Table of Contents

Rancher 1.6 Docs 1.1
第1章 简介 1.2
第2章 快速开始 1.3
第3章 安装Rancher 1.4
第4章 升级Rancher Server 1.5
第5章 系统设置 1.6
- 5-1 访问控制 1.6.1
- 5-2 账户 1.6.2
- 5-3 注册主机和应用商店 1.6.3
- 5-4 主机驱动 1.6.4

第6章 环境 1.7
第7章 主机 1.8
第8章 镜像仓库 1.9
第9章 证书 1.10
第10章 系统服务 1.11
- 10-1 基础设施服务 1.11.1
- 10-2 网络 1.11.2
- 10-3 网络策略 1.11.3
- 10-4 负载均衡器 1.11.4
- 10-5 DNS服务 1.11.5
- 10-6 Metadata服务 1.11.6
- 10-7 存储服务 1.11.7
- 10-8 调度任务 1.11.8
- 10-9 审计日志 1.11.9
- 10-10 服务账号 1.11.10

第11章 Cattle 1.12
- 11-1 应用 1.12.1
- 11-2 服务 1.12.2
- 11-3 卷 1.12.3
- 11-4 负载均衡 1.12.4
- 11-5 外部服务 1.12.5
- 11-6 服务别名 1.12.6
- 11-7 健康检查 1.12.7
- 11-8 调度服务 1.12.8
- 11-9 升级服务 1.12.9
- 11-10 Cattle环境中的内部DNS服务 1.12.10
- 11-11 外部DNS服务 1.12.11
- 11-12 Rancher-Compose 1.12.12
- 11-13 密文 - 实验性的 1.12.13

11-14 Webhooks 1.12.14
11-15 标签 1.12.15
第12章 Kubernetes 1.13
12-1 启动Kubernetes 1.13.1
12-2 弹性平面 1.13.2
12-3 Cloud Providers 1.13.3
12-4 私有镜像仓库 1.13.4
12-5 RBAC 1.13.5
12-6 灾难恢复 1.13.6
12-7 备份 1.13.7
12-8 持久化存储 1.13.8
12-9 插件支持 1.13.9
12-10 Ingress支持与配置 1.13.10
12-11 升级Kubernets 1.13.11
12-12 删除Kubernets 1.13.12
第13章 Mesos 1.14
第14章 Swarm - 实验性的 1.15
第15章 Windows - 实验性的 1.16
第16章 以原生Docker命令行的形式使用Rancher 1.17
第17章 应用商店 1.18
17-1 应用商店 1.18.1
17-2 创建私有应用商店 1.18.2
第18章 Rancher命令行 1.19
18-1 Rancher命令行 1.19.1
18-2 命令和选项 1.19.2
18-3 变量替换 1.19.3
第19章 API文档 1.20
第20章 Rancher社区贡献 1.21
第21章 常见问题 1.22
21-1 Rancher Server的常见问题 1.22.1
21-2 Rancher Agent的常见问题 1.22.2
21-3 常见的故障排查与修复方法 1.22.3
第22章 历史文档 1.23

Rancher 1.6 Docs

Rancher 1.6 中文文档

Rancher概览

Rancher是一个开源的企业级容器管理平台。通过Rancher，企业再也不必自己使用一系列的开源软件去从头搭建容器服务平台。Rancher提供了在生产环境中使用的管理Docker和Kubernetes的全栈化容器部署与管理平台。

Rancher由以下四个部分组成：

基础设施编排

Rancher可以使用任何公有云或者私有云的Linux主机资源。Linux主机可以是虚拟机，也可以是物理机。Rancher仅需要主机有CPU，内存，本地磁盘和网络资源。从Rancher的角度来说，一台云厂商提供的云主机和一台自己的物理机是一样的。

Rancher为运行容器化的应用实现了一层灵活的基础设施服务。Rancher的基础设施服务包括网络， 存储， 负载均衡， DNS和安全模块。Rancher的基础设施服务也是通过容器部署的，所以同样Rancher的基础设施服务可以运行在任何Linux主机上。

容器编排与调度

很多用户都会选择使用容器编排调度框架来运行容器化应用。Rancher包含了当前全部主流的编排调度引擎，例如Docker Swarm， Kubernetes， 和Mesos。同一个用户可以创建Swarm或者Kubernetes集群。并且可以使用原生的Swarm或者Kubernetes工具管理应用。

除了Swarm，Kubernetes和Mesos之外，Rancher还支持自己的Cattle容器编排调度引擎。Cattle被广泛用于编排Rancher自己的基础设施服务以及用于Swarm集群，Kubernetes集群和Mesos集群的配置，管理与升级。

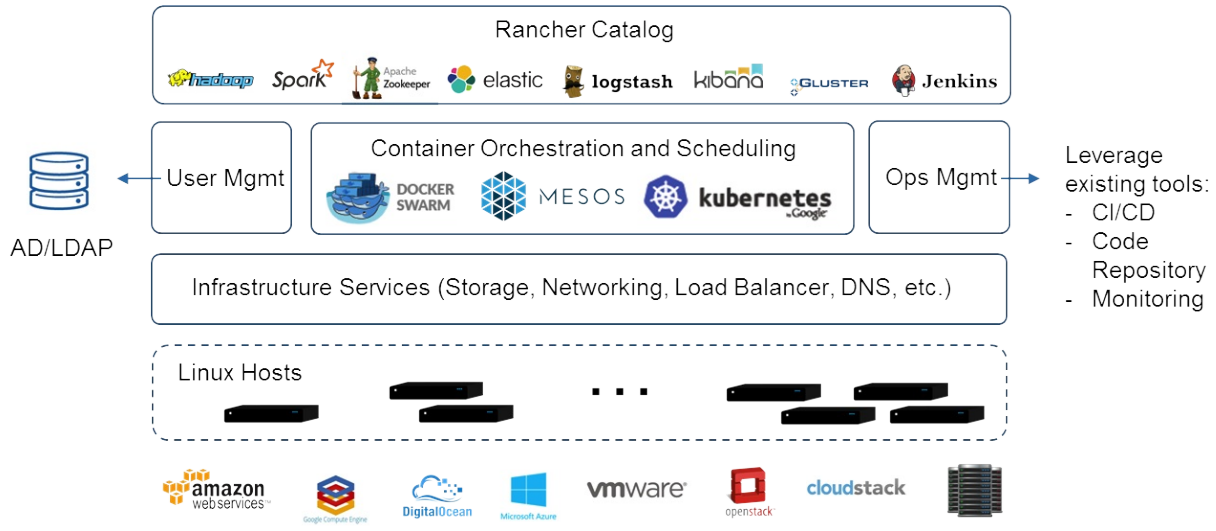
应用商店

Rancher的用户可以在应用商店里一键部署由多个容器组成的应用。用户可以管理这个部署的应用，并且可以在这个应用有新的可用版本时进行自动化的升级。Rancher提供了一个由Rancher社区维护的应用商店，其中包括了一系列的流行应用。Rancher的用户也可以创建自己的私有应用商店。

企业级权限管理

Rancher支持灵活的插件式的用户认证。支持Active Directory，LDAP， Github等 认证方式。 Rancher支持在环境级别的基于角色的访问控制 (RBAC)，可以通过角色来配置某个用户或者用户组对开发环境或者生产环境的访问权限。

下图展示了Rancher的主要组件和功能：



快速安装指南

在本节中，我们将进行简单快速的Rancher安装，即在一台Linux机器上安装Rancher，并使其能够完成所有Rancher必要功能。

准备LINUX主机

先安装一个64位的Ubuntu 16.04 Linux主机，其内核必须高于3.10。你可以使用笔记本、虚拟机或物理服务器。请确保该Linux主机内存不低于1GB。在该主机上安装 支持的Docker版本。

在主机上安装Docker的方法请参照Docker网站的安装说明。

> 注意： 目前Rancher尚不支持Docker for Windows以及Docker for Mac。

RANCHER SERVER标签

Rancher Server当前版本中有2个不同的标签。对于每一个主要的release标签，我们都会提供对应版本的文档。

- rancher/server:latest 此标签是我们的最新一次开发的构建版本。这些构建已经被我们的CI框架自动验证测试。但这些release并不代表可以在生产环境部署。
- rancher/server:stable 此标签是我们最新一个稳定的release构建。这个标签代表我们推荐在生产环境中使用的版本。

请不要使用任何带有 rc{n} 前缀的release。这些构建都是Rancher团队的测试构建。

启动RANCHER SERVER

你只需要一条命令就可以启动Rancher Server。当Rancher Server容器启动以后，我们将能查看到相关的日志。

```
$ sudo docker run -d --restart=unless-stopped -p 8080:8080 rancher/server:stable
# Tail the logs to show Rancher
$ sudo docker logs -f <CONTAINER_ID>
```

启动Rancher Server只需要几分钟时间。当日志中显示 .... Startup Succeeded, Listening on port...的时候，Rancher UI就能正常访问了。配置一旦完成，这行日志就会立刻出现。需要注意的是，这一输出之后也许还会有其他日志，因此，在初始化过程中这不一定是最后一行日志。

Rancher UI的默认端口是 8080。所以为了访问UI，需打开http://:8080。需要注意的事，如果你的浏览器和Rancher Server是运行在同一主机上的，你需要通过主机的真实IP地址访问，比如 http://192.168.1.100:8080 ，而不是 http://localhost:8080 或http://127.0.0.1:8080, 以防在添加主机的时候使用了不可达的IP而出现问题。

> 注意：
>
> 1. 初始安装时Rancher的访问控制并未配置，任何能够访问你的IP地址的人，都可以访问你的UI和API。我们建议你配置 访问控制.
> 2. 国内的公有云主机，如果需要使用80和8080端口，需备案后才可以使用。

添加主机

在这里，为了简化操作，我们将添加运行着Rancher Server的主机为Rancher内的主机。在实际的生产环境中，请使用专用的主机来运行Rancher Server。

想要添加主机，首先你需要进入UI界面，点击基础架构，然后你将看到主机界面。点击添加主机，Rancher将提示你选择主机注册URL。这个URL是Rancher Server运行所在的URL，且它必须可以被所有你要添加的主机访问到——当Rancher Server会通过NAT防火墙或负载均衡器被暴露至互联网时，这一设定就非常重要了。如果你的主机有一个私有或本地的IP地址，比如192.168..，Rancher将提示一个警告信息，提醒你务必确保这个URL可以被主机访问到。

因为我们现在只会添加Rancher Server主机自身，你可以暂时忽略这些警告。点击保存。默认选择自定义选项，你将得到运行Rancher agent容器的Docker命令。这里还有其他的公有云的选项，使用这些选项，Rancher可以使用Docker Machine来启动主机。

Rancher UI会给你提供一些指示，比如你的主机上应该开放的端口，还有其他一些可供选择的信息。鉴于我们现在添加的是Rancher Server运行的主机，我们需要添加这个主机所使用的公网IP。页面上的一个选项提供输入此IP的功能，此选项会自动更新Docker命令中的环境变量参数以保证正确。

然后请在运行Rancher Server的主机上运行这个命令。

当你在Rancher UI上点击关闭按钮时，你会被返回到基础架构->主机界面。一两分钟之后，主机会自动出现在这里。

基础设施服务

当你第一次登陆Rancher时，你将自动进入默认环境。默认已经为此环境选择了Cattle环境模板来启动基础设施服务。要想充分利用Rancher的一些功能，如DNS、元数据、网络、健康检查，你需要启动这些基础设施服务。这些基础设施可以在应用栈 -> 基础设施中找到。在主机被添加至Rancher之前，这些栈会处于 unhealthy 状态。主机添加完成后，建议等到所有基础设施服务都处于active状态之后再添加服务。

在主机上，所有属于基础设施服务的容器将被隐藏，除非你单击“显示系统容器”复选框。

通过UI创建一个容器

导航到应用页面，如果你看到了欢迎屏幕，可以在欢迎屏幕中单击定义服务的按钮。如果你的Rancher设置中已有服务，你可以在任何现有应用中点击添加服务，或者创建一个新的应用来添加服务。应用只是将服务组合在一起的便捷方式。 如果需要创建新的应用，请单击添加应用，填写名称和描述，然后单击创建。 接着，在新的应用中单击添加服务。

给服务取个名字，比如“第一个服务”。你可以使用我们的默认设置，然后单击创建。Rancher将开始在主机上启动容器。不论你的主机IP地址是什么，第一个容器的IP地址都将在 10.42.. 的范围内，因为Rancher已使用ipsec基础设施服务创建了一个托管网络。各容器之间是通过这个托管网络进行跨主机通信的。

如果你单击第一个容器的下拉列表，你将可以进行各种管理操作，如停止容器、查看日志或访问容器控制台。

通过DOCKER原生CLI创建一个容器

Rancher会显示主机之上的所有容器，即使有些容器是在UI之外创建的。在主机的shell终端中创建一个容器。

```
$ docker run -d -it --name=second-container ubuntu:14.04.2
```

在UI中，你将看到第二个容器在你的主机上出现！

Rancher会对Docker守护进程中发生的事件做出反应，调整自己以反映现实情况。你可以在此了解更多通过Docker原生CLI使用Rancher的事宜。

如果你查看第二个容器的IP地址，你将发现它不在10.42.. 范围内。它的IP地址是Docker守护进程分配的常用IP地址。这是通过CLI创建Docker容器的预期行为。

如果我们想通过CLI创建一个Docker容器，但仍希望它使用Rancher托管网络的IP地址，该怎么做呢？我们只需要在命令中添加一个标签(io.rancher.container.network=true)，让Rancher知道你希望此容器成为托管网络的一部分。

```
$ docker run -d -it --label io.rancher.container.network=true ubuntu:14.04.2
```

创建一个多容器应用

上文中我们已经介绍了如何创建单个容器以及这些单个容器之间如何进行跨主机网络通信。然而，现实情况中，大多数应用程序其实是由多个服务构成的，而每个服务又是由多个容器构成的。比如说，一个LetsChat应用程序，就是由下列几项服务构成的：

1. 一个负载均衡器。负载均衡器把Internet流量转发给“LetsChat”应用程序。
2. 一个由两个“LetsChat”容器组成的web服务。
3. 一个由一个“Mongo”容器组成的数据库服务。

负载均衡器的目标是web服务（即LetsChat），Web服务将连接到数据库服务（即Mongo）。

在本节中，我们将介绍如何在Rancher中创建和部署LetsChat应用程序。

导航到应用页面，如果你看到了欢迎屏幕，可以在欢迎屏幕中单击定义服务的按钮。 如果你的Rancher设置中已有服务，你可以在任何现有应用中点击添加应用，来创建一个新的应用。填写名称和描述，然后单击创建。 接着，在新的应用中单击添加服务。

首先，我们将创建一个名为database的数据库服务，并使用mongo镜像。单击创建。你将立即被带到应用页面，页面中将包含新创建的数据库服务。

接下来，再次点击添加服务以添加其他服务。我们将添加一个LetsChat服务并链接到database服务。让我们使用名称web以及sdelements/lets-chat镜像。在UI中，我们可以移动滑块，将服务扩容至2个容器。在服务链接中，添加database服务并将其命名为mongo。就像Docker的做法一样，Rancher会在letschat容器里加入这个链接所需要的相关环境变量。单击创建。

最后，我们将创建我们的负载均衡器。单击添加服务按钮旁的下拉菜单图标。选择添加负载均衡。提供一个类似于letschatapplb这样的名字。输入访问端口（例如80端口），选择目标服务（即web），并选择目标端口（即8080端口）。web服务正在侦听8080端口。单击创建。

至此，我们的LetsChat应用程序已完成！在应用页面上，你可以查找到超链接形式的负载均衡所暴露端口。点击该链接，将会打开一个新的页面，你将能看到LetsChat应用程序了。

使用RANCHER CLI创建一个多容器应用程序

在本节中，我们将介绍如何使用我们的命名行工具Rancher CLI创建和部署跟上一节中我们创建的一样的LetsChat应用程序。

当在Rancher中创建服务时，Rancher CLI工具与颇受欢迎的Docker Compose工具的工作方式类似。 它接收相同的docker-compose.yml文件，并在Rancher上部署应用程序。 你可以在rancher-compose.yml文件中指定更多的属性，该文件将扩展并覆盖docker-compose.yml文件。

在上一节中，我们创建了一个具有一个负载均衡器的LetsChat应用程序。如果你已经在Rancher中创建了它，你可以直接在UI中下载这些文件，只需在应用的下拉菜单中选择导出配置即可。docker-compose.yml文件与 rancher-compose.yml 文件与下方示例类似：

