TKE 企业版

部署文档 v2.6

腾讯云计算（北京）有限责任公司

**目 录**

[第一部分 前言 1](#_Toc34216885)

[读者对象 1](#_Toc34216886)

[文档目的 1](#_Toc34216887)

[第二部分 部署和配置简介 1](#_Toc34216888)

[第三部分 开始部署 2](#_Toc34216889)

[检查部署需求是否满足 2](#_Toc34216890)

[解决不满足的需求 2](#_Toc34216891)

[软件需求 2](#_Toc34216892)

[解决工具软件不满足的步骤 3](#_Toc34216893)

[硬件需求 3](#_Toc34216894)

[网络需求 4](#_Toc34216895)

[global 平台部署 4](#_Toc34216896)

[server\_list.json 文件格式说明 4](#_Toc34216897)

[all in one 方案 4](#_Toc34216898)

[poc 部署方案 5](#_Toc34216899)

[正式生产环境部署方案 7](#_Toc34216900)

[业务服务集群部署 12](#_Toc34216901)

[业务服务集群组件、global 产品和第三方插件部署 12](#_Toc34216902)

[业务集群必备组件 12](#_Toc34216903)

[业务集群必备组件 cert-manager 12](#_Toc34216904)

[业务集群必备组件 alauda-cluster-base 13](#_Toc34216905)

[业务集群必备组件 普罗米修斯 14](#_Toc34216906)

[业务集群可选组件 18](#_Toc34216907)

[业务集群部署 alb2 18](#_Toc34216908)

[nevermore 18](#_Toc34216909)

[nginx-ingress 19](#_Toc34216910)

[cron-hpa-controller【定时扩缩容的组件，可不部署】 20](#_Toc34216911)

[metrics-server 【devops 自动扩缩容需要用到这个组件】 21](#_Toc34216912)

[Container-platform（容器管理平台） 22](#_Toc34216913)

[devops 23](#_Toc34216914)

[第四部分 平台配置及测试 36](#_Toc34216915)

[检查部署是否成功 36](#_Toc34216916)

[检查 global 平台是否部署成功 36](#_Toc34216917)

[检查业务服务集群是否部署成功 36](#_Toc34216918)

[平台访问地址及登录用户名、密码，以及兼容的浏览器 37](#_Toc34216919)

[平台访问地址 37](#_Toc34216920)

[平台登录权限 37](#_Toc34216921)

[兼容的浏览器版本 37](#_Toc34216922)

[部署成功后操作 38](#_Toc34216923)

[备份 etcd 数据 38](#_Toc34216924)

[添加 gc 脚本 38](#_Toc34216925)

[第五部分 FAQ 38](#_Toc34216926)

[平台部署错误 38](#_Toc34216927)

[安装 chart 失败 38](#_Toc34216928)

[对接 global 集群失败 38](#_Toc34216929)

# 第一部分 前言

本文档介绍了如何安装和配置平台。

## 读者对象

本文档适用于具备基本的 linux、网络、存储、容器、Kubernetes 知识，想要安装和配置平台的用户，包括：

* 实施顾问和平台管理员
* 规划平台架构的售前工程师
* 负责整个项目生命周期的项目经理

## 文档目的

* 实施工程师依据本文档部署平台。
* 实施工程师依据本文档，配置平台参数。
* 平台管理员依据本文档测试平台功能。

# 第二部分 部署和配置简介

平台由 global 、global 插件、业务服务集群和集群组件构成，global 是平台的核心，管理、运维功能由 global 提供。global 提供了丰富的插件来扩展 global 的功能，。

业务服务集群是 global 管理的运行客户业务服务的 Kubernetes 集群，集群组件是运行在客户业务服务集群之上的功能模块，提供平台功能、搜集相关数据。

平台支持在 centos 操作系统上部署，操作系统版本和配置请参考本文档中软件需求部分。

平台中的 global 、global 插件、业务服务器集群、业务服务集群插件的部署请看本文第四部分。

# 第三部分 开始部署

## 检查部署需求是否满足

## 解决不满足的需求

### 软件需求

|  |  |  |
| --- | --- | --- |
| 需求项 | 具体要求 | 不满足后的解决办法 |
| 操作系统 | centos 7.6 | 7.5 7.4 也可以安装部署，但需要升级 kernel (ovn 要求) 和手动安装 docker |
| kernel 版本 | 大于 3.10.0-957 | 必须满足 |
| 操作系统安装要求 | 最小安装 | 如果不满足，可能出现配置不同造成部署失败 |
| 工具软件 | curl tar ip ssh sshpass jq netstat timedatectl ntpdate nslookup base64 tr head openssl md5sum | 手动添加源，并安装软件，请看表格后面的操作方式 |
| 用户权限 | root | 具备sudo 权限的用户也可部署，但有一定几率部署失败 |
| swap | 关闭 | 必须关闭 |
| 防火墙 | 关闭 | 必须关闭 |
| selinux | 关闭 | 必须关闭 |
| 时间同步 | 所有服务器要求时间必须同步，误差不得超过 1 秒 | 必须同步 |
| 时区 | 所有服务器时区必须统一 | 必须统一 |
| /etc/sysctl.conf | vm.max\_map\_count=262144 net.ipv4.ip\_forward = 1 | 必须满足 |
| hostname 格式 | 字母开头，只能是字母、数字和短横线-组成，不能用短横线结尾，长度在 4-23 之间。 | 必须满足 |
| /etc/hosts | 所有服务器可以通过hostname 解析成 ip，可以将localhost解析成127.0.0.1，hosts 文件内，不能有重复的 hostname | 必须满足 |
| /tmp/权限 | 要求 /tmp 目录的权限是 777 | 必须满足 |

#### 解决工具软件不满足的步骤

**注意**：因环境差异，kaldre 目录做本地源不会100% 安装软件成功，此时，就需要自行配置软件源去安装。

1. 解压缩安装包
2. 将安装包内的 kaldr 目录，拷贝到每一台服务器的 /tmp 目录下。

* 在每一台服务器上执行如下命令，添加软件源。
  + 适用 于 Centos ：
  + mkdir /etc/yum.repos.d/old  
    mv /etc/yum.repos.d/\*.repo /etc/yum.repos.d/old  
    cat <<EOF >/etc/yum.repos.d/alauda.repo   
    [alauda]  
    name=Alauda  
    baseurl=file:///tmp/kaldr/yum/  
    enabled=1  
    gpgcheck=0  
    EOF  
    yum clean all && yum makecache
  + 适用于 Ubuntu：
  + mv /etc/apt/sources.list /etc/apt/sources.list.old  
    echo "deb [arch=amd64] file:///tmp/kaldr/apt/ ubuntu-xenial main" >/etc/apt/sources.list  
    add-apt-repository "deb [arch=amd64] file:///tmp/kaldr/apt/ ubuntu-xenial main"  
    cat /tmp/kaldr/apt/alauda.key | apt-key add -  
    apt-get update
* **补充说明**：可通过以下命令行，确认某个软件属于哪个包。
  + 适用 于 Centos：yum provides <程序名>
  + 适用于 Ubuntu ：apt-file search <程序名>

1. 安装软件包，参考以下命令行。
   * 适用 于 Centos：yum install -y <上一步获取到的软件包的名字>
   * 适用于 Ubuntu ：apt-get install -y <上一步获取到的软件包的名字>

### 硬件需求

**注意**：这是实施部署的前提条件，必须满足硬件需求，不满足硬件需求就无法保证正常部署，或部署成功的平台无法长时间稳定运行。

### 网络需求

**注意**：这是实施部署的前提条件，必须满足网络需求，不满足网络需求就无法保证正常部署，或部署成功的平台无法长时间稳定运行。

## global 平台部署

### server\_list.json 文件格式说明

"server\_role": { #服务器角色  
 "master":true, #有这个键值，代表这台服务器角色是 k8s 的 master  
 "global":true, #有这个键值，代表这是运行 global 组件的服务器  
 "log":true #有这个键值，代表这是运行日志组件的服务器  
},  
"ip\_addr": "1.1.1.1", #服务器 ip 地址  
"ssh\_port": "22", #ssh 端口  
"ssh\_user": "root", #ssh 用户名  
"ssh\_passwd": "", #ssh 密码  
"ssh\_key\_file": "/root/.ssh/id\_rsa" #ssh 密钥，如果这个值不为空，那么就优先通过密钥登录，就是出现错误也不会选择 password 认证登录。

### all in one 方案

部署命令执行的位置：all in one 服务器上

**部署过程**：

1. 解压缩安装包。
2. 进入到解压缩后的目录。
3. 执行以下命令进行部署：

* **部署方式 1**： 部署脚本配置如下参数，部署一个通过 https://<本机 ip> 访问的环境：
* bash ./up-cpaas.sh \  
   --network-interface=<网卡名，默认eth0> \  
   --enabled-features=<需要安装那些产品，默认acp,devops,tke,asm> \  
   --single-mode
* **部署方式 2**： 部署脚本配置如下参数，部署一个通过 http://<本机 ip> 访问的环境：
* bash ./up-cpaas.sh \  
   --network-interface=<网卡名，默认eth0> \  
   --enabled-features=<需要安装那些产品，默认acp,devops,tke,asm > \  
   --use-http \  
   --single-mode
* **部署方式 3**： 部署脚本配置如下参数，部署一个通过 https://<域名> 访问的环境：
* bash ./up-cpaas.sh \  
   --network-interface=<网卡名，默认eth0> \  
   --enabled-features=<需要安装那些产品，默认acp,devops,tke,asm > \  
   --domain-name=<域名> \  
   --key-file=<域名证书私钥文件> \  
   --cert-file=<域名证书公钥文件> \  
   --single-mode

