namespace

mid	服务授权的namespace
istio-system	istio服务的namespace

✅ Pod说明

以下给出的是所有的Pod的说明，实际部署过程中会根据部署的场景和功能，pod会有所删减。

kube-system

pod名	**含义**
Coredns-xxx	内部dns服务
kube-apiserver-master-xxx	Master组件
kube-controller-manager-master-xxx	
etcd-master-xx	
kube-proxy-xx	
Kube-scheduler-master-xx	

rpa

Pod名	**含义**
host-scheduler-xxx	定时计划调度、定时任务触发
view-commander-xxx	commander前端页面
view-global-ng-xxx	租户前端页面
webapi-commander-xxx	commande主要http接口（webhttp接口，creator，worker登录授权）
webapi-global-xxx	全局Api模块（租户，license，系统配置，租户配置）

webapi-open-xxx	对外提供openapi接口
websocket-creator-xxx	Creator websocket接入
host-mqsubscriber-xxx	消息处理（定时计划任务创建，worker，creator在线维护）
websocket-vnc-xxx	worker实时监控
websocket-worker-xxx	Worker websocket接入

Mage

Pod名	含义
bert-service-tf-xxx	
doc-classifier-xxx	
doc-classifier-trainer-xxx	
document-mining-auth-xxx	
document-mining-innerservice-xxx	
document-mining-openapi-xxx	
document-mining-rpc-xxx	
file-analyze-xxx	
info-engine-xxx	
mage-entitynormal-xxx	
ocr-ctpn-server-xxx	
ocr-ctpn-tf-server-xxx	
ocr-inner-tool-xxx	
ocr-server-dispatch-xxx	
ocr-text-recognition-server-xxx	

ocr-text-recognition-tf-server-xxx	
poi-search-engine-xxx	
poi-text-match-xxx	

mid

Pod名	含义
license-manager	

istio-system

Pod名	含义
istio-ingressgateway-xxxx	
Istiod-xxx	

1.7 端口使用说明

以下端口为在不修改任何端口的情况下启用的默认端口，根据部署功能不同，端口有所增减可能。

端口	描述
2379/12379/2380/12380	etcd
8888/1514	harbor
6060/6061	Identity
9000/10000	Minio

3306/6033/6034	Mysql
80	Commander
81	租户系统
82	mage
6444	kubelet
4369/5672/15672/25672	rabbitmq
6379/26379	redis
88	siber
44134/44135	Tiller
10257	kube-controlle
10259	kube-scheduler
30000-33000	ocr的应用端口
3000/9090/9104/9121/9419/9114/9100/31388/31390	monitor备用端口

其中,6061,80,81,82,10000需对外开放

✅ 负载均衡端口说明

负载均衡端口对外开放端口如下：可根据实际端口情况进行更改

注意：负载均衡一定要部署前创建好负载规则，否则会导致部署失败。

lb端口	后端端口	服务说明
6061	6061	identity的端口，登录时会跳转到此端口
80	80	commander的默认端口

81	81	租户系统的默认端口
82	82	mage的默认端口
10000	10000	minio的端口

1.8 各服务版本说明

以下版本为部署模块包中自带的软件版本，可供参考

组件	**版本**	
k8s	1.18.12	
docker	19.03.14	
Istio	5	
flannel	0.13.1	
coredns	7	
redis	5.0.10	
mysql	5.7.32	
minio	2020-01-16	
etcd	3-0	
nginx	1.11.12	
harbor	2.0.2	
rabbitmq	3.0	

第2章 安装要求

2.1 操作系统要求

服务端支持的服务器操作系统主要为Centos或Redhat系统。

- Centos7.x
- Redhat7.x

推荐安装的操作系统为Centos7.8，内核版本为3.10.x。

注意：

- 操作系统内核版本

可使用以下命令获取服务器操作系统信息:

- 检查服务器操作系统版本（只限Redhat系列）

Plain Text

```
[root@localhost ~]# cat /etc/redhat-release
CentOS Linux release 7.9.2009 (Core)
```

- 检查操作系统内核版本

Plain Text

```
[root@localhost ~]# uname -r
3.10.0-1160.el7.x86_64
```

- 检查操作系统硬件架构

Plain Text

```
[root@localhost ~]# uname -m
x86_64
```

- 查看磁盘大小

Plain Text

```
[root@localhost ~]# df -hT /home
文件系统                类型   容量   已用   可用 已用% 挂载点
/dev/mapper/centos-root xfs    198G   0G    198G    0% /
```

- 查看内存大小

Plain Text

```
[root@localhost ~]# free -h
              total        used        free      shared  buff/cache   available
Mem:           972M        270M        283M         19M        418M        534M
Swap:            0B          0B          0B
```

- 查看cpu核数

Plain Text

```
[root@localhost ~]# grep processor /proc/cpuinfo|wc -l
1
```

2.2 系统硬件配置要求

目前私有部署都需要基于k8s部署，所以硬件要求较比之前有所提高。由于用户需求模块不同，导致硬件配置要求也不一致。可根据具体部署模块并利用以下公式进行推算。**生产环境推荐使用推荐配置**

单机配置要求

产品名称	最低要求（CPU/内存/磁盘）	推荐配置（CPU/内存/磁盘）	显卡
Uibot Commander	6C 12G 100G	8C 16G 200G	-
Uibot Mage	6C 12G 100G	8C 16G 200G	-
Captcha 验证码识别	1C 2G	2C 4G	-
nlp 信息抽取，标准地址	1C 2G	2C 4G	-
OCR 通用文字识别（标准版CPU）	4C 8G	8C 16G	-
OCR 通用文字识别（标准版GPU）	-	-	12G
OCR 通用文字识别（高级版CPU）	2C 4G	4C 8G	-
OCR 通用文字识别（高级版GPU）	-	-	12G

OCR 通用表格识别（高级版CPU）	2C 4G	4C 8G	-
OCR 通用表格识别（高级版GPU）	-	-	11G
OCR 通用票据识别（高级版CPU）	2C 4G	4C 8G	-
OCR 通用票据识别（高级版GPU）	-	-	11G
OCR 通用卡证识别-身份证（高级版CPU）	2C 4G	4C 8G	
OCR 通用卡证识别-身份证（高级版GPU）	-	-	11G
OCR 通用卡证识别-银行卡（高级版CPU）	2C 4G	4C 8G	
OCR 通用卡证识别-银行卡（高级版GPU）	-	-	11G
OCR 通用卡证识别-机动车登记证（高级版CPU）	2C 4G	4C 8G	
OCR 通用卡证识别-机动车登记证（高级版GPU）	-	-	11G
OCR 通用卡证识别-行驶证（高级版CPU）	2C 4G	4C 8G	
OCR 通用卡证识别-行驶证（高级版GPU）	-	-	11G
OCR 通用卡证识别-营业执照（高级版CPU）	2C 4G	4C 8G	
OCR 通用卡证识别-营业执照（高级版GPU）	-	-	11G
OCR 通用卡证识别-护照（高级版CPU）	2C 4G	4C 8G	
OCR 通用卡证识别-护照	-	-	11G

（高级版GPU）			

多机配置要求

产品名称	节点最低要求	推荐配置（CPU/内存/磁盘）	显卡	Master最低要求（CPU/内存/磁盘）	Node最低要求（CPU/内存/磁盘）
Uibot Commander	3	8C 16G 200G	-	4C 8G 50G	2C 4G 50G
Uibot Mage	3	8C 16G 200G	-	4C 8G 50G	2C 4G 50G
Captcha 验证码识别	3	2C 4G	-	1C 2G	-
nlp 标准地址、信息抽取	3	2C 4G	-	1C 2G	-
OCR 通用文字识别（标准版 CPU）	3	8C 16G	-	4C 8G	-
OCR 通用文字识别（标准版 GPU）	3	-	12G	-	-
OCR 通用文字识别（高级版 CPU）	3	4C 8G	-	2C 4G	-
OCR 通用文字识别（高级版 GPU）	3	-	12G	-	-
OCR 通用表格识别（高级版 CPU）	3	4C 8G	-	2C 4G	-
OCR 通用表格识别（高级版 GPU）	3	-	11G	-	-