DOCKER-COMPOSE.YML示例

```
version: '2'
services:
  letschatapplb:
    #If you only have 1 host and also created the host in the UI,
    # you may have to change the port exposed on the host.
    ports:
    - 80:80/tcp
    labels:
      io.rancher.container.create_agent: 'true'
      io.rancher.container.agent.role: environmentAdmin
    image: rancher/lb-service-haproxy:v0.4.2
  web:
    labels:
      io.rancher.container.pull_image: always
    tty: true
    image: sdelements/lets-chat
    links:
    - database:mongo
    stdin_open: true
  database:
    labels:
      io.rancher.container.pull_image: always
    tty: true
    image: mongo
    stdin_open: true
```

RANCHER-COMPOSE.YML示例

```
version: '2'
services:
  letschatapplb:
    scale: 1
    lb_config:
      certs: []
      port_rules:
      - hostname: ''
        path: ''
        priority: 1
        protocol: http
        service: web
        source_port: 80
        target_port: 8080
    health_check:
      port: 42
      interval: 2000
      unhealthy_threshold: 3
      healthy_threshold: 2
      response_timeout: 2000
  web:
    scale: 2
  database:
    scale: 1
```

在Rancher UI中单击下载CLI（该按钮位于页面的右下角），即可下载Rancher CLI二进制文件，Windows、Mac和Linux的二进制文件均可下载。

若想使用Rancher CLI在Rancher中启动服务，你需要设置一些环境变量。你需要在Rancher UI中创建一个账户API Key。单击API -> 密钥。单击添加账户API Key。填写一个名字，然后单击创建。保存Access Key（用户名）和Secret Key（密码）。通过运行rancher config配置RancherCLI，使用Rancher URL、Access Key和Secret Key。

```
# Configure Rancher CLI
$ rancher config
# Set the Rancher URL
URL []: http://<SERVER_IP>:8080/
# Set the access key, i.e. username
Access Key []: <accessKey_of_account_api_key>
# Set the secret key, i.e. password
Secret Key []:  <secretKey_of_account_api_key>
```

现在进入保存了docker-compose.yml和rancher-compose.yml 文件的目录中，运行下面这个命令。

```
$ rancher up -d -s NewLetsChatApp
```

在Rancher中，一个叫做NewLetsChatApp的应用将被创建，且所有服务都将在Rancher中运行起来。

安装 Rancher Server

Rancher是使用一系列的Docker容器进行部署的。运行Rancher跟启动两个容器一样简单。一个容器作为管理服务器部署，另外一个作为集群节点的Agent部署

- Rancher Server - 单容器部署 (non-HA)
- Rancher Server - 单容器部署 (non-HA) - 使用外置数据库
- Rancher Server - 单容器部署 (non-HA)- 挂载MySQL数据库的数据目录
- Rancher Server - 多节点的HA部署
- Rancher Server - 使用AWS的Elastic/Classic Load Balancer作为Rancher Server HA的负载均衡器
- Rancher Server - 使用TLS认证的AD/OPENLDAP
- Rancher Server - 在HTTP代理后方启动 Rancher Server
- Rancher Server - 通过SSL连接MySQL

> 注意： 你可以运行Rancher Server的容器的命令 docker run rancher/server --help 来获得所有选项以及帮助信息。

安装需求

- 所有安装有支持的Docker版本的现代Linux发行版。 RancherOS, Ubuntu, RHEL/CentOS 7 都是经过严格的测试。
  - 对于 RHEL/CentOS, 默认的 storage driver, 例如 devicemapper using loopback, 并不被Docker推荐。 请参考Docker的文档去修改使用其他的storage driver。
  - 对于 RHEL/CentOS, 如果你想使用 SELinux, 你需要安装额外的 SELinux 组件.
- 1GB内存
- 精确的时钟同步服务 (例如 ntpd)
- MySQL服务器需要 max_connections 的设置 > 150
  - MYSQL配置需求
    - 选项1: 用默认COMPACT选项运行Antelope
    - 选项2: 运行MySQL 5.7，使用Barracuda。默认选项ROW_FORMAT需设置成Dynamic
  - 推荐设定
    - max_packet_size >= 32M
    - innodb_log_file_size >= 256M (如果你已有现存数据库，请根据实际情况更改此设定)
    - innodb_file_per_table=1
    - innodb_buffer_pool_size >= 1GB (对于更高需求的配置，请在专属MySQL服务器机器上使用4-8G的值)

> 注意： 目前Rancher中并不支持Docker for Mac

RANCHER SERVER 标签

Rancher Server当前版本中有2个不同的标签。对于每一个主要的release标签，我们都会提供对应版本的文档。

- rancher/server:latest 此标签是我们的最新一次开发的构建版本。这些构建已经被我们的CI框架自动验证测试。但这些release并不代表可以在生产环境部署。
- rancher/server:stable 此标签是我们最新一个稳定的release构建。这个标签代表我们推荐在生产环境中使用的版本。

请不要使用任何带有 rc{n} 前缀的release。这些构建都是Rancher团队的测试构建。

启动 RANCHER SERVER - 单容器部署 (NON-HA)

在安装了Docker的Linux服务器上，使用一个简单的命令就可以启动一个单实例的Rancher。

```
$ sudo docker run -d --restart=unless-stopped -p 8080:8080 rancher/server
```

RANCHER UI

UI以及API会使用 8080 端口对外服务。下载Docker镜像完成后，需要1到2分钟的时间Rancher才能完全启动并提供服务。

访问如下的URL: http://:8080。 是运行Rancher Server的主机的公共IP地址。

当UI已经启动并运行，你可以先添加主机 或者在应用商店中选择一个容器编排引擎。在默认情况下，如果没有选择不同的容器编排引擎，当前环境会使用Cattle引擎。在主机被添加都Rancher中后，你可以开始添加服务或者从应用商店通过应用模版启动一个应用。

启动 RANCHER SERVER - 单容器部署 - 使用外部数据库

除了使用内部的数据库，你可以启动一个Rancher Server并使用一个外部的数据库。启动命令与之前一样，但添加了一些额外的参数去说明如何连接你的外部数据库。

> 注意： 在你的外部数据库中，只需要提前创建数据库名和数据库用户。Rancher会自动创建Rancher所需要的数据库表。

以下是创建数据库和数据库用户的SQL命令例子

```
> CREATE DATABASE IF NOT EXISTS cattle COLLATE = 'utf8_general_ci' CHARACTER SET = 'utf8';
> GRANT ALL ON cattle.* TO 'cattle'@'%' IDENTIFIED BY 'cattle';
> GRANT ALL ON cattle.* TO 'cattle'@'localhost' IDENTIFIED BY 'cattle';
```

启动一个Rancher连接一个外部数据库，你需要在启动容器的命令中添加额外参数。

```
$ sudo docker run -d --restart=unless-stopped -p 8080:8080 rancher/server \
    --db-host myhost.example.com --db-port 3306 --db-user username --db-pass password --db-name cattle
```

大部分的输入参数都有默认值并且是可选的，只有MySQL server的地址是必须输入的。

```
--db-host               IP or hostname of MySQL server
--db-port               port of MySQL server (default: 3306)
--db-user               username for MySQL login (default: cattle)
--db-pass               password for MySQL login (default: cattle)
--db-name               MySQL database name to use (default: cattle)
```

> 注意： 在之前版本的Rancher Server中，我们需要使用环境变量去连接外部数据库。在新版本中，这些环境变量会继续生效，但Rancher建议使用命令参数代替。

启动 RANCHER SERVER - 单容器部署 - 挂载MYSQL数据库的数据目录

在Rancher Server容器中，如果你想使用一个主机上的卷来持久化数据库，如下命令可以在启动Rancher时挂载MySQL的数据卷。

```
$ sudo docker run -d -v <host_vol>:/var/lib/mysql --restart=unless-stopped -p 8080:8080 rancher/server
```

使用这条命令，数据库就会持久化在主机上。如果你有一个现有的Rancher Server容器并且想挂在MySQL的数据卷，可以参考以下的Rancher升级介绍。

启动 RANCHER SERVER - 多节点的HA部署

在高可用(HA)的模式下运行Rancher Server与使用外部数据库运行Rancher Server一样简单，需要暴露一个额外的端口，添加额外的参数到启动命令中，并且运行一个外部的负载均衡就可以了。

HA部署需求

- HA 节点:
  - 所有安装有支持的Docker版本的现代Linux发行版 RancherOS, Ubuntu, RHEL/CentOS 7 都是经过严格的测试。
    - 对于 RHEL/CentOS, 默认的 storage driver, 例如 devicemapper using loopback, 并不被Docker推荐。 请参考Docker的

文档去修改使用其他的storage driver。
    - 对于 RHEL/CentOS, 如果你想使用 SELinux, 你需要 安装额外的 SELinux 组件.
  - 9345, 8080 端口需要在各个节点之间能够互相访问
  - 1GB内存
- MySQL数据库
  - 至少 1 GB内存
  - 每个Rancher Server节点需要50个连接 (例如：3个节点的Rancher则需要至少150个连接)
  - MYSQL配置要求
    - 选项1: 用默认COMPACT选项运行Antelope
    - 选项2: 运行MySQL 5.7，使用Barracuda。默认选项ROW_FORMAT需设置成Dynamic
- 外部负载均衡服务器
  - 负载均衡服务器需要能访问Rancher Server节点的 8080 端口

> 注意： 目前Rancher中并不支持Docker for Mac

大规模部署建议

- 每一个Rancher Server节点需要有4 GB 或者8 GB的堆空间，意味着需要8 GB或者16 GB内存
- MySQL数据库需要有高性能磁盘
- 对于一个完整的HA，建议使用一个有副本的Mysql数据库。另一种选择则是使用Galera集群并强制写入一个MySQL节点。

在每个需要加入Rancher Server HA集群的节点上，运行以下命令：

```
# Launch on each node in your HA cluster
$ docker run -d --restart=unless-stopped -p 8080:8080 -p 9345:9345 rancher/server \
     --db-host myhost.example.com --db-port 3306 --db-user username --db-pass password --db-name cattle \
     --advertise-address <IP_of_the_Node>
```

在每个节点上， 需要在每个节点上唯一，因为这个IP会被添加到HA的设置中。

如果你修改了 -p 8080:8080 并在host上暴露了一个不一样的端口，你需要添加 --advertise-http-port 参数到命令中。

> 注意： 你可以使用 docker run rancher/server --help 获得命令的帮助信息

配置一个外部的负载均衡器，这个负责均衡负责将例如80或443端口的流量，转发到运行Rancher Server的节点的8080端口中。负载均衡器必须支持websockets 以及 forwarded-for 的Http请求头以支持Rancher的功能。参考 使用SSL 这个配置的例子。

ADVERTISE-ADDRESS选项

选项	例子	描述
IP address	--advertise-address 192.168.100.100	使用指定IP
Interface	--advertise-address eth0	从指定网络接口获取
awslocal	--advertise-address awslocal	从这里获取http://169.254.169.254/latest/meta-data/local-ipv4
ipify	--advertise-address ipify	从这里获取https://api.ipify.org

HA模式下的RANCHER SERVER节点

如果你的Rancher Server节点上的IP修改了，你的节点将不再存在于Rancher HA集群中。你必须停止在--advertise-address配置了不正确IP的Rancher Server容器并启动一个使用正确IP地址的Rancher Server的容器。

使用AWS的ELASTIC/CLASSIC LOAD BALANCER作为RANCHER SERVER HA的负载均衡器

我们建议使用AWS的ELB作为你Rancher Server的负载均衡器。为了让ELB与Rancher的websockets正常工作，你需要开启proxy protocol模式并且保证HTTP support被停用。 默认的，ELB是在HTTP/HTTPS模式启用，在这个模式下不支持websockets。需要特别注意listener的配置。

如果你在配置ELB中遇到问题，我们建议你参考terraform version。

> 注意： 如果你正在使用自签名的证书, 请参考我们SSL部分里的如何在AWS里配置ELB.

LISTENER 配置 - PLAINTEXT

简单的来说，使用非加密的负载均衡，需要以下的listener配置：

Configuration Type	Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port
Plaintext	TCP	80	TCP	8080 (或者使用启动Rancher Server时 --advertise-http-port 指定的端口)

启用 PROXY PROTOCOL

为了使websockets正常工作，ELB的proxy protocol policy必须被启用。

- 启用 proxy protocol 模式

```
$ aws elb create-load-balancer-policy --load-balancer-name <LB_NAME> --policy-name <POLICY_NAME> --policy-type-name Proxy
ProtocolPolicyType --policy-attributes AttributeName=ProxyProtocol,AttributeValue=true
$ aws elb set-load-balancer-policies-for-backend-server --load-balancer-name <LB_NAME> --instance-port 443 --policy-names
 <POLICY_NAME>
$ aws elb set-load-balancer-policies-for-backend-server --load-balancer-name <LB_NAME> --instance-port 8080 --policy-name
s <POLICY_NAME>
```

- Health check可以配置使用HTTP:8080下的 /ping 路径进行健康检查

使用TERRAFORM进行配置

以下是使用Terraform配置的例子：

```
resource "aws_elb" "lb" {
  name               = "<LB_NAME>"
  availability_zones = ["us-west-2a","us-west-2b","us-west-2c"]
  security_groups = ["<SG_ID>"]

  listener {
    instance_port     = 8080
    instance_protocol = "tcp"
    lb_port           = 443
    lb_protocol       = "ssl"
    ssl_certificate_id = "<IAM_PATH_TO_CERT>"
  }

}

resource "aws_proxy_protocol_policy" "websockets" {
  load_balancer  = "${aws_elb.lb.name}"
  instance_ports = ["8080"]
}
```

使用AWS的APPLICATION LOAD BALANCER(ALB) 作为RANCHER SERVER HA的负载均衡器

我们不再推荐使用AWS的Application Load Balancer (ALB)替代Elastic/Classic Load Balancer (ELB)。如果你依然选择使用ALB，你需要直接指定流量到Rancher Server节点上的HTTP端口，默认是8080。

使用TLS认证的AD/OPENLDAP

为了在Rancher Server上启用Active Directory或OpenLDAP并使用TLS，Rancher Server容器在启动的时候需要配置LDAP证书，证书是LDAP服务提供方提供。证书保存在需要运行Rancher Server的Linux机器上。

启动Rancher并挂载证书。证书在容器内部 必须 命名为ca.crt。

```
$ sudo docker run -d --restart=unless-stopped -p 8080:8080 \
  -v /some/dir/cert.crt:/var/lib/rancher/etc/ssl/ca.crt rancher/server
```

你可以使用Rancher Server的日志检查传入的 ca.crt 证书是否生效

```
$ docker logs <SERVER_CONTAINER_ID>
```

在日志的开头，会显示证书已经被正确加载的信息。

```
Adding ca.crt to Certs.
Updating certificates in /etc/ssl/certs... 1 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d....done.
Certificate was added to keystore
```

在HTTP代理后方启动 RANCHER SERVER

为了设置HTTP Proxy，Docker守护进程需要修改配置并指向这个代理。在启动Rancher Server前，需要编辑配置文件 /etc/default/docker 添加你的代理信息并重启Docker服务。

```
$ sudo vi /etc/default/docker
```

在文件中，编辑 #export http_proxy="http://127.0.0.1:3128/" 并修改它指向你的代理。保存修改并重启Docker。重启Docker的方式在每个OS上都不一样。

> 注意： 如果你使用systemd运行Docker, 请参考Docker官方的文档 去配置http proxy设置。

为了使得应用商店加载正常，HTTP代理设置必须在Rancher Server运行的环境变量中。

```
$ sudo docker run -d \
    -e http_proxy=<proxyURL> \
    -e https_proxy=<proxyURL> \
    -e no_proxy="localhost,127.0.0.1" \
    -e NO_PROXY="localhost,127.0.0.1" \
    --restart=unless-stopped -p 8080:8080 rancher/server
```

如果你不使用应用商店，则使用你平常的Rancher Server命令即可。

当向Rancher添加主机时，在HTTP代理中不需要额外的设置和要求。

通过SSL连接MYSQL的RANCHER SERVER

> 注意： 目前在Rancher 1.6.3以上版本才支持

重要提示

如果你正在使用LDAP或者AD认证方式，并且这些认证方式的证书发放方CA并不是MySQL服务器SSL的证书发放方CA，这篇指南无法适用于你的情况。

前提条件

- MySQL服务器的证书或CA证书

步骤

1. 拷贝MySQL服务器的证书或CA证书到Rancher Server的主机上。当启动rancher/server容器的时候你必须将证书挂载到/var/lib/rancher/etc/ssl/ca.crt。
2. 更改以下的模板的对应参数，构建一个JDBC URL: jdbc:mysql://:/?useUnicode=true&characterEncoding=UTF-8&characterSetResults=UTF-8&prepStmtCacheSize=517&cachePrepStmts=true&prepStmtCacheSqlLimit=4096&socketTimeout=60000&connectTimeout=60000&sslServerCert=/var/lib/rancher/etc/ssl/ca.crt&useSSL=true
3. 使用环境变量CATTLE_DB_CATTLE_MYSQL_URL和CATTLE_DB_LIQUIBASE_MYSQL_URL来导入上面的JDBC URL到容器里面。
4. 加入环境变量CATTLE_DB_CATTLE_GO_PARAMS="tls=true"到容器里面。但是如果服务器证书的标题名字不符合服务器的主机名，你需要使用的是CATTLE_DB_CATTLE_GO_PARAMS="tls=skip-verify".

