

# 格言格语

## 活字格集群负载均衡方案

开始学习 →

联系专家 🧑

备案号：[陕ICP备2023004798号-2](#)

Released under the MIT License.

Copyright © 2019-present GrapeCity

# 简介

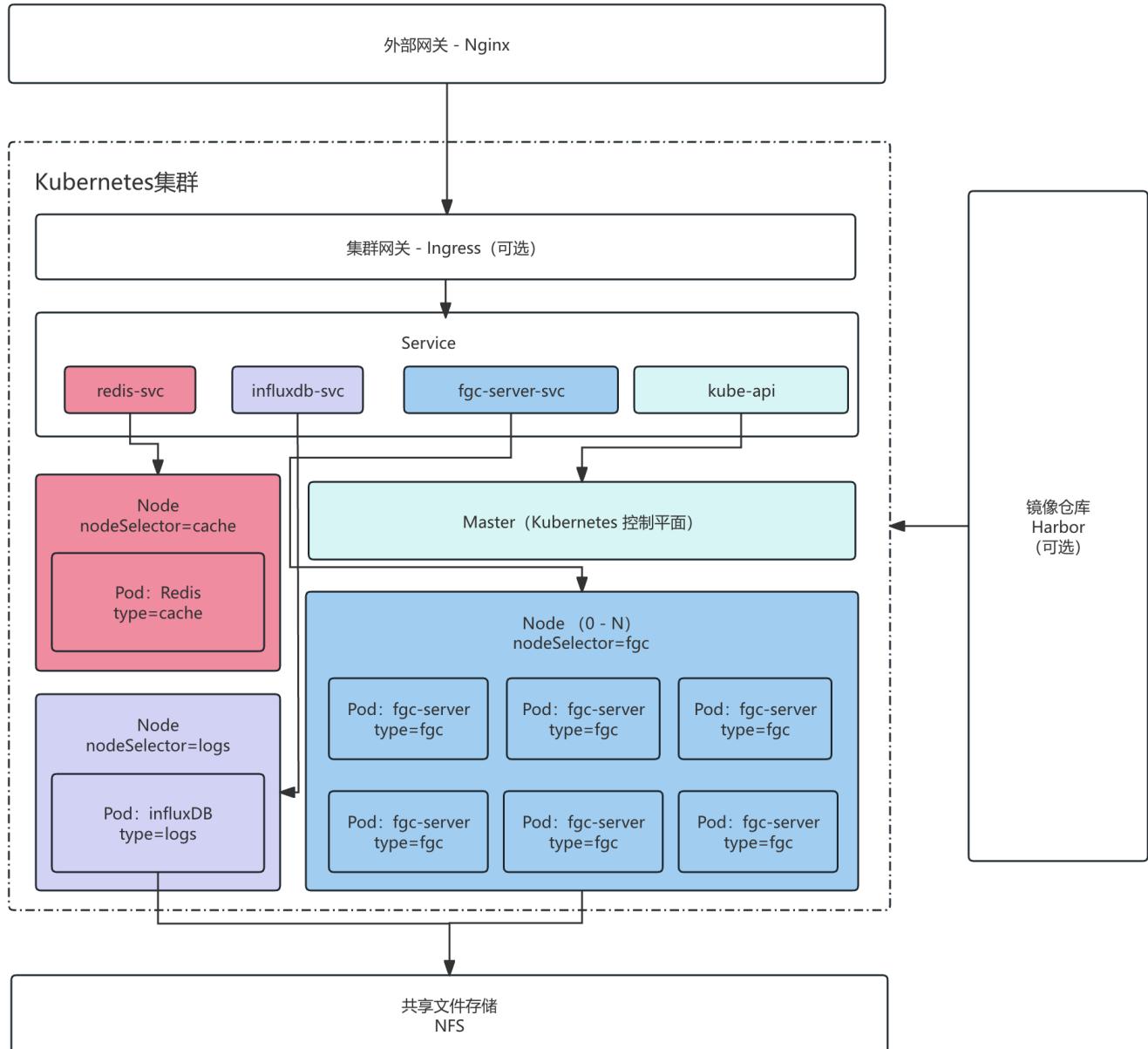
负载均衡（Load Balance）是一种将工作任务分摊到多个操作单元上进行执行的技术，是集群技术（Cluster）的一种应用。通过将负载（工作任务）进行平衡、分摊到多个操作单元上进行运行，可以提高并发处理能力，从而提高应用处理性能。

负载均衡技术的基本原理是通过运行在前端的负载均衡服务器，根据执行的负载均衡算法，将流量分配到后端服务器上，从而提高整个系统的扩展能力，实现服务的并行扩展，同时，负载均衡技术还可以起到对外网屏蔽内网服务器，从而提高系统的可用性。

---

## 活字格负载均衡架构

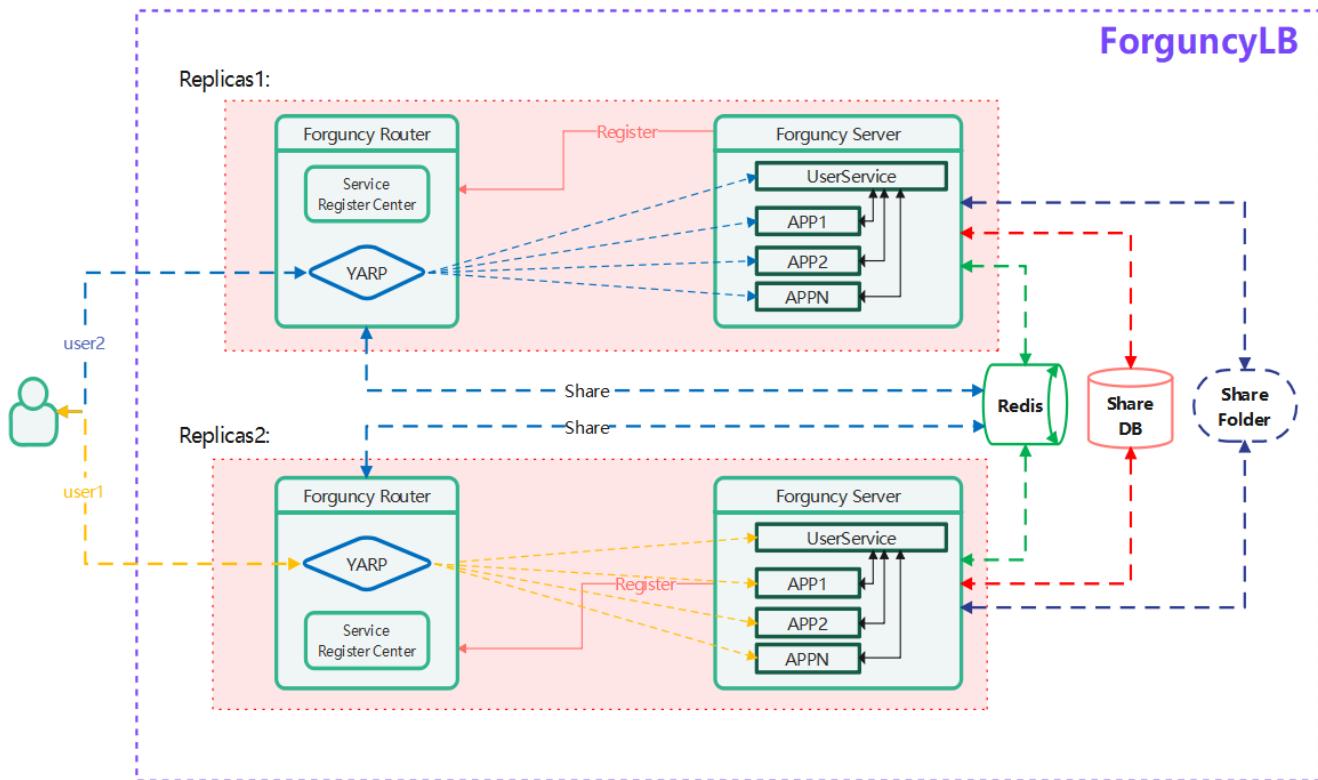
### 集群整体架构



- 活字格集群的底层平台基座为 Kubernetes。
- 活字格集群包含 3 个服务，分别是：
  - **fgc-server**：主服务。
  - **redis**：缓存共享服务。
  - **influxDb**：日志数据库服务。
- 活字格集群需要共享文件服务来保证集群数据的共享与持久化。
- 如果有内网需求，请配置私有化的镜像仓库。
- 为保证集群安全与高可用，推荐集群外部架设网关服务。

## 活字格服务 Pod 架构

## ForguncyLB



- 活字格的应用运行在 Pod 内部的 fgc-server 中。
- 活字格服务器（fgc-server）通过统一的路由（Forguncy Router）对外暴露端口（默认端口号 80），路由将所有请求到端口 80 的流量路由到不同的 App 以及 UserService。
- 活字格负载均衡基于会话保持功能。

# 环境要求

本节介绍安装活字格集群之前的初始环境准备工作。

## 前提条件

- 您需要准备 **至少 2 台** Linux 服务器作为集群节点。
- 请将准备的所有节点归属在同一子网下，确保互相之间可以正常通信并通过 **SSH** 服务进行登录。
- 集群节点的操作系统和版本请保持一致

### 操作系统建议

操作系统请遵照活字格的推荐操作系统进行安装：

- Ubuntu 16.04 / 18.04 / 22.04 / 24.04
- RedHat 7.6
- CentOS 7
- 中标麒麟 v7 update6
- 银河麒麟 v10 SP2 / SP3

以上系统均为 64 位。x86 和 arm 架构均可。

- 请确保在所有集群节点上都可以使用 **sudo**、**curl** 和 **openssl** 命令。

## 防火墙规则

如果您的基础设施环境已启用防火墙，您需要在防火墙设置中放行所需的端口和协议。如果您的基础设施环境未启用防火墙，您可以跳过此步骤。

服务	协议	起始端口	结束端口	备注
ssh	TCP	22		
etcd	TCP	2379	2380	Kubernetes 所需端口
apiserver	TCP	6443		Kubernetes 所需端口
calico	TCP	9099	9100	CNI 所需端口
bgp	TCP	179		
nodeport	TCP	30000	32767	Kubernetes 所需端口
master	TCP	10250	10258	Kubernetes 所需端口
dns	TCP	53		
dns	UDP	53		
metrics-server	TCP	8443		集群监控所需端口
forguncy-server	TCP	22345		活字格集群所需端口

此外，在离线安装环境下，您还需关注如下服务端口：

- 私有化镜像仓库的端口；
- 内网的私有化包管理仓库端口；
- ElasticSearch 与可视化界面 kibana 的端口。

### 建议

集群安装过程较为复杂且依赖服务较多，建议关闭防火墙进行安装是更好的选择。安装完成后，再依次配置防火墙策略。

## 高可用性

### Kubernetes

在生产环境中，为确保集群基础平台 Kubernetes 具备高可用性，建议准备 **至少 5 台 Linux 服务器**，其中 3 台作为控制平面节点，另外 2 台作为工作节点。

- 控制平面（Control Plane）是 Kubernetes 的核心组件，包括 `etcd`、`kube-apiserver`、`kube-scheduler` 和 `kube-controller-manager` 等。为了确保高可用，控制平面需要冗余部署。
- 数据平面（Data Plane）是运行实际工作负载的节点，也称为工作节点。活字格的服务最终会运行在工作节点上。因此，为了高可用，推荐 **至少 2 个工作节点**，这样即便一个节点故障，活字格的 Pod 服务也可以在另一个节点上恢复运行。

#### 节点数量说明

对于控制平台来说，尽管 **至少 2 个节点** 就能够实现冗余，但推荐 3 个节点与 `etcd` 保持一致。作为 Kubernetes 的状态数据存储组件，`etcd` 在集群中至少需要 3 个节点，才能通过分布式一致性算法（Raft）进行选举，从而实现高可用。奇数节点能够确保单点故障下保持可用。

## 计算与存储资源

- 在生产环境中，为确保集群具有足够的计算和存储资源，建议每台集群节点配置 **至少 8 个 CPU 核心、16 GB 内存和 200 GB 磁盘空间**。除此之外，建议在每台集群节点的容器运行时目录额外挂载至少 200 GB 磁盘空间用于存储容器运行时数据。
- 在生产环境中，为确保活字格集群服务之间的数据同步，请额外提供一台文件共享服务器，共享目录需挂载 **至少 2T 磁盘空间** 用于共享活字格服务在运行期间生成的文件与附件存储。