OCR 通用票据识别（高级版CPU）	3	4C 8G	-	2C 4G	-
OCR 通用票据识别（高级版GPU）	3	-	11G	-	-
OCR 通用卡证识别-身份证（高级版CPU）	3	4C 8G		2C 4G	-
OCR 通用卡证识别-身份证（高级版GPU）	3	-	11G	-	-
OCR 通用卡证识别-银行卡（高级版CPU）	3	4C 8G		2C 4G	-
OCR 通用卡证识别-银行卡（高级版GPU）	3	-	11G	-	-
OCR 通用卡证识别-机动车登记证（高级版CPU）	3	4C 8G		2C 4G	-
OCR 通用卡证识别-机动车登记证（高级版GPU）	3	-	11G	-	-
OCR 通用卡证识别-行驶证（高级版CPU）	3	4C 8G		2C 4G	-
OCR 通用卡证识别-行驶证（高级版GPU）	3	-	11G	-	-
OCR 通用卡证识别-营业执照（高级版CPU）	3	4C 8G		2C 4G	-
OCR 通用卡证识	3	-	11G	-	-

别-营业执照（高级版GPU）					
OCR 通用卡证识别-护照（高级版CPU）	3	4C 8G		2C 4G	-
OCR 通用卡证识别-护照（高级版GPU）	3	-	11G	-	-

中间件所需配置

- 由于中间件安装在Master上, Master配置中已经包含了中间件的配置, 不需要单独计算。
- 并且Master已经包含了一个基础模块的配置(Mage, Commander其中任意一个)。

配置计算

- 私有部署是基于kubernetes环境来实现, 环境配置是以机器角色(master,node)的方式来规划的。
- master：部署k8s基础服务,比如:apiserver、kube-controller、kube-scheduler以及中间件服务
- node：实际运行服务模块的节点
- 如果是多节点方式部署需要满足Master = 3,Node >= 3。 前置检查会根据多节点数量,分成以下两种情况
  - 3 =< 服务器数量 <= 5, master = 3; Node = 服务器数量
  - 6 =< 服务器数量, master = 3; Node = 服务器数量 - 3
- 通用计算公式:
  - cpu = 机器角色Master的cpu数量 + 部署功能模块2的cpu数量 + ... + 部署功能模块n的cpu数量
  - 内存 = 机器角色Master的内存大小 + 部署功能模块内存的内存大小 + ... + 部署功能模块n的内存大小
  - 数据盘 = 机器角色,功能模块所需的数据盘,取最大值
- 举个例子（以下计算方式为推荐配置计算）
  - 客户准备了1台机器,希望进行私有部署mage + 文本分类 + 验证码

Plain Text

```
    根据公式计算:
        cpu =  8(master) + 4(mage) + 4(文本分类) + 2(验证码) = 18(逻辑核)
        内存 = 16(master) + 8(mage) + 8(文本分类) + 4(验证码) = 36(G)
        硬盘 = 200G
    结论:
        1台机器,需要满足cpu大于等于18核,内存大于等于36G,硬盘大于等于200G
```

- 客户准备了3台机器,希望进行私有部署mage + 信息抽取 + 通用文字识别标准版

Plain Text

```
    根据公式计算:
        cpu =  8(master) + 4(mage) + 4(信息抽取) + 16(通用文字识别标准版) = 32(逻辑核)
        内存 = 16(master) + 8(mage) + 8(信息抽取) + 32(通用文字识别标准版) = 64(G)
        硬盘 = 200G
    结论:
        3台机器,每台都需要满足cpu大于等于32核,内存大于等于64G,硬盘大于等于200G
```

- 客户准备了4台机器,希望进行私有部署mage + commander + 验证码

Plain Text

```
    分析:
        根据前置检查分析结果, 会分成3个master,4个node
    根据公式计算:
        master配置:
          cpu =  8(master) + 4(mage) +4(commander) + 2(验证码) = 18(逻辑核)
          内存 = 16(master) + 8(mage) + 8(commander) + 4(验证码) = 36(G)
          硬盘 = 200G
        node配置:
          cpu =  4(mage) +4(commander) + 2(验证码) = 10(逻辑核)
          内存 = 8(mage) + 8(commander) + 4(验证码) = 20(G)
          硬盘 = 200G

    结论:
        3台机器,每台都需要满足cpu大于等于18核,内存大于等于36G,硬盘大于等于200G
        1台机器,每台都需要满足cpu大于等于10核,内存大于等于20G,硬盘大于等于200G
```

- 客户准备了6台机器,希望进行私有部署mage + 标准地址 + 通用文字识别标准版

Plain Text

```
根据公式计算：
    master配置：
        cpu = 8(master)
        内存= 16(master)
        硬盘= 200G(master)
    node配置：
        cpu = 4(mage) + 4(标准地址) + 16(通用文字识别标准版) = 24(逻辑核)
        内存= 8(mage) + 8(标准地址) + 32(通用文字识别标准版) = 48(G)
        硬盘 = 200G
结论：
    3台机器master配置需要满足cpu大于等于8,内存大于等于16G,硬盘大于等于200G;
    3台机器node配置需要满足cpu大于等于24,内存大于等于48G,硬盘大于等于200G;
```

2.3 网络环境要求

- 要求服务器的IP地址为固定IP地址
- 如果服务器IP属于172.1x.x.x网段，需要修改docker网段的ip。否则容易引起网络冲突，导致服务无法连接
- 防火墙配置完成或关闭防火墙，具体端口含义可关注端口说明
- 如部署高可用，需保证后端服务器之间通信正常。

2.4 其它要求

- 要求使用root权限进行部署，部署完成后可修改root密码，然后使用普通用户进行管理。
- 关闭swap，禁用swap开机启用
- 关闭selinux，关闭防火墙。如防火墙为客户硬性要求，可请求技术支持协助
- 服务器系统之间需要配置好时间同步，误差不得高于30秒
- 部署分为单机部署和多机部署。多机部署要求服务器数量不低于3台。这是因为k8s master部分要支持高可用的功能，依赖于etcd组件的高可用，而etcd组件的高可用至少需要三节点
- 负载均衡(lb)一定要在部署前搭建完成，否则会导致部署失败。

2.4 运维用户权限设置

由于部署服务端需要部署k8s，所以需要提供root账号进行部署。可在部署成功后修改root账号密码。用普通用户进行后续维护管理。进行以下设置可实现普通用户管理维护。本实例中xxx代表用户名，可根据用户实际情况更改

- 将维护用户加入到/etc/sudoers

Shell

```
vim /etc/sudoers
# 在以下文件中加入一行
xxx ALL=(ALL) NOPASSWD: ALL
```

- 将维护用户加入到docker组

Shell

```
usermod -G docker xxx
```

- 将root下的.kube目录拷贝到维护用户的家目录下

Shell

```
mkdir /home/xxx/.kube
cp -r /root/.kube/config /home/xxx/.kube/
chown xxx:xxx /home/xxx/.kube/config
```

- 将部署harbor的目录下的env文件属主修改为维护用户

Plain Text

```
find /home/harbor/ -name env |xargs chown xxx
```

- 启用维护用户可开机systemctl管理权限，xxx用户可以使用sudo来管理这些服务

Plain Text

```
vim /etc/sudoers.d/xxx
%xxx ALL= NOPASSWD: /usr/bin/systemctl start kubelet
%xxx ALL= NOPASSWD: /usr/bin/systemctl stop kubelet
%xxx ALL= NOPASSWD: /usr/bin/systemctl start harbor
%xxx ALL= NOPASSWD: /usr/bin/systemctl stop harbor
%xxx ALL= NOPASSWD: /usr/bin/systemctl start tiller
%xxx ALL= NOPASSWD: /usr/bin/systemctl stop tiller
%xxx ALL= NOPASSWD: /usr/bin/systemctl start docker
%xxx ALL= NOPASSWD: /usr/bin/systemctl stop docker
```