例子

```
$ export JDBC_URL="jdbc:mysql://<DB_HOST>:<DB_PORT>/<DB_NAME>?useUnicode=true&characterEncoding=UTF-8&characterSetResults=UTF-8&prepStmtCacheSize=517&cachePrepStmts=true&prepStmtCacheSqlLimit=4096&socketTimeout=60000&connectTimeout=60000&sslServerCert=/var/lib/rancher/etc/ssl/ca.crt&useSSL=true"

$ cat <<EOF > docker-compose.yml
version: '2'
  services:
    rancher-server:
      image: rancher/server:stable
      restart: unless-stopped
      command: --db-host <DB_HOST> --db-port <DB_PORT> --db-name <DB_NAME> --db-user <DB_USER> --db-pass <DB_PASS>
      environment:
        CATTLE_DB_LIQUIBASE_MYSQL_URL: $JDBC_URL
        CATTLE_DB_CATTLE_MYSQL_URL: $JDBC_URL
        CATTLE_DB_CATTLE_GO_PARAMS: "tls=true"
      volumes:
        - /path/to/mysql/ca.crt:/var/lib/rancher/etc/ssl/ca.crt
      ports:
        - "8080:8080"
EOF

$ docker-compose up -d
```

重要: 你必须在两个环境变量里都写入构建好的JDBC_URL，还必须加入--db-xxx参数!

升级Rancher Server

> 注意： 如果你正准备升级到v1.6.x，请阅读我们相关的版本注解v1.6.0。这里面有相关升级需要的注意事项。根据你安装Rancher Server方式的不同，你的升级步骤可能不一样。

- Rancher服务 - 单个容器(non-HA)
- Rancher服务 - 单个容器(non-HA) - 外部数据库
- Rancher服务 - 单个容器(non-HA) - 绑定挂载MySQL卷
- Rancher服务 - 多活HA模式
- Rancher服务 - 无因特网访问

> 注意： 如果你在原始的Rancher服务中设置了任何的环境变量或者传了一个ldap证书，则需要在任何新的命令中添加这些环境变量或者证书。

RANCHER SERVER标签

Rancher Server当前版本中有2个不同的标签。对于每一个主要的release标签，我们都会提供对应版本的文档。

- rancher/server:latest 此标签是我们的最新一次开发的构建版本。这些构建已经被我们的CI框架自动验证测试。但这些release并不代表可以在生产环境部署。
- rancher/server:stable 此标签是我们最新一个稳定的release构建。这个标签代表我们推荐在生产环境中使用的版本。 请不要使用任何带有 rc{n} 前缀的release。这些构建都是Rancher团队的测试构建。

基础设施服务

当Rancher Server升级之后，你的基础设施服务可能也需要升级。我们建议在升级Rancher Server之后检查一下基础设施服务，看是否有可升级的。如果有可升级的，那么按照下面的顺序一个个升级：

1. network-policy-manager (如果安装了，这是一个可选的Rancher组件)
2. network-services
3. ipsec
4. 剩余的基础设施服务

> 注意： 确保在升级一个基础设施服务之前，它已经完成了之前的升级，这个是很重要的。升级完成之后，在栈菜单中选择“完成升级”，然后继续。

有时候，Rancher会要求你升级其中的一个或者多个基础设施服务，以便Rancher继续工作。你可以通过API设置来修改升级策略，以防止自动升级，但不推荐。

从v1.6.1开始

我们引入了一个API设置，可以允许你控制基础设施服务的升级策略。upgrade.manager设置接受3个值。

- mandatory - 这个是默认的值。该值只会自动升级必需要升级的基础设施服务，以使Rancher Server正常工作。
- all - 任何可用于基础设施服务的更新模版都将会自动升级。如果基础设施服务具有新的模板版本，但是基础设施服务的默认版本仍然较旧，则不会自动升级到最新版本。
- none - 没有基础设施服务将会升级。 警告： 这可能导致你的Rancher Server停止运行，因为它可能阻止了必要的基础设施服务升级

RANCHER AGENTS

每个Rancher agent版本都对应于一个固定的Rancher Server版本。如果升级了Rancher服务并且Rancher代理也需要升级，我们将自动将代理升级到最新版本。

单独升级一个容器(NON-HA)

如果你没有使用外部DB或绑定的MySQL卷来启动Rancher Server，则Rancher Server数据库位于Rancher Server容器中。我们将使用正在运行的Rancher Server容器来创建一个数据容器，该数据容器将用于使用--volumes-from启动新的Rancher Server容器。或者，你可以将数据库从容器中复制到主机上的目录并绑定挂载数据库。

停掉容器

```
$ docker stop <container_name_of_original_server>
```

创建一个rancher-data容器。注意：如果你已经升级了并且已经有了一个rancher-data容器，该步可以跳过。

```
$ docker create --volumes-from <container_name_of_original_server> \
 --name rancher-data rancher/server:<tag_of_previous_rancher_server>
```

拉取Rancher Server的最新镜像。注意：如果你跳过该步并尝试运行latest镜像，这将不会自动拉取最新的镜像。

```
$ docker pull rancher/server:latest
```

用rancher-data中的数据库启动一个Rancher Server容器。启动之后，Rancher中的任何变化将会被保存在rancher-data容器中。如果你在服务器中看到有关日志锁的异常，请参考如何修复日志锁。

> 注意： 根据你Rancher Server时间的长短，某些数据库迁移可能需要比预期的更长的时间。 升级过程中请不要停止升级，因为下次升级时会遇到数据库迁移错误。 bash $ docker run -d --volumes-from rancher-data --restart=unless-stopped \ -p 8080:8080 rancher/server:latest

删掉旧的Rancher Server容器。注意：如果你只是停止了容器，当你使用--restart=always，并且机器重启之后，该容器将会重启。我们建议使用--restart=unless-stopped并且当升级成功之后删除它。

单独升级一个容器(NON-HA) - 外部数据库

如果你使用外部数据库启动Rancher Server，你可以先停止原来的Rancher Server容器，并使用相同的使用外部数据库的安装说明。升级你的Rancher Server之前，建议你备份外部数据库。 新服务器启动并运行后，可以删除旧的Rancher Server容器。

单独升级一个容器(NON-HA) - 绑定挂载的MYSQL卷

停掉正在运行的Rancher Server容器

```
$ docker stop <container_name_of_original_server>
```

将数据库文件从服务器容器中复制出来。注意：如果已经将数据库存储在主机上，则可以跳过此步骤。另外，如果将DB复制出来，根据Docker复制出来的方式，它将会在 / /mysql/里面。当挂载到容器中时，一定要考虑到这一点。如果你启动的时候绑定挂载，则不需要mysql /

```
$ docker cp <container_name_of_original_server>:/var/lib/mysql <path on host>
```

现在为文件夹设置UID/GID，以便容器内的mysql用户拥有正确的mysql mount的所有权。

```
$ sudo chown -R 102:105 <path on host>
```

启动新的服务器容器

```
$ docker run -d -v <path_on_host>:/var/lib/mysql -p 8080:8080 \
  --restart=unless-stopped rancher/server:latest
```

> 注意： 如果已经从先前的容器中复制了数据库，那么在主机路径的末尾必需加上'/'，否则，目录的位置会出错。

删掉旧的Rancher Server容器。注意：如果你只是停止了容器，当你使用--restart=always，并且机器重启之后，该容器将会重启。我们建议使用--restart=unless-stopped并且当升级成功之后删除它。

升级HA架构

当以高可用(HA)的方式启动Rancher Server，新的Rancher HA设置将继续使用用于安装原始HA设置的外部数据库。

> 注意： 当升级HA架构的Rancher Server时，Rancher Server在升级过程中将会停止服务。

升级你的Rancher Server之前，建议你备份外部数据库。

在HA架构中的每台Server节点上，停止并删除正在运行的Rancher Server容器，然后按照相同的安装HA模式的Rancher Server说明来启动一个新的Rancher服务容器，但是使用的是一个新的Rancher Server镜像版本。

```
# On all nodes, stop all Rancher Server containers
$ docker stop <container_name_of_original_server>
# Execute the scrip with the latest rancher/server version
$ docker run -d --restart=unless-stopped -p 8080:8080 -p 9345:9345 rancher/server --db-host myhost.example.com --db-port 3306
--db-user username --db-pass password --db-name cattle --advertise-address <IP_of_the_Node>
```

> 注意： 当你正在一个运行Rancher Server 1.2之前版本的HA时，你需要删除所有的正在运行的Rancher HA容器。$ sudo docker rm -f $(sudo docker ps -a | grep rancher | awk {'print $1'})

没有互联网访问的RANCHER SERVER

在没有互联网的情况下，为了能够升级成功，用户需要下载最新的基础设施服务镜像 。

访问控制

什么是访问控制？

访问控制是用来控制哪些用户可以访问你的Rancher服务。在默认情况下，Rancher没有启用访问控制。这意味着知道你的Rancher服务IP的人都可以访问你的Rancher服务和API。你的Rancher服务是对外开放的！我们强烈建议你在启动Rancher后立即配置访问控制，这样你可以按照需要分享你的Rancher服务。用户在访问你的Rancher服务之前，需要进行身份认证。同时，只有拥有合法的API密钥才能使用Rancher API。

Rancher认证的第一个账户将成为 管理员 账户。 想要获取有关详细信息，请参阅 管理员权限。

启用访问控制

在 系统管理 选项卡中, 点击访问控制。

在你配置了Rancher的访问控制后，访问控制将被启用。访问控制使你能够管理不同的环境并把它们分享给不同的个人或团队。

当访问控制启用后，Rancher API将被锁定。这时需要用户进行身份认证， 或者使用API 密钥来访问它。

ACTIVE DIRECTORY

选择ACTIVE DIRECTORY图标。 如果你想要通过TLS来使用ACTIVE DIRECTORY，请确保你已经使用了相应的证书来启动Rancher Server。填写相关信息后，通过点击身份认证进行认证校验。 当ACTIVE DIRECTORY认证成功后，你将自动以已认证的用户名身份登录，同时把你的账号会被设置为管理员。

用户SEARCH BASE与组SEARCH BASE 在配置ACTIVE DIRECTORY时，你将需要输入用户的SEARCH BASE。 此SEARCH BASE允许Rancher搜索你设置的ACTIVE DIRECTORY中的用户。如果你的用户和用户组位于相同的SEARCH BASE中，那么你仅仅需要填写用户的SEARCH BASE，但是如果你的用户组在不同的SEARCH BASE，你可以把该SEARCH BASE填写在用户组SEARCH BASE字段下。 此字段专用于用户组搜索，该项是可选的。

AZURE AD

选择Azure AD图标。 填写相应信息并单击Azure认证进行认证校验。 当认证成功后，你将自动以已认证的用户名身份登录。并且把你的账号设置为了管理员权限。

GITHUB

选择GitHub图标，并按照用户界面中的说明将Rancher注册为GitHub应用程序。 点击使用GitHub进行身份认证后，当认证成功后，你将自动以已认证的Github账号登录。并且把你的账号设置为了管理员权限。

本地身份认证

本地身份认证允许你创建自己的一组账户，这些账户存储在Rancher数据库中。

选择本地图标。 通过提供登录用户名，全名和密码来创建管理员用户。 点击启用本地认证来启用本地身份认证。 通过单击此按钮，管理员用户将被创建并保存在数据库中。这时你将自动用刚刚创建的管理员帐户登录到Rancher服务。

OPENLDAP

填写对应信息后，通过点击身份认证进行认证校验。当OpenLDAP认证成功后，你将自动以已认证的用户名身份登录。并且把你的账号设置为了管理员权限。

用户SEARCH BASE与组SEARCH BASE 在配置ACTIVE DIRECTORY时，你将需要输入用户的SEARCH BASE。 此SEARCH BASE允许Rancher搜索你设置的ACTIVE DIRECTORY中的用户。如果你的用户和用户组位于相同的SEARCH BASE中，那么你仅仅需要填写用户的SEARCH BASE，但是如果你的用户组在不同的SEARCH BASE，你可以把该SEARCH BASE填写在用户组SEARCH BASE字段下。 此字段专用于用户组搜索，该项是可选的。

SHIBBOLETH

选择Shibboleth图标。 填写Shibboleth帐户的配置信息，点击保存保存信息，然后点击测试来测试访问控制是否正常工作。

在使用Shibboleth时，你应该注意一些已知的问题。

- 不支持搜索或查找功能。 在添加用户时，请确保输入的用户ID是准确的，这样才能保证用户被添加成功。
- 当添加用户到一个环境时, 不支持组ID，除非管理员是该组的成员之一。

站点访问

根据你的身份认证类型，Rancher提供不同级别的站点访问。

活动目录/GITHUB/SHIBBOLETH

如果你已通过AD或GitHub进行身份认证，则将有3个选项可用。

- 允许任何合法用户 - GitHub或活动目录中的任何用户都可以访问你的Rancher服务。 不推荐将此设置用于GitHub，因为它将使GitHub中的任何用户都可以访问Rancher服务。
- 允许环境成员和已授权用户和组织 - 一个环境的成员用户或拥有者用户和添加到已授权用户和用户组的用户一样，都有权限访问Rancher服务。
- 限制访问只有已授权用户和用户组可以访问 - 只有添加到已授权用户和用户组的用户才能访问Rancher服务。 即使用户已被添加到环境中，如果没有被添加到已授权用户和用户组，他们将仍然无法访问Rancher服务。

任何具有Rancher服务访问权限的人都将被授予 用户权限。他们将无法查看系统管理页面。 如果想要他们查看，你将需要明确地将其帐户更改为管理员帐户。

为了让用户查看不同的环境, 他们将需要被环境的所有者添加到环境中。

AZURE AD/OPENLDAP

对于Azure AD和OpenLDAP，你的设置中的任何用户都可以访问Rancher站点。

本地身份认证

启用本地身份认证后，管理员可以通过访问系统管理> 账号设置选项卡来创建其他管理员/用户。 点击添加账号并填写你要添加的帐户的详细信息。 你可以选择其帐户类型为管理员或用户。 管理员可以查看系统管理页面，普通用户无法看到该页面。

一旦帐户被创建后，该账户可以被添加到任何环境中。

账户类型

帐户类型决定帐户是否可以访问系统管理页面。对于Rancher中的每个环境，可以设置不同级别的成员角色来对特定环境进行访问。

管理员

认证Rancher的第一个用户成为Rancher的管理员。 只有管理员才有权限查看系统管理页面

在管理环境时，管理员可以查看Rancher中的所有环境， 即使管理员没有被加入到该环境的成员中。 在非管理员的环境下拉菜单中，用户只能看到他们所在的环境。

管理员可以将其他用户添加为Rancher管理员。 在用户登录Rancher后，他们可以在 系统管理 > 账号设置页面上更改用户角色。 在系统管理> 账号设置帐户标签中，点击帐户名称旁边的编辑，并将帐户类型更改为管理员。 点击保存。

用户

除了用来启用Rancher认证的用户外，任何其他用户都将自动拥有用户权限。 他们将无法看到系统管理页面

他们只能看到他们所在的环境。

禁用访问控制

如果你决定不再需要访问控制，请单击禁用访问控制按钮。 这将使你的Rancher服务向公众开放，任何人都可以访问你的API。 这是不推荐的。

配置会话超时

默认情况下，会话在其创建后16小时到期。 如果你觉得时间太长，可以更新会话到期时间。

1. 点击系统管理 -> 系统设置 -> 高级设置, 点击 我确认已经知道修改高级设置可能导致问题。
2. 找到 api.auth.jwt.token.expiry设置，然后点击修改按钮。
3. 更新超时会话值后，然后单击保存按钮。 值以毫秒为单位。

限制并发登陆

从v1.6.3版开始支持

1. 在默认情况下，用户可以同时在线。你可以通过API来禁用同时在线的功能。 api.auth.restrict.concurrent.sessions.
2. 找到系统管理 -> 系统设置 -> 高级选项, 点击 我确认已经知道修改高级设置可能导致问题.
3. 找到api.auth.restrict.concurrent.sessions设置，然后点击修改按钮。 将值设置为true，然后点击保存。

一旦该设置生效，在用户下次登陆的时候，之前的登陆口令将会被删除。这样，之前登陆的用户将会被登出。

账户

什么是账户?