## 日志监控

在生产环境中，为确保集群的日志监控模块高可用，请额外提供 ElasticSerach 集群，用来负责日志和监控数据的采集与分析工作。

- ElasticSerach 在设计上，默认会为每个索引提供副本，为确保高可用，推荐 **至少 2 个节点** 实现索引的自动备份。
- 对于活字格集群来说，ElasticSearch 的并发比较稳定，如果对于高可用要求不高，单节点的 ElasticSearch 也可以满足大多数的应用场景。

# 推荐环境预览

## 普通集群

模块名称	推荐/可选	节点数量	配置
Kubernetes 集群	推荐	2	8 个 CPU 核心、16 GB 内存和 200G 以上的磁盘空间
文件共享服务	推荐	1	8 个 CPU 核心、16 GB 内存和 2T 以上的磁盘空间
ElasticSearch 服务	可选	1	8 个 CPU 核心、16 GB 内存和 200G 以上的磁盘空间

## 高可用集群

模块名称	推荐/可选	节点数量	配置
Kubernetes 集群	推荐	5	8 个 CPU 核心、16 GB 内存和 200G 以上的磁盘空间
文件共享服务	推荐	1	8 个 CPU 核心、16 GB 内存和 2T 以上的磁盘空间
ElasticSearch 服务	推荐	2	8 个 CPU 核心、16 GB 内存和 200G 以上的磁盘空间

# 基础平台

活字格集群在负载均衡的方案选择上，仍遵循主流开发的标准实践，所依赖的基础平台为 [Kubernetes](#)。

## Kubernetes

Kubernetes（简称 K8s）是一个开源的容器编排平台，用于自动化部署、扩展和管理容器化应用。它通过统一的 API 和灵活的架构，支持跨多云和本地环境的高效应用管理，现已成为最为主流的集群构建基础平台框架。活字格集群的部署运维也是依赖于 Kubernetes 进行实现。

# 本地环境

一个标准的 Kubernetes 集群无论是需要的资源还是安装复杂度都较高。在学习和测试的过程中，准备一个轻量的 Kubernetes 环境是很有必要的。

现在存在许多为学习者订制的 Kubernetes 学习环境，其中

- [minikube](#)
- [Docker Desktop](#)

以其简单易用的特点成为最受欢迎的两个选择。这两个平台都支持单节点下运行 Kubernetes 环境，并提供与生产环境一致的 Kubernetes 功能，适合学习与调试功能。

# 在线环境

除了本地环境之外，您也可以通过一些 web 服务，在线体验与学习 Kubernetes。

- [killercoda for kubernetes](#)
- [Play with k8s](#)

这些网站会为您提供限时的多节点 Kubernetes 集群体验环境，对于初步接触 Kubernetes 和测试功能的场景提供了较大的便利。

## 集群环境

在生产环境中，可以选择以下安装方式：

- [标准安装](#)

遵照官方文档标准流程执行安装操作。流程可控，灵活性高，但操作流程复杂。

- [KubeKey](#)

借助第三方工具 KubeKey 进行自动化安装。安装流程简单，但安装版本会有延后。

## 直接开始

如果您已经拥有了自己的 Kubernetes 环境，请开启下一章的学习。

# 文件共享

本节介绍活字格集群下的文件共享。

## 共享路径

活字格负载均衡服务基于共享文件的方式实现，不同服务器访问的是同一份文件数据。在运行活字格服务的容器内部，存放文件数据的路径说明如下：

容器内部路径	数据说明	共享或本机
/opt/ForguncyAttach	业务系统上传的附件保存路径	共享存储
/opt/ForguncyLogs	系统日志保存路径	共享存储
/opt/ForguncyRestore	系统备份数据保存路径	共享存储
/opt/ForguncySites	APP 应用数据保存路径	共享存储
/opt/ForguncySitesBin	APP 程序运行路径	共享存储
/opt/ForguncyServer	主服务程序运行文件路径	Pod 独享
/opt/ForguncyServerLBCache	插件缓存路径	Pod 独享

- 使用共享存储的路径，必须使用共享文件服务器进行持久化存储，放置 Pod 漂移或重启时数据丢失。
- Pod 独享的路径，请不要存储任何业务数据。

### 务必留意

Pod 作为 Kubernetes 运行的最小单位，随时会被销毁、重启以及转移，因此您的任何业务数据都不应当存储在 Pod 中。

## 文件共享服务器

在生产环境下，为确保数据可以更为安全的持久化存储，您需要提供文件共享服务。本节教程将引导您构建基于 NFS 的文件共享服务。

## NFS

NFS (Network File System, 网络文件系统) 是一种分布式文件系统协议。它允许不同机器上的用户通过网络像访问本地文件一样访问远程文件系统。NFS 通常用于共享文件数据，使得多台计算机能够访问同一文件系统。

## NFS 服务器

1. 您需要在您的文件共享服务器上安装 NFS 的服务器程序。

CentOS    Ubuntu

```
1  yum install nfs-utils -y
```

bash

2. 将您需要共享的目录导出。

```
1 # 创建共享目录
2 mkdir -p /nfs_share
3
4 # 为目录配置权限
5 chmod -R 777 /nfs_share
6
7 # 编辑 /etc/exports 文件来定义哪些目录将被共享以及如何共享它们:
8 vim /etc/exports
9
10 /nfs_share *(rw,sync,no_subtree_check)
```

bash

- `*` : 允许所有客户端访问，也可以用特定的 IP 地址或子网（如 `192.168.1.0/24`）。
- `rw` : 允许读写访问。
- `sync` : 数据同步写入磁盘。
- `no_subtree_check` : 减少子树检查，提高性能。

3. 启动 NFS 服务并设置开机自启

```
1 systemctl start nfs-server
2 systemctl enable nfs-server
3 systemctl status nfs-server
```

bash

4. 使用 `exportfs` 命令来查看当前导出的文件系统，确认配置正确：

```
1 exportfs -v
```

bash

5. 创建活字格集群所需要的共享目录

### 共享目录映射

我们需要将活字格 Pod 中 5 个需要共享的路径映射到我们的共享文件目录上。您可以随意命名路径名称，但该路径需要与 Chart 的[配置文件](#)中保持一致。

```
1 mkdir /nfs_share/Forguncy/ForguncyAttach
2 mkdir /nfs_share/Forguncy/ForguncyLogs
3 mkdir /nfs_share/Forguncy/ForguncyRestore
4 mkdir /nfs_share/Forguncy/ForguncySites
5 mkdir /nfs_share/Forguncy/ForguncySitesBin
6 chmod -R 777 /nfs_share/*
```

bash

## NFS 客户端

1. 客户端需要在自身节点上安装 nfs 客户端服务。这里的客户端节点为活字格集群中的所有节点。

CentOS    Ubuntu

```
1 yum install nfs-utils -y
```

bash

2. 选择一个需要挂载共享目录的路径，这里直接创建一个路径 `/nfs_share_client`。

```
1 mkdir -p /nfs_share_client
```

bash

### 3. 将共享服务器导出的共享目录挂载至上一步指定的路径上。

```
1 # 10.158.37.76 为示例共享服务器 IP  
2 mount 10.158.37.76:/nfs_share /nfs_share_client
```

bash

### 4. 进入到共享目录中，查看是否存在创建的活字格映射目录

```
1 ls /nfs_share_client | grep Forguncy
```

bash

需要移除挂载？

```
1 umount /nfs_share_client
```

bash

# Chart 安装

本节介绍活字格在集群下的安装。

还没有安装 Helm ?

活字格的集群安装依赖于包管理器 Helm，如果您还未安装，可以参阅[这里](#)进行 Helm 的安装工作。

## 活字格 chart

您可以在活字格的[官方文档](#)下载活字格最新版本的 chart 包。解压后，会得到如下目录：

```
1 .
2   └── Chart.yaml
3   └── charts
4   └── templates
5     ├── deployment-fgc.yaml
6     ├── deployment-influx.yaml
7     ├── deployment-redis.yaml
8     ├── pvc.yaml
9     ├── pvs.yaml
10    ├── service-fgc.yml
11    ├── service-influx.yml
12    └── service-redis.yml
13   └── values.yaml
```

### chart 目录

chart 是 Kubernetes 应用所需要的全部资源定义，因此其本质是全部配置文件的整合。活字格集群所需要的全部资源都被定义在当前目录下的多个文件中。关于 chart 目录以及更详细的说明，可参阅 Helm [官方文档](#)

## 标准配置

对于活字格的 chart 目录，官方已经定义为标准模板。因此，如果您对于集群本身没有自定义资源的需求，仅需修改 `values.yaml` 即可。

```

1   image:
2     repository: armdocker.hzgcloud.com/fpclb/fgc_k8s_arm # 镜像仓库地址
3     tag: latest # 镜像仓库中镜像的tag
4   persistentVolume:
5     storage: "100Gi" #存储数据大小
6     nfs_attach:
7       path: "/nfs_share_client/Forguncy/ForguncyAttach" #附件存储路径
8       server: "<your-nfs-server>""
9     nfs_log:
10      path: "/nfs_share_client/nfs-share/Forguncy/ForguncyLogs" #日志存储路
11      server: "<your-nfs-server>""
12     nfs_restore:
13       path: "/nfs_share_client/Forguncy/ForguncyRestore" #备份存储路径
14       server: "<your-nfs-server>""
15     nfs_site:
16       path: "/nfs_share_client/Forguncy/ForguncySites" #网站存储路径
17       server: "<your-nfs-server>""
18     nfs_bin:
19       path: "/nfs_share_client/Forguncy/ForguncySitesBin" #网站可执行文件存
20       server: "<your-nfs-server>""

```

### 路径映射请留意！

- `values.yaml` 文件中的五个 `path` 字段，请配置为您在集群节点上挂载共享目录中对应的路径。
- 请确保与 nfs 服务器通信正常。

## 自定义配置

- 如果您希望调整 chart 本身的配置信息，请修改 `Chart.yaml`。
- 如果您对于活字格集群资源上有自定义需求，可自行修改目录中的 `templates` 文件夹下的配置。
  - 如果您希望调整活字格服务本身的资源配置，请修改 `deployment-fgc.yaml`。
  - 如果您希望调整内部缓存共享服务 redis 的资源配置，请修改 `deployment-redis.yaml`。
  - 如果您希望调整日志模块的资源配置，请修改 `deployment-influx.yaml`。

- 如果您希望调整集群中存储卷的配置，请修改 `pv*.yaml` 相关的配置文件。
- 如果您希望调整活字格服务访问相关的配置，请修改 `service*.yaml` 相关的配置文件。

## 配置规则

chart 的配置规则完全遵循 Kubernetes 的资源描述。因此配置方式您可以参阅 kubernetes [官方文档](#)。

# 安装

借助于 Helm 的能力，活字格的安装非常简单，仅需在控制平面节点上执行如下命令即可：

```
1 helm install fgc-server <chartname>.tgz -n fgc-system --create-namespace bash
```

- `fgc-server`：release 名称，可自定义。
- `<chartname>.tgz`：chart 名称。配置文件打包后的产物。当然，您也可以传入 chart 文件所在的路径，如当前目录`.`。

## 如何打包？

进入包含 chart 的目录，并运行以下命令：

```
1 helm package mychart bash
```

- `mychart` 是 chart 的目录名称。
- 如果命令执行成功，会在当前目录生成一个 `.tgz` 文件，例如 `mychart-1.0.0.tgz`。

- `-n fgc-system --create-namespace`：指定活字格集群所在的命名空间。如果您已经在指定的命名空间下，可忽略该参数。如果您的命名空间已存在，可忽略后缀 `--create-namespace`。

命令执行成功后，便可查看活字格集群的各个资源初始化情况：

```
1 kubectl get pods,svc -n fgc-system bash
2
3   NAME                               READY   STATUS    RESTARTS
4
```

5	pod/fgc-server-forguncy-pod-5f8dcf76fb-ndhsf	1/1	Running	0
6	pod/fgc-server-influx-pod-65b56f44c8-nm6bm	1/1	Running	0
7	pod/fgc-server-redis-pod-5684b45b75-fdllr	1/1	Running	0
8	NAME	TYPE	CLUSTER-IP	EXTERNAL
9	service/fgc-server-forguncy-service	NodePort	192.168.194.189	<none>
10	service/fgc-server-redis-service	NodePort	192.168.194.219	<none>
11	service/fgc-server-influx-service	NodePort	192.168.194.245	<none>