**注 1**：只有上述例子中的参数经过测试，改变参数的做法很可能造成部署不成功。 **注 2**：所有有默认值的参数，都是可选的，如果部署时不添加这个参数，就使用默认值。

**注 3**：all in one 不支持部署 API 管理平台。

### poc 部署方案

部署命令执行的位置： init 服务器上

**部署过程**：

1. 解压缩安装包。
2. 进入到解压缩后的目录。
3. 在 up-cpaas.sh 脚本所在的目录下创建 server\_list.json 文件，示例如下：

* [  
   {  
   "server\_role": {  
   "master":true,  
   "global":true,  
   "log":true  
   },  
   "ip\_addr": "1.1.1.1",  
   "ssh\_port": "22",  
   "ssh\_user": "root",  
   "ssh\_passwd": "",  
   "ssh\_key\_file": "/root/.ssh/id\_rsa"  
   },  
   {  
   "server\_role": {  
   "master":true,  
   "global":true,  
   "log":true  
   },  
   "ip\_addr": "2.2.2.2",  
   "ssh\_port": "22",  
   "ssh\_user": "root",  
   "ssh\_passwd": "",  
   "ssh\_key\_file": "/root/.ssh/id\_rsa"  
   },  
   {  
   "server\_role": {  
   "master":true,  
   "global":true,  
   "log":true  
   },  
   "ip\_addr": "3.3.3.3",  
   "ssh\_port": "22",  
   "ssh\_user": "root",  
   "ssh\_passwd": "",  
   "ssh\_key\_file": "/root/.ssh/id\_rsa"  
   }  
  ]

1. 执行以下命令进行部署：

* **部署方式 1**：部署一个通过 https://<域名> 访问的高可用环境：
* bash ./up-cpaas.sh \  
   --network-interface=<网卡名，默认是eth0> \  
   --cpaas-namespaces=<ns 名，默认是cpaas-system> \  
   --enabled-features=<需要安装那些产品，默认acp,devops,tke,asm > \  
   --global-network-mode=<选择 global 集群使用那种网络，默认是 ovn，可选 flannel 和 calico> \  
   --kube-cluster-subnet<global 集群的cluster ip 范围，默认是10.96.0.0/12，掩码建议不要小于16位> \  
   --domain-name=<域名> \  
   --kube\_controlplane\_endpoint=<k8s vip> \  
   --key-file=<域名证书私钥文件> \  
   --cert-file=<域名证书公钥文件> \  
   --root-username=<管理员用户名，必须是邮箱格式，默认是 admin@cpaas.io>
* **部署方式 2**：如果客户没有提供 lb 和 vip ，也可以用下面的命令进行部署，但不再是高可用的环境了。
* bash ./up-cpaas.sh \  
   --network-interface=<网卡名，默认是eth0> \  
   --make-k8s-lb \  
   --domain-name=<域名> \  
   --key-file=<域名证书私钥文件> \  
   --cert-file=<域名证书公钥文件> \  
   --enabled-features=<需要安装那些产品，默认acp,devops,tke,asm >
* **部署方式 3**：如果客户连域名也不提供，也可以用下面的命令进行部署，但不再是高可用的环境了。
* bash ./up-cpaas.sh \  
   --network-interface=<网卡名，默认是eth0> \  
   --make-k8s-lb \  
   --enabled-features=<需要安装那些产品，默认acp,devops,tke,asm > \  
   --use-http

**注 1**： 只支持上面给出的部署例子，部署参数组合在上述例子之外的都未经测试，不能保证支持。

**注 2**： 所有有默认值的参数，都是可选的，如果部署时不添加这个参数，就使用默认值。

**注 3**： 不提供域名证书，也不使用--use-http 参数，脚本会自动签发证书，但是 chrome 浏览器会判断出这个证书不是合法机构签发的，会报错禁止浏览，需要更改 chrome 浏览器的的启动参数，容忍不合法的证书，修改方法请自行搜索。

### 正式生产环境部署方案

部署命令执行的位置： init 服务器上

**部署过程**：

1. 解压缩安装包。
2. 进入到解压缩后的目录。
3. 在 up-cpaas.sh 脚本所在的目录下创建 server\_list.json 文件，示例如下：
   * 6 节点 server\_list.json 文件详情：
   * [  
      {  
      "server\_role": {  
      "master":true,  
      "global":true  
      },  
      "ip\_addr": "1.1.1.1",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      },  
      {  
      "server\_role": {  
      "master":true,  
      "global":true  
      },  
      "ip\_addr": "2.2.2.2",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      },  
      {  
      "server\_role": {  
      "master":true,  
      "global":true  
      },  
      "ip\_addr": "3.3.3.3",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      },  
      {  
      "server\_role": {  
      "log":true  
      },  
      "ip\_addr": "4.4.4.4",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      },  
      {  
      "server\_role": {  
      "log":true  
      },  
      "ip\_addr": "5.5.5.5",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      },  
      {  
      "server\_role": {  
      "log":true  
      },  
      "ip\_addr": "6.6.6.6",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      }  
     ]
   * 9 节点 server\_list.json 文件详情：
   * [  
      {  
      "server\_role": {  
      "master":true  
      },  
      "ip\_addr": "1.1.1.1",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      },  
      {  
      "server\_role": {  
      "master":true  
      },  
      "ip\_addr": "2.2.2.2",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      },  
      {  
      "server\_role": {  
      "master":true  
      },  
      "ip\_addr": "3.3.3.3",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      },  
      {  
      "server\_role": {  
      "log":true  
      },  
      "ip\_addr": "4.4.4.4",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      },  
      {  
      "server\_role": {  
      "log":true  
      },  
      "ip\_addr": "5.5.5.5",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      },  
      {  
      "server\_role": {  
      "log":true  
      },  
      "ip\_addr": "6.6.6.6",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      },  
      {  
      "server\_role": {  
      "global":true  
      },  
      "ip\_addr": "7.7.7.7",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      },  
      {  
      "server\_role": {  
      "global":true  
      },  
      "ip\_addr": "8.8.8.8",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      },  
      {  
      "server\_role": {  
      "global":true  
      },  
      "ip\_addr": "9.9.9.9",  
      "ssh\_port": "22",  
      "ssh\_user": "root",  
      "ssh\_passwd": "",  
      "ssh\_key\_file": "/root/.ssh/id\_rsa"  
      }  
     ]
4. 执行如下命令部署：

* **部署方式 1**：部署一个通过 https://<域名> 访问的高可用环境：
* bash ./up-cpaas.sh \  
   --network-interface=<网卡名，默认是eth0> \  
   --cpaas-namespaces=<ns 名，默认是cpaas-system> \  
   --enabled-features=<需要安装那些产品，默认acp,devops,tke,asm > \  
   --global-network-mode=<选择 global 集群使用那种网络，默认是 kube-ovn，可选 flannel 和 calico> \  
   --kube-cluster-subnet<global 集群的cluster ip 范围，默认是10.96.0.0/12，掩码建议不要小于16位> \  
   --domain-name=<域名> \  
   --kube\_controlplane\_endpoint=<k8s vip> \  
   --cert-sans=<可选参数，如果没有容灾需求可以不加，两地 global 集群 vip 的 vip> \  
   --key-file=<域名证书私钥文件> \  
   --cert-file=<域名证书公钥文件> \  
   --root-username=<管理员用户名，必须是邮箱格式，默认是 admin@cpaas.io>
* **部署方式 2**：如果客户没有提供证书，也可以用下面的命令进行部署，脚本会自动创建一个证书，但是浏览器访问 ui 的时候会提示证书不是正规机构颁发的，有安全隐患。
* bash ./up-cpaas.sh \  
   --network-interface=<网卡名，默认是eth0> \  
   --cpaas-namespaces=<ns 名，默认是cpaas-system> \  
   --enabled-features=<需要安装那些产品，默认acp,devops,tke,asm > \  
   --global-network-mode=<选择 global 集群使用那种网络，默认是 kube-ovn，可选 flannel 和 calico> \  
   --kube-cluster-subnet<global 集群的cluster ip 范围，默认是10.96.0.0/12，掩码建议不要小于16位> \  
   --domain-name=<域名> \  
   --kube\_controlplane\_endpoint=<k8s vip> \  
   --cert-sans=<可选参数，如果没有容灾需求可以不加，两地 global 集群 vip 的 vip> \  
   --root-username=<管理员用户名，必须是邮箱格式，默认是 admin@cpaas.io>
* **部署方式 3**：如果客户没有提供证书，也不希望浏览器提示证书不是正规机构颁发，可以选择使用 http 协议访问，执行如下命令部署。
* bash ./up-cpaas.sh \  
   --use-http \  
   --network-interface=<网卡名，默认是eth0> \  
   --cpaas-namespaces=<ns 名，默认是cpaas-system> \  
   --enabled-features=<需要安装那些产品，默认acp,devops,tke,asm > \  
   --global-network-mode=<选择 global 集群使用那种网络，默认是 kube-ovn，可选 flannel 和 calico> \  
   --kube-cluster-subnet<global 集群的cluster ip 范围，默认是10.96.0.0/12，掩码建议不要小于16位> \  
   --domain-name=<域名> \  
   --kube\_controlplane\_endpoint=<k8s vip> \  
   --cert-sans=<可选参数，如果没有容灾需求可以不加，两地 global 集群 vip 的 vip> \  
   --root-username=<管理员用户名，必须是邮箱格式，默认是 admin@cpaas.io>

**注 1**： 只支持上面给出的部署例子，部署参数组合在上述例子之外的都未经测试，不保证能够支持。

**注 2**： 所有有默认值的参数，都是可选的，如果部署时不添加这个参数，就使用默认值。

**注 3**： 不提供域名证书，也不使用--use-http 参数，脚本会自动签发证书，但是 chrome 浏览器会判断出这个证书不是合法机构签发的，会报错禁止浏览，需要更改 chrome 浏览器的启动参数，容忍不合法的证书，修改方法请自行搜索。