有权访问Rancher的每个用户都有Rancher帐号。对于本地身份认证设置，你可以为用户创建帐户，对于其他身份认证方式，当用户登录到Rancher时，会为该用户自动创建一个帐户。

AD域/GITHUB/OPENLDAP验证

当AD域，Azure AD，GitHub，或者OpenLDAP验证开启的时候，帐户选项卡显示已登录并针对Rancher进行身份认证的用户列表。为了登录，它们必须被赋予访问站点的权限或被添加到环境中。

本地验证

启用本地身份认证后，你可以在账号设置中添加用户。点击添加帐号按钮将帐号添加到Rancher数据库。创建帐号时，帐号类型可以指定为管理员或用户。

账户类型

Rancher通过帐户类型确定用户是否可以访问系统管理页面。对于Rancher中的每个环境，都可以通过设置账号的成员角色来为环境实现不同级别的访问控制。

管理员

认证Rancher的第一个用户成为Rancher的管理员。 只有管理员才有权限查看系统管理页面

在管理环境时，管理员可以查看Rancher中的所有环境， 即使管理员没有被加入到该环境的成员中。 在非管理员的环境下拉菜单中，用户只能看到他们所在的环境。

管理员可以将其他用户添加为Rancher管理员。 在用户登录Rancher后，他们可以在 系统管理 > 账号设置页面上更改用户角色。 在系统管理> 账号设置帐户标签中，点击帐户名称旁边的编辑，并将帐户类型更改为管理员。 点击保存。

用户

除了用来启用Rancher认证的用户外，任何其他用户都将自动拥有用户权限。 他们将无法看到系统管理页面

他们只能看到他们所在的环境。

系统设置

在Rancher的系统管理 -> 系统设置页面，你可以定制Rancher产品。

主机注册

在启动任何主机之前，你需要完成主机注册地址设置。主机注册地址是用来设置Rancher Server将如何与主机进行连接。如果你已经设置了访问控制，则不会提示你设置主机注册地址，因为Rancher假定你的URL是可以被访问到的。

Rancher通过该设置确定你的主机连接Rancher API所用的URL。默认情况下，Rancher将使用访问UI的URL。如果你选择更改注册地址，请确保设置用于连接到Rancher API的端口。如果你使用配置了SSL的Rancher Server，请确保将协议更改为https。此注册地址设置决定了添加自定义主机的命令。

如果为Rancher启用了访问控制功能，则只有管理员才能更改主机注册地址。默认情况下，第一个管理员是配置启用访问控制的用户。如果仍未配置访问控制，则该任何用户都可以更新主机注册地址。可以在系统设置 -> 主机注册地址选项卡中更新此选项。

应用商店

默认情况下，应用商店有三类可使用的应用：

- Rancher基础设施所有基础设施服务的模版。
- Rancher官方认证包括Rancher认证过的应用模版。
- Rancher社区共享包含社区共享的模版。

由于系统设置页面只有管理员有访问权限，因此只有管理员有权限添加私有的应用商店。添加一个应用商店就是添加一个应用名称和git地址。正确的git地址的格式可以参考这里。无论什么时候添加应用商店，你都可以马上使用它。

如果要添加自己创建的私有应用商店，那么git仓库的文件必须遵守Rancher应用商店的特定格式。

信息统计

默认情况下，Rancher会询问你是否允许收集匿名统计信息。这些数据使我们更好的了解我们的用户群，帮助改进Rancher产品。点击这里了解更多。

主机驱动

Docker-machine驱动可被添加到Rancher中，以便这些驱动可以将主机添加到Rancher中。只有管理员可以设置哪些主机驱动可见，这个在系统管理 -> 主机驱动。

只有启用的主机驱动才能在基础架构 -> 添加主机的页面上显示出来。默认情况下，Rancher提供了许多主机驱动，但是只有一些是启用状态。

添加主机驱动

你可以通过点击添加主机驱动轻松添加自己的主机驱动。

1. 提供下载URL。这个地址是64位Linux驱动的二进制文件的地址。
2. (可选) 为驱动提供自定义添加主机界面的自定义UI的URL。参考ui-driver-skel repository以了解更多信息。
3. (可选) 提供一个校验和以检验下载的驱动是否匹配期望的校验和。
4. 完成之后，点击创建。

点击创建后，Rancher就会添加这个额外的驱动，并将其显示在添加主机页面的驱动选项里。

环境

什么是一个环境?

Rancher 支持将资源分组归属到多个环境。每个环境具有自己独立的基础设施资源及服务，并由一个或多个用户、团队或组织管理。例如，你可以创建独立的“开发”、“测试”及“生产”环境以确保环境之间的安全隔离，将“开发”环境的访问权限赋予全部人员，但限制“生产”环境的访问权限给一个小的团队。

所有主机和Rahcner资源, 比如容器, 基础设施服务等, 都在环境中被创建, 并且属于一个环境。

添加环境

要添加一个环境，把鼠标移动到位于左上角的当前环境， 此时会出现一个带有所有可用的环境下拉框，以及一个 环境管理 连接。点击 环境管理。

导航到 环境 页面后，你会看到一个环境列表和一个环境模板列表。如果你是 Rancher 的 管理员用户，你会看到一个所有环境的列表，即使你不是该环境的成员。任何环境模板都对所有用户可见。

点击 添加环境。每个环境都有自己的名字和描述，你可以选择你要使用的环境模板。在环境模板中，你可以看到哪个基础设施服务是启用的。

> 注意： 如果没有配置 访问控制， 所有环境都可以被其它 Rancher 的用户访问到。 环境没有任何所属关系。

有两种方法可以将成员添加到一个环境里:

- 输入用户名，点击 + 把用户添加到成员列表中。如果该用户名不在列表中，则不会被加入到环境中。
- 在某些认证方式下，右侧有一个 + 下拉框按钮，这个下拉框会出现组织/团队，你可以将用户或团队加入到环境里。

你可以把每个成员(既个人、团队、或组织)的角色设置为所有者、成员、受限制的成员或只读用户中的一个。默认情况下，新添加的用户角色为成员。通过用户名旁边的下拉框，可以改变相应用户的角色. 对于环境所有者，你可以随时编辑成员列表以及成员角色。 只有环境的所有者能编辑环境的成员以及其角色。

注意： 只有所有者和管理员才能查看环境的基础设施服务。

点击 创建 会创建一个环境，所有在成员列表中的用户都立即可以看到这个环境。创建完环境并且添加主机后，Rancher会开始自动部署已启用的基础设施服务。

停用和删除环境

创建环境后，所有者可能想停用或删除该环境。

环境被停用后，这个环境不再对环境成员可见，但环境的所有者还可以看到并启用这个环境。在环境停用后你不能变更环境的成员，直到该环境被再次启用。环境被停用后所有资源不能再变更，如果你要变更你的基础设施服务，你需要在环境停用之前变更。

要删除一个环境，先要停用这个环境。环境删除后，这个环境所有的镜像仓库，负载均衡，API keys 都会从 Rancher 移除。所有通过 Rancher UI 创建，用 Docker Machine 启用的主机也会在云提供商中被用 Docker Machine 移除。如果你已经通过自定义的方式添加了一台主机，那么这台主机在云提供商中不会被移除。

成员编辑

只有环境的所有者可以编辑环境的成员。在环境管理页面，你可以点击编辑进入环境成员编辑页面， 在编辑页面，你可以通过下拉框添加环境成员。

如果要删除环境成员，可以点击成员列表旁边的X。注意，单个成员被删除时，如果被删除的成员所属的团队或组织是这个环境的成员，那么他们仍然可以访问这个环境，

所有者可以更改任何环境成员的角色，你只需要选择成员的相应角色。

成员角色

所有者

所有者有在环境中添加和删除用户的权限，也可以修改环境的状态。在环境的成员列表中，所有者还可以改变环境成员的角色。

因为无法编辑环境模版，所有者可以通过应用商店来修改环境的基础设施服务。环境模版只能在创建环境时使用。

成员

一个环境的成员可以在Rancher里面做任何不影响环境本身的操作。成员不能添加 / 移除其他成员，不能改变其他已存在成员的角色，也不能查看任何基础设施服务。

受限

环境的受限成员只能够做与应用和服务相关的操作。受限成员能对所有服务的容器做任何操作，即，启动、停止、删除、升级、克隆和编辑。从应用、 服务和容器操作的角度来说，受限成员是不受限制的。

对受限成员的限制体现在他们对主机的操作上。受限成员只能查看一个环境的主机，而不能添加，编辑，移除环境的主机。

> 注意： 受限成员不能添加、移除主机标签，只有成员和所有者才能改变主机的标签。

只读

只读成员只能查看环境的资源。 他们可以查看 主机、应用、服务和容器。但只读成员不能对它们作任何创建、编辑和移除操作。

> 注意： 只读成员可以查看容器的日志。

为了使非所有者可以设置环境的成员，你可以通过更新API配置project.set.member.roles来实现这一点。

什么是环境模版

环境模版可以让用户定义需要部署的基础设施服务组合。基础设施服务包括（但不限于）容器编排 (即Cattle，Kubernetes、Mesos、Swarm)、网络、Rancher 服务 (即 健康检查、DNS、Metadata、调度、服务发现、存储。

容器的编排方式很多，Rancher提供了一套默认的模版以及推荐使用的基础设施服务用于容器编排。其中的一些基础设施服务（Rancher调度器只能在Cattle环境下使用 ），其他的编排引擎也依赖他们，因为这些服务被用来启动其它基础设施服务。除了默认的模版，你也可以创建自己的模版。通过自己创建模版，你可以选者环境中任何你想要的基础设施服务组合。只有所有者或管理员可以查看和编辑环境的基础设施服务。

在和其它用户共享环境前， 我们推荐先设置好访问控制。用户被加入一个环境后, 他们就拥有了创建服务和管理资源的权限。

> 注意： 基础设施资源不可跨环境共享。镜像仓库、证书 和环境API密钥也不能跨环境。

添加环境模版

要添加一个新环境，你可以把鼠标移动到左上角的环境下拉框。下拉框中会出现所有可用的环境以及环境管理的链接。 点击环境管理

在环境页面后，你可以看到一个环境列表和一个环境模版列表。 点击添加模版

为模版选择一个 名称 和 描述， 选择分享自己模版的方式。 模版可以是私有（只有自己可见）和公有（管理员可见）。

基础设施服务包括，但不限于容器编排、存储和网络。默认的基础设施服务会自动启动。

编辑 & 删除环境模版

创建环境模版后，你可以在模版中编辑启用哪个基础设施服务。虽然环境模版是可以编辑的，但已经存在的基于模版创建的环境不会随模版自动更新。

你可以在任何时候删除一个环境模版，因为它们只有在启动环境的时候才会被用到（用来指定哪些基础设施服务会被启用）。环境与环境模版没有直接绑定关系，所以删除环境模版不会影响环境。

权限键:

- C = 创建
- R = 读取 (查看)
- U = 更新
- D = 删除

成员关联的权限

-	所有者	成员	受限	只读
环境成员	RUD	R	R	R
主机	CRUD	CRUD	R	R
容器	CRUD	CRUD	CRUD	R
存储	CRUD	CRUD	CRUD	R
密文	CRUD	CRUD	CRUD	R
证书	CRUD	CRUD	CRUD	R
镜像仓库	CRUD	CRUD	CRUD	R
Webhooks	CRD	CRD	CRD	R
用户应用	CRUD	CRUD	CRUD	R
基础设施应用	CRUD	RUD	R	R

-	所有者	成员	受限	只读
用户容器	start, stop, delete, restart, exec	start, stop, delete, restart, exec	start, stop, delete, restart, exec	-
基础设施容器	start, stop, delete, restart, exec	start, stop, delete, restart, exec	-	-

> 注意： 了解更多基础设施服务。用户应用，容器和服务没有被定义在基础设施服务中。

账户类型关联的权限

-	管理员	用户
私有模版	CRUD	CRUD
共有模版	CRUD	R
环境	CRUD	CRUD

入门指南

在Rancher中，主机是调度资源的基本单位（直观的理解就是所发生的操作最终都会落到某台主机上），它可以是虚拟的或者物理的Linux服务器。Rancher管理的主机需要满足以下的条件：

- 任何可以运行支持的Docker版本的 Linux 发行版本，例如：RancherOS，Ubuntu，RHEL/CentOS 7。不过针对RHEL/CentOS系列，有些需要注意的地方：-
  - Docker 并不推荐使用 RHEL/CentOS 默认的存储驱动（devicemapper），可以参考这篇文档来修改。
  - 如果启用 SELinux，需要安装额外的模块。
  - 内核版本要求是 3.10.0-514.2.2.el7.x86_64 及以上，建议使用 RHEL/CentOS 7.3 或者更高的发行版本。
- 1GB的内存。
- 推荐 w/AES-NI 架构的 CPU。
- 主机可以通过HTTP或HTTPS来访问Rancher Server服务，默认端口是8080。
- 主机与主机之间是可以相互访问的，从而确保容器之间的跨主机通信。

另外，Rancher也可以管理由Docker Machine驱动的主机，只要这些主机满足上面的条件即可。

在Rancher的操作界面上，选择 基础架构->主机，点击 添加主机按钮，然后再做些工作，就可以让 Rancher 管理到这台新主机了。

DOCKER版本适用对比

版本	Rancher适用？	K8S适用？	安装脚本
1.9.x和更低的版本	No	-	-
1.10.0 - 1.10.2	No	-	-
1.10.3 (和更高的版本)	No (Yes v1.6.5以及更低版本中)	No	curl https://releases.rancher.com/install-docker/1.10.sh | sh
1.11.x	No	-	curl https://releases.rancher.com/install-docker/1.11.sh | sh
1.12.0 - 1.12.2	No	-	-
1.12.3 (和更高的版本)	Yes	Yes	curl https://releases.rancher.com/install-docker/1.12.sh | sh
1.13.x	Yes	No	curl https://releases.rancher.com/install-docker/1.13.sh | sh
17.03.x-ce	Yes	No	curl https://releases.rancher.com/install-docker/17.03.sh | sh
17.03.x-ee	Yes	No	n/a
17.04.x-ce	No	-	curl https://releases.rancher.com/install-docker/17.04.sh | sh
17.05.x-ce	No	No	curl https://releases.rancher.com/install-docker/17.05.sh | sh
17.06.x-ce	Yes (v1.6.3以及更高版本)	No	curl https://releases.rancher.com/install-docker/17.06.sh | sh
17.06.x-ee	Yes (v1.6.3以及更高版本)	No	n/a

> 注意： 我们不会支持Docker的edge版本，但是我们会支持Docker的稳定版本。

安装特定版本的DOCKER

一般会使用 curl https://get.docker.com | sh 脚本来安装最新版的 Docker 。但是，最新版的 Dokcer 有可能不适用于正准备安装或已经在使用中的 Rancher 版本。因此，一种推荐的做法是：安装特定版本的 Docker。按照上方的对比表，选择 Rancher 适用的安装脚本执行即可。

> 注意： 如果从操作界面上添加云主机，可以通过 高级选项 里面的 Docker Install URL来选择需要安装的 Docker 版本。

主机是如何工作的？

Rancher Agent容器在主机上启动成功之后，这台主机就连接到了Rancher Server上。在添加主机->Custom（自定义）界面中的很长的URL就是主机注册口令。这个注册口令在主机第一次连接Rancher Server的时候会被用到。在注册的过程中，Rancher Server会生成一个Agent账号和一个API密钥。之后Agent与Server直接的全部通信都要使用这个密钥，它们之间的认证逻辑和其他的API密钥认证逻辑是一样的。

从设计的角度而言，Rancher Agent是运行在独立于Rancher Server的主机上，所以Rancher Agent本身是不可信的。采用接口访问密钥的机制，可以确保Rancher Agent只访问被授权的接口，而Rancher Server只响应可信的请求。但是目前是单向认证，只认证从Rancher Agent到Rancher Server的请求，并没有认证从Rancher Server到Rancher Agent的响应。因此，用户可以根据需要，使用TLS和证书来做校验。

每个环境的注册口令，都是由Rancher Server生成并保存到数据库，然后和API密钥一起下发给Rancher Agent使用。Rancher Agent和Rancher Server之间是采用AES对称加密的点对点通讯。

添加主机

第一次添加主机时，Rancher Server会要求配置主机注册地址。这个地址可以是域名或者IP地址（如果80端口不可访问，还需要加上可访问的端口号，默认 8080），能够访问Rancher接口即可。任何时候都可以改变 主机注册地址，相关操作可以查看 系统管理 下的 系统设置。设置好主机注册地址后，点击 保存.