当集群中所有 Pod 的状态都为 `Running` 且 Ready 均符合当前的标准数，此时安装顺利完成。现在您可以访问 `fgc-server-forguncy-service` 对外暴露的 `31291` 端口访问活字格服务了。

### 对外服务端口

除非您在 chart 中的 service 配置中明确指定端口，否则 `fgc-server-forguncy-service` 对外暴露的端口是随机的。

## 集群初始化失败

如果您的集群在最后初始化失败，可通过如下方式进行问题的调查：

- 查看 pv, pvc 的初始化情况：

```
1 kubectl get pv,pvc -n fgc-system
```

bash

如果您的共享目录配置异常，或者集群中缺少默认的 StorageClass，都可能导致在初始化时申请存储卷时出现错误。

- 查看 Pod 的初始化描述：

```
1 kubectl describe pod <pod 名称> -n fgc-system
```

bash

除了 pod, pv、pvc、deployment、node 等各种资源都可以通过 `describe` 命令来查看状态。重点关注最后的 `events` 内容。

- 查看 Pod 内部的日志：

bash

```
1  kubectl logs <pod 名称> -n fgc-system
```

此命令需要 Pod 初始化完成，输出的日志信息为 Pod 内部运行的业务日志信息，例如活字格服务器初始化时的相关日志。

# 配置设置

本节介绍活字格集群的负载均衡配置。

## 开启负载均衡

安装成功后的活字格访问和单点下几乎没有区别，唯一的变化就是活字格服务器默认的 `22345` 端口成为了内部端口。您需要通过 集群提供的 service 暴露出来的端口进行访问管理控制台。

### 访问管理控制台

查看活字格集群提供的服务端口：

```
1 kubectl get svc -n fgc-system
```

bash

在浏览器中访问地址：<http://<nodeIP>:<svcPort>/UserService/AdminPortal/> 进入控制台

- `nodeIP`：集群中任意节点的 IP 地址。
- `svcPort`：活字格集群对外暴露的 service 端口号。

进入控制台中，跳转至「设置 → 负载均衡设置」，开启负载均衡。

- 请务必把用户数据切换至外联库
- redis 服务请设置为 `fgc-server-redis-service:6379`

### redis 服务访问

如无特殊需求，请直接配置 `fgc-server-redis-service:6379`。

`fgc-server-redis-service` 是活字格集群启动的 redis 服务，在 Kubernetes 中，Pod 会随时被销毁，为确保准确找到服务，请使用 service 名称而非 Pod IP。端口号为默认 6379。

配置完成后，重启管理控制台服务。之后为集群加入授权，就可以动态调整集群的副本数了。

```
1 # 将活字格集群的 pod 数量变为 5 个节点
2
```

bash

```
kubectl scale deployment fgc-deploy --replicas=5
```

## 应用发布

### 务必留意

集群下的活字格应用强制要求数据库必须使用外联库！！！

集群下的应用发布和单点的操作完全一致，只有配置服务器的地址时需要留意：

### 服务器地址请按照如下规则配置

`<nodeIP>:<svcPort>`

如果 Kubernetes 集群外侧挂载了网关，那么直接配置网关地址即可。网关内部按照 `<nodeIP>:<svcPort>` 进行路由配置。

# 升级

本节介绍活字格集群的升级操作。

## 直接升级

如果你的 chart 中定义的活字格镜像版本为 latest。那么只需要拉取最新的镜像。之后删除原本运行的活字格 pod 即可。新的 pod 会自动使用最新版本的镜像进行初始化。

```
1  kubectl delete pod <forguncy-pod-name> -n <fgc-namespace>
```

bash

- <forguncy-pod-name> : 活字格 pod 名称。
- <fgc-namespace> : 活字格服务的命名空间

### 容器拉取策略

为了确保避免镜像自动升级版本导致应用出现问题，请在 deployment 中配置镜像拉取策略为 IfNotPresent。这样只有当我们手动更新镜像后，才会有升级的行为。

尽管直接升级简单直接，但是为了更好的版本管理与维护，建议通过 Helm 执行升级。

## 通过 Helm 升级

您可以通过修改 release 配置后，使用 `helm upgrade` 命令。

release 版本是增量修订。每次安装，升级或回滚时，修订版本号都会增加 1。第一个修订版本号始终为 1。您可以使用 `helm history [RELEASE]` 查看特定版本的修订版号。

示例，将 chart 配置中 `values.yaml` 中的镜像版本号进行修改后：

```
1  helm upgrade -f values.yaml f fgc-server .
```

bash

## 跳过打包操作

示例中的操作没有执行 `package`，直接在 `values.yaml` 所在的当前的目录直接操作。因此传入的是当前路径 `.`

NAME	READY	STATUS	RESTARTS	AGE
fgc-server-forguncy-pod-7c86c464fd-cg4qm	1/1	Running	0	17h
fgc-server-forguncy-pod-7d6d9ccff6-cxv2l	0/1	ContainerCreating	0	8s
fgc-server-influx-pod-59786cdc6d-bxdvx	1/1	Running	0	17h
fgc-server-influx-pod-5b5fbf84c4-fmd2m	0/1	ContainerCreating	0	8s
fgc-server-redis-pod-5979cc448c-wkd99	1/1	Running	0	17h

执行 `upgrade` 命令后，Kubernetes 会自动初始化新的 pod，之后将旧的 pod 进行销毁。

您也可以通过 `helm list` 查看升级后的 release 版本号。

```
1 helm list -n fgc-system
```

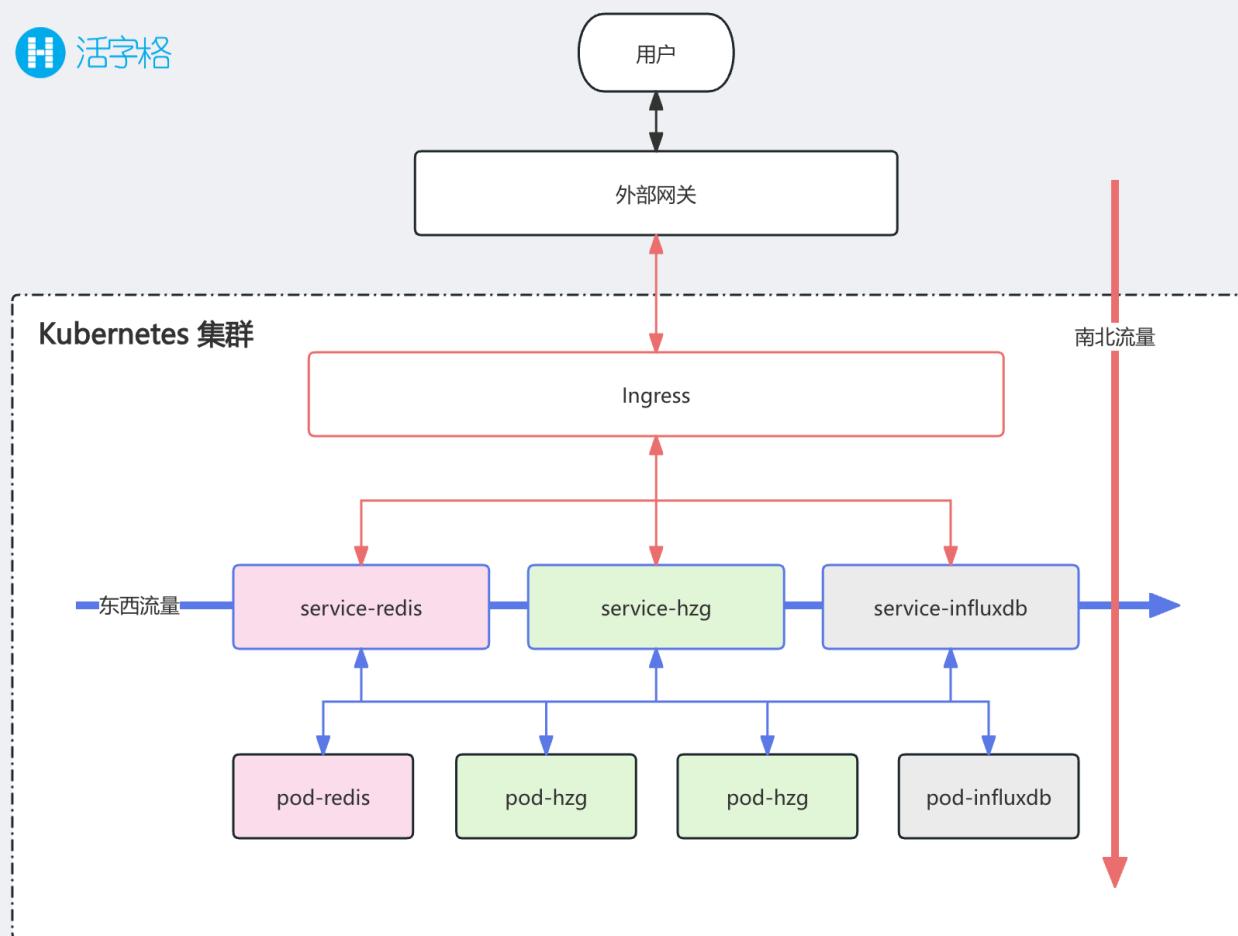
bash

# 网关代理

从整体平台架构出发，网关负责的是整个集群南北流量的管控。而活字格只是 Kubernetes 生态中的一个应用。因此，活字格集群方案中，并没有在初始配置中加入网关部分。对于流向活字格的南北流量，需要依赖于 Kubernetes 的 ingress，或者自行在集群外部构建网关服务进行流量的管控与代理。

## 集群流量走向

在一个标准的集群网络中，存在两个方向的网络流量，分别是南北流量和东西流量。



- 南北流量又称为纵向流量，主要指从请求端一直打到真正处理的服务端的全网络链路。在 Kubernetes 集群中，负责处理此方向流量的模块是 `Ingress`
- 东西流量又称为横向流量，主要指内部服务与服务之间的相关通信的全网络链路。在 Kubernetes 集群中，则是由 `Service` 来负责。

## 外部网关

考虑到引入 Ingress，会增加您对 Kubernetes 的认识复杂度以及维护成本，这里建议只配置外部网关即可，默认选择 Nginx，示例配置如下：

```

nginx
1 user nginx;
2 worker_processes auto;
3 error_log /var/log/nginx/error.log;
4 pid /run/nginx.pid;
5
6 include /usr/share/nginx/modules/*.conf;
7
8 events {
9     worker_connections 1024;
10 }
11
12 http {
13     log_format main '$remote_addr - $remote_user [$time_local] "$request"
14                               '$status $body_bytes_sent "$http_referer" '
15                               '"$http_user_agent" "$http_x_forwarded_for"';
16
17     access_log /var/log/nginx/access.log main;
18
19     sendfile          on;
20     tcp_nopush        on;
21     tcp_nodelay       on;
22     keepalive_timeout 65;
23     types_hash_max_size 2048;
24
25     include           /etc/nginx/mime.types;
26     default_type      application/octet-stream;
27
28     # 或更大值，比如90s，根据你的需求设置
29     proxy_read_timeout 600s;
30     # 同样，根据需要增大这个值
31     fastcgi_read_timeout 600s;
32     client_max_body_size 1G;
33
34     include /etc/nginx/conf.d/*.conf;
35 }
```

```

36     upstream forguncy_servers {
37         ip_hash;
38         # 集群中node节点: svc端口
39         server <node-1-ip>:<svc-fgc-port>;
40         server <node-2-ip>:<svc-fgc-port>;
41         server <node-3-ip>:<svc-fgc-port>;
42         server <node-4-ip>:<svc-fgc-port>;
43     }
44     # 管理控制台
45     server {
46         listen 22345;
47         location / {
48             proxy_pass http://forguncy_servers;
49             proxy_http_version 1.1;
50             proxy_set_header Upgrade $http_upgrade;
51             proxy_set_header Connection "upgrade";
52
53             proxy_set_header X-Real-IP $remote_addr;
54             proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
55             # 确保传递正确的 Host 头
56             proxy_set_header Host $http_host;
57         }
58     }
59
60     # app
61     server {
62         listen 80;
63         location / {
64             proxy_pass http://forguncy_servers;
65             proxy_http_version 1.1;
66             proxy_set_header Upgrade $http_upgrade;
67             proxy_set_header Connection "upgrade";
68
69             proxy_set_header X-Real-IP $remote_addr;
70             proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
71
72             proxy_set_header Host $http_host;
73         }
74     }
75
76 }
```