## 业务服务集群部署

登录 global 的 UI 后，部署业务服务集群或对接业务服务集群。

添加tke集群

图片包含 屏幕截图

描述已自动生成

新建独立集群

图片包含 屏幕截图

描述已自动生成

如果部署的是高可用集群，需要选择vip，默认端口6443。此vip需要能够将6443端口数据转发至所有master

图片包含 屏幕截图

描述已自动生成

定义master节点，点击保存，如果是高可用集群，需要定义至少3台master

定义完所有master节点后，点击“提交”。

创建完成后添加业务节点

图片包含 屏幕截图

描述已自动生成图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成图片包含 屏幕截图

描述已自动生成

添加完所有节点后点击提交

完成后效果

图片包含 屏幕截图

描述已自动生成

## 业务服务集群组件、global 产品和第三方插件部署

### 业务集群必备组件

#### 业务集群必备组件helm

helm安装方法 (for tke2.6.1)

图片包含 屏幕截图

描述已自动生成图片包含 屏幕截图

描述已自动生成

registry=$(docker info |grep 60080 |tr -d ' ')

chart\_repo\_url=http://chartmuseum:chartmuseum@${registry/60080/8088}

docker run -ti --rm -v /usr/local/bin/:/var/log/abc $registry/claas/helm:v2.14.3 sh -c "cp /systembin/helm /var/log/abc"

helm init --stable-repo-url=$chart\_repo\_url --tiller-image=$registry/claas/tiller:v2.14.3

[root@10 ~]# helm list

Error: configmaps is forbidden: User "system:serviceaccount:kube-system:default" cannot list configmaps in the namespace "kube-system"

# 报以上错误请用以下命令加权限

kubectl create serviceaccount --namespace kube-system tiller

kubectl patch deploy --namespace kube-system tiller-deploy -p '{"spec":{"template":{"spec":{"serviceAccount":"tiller"}}}}'

kubectl create clusterrolebinding tiller-cluster-rule --clusterrole=cluster-admin --serviceaccount=kube-system:tiller

#### 业务集群必备组件 cert-manager

**前提条件**：本文档旨在说明如何在 Kubernetes 集群安装 cert-manager ，操作的前提是当前 Kubernetes 集群已经安装好了 helm。

在业务集群master上执行

registry=$(docker info |grep 60080 |tr -d ' ')  
ACP\_NAMESPACE=cpaas-system ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
helm install \  
 --name cert-manager \  
 --namespace ${ACP\_NAMESPACE} \  
 --set global.registry.address=$registry \  
 stable/cert-manager

#清理操作

# helm delete --purge cert-manager

# kubectl delete crd certificates.certmanager.k8s.io

# kubectl delete crd challenges.certmanager.k8s.io

# kubectl delete crd clusterissuers.certmanager.k8s.io

# kubectl delete crd issuers.certmanager.k8s.io

# kubectl delete crd orders.certmanager.k8s.io

#### 业务集群必备组件 alauda-cluster-base

**说明**：

* 部署 alauda-cluster-base 依赖 cert-manager 和 globla 集群的 captain。
* 本文档旨在说明如何在 Kubernetes 集群安装 alauda-cluster-base。

在 global 的第一台 master 节点上执行以下命令：

ROOT\_USERNAME=admin@cpaas.io ## 默认为admin@cpaas.io，需要与登录global界面时使用的邮箱一致。  
REGISTRY\_ENDPOINT=$(docker info |grep 60080 |tr -d ' ')  
REGION\_NAME=cls-n9wfggmk ## 想要部署 alauda-cluster-base 的集群的名字，需要自行修改,tke集群名获取可从页面获取

图片包含 屏幕截图

描述已自动生成  
ACP\_NAMESPACE=cpaas-system ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
  
mkdir /tmp/region\_helmrequest  
cat << EOF >/tmp/region\_helmrequest/${REGION\_NAME}-alauda-cluster-base.yaml  
apiVersion: app.alauda.io/v1alpha1  
kind: HelmRequest  
metadata:  
 finalizers:  
 - captain.alauda.io  
 generation: 1  
 name: ${REGION\_NAME}-alauda-cluster-base  
 namespace: ${ACP\_NAMESPACE}  
spec:  
 chart: stable/alauda-cluster-base  
 namespace: ${ACP\_NAMESPACE}  
 releaseName: ${REGION\_NAME}-alauda-cluster-base  
 clusterName: ${REGION\_NAME}  
 values:  
 global:  
 auth:  
 default\_admin: ${ROOT\_USERNAME}  
 labelBaseDomain: cpaas.io  
 namespace: ${ACP\_NAMESPACE}  
 registry:  
 address: ${REGISTRY\_ENDPOINT}  
 version: $(helm search | grep '^stable/alauda-cluster-base ' | awk '{print $2}')  
EOF  
kubectl create -f /tmp/region\_helmrequest/${REGION\_NAME}-alauda-cluster-base.yaml

#清理操作

#kubectl delete -f /tmp/region\_helmrequest/${REGION\_NAME}-alauda-cluster-base.yam

#### 业务集群必备组件 普罗米修斯

**说明**：

* 部署普罗米修斯依赖 alauda-cluster-base ，部署前一定要确认alauda-cluster-base 已经安装部署成功。
* 本文档旨在说明如何在 Kubernetes 集群安装 prometheus operator 和 kube-prometheus ，操作的前提是当前 Kubernetes 集群已经安装好了 helm。
* 部署普罗米修斯默认部署在变量 ACP\_NAMESPACE 定义的命名空间下，若无特别需要，请勿修改 namespace、Service Mesh 等需要依赖普罗米修斯，修改 namespace 时如果填写错误，会增大排查难度。

1. 准备工作，监控 etcd 先判断 etcd-ca 是否存在，在要部署普罗米修斯的集群的 master 节点操作。

* kubectl get secrets -n cpaas-system | grep etcd-ca
* 若不存在则按以下命令添加，若存在，跳过这一步。
* **注意**：命令要修改 ns ，不能复制粘贴。
* kubectl get secrets -n kube-system etcd-ca -o yaml >/tmp/etcd-ca.yaml  
  sed -i '/namespace:/{s/kube-system/<改成部署时 配置的 ns 的值>/g}' /tmp/etcd-ca.yaml  
  kubectl apply -f /tmp/etcd-ca.yaml
* **注意**：如果上一步报错，提示找不到 etcd-ca，执行下面的命令创建
* kubectl create secret tls etcd-ca --cert=/etc/kubernetes/pki/etcd/ca.crt --key=/etc/kubernetes/pki/etcd/ca.key -n <改成部署时 配置的 ns 的值>  
    
  kubectl create secret tls etcd-peer --cert=/etc/kubernetes/pki/etcd/peer.crt --key=/etc/kubernetes/pki/etcd/peer.key -n <改成部署时 配置的 ns 的值>

1. 安装 prometheus-operator，在要部署的业务集群的第一台 master 节点上执行以下命令：

* REGISTRY\_ENDPOINT=$(docker info |grep 60080 |tr -d ' ')  
  REGION\_NAME=cls-n9wfggmk ## 想要部署 alauda-cluster-base 的集群的名字，需要自行修改  
  ACP\_NAMESPACE=cpaas-system ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
  helm install --version $(helm search | grep '^stable/prometheus-operator ' | awk '{print $2}') \  
   --namespace=${ACP\_NAMESPACE} \  
   --name=prometheus-operator \  
   --set global.namespace=${ACP\_NAMESPACE} \  
   --set global.registry.address=${REGISTRY\_ENDPOINT} \  
   stable/prometheus-operator --wait --timeout 3000

1. 安装 kube-prometheus。

* 安装 kube-prometheus 有以下两种方式，一种为使用本地目录的方式存储监控数据，一种为使用 pvc 的方式存储。可自行选择。若无特殊需要，本地目录的方式即可。
  + 本地目录方式部署 kube-prometheus。
  + 给集群中的一个 node 添加 monitoring=enabled 的 label，用于 local volume 的调度，在要部署普罗米修斯的集群的 master 节点上操作，命令如下：
  + kubectl label --overwrite nodes region-1 monitoring=enabled
  + **注意**：需要将 test 替换为这个 node 的实际 hostname。在该 node 上创建以下目录用作持久化目录，保证空间 granafa 5G/prometheus 30G/alertmanager 5G，ssh 到这台服务器上，执行如下命令：
  + mkdir -p /cpaas/monitoring/{grafana,prometheus,alertmanager}  
    chmod -R 777 /cpaas/monitoring
  + **注意**：下面的操作在要部署的业务集群的第一台 master 节点上操作，执行如下命令：

REGISTRY\_ENDPOINT=$(docker info |grep 60080 |tr -d ' ')  
DOMAIN\_NAME=192.168.122.10 ## 需要修改为 global 界面的访问地址，也就是部署 global 的时候，--domain-name 参数的值  
REGION\_NAME=cls-n9wfggmk ## 想要部署 alauda-cluster-base 的集群的名字  
ACP\_NAMESPACE=cpaas-system ## 改成部署 global 时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
  
helm install --version $(helm search | grep '^stable/kube-prometheus ' | awk '{print $2}') \  
 --namespace=${ACP\_NAMESPACE} \  
 --name=kube-prometheus \  
 --set global.namespace=${ACP\_NAMESPACE} \  
 --set global.platform=ACP \  
 --set global.labelBaseDomain=cpaas.io \  
 --set prometheus.service.type=NodePort \  
 --set grafana.service.type=NodePort \  
 --set global.registry.address=${REGISTRY\_ENDPOINT} \  
 --set grafana.storageSpec.persistentVolumeSpec.local.path=/cpaas/monitoring/grafana \  
 --set prometheus.storageSpec.persistentVolumeSpec.local.path=/cpaas/monitoring/prometheus \  
 --set alertmanager.storageSpec.persistentVolumeSpec.local.path=/cpaas/monitoring/alertmanager \  
 --set alertmanager.configForACP.receivers[0].name=default-receiver \  
 --set alertmanager.configForACP.receivers[0].webhook\_configs[0].url=https://${DOMAIN\_NAME}/v1/alerts/${REGION\_NAME}/router \  
 stable/kube-prometheus  
   