Rancher支持添加云提供商（例如AWS，DigitalOcean，阿里云，vSphere等）所提供的主机或者本地（例如VirtualBox，VMWare等）设置好的主机。对于云提供商，Rancher可以通过docker-machine来添加的，所以本质上实现了 Docker Machine 驱动的厂商的云主机，都可以被添加。

接下来，选择:

- 添加自定义主机
- 添加 AWS EC2 主机
- 添加 Azure 主机
- 添加 DigitalOcean 主机
- 添加 Exoscale 主机
- 添加 Packet 主机
- 添加 Rackspace 主机
- 添加其他云提供商的主机

当主机被添加到Rancher时，这台主机会运行一个合适版本的rancher/agent容器。

主机标签

在Rancher中，可以通过添加标签的方式来管理某台主机，做法就是在 rancher/agnet 容器启动的时候，以环境变量的方法把标签加进去。在操作界面上可以看到，标签其实是一些键不可以重复的键值对。值得注意的是，如果有两个相同的键但是值不一样，那么最后添加的值将会被Rancher使用。

给主机增加标签后，你可以根据需求来调度服务 / 负载均衡。如果不希望某个服务运行在某台主机上或者要求某个服务必须运行在某台主机上，可以在添加服务时通过主机标签来进行配置。

在你需要使用外部DNS服务（类似 Bind9 这类）或者通过程序来控制DNS记录的时候，如果所需的IP不是主机IP的话，那就要在运行 rancher/agent 时增加标签 io.rancher.host.external_dns_ip=。切记，当要某个容器服务要使用外部DNS服务时，一定要增加这个标签。

通过UI添加云提供商的主机时，你可以在UI上添加主机标签，rancher/agent 会保证这些标签都会自动添加到主机上。

如果通过UI添加自定义主机，当增加标签时，UI上的运行注册脚本会增加对应的环境变量：CATTLE_HOST_LABELS。比如，增加一个标签：foo=bar，会出现下面的效果：

```
$  sudo docker run -e CATTLE_HOST_LABELS='foo=bar' -d --privileged \
    -v /var/run/docker.sock:/var/run/docker.sock rancher/agent:v0.8.2 \
    http://<rancher-server-ip>:8080/v1/projects/1a5/scripts/<registrationToken>

# 当再增加一个标签：hello=world
$  sudo docker run -e CATTLE_HOST_LABELS='foo=bar&hello=world' -d --privileged \
    -v /var/run/docker.sock:/var/run/docker.sock rancher/agent:v0.8.2 \
    http://<rancher-server-ip>:8080/v1/projects/1a5/scripts/<registrationToken>
```

> 注意： rancher/agent 的版本与 rancher/server 的版本是相关的，执行添加自定义主机的时候，需要注意注册脚本的 rancher/agent 是否正确。正常情况下，通过操作界面获取的脚本内的版本信息都是正确的，用户不需要做额外修改。

自动添加的标签

Rancher 会自动创建一些和Linux内核版本信息以及Docker引擎版本信息相关的标签。

键	值	描述
io.rancher.host.linux_kernel_version	Linux内核版本 (e.g. 3.19)	主机当前运行的内核版本
io.rancher.host.docker_version	Docker引擎版本 (e.g. 1.10)	主机运行的 Docker 版本
io.rancher.host.provider	云提供商信息	目前仅适用于某些云提供商
io.rancher.host.region	云提供商地域	目前仅适用于云提供商
io.rancher.host.zone	云提供商区域	目前仅适用于云提供商

主机调度地址

为了使Rancher可以在有多个IP的主机上暴露端口，需要给这些主机进行配置，从而使Rancher知道哪些IP可以使用。配置调度IP的方法取决于这台主机是已经存在的主机（运行了 rancher/agent 容器的服务器）还是还没有添加的主机（还没有运行 rancher/agent 容器的服务器）。

给已有主机添加调度地址

在环境中已存在的主机，可以通过增加io.rancher.scheduler.ips标签来设置调度IP。通过操作界面，点击这台主机的编辑按钮，然后增加 调度IP。如果是通过接口的方式，只需要给主机添加标签 io.rancher.scheduler.ips 和值（多个地址，可以通过逗号分隔）即可。

> 注意： 在没有添加调度地址前，如果某些容器已经暴露了端口，那么这些容器的端口暴露在0.0.0.0上。那就意味着，已有的容器已经占用了全部的IP地址，后来添加的调度地址也被占用了。

给新主机添加调度地址

对于新添加的自定义主机 需要像下面的例子，给注册脚本增加一个环境变量 CATTLE_SCHEDULER_IPS ：

```
$  sudo docker run -e CATTLE_SCHEDULER_IPS='1.2.3.4,<IP2>,..<IPN>' -d --privileged \
    -v /var/run/docker.sock:/var/run/docker.sock rancher/agent:v0.8.2 \
    http://<rancher-server-ip>:8080/v1/projects/1a5/scripts/<registrationToken>
```

在代理后的主机

如果当前的环境是在代理之后，要给Rancher添加主机，需要修改这台主机的Docker deamon指向代理。相关的细节可以浏览本文，在此不在累述。

访问云提供商的主机

当用Rancher添加云提供商的主机时，实质上是采用Docker Machine执行的工作。

克隆主机

在云提供商上启动主机需要使用访问密钥，Rancher提供了克隆的方法，来轻松地创建另一个主机，而无需再次输入所有认证配置。在操作界面上，从 基础架构 进入 主机 页面，点击某台主机的下拉菜单，选择 克隆，然后就会进入之前的认证配置都已经填写的 添加主机 页面。

修改主机

在操作界面上，从 基础架构 进入 主机 页面，点击需要修改的主机的下拉菜单，选择 编辑，就可以修改这台主机的名称，描述以及标签。

启停主机

停止一台主机后，操作界面上会显示 Inactive 状态。在这种状态下，不会再有容器服务被部署到这台主机。而处于 Active 状态下的主机，容器服务会被正常的部署、停止、重启或销毁。

如果需要停止一台主机，从 基础架构 进入 主机 页面，点击需要停止的这台主机的下拉菜单，选择 停用 即可。

如果需要把一台停止的主机重新激活，从 基础架构 进入 主机 页面，点击已经停止的这台主机的下拉菜单，选择 激活 即可。

> 注意： 在Rancher中如果主机宕机了（比如处于 reconnecting 或 inactive 的状态），需要给服务配置健康检查 以便于Rancher把这台宕掉的主机上的容器服务迁移到其他主机上执行。

在RANCHER内删除主机

在 Rancher 内删除主机的操作需要进行几个步骤：从 基础架构 进入 主机 页面，点击需要删除的主机的下拉菜单，选择 停用。当主机完成停止以后，将会显示 Inactive 状态。然后点击下拉菜单，选择 删除，Rancher 会执行对这台主机的删除操作。当显示 Removed 状态时，就表示这台主机已经被删除了。但是，仍然可以在操作界面上看到这台主机，只有当点击下拉菜单，选择 清理后，这台主机才会从操作界面上消失。

如果这台主机是由 Rancher 调用 docker-machine 基于云提供商的驱动创建，按照上述的删除操作执行后，被删除的主机也会在云提供商的管理界面中消失。但是，如果是采用 添加自定义主机 的方式所添加的云提供商主机，被删除的主机还会在云提供商的管理界面中被查看到。而且这台主机内的容器服务（例如 rancher/agent）还是保留着的。可以认为通过自定义添加的云提供商的主机被删除后，只是从Rancher的调度中解离出去，但是它原来的生命周期Rancher不会干涉。

> 注意： 对自定义主机，包括Rancher Agent在内的全部容器都会保留在该主机上。同时，Rancher网络驱动创建的docker0上的IP也将会保留。

在RANCHER外删除主机

在Rancher外删除主机，也就是意味着不是按照Rancher操作界面或者API来删除主机。最简单的例子就是，在Rancher集群内，有一台云提供商提供的主机。通过云提供商的管理界面删除了这台主机，这个删除行为Rancher是无法感知的。Rancher会一直尝试重新连接这台已经删除的主机，并显示 Reconnecting 的状态，之后主机会显示 Disconnected 状态。 因此，为了同步回删除的状态，还需要从 基础架构 进入 主机 页面，点击已经删除的主机的下拉菜单，选择 删除。 你也可以配置一个延迟用于自动删除失联主机。这个配置叫做host.remove.delay.seconds，你可以在系统 -> 设置 -> 高级选项中找到这个配置。

镜像库

你可以在Rancher配置镜像仓库的认证信息，使Rancher可以访问你的私有镜像仓库（DockerHub, Quay.io和其他私有镜像库）。 在每个环境中，你可以给每个私有仓库地址配置一个认证信息，从而使Rancher可以拉取私有镜像。如果你给同一个镜像仓库配置了多个认证信息，那么Rancher只会使用最近添加的一个认证信息。 Rancher支持在Cattle和Kubernetes环境中使用多种镜像仓库。

添加镜像库

在基础架构 -> 镜像库 页面, 点击 添加镜像库.

对于不同的镜像库，你都需要提供邮箱地址, 用户名, and 密码。 对于一个 自定义 镜像库, 你还需要提供镜像库地址。 点击 创建。

> 注意： 对于自定义的镜像库地址，不需要加上 http:// 或 https://， 因为我们假设地址只是一个IP或者主机名。

如果你对已经存在的镜像仓库地址添加了认证信息，Rancher会开始使用新的认证信息。

不安全的镜像库

为了访问不安全的镜像库，你需要配置主机上的Docker守护进程。DOMAIN 和 PORT 是私有镜像库的域名和端口。

```
# 编辑配置文件"/etc/default/docker"
$ sudo vi /etc/default/docker
# 将这行添加到文件最后，如果已经存在选项，确定你将它添加到当前选项的列表中。
$ DOCKER_OPTS="$DOCKER_OPTS --insecure-registry=${DOMAIN}:${PORT}"
# 重启docker服务
$ sudo service docker restart
```

自签名证书

为了在镜像库使用自签名证书，你需要配置主机上的Docker守护进程。 DOMAIN 和 PORT 是私有镜像库的域名和端口。

```
# 下载域名的证书
$ openssl s_client -showcerts -connect ${DOMAIN}:${PORT} </dev/null 2>/dev/null|openssl x509 -outform PEM >ca.crt
# 拷贝证书到合适的目录
$ sudo cp ca.crt /etc/docker/certs.d/${DOMAIN}/ca.crt
# 将证书添加到文件中
$ cat ca.crt | sudo tee -a /etc/ssl/certs/ca-certificates.crt
# 重启docker服务，让改动生效
$ sudo service docker restart
```

使用亚马逊的ECR镜像库

在Rancher使用亚马逊的 EC2 容器镜像库 需要额外的配置。ECR使用AWS的原生认证服务IAM去管理访问权限。AWS提供了API，让用户可以基于请求的IAM权限为Dokcer生成临时的认证信息。由于认证信息在12小时后会无效，每12小时需要生成一个新的认证信息。你可以使用AWS ECR 认证更新器 来发布一个自动更新认证信息的服务。

在Rancher中使用该镜像时，请使用AWS提供的全名地址，例如：

```
aws-account-number.dkr.ecr.us-west-2.amazonaws.com/my-repo:latest.
```

使用GOOGLE容器镜像仓库

如果你想使用Google容器镜像仓库，你需要使用一个服务账户JSON密钥文件。请给这个文件配置正确的权限，使其可以正常访问容器镜像所在的Google云存储服务。

请根据你镜像所在的区域来配置你的镜像仓库地址。使用_json_key作为用户名，并且使用服务账户JSON密钥文件中的全部内容作为密码。

使用镜像库

当你添加了镜像库以后，你就可以使用这个私有镜像库的镜像来部署服务和容器。镜像名字的格式和使用docker run命令是一样的。

```
[registry-name]/[namespace]/[imagename]:[version]
```

我们默认你尝试从DockerHub拉取镜像。

编辑镜像库

一个镜像库的所有操作选项可以通过镜像库列表右边的下拉菜单看到。 对于任意启用中的镜像库，你可以停用它，停用后Rancher将无法访问该镜像库的私有镜像，所以新的容器无法使用该镜像库的私有镜像（已经登陆过该镜像仓库的主机，仍然可以拉取私有镜像）。 对于任意停用中的镜像库，你有两个选项。 一个是启用，这会允许容器使用镜像库中的镜像。你的环境中的任何成员可以使用你的认证信息，而无需重新输入密码。如果你不想这样，你应该删除镜像库，这会从环境中删除镜像库的认证信息。你可以编辑任意镜像库，去修改认证信息，但不能改变镜像库的地址。密码不会出现在“编辑”页面中，所以你需要重新输入密码。

> 注意： 如果一个镜像库无效了（如停用、移除或被新的认证信息覆盖），任何使用私有镜像的服务 会继续运行。 由于镜像已经被拉取到主机上了，因此不会受镜像库的权限变化所影响。因此，扩容的服务或者新的容器都能运行。 当运行容器时，Rancher不会检查认证信息是否有效，我们假设你已经给了该主机访问该镜像的权限。

更改默认的镜像库

任何没有指定镜像库前缀的镜像，Rancher会默认从DockerHub中拉取。通过在API中更新配置，可以把默认镜像库从DockerHub改到另外一个。

1. 在 系统管理 -> 系统设置 -> 高级设置, 点击 我确认已经知道修改高级设置可能导致问题.
2. 找到registry.default 设置然后点击编辑按钮。
3. 添加镜像库的值然后点击 保存.

一旦 registry.default 设置被更新，任何没有镜像库前缀的镜像（如 ubuntu:14.0.4）会从新的默认镜像库拉取。

如果你使用的私有镜像库需要认证信息，为了使默认镜像库生效，你需要把该镜像库添加到Rancher中。

> 注意： 已存在在环境中的所有服务仍会使用原来的默认镜像库 (如 DockerHub)。为了使基础设施应用使用新的默认镜像库，需要删除它们然后再重新部署它们，这样才能使用新的默认镜像库。你可以通过 应用商店 -> 官方认证找到并部署基础设施服务。

限制镜像库的使用

默认的，Rancher可以拉取任何被添加的镜像库里的镜像。但管理员可能想要限制哪个镜像库可以使用。 你可以通过API更新配置，来限制哪些镜像库可以用于拉取镜像。

1. 在 系统管理 -> 系统设置 -> 高级设置, 点击我确认已经知道修改高级设置可能导致问题。
2. 找到 registry.whitelist 设置然后点击编辑按钮。
3. 把你想加到白名单中的镜像库加上，如果多于一个，那么镜像库间用逗号分隔。

一旦registry.whitelist 设置被更新，在拉取镜像前，会确认镜像所在的镜像库是否在白名单中，如果不是那么拉取会失败。

> 注意： 一旦你设置了镜像库白名单，你将无法使用DockerHub。 为了包含DockerHub， 你需要将index.docker.io加到设置中。

证书

添加证书

可以在基础架构 -> 证书页面把证书添加到你的环境里。这个页面已经列出了所有已添加到Rancher环境的证书。你可以点击添加证书添加一个新证书。

1. 填写证书名称和描述。
2. 填写证书私钥。你可以点击从文件读取导入文件或粘贴私钥到文本框。
3. 填写证书。你可以点击从文件读取 导入文件或粘贴证书到文本框。
4. (可选) 如果你有其它的证书链，你也可以通过从文件读取导入文件或粘贴证书链到文本框。

使用证书

添加到环境的证书可用作负载均衡的SSL终端或Kubernetes入口的TSL终端。

基础设施服务

当启动Rancher时，每个环境的创建都基于环境模版。在启动一个环境时，你可以在环境模版中选择需要启动的基础设施服务。这些基础设施服务包括编排引擎，外部DNS，网络，存储，框架服务 (例如： 内部DNS，Metadata服务，和健康检查服务)。

基础设施服务位于Rancher应用商店和社区应用商店中的infra-templates文件夹中。Rancher应用商店和社区应用商店是默认开启的，它们提供了一系列可以在环境模版中使用的基础服务。