- 使用 `ip_hash` 将流量按照轮询的方式打到每个 node 上。
- 对管理控制台和活字格应用配置反向代理。

不熟悉该配置文件？

您可以参阅[这里](#)学习 Nginx 的使用场景以及反向代理的具体说明

## Ingress

如果您需要为 Kubernetes 集群配置 Ingress。请确保开启 Ingress 的会话保持选项：

```
1  nginx.ingress.kubernetes.io/affinity: "cookie"
2  nginx.ingress.kubernetes.io/session-cookie-name: "route"
3  nginx.ingress.kubernetes.io/session-cookie-expire: "172800"
```

yaml

# 离线安装

本节介绍在离线环境下安装活字格集群。

## 包管理器

请确保您的内网机器中包含包管理器。

- 如果您拥有私有化的镜像包源，请确保内网节点中的包管理器可以正常连接到您的镜像包源。
- 如果您的系统是纯净版，在安装的过程中可能存在部分基础库的缺失，可自行下载相应的安装包手动进行安装。
  - yum 包管理器对应的包格式为 `rpm`
  - apt 包管理器对应的包格式为 `deb`

## DNS

请明确您内网的 DNS 地址，确保机器之间的地址解析是正常的。

## 私有化镜像仓库

离线环境最大的问题就是镜像以及相关依赖无法直接从网络上下载，因此您需要提前将依赖包以及镜像进行下载，并带入至离线环境中。

在安装过程中，框架只会从仓库中读取镜像，因此，您需要构建一个本地的镜像仓库，讲下载好的镜像导入镜像仓库中。之后配置仓库地址为私有化的仓库即可。

### Harbor

[Harbor](#) 是一个开源的企业级容器镜像仓库，用于存储、管理和分发容器镜像，同时提供安全性和性能优化的功能。是市面上私有化镜像仓库的首选。

## 前提条件

- **Docker**：Harbor 服务对于 docker 是强依赖，因此您需要在镜像仓库节点上优先安装 docker 服务。安装方式可参考[文档](#)。

### 无法安装 Docker Desktop?

Docker Desktop 是安装 docker 最便捷的方式，但如果您的环境只有终端，那么需要手动安装 docker 的各项依赖，其中，docker engine 与 docker compose 是必须要安装的。

- **OpenSSL**：安装过程中对 OpenSSL 有依赖，需要提前在节点上进行安装。
- 节点的物理架构：harbor 官方最新版仅提供 amd64 的安装包。如果您的设备是 arm64 架构，需要选择第三方的编译安装包。可访问[地址](#)进行下载。

## 安装

### 1. 将下载好的离线安装包进行解压

```
1 tar -zxvf harbor-offline-installer-aarch64-v2.10.2.tgz
```

bash

### 2. 将解压后的镜像导入至 docker 中：

```
1 docker load -i harbor.v2.10.2.tar.gz
```

bash

### 3. 按需修改配置文件

```
1 mv harbor.yml.tpl harbor.yml
2
3 vim harbor.yml
```

bash

配置文件中包含了访问的 hostname 与相应服务的密码。密码可以按需修改，也可以后续进入系统中修改。hostname 如配置域名，请提供相关证书。

### 4. 在 harbor 目录下直接运行脚本

```
1 ./prepare && ./install.sh
```

bash

5. 安装成功后可访问定义好的 hostname 进行访问。harbor 默认映射到 80 端口。

## 开机自启

harbor 的主容器会自动重启，但是有几个依赖容器并没有加入到重启策略中。因此需要手动触发 docker-compose 来确保所有容器统一重启。

1. 在 `/lib/systemd/system` 中创建 `harbor.service`。

```
1 [Unit]
2 Description=Harbor
3 After=docker.service systemd-networkd.service systemd-resolved.service
4 Requires=docker.service
5 Documentation=http://github.com/vmware/harbor
6
7 [Service]
8 Type=simple
9 Restart=on-failure
10 RestartSec=5
11 ExecStart=/usr/local/bin/docker-compose -f /opt/harbor/docker-compose.yml
12 ExecStop=/usr/local/bin/docker-compose -f /opt/harbor/docker-compose.yml
13
14 [Install]
15 WantedBy=multi-user.target
```

其中 `/opt/harbor/docker-compose` 为 harbor 的安装目录。

2. 开启 service

```
1 systemctl enable harbor
2 systemctl start harbor
```

bash

## 推送本地镜像至仓库

已经下载好的本地镜像需要导入到镜像仓库，才能被真正使用。

## 1. 使用 `docker load` 命令将本地镜像推送至 Docker 中。

示例，将导出到本地的活字格镜像 `fgc_k8s_arm.tar` 进行导入：

```
1 docker load -i fgc_k8s_arm.tar
```

bash

### 从 docker 导出镜像

可以使用命令 `docker save` 命令，将 docker 中已有的镜像导出到本地。方便镜像在系统间转移。

示例，将活字格镜像 `armdocker.hzgcloud.com/fgclb/fgc_k8s_arm:10.0.103.0` 保存到 `fgc_k8s_arm.tar` 文件：

```
1 docker save -o fgc_k8s_arm.tar armdocker.hzgcloud.com/fgclb/fgc_k8s_arm:10.0.103.0
```

bash

## 2. 为导入后的镜像打标签。

示例，为活字格镜像进行标签标记：

```
1 docker tag armdocker.hzgcloud.com/fgclb/fgc_k8s_arm:10.0.103.0 hzg.harbor
```

bash

- `armdocker.hzgcloud.com/fgclb/fgc_k8s_arm:10.0.103.0`：导入的活字格镜像。您也可以直接输入镜像 ID。

### 镜像标签

标签是镜像的版本标识。通过标签，您可以明确标识镜像的具体版本以及所归属的环境仓库。镜像标签的命名规则并不是严格的。但为了确保可读性和统一性，需要遵循一些最佳实践：

- 标签名的格式通常为：`<repository>:<tag>`。

其中：

- `<repository>` 是镜像的仓库名（例如 `armdocker.hzgcloud.com/fgclb/fgc_k8s_arm`）。仓库名需要和实际的路径严格匹配。docker 会根据仓库名去对应的路径获取镜像。
- `<tag>` 是镜像的标签（例如 `10.0.103.0`, `latest`）。
- 标签是大小写敏感的。

通过 `tag` 命令，将导入的活字格镜像重新标记了新的标签。新标签的仓库名修改为本地的 harbor 仓库对应的路径：`hzg.harbor.local/k8s/fgc_k8s_arm`。

标签成功后，您会在 docker 镜像列表中看到两个镜像 ID 相同但名称不同的活字格镜像。因此后续操作请使用 `镜像名称:版本号` 的方式指定镜像，不要使用镜像 ID，避免出现镜像混淆的问题。

### 3. 将标签后的镜像推送至私有化仓库

示例，将标签后的活字格镜像推送至 harbor 仓库：

1

```
docker push hzg.harbor.local/k8s/fgc_k8s_arm:10.0.103.0
```

bash

docker 会根据镜像标签，将该镜像推送至仓库 `hzg.harbor.local` 中对应的路径 `/k8s` 下，命名 `fgc_k8s_arm`，版本号 `10.0.103.0`。

至此，我们可以将安装活字格集群需要的所有镜像，都通过如上的方法导入到私有化镜像仓库中。后续的安装操作和在线操作没有任何区别。

# 可视化

本节介绍活字格集群下的可视化监控。

## 活字格监控

活字格的管理控制台中提供了可视化的监控，包含了系统资源的使用、活字格应用与相关进程的调用情况以及活字格运行的全部系统日志与业务日志。您可以进入管理控制台中「日志与监控」进行查看。

相比于单点下的活字格监控，活字格集群的监控提供了以 Pod 为维度的监控方式。您可以指定任意一个 Pod 查看其运行情况：

The screenshot shows the FG Admin Control Panel interface. The top navigation bar includes the title '活字格' (FG), the version '10.0.2.0', and a user account 'Administrator'. The left sidebar has several icons: '内建用户' (Built-in User), '应用管理' (Application Management), '权限' (Permissions), '第三方' (Third-party), '日志和监控' (Logs and Monitoring) which is currently selected, '设置' (Settings), and '帮助支持' (Help Support). The main content area is titled '日志和监控' (Logs and Monitoring) and '监控' (Monitoring). It displays a list of monitored pods: '68cc9d49b9-mrwcf' (highlighted with a red box), '68cc9d49b9-d82gb', and '68cc9d49b9-vgsnf'. Below this, there are two sections: '活字格应用CPU使用率(%)' (FG Application CPU Usage %) and another section that is mostly blank with the message '暂无数据, 请到监控设置中开启监控' (No data, please enable monitoring in the monitoring settings). A status message at the top right says '① 监控为付费功能, 目前为试用版, 可进行试用' (Monitoring is a paid feature, currently a trial version, can be used).

如果您希望多了解在活字格下监控相关的知识，请参阅[这里](#)进行学习。

## Kubernetes 监控

除了活字格应用层面的监控，您也需要了解 Kubernetes 集群的运行情况。Kubernetes 的可视化方案有很多：

- Kubernetes Dashboard；
- KubeSphere；
- kuboard；
- 其他。

您可以选择适合您需求的方案进行安装。这里为您提供两种监控的安装方式。

## Kubernetes Dashboard

[Kubernetes Dashboard](#) 是 Kubernetes 官方提供的一个基于 Web 的用户界面，用于管理和可视化 Kubernetes 集群的资源。通过 Dashboard，您可以更直观地查看和管理集群中的资源，例如 Pods、Deployments、Services、Nodes 等。

您可以通过 Helm 快速安装 Kubernetes Dashboard。

```
1 # 添加 kubernetes-dashboard 仓库
2 helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/
3
4 # 使用 kubernetes-dashboard Chart 部署名为 `kubernetes-dashboard` 的 Helm Release
5 helm upgrade --install kubernetes-dashboard kubernetes-dashboard/kubernetes-
```

首次安装后，会自动代理服务，并将访问地址在控制台输出。后面运行需要您手动代理地址。

```
1 # 此时访问当前地址的 8443 端口即可
2 kubectl -n kubernetes-dashboard port-forward svc/kubernetes-dashboard-kong
```

如果不希望每次手动代理，可以自行修改配置，将 Service 的类型改为 [NodePort](#)。

为了保护集群数据，默认情况下，Dashboard 会使用最少的 RBAC 配置进行部署。当前，Dashboard 仅支持使用 Bearer 令牌登录。因此，您需要构建一个具备权限的 [Service Account](#)，并在每次访问时生成相应的令牌进行访问。您可以参阅[官方文档](#)了解更多。