PVC 方式部署 kube-prometheus，在要部署普罗米修斯的 Kubernetes 集群的第一台 master 节点上执行以下命令：

* + REGISTRY\_ENDPOINT=$(docker info |grep 60080 |tr -d ' ')  
    DOMAIN\_NAME=<vip or domainname> ## 需要修改为 global 界面的访问地址，也就是部署 global 的时候，--domain-name 参数的值  
    REGION\_NAME=<name> ## 想要部署 alauda-cluster-base 的集群的名字  
    ACP\_NAMESPACE=<cpaas-system> ## 改成部署 global 时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
    storage=test ## 需要将 test 替换为当前集群的 StorageClass 的名字   
      
    helm install --version $(helm search | grep '^stable/kube-prometheus ' | awk '{print $2}') \  
     --namespace=${ACP\_NAMESPACE} \  
     --name=kube-prometheus \  
     --set global.namespace=${ACP\_NAMESPACE} \  
     --set global.platform=ACP \  
     --set global.labelBaseDomain=cpaas.io \  
     --set prometheus.service.type=NodePort \  
     --set grafana.service.type=NodePort \  
     --set global.registry.address=${REGISTRY\_ENDPOINT} \  
     --set grafana.storageSpec.volumeClaimTemplate.spec.storageClassName=${storage} \  
     --set prometheus.storageSpec.volumeClaimTemplate.spec.storageClassName=${storage} \  
     --set alertmanager.storageSpec.volumeClaimTemplate.spec.storageClassName=${storage} \  
     --set alertmanager.configForACP.receivers[0].name=default-receiver \  
     --set alertmanager.configForACP.receivers[0].webhook\_configs[0].url=https://${DOMAIN\_NAME}/v1/alerts/${REGION\_NAME}/router \  
     --set alertmanager.configForACP.receivers[0].webhook\_configs[0].send\_resolved='false' \ stable/kube-prometheus

1. 查看所有 pod 是否正常启动（pod 为 Running 或者 Completed 状态），在部署普罗米修斯的集群的 master 节点上执行以下命令：

* kubectl get pods -n cpaas-system | grep prometheus

1. 集群对接监控，在部署普罗米修斯的集群的 master 节点上操作。

* ip=192.168.122.20 ## 需要修改为业务集群任意一个 master 节点的外网 ip，若没有外网地址，使用默认的实际ip  
  ACP\_NAMESPACE=cpaas-system ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
  cat << eof > /tmp/prometheus-feature.yaml  
  apiVersion: infrastructure.alauda.io/v1alpha1  
  kind: Feature  
  metadata:  
   labels:  
   instanceType: prometheus  
   type: monitoring  
   name: prometheus  
  spec:  
   accessInfo:  
   grafanaAdminPassword: admin # grafana 默认密码  
   grafanaAdminUser: admin # grafana 默认用户  
   grafanaUrl: http://$ip:30902 # grafana 地址  
   name: kube-prometheus # 安装 kube-prometheus chart 时指定的名称  
   namespace: ${ACP\_NAMESPACE} # kube-prometheus chart 所在的命名空间  
   prometheusTimeout: 10 # prometheus 请求超时时间  
   prometheusUrl: http://$ip:30900 # prometheus 地址  
   instanceType: prometheus  
   type: monitoring  
   version: "1.0"  
  eof  
     
  kubectl apply -f /tmp/prometheus-feature.yaml

1. 修改kubernetes 配置文件，监听网卡ip。

业务集群所有master节点

/etc/kubernetes/manifests/kube-scheduler.yaml

/etc/kubernetes/manifests/kube-controller-manager.yaml

删除以下配置文件中以下行内容

- --bind-address=**127.0.0.1**

再执行以下命令

kubectl edit cm -n kube-system kube-proxy

将其中

metricsBindAddress: 127.0.0.1:10249

改为

metricsBindAddress: 0.0.0.0:10249

然后重建所有kube-system下kube-proxy的pod

1. 清理操作(如果需要重装监控组件，则执行此操作)。

#清理操作

# kubectl delete -f /tmp/prometheus-feature.yaml

# helm delete --purge kube-prometheus

#如果使用本地路径方式部署，则还需要执行以下命令

# kubectl delete pvc prometheus-kube-prometheus-db-prometheus-kube-prometheus-0 -n cpaas-system

# kubectl delete pvc alertmanager-kube-prometheus-db-alertmanager-kube-prometheus-0 -n cpaas-system

# helm delete --purge prometheus-operator

### 业务集群可选组件

#### 业务集群部署 alb2

**说明**：在 UI 页面部署。

图片包含 屏幕截图

描述已自动生成页面-> comtainer platform平台->管理页面

图片包含 屏幕截图

描述已自动生成

定义名称，选择类型（如果选择的是高可用，地址栏则需要填写vip或者域名）

选择相应的标签，如果是高可用，必须先在需要部署负载均衡的节点上添加标签

#### nevermore

在 global 的第一台 master 节点上执行以下命令：

REGISTRY\_ENDPOINT=$(docker info |grep 60080 |tr -d ' ')  
DOMAIN\_NAME=192.168.122.10 ## 需要修改为 global 界面的访问地址   
REGION\_NAME=cls-n9wfggmk ## 想要部署 nevermore 的集群的名字  
ACP\_NAMESPACE=cpaas-system ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
https\_or\_http=http ## 默认值是 https，如果部署的时候使用了--use-http 参数，就需要改成 http  
token=$(kubectl describe secrets $(kubectl get secret -n kube-system | awk '/^clusterrole-aggregation-controller-token/{print $1}') -n kube-system|grep ^token|awk '{print $2}')  
IS\_OCP=false ## 默认是 false ，如果对接的是OCP集群，这个值需要改成true  
CONTAINER\_ENGINE=docker ## 默认是 docker，如果底层使用的是crio管理容器，这个值需要改成 crio  
  
mkdir /tmp/region\_helmrequest  
cat << EOF >/tmp/region\_helmrequest/${REGION\_NAME}-alauda-log-agent.yaml  
apiVersion: app.alauda.io/v1alpha1  
kind: HelmRequest  
metadata:  
 finalizers:  
 - captain.alauda.io  
 generation: 1  
 name: ${REGION\_NAME}-alauda-log-agent  
 namespace: ${ACP\_NAMESPACE}  
spec:  
 chart: stable/alauda-log-agent  
 namespace: ${ACP\_NAMESPACE}  
 releaseName: ${REGION\_NAME}-alauda-log-agent  
 clusterName: ${REGION\_NAME}  
 values:  
 global:  
 labelBaseDomain: cpaas.io  
 namespace: ${ACP\_NAMESPACE}  
 isOCP: ${IS\_OCP}  
 registry:  
 address: ${REGISTRY\_ENDPOINT}  
 nevermore:  
 apiGatewayHost: ${https\_or\_http}://${DOMAIN\_NAME}  
 token: ${token}  
 region: ${REGION\_NAME}  
 containerEngine: ${CONTAINER\_ENGINE}  
 version: $(helm search | grep '^stable/alauda-log-agent ' | awk '{print $2}')  
EOF  
kubectl create -f /tmp/region\_helmrequest/${REGION\_NAME}-alauda-log-agent.yaml

#清理操作

# kubectl delete -f /tmp/region\_helmrequest/${REGION\_NAME}-alauda-log-agent.yaml

#### nginx-ingress

**说明**：

1. 在集群内选择运行 ingress 的节点，建议选择让 ingress 运行在 master 节点上。在 master 节点上，添加 ingress=true 的 label，可以在要部署 ingress 的 master 节点上执行 for i in $(kubectl get no | awk '{if ($3 == "master")print $1}'); do echo kubectl label nodes $i ingress=true --overwrite ; done这个命令，即可给所有的 master 节点上打这个标签。
2. 在 global 的第一台 master 节点上执行以下命令：

* REGISTRY\_ENDPOINT=$(docker info |grep 60080 |tr -d ' ')  
  REGION\_NAME=demo1 ## 想要部署 ingress 的集群的名字  
  ACP\_NAMESPACE=cpaas-system ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
  replicas=3 ## ingress 的副本数，默认1，我们建议让 ingress 部署在 master 节点上，可以在业务集群的 master 节点上执行下面的命令，获取副本数 kubectl get no | awk '{if ($3 == "master")print $1}' | wc -l  
    
  mkdir /tmp/region\_helmrequest  
  cat << EOF >/tmp/region\_helmrequest/${REGION\_NAME}-nginx-ingress.yaml  
  apiVersion: app.alauda.io/v1alpha1  
  kind: HelmRequest  
  metadata:  
   finalizers:  
   - captain.alauda.io  
   generation: 1  
   name: ${REGION\_NAME}-nginx-ingress  
   namespace: ${ACP\_NAMESPACE}  
  spec:  
   chart: stable/nginx-ingress  
   namespace: ${ACP\_NAMESPACE}  
   releaseName: ${REGION\_NAME}-nginx-ingress  
   clusterName: ${REGION\_NAME}  
   values:  
   global:  
   replicas: ${replicas}  
   labelBaseDomain: cpaas.io  
   namespace: ${ACP\_NAMESPACE}  
   registry:  
   address: ${REGISTRY\_ENDPOINT}  
   version: $(helm search | grep '^stable/nginx-ingress ' | awk '{print $2}')  
    