第3章 系统安装与配置

3.1 部署前准备工作

- 需要确保selinux为关闭状态(Disabled)

Shell

```
# 查看selinux状态
[root@pvt-test-commander-01 laipvt]# getenforce
Disabled
# 如果不是Disabled，需要关闭selinux
[root@pvt-test-commander-01 laipvt]# setenforce 0
```

- 关闭防火墙

Shell

```
[root@pvt-test-commander-01 laipvt]# systemctl stop firewalld
[root@pvt-test-commander-01 laipvt]# systemctl disable firewalld
```

- 如果多机部署，需要检查服务器之间时间是否正确，误差不能大于30秒

Shell

```
[root@pvt-test-commander-01 laipvt]# date
```

- 如果部署mage类型为GPU，需要检查显卡驱动是否正常安装（要求cuda版本不能高于10.2）

Shell

```
[root@pvt-test-commander-01 laipvt]# nvidia-smi
```

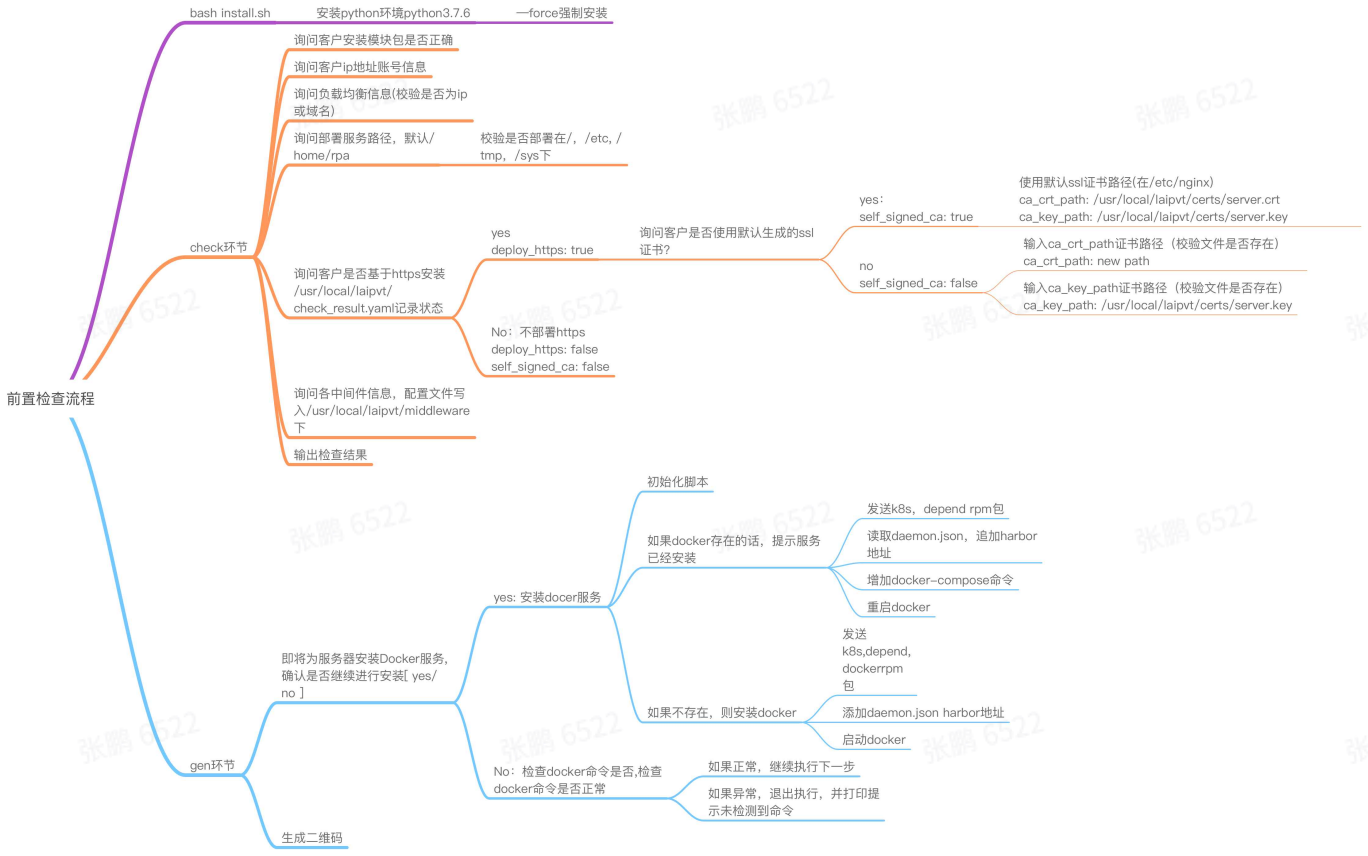
- 检查磁盘是否正确开机挂载。

Shell

```
[root@pvt-test-commander-01 laipvt]# cat /etc/fstab

#
# /etc/fstab
# Created by anaconda on Tue Apr 20 03:55:28 2021
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
UUID=3c211154-bab2-45c8-8cd5-ceea39407cec /                       ext4
defaults        1 1
```

前置检查流程图



3.2 执行前置检查

前置检查包可以判断服务器是否满足需要部署模块所需要的最小配置,将后续部署问题扼杀在摇篮中,保证部署成功率。前置检查通过后，才可以进行Commander或mage安装操作

客户可在内网中通过U盘、sftp、scp等传输工具将从来也人员处获取的前置检查包**checker-xxxx.tar.gz的压缩包**上传至服务器中，将其放入到要部署的目录下。在服务器有网络的情况下也可直接通过wget -c下载。此处用wget命令演示

- 下载前置检查包

Shell

```
[root@pvt-test-commander-01 laipvt]# wget -c
 http://private-deploy.oss-cn-beijing.aliyuncs.com/laipvt-test/check_packages/20
```

- 解压前置检查包

Shell

```
[root@pvt-test-commander-01 ~]# tar xvf checker-z0Lbrkw2j.tar.gz
```

- 进入到前置检查解压后的目录下,安装python环境

Shell

```
[root@pvt-test-commander-01 ~]# cd checker-z0Lbrkw2j
[root@pvt-test-commander-01 checker-z0Lbrkw2j]# bash install.sh
[2021-05-28 19:43:25] - [INFO]: Install Python3 env.
Python Path: /usr/local/python 3.7.6/bin/python3
Python Version: Python 3.7.6
[2021-05-28 19:43:25] - [INFO]: Install Succeed
```