创建 [Service Account](#) 并绑定相应权限角色：

yaml

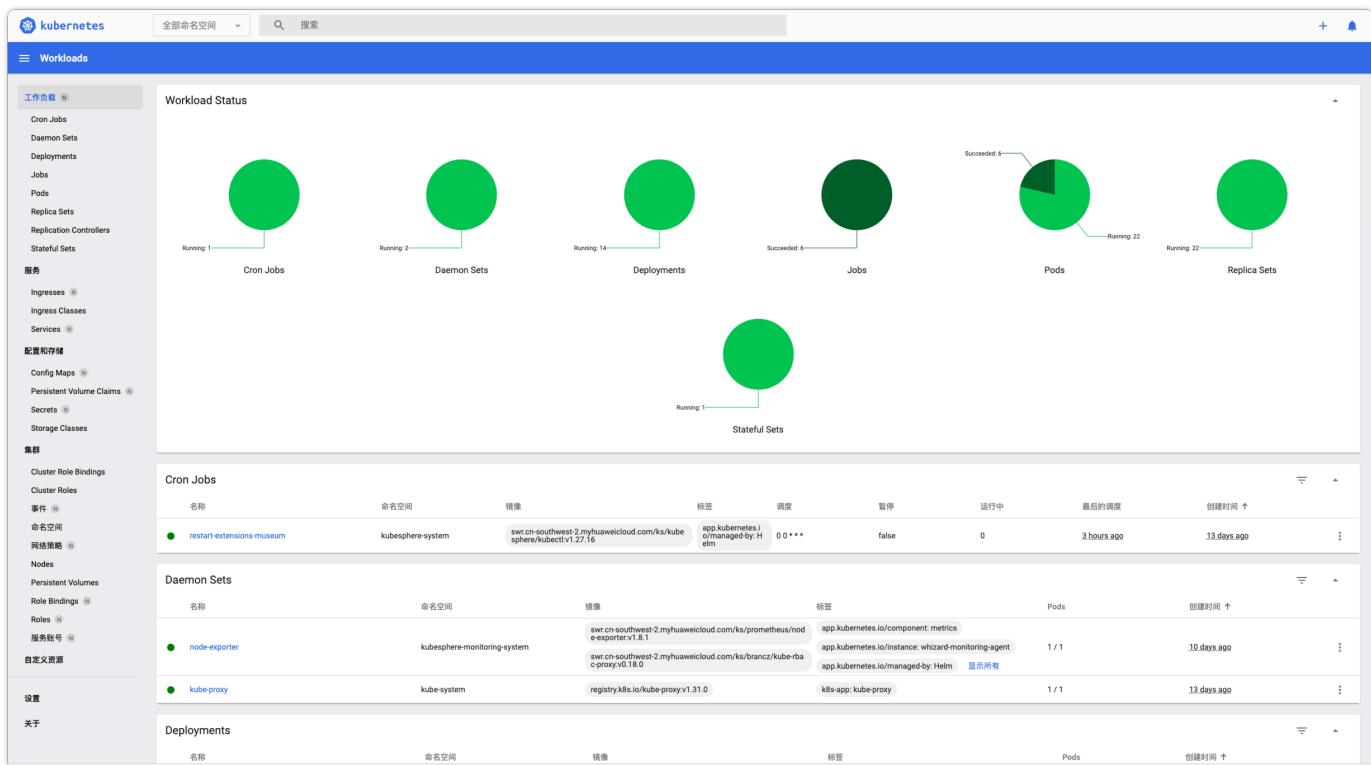
```
1 apiVersion: v1
2   kind: ServiceAccount
3   metadata:
4     name: admin-user
5     namespace: kubernetes-dashboard
6   ---
7   apiVersion: rbac.authorization.k8s.io/v1
8   kind: ClusterRoleBinding
9   metadata:
10    name: admin-user
11   roleRef:
12     apiGroup: rbac.authorization.k8s.io
13     kind: ClusterRole
14     name: cluster-admin
15   subjects:
16     - kind: ServiceAccount
17       name: admin-user
18       namespace: kubernetes-dashboard
```

为创建的用户 `admin-user` 生成令牌:

bash

```
1 kubectl -n kubernetes-dashboard create token admin-user
```

登录进系统后，便可查看当前 Kubernetes 集群信息。



## KubeSphere

KubeSphere 愿景是打造一个以 Kubernetes 为内核的云原生分布式操作系统，它的架构可以非常方便地使第三方应用与云原生生态组件进行即插即用（plug-and-play）的集成，支持云原生应用在多云与多集群的统一分发和运维管理。

KubeSphere 安装同样很简单，直接使用 Helm 安装即可：

```
1 helm upgrade --install -n kubesphere-system --create-namespace ks-core http://
```

### 访问 Docker Hub 受限

KubeSphere 提供了官方的国内镜像源，您只需要在安装命令后添加如下配置，便可修改默认的镜像拉取地址。

```
1 --set global.imageRegistry=swr.cn-southwest-2.myhuaweicloud.com/ks
2 --set extension.imageRegistry=swr.cn-southwest-2.myhuaweicloud.com/ks
```

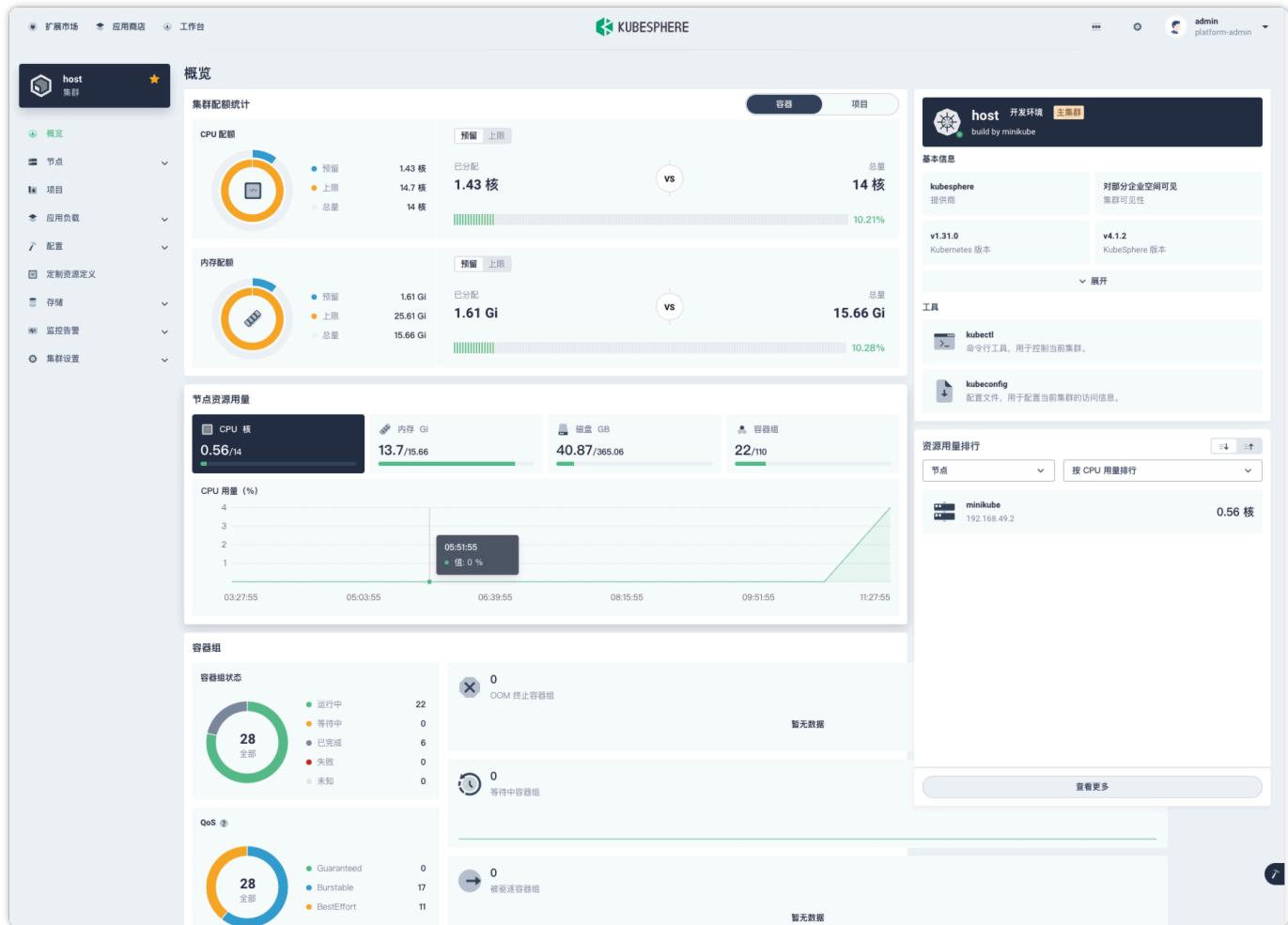
安装完成后，输出信息会显示 KubeSphere Web 控制台的 IP 地址和端口号，默认的 NodePort 是 30880。

您可以执行如下命令查看 Pod 状态。

```
1 kubectl get pods -n kubesphere-system
```

bash

当 Pod 状态都为 Running 时，使用默认的账户和密码 (admin/P@88w0rd) 通过 [\(<NodeIP>:30880\)](http://<NodeIP>:30880) 访问 KubeSphere Web 控制台。



图中的监控画面需要手动安装 KubeSphere 提供的监控插件。

# minikube

[minikube](#) 可在 macOS、Linux 和 Windows 上快速设置本地 Kubernetes 集群，专注于简化 Kubernetes 的学习和开发。

## 您需要什么

- 2 个或更多 CPU
- 2GB 空闲内存
- 20GB 空闲磁盘空间
- 互联网连接
- 容器或虚拟机管理器，例如：[Docker](#)（推荐）、[VirtualBox](#)、[Hyper-V](#) 等。

## 下载安装

您可以访问 minikube 提供的[安装命令配置界面](#)，按照您当前环境进行配置，将获取到的命令直接在终端命令行中运行即可。

## 1 Installation 安装指南

Click on the buttons that describe your target platform. For other architectures, see [the release page](#) for a complete list of minikube binaries.

点击描述您目标平台的按钮。其他架构的完整 minikube 二进制文件列表，请查看发布页面。

Operating system 操作系统

[Linux](#) [macOS](#) [Windows](#) Linux、macOS、Windows

Architecture 建筑学

[x86-64](#) [ARM64](#) [ARMv7](#) [ppc64](#) [S390x](#) x86-64、ARM64、ARMv7、ppc64、S390x

Release type 发布类型

[Stable 稳定版](#)

Installer type 安装器类型

[Binary download](#) [Debian package](#) [RPM package](#) 二进制下载 Debian 软件包及 RPM 软件包

To install the latest minikube **stable** release on **ARM64 Linux** using **binary download**:

在 ARM64 Linux 上通过二进制下载安装最新稳定版本的 minikube:

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-arm64
sudo install minikube-linux-arm64 /usr/local/bin/minikube && rm minikube-linux-arm64
```

### 镜像无法下载?

由于网络原因，可能会存在镜像拉取失败的情况。此时我们可以选择使用[镜像加速](#)来解决该问题。

## 启动集群

完成 minikube 的安装后，只需一条命令即可完成在 minikube 下的 Kubernetes 环境启动：

[minikube start](#)。

```

→ ~ minikube start
😊 Darwin 15.1.1 (arm64) 上的 minikube v1.34.0
💡 自动选择 docker 驱动
📌 使用具有 root 权限的 Docker Desktop 驱动程序
👍 Starting "minikube" primary control-plane node in "minikube" cluster
🚜 正在拉取基础镜像 v0.0.45 ...
💾 正在下载 Kubernetes v1.31.0 的预加载文件...
    > preloaded-images-k8s-v18-v1...: 307.61 MiB / 307.61 MiB 100.00% 32.05 M
    > gcr.io/k8s-minikube/kicbase...: 441.45 MiB / 441.45 MiB 100.00% 24.89 M
🔥 Creating docker container (CPUs=2, Memory=4000MB) ...
🌐 正在 Docker 27.2.0 中准备 Kubernetes v1.31.0...
    • 正在生成证书和密钥...
    • 正在启动控制平面...
    • 配置 RBAC 规则 ...
🔧 配置 bridge CNI (Container Networking Interface) ...
🌐 正在验证 Kubernetes 组件...
    • 正在使用镜像 gcr.io/k8s-minikube/storage-provisioner:v5
⭐ 启用插件: default-storageclass, storage-provisioner
🌟 完成! kubectl 现在已配置, 默认使用"minikube"集群和"default"命名空间

```

可以使用 `status` 命令查看集群状态:

```
1 minikube status
```

bash

## 与您的集群交互

如果您已经安装了 [kubectl](#), 您现在可以使用它来访问您的新集群:

```

1 # 查看节点信息
2 kubectl get nodes
3
4 # 查看 Pod 信息
5 kubectl get pods -A

```

bash

## 管理您的集群

minikube 集成了一整套命令，方便我们直接管理集群。这里只列出常用命令，您可以访问[命令文档](#)了解更多。

- 暂停集群，不影响已部署的应用程序

```
1 minikube pause
```

bash

- 取消暂停集群

```
1 minikube unpause
```

bash

- 停止集群

```
1 minikube stop
```

bash

- 删除集群

```
1 minikube delete
```

bash

- 将指定目录挂载到 minikube 上

```
1 minikube mount [flags] <source directory>:<target directory>
```

bash

- 管理插件

插件是 minikube 的维护扩展，用于为集群提供附加功能。

```
1 # 查看插件列表  
2 minikube addons list  
3  
4 # 开启某个插件功能  
5 minikube addons enable <addon name>  
6  
7  
8
```

bash