  EOF  
  kubectl create -f /tmp/region\_helmrequest/${REGION\_NAME}-nginx-ingress.yaml

#### cron-hpa-controller【定时扩缩容的组件，可不部署】

在 global 的第一台 master 节点上执行以下命令：

REGISTRY\_ENDPOINT=$(docker info |grep 60080 |tr -d ' ')  
REGION\_NAME=<name> ## 想要部署 cron-hpa-controller 的集群的名字  
ACP\_NAMESPACE=<cpaas-system> ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
  
mkdir /tmp/region\_helmrequest  
cat << EOF >/tmp/region\_helmrequest/${REGION\_NAME}-cron-hpa-controller.yaml  
apiVersion: app.alauda.io/v1alpha1  
kind: HelmRequest  
metadata:  
 finalizers:  
 - captain.alauda.io  
 generation: 1  
 name: ${REGION\_NAME}-cron-hpa-controller  
 namespace: ${ACP\_NAMESPACE}  
spec:  
 chart: stable/cron-hpa-controller  
 namespace: ${ACP\_NAMESPACE}  
 releaseName: ${REGION\_NAME}-cron-hpa-controller  
 clusterName: ${REGION\_NAME}  
 values:  
 global:  
 namespace: ${ACP\_NAMESPACE}  
 registry:  
 address: ${REGISTRY\_ENDPOINT}  
 version: $(helm search | grep '^stable/cron-hpa-controller ' | awk '{print $2}')  
  
EOF  
kubectl create -f /tmp/region\_helmrequest/${REGION\_NAME}-cron-hpa-controller.yaml

#### metrics-server 【devops 自动扩缩容需要用到这个组件】

在想要部署 metrics-server 的集群的第一台 master 节点上执行以下命令：

REGISTRY\_ENDPOINT=$(docker info |grep 60080 |tr -d ' ')  
ACP\_NAMESPACE=cpaas-system ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
  
cat <<EOF>/tmp/metrics-server.yaml  
apiVersion: extensions/v1beta1  
kind: Deployment  
metadata:  
 name: metrics-server  
 namespace: ${ACP\_NAMESPACE}  
 labels:  
 k8s-app: metrics-server  
spec:  
 selector:  
 matchLabels:  
 k8s-app: metrics-server  
 template:  
 metadata:  
 name: metrics-server  
 labels:  
 k8s-app: metrics-server  
 spec:  
 containers:  
 - name: metrics-server  
 image: ${REGISTRY\_ENDPOINT}/alaudak8s/metrics-server-amd64:v0.3.1  
 imagePullPolicy: IfNotPresent  
 command:  
 - /metrics-server  
 - --kubelet-preferred-address-types=InternalIP  
 - --kubelet-insecure-tls  
 volumeMounts:  
 - name: tmp-dir  
 mountPath: /tmp  
 volumes:  
 # mount in tmp so we can safely use from-scratch images and/or read-only containers  
 - name: tmp-dir  
 emptyDir: {}  
   
---  
apiVersion: v1  
kind: Service  
metadata:  
 name: metrics-server  
 namespace: ${ACP\_NAMESPACE}  
 labels:  
 kubernetes.io/name: "Metrics-server"  
spec:  
 selector:  
 k8s-app: metrics-server  
 ports:  
 - port: 443  
 protocol: TCP  
 targetPort: 443  
---  
apiVersion: apiregistration.k8s.io/v1beta1  
kind: APIService  
metadata:  
 name: v1beta1.metrics.k8s.io  
spec:  
 service:  
 name: metrics-server  
 namespace: ${ACP\_NAMESPACE}  
 group: metrics.k8s.io  
 version: v1beta1  
 insecureSkipTLSVerify: true  
 groupPriorityMinimum: 100  
 versionPriority: 100  
EOF  
kubectl create -f /tmp/metrics-server.yaml

#### Container-platform（容器管理平台）

**说明**：

* <> 代表要根据环境情况替换尖括号内的值，替换完毕之后，不包括尖括号。
* 如果在部署 global 的时候，--enabled-features 这个参数没有加上 acp ，需要手动部署容器管理平台的 globla 组件。

在 global 集群的第一个 master 节点上执行以下命令：

ACP\_NAMESPACE=<cpaas-system> ## 改成部署平台时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
REGISTRY\_ENDPOINT =$(docker info |grep 60080 |tr -d ' ')  
kubectl captain create \  
 --version $(helm search | grep '^stable/alauda-container-platform' | awk '{print $2}') \  
 --configmap=acp-config \  
 --namespace=${ACP\_NAMESPACE} \  
 --chart=stable/alauda-container-platform \  
 alauda-container-platform  
   
kubectl captain create \  
 --version $(helm search | grep '^stable/public-chart-repo' | awk '{print $2}') \  
 --set global.registry.address=${REGISTRY\_ENDPOINT} \  
 --set global.namespace=${ACP\_NAMESPACE} \  
 --namespace=${ACP\_NAMESPACE} \  
 --chart=stable/public-chart-repo \  
 public-chart-repo

#### devops

**说明**：

* <> 代表要根据环境情况替换尖括号内的值，替换完毕之后，不包括尖括号。
* 如果在部署 global 的时候，--enabled-features 这个参数没有加上 devops ，需要手动部署 devops 的 globla 组件。

在 global 集群的第一个 master 节点上执行以下命令：

ACP\_NAMESPACE=cpaas-system ## 改成部署平台时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
kubectl captain create \  
 --version $(helm search | grep '^stable/alauda-devops ' | awk '{print $2}') \  
 --configmap=acp-config \  
 --namespace=${ACP\_NAMESPACE} \  
 --chart=stable/alauda-devops \  
 alauda-devops

##### devops 所需的 harbor、jenkins、gitlab、nexus、sonarqube 第三方插件部署

**说明**：若使用 PVC 方式部署以下插件。需要事先准备好对应的 PVC，PVC 默认我们提供的是对接 cephfs，因此需要先确保集群对接了 ceph，然后创建对应的 PVC。

cat <<EOF > cephfs.yaml  
   
kind: PersistentVolumeClaim  
apiVersion: v1  
metadata:  
 name: claim1 ### 对应 pvc 的名字  
 namespace: default  
 annotations:  
 volume.beta.kubernetes.io/storage-class: cephfs  
 volume.beta.kubernetes.io/storage-provisioner: ceph.com/cephfs  
 finalizers:  
 - kubernetes.io/pvc-protection  
spec:  
 accessModes:  
 - ReadWriteMany  
 resources:  
 requests:  
 storage: 1Gi ###对应 pvc 的大小  
 volumeMode: Filesystem  
EOF  
   
kubectl create -f cephfs.yaml

###### 部署 jenkins

jenkins 的部署方式有以下四种，我们推荐在业务集群部署 jenkins，且最好以 PVC 的方式部署。

* 在 global 集群以本地目录的方式部署 jenkins，（对接了GitLab）。
* NODE\_NAME="master-1" ###需要修改为集群中实际存放jenkins数据的某个节点的hostname  
  path="/root/alauda/jenkins" ###默认数据目录为/root/alauda/jenkins，若有需要可更改。  
  password="Jenkins12345" ###默认密码为Jenkins12345，若有需要可更改  
  REGISTRY=$(docker info |grep 60080 |tr -d ' ')  
  ACP\_NAMESPACE=<cpaas-system> ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
     
  cat <<EOF > values.yaml  
     
  global:  
   registry:  
   address: ${REGISTRY}  
  Master:  
   ServiceType: NodePort  
   NodePort: 32001  
   AdminPassword: "$password"  
   gitlabConfigs:  
   - name: <GitLab服务器名字>  
   manageHooks: true  
   serverUrl: <GitLab服务器地址>  
   token: <GitLab凭据token>  
   Location:  
   url: <Jenkins Location URL>  
  Persistence:  
   Enabled: false  
   Host:  
   NodeName: ${NODE\_NAME}  
   Path: $path  
  AlaudaACP:  
   Enabled: true  
  alaudapipelineplugin:  
   consoleURL: ""  
   apiEndpoint: ""  
   apiToken: ""  
   account: ""  
   spaceName: ""  
   clusterName: ""  
   namespace: ""  
  AlaudaDevOpsCredentialsProvider:  
   globalNamespaces: "${ACP\_NAMESPACE}-global-credentials,${ACP\_NAMESPACE}"  
  Erebus:  
   Namespace: "${ACP\_NAMESPACE}"  
   URL: "https://erebus.${ACP\_NAMESPACE}.svc.cluster.local:443/kubernetes"  
     
  EOF  
    
  helm install stable/jenkins --name jenkins --namespace default -f values.yaml
* 在 global 以 pvc 的方式部署 jenkins，（对接了GitLab）。
* password="Jenkins12345" ###默认密码为Jenkins12345，若有需要可更改  
  pvc\_name="jenkinspvc" ###默认pvc的名字为jenkinspvc，需要事先在default命名空间下准备好这个pvc，可更改，但也需要事先创建好对应的pvc。  
  REGISTRY=$(docker info |grep 60080 |tr -d ' ')  
  ACP\_NAMESPACE=<cpaas-system> ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
    
  cat <<EOF > values.yaml  
     
  global:  
   registry:  
   address: ${REGISTRY}  
  Master:  
   ServiceType: NodePort  
   NodePort: 32001  
   AdminPassword: "$password"  
   gitlabConfigs:  
   - name: <GitLab服务器名字>  
   manageHooks: true  
   serverUrl: <GitLab服务器地址>  
   token: <GitLab凭据token>  
   Location:  
   url: <Jenkins Location URL>  
  Persistence:  
   Enabled: true  
   ExistingClaim="$pvc\_name"  
  AlaudaACP:  
   Enabled: true  
  alaudapipelineplugin:  
   consoleURL: ""  
   apiEndpoint: ""  
   apiToken: ""  
   account: ""  
   spaceName: ""  
   clusterName: ""  
   namespace: ""  
  AlaudaDevOpsCredentialsProvider:  
   globalNamespaces: "${ACP\_NAMESPACE}-global-credentials,${ACP\_NAMESPACE}"  
  Erebus:  
   Namespace: "${ACP\_NAMESPACE}"  
   URL: "https://erebus.${ACP\_NAMESPACE}.svc.cluster.local:443/kubernetes"  
     