- 开始执行前置检查

Shell

```
[root@pvt-test-commander-01 checker-z0Lbrkw2j]# /usr/local/python-
3.7.6/bin/python3 run.py --check

 _____                  __               _____          ____  ____
_________
|_   _|                /  \             |_   _|        |_  _||_  _|        |_
___  |
  | |                 / /\ \              | |            \ \  / /            |
|_  \_|
  | |   _            / ____ \             | |             \ \/ /             |
_|  _
 _| |__/ |         _/ /    \ \_          _| |_             _|  |_            _|
```

```
|___/ |
|________|        |____|  |____|        |_____|          |______|
|_________|

请根据提示完成前置检查操作
您此次选择部署的模块有:
# 这里的模块跟申请时选择的产品有关
  UiBot Commander
请问是否继续(yes/no): yes


请输入服务器的ip地址: 172.16.69.130
# 输入服务器的ip
请输入用于登录服务器 172.16.69.130 的用户: root
请输入用于登录服务器 172.16.69.130 的密码: ******
请输入用于登录服务器 172.16.69.130 的端口号: 22
# 输入ssh端口号
请问是否要继续增加服务器(yes/no): no
# 如果高可用部署的话，继续yes增加服务器
请输入负载均衡服务的ip地址: 172.16.69.130
# 单机与服务器ip一致，高可用的话填实际lb地址
请输入要部署Docker的路径(默认为 /home/rpa):
# 部署服务的具体路径
是否需要修改中间件mysql的配置(yes/no): no
# 选yes可根据实际情况更改
是否需要修改中间件minio的配置(yes/no): no
是否需要修改中间件redis的配置(yes/no): no
是否需要修改中间件etcd的配置(yes/no): no
是否需要修改中间件rabbitmq的配置(yes/no): no
是否需要修改中间件elasticsearch的配置(yes/no): no
是否需要修改中间件identity的配置(yes/no): no
是否需要修改中间件nginx的配置(yes/no): no
是否需要修改中间件harbor的配置(yes/no): no
是否需要修改中间件siber的配置(yes/no): no
是否需要修改中间件monitor的配置(yes/no): no
是否需要修改中间件ocr的配置(yes/no): no
是否需要修改中间件keepalived的配置(yes/no): no
以下为前置检查结果，请拍照上传到私有部署授权平台
本次要部署的服务包括: commander
服务器检查结果:
172.16.69.130: 不通过
        硬盘centos-root(单位GB)检查失败，要求值：200，实际值：37。检查命令'df -hT |
grep centos-root'
        内存(单位GB)检查失败，要求值：24，实际值：1。检查命令'free -g'
        CPU(单位个)检查失败，要求值：12，实际值：1。检查命令'lscpu'
```

```
CPU(单位个)检查失败，要求值：12，实际值：1。检查命令：lscpu
检查结果：不通过

检查未通过，请联系技术支持！机器配置如下：
 --------------- --------- -------------- -------------- ---------- ------------ ------------ ----------------------------------
      IP地址       Cpu个数    内存大小(GB)    硬盘大小(GB)    操作系统    防火墙状态    Selinux状态                机器角色
 --------------- --------- -------------- -------------- ---------- ------------ ------------ ----------------------------------
  172.16.69.130       1           1              37        centos 7    False      False      master, node, harbor, licserver
 --------------- --------- -------------- -------------- ---------- ------------ ------------ ----------------------------------
```

注意：以上为检查不通过的输出，需要根据提示情况进行修改提升硬件配置，如果检查通过的话，方可进行部署，以下为检查通过的提示信息

Plain Text

```
服务器检查结果：
172.16.69.130: 通过
检查结果：通过

机器配置如下：
 --------------- --------- -------------- -------------- ---------- ------------ ------------ ----------------------------------
      IP地址       Cpu个数    内存大小(GB)    硬盘大小(GB)    操作系统    防火墙状态    Selinux状态                机器角色
 --------------- --------- -------------- -------------- ---------- ------------ ------------ ----------------------------------
  172.16.69.130      16           32             300        centos 7     False      False      master, node, harbor, licserver
 --------------- --------- -------------- -------------- ---------- ------------ ------------ ----------------------------------
```

- 此步骤会安装Docker和K8S包，执行以下步骤会生成机器二维码

提示：由于前置检查是按推荐配置计算的，可按最低配置计算。可在以下步骤中添加--force参数强制进行安装

Shell

```
[root@pvt-test-commander-01 checker-z0Lbrkw2j]# /usr/local/python-3.7.6/bin/python3 run.py --gen --force
请根据提示完成配置，获取机器码
即将为服务器 172.16.69.130 安装docker服务，确认是否要继续(yes/no): yes

请拍照保存当前机器(172.16.69.130)的二维码
访问 https://pvtadmin.wul.ai 申请授权
```

- 将上图的前置检查结果和二维码发送给来也人员，用来获取部署的服务模块包
- 检查docker是否安装成功

Shell

```
[root@pvt-test-commander-01 checker-z0Lbrkw2j]# docker ps
Client: Docker Engine - Community
 Version:           19.03.14
 API version:       1.40
 Go version:        go1.13.15
 Git commit:        5eb3275d40
 Built:             Tue Dec  1 19:20:42 2020
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.14
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       5eb3275d40
  Built:            Tue Dec  1 19:19:17 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.3.9
  GitCommit:        ea765aba0d05254012b0b9e595e995c09186427f
 runc:
  Version:          1.0.0-rc10
  GitCommit:        dc9208a3303feef5b3839f4323d9beb36df0a9dd
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```

- 检查k8s是否安装成功

Plain Text

```
[root@pvt-test-commander-01 checker-z0Lbrkw2j]# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"18", GitVersion:"v1.18.12",
GitCommit:"7cd5e9086de8ae25d6a1514d0c87bac67ca4a481", GitTreeState:"archive",
BuildDate:"2020-12-07T06:20:33Z", GoVersion:"go1.13.6", Compiler:"gc",
Platform:"linux/amd64"}
```

3.3 模块包安装部署

客户可在内网中通过U盘、sftp、scp等传输工具将从来也人员处获取的服务模块包**xxxx.tar.gz的压缩包**上传至服务器中，将其放入到要部署的目录下。在服务器有网络的情况下也可直接通过wget -c下载。

- 执行一键部署，使用绝对路径。比如我将包放置在/root目录下

Shell

```
[root@pvt-test-commander-01 checker-z0Lbrkw2j]# /usr/local/python-3.7.6/bin/laipvt -f /root/xxx.tar.gz
[INFO] 2021-05-26 10:31:41,775 util.py [line:252] wrapper: 即将执行basesystem下的unpack方法
[INFO] 2021-05-26 10:31:45,180 util.py [line:252] wrapper: 即将执行basesystem下的kubernetes_unpack方法
[INFO] 2021-05-26 10:32:14,021 util.py [line:252] wrapper: 即将执行basesystem下的install_harbor方法
[INFO] 2021-05-26 10:32:14,021 harbor.py [line:35] _generic_config: 渲染分发 Harbor 安装包及配置文件
[DEBUG] 2021-05-26 10:32:14,021 template.py [line:19] _fill: 渲染模板: /root/z0Lbrkw2j/harbor/config.tmpl到目录: /tmp/harbor.yml
[DEBUG] 2021-05-26 10:32:14,026 template.py [line:19] _fill: 渲染模板: /root/z0Lbrkw2j/harbor/harbor.service.tmpl到目录: /tmp/harbor.service
[INFO] 2021-05-26 10:32:14,027 middlewareinterface.py [line:82] send_config_file: 分发配置文件/root/z0Lbrkw2j/harbor 到 172.17.228.228:/home/rpa/harbor
[INFO] 2021-05-26 10:32:21,175 middlewareinterface.py [line:82] send_config_file: 分发配置文件/tmp/harbor.yml 到 172.17.228.228:/home/rpa/harbor/harbor.yml
[INFO] 2021-05-26 10:32:21,176 middlewareinterface.py [line:82] send_config_file: 分发配置文件/tmp/harbor.service 到 172.17.228.228:/usr/lib/systemd/system/harbor.service
[INFO] 2021-05-26 10:32:21,176 harbor.py [line:49] _install_harbor: 安装harbor
[INFO] 2021-05-26 10:33:55,326 harbor.py [line:70] _start: 启动 172.17.228.228 harbor服务
[INFO] 2021-05-26 10:34:30,023 harbor.py [line:96] _check: 检查 172.17.228.228:8888 上的 Harbor 健康性
[INFO] 2021-05-26 10:34:30,031 harbor.py [line:99] _check: Harbor 服务运行正常
[INFO] 2021-05-26 10:34:30,031 harbor.py [line:110] _change_password: 修改 Harbor 登陆密码
[INFO] 2021-05-26 10:34:30,053 harbor.py [line:118] _change_password: Harbor 密码修改成功
-----------
[INFO] 2021-05-26 10:42:31,948 siber.py [line:43] _send_docker_compose_file: 分发siber docker-compose配置文件至 172.17.228.228
[INFO] 2021-05-26 10:42:32,521 siber.py [line:72] _start: 启动 172.17.228.228:
```