```
# 关闭某个插件功能  
minikube addons disable <addon-name>
```

# Docker Desktop

[Docker Desktop](#) 是一个一键安装的应用程序，适用于您的 Mac、Linux 或 Windows 环境，可让您构建、共享和运行容器化应用程序和微服务，也是现在 docker 官方默认的可视化安装方式。

它提供了一个简单的 GUI（图形用户界面），可让您直接从您的机器管理您的容器、应用程序和镜像。

当您安装了 Docker Desktop 后，也就意味着您的环境中已经包含了如下功能：

- Docker Engine
- Docker CLI
- Docker Build
- Docker Compose
- Kubernetes

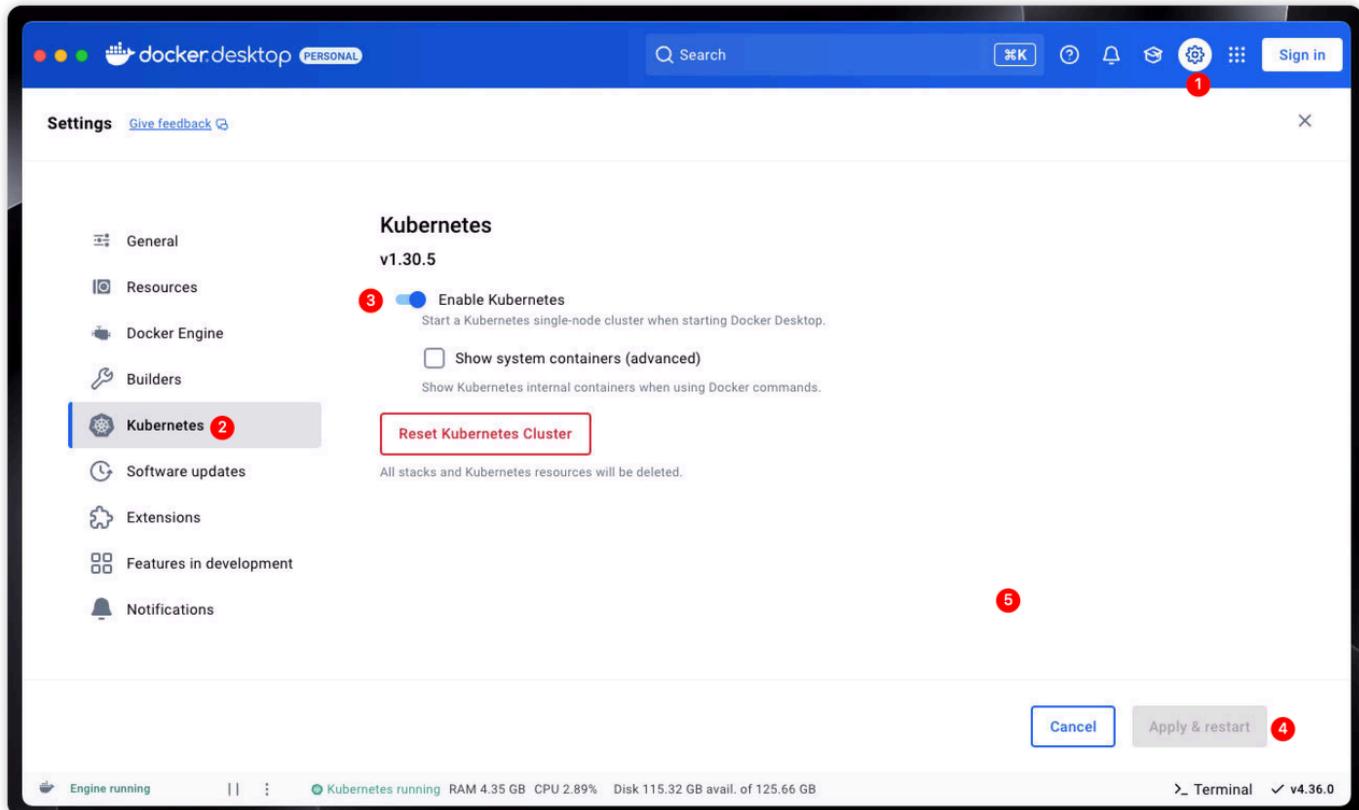
是的，Docker Desktop 中已经包含了一个基于 minikube 的 Kubernetes 环境，您只需要开启其功能即可。

---

## 开启 Kubernetes 集群

当您安装好 Docker Desktop 后，按照如下操作进行集群的开启：

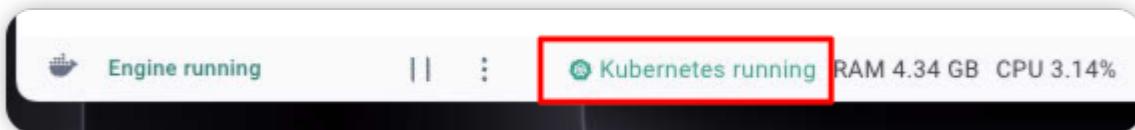
1. 从 Docker Desktop 仪表板中，请选择 **Settings** 选项。
2. 在左边侧栏选择 **Kubernetes**。
3. 勾选复选框 **Enable Kubernetes**。
4. 选择 **Apply & Restart**。



## 镜像无法下载?

由于网络原因，可能会存在镜像拉取失败的情况。此时我们可以选择使用[镜像加速](#)来解决该问题。

当 Kubernetes 启动并成功运行时，Docker Desktop 的底部状态栏和 Docker 菜单会额外显示一个状态栏。



## 与您的集群交互

Docker Desktop 并没有内置集成 [kubectl](#)，您需要手动进行安装，之后可以使用它来访问您的新集群：

```

1 # 查看节点信息
2 kubectl get nodes
3
4

```

bash

```
5 # 查看 Pod 信息  
kubectl get pods -A
```

# kubectl

kubectl 是使用 Kubernetes API 与 Kubernetes 集群的控制面板进行通信的命令行工具。

您可以通过 kubectl 在命令行中直接对 Kubernetes 集群进行交互。

## 安装方式

如果您没有提前安装 kubectl 的话, 请在开启 Kubernetes 前进行 kubectl 的安装。

### 包管理器 (推荐)

Red Hat/CentOS    Debian/Ubuntu    MacOS

```
1 sudo yum install -y kubectl
```

bash

如果您的系统是 Windows, 推荐访问 [Kubernetes 发布页面](#) 直接下载特定于您体系结构 (如 amd64、arm64 等) 下的安装程序。

### 二进制文件安装

```
1 # x86-64
2 curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/
3
4 # arm64
5 curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/
6
7 # 安装
8 sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

bash

## 验证

安装完成后, 可执行如下命令验证:

bash

```
1  kubectl version --client
```

## 指向环境

如果您拥有不只一个 Kubernetes 环境，例如本地同时安装了 minikube 与 Docker Desktop。可以通过更改上下文信息，确保 kubectl 指向您希望访问的集群环境。

bash

```
1  # 查看当前的上下文列表
2  kubectl config get-contexts
3
4  # 切换上下文指向 从 minikube 转向 docker desktop
5  kubectl config use-context docker-desktop
```

### 便捷工具

[kubectx](#) 是一个用于在 kubectl 中更快地在不同上下文（集群）之间切换的工具。

其效果等价于 `kubectl config use-context`

## 常用命令

常用命令可参考如下示例，更多详情请参考官网：[命令行工具 \(kubectl\)](#)

### 留意

在使用命令时需留意当前所在的命名空间。否则 kubectl 默认只获取 default 命名空间下的资源

- 查看集群节点

bash

```
1  kubectl get node
```

- 获取集群下的所有命名空间

```
1 kubectl get namespace
```

bash

- 查看集群中的 pods，其余资源类似

```
1 kubectl get pod -n <your namespace>
```

bash

- 查看资源的运行情况

```
1 kubectl describe pod <pod name> -n <your namespace>
```

bash

- 对于已经运行的容器，查看日志

```
1 kubectl logs <pod name> -n <your namespace>
```

bash

- 扩容缩容 Pod 数量

```
1 kubectl scale --replicas=3 deploy <deployment name>
```

bash

# 标准安装

## 我对 Kubernetes 并不熟悉

本教程为安装 Kubernetes 集群的标准流程。遵循本流程可以顺利引导构建属于您的 Kubernetes 集群，无需对 Kubernetes 有太多的接触。

当然，如果您在安装过程中遇到错误，请检查您的操作是否与教程保持一致，并确保您的基础环境和教程相同。或者您对您的环境有准确的把握。

## 安装流程说明

本教程的安装流程遵循官方说明，您可跳转至[官网文档](#)做更多了解。

# 示例环境

节点名称	节点地址	功能
k8s-master	192.168.1.4	控制面板节点，负责管理整个集群，包括调度、控制和监控。
k8s-worker	192.168.1.5	工作节点，运行实际的应用工作负载。
nfs-server	192.168.1.6	文件共享节点，通过 NFS 提供文件共享服务
es-server	192.168.1.7	ElasticSearch 节点，运行日志监控采集分析服务

## 系统信息

1 Distributor ID: Ubuntu  
2 Description: Ubuntu 24.04.1 LTS Release: 24.04  
3 codename: noble

text

# 安装流程

以下是手动安装 Kubernetes 集群的标准流程：

1. 环境初始化
2. 容器运行时
3. 安装工具依赖，包括 kubeadm、kubelet 和 kubectl
4. 初始化控制平面
5. 配置网络插件
6. 添加工作节点
7. 验证集群状态

OK，让我们开始集群的搭建吧！

# 环境准备

本节介绍集群节点的基础环境准备工作操作。

## 🎯 目标节点

所有操作面向集群所有节点

## 设置主机名

为集群节点设置主机名，确保可以通过主机名进行通信。

可通过 `hostnamectl` 命令查看当前主机名，如果符合预期，可跳过当前步骤。

```
1 # 编辑 hostname文件  
2 vim /etc/hostname  
3 # 填写主机名  
4 k8s-master
```

bash

主机名修改后请重启主机，确保主机名生效。

## hosts 解析

将需要通信的所有节点配置在各自的 hosts 文件中。

```
1 echo -e "192.168.1.4 k8s-master\n192.168.1.5 k8s-worker\n" | sudo tee -a /e
```

bash

## 关闭防火墙服务

### 1. 关闭防火墙服务

```
1 sudo ufw disable
```

bash

检查防火墙状态

```
1 sudo ufw disable
```

bash

输出结果类似以下内容，表明已关闭：

```
1 Status: inactive
```

bash

## 2. 禁用防火墙服务，防止系统重启后自动启动

```
1 sudo systemctl stop ufw
2 sudo systemctl disable ufw
```

bash

## 3. 验证防火墙状态

```
1 sudo systemctl status ufw
```

bash

### 建议

如后期需要开启防火墙，请在安装成功后开启，并公开相应的端口

## 永久关闭 selinux

selinux 负责 Linux 在系统底层资源调用的安全管控，在使用 Kubernetes 时需要关闭。

Ubuntu 默认不启用 selinux，可跳过此步骤。

### 1. 编辑 SELinux 配置文件

```
1 sudo vi /etc/selinux/config
2
```

bash

```
3 # 修改 SELINUX=enforcing 为 SELINUX=disabled
4 SELINUX=disabled
```

2. 保存并重启系统。系统重启成功后验证 SELinux 状态：

```
1 sestatus
```

bash

## 关闭 swap

swap 会影响系统性能，因此 Kubernetes 官方推荐关闭 swap，否则初始化控制面板时会强制停止。

只需打开 `/etc/fstab`，将如下行禁用即可

```
1 #/swap.img none swap sw 0 0
```

bash

可使用命令 `free -h` 查看。如果 swap 都是 0 表明系统停止 swap。

## 系统时间同步

集群服务需要多个节点的时间保持一致。

可通过 `date` 或者 `timedatectl` 命令查看当前节点的时间。

如果时间不正确，请替换为东八区-上海

```
1 timedatectl set-timezone Asia/Shanghai
```

bash

## 确认网络通信的模块开启

bash

```

1 modprobe overlay
2 modprobe br_netfilter

```

## 配置流量链

有一些 ipv4 的流量不能走 iptables 链，因为 linux 内核的一个过滤器，每个流量都会经过他，然后再匹配是否可进入当前应用进程去处理，所以会导致流量丢失。

### 1. 配置 k8s.conf 文件（k8s.conf 文件本身不存在，需要自己创建的）

bash

```

1 sudo vim /etc/sysctl.d/k8s.conf
2
3 # 新增如下内容
4 vm.swappiness=0
5 net.bridge.bridge-nf-call-ip6tables=1
6 net.bridge.bridge-nf-call-iptables=1
7 net.ipv4.ip_forward=1

```

### 2. 执行如下命令重载系统参数

bash

```
1 sudo sysctl --system
```

### 3. 验证结果

bash

```

1 # 此命令结果必须为 1
2 cat /proc/sys/net/ipv4/ip_forward

```

# 容器运行时

容器运行时是容器生态中的一个核心组件，它负责容器的具体运行和管理。它抽象了底层操作系统资源，提供了一种轻量级的方式来启动和管理应用程序容器。我们所熟悉的 docker，只是容器运行时的一种。