  EOF  
     
  helm install stable/jenkins --name jenkins --namespace default -f values.yaml
* 在业务集群以本地路径的方式部署 jenkins，（对接了GitLab）。
* NODE\_NAME="192.168.122.26" ###需要修改为集群中实际存放jenkins数据的某个节点的hostname  
  global\_vip="192.168.122.10" ###需要修改为平台的访问地址，如果访问地址是域名，就必须配置成域名，因为 jenkins 需要访问 global 平台的 erebus，如果平台是域名访问的话，erebus 的 ingress 策略会配置成只能域名访问。  
  path="/cpaas/data/jenkins" ###默认数据目录为/root/alauda/jenkins，若有需要可更改。  
  password="Jenkins12345" ###默认密码为Jenkins12345，若有需要可更改  
  REGISTRY=$(docker info |grep 60080 |tr -d ' ')  
  TOKEN=eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9..GITHW0UfSdIPsymcNzymKITX735U-aS6oNw1xJ5MU55e165Oi25aUekQ6AcS96sVhxTwdOtEn2U9NUuSF1dwcWk3CNW4kCqf-b0j0Lj1HDWLKLom1zGDJKmYZRgDAB1vFpFfvSPzA4O8j6zrbG0Bh3f1TK-rqHC4aqyKAQ6p5UTdyMAadKKKT6GF1NYBzrXbhW0bw5JpXoqIIv8-KjAOgI3SrjN62Nap9FYYLLiL3MifvtK0jch5s06LKi4yvAvOx2ocBZzBrWHqYMzpqwnhNrrxdWOCQfca7f29YJNbZ4FNGSOG2gylzADkaC8HgUsXRngOE8aXVCzZg-nqkq0csw
* ### 如何获取token，到devops-apiserver所在集群（一般为global集群）执行：ACP\_NAMESPACE=cpaas-system ; echo $(kubectl get secret -n ${ACP\_NAMESPACE} $(kubectl get secret -n ${ACP\_NAMESPACE} | grep devops-apiserver-token |awk '{print $1}') -o jsonpath={.data.token} |base64 --d)
* ACP\_NAMESPACE=cpaas-system ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
    
  cat <<EOF > values.yaml  
     
  global:  
   registry:  
   address: ${REGISTRY}  
  Master:  
   ServiceType: NodePort  
   NodePort: 32001  
   AdminPassword: "$password"  
   gitlabConfigs:  
   - name: gitlab  
   manageHooks: true  
   serverUrl: http://192.168.122.20:31101  
   token: xxxx  
   Location:  
   url: http://192.168.122.20:32001  
  Persistence:  
   Enabled: false  
   Host:  
   NodeName: ${NODE\_NAME}  
   Path: $path  
  AlaudaACP:  
   Enabled: false  
  alaudapipelineplugin:  
   consoleURL: ""  
   apiEndpoint: ""  
   apiToken: ""  
   account: ""  
   spaceName: ""  
   clusterName: ""  
   namespace: ""  
  AlaudaDevOpsCredentialsProvider:  
   globalNamespaces: "${ACP\_NAMESPACE}-global-credentials,${ACP\_NAMESPACE},kube-system"  
  AlaudaDevOpsCluster:  
   Cluster:  
   masterUrl: "https://$global\_vip:6443"  
   token: ${TOKEN}  
  Erebus:  
   Namespace: "${ACP\_NAMESPACE}"  
   URL: "https://$global\_vip:443/kubernetes"  
     
  EOF  
     
  helm install stable/jenkins --name jenkins --namespace default -f values.yaml

#清理操作

#helm delete --purge jenkins

#清除本地数据

* 在业务集群以 PVC 的方式部署 jenkins，（对接了GitLab）。
* global\_vip="1.1.1.1" ###需要修改为平台的访问地址，如果访问地址是域名，就必须配置成域名，因为 jenkins 需要访问 global 平台的 erebus，如果平台是域名访问的话，erebus 的 ingress 策略会配置成只能域名访问。  
  password="Jenkins12345" ###默认密码为Jenkins12345，若有需要可更改  
  pvc\_name="jenkinspvc" ###默认pvc的名字为jenkinspvc，需要事先在default命名空间下准备好这个pvc，可更改，但也需要事先创建好对应的pvc。  
  REGISTRY=$(docker info |grep 60080 |tr -d ' ')  
  TOKEN= ### 如何获取token，到devops-apiserver所在集群（一般为global集群）执行：echo $(kubectl get secret -n ${ACP\_NAMESPACE} $(kubectl get secret -n ${ACP\_NAMESPACE} | grep devops-apiserver-token |awk '{print $1}') -o jsonpath={.data.token} |base64 -d)  
  ACP\_NAMESPACE=<cpaas-system> ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
    
  cat <<EOF > values.yaml  
     
  global:  
   registry:  
   address: ${REGISTRY}  
  Master:  
   ServiceType: NodePort  
   NodePort: 32001  
   AdminPassword: "$password"  
   gitlabConfigs:  
   - name: <GitLab服务器名字>  
   manageHooks: true  
   serverUrl: <GitLab服务器地址>  
   token: <GitLab凭据token>  
   Location:  
   url: <Jenkins Location URL>  
  Persistence:  
   Enabled: true  
   ExistingClaim: "$pvc\_name"  
  AlaudaACP:  
   Enabled: false  
  alaudapipelineplugin:  
   consoleURL: ""  
   apiEndpoint: ""  
   apiToken: ""  
   account: ""  
   spaceName: ""  
   clusterName: ""  
   namespace: ""  
  AlaudaDevOpsCredentialsProvider:  
   globalNamespaces: "${ACP\_NAMESPACE}-global-credentials,${ACP\_NAMESPACE},kube-system"  
  AlaudaDevOpsCluster:  
   Cluster:  
   masterUrl: "https://$global\_vip:6443"  
   token: ${TOKEN}  
  Erebus:  
   Namespace: "${ACP\_NAMESPACE}"  
   URL: "https://$global\_vip:443/kubernetes"  
     
  EOF  
     
  helm install stable/jenkins --name jenkins --namespace default -f values.yaml
* 部署成功后到 Jenkins 修改配置访问集群任意一个节点 ip+32001 端口，访问 jenkins 页面，用户名：admin 密码：Jenkins12345。
* 单击 **系统管理->系统设置**：
* a. Alauda Jenkins Sync 里勾选 **启用** 按钮，添加 ***Jenkins 服务名称***，例如：jenkinstest，名称不能和别的已经部署的 jekins 服务名称一样（比如：global 里的 jenkins），界面集成的时候必须与这个名字一致。
* b. 到 Kubernetes Cluster Configuration 下的 Credentials，单击添加，添加一个 serviceaccount token （类型选择 secret text， Secret 位置输入token， ID 输入任意凭据名称（例如：test-token））。<br>如何获取 token？到 devops-apiserver 所在集群（一般为 global 集群）执行命令行：<br>echo $(kubectl get secret -n ${ACP\_NAMESPACE} $(kubectl get secret -n ${ACP\_NAMESPACE} | grep devops-apiserver-token |awk '{print $1}') -o jsonpath={.data.token} |base64 -d)。<br>添加之后。Credentials 选择刚才创建的 secret。
* c. 都配置成功后，在 Kubernetes Cluster Configuration 下单击 [Test Connection] 测试连接，提示 Connect to succeed 后，单击保存。

对接jenkins

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

名称非常重要：必须和部署jenkins chart时名称一致

图片包含 屏幕截图

描述已自动生成

绑定项目

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

密码为jenkins用户的token，token获取方法如下

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

###### 部署 gitlab

部署 gitlab 有以下两种方式，推荐部署在业务集群，且以 PVC 的方式部署。

* 以本地路径的方式部署。
* REGISTRY=$(docker info |grep 60080 |tr -d ' ')  
  NODE\_NAME=192.168.122.25 ###需要修改为选择部署gialab的节点的hostname  
  NODE\_IP="192.168.122.20" ###这个ip为gitlab的访问地址，需要修改为部署集群中任意master节点一个节点的实际ip  
  potal\_path="/cpaas/data/gitlab/portal" ###potal的数据目录，一般不需要修改，若有规划，可修改为别的目录  
  database\_path="/cpaas/data/gitlab/database" ###database的目录，一般不需要修改，若有规划，可修改为别的目录  
  redis\_path="/cpaas/data/gitlab/redis" ###redis的目录，一般不需要修改，若有规划，可修改为别的目录  
  helm install stable/gitlab-ce --name gitlab-ce --namespace default \  
   --set global.registry.address=${REGISTRY} \  
   --set portal.debug=true \  
   --set gitlabHost=${NODE\_IP} \  
   --set gitlabRootPassword=Gitlab12345 \  
   --set service.type=NodePort \  
   --set service.ports.http.nodePort=31101 \  
   --set service.ports.ssh.nodePort=31102 \  
   --set service.ports.https.nodePort=31103 \  
   --set portal.persistence.enabled=false \  
   --set portal.persistence.host.nodeName=${NODE\_NAME} \  
   --set portal.persistence.host.path="$potal\_path" \  
   --set portal.persistence.host.nodeName="${NODE\_NAME}" \  
   --set database.persistence.enabled=false \  
   --set database.persistence.host.nodeName=${NODE\_NAME} \  
   --set database.persistence.host.path="$database\_path" \  
   --set database.persistence.host.nodeName="${NODE\_NAME}" \  
   --set redis.persistence.enabled=false \  
   --set redis.persistence.host.nodeName=${NODE\_NAME} \  
   --set redis.persistence.host.path="$redis\_path" \  
   --set redis.persistence.host.nodeName="${NODE\_NAME}"