```
    siber服务
20  [INFO] 2021-05-26 10:42:53,144 siber.py [line:77] _start: 启动 172.17.228.228
    siber 服务成功
21  [INFO] 2021-05-26 10:43:13,153 siber.py [line:55] _init_siber: 初始化 siber-
    mongo 数据
22  [INFO] 2021-05-26 10:43:14,191 siber.py [line:64] _init_siber: 初始化 siber-
    mongo 数据成功
23  [INFO] 2021-05-26 10:43:34,211 siber.py [line:90] _check: 机器:172.17.228.228
    Siber-nginx代理端口88访问正常
24  [root@pvt-test-commander-01 checker-z0Lbrkw2j]#
```

以上，**表示服务安装成功**，如果部署过程中，遇到错误或者窗口关闭等异常情况，可在解决问题后继续执行此命令安装

至此，服务安装已经结束，可使用命令查看或者使用网页测试部署是否正常

3.4 验证部署

✅ 通过k8s验证服务是否正常运行

以下状态表示，服务正常部署。绝大情况下，网页可正常登录，功能正常

Shell

```
1   [root@pvt-test-commander-01 checker-z0Lbrkw2j]# kubectl get po -A
2   NAMESPACE      NAME                                    READY   STATUS
    RESTARTS   AGE
3   istio-system   istio-ingressgateway-766cd89bbb-8vxvw   1/1     Running   2
    2d9h
4   istio-system   istio-ingressgateway-766cd89bbb-qhsr2   1/1     Running   2
    2d9h
5   istio-system   istiod-9f76c5d9c-ghj4q                  1/1     Running   3
    2d9h
6   istio-system   istiod-9f76c5d9c-kd42v                  1/1     Running   2
    2d9h
7   kube-system    coredns-7d9985555d-95fsn                1/1     Running   2
    2d9h
8   kube-system    coredns-7d9985555d-klsqz                1/1     Running   2
    2d9h
9   kube-system    etcd-master-01                          1/1     Running   2
    2d9h
10  kube-system    kube-apiserver-master-01                1/1     Running   2
    2d9h
```

```
11 kube-system    kube-controller-manager-master-01       1/1     Running   3
   2d9h
12 kube-system    kube-flannel-ds-ns2r5                   1/1     Running   2
   2d9h
13 kube-system    kube-proxy-vfc97                        1/1     Running   2
   2d9h
14 kube-system    kube-scheduler-master-01                1/1     Running   3
   2d9h
15 mid            license-manager-55646b7fc9-zrvmk        2/2     Running   4
   2d9h
16 rpa            host-mqsubscriber-55bd4cd99f-wbxdj      1/1     Running   7
   2d9h
17 rpa            host-scheduler-55d5856d4f-75kf8         1/1     Running   7
   2d9h
18 rpa            view-commander-5b64d55fb9-brwgd         2/2     Running   4
   2d9h
19 rpa            view-global-ng-85d99f7b7c-h2mrx         2/2     Running   4
   2d9h
20 rpa            webapi-commander-bd5f55f89-ncn6t        2/2     Running   9
   2d9h
21 rpa            webapi-global-dd88cc94f-hxr4f           2/2     Running   9
   2d7h
22 rpa            webapi-open-c786f9545-r8495             2/2     Running   9
   2d9h
23 rpa            websocket-creator-867955d489-b97w5      2/2     Running   10
   2d9h
24 rpa            websocket-vnc-c665cd7c7-w4d88           2/2     Running   10
   2d9h
25 rpa            websocket-worker-5bc9f9cff-sdxgj        2/2     Running   10
   2d9h
26
27 [root@pvt-test-commander-01 checker-z0Lbrkw2j]# docker ps|grep -v 'Up'
28 CONTAINER ID        IMAGE
   COMMAND                  CREATED             STATUS              PORTS
   NAMES
29 [root@pvt-test-commander-01 checker-z0Lbrkw2j]#
```

✅ 通过网页验证服务是否正常

可通过谷歌或火狐浏览器登录mage或commander的地址，验证是否访问正常

- 如果仅部署了Mage的情况

- 地址：http://负载均衡ip:82
- 用户名：admin
- 密码：123456

- 如果部署包含了Commander，需要到租户系统中创建机构后，使用机构账号才可以登录commander或mage
  - 地址：http://负载均衡ip:81
  - 用户名：admin
  - 密码: 123456

3.5 更新授权文件

若客户mage环境或commander环境授权过期的情况下，可参考以下方法更新授权文件

更新commander授权

- 网页登录租户平台
- 点击左侧菜单栏：系统配置----License
- 点击上传license，将新的license文件上传即可

更新mage授权

- 将lic文件传到每台服务器的部署目录下的xxx/license-manager/data/目录下，并修改名字为license.lcs
- 在master1主机上执行kubectl -n mid rollout deploy 命令

例：客户有套mage高可用环境，环境信息如下：

（如果为单机，那么只需要在master1执行即可）

Plain Text

```
# 192.168.1.10 master1
# 192.168.1.11 master2
# 192.168.1.12 master3
# 将mage部署在/home/rpa下，新申请的mage授权文件为xxx.lcs，这里我已经将xxxx.lcs上传至每台服务器的root目录下
```

Shell

```
# 分别将xxxx.lcs上传至192.168.1.10/11/12每台服务器的部署目录/home/rpa/license-manager/data/
# 在每台机器上执行以下操作
cp /root/xxxx.lcs /home/rpa/license-manager/data/
# 开始替换
cd /home/rpa/license-manager/data/
# 备份旧的替换文件
mv license.lcs license$(date +%y%m%d).lcs.bak
# 将新的授权文件重命名为license.lcs
mv xxxx.lcs license.lcs

# 每台机器都执行成功以后，登录到master1主机，执行以下操作
kubectl -n mage rollout restart deploy
```

第4章 维护指南

Linux服务器维护

关闭防火墙

Shell

```
# 临时关闭selinux
setenforce 0

# 获取selinux状态,Enforcing是开启状态，Permissive是临时关闭状态，disabled是禁用
getenforce

# 禁用selinux
sed -i 's/\(SELINUX=\)[a-z].*/\1disabled/' /etc/selinux/config

# 关闭防火墙
systemctl stop firewalld

# 开机禁用防火墙
systemctl diable firewalld
```

查看服务器基础信息

Shell

```shell
# 查看机器的处理器架构
uname -m

# 查看详细的系统信息
uname -a

# 查看操作系统位数
getconf LONG_BIT

# 查看主机名
hostname

# 查看当前登录用户
whoami

# 查看系统发行版信息
lsb_release -a
cat /etc/*-release
cat /etc/issue

# 查看 CPU 信息
cat /proc/cpuinfo
lscpu

# 查看内存信息
cat /proc/meminfo
free --si -h

# 查看网卡信息
ip addr

# 查看磁盘信息
df -TH
fdisk -l /dev/sdb

# 查看详细的硬件信息
dmidecode
dmidecode | grep "System Information" -A9    # 查看服务器型号、序列号等
dmidecode | grep "Chassis Information" -A16  # 查看主板信息
dmidecode | grep "Memory Device" -A16        # 查看内存条信息

```

```
41
42  # 查看系统日志
43  cat /var/log/messages
44
45  # 查看当前用户的计划任务
46  crontab -l
47
48  # 查看系统级的计划任务
49  cat /etc/crontab
50  cat /etc/cron.d/*
```

关闭swap交换分区

Shell

```
1  # 临时关闭swap分区,当前会话生效，重启失效
2  swapoff -a
3
4  # 开机禁用swap分区
5  sed -i 's/.*swap.*/#&/' /etc/fstab
```

查看端口占用

Shell

```
1  # 查看当前监听的所有端口
2  netstat -lntpu
```

列出所有系统服务

Shell

```
1  # 列出所有系统服务
2  chkconfig --list
3
4  # 列出所有启动的服务
5  chkconfig --list|grep on
```

开机服务启停

Shell