## 主流容器运行时

- Docker Engine：曾是容器运行时的标准实现，集镜像管理和容器运行于一体。
- containerd：从 Docker 项目中分离出来的轻量级运行时，专注于核心运行功能。
- CRI-O：专为 Kubernetes 的 CRI（容器运行时接口）设计的运行时，轻量、专注。
- runc：OCI（开放容器标准）规范的低级容器运行时，执行容器的创建和管理。

Kubernetes 在早期版本支持 Docker 作为默认的容器运行时，但其架构逐渐过渡为支持 CRI (Container Runtime Interface)，以支持多种运行时并增强灵活性。因此，在现在的主流 Kubernetes 版本中，官方已经推荐使用 containerd 作为标准的容器运行时了。

本节介绍容器运行时 containerd 的安装。

### 🎯 目标节点

所有操作面向集群所有节点

## containerd 安装

[containerd 官方](#)推荐了两种安装方式：

- 二进制安装
- 包管理器安装

## 二进制安装

### 适用场景

如果您的安装环境是离线环境，请选择本方式进行安装，否则推荐[包管理器](#)。

如果您选择使用二进制的方式进行安装，请确保下载并安装如下 3 个二进制包，下载时请确认您的版本和架构信息：

- [containerd](#): containerd 的主程序

#### 务必留意

1. 下载后请将二进制包解压至指定路径 `/usr/local`

```
1 tar Cxzvf /usr/local containerd-1.6.2-linux-amd64.tar.gz
```

bash

2. Kubernetes 要求使用 systemd 来启动容器运行时，因此，我们务必要下载

[containerd.service](#)，并将其导至 `/usr/local/lib/systemd/system/containerd.service`。之后运行如下命令：

```
1 systemctl daemon-reload  
2 systemctl enable --now containerd
```

bash

- [runc](#): 用来执行容器的创建和管理

#### 务必留意

下载后请将安装包直接安装到路径 `/usr/local/sbin/runc`

```
1 install -m 755 runc.amd64 /usr/local/sbin/runc
```

bash

- [CNI 插件](#): 容器之间的网络通信插件

#### 务必留意

下载后请将安装包解压至指定路径 `/opt/cni/bin`

```
1 $ mkdir -p /opt/cni/bin  
2 $ tar Cxzvf /opt/cni/bin cni-plugins-linux-amd64-v1.1.1.tgz
```

bash

## 包管理器安装

CentOS    Ubuntu

```
1 sudo dnf install containerd.io
```

bash

### 需要留意

`containerd.io` 软件包包含 `runc`，但不包含 CNI 插件。

### 包管理器无法下载？

containerd.io 的 DEB 和 RPM 格式软件包由 Docker（而非 containerd 项目）分发。您需要设置 `apt-get` 或 `dnf/yum` 的存储库才可以安装 containerd.io 软件包。具体配置的方法请参阅 Docker 的[帮助手册](#)

## 启动 containerd 服务

安装完成后，输入如下命令启动 containerd：

```
1 sudo systemctl enable containerd
2
3 sudo systemctl start containerd
```

bash

可通过如下方式进行验证：

```
1 # 查看 containerd 状态
2 systemctl status containerd
3
4 # 查看 ctr 的版本信息
5 sudo ctr version
```

bash

## 修改配置属性

此时的 containerd 虽然已经运行起来，但是如果希望和 Kubernetes 集群配合使用，您仍需要做一些配置上的调整。

## 1. 生成一份默认的配置文件

```
1 sudo mkdir -p /etc/containerd && \
2     sudo containerd config default > /etc/containerd/config.toml
```

## 2. 进入配置文件中，修改配置信息

```
1 vim /etc/containerd/config.toml
```

## 3. 配置文件中有如下几处需要留意/修改：

- `sandbox_image`：该属性用于配置 Kubernetes 的 pause 镜像地址。请将版本变更为您需要安装的对应版本地址。

```
1 sandbox_image = "registry.k8s.io/pause:3.9"
```

- `SystemdCgroup`：Kubernetes 的 cgroup 需要与操作系统、容器运行时保持一致，默认统一为 systemd，需要 containerd 开启 systemd。

```
1 [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
2 SystemdCgroup = true
```

### 没有 `SystemdCgroup` 属性？

在 containerd 的最新几个版本中，已经默认为 systemd。如果您不确定的话，可以手动添加该属性。需要留意其所在的层级。

- `config_path`：配置镜像加速地址（可选）。containerd 默认从 docker.io 进行镜像的拉取。如果您的网络访问 docker.io 不便，需要手动配置您自己的镜像加速地址：

```
1 [plugins."io.containerd.grpc.v1.cri".registry]
2 config_path = "/etc/containerd/certs.d"
```

## 需要留意

配置项 `/etc/containerd/certs.d` 是手动在 `/etc/containerd` 目录下创建的目录。此规则是 containerd 提供的自定义镜像源方式：即在配置文件的目录下自定义 registry server 目录。

`certs.d` 中存放的都是需要代理的仓库源，每个源一个路径，具体的配置保存在 `hosts.toml` 中。更多说明请参考[官方文档](#)。

这里直接使用 `_default` 配置默认镜像加速，镜像加速为私有化仓库 [Harbor](#)，您可以变更为您自己的加速地址。示例如下：

```
1 # 创建默认加速目录                                bash
2 mkdir -p /etc/containerd/certs.d/_default
3
4 # 将仓库地址写入到配置文件 hosts.toml 中
5 cat > /etc/containerd/certs.d/_default/hosts.toml << EOF
6 [host."https://hzg.harbor.local"]
7   capabilities = ["pull", "resolve"]
8   # 如果是 http, 请置为 true
9   skip_verify = false
10 EOF
```

## 4. 配置完成后，重启 containerd 服务

```
1 systemctl restart containerd                                bash
2
3 # 拉取镜像进行测试
4 ctr i pull docker.io/library/nginx:alpine
```

# kubeadm、kubelet 与 kubectl

本节介绍 Kubernetes 基础软件包：kubeadm、kubelet 与 kubectl 的安装。

## 🎯 目标节点

所有操作面向集群所有节点

## 安装

您需要在每台节点机器上安装以下的软件包：

- kubeadm：用来初始化集群的指令。
- kubelet：在集群中的每个节点上用来启动 Pod 和容器等。
- kubectl：用来与集群通信的命令行工具。

### 务必留意

三个软件包的版本请确保一致，避免因为版本偏差导致一些预料之外的错误和问题。

此外，如果您的包管理器未配置 Kubernetes 的仓库地址，请参阅[这里](#)进行配置。

Ubuntu    CentOS

bash

```
1  sudo apt-get update
2  sudo apt-get install -y kubelet kubeadm kubectl
3  # 锁定版本号避免自动更新
4  sudo apt-mark hold kubelet kubeadm kubectl
5
6  # 设置开机启动
7  sudo systemctl enable --now kubelet
```

此时 kubelet 每隔几秒就会重启，因为它陷入了一个等待 kubeadm 指令的死循环。这是正常现象，当我们将集群的控制平面初始化成功后，kubelet 的状态就会恢复正常。

# 初始化控制平面

本节介绍通过 kubeadm 初始化集群的控制平面。

## 🎯 目标节点

本节操作面向控制平面节点。

## 配置信息（可选）

### 1. 使用 kubeadm 命令生成默认的配置文件

```
1  kubeadm config print init-defaults > kubeadm-config.yaml
```

bash

### 2. 请关注/修改如下配置项：

- `advertiseAddress`：修改为控制面板节点的 IP。

```
1  localAPIEndpoint:  
2    advertiseAddress: 192.168.1.4,  
3    bindPort: 6443
```

yaml

- `imageRepository`：修改为您的仓库地址。

```
1  imageRepository: hzg.harbor.local/google_containers  
2  kind: ClusterConfiguration
```

yaml

- `kubernetesVersion`：修改为您要安装的 Kubernetes 版本号。

```
1  kubernetesVersion: 1.31.3
```

yaml

- `serviceSubnet` 与 `podSubnet`：分配网络地址。我们需要为集群内部的服务和 Pod 分配对应的网段。

```

1   networking:
2     dnsDomain: cluster.local
3     serviceSubnet: 192.168.0.0/16
4     podSubnet: 192.169.0.0/16

```

yaml

3. 修改完成后保存配置，并提前预拉取镜像。

```
1   kubeadm config images pull --config=kubeadm-config.yaml
```

bash

## 初始化控制平面

直接执行如下命令，指定配置文件初始化控制平面：

```
1   kubeadm init --config=kubeadm-config.yaml
```

bash

当然，如果您未在第一步初始化配置文件，也可以直接通过命令行参数，将配置在命令行中进行指定：

```

1   kubeadm init \
2     --apiserver-advertise-address=192.168.1.4 \
3     --control-plane-endpoint=k8s-master \
4     --kubernetes-version=v1.31.3 \
5     --service-cidr=192.168.0.0/16 \
6     --pod-network-cidr=192.169.0.0/16 \
7     --cri-socket=unix:///run/containerd/containerd.sock \
8     --image-repository=hzg.harbor.local/google_containers

```

bash

初始化成功后，您会在控制台中看到如下信息：

```

1   Your Kubernetes control-plane has initialized successfully!
2   To start using your cluster, you need to run the following as a regular user:
3
4     mkdir -p $HOME/.kube
5     sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
6

```

bash

```
6      sudo chown $(id -u):$(id -g) $HOME/.kube/config
7
8 Alternatively, if you are the root user, you can run:
9
10    export KUBECONFIG=/etc/kubernetes/admin.conf
11
12 You should now deploy a pod network to the cluster.
13 Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
14 https://kubernetes.io/docs/concepts/cluster-administration/addons/
15
16 Then you can join any number of worker nodes by running the following on ea
17
18    kubeadm join 192.168.0.163:6443 --token abcdef.0123456789abcdef \
19      --discovery-token-ca-cert-hash
20      sha256:50b63996388eb4bd55819701e2d53aa34ad892842b7009837c100f4632f25e08
```

其中，高亮的部分表示您需要按照提示，在您的控制平面节点下执行的命令。

至此，您的 Kubernetes 集群中控制平面初始化顺利完成。您可以通过 `kubectl` 命令查看集群状态了。

```
1  kubectl get nodes                                bash
```

# 网络插件

本节介绍为您的 Kubernetes 集群配置容器网络接口（CNI）。

## 🎯 目标节点

本节操作面向控制平面节点。

## CNI

CNI (Container Network Interface) 是 Kubernetes 网络的重要组成部分。其定义了网络插件需要实现的接口，使容器运行时能够与网络插件进行交互。主要负责在 Pod 创建时配置网络，在 Pod 删除时清理网络。

目前常见的 CNI 插件有：

- Calico
- Flannel
- Canal

本节教程介绍的是 Calico 的安装。

## 务必留意

您必须部署一个基于 Pod 网络插件的容器网络接口（CNI），以便集群中的 Pod 可以相互通信。在安装网络之前，集群 DNS (CoreDNS) 将不会启动。

## Calico

Calico 是一套开源的网络和网络安全方案，用于容器、虚拟机、宿主机之间的网络连接，可以用在 Kubernetes 平台上。

您已经在上一节将控制平台初始化成功，这意味着安装 Calico 时，可以直接利用 Kubernetes 的配置安装能力。关于更为详细的安装说明，可参阅[官方文档](#)。

## tigera-operator

Calico 在新版更新后，已经弃用直接通过配置文件的方式初始化本体了。需要先安装 `tigera-operator` 作为 Calico 的控制器。`tigera-operator` 会管理 Calico 相关组件的部署、配置与升级。

```
1  kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3
```

bash

### 命令失败？

上述命令是直接调用官方线上的配置文件进行安装。如果无法访问线上的配置地址，可访问该[地址](#)，将其配置内容保存至您的设备上的 `yaml` 文件。

## 自定义配置

下载默认配置，按照您的集群信息修改配置内容：

```
1  wget https://raw.githubusercontent.com/projectcalico/calico/v3.29.1/manifes
```

bash

### 命令失败？

可切换成如下命令进行下载：

```
1  curl https://projectcalico.docs.tigera.io/manifests/custom-resources.yaml -o
```

bash

请确保配置的 `cidr` 的值，和您在初始化控制平台时，指定的 Pod 的 [IP 网段（`podSubnet`）](#) 保持一致。

```
1  ...上文配置...
```

yaml

```
2  spec:
3    calicoNetwork:
4      ipPools:
5        - name: default-ipv4-ippool
6          blockSize: 26
```

```
cidr: 192.169.0.0/16
```

```
...下文配置...
```