清除操作

helm delete -purge gitlab-ce

#清除本地数据

* PVC 的方式部署 gitlab
* NODE\_IP=1.1.1.1 ###此参数为部署时指定的访问地址，写当前集群中任意一个master节点的ip即可  
  portal\_pvc="portalpvc" ###默认pvc的名字为portalpvc，需要事先在default命名空间下准备好这个pvc，可更改，但也需要事先创建好对应的pvc。  
  database\_pvc="databasepvc" ###默认pvc的名字为databasepvc，需要事先在default命名空间下准备好这个pvc，可更改，但也需要事先创建好对应的pvc。  
  redis\_pvc="redispvc" ###默认pvc的名字为redispvc，需要事先在default命名空间下准备好这个pvc，可更改，但也需要事先创建好对应的pvc。  
  REGISTRY=$(docker info |grep 60080 |tr -d ' ')  
  helm install stable/gitlab-ce --name gitlab-ce --namespace default \  
   --set global.registry.address=${REGISTRY} \  
   --set portal.debug=true \  
   --set gitlabHost=${NODE\_IP} \  
   --set gitlabRootPassword=Gitlab12345 \  
   --set service.type=NodePort \  
   --set service.ports.http.nodePort=31101 \  
   --set service.ports.ssh.nodePort=31102 \  
   --set service.ports.https.nodePort=31103 \  
   --set portal.persistence.enabled=true \  
   --set portal.persistence.existingClaim=$portal\_pvc \  
   --set database.persistence.enabled=true \  
   --set database.persistence.existingClaim=$database\_pvc \  
   --set redis.persistence.enabled=true \  
   --set redis.persistence.existingClaim=$redis\_pvc \
* 部署完成之后，通过 NODE\_IP+31101 端口访问 gitlab，默认用户名为：root，密码为：Gitlab12345。

对接gitlab

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

访问地址与API地址都填写一致。

绑定项目

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成图片包含 屏幕截图

描述已自动生成

密码获取方法

登录gitlab

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

分配仓库

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

###### 部署 harbor 镜像仓库

部署 harbor 有以下两种部署方式，推荐部署在业务集群，且以 PVC 的方式部署。

* 本地目录的方式部署 harbor。
* REGISTRY=$(docker info |grep 60080 |tr -d ' ')  
  NODE\_IP="192.168.122.20" ###此参数为部署时指定的访问地址，写当前集群中任意一个master节点的ip即可  
  NODE\_NAME="192.168.122.24" ###需要修改为选择部署harbor节点的hostname  
  HOST\_PATH=/cpaas/data/harbor ###这个目录为harbor的数据目录路径，一般不需要修改，若有别的规划，可修改。  
  harbor\_password="Harbor12345" ####harbor的密码，默认不需要修改，若有规划，可改  
  db\_password="Harbor4567" ####harbor数据库的密码，默认不需要修改，若有规划，可改  
  redis\_password="Redis789" ###harbor的redis的密码，默认不需要修改，若有规划，可改  
  helm install --name harbor --namespace default stable/harbor \  
   --set global.registry.address=${REGISTRY} \  
   --set externalURL=http://${NODE\_IP}:31104 \  
   --set harborAdminPassword=$harbor\_password \  
   --set ingress.enabled=false \  
   --set service.type=NodePort \  
   --set service.ports.http.nodePort=31104 \  
   --set service.ports.ssh.nodePort=31105 \  
   --set service.ports.https.nodePort=31106 \  
   --set database.password=$db\_password \  
   --set redis.usePassword=true \  
   --set redis.password=$redis\_password \  
   --set database.persistence.enabled=false \  
   --set database.persistence.host.nodeName=${NODE\_NAME} \  
   --set database.persistence.host.path=${HOST\_PATH}/database \  
   --set redis.persistence.enabled=false \  
   --set redis.persistence.host.nodeName=${NODE\_NAME} \  
   --set redis.persistence.host.path=${HOST\_PATH}/redis \  
   --set chartmuseum.persistence.enabled=false \  
   --set chartmuseum.persistence.host.nodeName=${NODE\_NAME} \  
   --set chartmuseum.persistence.host.path=${HOST\_PATH}/chartmuseum \  
   --set registry.persistence.enabled=false \  
   --set registry.persistence.host.nodeName=${NODE\_NAME} \  
   --set registry.persistence.host.path=${HOST\_PATH}/registry \  
   --set jobservice.persistence.enabled=false \  
   --set jobservice.persistence.host.nodeName=${NODE\_NAME} \  
   --set jobservice.persistence.host.path=${HOST\_PATH}/jobservice \  
   --set AlaudaACP.Enabled=false

#清除操作

#helm delete --purge harbor

#清除本地数据

* PVC 的方式部署 harbor。
* REGISTRY=$(docker info |grep 60080 |tr -d ' ')  
  NODE\_IP="1.1.1.1" ######此参数为部署时指定的访问地址，写当前集群中任意一个master节点的ip即可  
  database\_pvc=habordatabase ###harbor数据库使用的pvc，需要事先在default下创建这个pvc  
  redis\_pvc=harborredis ###harbor的redis使用的pvc，需要事先在default下创建这个pvc  
  chartmuseum\_pvc=harbormuseum ###harbor使用的pvc，需要事先在default下创建这个pvc  
  registry\_pvc=harborregistry ###harbor的registry使用的pvc，需要事先在default下创建这个pvc  
  jobservice\_pvc=harborjob ###harbor使用的pvc，需要事先在default下创建这个pvc  
  harbor\_password="Harbor12345" ####harbor的密码，默认不需要修改，若有规划，可改  
  db\_password="Harbor4567" ####harbor数据库的密码，默认不需要修改，若有规划，可改  
  redis\_password="Redis789" ###harbor的redis的密码，默认不需要修改，若有规划，可改  
  helm install --name harbor --namespace default stable/harbor \  
   --set global.registry.address=${REGISTRY} \  
   --set externalURL=http://${NODE\_IP}:31104 \  
   --set harborAdminPassword=$harbor\_password \  
   --set ingress.enabled=false \  
   --set service.type=NodePort \  
   --set service.ports.http.nodePort=31104 \  
   --set service.ports.ssh.nodePort=31105 \  
   --set service.ports.https.nodePort=31106 \  
   --set database.password=$db\_password \  
   --set redis.usePassword=true \  
   --set redis.password=$redis\_password \  
   --set database.persistence.enabled=true \  
   --set database.persistence.existingClaim=${database\_pvc} \  
   --set redis.persistence.enabled=true \  
   --set redis.persistence.existingClaim=${redis\_pvc} \  
   --set chartmuseum.persistence.enabled=true \  
   --set chartmuseum.persistence.existingClaim=${chartmuseum\_pvc} \  
   --set registry.persistence.enabled=true \  
   --set registry.persistence.existingClaim=${registry\_pvc} \  
   --set jobservice.persistence.enabled=true \  
   --set jobservice.persistence.existingClaim=${jobservice\_pvc} \  
   --set AlaudaACP.Enabled=false \
* 部署完成后通过传入的部署 ip+31104 端口访问 harbor。默认用户名为：admin，密码为：Harbor12345。

对接harbor

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

绑定项目

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

分配仓库

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

###### 部署 nexus(可选)

存储方式：host path 或 PVC（推荐）。

* 以 hostPath 方式部署。
* NODE\_IP=<要集群的 master 的 ip，nexus 暴露的端口是 nodePorts，集群内所有服务器都可以，但 master 节点最好，因为不用担心 master 节点被删除>  
   NODE\_NAME=node1 ## 集群中实际存放 nexus 数据的某个节点的hostname，也就是定点部署 nexus 的节点的 hostname  
   HOST\_PATH=/alauda/nexus  
   REGISTRY=$(docker info |grep 60080 |tr -d ' ')  
   helm install stable/nexus --name nexus \  
   --set global.registry.address=${REGISTRY} \  
   --set nexus.service.nodePort=32010 \  
   --set nexusProxy.env.nexusHttpHost=${NODE\_IP} \  
   --set nexusProxy.env.nexusDockerHost=${NODE\_IP} \  
   --set persistence.host.nodeName=${NODE\_NAME} \  
   --set persistence.host.path=${HOST\_PATH}
* 以 StorageClass 方式，自动创建 PVC 部署, 需要提前创建好对应 STORAGE\_CLASS 的 PV。
* NODE\_IP=<要集群的 master 的 ip，nexus 暴露的端口是 nodePorts，集群内所有服务器都可以，但 master 节点最好，因为不用担心 master 节点被删除>  
   NEXUS\_PVC=nexus-pvc ## 给 nexus 使用的 pvc 的名字  
   REGISTRY=$(docker info |grep 60080 |tr -d ' ')  
   helm install stable/nexus --name nexus \  
   --set global.registry.address=${REGISTRY} \  
   --set nexus.service.nodePort=32010 \  
   --set nexusProxy.env.nexusHttpHost=${NODE\_IP} \  
   --set nexusProxy.env.nexusDockerHost=${NODE\_IP} \  
   --set persistence.enabled=true  
   --set persistence.existingClaim=${NEXUS\_PVC}
* 已经存在 PVC 时，可以直接用 PVC 部署，需要提前创建好对应 PVC。
* NODE\_IP=<要集群的 master 的 ip，nexus 暴露的端口是 nodePorts，集群内所有服务器都可以，但 master 节点最好，因为不用担心 master 节点被删除>  
   NEXUS\_PVC=nexus-pvc  
   REGISTRY=$(docker info |grep 60080 |tr -d ' ')  
   helm install . --name nexus \  
   --set global.registry.address=${REGISTRY} \  
   --set nexus.service.nodePort=32010 \  
   --set nexusProxy.env.nexusHttpHost=${NODE\_IP} \  
   --set nexusProxy.env.nexusDockerHost=${NODE\_IP} \  
   --set persistence.enabled=true  
   --set persistence.existingClaim=${NEXUS\_PVC}
* **说明**：如需开启 nexusBackup，需要--set nexusBackup.enabled=true，以及 nexusBackup 对应的 persistence，同上方式，具体参数见 chart 中 README.md。