当创建一个环境模版时，默认开启运行一个环境所需的一系列基础服务。

网络

Rancher实现了一个CNI框架，用户可以在Rancher中选择不同的网络驱动。为了支持CNI框架，每个Rancher环境中都需要部署网络服务，默认情况下，每个环境模版都会启用网络服务。除了网络服务这个基础设施服务之外，你还需要选择相关的CNI驱动。在默认的环境模版中，IPSec驱动是默认启用的，它是一种简单且有足够安全性的隧道网络模型。当你一个网络驱动在环境中运行时，它会自动创建一个默认网络，任何使用托管网络的服务其实就是在使用这个默认网络。这些服务运行着内部DNS服务器并且负责管理路由来暴露主机端口（通过iptable实现）。

与先前版本的区别

当使用1.2版本之前的IPsec网络时，容器使用托管网络将会被分配两个IP，分别是Docker网桥IP（172.17.0.0/16）和Rancher托管IP（10.42.0.0/16）。之后的版本中，则集成了CNI网络框架的标准，容器只会被分配Rancher托管IP（10.42.0.0/16）。

使用CNI的影响

Rancher托管IP不会显示在Docker元数据中，这意味着通过docker inspect无法查到IP。因为Rancher使用IPtables来管理端口映射,任何端口映射也无法通过docker ps显示出来。

容器间连通性

默认情况下，同一环境下的托管网络之间的容器是可达的。如果你想要控制这个行为，你可以部署网络策略服务。 如果你在跨主机容器通信中碰到问题，可以移步常见的故障排查与修复方法。

网络类型

在UI上创建服务时，切换到“网络”页签上可以选择网络类型，但是UI上默认不提供“Container”网络类型，如果要使用“Container”类型，则需要通过Rancher CLI/Rancher Compose/Docker CLI来创建。

托管网络

默认情况下，通过UI创建容器会使用托管网络，在容器中使用ip addr或者ifconfig可以看到eth0和lo设备，eth0的IP从属于Rancher托管子网中，默认的子网是10.42.0.0/16，当然你也可以修改这个子网。 注意：如果在基础设施服务中删除了网络驱动服务，那么容器的网络设置将会失效。

通过DOCKER CLI创建容器

任何通过Docker CLI创建的容器，只要添加–label io.rancher.container.network=true的标签，那么将会自动使用托管网络。不用这个标签，大部分情况下使用的是bridge网络。 如果容器只想使用托管网络，你需要使用–net=none和–label io.rancher.container.network=true。

NONE

当容器使用none网络类型，基本上等同于Docker中的—net=none。在容器中也不会看到任何网络设备除了lo设备。

HOST

当容器使用host网络类型，基本上等同于Docker中的—net=host。在容器中能够看到主机的网络设备。

BRIDGE

当容器使用bridge网络类型，基本上等同于Docker中的—net=bridge。默认情况下，容器中可以看到172.17.0.0/16的网段IP。

CONTAINER

当容器使用container网络类型，基本上等同于Docker中—net=container:。在容器中可以看到指定容器的网络配置。

RANCHER IPSEC使用例子

通过编写YAML文件，利用CNI框架来驱动，就可以构建Rancher的网络基础服务。下面是IPSEC网络驱动的YAML文件样例：

```
ipsec:
  network_driver:
    name: Rancher IPsec
    default_network:
      name: ipsec
      host_ports: true
      subnets:
      - network_address: $SUBNET
      dns:
      - 169.254.169.250
      dns_search:
      - rancher.internal
    cni_config:
      '10-rancher.conf':
        name: rancher-cni-network
        type: rancher-bridge
        bridge: $DOCKER_BRIDGE
        bridgeSubnet: $SUBNET
        logToFile: /var/log/rancher-cni.log
        isDebugLevel: ${RANCHER_DEBUG}
        isDefaultGateway: true
        hostNat: true
        hairpinMode: true
        mtu: ${MTU}
        linkMTUOverhead: 98
        ipam:
          type: rancher-cni-ipam
          logToFile: /var/log/rancher-cni.log
          isDebugLevel: ${RANCHER_DEBUG}
          routes:
          - dst: 169.254.169.250/32
```

NAME

网络驱动的名字

DEFAULT NETWORK

默认网络定义的是当前环境的网络配置

HOST PORTS

默认情况下，可以在主机上开放端口，当然你可以选择不开放

SUBNETS

你可以给Overlay网络定义一个子网

DNS && DNS SEARCH

这两个配置Rancher会自动放到容器的DNS配置中

CNI 配置

你可以将CNI的具体配置放在cni_config下面，具体的配置将会依赖你选择的CNI插件

BRIDGE

Rancher IPSEC实际上利用了CNI的bridge插件，所以你会看到这个设置，默认是docker0

BRIDGE SUBNET

这个配置可以理解为主机上容器的子网，对于Rancher IPSEC就是10.42.0.0/16

MTU

不同的云厂商在网络中配置了不同的MTU值。这个选项可以根据你的需要进行修改。这个选项也是Rancher需要的选项。需要明确的是MTU的配置需要在每一个网络组件上进行设置；在主机上，在Docker Deamon上，在IPsec或者VXLAN的基础服务里都要进行设置。同时同一个环境中的全部主机都需要有相同的设置。如果同一个环境中的全部主机，有着不同的MTU值，那么将会有随机的网络错误发生。

Rancher的IPsec Overlay网络有一个98字节的开销 容器网络接口的MTU = 网络的MTU - 98

例如，你有一个云厂商的MTU值为1200字节，那么如果你在容器中输入ip addr或者ifconfig时，你将会看到1102 (= 1200 - 98)字节的MTU值。

修改MTU

MTU的配置需要在每一个网络组件上进行设置；在主机上，在Docker Deamon上，在IPsec或者VXLAN的基础服务里都要进行设置（需要创建一个新的环境模版）。同时同一个环境中的全部主机都需要有相同的设置。你可以按照下面的步骤来修改MTU。

- 修改主机的MTU
  - 我们应该在主机的网络接口上修改这个值，请参考你使用的Linux发行版的文档，来了解如何修改MTU。
- 修改Docker网桥的MTU
  - 在大多数情况下，这将会是docker0。如下列，你可以通过在/etc/docker/daemon.json中设置MTU。更多详情，请参考Docker的官方文档自定义网桥docker0

```
{
  "mtu": 1450
}
```

- 创建一个新的环境模版来设置IPsec或者VXLAN基础设施服务所需的MTU值。
- 使用这个新建的环境模版来创建一个新的环境。

MTU只能在环境模版中进行配置。不建议在已有的环境中配置一个不同的MTU值，因为这个值仅会在新创建的容器中生效。

网络策略

Rancher允许用户在环境中配置网络策略。网络策略允许你在一个环境中定义特定的网络规则。所有的容器默认可以互相通信，但是有时你可能需要对的容器间通信做一些限制。

启动NETWORK POLICY MANAGER

当配置环境模版时，你可以启动 Network Policy Manage 组件。

如果你已经有一个启动的Rancher环境，你可以从Rancher应用商店中启动 Network Policy Manager

> 注意： Network Policy Manager现在只能在使用Cattle编排引擎的时候使用。环境模版基于编排引擎确定哪些组件可用，Rancher支持几乎所有的编排引擎。

通过UI管理网络策略规则

网络策略规则可以在每个环境设置页面中配置。点击左上角下拉列表中的环境管理，然后在需要配置的环境右侧点击编辑按钮

在界面上有四个选择，允许允许网络通信，禁止限制网络通信

- 链接服务之间：这个选项用来控制两个服务中链接的容器
- 服务内部: 这个选项用来控制服务内的容器
- 应用内部: 这个选项用来控制相同应用中不同服务
- 其他: 这个选项用来控制上面不包含的情况

一个通常的配置是在其他选择禁止，其他的都选择允许。

> 注意： 规则生效的顺序为从左至右

通过API管理网络策略规则

对于网络资源，defaultPolicyAction和policy 字段定义了容器间通信的工作规则。policy字段是内容为网络策略规则的有序数组。通过Rancher的API，可以配置环境的网络策略

获取网络的API地址

要配置网络策略，需要找到相应的网络资源。网络是环境的一部分，找到网络的URL为:

```
http://<RANCHER_SERVER_IP>/v2-beta/projects/<PROJECT_ID>/networks/<NETWORK_ID>`
```

怎么查找需要配置的网络的URL:

1. 点击API打开高级选项。在 环境API Keys，点击 Endpoint (v2-beta).
   > 注意：: 在UI上是环境，在API是project。
2. 在环境的links属性中查找networks，点击链接。
3. 查询你环境中启动的网络驱动的名字。例如：可能为 ipsec。点击该网络驱动的self
4. 在右边的Operations中，点击Edit，在defaultPolicyAction中，你可以修改默认的网络策略，同时在policy字段，你可以管理你的网络策略规则。

默认策略

默认所有容器间可以互相通信，在API中，你可以看到defaultPolicyAction被设置成allow。

可以通过修改defaultPolicyAction为deny来限制所有容器间的通信

网络策略规则

网络策略规则配置容器可以和一系列特定的容器通信

链接服务之间的容器

假设: 服务A链接服务B。

开启服务A和服务B之间的通信:

```
{
  "within": "linked",
  "action": "allow"
}
```

> 注意： 服务B的容器不会初始化一个链接到服务A。

关闭服务A和服务B之间的通信:

```
{
  "within": "linked",
  "action": "deny"
}
```

在环境内任一链接服务之间的网络策略规则适用于所有有链接的服务

同一服务中的容器

开通同一服务内容器的通信:

```
{
  "within": "service",
  "action": "allow"
}
```

关闭同一服务内容器的通信:

```
{
  "within": "service",
  "action": "deny"
}
```

同一应用中的容器

开通同一应用内容器的通信:

```
{
  "within": "stack",
  "action": "allow"
}
```

关闭同一应用内容器的通信:

```
{
  "within": "stack",
  "action": "deny"
}
```

基于标签的容器通信策略

通过标签开通容器间的通信:

```
{
  "between": {
    "groupBy": "<KEY_OF_LABEL>"
  },
  "action": "allow"
}
```

通过标签关闭容器间的通信:

```
{
  "between": {
    "groupBy": "<KEY_OF_LABEL>"
  },
  "action": "deny"
}
```

例子

容器隔离

环境内的容器都无法和彼此通信

- 设置defaultActionPolicy为deny.

应用隔离

同一个应用中的容器可以彼此通信，但是不能和其他应用中的容器通信

- 设置defaultActionPolicy为deny.
- policy中添加如下规则:

  ```
  {
   "within": "stack",
   "action": "allow"
  }
  ```

标签隔离

包含匹配的标签的容器之间可以通信，这个规则通过标签去划分可以相互通信的容器

假设在环境中，我们有如下一系列的应用

```
stack_one:
  service_one:
    label: com.rancher.department = qa
  service_two:
    label: com.rancher.department = engineering
  service_three:
    label: com.rancher.location = cupertino

stack_two:
  service_one:
    label: com.rancher.department = qa
  service_two:
    label: com.rancher.location = cupertino

stack_three:
  service_one:
    label: com.rancher.department = engineering
  service_two:
    label: com.rancher.location = phoenix
```

包含com.rancher.department标签的容器可以相互通信

- 设置defaultActionPolicy为deny.
- 在policy中添加如下规则:

```
{
  "between": {
    "groupBy": "com.rancher.department"
  },
  "action": "allow"
}
```

上面有两个不同的标签键值对(例如 com.rancher.department)。

- 容器包含com.rancher.department = engineering彼此间可以通信，但是和其他的容器不能通信。在上面例子中，任何 stack_one.service_two 中的容器和 stack_three.service_one中的容器可以彼此通信，但是其他的不能。
- 容器包含 com.rancher.department = qa彼此间可以通信，但是和其他的不能。在上面的例子中，任何stack_one.service_two 中的容器可以和任何stack_two.service_two中的容器通信，但是其他的不能。
- 容器不包含key com.rancher.department不能和其他容器通信。

负载均衡器

Rancher支持在Rancher内使用不同负载均衡器驱动。你可以通过向目标服务添加规则使负载均衡器将网络和应用程序流量分配到容器中。Rancher将自动将目标服务的容器自动注册为Rancher的负载平衡目标。

默认情况下，Rancher提供了一个基于HAProxy的托管负载均衡器，你可以手动扩容到多个主机上。我们计划添加额外的负载均衡器驱动，所有负载均衡器的选项将是相同的，不管负载均衡器种类。

对于Cattle引擎的环境，可以参考UI和Rancher Compose了解更多信息，并且在UI和Rancher Compose中有相关的例子。

对于Kubernetes的环境，详细了解如何启动云厂商提供的外部负载均衡器服务，或者使用Rancher负载均衡器实现Kubernetes环境中的Ingress支持。

DNS服务

Rancher提供了一个分布式DNS服务的基础设施服务。这个服务是通过Rancher自己的轻量级DNS服务器和高可用性的控制平面实现的。每个健康容器在链接到另一个服务或添加一个服务别名时将自动被添加到DNS服务。当使用服务名称查询时，DNS服务返回实施该服务的健康容器的IP地址的随机列表。

- 默认情况下，同一应用中的所有服务都将添加到DNS服务中，而不需要显示的设置服务链接，链接可以在服务中的服务链接下进行设置。
- 你可以通过服务名称解析相同应用中的容器。
- 如果你需要一个与服务名称不同的自定义的DNS名称，你需要设置一个链接以设置自定义的DNS名称。
- 如果使用服务别名，则仍然需要链接。
- 要使不同栈中的服务可解析，可以使用<服务名>.<栈名>，而不需要显示的设置服务链接，链接可以在服务中的服务链接下进行设置。

因为Rancher的Overlay网络为每个容器提供了不同的IP地址，所以不需要处理端口映射，并且不需要处理像重复的服务在不同端口上侦听的情况。因此，简单的DNS服务足以处理服务发现。

了解更多关于Cattle环境的内部DNS服务。

Metadata 服务

Rancher通过基础设施中的Metadata服务为服务和容器提供数据。这些数据用来管理运行中的docker实例。这些数据可以通过调用基于HTTP的API来访问。这些数据包括创建容器，服务时的静态数据，也包括运行时数据，例如：在同一个服务里的其他容器的相关信息。

通过Rancher的Metadata服务，你可以进到任何使用Rancher托管网络的容器的命令行中，并查看运行在Rancher中的容器的信息。通过Metadata服务你可以获取容器，服务，容器所在的应用，容器所在的主机。Metadata是JSON格式的。

有多种方式可以将容器运行在Rancher托管网络中。Rancher网络的原理详见网络相关文档。

如何获取METADATA

通过Rancher UI，你可以通过容器的下拉菜单的执行命令行进入运行命令界面。在鼠标悬停在容器上时，会显示容器名和右侧的下拉菜单。

你可以通过curl命令获取metadata信息。

```
# If curl is not installed, install it
$ apt-get install curl
# Basic curl command to obtain a plaintext response
$ curl http://rancher-metadata/<version>/<path>
```

curl请求的路径取决于你想要获取的metadata信息和格式。

Metadata	路径	描述
容器	self/container	提供运行命令的容器的metadata信息
容器所在服务	self/service	提供运行命令的容器对应服务的metadata信息
容器所在应用	self/stack	提供运行命令的容器对应应用的metadata信息
容器所在主机	self/host	提供运行命令的容器对应主机的metadata信息
其他容器	containers	提供所有容器的metadata信息。在纯文本格式时，提供了带上索引序号的所有容器。在JSON格式，提供了所有容器的所有metadata信息。使用序号或者名字。都可以获取指定容器的metadata信息。
其他服务	services	提供了所有服务的metadata信息。在纯文本格式时，提供了带上索引序号的所有服务。在JSON格式，提供了所有服务的所有metadata信息。在路径中使用序号或者名字，都可以获取指定服务的metadata信息。当访问容器详细信息时，在V1 (2015-07-25)只返回容器名称，但是在V2 (2015-12-19)，容器实例也会返回。
其他应用	stacks/	提供了所有应用的metadata信息。在纯文本格式，提供了带上索引序号的所有应用。在JSON格式，提供了所有应用的所有metadata信息。使用序号或者名字。都可以获取指定容器的metadata信息。在路径中使用序号或者名字，都可以获取指定应用的metadata信息。当访问container详细信息时，在V1 (2015-07-25)只返回容器名称，但是在V2 (2015-12-19)，容器实例也会返回。

METADATA的版本

在curl命令中，我们强烈建议使用确定的版本号，但是你也可以选者latest。

> 注意： 因为latest版本会包含最新的代码变动，各个版本的返回的数据可能不同，需要确认是否和你之前的代码能够兼容。

metadata的版本是基于日期的。

Version Reference	Version
V2	2015-12-19
V1	2015-07-25

版本变化

V1 VS. V2

当通过http请求访问路径 /services//containers或者/stacks//services//containers时, V1 返回容器名称，V2返回容器实例。更多详细信息在V2 metadata服务中提供。

范例 在Rancher中，名为foostack的应用包含一个有三个容器的服务 barservice。

```
# 在V1只返回service的container names
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-07-25/services/barservice/containers'
["foostack_barservice_1", "foostack_barservice_2", "foostack_barservice_1"]

# 在V2中返回service的container objects
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-12-19/services/barservice/containers'
[{"create_index":1, "health_state":null,"host_uuid":...
...
# 获取service中所有容器的metadata信息
...
...}]

# 在V2，可以获取指定的container object
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-12-19/services/barservice/containers/foostack_barserv
ice_1'
[{"create_index":1, "health_state":null,"host_uuid":...
...
# 获取service中所有容器的metadata信息
...
...}]

# 通过路径 /stacks/<service-name>，可以访问services和containers

# 使用V1只返回service的container names
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-07-25/stacks/foostack/services/barservice/containers'
["foostack_barservice_1", "foostack_barservice_2", "foostack_barservice_1"]

# 使用V2返回service的container objects
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-12-19/stacks/foostack/services/barservice/containers'
[{"create_index":1, "health_state":null,"host_uuid":...
...
# 获取service中所有容器的metadata信息
...
...}]
```

纯文本 VS JSON

Metadata返回有纯文本和JSON两种格式，根据需要选择相应格式.