## 安装

配置完成后，直接运行如下命令进行安装：

bash

```
1 kubectl apply -f custom-resources.yaml
2
3 # 稍等片刻后，查看 Pod 运行情况
4 kubectl get pod -A
```

# 添加工作节点

本节介绍为您的 Kubernetes 集群加入控制节点。

## 🎯 目标节点

本节操作面向工作节点。

## 加入集群

当您初始化控制平台成功后，控制台会输出如下信息：

```
1 Then you can join any number of worker nodes by running the following on ea  
2  
3     kubeadm join 192.168.0.163:6443 --token abcdef.0123456789abcdef \  
4     --discovery-token-ca-cert-hash  
5     sha256:50b63996388eb4bd55819701e2d53aa34ad892842b7009837c100f4632f25e08
```

高亮的命令就是您需要复制到工作节点中执行的命令。该命令将当前的节点加入到 Kubernetes 集群中。

## 务必留意

请确保您的工作节点已经完成[环境准备](#)、[containerd](#)、[kubeadm](#)、[kubelet](#) 以及 [kubectl](#) 的安装工作。

在当前节点执行完 join 命令后，可回到控制平台节点，执行如下命令查看是否加入成功：

```
1     kubectl get node
```

# KubeKey

本节介绍使用第三方工具：由 [KubeSphere](#) 提供的脚手架 KubeKey 进行 Kubernetes 集群的安装。

## 需要留意

从标准化与流程化出发，当前 KubeKey 支持一键安装的 Kubernetes 版本号仅支持到 V1.28.0，且对于安装机器的要求版本也较低。如果您希望安装更新版本的 Kubernetes，请选择[标准安装](#)的方式。

您可以使用任何节点作为任务机来执行安装任务，也可以在安装之前或之后根据需要新增节点（例如，为了实现高可用性）。

- **Control plane node**: 主节点，通常托管控制平面，控制和管理整个系统。
- **Worker node**: 工作节点，运行部署在工作节点上的实际应用程序。

## 准备机器

### 系统要求

- Ubuntu 16.04, 18.04, 20.04
- Debian Buster, Stretch
- CentOS 7.x
- Red Hat Enterprise Linux 7
- SUSE Linux Enterprise Server 15 /openSUSE Leap 15.2

### 节点要求

- 所有节点必须都能通过 [SSH](#) 访问。
- 所有节点时间同步。
- 所有节点都应使用 [sudo](#) / [curl](#) / [openssl](#) / [tar](#)。

### 容器运行时

集群必须有一个可用的容器运行时。如果您使用 KubeKey 搭建集群，KubeKey 会默认安装最新版本的 Docker。或者，您也可以在创建集群前手动安装 Docker 或其他容器运行时。

容器运行时	版本号
Docker	19.3.8+
containerd	最新版

## 网络和 DNS

- 请确保 `/etc/resolv.conf` 中的 DNS 地址可用，否则，可能会导致集群中的 DNS 出现问题。
- 请关闭防火墙，或者配置特定的端口通信。
- 支持的 CNI 插件：Calico 和 Flannel。

本示例包括以下两台主机，其中主节点作为任务机：

主机 IP	主机名	角色
198.19.249.234	k8s-master	control plane, etcd
198.19.249.72	k8s-node	worker

## 下载 KubeKey

```

1 # 可以正常访问 Github
2 curl -sfL https://get-kk.kubesphere.io | VERSION=v3.0.7 sh -
3
4 # 否则
5 export KKZONE=cn
6 curl -sfL https://get-kk.kubesphere.io | VERSION=v3.0.7 sh -

```

bash

## 安装依赖项

bash

```
1 sudo apt install socat conntrack ebttables ipset -y
```

## 创建并修改配置项

1. 创建配置文件。如果不指定版本号，默认为 v1.23.10

```
1 ./kk create config --with-kubernetes v1.23.10
```

2. 编辑配置文件。关于配置的详细说明，请参考[官方文档](#)

yaml

```
1 apiVersion: kubekey.kubesphere.io/v1alpha2
2 kind: Cluster
3 metadata:
4   name: kube-fgc
5 spec:
6   hosts:
7     - {
8       name: k8s-master,
9       address: 198.19.249.234,
10      internalAddress: 198.19.249.234,
11      user: gerald,
12      password: "your-password",
13    }
14    - {
15      name: k8s-node,
16      address: 198.19.249.72,
17      internalAddress: 198.19.249.72,
18      user: gerald,
19      password: "your-password",
20    }
21   roleGroups:
22     etcd:
23       - k8s-master
24     control-plane:
25       - k8s-master
26     worker:
```

```

28      - k8s-node
29
30  registry:
31      - k8s-master
32
33  controlPlaneEndpoint:
34      domain: lb.kubesphere.local
35      address: ""
36      port: 6443
37
38  kubernetes:
39      version: v1.23.10
40      clusterName: cluster.local
41      autoRenewCerts: true
42      containerManager: docker
43
44  etcd:
45      type: kubekey
46
47  network:
48      plugin: calico
49      kubePodsCIDR: 10.233.64.0/18
50      kubeServiceCIDR: 10.233.0.0/18
51      multusCNI:
52          enabled: false
53
54  registry:
55      privateRegistry: ""
56      namespaceOverride: ""
57      registryMirrors: ["your-registry-mirrors"]
58      insecureRegistries: []
59
60  addons: []

```

## 执行安装脚本

```
1 ./kk create cluster -f config-sample.yaml
```

bash

## 卸载集群

如果您希望卸载由 KubeKey 安装的集群。请务必使用安装时的脚本，否则卸载过程中会出现不可预估的问题。

```
1 ./kk delete cluster -f config-sample.yaml
```

# Helm

[Helm](#) 是 Kubernetes 的包管理器，就像 CentOS 中的 [yum](#) 或者 Ubuntu 中的 [apt](#) 一样，帮助开发者和运维人员更为高效便捷地定义、安装和管理 Kubernetes 应用。

Helm 使用一种称为 [Chart](#) 的打包格式来描述应用的 Kubernetes 资源和相关的元信息，简单来说，Helm 中的应用都是 [Chart](#) 格式。通过管理 [Chart](#) 包，简化了复杂应用的部署流程。

## 安装

### 先决条件

想成功和正确地使用 Helm，需要以下前置条件：

- 一个 Kubernetes 集群
- 本地的 [kubectl](#)

### 安装在哪里？

为了更为便捷的管理 Kubernetes 中的应用，建议将 Helm 安装在控制平面所在的节点上。

Helm 项目提供了两种获取和安装 Helm 的方式。这是官方提供的获取 Helm 发布版本的方法。另外，Helm 社区提供了通过不同包管理器安装 Helm 的方法。

## 二进制安装（推荐）

1. 下载[需要的版本](#)。
2. 解压文件。

1

```
tar -zxvf helm-v3.0.0-linux-amd64.tar.gz
```

bash

3. 在解压目录中找到 helm 程序，移动到指定目录 [/usr/local/bin/helm](#) 中

```
1 mv linux-amd64/helm /usr/local/bin/helm
```

bash

#### 4. 验证是否安装成功

```
1 helm version
```

bash

## 使用脚本安装

Helm 提供了安装脚本，可以自动拉取最新版本并本地安装

```
1 $ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/get_helm.sh
2 $ chmod 700 get_helm.sh
3 $ ./get_helm.sh
```

bash

## 包管理器安装

Helm 社区提供了通过操作系统包管理器安装 Helm 的方式。您可以参阅[官方文档](#)。

## 使用

在 Helm 中存在三大概念：

- **Chart** 代表着 Helm 包，它包含了运行在 Kubernetes 中应用所具有的所有资源定义。
- **Repository** 用来存放和共享 Chart 的地方。您可以把它理解为线上的共享仓库。
- **Release** 是运行在 Kubernetes 集群中的 Chart 实例。每次安装/升级，都会生成一个新的 release。

概括来说：Helm 会安装 charts 到 Kubernetes 集群中，每次安装都会创建一个新的 release。您可以在 repository 找到您想要安装的 chart。

## 常用命令

您可以通过如下命令对 Kubernetes 的应用进行管理。活字格的服务也是通过这些 Helm 命令完成安装与升级的工作。

- `helm install`：安装一个 helm 包。

```
1  helm install <release 名称> <chart 名称 或者 包地址>
```

bash

- `helm uninstall`：卸载 release。

```
1  helm uninstall <release 名称>
```

bash

- `helm upgrade` 与 `helm rollback`：升级 release 和失败时回滚。

```
1  helm upgrade -f <要更新的配置文件> <release 名称> <chart 名称 或者 包地址>
```

bash

- `helm repo` 与 `helm search`：配置 helm 的仓库以及在仓库中查询应用包。

```
1  # 查询仓库列表
2  helm repo list
3
4  # 添加新的仓库
5  helm repo add <仓库地址>
6
7  # 移除仓库
8  helm repo remove <仓库地址>
```

bash

# 镜像加速

镜像加速是指通过优化镜像下载的速度和效率，加速从镜像仓库拉取容器镜像的过程。这一技术在网络质量不佳或镜像仓库地理位置较远时尤为重要。

容器镜像通常托管在公共或私有的镜像仓库中，如 Docker Hub、Google Container Registry (GCR)、阿里云镜像服务等。当用户在国内或低速网络环境中拉取镜像时，可能会遇到下载缓慢的问题，这会显著影响容器启动和部署的效率。

通过配置镜像加速服务，可以显著提升镜像的下载速度，从而减少容器化应用的启动时间。

## 获取镜像加速服务

目前，各大云厂商都已经面向个人用户提供了镜像加速服务。您可以选择一个合适的服务厂商，获取其镜像加速服务。

离线环境无法访问云服务？

不用担心，您可以自行构建私有化的[镜像仓库](#)！同样可以实现镜像加速的效果。

这里以华为云为例，其他厂商类似。

1. 登录[容器镜像服务控制台](#)
2. 在左侧导航栏选择“镜像资源 > 镜像中心”。
3. 点击[镜像加速器](#)，复制加速器地址。

The screenshot shows the 'Container Image Service SWR' interface on the Huawei Cloud control panel. A modal window titled 'Mirror Accelerator' is open, providing instructions for configuration:

- 1. 安装/升级容器引擎客户端**: Recommended to install version 1.11.2 or higher of the container engine client.
- 2. 加速器地址**: Displays the URL `https://[REDACTED].mirror.swr.myhuaweicloud.com`. A red circle with the number '3' is placed over this field.
- 3. 配置镜像加速器**: Instructions for users with clients > 1.11.2. It advises using root to log in to the container engine and modifying the `/etc/docker/daemon.json` file. A code block shows the configuration entry:
 

```
{
    "registry-mirrors": [ "https://[REDACTED]mirror.swr.myhuaweicloud.com" ]
}
```
- 4. 重启容器引擎**: After configuration, run `systemctl restart docker` to restart the container engine. If it fails, check other configuration files like `/etc/sysconfig/docker` or `/etc/default/docker`.
- 5. 确认配置结果**: Check `docker info` for the 'Registry Mirrors' field. If it shows the configured address, the configuration is successful.

The modal has a '确定' (Confirm) button at the bottom right. The background shows a list of search results on the right side of the page.

## 配置加速器

获取到镜像加速地址后，您需要将其加入到您的容器运行时配置中。

### Docker

- 如果您是通过命令行安装的 docker，那么请直接编辑 `/etc/docker/daemon.json` 文件（如果文件不存在，请直接创建）。

```

1  {
2    "registry-mirrors": ["https://<your-mirror-url>"]
3  }

```

- 如果您的 docker 拥有可视化界面，例如 Docker Desktop，直接在“设置界面 > Docker Engine”中进行配置。配置内容和上文相同。

配置完成后，可通过如下命令查看字段 Registry Mirrors 是否变更为指定镜像加速器

bash

```
1 docker info
```

也可以拉取一个镜像进行测试。

bash

```
1 docker pull nginx
```

## Containerd

对于 Kubernetes 较新的版本，官方已经推荐容器运行时为 containerd。这是一个原生支持 CRI 的容器运行时，因此更轻量且更高效。

基于 containerd 的镜像加速配置，您可以参考[容器运行时](#)。