```
# 检查某个服务是否是开机启用,enable表示是开机自动启动，disabled表示开机不启用。
systemctl is-enabled docker

# 检查某个服务状态,inactive表示停止状态，active表示活动状态
systemctl status docker|grep 'Active'

# 启用某个服务，重启后会失效
systemctl start docker

# 添加服务开机自启动
systemctl enable docker

# 移除服务开机自启动
systemctl disable docker

# 停止某个服务
systemctl stop docker
```

服务器之间传输文件

利用python可以快速搭建一个http服务，提供一个文件浏览的web服务，该服务是前台运行的，control+c会关闭该服务。可以随便更换端口

Shell

```
# python各版本启用方式

# python2.x
python2.7 -m SimpleHTTPServer 8899

# python3.x
python3 -m http.server 8899
```

例子：将192.168.1.10的/root/mage.tar.gz文件传输到192.168.1.11的/tmp目录下

（在哪个目录下执行，那么就传输哪个目录下文件）

Shell

```
# 在192.168.1.10的/root目录下执行
python2.7 -m SimpleHTTPServer 8899
```

Shell

```
# 在192.168.1.11上执行
wget -c 192.168.1.10:8899/mage.tar.gz
```

时间同步

Plain Text

```
# 服务器有外网的情况下,如果没有ntpdate命令，可使用yum下载
ntpdate time.windows.com

# 有内网时间服务器的情况下
ntpdate x.x.x.x
```

删除安装包

Shell

```
# 服务安装完成后，前置检查包和模块安装包都可以删除掉,本案例的前置检查包和安装以及解压包都放在/root目录下

rm -rf /tmp/*
rm -rf /root/check-*
rm -rf /root/8Pwa1alex*
```

一键卸载commander或mage环境

注意：由于该脚本会删除docker和k8s环境。所以仅可使用在集成部署的docker环境中使用。如果是客户自身的docker或k8s环境，请不要使用该脚本。否则造成的损失无法弥补

脚本下载地址:

https://private-deploy.oss-cn-beijing.aliyuncs.com/pengyongshi/uninstall_uibot.sh

使用方法：

```shell
sh uninstall_uibot.sh -d [指定安装目录，默认应该为/home/rpa]
```

例：假如我的部署目录为/data/rpa,执行以下即可。**路径千万别填错。因为有rm -rf 命令**

```shell
sh uninstall_uibot.sh -d /data/rpa
```

获取机器二维码

```shell
# 在任意路径下执行以下命令可获取机器二维码
docker run -v /etc/machine-id:/etc/machine-id --rm --entrypoint /lmcli qrcode mid get -q --qrcode-hd

# 在任意路径下执行以下命令可获取机器码
docker run -v /etc/machine-id:/etc/machine-id --rm --entrypoint /lmcli qrcode mid get
```

NVIDIA显卡驱动安装

安装准备

前提是机器上面有支持CUDA的Nvidia GPU，可从官方查询支持CUDA的GPU列表。另服务器需要有外网访问权限

参考链接：

本文例子使用的是centos7.9，如果是redhat可参考下方的官方链接进行安装。如果为高可用环境，每台gpu主机都需要执行以下操作。

cuda-10.2安装下载链接

nvidia-docker2安装下载链接

k8s安装nvidia设备插件

k8s-device-plugin

如何在Kubernetes集群中利用GPU进行AI训练

k8s如何调用GPU

安装显卡驱动

安装准备

可以通过以下命令查看是否有GPU显卡,如果有输出代表服务器上有GPU显卡

Shell

```
[root@master-03 ~]# lspci |grep -i nvidia
0000:3f:00.0 3D controller: NVIDIA Corporation Device 1eb8 (rev a1)
0000:40:00.0 3D controller: NVIDIA Corporation Device 1eb8 (rev a1)
0000:43:00.0 3D controller: NVIDIA Corporation Device 1eb8 (rev a1)
0000:47:00.0 3D controller: NVIDIA Corporation Device 1eb8 (rev a1)
0000:8e:00.0 3D controller: NVIDIA Corporation Device 1eb8 (rev a1)
0000:92:00.0 3D controller: NVIDIA Corporation Device 1eb8 (rev a1)

# 如果以下命令有结果输出，表明驱动已经被安装，可忽略安装显卡驱动环节
[root@master-03 ~]# nvidia-smi
```

开始安装

安装显卡驱动（mage代码在cuda11上跑不起来，所以需要安装10.2版本）

Shell

```
wget -c https://private-deploy.oss-cn-beijing.aliyuncs.com/pengyongshi/cuda-repo-rhel7-10-2-local-10.2.89-440.33.01-1.0-1.x86_64.rpm
sudo rpm -i cuda-repo-rhel7-10-2-local-10.2.89-440.33.01-1.0-1.x86_64.rpm
sudo yum clean all
sudo yum -y install nvidia-driver-latest-dkms cuda
sudo yum -y install cuda-drivers
```

可以使用nvidia-smi命令检测cuda是否安装成功,如果执行可以看到以下输出，表示驱动安装成功

Shell

```
[root@master-03 ~]# nvidia-smi
Mon Jul 26 18:28:21 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:3F:00.0 Off |                    0 |
| N/A   52C    P0    26W /  70W |   1848MiB / 15109MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla T4            Off  | 00000000:40:00.0 Off |                    0 |
| N/A   52C    P0    26W /  70W |    260MiB / 15109MiB |      0%      Default |
+-------------------------------+----------------------+----------------------
```

安装nvidia-docker

需要安装docker后，才可以安装nvidia-docker。因为在前置检查中获取二维码环节已经安装过docker，所以本环节跳过docker安装步骤。

安装要求

- GNU/Linux X86_64 kernel version > 3.10
- Docker >= 19.03
- NVIDIA GPU with Architecture >= Kepler (or compute capability 3.0)
- NVIDIA Linux drivers >= 418.81.07

开始安装

首先，需要备份/etc/docker/daemon.json文件，因为在安装nvidia-docker2时会覆盖该文件

Shell

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
   && curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.repo | sudo tee /etc/yum.repos.d/nvidia-docker.repo
sudo yum clean expire-cache
sudo yum install -y nvidia-docker2
```

以上步骤安装完后，会默认替换掉/etc/docker/daemon.json文件,将之前备份的daemon.json中的内容追加到文件里

JSON

```json
{
    "default-runtime": "nvidia",
    "runtimes": {
        "nvidia": {
            "path": "nvidia-container-runtime",
            "runtimeArgs": []
        }
    },
    "insecure-registries":[
        "10.116.28.239:8888"
    ]
}
```

重启docker

Shell

```
systemctl restart docker
```

安装完成后，可使用以下容器来测试安装是否成功

Shell

```
sudo docker run --rm --gpus all nvidia/cuda:10.2-base nvidia-smi
```

如果成功的话，控制台将会有以下输出

Plain Text

```
Mon Jul 26 11:11:33 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:3F:00.0 Off |                    0 |
| N/A   52C    P0    26W /  70W |   1848MiB / 15109MiB |      0%      Default |
```

安装kubernetes nvidia插件

安装要求

- Kubernetes version >= 1.10
- Nvidia-docker version > 2.0
- NVIDIA drivers >= 384.81
- Docker configured with nvidia as the `default runtime`

要在 Kubernetes 中使用 GPU，需要NVIDIA 设备插件。NVIDIA Device Plugin 是一个 daemonset，它会自动枚举集群每个节点上的 GPU 数量，并允许 pod 在 GPU 上运行。这个只需要在master1上安装即可。

部署设备插件首选的方法是使用`helm`,这里我们使用的是`kubectl apply`安装

创建nvidia-device-plugin.yaml文件

Shell

```
1  vim /var/lib/kubelet/plugins/nvidia-device-plugin.yaml
```

YAML

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nvidia-device-plugin-daemonset
  namespace: kube-system
spec:
  selector:
    matchLabels:
      name: nvidia-device-plugin-ds
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ""
      labels:
        name: nvidia-device-plugin-ds
    spec:
      tolerations:
      - key: CriticalAddonsOnly
        operator: Exists
      - key: nvidia.com/gpu
        operator: Exists
        effect: NoSchedule
      priorityClassName: "system-node-critical"
      containers:
      - image: nvidia/k8s-device-plugin:1.0.0-beta6
        name: nvidia-device-plugin-ctr
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop: ["ALL"]
        volumeMounts:
          - name: device-plugin
            mountPath: /var/lib/kubelet/device-plugins
      volumes:
        - name: device-plugin
          hostPath:
            path: /var/lib/kubelet/device-plugins
```

Shell