纯文本

通过curl命令，会获得请求路径的纯文本格式返回。你可以通过从第一层路径开始，层层推进，找到你需要的信息。

```
$ curl 'http://rancher-metadata/2015-12-19/self/container'
create_index
dns/
dns_search/
external_id
health_check_hosts/
health_state
host_uuid
hostname
ips/
labels/
memory_reservation
milli_cpu_reservation
name
network_from_container_uuid
network_uuid
ports/
primary_ip
primary_mac_address
service_index
service_name
stack_name
stack_uuid
start_count
state
system
uuid
$ curl 'http://rancher-metadata/2015-12-19/self/container/name'
# Note: Curl 不会返回新的行，只有一个数据时返回会输出在同一行
Default_Example_1$root@<container_id>
$ curl 'http://rancher-metadata/2015-12-19/self/container/label/io.rancher.stack.name'
Default$root@<container_id>
# Arrays可以通过序号或者名字访问
$ curl 'http://rancher-metadata/2015-12-19/services'
0=Example
# 使用序号或者名字
$ curl 'http://rancher-metadata/2015-12-19/services/0'
$ curl 'http://rancher-metadata/2015-12-19/services/Example'
```

JSON

JSON格式的返回可以通过在curl命令中增加header Accept: application/json

```
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-12-19/self/container'
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-12-19/self/stack'
# 获取stack中另一个service的信息
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-12-19/services/<service-name>'
```

METADATA属性

容器

属性	描述
create_index	容器启动的序号 例如 2 代表的是服务中启动的第二个容器。注意: Create_index不会被重用。 如果你的服务包含两个容器，删除了第二个容器，下一个启动的容器的create_index会是3，即使服务中只包含2个容器
dns	容器的DNS服务器。
dns_search	容器的搜索域。
external_id	在主机上的Docker容器ID。
health_check_hosts	列出运行健康检查的主机的的UUIDs。
health_state	开启健康检查的容器的健康状态 健康检查。
host_uuid	Rancher Server分配给主机的唯一标识。
hostname	容器的hostname。
ips	支持多NIC时的IP列表
labels	容器标签列表。格式为key:value。
memory_reservation	容器可以使用内存的软限制。
milli_cpu_reservation	容器可以使用CPU的软限制，值为正整数，1代表1/1000CPU。所以，1000 代表1个CPU，500代表半个CPU。
name	容器的名字。
network_from_container_uuid	容器网络来源的容器UUID。
network_uuid	Rancher分配的网络唯一标识
ports	列出容器使用的端口。格式为： hostIP:publicIP:privateIP[/protocol].
primary_ip	容器IP
primary_mac_address	容器的MAC地址
service_index	服务中容器名称的最后一个数字
service_name	服务名称(如果存在)
stack_name	服务所在的应用的名称(如果存在)
stack_uuid	Rancher分配的应用的唯一标识
start_count	容器启动的次数
state	容器状态
system	容器是否是Rancher基础设施服务
uuid	Rancher分配容器唯一标识

服务

属性	描述
containers	列出服务中的容器名称
create_index	服务中最后启动的容器的序号 例如 2代表的是服务中启动的第二个容器。注意: Create_index不会被重用。 如果你的服务包含2个容器，删除了第二个容器，下一个启动的容器的create_index会是3，即使服务中只包含2个容器
expose	对主机暴露，但是不对外暴露的端口
external_ips	内部服务的IP列表
fqdn	服务的全称域名
health_check	服务的健康检查配置
hostname	内部服务的CNAME
kind	Rancher的服务类型
labels	服务标签列表，格式为 key:value.
lb_config	负载均衡的配置
links	列出服务的链接，格式为stack_name/service_name:service_alias. links(例如 stack_name/service_name 获取所有链接)根据返回的service_alias,获取进一步的详细信息。
metadata	用户添加的metadata
name	服务名称
ports	服务使用的端口。格式hostIP:publicIP:privateIP[/protocol].
primary_service_name	主服务名，如果有从服务
scale	服务中容器的规模数量
sidekicks	从容器服务的名称列表
stack_name	服务所在的应用的名称
stack_uuid	Rancher分配的应用的唯一标识
system	是否是基础设施服务
uuid	Rancher分配的服务的唯一标识

应用

属性	描述
environment_name	应用所在的环境的名字
environment_uuid	Rancher分配的环境的唯一标识
name	应用名称
services	应用中的服务列表
system	应用是否为基础设施服务
uuid	Rancher分配的应用的唯一标识

主机

属性	描述
agent_ip	Rancher Agent的IP，例如 CATTLE_AGENT_IP环境变量值。
hostname	主机的名称
labels	主机标签列表。格式为key:value.
local_storage_mb	主机的存储大小，单位为MB
memory	主机的内存大小，单位为MB
milli_cpu	主机的CPU。数值为整数，1代表1/1000的cpu。所以，1000代表1 CPU.
name	主机的名称
uuid	Rancher分配的主机的唯一标识

为服务添加用户自定义METADATA

Rancher支持为服务添加用户metadata。现在只支持通过Rancher Compose添加，metadata是rancher-compose.yml的一部分。metadatakey对应的部分，yaml会被转化成在metadata-service中使用的JSON格式

EXAMPLE RANCHER-COMPOSE.YML

```
service:
  # Scale of service
  scale: 3
  # User added metadata
  metadata:
    example:
      name: hello
      value: world
    example2:
      foo: bar
```

服务启动后，可以使用metadata服务在.../self/service/metadata或者.../services//metadata看到metadata数据

按照JSON格式查询

```
$ curl --header 'Accept: application/json' 'http://rancher-metadata/latest/self/service/metadata'
{"example":{"name":"hello","value":"world"},"example2":{"foo":"bar"}}
```

按照纯文本格式查询

```
$ curl 'http://rancher-metadata/latest/self/service/metadata'
example/
$ curl 'http://rancher-metadata/latest/self/service/metadata/example'
name
value
$ curl 'http://rancher-metadata/latest/self/service/metadata/example/name'
# # Note: Curl 不会返回新的行，只有一个数据时返回会输出在同一行
hello$root@<container_id>
```

存储服务

Rancher提供了不同的存储服务，从而使用户可以将存储卷映射到容器中。

配置存储服务

当我们创建环境模板时，用户可以从应用商店选择需要在环境中的使用存储服务。

或者，如果用户已经创建了一个环境，你可以从 应用商店中选择并启动一个存储服务。

> 注意： 某些存储服务可能无法和一些容器编排调度引擎（例如，kubernetes）兼容。环境模板可以根据当前的编排调度框架限定可以使用的存储服务，而应用商店中则会显示全部的存储服务。

查看存储驱动

在存储服务启动后，在基础架构 -> 存储的界面中可以看到一个存储驱动已经被创建出来。在这个界面中，用户可以查看当前环境中所有可用的存储驱动。存储驱动的名称和刚刚启动的存储服务所在的应用的名称保持一致。

对于每一种存储驱动，主机上运行的存储服务都会被显示出来。正常情况下，环境里的所有主机都会出现在该页面。同时，存储驱动提供的卷列表以及卷的状态也会被显示出来。你可以看到每个卷的名称（比如，主机上的卷名称），以及每个卷的挂载点。对于每个挂载点，其容器信息以及该挂载点在容器中映射的路径都会被显示出来。

卷的作用域

Rancher的存储服务中，卷的作用域可以在不同的级别生效。目前，只有Rancher Compose支持创建不同的存储作用域。UI上仅仅支持环境级别的卷创建操作。

应用级别

应用级别的存储卷，应用中的服务如果引用了相同的卷都将共享同一个存储卷。不在该应用内的服务则无法使用此存储卷。

Rancher中，应用级别的存储卷的命名规则为使用应用名称为前缀来表明该卷为此应用独有，且会以随机生成的一组数字结尾以确保没有重名。在引用该存储卷时，你依然可以使用原来的卷名称。比如，如果你在应用 stackA中创建了一个名为foo 的卷， 你的主机上显示的卷名称将为 `stackA_foo_<randomNumber>` ，但在你的服务中，你依然可以使用foo。

环境级别

环境级别的存储卷，该环境中的所有服务如果 引用了相同的卷将共享同一个存储卷。不同应用中的不同服务也可以共享同一个存储卷。目前，通过UI只可以创建环境级别的卷。

在UI中使用存储驱动

在你的存储服务启动后且状态为active，使用共享存储卷的服务就可以被创建了。在创建服务时，在卷选项卡中，输入卷以及卷驱动

卷 语法和Docker语法相同， `<volume_name>:</path/in/container>` 。Docker卷默认挂载为读写模式，但是你可以通过在卷的末尾添加:ro将其挂载为只读模式。

卷驱动和存储驱动的名字一致，为存储驱动的应用名。

如果 `<volume_name>` 在存储驱动中已经存在，在存储卷作用域范围内，将使用相同的存储卷。

创建新卷

一个卷可以被分为两部分创建：

1. 创建服务时，如果 卷选项卡中的卷在存储驱动中还不存在，环境级别的存储卷将被创建。如果卷已经存储，将不会再创建新卷。

   > 注意：该设定并不适用于Rancher EBS，使用Rancher EBS时，必须首先定义一个卷。

2. 在基础架构 -> 存储界面中，选择添加卷。输入卷名称以及驱动信息如果你需要的话。该卷在被一个服务使用之前将一直保持 inactive 状态。

在RANCHER COMPOSE中使用存储驱动

在基础设施应用中的存储服务启动后，你可以开始创建卷了。在下面的例子中，我们将使用Rancher NFS 存储服务。 在Docker Compose文件中volumes下可以定义卷。在同一个Docker Compose中每个卷可以和多个服务关联。此功能只在Compose v2格式下生效。

```
version: '2'
services:
  foo:
    image: busybox
    stdin_open: true
    volumes:
    - bar:/var/lib/storage
volumes:
  bar:
    driver: rancher-nfs
```

应用级别

默认情况下，所有的卷将为应用级别。在同一个Compose文件中所有引用同一个卷的服务或应用将共享同一个卷。

如果再同一个Compose文件中创建了一个新应用，一个新卷也会被创建。当应用被删除时，卷也会被删除。 在上面的例子上，卷bar即为应用级别。

环境级别

如果需要多个应用共享卷，你需要使用一个环境级别的卷。在这个例子里，必须先创建好卷，之后才可以启动使用这个卷对服务或应用。为了使用环境级别的卷，你需要添加external选项到这个卷里。

```
version: '2'
services:
  foo:
    image: busybox
    stdin_open: true
    volumes:
    - bar:/var/lib/storage
volumes:
  bar:
    driver: rancher-nfs
    external: true
```

如果在当前环境中找不到一个叫bar的环境级别的卷，那么将会有报错提示。环境级别的卷只能通过UI进行删除。

Rancher NFS

Rancher支持将NFS卷作为容器的一个存储选项

使用NFS之前的准备工作 在部署Rancher NFS驱动之前，你需要先准备一个NFS服务器。例如，你可以使用如下命令在Ubuntu 16.04上安装NFS服务器。

```
sudo apt-get update
sudo apt-get install nfs-kernel-server
```

在这个服务器上，你需要设置一个基础目录。首选，你需要创建一个共享目录。

```
sudo mkdir /nfs
sudo chown nobody:nogroup /nfs
```

修改exports文件(/etc/exports).

```
/nfs    *(rw,sync,no_subtree_check,no_root_squash)
```

在全部修改完成后，你需要重新启动NFS内核服务器。

```
sudo systemctl restart nfs-kernel-server
```

在AMAZON EFS上使用RANCHER NFS驱动

Rancher的NFS驱动可以连接Amazon的EFS。当我们在Amazon EFS上使用Rancher NFS驱动时，环境内全部的主机都需要是EC2主机，并且这些主机要部署在与EFS所在区域相同的同一个可用区内。

配置RANCHER NFS

当设置一个环境模版的时候，你可以选择启用Rancher NFS应用，这样以后用这个模版创建的环境都会包括Rancher的NFS服务。

或者，如果你已经设置好了一个环境，你可以在应用商店中找到并部署Rancher NFS服务。

> 注意： 某些存储服务可能与容器编排引擎不兼容(例如 Kubernetes)。环境模版会根据你选择的编排引擎显示其兼容的存储服务。但是在应用商店中可以看到全部的应用，不会按照编排引擎进行过滤。

为了部署Rancher NFS，你需要指定如下配置：

- NFS Server: NFS服务器的IP地址或者主机名称
- Export Base Directory: NFS服务器输出的共享目录
- NFS Version: 你所用的NFS版本，当前使用的是版本4
- Mount Options: 用逗号分隔的默认挂载选项， 例如: ‘proto=udp’. 不要配置nfsvers选项，这个选项会被忽略。
- On Remove: 当移除Rancher NFS卷的时候，底层数据是否应该被保留或者清理。选项有purge和retain，默认值为purge。从Rancher 1.6.6开始支持。

RANCHER NFS驱动选项

当通过Rancher NFS驱动创建卷时，你可以通过一些选项来自定义自己的卷。这些选项是一些键值对，可以通过UI的驱动选项添加，也可以通过compose文件的driver_opts属性来添加。

驱动选项

- Host - (host): NFS主机
- Export - (export): 当一个卷配置了host和export，将不会创建子文件夹，export的根目录将会被挂载。
- Export Base - (exportBase): 默认情况下，卷可以配置host和export base，这样会在NFS服务器上创建一个名字唯一的子文件夹。
- Mount Options - (mntOptions): 用逗号分隔的默认挂载选项。
- On Remove - (onRemove): 当移除Rancher NFS卷的时候，底层数据是否应该被保留或者清理。选项有purge和retain，默认值为purge。从Rancher 1.6.6开始支持。

在UI中使用RANCHER NFS

创建卷

当Rancher NFS在Rancher中部署成功后，你还需要在基础架构 -> 存储里创建NFS卷，之后才可以在服务中使用NFS卷。

1. 点击添加卷。
2. 输入在服务中使用的卷的名称。
3. 可选: 添加额外的驱动选项。

在服务中使用卷

一旦卷在UI中被创建成功，你可以在服务中使用这个共享存储了。当创建一个服务时，在卷页签，可以输入卷和卷驱动。

volume的语法格式与Docker相同， `<volume_name>:</path/in/container>` 。Docker的卷默认是以读写模式进行挂载的，但是你可以通过在卷结尾处添加:ro使其以只读的模式进行挂载。

卷驱动和存储驱动的名字一致，为存储驱动的应用名。默认情况下，Rancher NFS 存储驱动名称为rancher-nfs。

在COMPOSE文件中使用RANCHER NFS

在基础设施应用中的Rancher NFS启动后，你可以开始在Compose文件中创建卷了。

在Docker Compose文件中volumes下可以定义卷。在同一个Docker Compose中每个卷可以和多个服务关联。

> 注意： 此功能只在Compose v2格式下生效。

NFS卷示例

在这里例子中，我们将创建一个NFS卷同时创建使用这个卷的服务。所有该应用中的服务将共享同一个卷。

```
version: '2'
services:
  foo:
    image: alpine
    stdin_open: true
    volumes:
    - bar:/data
volumes:
  bar:
    driver: rancher-nfs
```