默认帐号 admin 的密码由 nexus 随机生成，需要到 pod 中 cat /nexus-data/admin.password，第一次登录后，会要求更改密码。

###### 部署 sonarqube(可选)

sonarqube 版本：7.9.1-community

在安装 chart 前，请先确认将用哪种方式来存储数据：

* Persistent Volume Claim (建议)
* Host path

如果 Kubernetes 集群已经有可用的 StorageClass 和 provisioner，在安装 chart 过程中会自动创建 PVC 来存储数据。 想了解更多关于 StorageClass 和 PVC 的内容，可以参考 Kubernetes 官方文档。

* **参考命令（使用 PVC）：**
* registry=$(docker info |grep 60080 |tr -d ' ')  
  ACP\_NAMESPACE=<cpaas-system> ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
  helm install stable/sonarqube \  
   --name sonarqube \  
   --set plugins.useDefaultPluginsPackage=true \  
   --set global.registry.address=$registry \  
   --namespace=${ACP\_NAMESPACE} \  
   --set global.namespace=${ACP\_NAMESPACE} \  
   --set service.type=NodePort \  
   --set service.nodePort=<node port 端口号，默认31342> \  
   --set postgresql.database.persistence.enabled=true \  
   --set postgresql.database.persistence.existingClaim=<pvc name> ## 给 sonarqube 使用的 pvc 的名字
* **参考命令（使用 storageClass）：**
* registry=$(docker info |grep 60080 |tr -d ' ')  
  ACP\_NAMESPACE=<cpaas-system> ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
  helm install stable/sonarqube \  
   --name sonarqube \  
   --set plugins.useDefaultPluginsPackage=true \  
   --set global.registry.address=$registry \  
   --namespace=${ACP\_NAMESPACE} \  
   --set global.namespace=${ACP\_NAMESPACE} \  
   --set service.type=NodePort \  
   --set service.nodePort=<node port 端口号，默认31342> \  
   --set postgresql.database.persistence.enabled=true \  
   --set postgresql.database.persistence.storageClass=<storageclass name> ## 给 sonarqube 使用的 storageclass 的名字
* **参考命令（使用 hostpath）：**
* registry=$(docker info |grep 60080 |tr -d ' ')  
  ACP\_NAMESPACE=<cpaas-system> ## 改成部署时， --acp2-namespaces 参数指定的值，默认是cpaas-system  
  helm install stable/sonarqube \  
   --name sonarqube \  
   --set plugins.useDefaultPluginsPackage=true \  
   --set global.registry.address=$registry \  
   --namespace=${ACP\_NAMESPACE} \  
   --set global.namespace=${ACP\_NAMESPACE} \  
   --set service.type=NodePort \  
   --set service.nodePort=<node port 端口号，默认31342> \  
   --set postgresql.database.persistence.enabled=false \  
   --set postgresql.database.persistence.host.nodeName=<node name> \  
   --set postgresql.database.persistence.host.path=<path > ## 本地路径，比如：/cpaas/data/sonarqube

# 第四部分 平台配置及测试

## 检查部署是否成功

### 检查 global 平台是否部署成功

**说明**：global 组件通过 helm 和 captain 来部署，生成的资源全部都存放在运行 global 组件的 Kubernetes 集群内。

在运行 global 组件的 Kubernetes 集群的 master 节点上执行以下命令：

## 检查 helm 部署的 chart 是否成功，执行如下命令查找部署失败的 chart   
helm list --failed   
  
## 检查 captain 部署的 chart 是否成功，执行如下命令查找部署失败的 chart  
kubectl get helmrequest --all-namespaces | awk '{if ($NF != "Synced")print}'  
  
## 检查 catpain 部署的 chart release 是否成功，执行如下命令查找失败的 release  
kubectl get rel --all-namespaces | awk '{if ($3 != "deployed")print}'  
  
## 检查 所有 pod 是否正常，执行如下命令查找失败的 pod  
kubectl get pod --all-namespaces | awk '{if ($4 != "Running" && $4 != "Completed")print}' | awk -F'[/ ]+' '{if ($3 != $4)print}'

### 检查业务服务集群是否部署成功

**说明**：业务服务集群的组件通过 helm 和 captain 来部署，生成的 helmrequest 资源存放在运行 global 组件的 Kubernetes 集群内，captain 部署的 chart 产生的 release 资源存放在业务服务集群内。

在运行 global 组件的 Kubernetes 集群的 master 节点上和业务服务集群的 master 节点上执行以下命令：

## 检查 helm 部署的 chart 是否成功，执行如下命令查找部署失败的 chart ，在业务服务集群的 master 节点上操作  
helm list --failed   
  
## 检查 captain 部署的 chart 是否成功，执行如下命令查找部署失败的 chart，在运行 glboal 组件的 k8s 集群的 master 节点上操作  
kubectl get helmrequest --all-namespaces | awk '{if ($NF != "Synced")print}'  
  
## 检查 catpain 部署的 chart release 是否成功，执行如下命令查找失败的 release，在业务服务集群的 master 节点上操作  
kubectl get rel --all-namespaces | awk '{if ($3 != "deployed")print}'  
  
## 检查 所有 pod 是否正常，执行如下命令查找失败的 pod，在业务服务集群的 master 节点上操作  
kubectl get pod --all-namespaces | awk '{if ($4 != "Running" && $4 != "Completed")print}' | awk -F'[/ ]+' '{if ($3 != $4)print}'

## 平台访问地址及登录用户名、密码，以及兼容的浏览器

### 平台访问地址

**all in one 部署方案** 和客户没有提供域名的 **poc 部署方案** 的访问地址：

* 没有使用 --use-http 参数的平台访问地址：https://<init 节点 ip>
* 使用了 --use-http 参数的平台访问地址：http://<init 节点 ip>

**正式生产环境部署方案** 和客户提供域名的 **poc 部署方案** 的访问地址：

* 没有使用 --use-http 参数的平台访问地址：https://<--domain-name 参数指定的地址>
* 使用了 --use-http 参数的平台访问地址：http://<--domain-name 参数指定的地址>

### 平台登录权限

用户名：--root-username 参数指定的值，默认为 admin@cpaas.io。

密码：password

### 兼容的浏览器版本

**仅兼容 Chrome 浏览器，且是最近的 3 个版本之一**。

## 部署成功后操作

### 备份 etcd 数据

1. 拷贝安装目录下的 backup\_recovery.sh 和 function.sh 脚本到每一个 master 服务器的 /tmp 下。
2. 在每一台 master 节点上执行以下命令：

* cd /tmp  
  bash backup\_recovery.sh run\_function=back\_k8s\_file

### 添加 gc 脚本

将安装目录下的 gc.sh 脚本 cp 到各个服务器 /root 目录下，然后为每个服务器添加 crontab ，每天凌晨运行一次即可，用于清理系统日志、本机没有容器使用的镜像和处在 Terminating、Evicted、Completed 这三种状态一天以上的 pod。

执行如下命令：

echo '1 2 \* \* \* bash /tmp/gc.sh' | crontab

**注意**： 上述命令会将原来的 crontab 全部删掉，建议使用它 crontab -e 命令添加。

# 第五部分 FAQ

## 平台部署错误

### 安装 chart 失败

**提示信息**： install chart error，安装日志里会显示出错的 chart 名字。

**出错原因**： 脚本通过 helm install 命令部署 chart 出错，详细日志会保存到第一台 master 节点的/cpaas/chart\_install.log 文件内。

**错误处理流程**：首先登录到第一台 master 节点上，执行 helm list -a 命令检查 chart ，发现错误的 chart 后，执行helm delete --purge <chart name> 删掉 chart ，然后执行bash /tmp/install\_chart.sh 继续安装 chart，如果还是不成功，建议清空环境重新部署。依旧出错，请反馈给运维。

### 对接 global 集群失败

**提示信息**：get global token Repeat 10 times to get global token failed，之后对接 global 集群也会失败，提示 access global region error。

**出错原因**：这种情况一般是chart 安装失败，global 的 chart 没起来，或者因为资源不够、global 集群的服务器速度太慢，获取 token 的时候 global 组件还未正常运行，所以获取token 失败了，没有正确的 token，global 没起来调用 api 自然失败。

**错误处理流程**：chart 安装失败，请参照上面的方法处理，等待所有 global 组件的 pod 状态都正常后，去 init 节点上，执行以下命令行后对接成功：

cat /tmp/add\_global\_region.sh | sed 's/Bearer "/Bearer '$(bash /tmp/get\_token.sh )'"/g' >/tmp/dd.sh ; bash /tmp/dd.sh