```
kubectl apply -f /var/lib/kubelet/plugins/nvidia-device-plugin.yaml
```

要测试是否可以部署 CUDA 作业，可以运行以下进行测试

将此 podspec 保存为`gpu-pod.yaml`. 现在，部署应用程序：

Shell

```
vim /tmp/gpu-test.yaml
```

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-operator-test
spec:
  restartPolicy: OnFailure
  containers:
  - name: cuda-vector-add
    image: "nvidia/samples:vectoradd-cuda10.2"
    resources:
      limits:
         nvidia.com/gpu: 1
```

Shell

```
kubectl apply -f /tmp/gpu-test.yaml
```

Shell

```
$ kubectl get pods gpu-operator-test
NAME                READY   STATUS      RESTARTS   AGE
gpu-operator-test   0/1     Completed   0          9d
```

检查日志输出以下内容则表示插件已成功完成安装：

SQL

```
$ kubectl logs gpu-operator-test
[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
```

kubernetes维护

通常情况下，客户有非root账号运维的请求，在安装时候需要使用root权限，之后需要以非root权限用户维护和管理。具体设置方法可参考第2.4章内容

k8s故障排查

故障排查的第一步是先给问题分类。问题是什么？是关于 Pods、Replication Controller 还是 Service?

- **节点运行状态为NotReady**

Shell

```
# 获取节点运行状态
[root@iZ2ze5xnx4yej77xhg8bwmZ harbor]# kubectl get nodes
NAME        STATUS     ROLES    AGE    VERSION
master-01   NotReady   master   145m   v1.18.12

# 获取节点的最近事件，排查节点NotReady原因
[root@iZ2ze5xnx4yej77xhg8bwmZ harbor]# kubectl describe nodes master-01|tail -n 10
Events:
  Type    Reason                   Age                    From     Message
  ----    ------                   ----                   ----     -------
  Normal  Starting                 3m22s                  kubelet  Starting kubelet.
  Normal  NodeHasSufficientMemory  3m22s (x2 over 3m22s)  kubelet  Node master-01 status is now: NodeHasSufficientMemory
  Normal  NodeHasNoDiskPressure    3m22s (x2 over 3m22s)  kubelet  Node master-01 status is now: NodeHasNoDiskPressure
  Normal  NodeHasSufficientPID     3m22s (x2 over 3m22s)  kubelet  Node master-01 status is now: NodeHasSufficientPID
```

```
01 status is now: NodeHasSufficientPID
  Normal  NodeNotReady             3m22s                  kubelet  Node master-01 status is now: NodeNotReady
  Normal  NodeAllocatableEnforced  3m19s                  kubelet  Updated Node Allocatable limit across pods
  Normal  NodeReady                3m19s                  kubelet  Node master-01 status is now: NodeReady

# 查询kubelet服务是否正常
[root@iZ2ze5xnx4yej77xhg8bwmZ harbor]# systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; vendor preset: disabled)
  Drop-In: /usr/lib/systemd/system/kubelet.service.d
           └─10-kubeadm.conf
   Active: inactive (dead) since 一 2021-05-31 18:53:49 CST; 2min 48s ago
     Docs: https://kubernetes.io/docs/
  Process: 12728 ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS $KUBELET_EXTRA_ARGS (code=exited, status=0/SUCCESS)
 Main PID: 12728 (code=exited, status=0/SUCCESS)

# 得知kubelet服务处于inactive状态，启动并加入开机自启
[root@iZ2ze5xnx4yej77xhg8bwmZ harbor]# systemctl start kubelet
[root@iZ2ze5xnx4yej77xhg8bwmZ harbor]# systemctl enable kubelet
```

- **Pod 停滞在 Pending 状态**

如果一个 Pod 停滞在 Pending 状态，表示 Pod 没有被调度到节点上。通常这是因为 某种类型的资源不足导致无法调度。 查看上面的 kubectl describe ... 命令的输出，其中应该显示了为什么没被调度的原因。 常见原因如下：

资源不足: 你可能耗尽了集群上所有的 CPU 或内存。此时，你需要删除 Pod、调整资源请求或者为集群添加节点

Shell

```
# 获取pod状态
[root@iZ2ze5xnx4yej77xhg8bwmZ harbor]# kubectl -n mage get po
NAME                                          READY   STATUS
RESTARTS   AGE
bert-service-tf-7c87459cf4-c8j5b              0/2     Pending            0
96m
doc-classifier-7fb584f4d5-xkwfd               2/2     Running            0
96m
doc-classifier-trainer-d84c69754-j9hpk        1/1     Running            0
96m
document-mining-auth-cf48686cb-82b8q          2/2     Running            0
104m
document-mining-innerservice-546c79469d-qj8vq     2/2     Running            25
104m
document-mining-openapi-86b7f89bd4-w7psv      2/2     Running            0
104m
document-mining-rpc-747c7cb44d-rn5kh          2/2     Running            0
104m
file-analyze-5c67858998-htjm9                 2/2     Running            0
104m

# 获取pod最近事件
[root@iZ2ze5xnx4yej77xhg8bwmZ harbor]# kubectl -n mage describe pod bert-
service-tf-7c87459cf4-c8j5b|tail -n 10
    Name:         istio-ca-root-cert
    Optional:     false
QoS Class:        Burstable
Node-Selectors:   <none>
Tolerations:      node.kubernetes.io/not-ready:NoExecute for 300s
                  node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type     Reason            Age   From               Message
  ----     ------            ----  ----               -------
  Warning  FailedScheduling  97m   default-scheduler  0/1 nodes are available: 1
Insufficient memory.

# 以上得知，内存不足导致Pod bert-service-tf-7c87459cf4-c8j5b处于Pending状态
```

- **Pod 停滞在 Waiting或ImagePullBackOff 状态**

如果 Pod 停滞在 `Waiting` 状态，则表示 Pod 已经被调度到某工作节点，但是无法在该节点上运行。 同样， `kubectl describe ...` 命令的输出可能很有用。 `Waiting` 状态的最常见原因是

拉取镜像失败。要检查的有三个方面：

确保镜像名字拼写正确

确保镜像已被推送到镜像仓库

用手动命令 `docker pull <镜像>` 试试看镜像是否可拉取

Shell

```
[root@iZ2ze5xnx4yej77xhg8bwmZ harbor]# kubectl -n mid get po
NAME                                  READY   STATUS             RESTARTS   AGE
license-manager-599f7cbc8c-dx2n5      2/2     Running            0          155m
license-manager-84cb76b78c-lv529      1/2     ImagePullBackOff   0          2m8s

# 查询Pod的镜像名字是否写错
kubectl -n rpa get  po license-manager-84cb76b78c-lv529 -o=jsonpath='{..image}'

# 手动拉取镜像看是否可以拉取成功
docker pull 镜像地址
```

- **Pod 处于 Running 态但是没有正常工作**

如果 Pod 行为不符合预期，很可能 Pod 有问题， 并且该错误在创建 Pod 时被忽略掉，没有报错。 通常，Pod 的定义中节区嵌套关系错误、字段名字拼错的情况都会引起对应内容被忽略掉。 例如，如果你误将 `command` 写成 `commnd`，Pod 虽然可以创建，但它不会执行 你期望它执行的命令行。

可以重启下Pod对应的deployment看看是否可以正常启动。

- **清除所有异常的Pod**

Shell

```
[root@iZ2ze5xnx4yej77xhg8bwmZ harbor]# kubectl -n rpa get pod|grep -v Running|awk '{print $1}'|xargs kubectl -n rpa delete pod
```

查看k8s资源

- **获取所有的namespace**

Plain Text

```
kubectl get namespace
```

- **获取rpa namespace下所有的deployment**

```
Plain Text
1  kubectl --namespace rpa get deployment
```

- **获取 rpa namespace下的configmap内容**

```
Plain Text
1  kubectl --namespace rpa get configmap -o yaml
```

重启Deployment

- 重启kube-system namespace下的coredns

`kubectl -n kube-system rollout restart deployment coredns`

- 重启istio-system下的所有deployment

```
Shell
1  kubectl -n istio-system rollout restart deployment
```

- 重启rpa下的所有deployment

```
Shell
1  kubectl -n rpa rollout restart deployment
```

扩/缩pod容器数量

扩、缩容pod主要是通过设置deployment里面的replicas的值来更改的

- 把kube-system namespace下的coredns扩到2个pod

```
Shell
1  kubectl -n kube-system scale deploy coredns --replicas=2
```

- 把istio-system namespace下的所有deployment设置成只运行一个pod

```
Shell
1  kubectl -n istio-system scale deployment --all --replicas=1
```

查看容器日志

- 查看kube-system namespace下的pod coredns

```shell
kubectl -n kube-system logs coredns-5fdns231-7821gh
```

- 动态查看rpa namespace下的pod webapi-commander的日志

```
```

Commander服务维护

查看commander运行的pod状态

```shell
[root@iZ2ze5xnx4yej77xhg8bwmZ harbor]# kubectl -n rpa get pod
NAME                                READY   STATUS    RESTARTS   AGE
host-mqsubscriber-548cd99fbb-s6xm9  1/1     Running   2          89m
host-scheduler-559bd7f4f4-7m8sm     1/1     Running   2          89m
view-commander-6b7b8fb69b-djjkf     2/2     Running   0          89m
view-global-ng-6d97779d68-lxhqx     2/2     Running   0          89m
webapi-commander-6cf57b9f74-jw5j4   2/2     Running   1          89m
webapi-global-76cb788b48-8ntql      2/2     Running   0          89m
webapi-open-5cc65455b4-jnvgk        2/2     Running   0          89m
websocket-creator-7d89974fbd-mthbc  2/2     Running   1          89m
websocket-vnc-c7db7f684-xbg9t       2/2     Running   1          89m
websocket-worker-784b4d8768-lfs9t   2/2     Running   2          89m
```

重启commander的服务

Shell

```
[root@iZ2ze5xnx4yej77xhg8bwmZ harbor]# kubectl -n rpa rollout restart deployment
deployment.apps/host-mqsubscriber restarted
deployment.apps/host-scheduler restarted
deployment.apps/view-commander restarted
deployment.apps/view-global-ng restarted
deployment.apps/webapi-commander restarted
deployment.apps/webapi-global restarted
deployment.apps/webapi-open restarted
deployment.apps/websocket-creator restarted
deployment.apps/websocket-vnc restarted
deployment.apps/websocket-worker restarted
```

1.3 查看Commander服务日志

Commander服务日志一般都在安装目录下的Logs目录下，默认是在/home/rpa/Logs。如果需要查看某个Pod的日志，进入到对应的服务目录下即可。也可以直接通过进入pod容器里获取。

服务运行日志可通过以下命令查看获取

Plain Text

```
kubectl -n rpa logs -f webapi-commander-758fc9758b-6rw2f webapi-commander
```

Mage服务维护

- **Mage服务重启**

Shell

```
[root@iZ2ze5xnx4yej77xhg8bwmZ harbor]# kubectl -n mage rollout restart deployments.apps
deployment.apps "bert-service-tf" restarted
deployment.apps "doc-classifier" restarted
deployment.apps "doc-classifier-trainer" restarted
deployment.apps "document-mining-auth" restarted
deployment.apps "document-mining-innerservice" restarted
deployment.apps "document-mining-openapi" restarted
deployment.apps "document-mining-rpc" restarted
deployment.apps "file-analyze" restarted
deployment.apps "info-engine" restarted
deployment.apps "mage-entitynormal" restarted
deployment.apps "ocr-ctpn-server" restarted
deployment.apps "ocr-ctpn-tf-server" restarted
deployment.apps "ocr-inner-tool" restarted
deployment.apps "ocr-server-dispatch" restarted
deployment.apps "ocr-text-recognition-server" restarted
deployment.apps "ocr-text-recognition-tf-server" restarted
deployment.apps "poi-search-engine" restarted
deployment.apps "poi-text-match" restarted
```

- **Mage授权服务重启**

Shell

```
[root@iZ2ze5xnx4yej77xhg8bwmZ harbor]# kubectl -n mid rollout restart deployments.apps
```

- **查看mage服务日志**

mage服务日志一般都在安装目录下的Logs目录下，默认是在/home/rpa/Logs。如果需要查看某个Pod的日志，进入到对应的服务目录下即可。也可以直接通过进入pod容器里获取。

服务运行日志可通过以下命令查看获取

Shell

```
kubectl -n rpa logs -f document-mining-rpc-758fc9758b-6rw2f document-mining-rpc
```

Docker维护

如何处理Docker卡顿

问题描述：执行docker ps或docker info命令都没有反应

问题解决：

- 查看docker有没有处于Active状态

Plain Text

```
[root@iZ2ze5xnx4yej77xhg8bwmZ rpa]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: failed (Result: start-limit) since 一 2021-05-31 16:42:06 CST; 13s ago
     Docs: https://docs.docker.com
  Process: 23916 ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock (code=exited, status=1/FAILURE)
 Main PID: 23916 (code=exited, status=1/FAILURE)

5月 31 16:42:04 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: docker.service failed.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: docker.service holdoff time over, scheduling restart.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: Stopped Docker Application Container Engine.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: start request repeated too quickly for docker.service
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: Failed to start Docker Application Container Engine.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: Unit docker.service entered failed state.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: docker.service failed.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: start request repeated too quickly for docker.service
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: Failed to start Docker Application Container Engine.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: docker.service failed
```

- 查看docker启动日志

Shell

```
[root@iZ2ze5xnx4yej77xhg8bwmZ rpa]# journalctl -xe
5月 31 16:42:03 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: Stopped Docker Application Container Engine.
5月 31 16:42:04 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: Starting Docker Application Container Engine...
5月 31 16:42:04 iZ2ze5xnx4yej77xhg8bwmZ dockerd[23916]: unable to configure the Docker daemon with file /etc/docker/daemon.json: invalid character ']' looking for beginning of value
5月 31 16:42:04 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: docker.service: main process exited, code=exited, status=1/FAILURE
5月 31 16:42:04 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: Failed to start Docker Application Container Engine.
5月 31 16:42:04 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: Unit docker.service entered failed state.
5月 31 16:42:04 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: docker.service failed.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: docker.service holdoff time over, scheduling restart.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: Stopped Docker Application Container Engine.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: start request repeated too quickly for docker.service
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: Failed to start Docker Application Container Engine.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: Unit docker.service entered failed state.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: docker.service failed.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: start request repeated too quickly for docker.service
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: Failed to start Docker Application Container Engine.
5月 31 16:42:06 iZ2ze5xnx4yej77xhg8bwmZ systemd[1]: docker.service faile
```

- 日志得知是/etc/docker/daemon.json文件出了问题，修改文件重启docker即可

中间件维护

Commander和mage的中间件服务都是用docker-compose命令管理容器，在使用docker-compose的命令时，默认会在当前目录下找docker-compose.yml文件

Plain Text

```
# 进入对应的中间件服务目录下

# 基于docker-compose.yml创建并启动的容器
docker-compose up -d

# 关闭并删除容器
docker-compose down

# 开启|关闭|重启已经存在的容器
docker-compose start|stop|restart

# 查看由docker-compose管理的容器
docker-compose ps
```

第5章 Docker基本使用

第6章 Uibot安全加固指南