使用HOST，EXPORTBASE和EXPORT的示例

下面的例子展示了如何在某个服务中，覆盖host和exportBase。

```
version: '2'
services:
  foo:
    image: alpine
    stdin_open: true
    volumes:
    - bar:/data
volumes:
  bar:
    driver: rancher-nfs
    driver_opts:
      host: 192.168.0.1
      exportBase: /thisisanothershare
```

你也可以给每个卷使用不同的exportBase，请看下面的例子。

```
version: '2'
services:
  foo:
    image: alpine
    stdin_open: true
    volumes:
    - bar:/bardata
    - baz:/bazdata
volumes:
  bar:
    driver: rancher-nfs
    driver_opts:
      host: 192.168.0.1
      exportBase: /thisisanothershare
  baz:
    driver: rancher-nfs
    driver_opts:
      host: 192.168.0.1
      exportBase: /evenanothershare
```

RANCHER NFS使用AWS EFS

在AWS上创建EFS文件系统之后，你可以部署Rancher NFS驱动来使用这个EFS文件系统。因为亚马逊EFS只在内部可达，所以只有与EFS在同一个可用区内的EC2主机可以访问EFS。因此，在创建存储驱动之前，你需要先添加EC2主机到Rancher环境中。

你可以使用下面的选项来部署Rancher NFS:

- NFS Server: xxxxxxx.efs.us-west-2.amazonaws.com
- Export Base Directory: /
- NFS Version: nfsvers=4

在移除卷时保留数据

驱动选项onRemove的默认值为purge。这意味着，当从Rancher中删除这个卷的时候，底层的数据也会被删除。如果你想要保留底层数据，你可以将这个选项设置为retain。你也可以给每个卷设置不同的onRemove值。如果nfs-driver选项onRemove被设置为retain，但是你想要在某个卷在Rancher中被删除时清理掉这个卷的底层数据，你可以通过docker-compose.yml在这个卷的driver_opts下面设置onRemove: purge。示例入下。

```
services:
  foo:
    image: alpine
    stdin_open: true
    volumes:
    - bar:/data
volumes:
  bar:
    driver: rancher-nfs
    driver_opts:
      onRemove: purge
```

如果nfs-driver选项onRemove被设置为purge，你可以在卷的driver_opts里设置onRemove: retain来保留数据，这样当这个卷在Rancher中被移除时，数据将会被保留下来。

```
services:
  foo:
    image: alpine
    stdin_open: true
    volumes:
    - bar:/data
volumes:
  bar:
    driver: rancher-nfs
    driver_opts:
      onRemove: retain
```

> 注意： 创建一个外部卷的时候，如果卷的名称和之前被删除的卷的名称相同，并且这个被删除的卷的数据被保留着，这时使用这个卷的容器可以访问被先前保留的数据。

Rancher EBS

Rancher提供对AWS EBS卷的支持，用户可以选择为容器选择AWS EBS存储。

使用EBS的限制

一个AWS EBS卷只可以挂载到一个AWS EC2实例中。因此，所有使用同一个AWS EBS卷的所有容器将会被调度到同一台主机上。

配置RANCHER EBS

当配置一个环境模板时，你可以选择启用Rancher EBS。这样从该环境模板创建的环境都会自动启动该存储驱动。

又或者，你已经创建了一个环境，你可以选择从应用商店中直接启动Rancher EBS。

> 注意： 某些存储服务可能无法和一些容器或编排调度系统（例如，kubernetes）所兼容。环境模板可以根据当前的编排调度系统限定可以使用的存储服务，而应用商店中则会显示全部的存储服务。

要启动Rancher EBS，你需要一个AWS Access Key以及Secret Key 以确保你有权限创建AWS EBS卷。同时，不同的驱动选项可能还需要其他额外的权限。

RANCHER EBS 驱动选项

当创建AWS EBS卷时，有一些其他的选项可以用于自定义卷。这些选项是一些键值对，可以通过UI的驱动选项添加，也可以通过compose文件的driver_opts属性来添加。

必填驱动选项

- 大小 - (size): EBS卷大小

可选驱动选项

- 卷类型 - (volumeType): 卷类型
- IOPS - (iops): IOPS 选项
- 指定的可用区 (ec2_az): 在指定的可用区中创建容器以及EBS卷。(比如. us-west-1a)

对于以下选项，你必须指定和ID绑定的可用区(ec2_az)

- Encrypted (encrypted): 卷是否需要被加密。注：如果需要启动此选项，则需要提供AWS KMS ID。
- AWS KMS ID (kmsKeyId): 用于创建加密卷的AWS Key Management Service customer master key (CMK) 的完整资源名称-ARN（Amazon Resource Name）
- Snapshot ID (snapshotID): 用于创建卷的快照。
- Volume ID (volumeID): 已创建的卷ID。

在界面中使用RANCHER EBS

创建卷

当 Rancher EBS在Rancher中启动后，需要先从 基础架构 -> 存储中创建EBS卷，然后才可以在服务中使用EBS卷。

1. 点击添加卷
2. 填写卷名称
3. 必填：填写 size选项
4. 可选：添加其他额外的驱动选项。注：如果要使用加密，快照ID或者卷ID，你需要指定对应的可用区。

在服务中使用卷

一旦卷在UI中被创建，服务就可以使用该共享存储。创建一个服务时，在 卷选项卡中，填写卷以及卷驱动信息。 卷 语法和Docker语法相同， `<volume_name>:</path/in/container>` 。Docker卷默认挂载为读写模式，但是你可以通过在卷的末尾添加:ro将其挂载为只读模式。 卷驱动和存储驱动的名字一致，为存储驱动的应用名。默认情况下，Rancher EBS 存储驱动名称为rancher-ebs。

在COMPOSE文件中使用RANCHER EBS

在基础设施应用中的Rancher EBS启动后，你可以开始在Compose文件中创建卷了。 在Docker Compose文件中volumes下可以定义卷。在同一个Docker Compose中每个卷可以和多个服务关联。

> 注意： 此功能只在Compose v2格式下生效。

举例：应用级别存储卷，指定SIZE、卷类型以及IOPS

在这里例子中，我们将创建一个使用应用级别的存储卷的服务。所有该应用中的服务将共享同一个卷。

```
version: '2'
services:
  foo1:
    image: busybox
    stdin_open: true
    volumes:
    - bar:/var/lib/storage
  foo2:
    image: busybox
    volumes:
    - bar:/var/lib/storage

volumes:
  bar:
    driver: rancher-ebs
    driver_opts:
      size: 10
      volumeType: gp2
      iops: 1000
```

举例：指定可用区的应用级别的存储卷

在这里例子中，我们将创建一个使用应用级别的存储卷的服务。所有该应用中的服务将共享同一个卷。 我们将指定卷的可用区，使用该AWS EBS卷的所有容器将会被调度到同一台主机上。

```
version: '2'
services:
  foo:
    image: busybox
    stdin_open: true
    volumes:
    - bar:/var/lib/storage

volumes:
  bar:
    driver: rancher-ebs
    driver_opts:
      size: 10
      ec2_az: us-west-2a
```

举例：应用级别加密卷

在这里例子中，我们将创建一个使用应用级别的存储卷的服务。所有该应用中的服务将共享同一个卷。 为了加密该卷，你需要在驱动选项中启用加密并指定加密密钥的ID以及该密钥所在的可用区。 使用该AWS EBS卷的所有容器将会被调度到同一台主机上。

```
version: '2'
services:
  foo:
    image: busybox
    stdin_open: true
    volumes:
    - bar:/var/lib/storage

volumes:
  bar:
    driver: rancher-ebs
    driver_opts:
      size: 10
      encrypted: true
      kmsKeyId: <KMS_KEY_ID>
      # Specifying the availability zone is required when using encryption and kmsKeyId
      ec2_az: <AVAILABILITY_ZONE_WHERE_THE_KMS_KEY_IS>
```

举例：基于快照的应用级别的存储卷

在这里例子中，我们将创建一个使用应用级别的存储卷的服务。所有该应用中的服务将共享同一个卷。 该卷将基于一个已有的AWS快照被创建出来。你需要指定快照ID以及该快照所在的可用区。 使用该AWS EBS卷的所有容器将会被调度到同一台主机上。

```
version: '2'
services:
  foo:
    image: busybox
    stdin_open: true
    volumes:
    - bar:/var/lib/storage

volumes:
  bar:
    driver: rancher-ebs
    driver_opts:
      size: 10
      snapshotID: <SNAPSHOT_ID>
      # Specifying the availability zone is required when using snapshotID
      ec2_az: <AVAILABILITY_ZONE_WHERE_THE_SNAPSHOT_IS>
```

举例：基于已有EBS卷的的应用级别的存储卷

在这里例子中，我们将创建一个使用应用级别的存储卷的服务。所有该应用中的服务将共享同一个卷。 你需要指定卷ID以及改卷所在的可用区。 使用该AWS EBS卷的所有容器将会被调度到同一台主机上。

```
version: '2'
services:
  foo:
    image: busybox
    stdin_open: true
    volumes:
    - bar:/var/lib/storage

volumes:
  bar:
    driver: rancher-ebs
    driver_opts:
      size: 10
      volumeID: <VOLUME_ID>
      # Specifying the availability zone is required when using volumeID
      ec2_az: <AVAILABILITY_ZONE_WHERE_THE_VOLUME_IS>
```

调度服务

Rancher的核心调度逻辑是Rancher的一部分，它可以处理端口冲突和根据主机 / 容器上的标签进行调度的能力。除了核心调度逻辑，Rancher还使用应用商店里的Rancher Scheduler支持额外的调度策略。

- 能够调度多IP的主机
- 基于资源约束的调度能力 (例如CPU和内存)
- 能够限制在主机上调度哪些服务

> 注意： 这些特性不适用于Kubernets，因为Kubernets自己处理pod的调度。

启用RANCHER调度程序

Rancher调度服务需要在环境中启用。如果你将它从你的环境中删除了，可以在应用商店 -> 官方认证中添加应用Rancher Scheduler。

多IP主机调度

默认情况下，Rancher假定在调度发布有暴露端口的服务以及启动负载均衡器时，主机上只有一个IP可用。如果你的主机有多个可以使用的IP，则需要配置主机以允许Rancher调度程序知道哪些IP可用。

当主机上有多个IP可用于调度时，当通过服务或者负载均衡器发布端口时，Rancher将对所有可用的调度IP进行编排。当主机上的所有可用的调度IP都被分配给那个端口之后，调度器将会报告端口冲突。

基于资源约束的调度

当Rancher主机被添加到Rancher时，它们将根据主机的大小自动限制资源。可以通过编辑主机来调整这些限制。在基础架构 -> 主机中，你可以从主机的下拉框中选择编辑。在主机的资源限制选项中，你可以更新内存或者CPU为你期望需要用到的最大值。

在UI上设置资源预留

创建服务时，可以在安全/主机选项卡中指定内存预留和mCPU预留。设置这些预留时，服务的容器只能安排在具有可用资源的主机上。主机上这些资源的最大限制是根据主机的资源限制确定的。如果将容器调度到主机上会迫使这些限制超过阈值，则容器将不会被调度到主机上。

在RANCHER COMPOSE中设置预留

docker-compose.yml示例

```
version: '2'
services:
  test:
    image: ubuntu:14.04.3
    stdin_open: true
    tty: true
    # Set the memory reservation of the container
    mem_reservation: 104857600
rancher-compose.yml示例

version: '2'
services:
  test:
    # Set the CPU reservation of the container
    milli_cpu_reservation: 10
    scale: 1
```

在主机上调度指定服务

通常，大部分的容器调度规则定义在了服务中。服务对可以运行容器的主机设置了一些规则或限制。例如，容器必须安排在具有特定标签的主机上。Rancher还可以支持只允许将特定的容器调度到某个主机上。例如，你可能希望某台专用主机只运行数据库容器。

> 注意： 当你在主机上添加运行容器的限制标签时，你将需要包含一个特定标签，以便将Rancher的基础设施服务调度到主机上。没有这些服务，容器将无法正常运行。

对于任何主机，你可以通过从主机的下拉列表中选择编辑来编辑主机以添加可运行容器必须具有的标签。 在容器标签需求选项中，你可以添加要在服务中使用哪些标签，以便将这些容器调度到主机上。UI将自动将标签（例如，io.rancher.container.system =`）标记为必需的标签。

审计日志

只有管理员用户有权限访问审计日志。审计日志在系统管理->审计日志。

Rancher的审计日志是不同事件类型的集合。

- 任何带有前缀api的事件是API的一次调用。事件类型将记录API操作，谁执行的操作以及API调用的方式（即通过UI，通过API密钥）。
- 任何没有带api前缀的事件都是Rancher Server做的事情。例如，在协调服务的容器期间，在实例创建时会产生一个instance.create事件。

服务账号

你创建了一个需要和Rancher API交互的容器，你需要创建服务账号API keys，这样我们就可以访问带有权限认证的API来。为了在服务中创建这些keys，需要给服务添加以下的标签。

Key	Value	描述
io.rancher.container.create_agent	true	标识服务账号栋API keys会被添加到每个容器的环境变量里。
io.rancher.container.agent.role	environment	标识账号的角色。创建服务账号的值为environment.

当服务中的容器启动时，以下环境变量会被加入到容器中

Key	Value
CATTLE_URL	主机注册地址的URL。
CATTLE_ACCESS_KEY	启动的服务所在环境的访问密钥。
CATTLE_SECRET_KEY	访问密钥对应的安全密钥。

应用

应用包含了一组服务。你可以把多个服务放在一起组成一个应用。

添加应用

在应用页，点击添加应用。你需要输入一个名称然后点击创建.

之后会进到这个刚创建的应用页面里。你可以开始在应用里添加服务，添加负载均衡，添加服务别名，或者添加外部服务。

> 注意： 在启动服务之前，你需要至少向Rancher环境添加一台主机。更多添加主机的内容，请查看文档。

你也可以通过导入compose文件来创建应用。在应用创建页面可以导入docker-compose.yml和rancher-compose.yml文件。你可以在创建应用页面里直接上传文件，也可以把文件中的内容通过复制粘贴输入到创建页面上。当你点击创建之后，一个由相关服务组成的应用就创建成功了。通过docker-compose.yml文件来创建的服务，仅会被创建但并不会被启动。你需要手动启动他们。

查看应用中的服务

在应用列表页面，你可以轻松的监控该环境内所有应用的状态。 你可以点击应用左侧的加号来展开应用，并查看应用里面的每个服务。 你也可以点击应用名称，进入应用详情页面。

应用详情页面展示了应用内的全部服务。你可以点击服务名称，进入服务详情页面。在服务详情页面，可以点击容器名称，进入容器详情页面。

应用配置

当应用被创建时，Rancher同事生成了docker-compose.yml文件和rancher-compose.yml文件。docker-compose文件可以用在Rancher之外。你可以通过原生的docker-compose命令来启动服务。更多文档请查看docker-compose.

rancher-compose.yml文件包含了Rancher启动服务时所需的额外信息。docker-compose文件内并不支持这些参数。

有了这两个文件，你也用可以用Rancher Compose命令行来启动服务。

查看配置

在应用的下拉列表里，你可以选择查看配置或者点击应用详情页右上角的文件图标。

导出配置

下面是导出应用配置的两种方法。

方法一：在应用的下拉菜单里点击导出配置按钮，可以下载一个zip包，包里包括docker-compose.yml和rancher-compose.yml文件。

方法二：在应用的下拉菜单里点击查看配置按钮，可以看到配置详情，点击docker-compose.yml和rancher-compose.yml旁边的按钮，可以将文件内容复制到剪贴板。

查看图形

你可以用另一种方法来查看应用。点击查看图形按钮，你可以通过可视化图形的方式，查看服务之间的关系。存在连接的两个服务，在图中会被用线连起来。

修改服务

可能你创建了不同的Rancher服务。但是在创建完成之后，所有服务的操作下拉菜单都是相同的。例如，服务与负载均衡的下拉菜单是相同的。

容器数量

对服务和负载均衡来说，你可以点击服务详情页面的加号快速对其进行扩容。扩容后，新的容器将会被添加到服务中。

> 注意： 对于负载均衡，如果你对其扩容的最终数量超过了有可用开放端口的主机数量。负载均衡将会卡在Updating-Active状态。如果卡住了，解决方法是停掉该负载均衡，并且把容器数量修改到和可用主机数量相同。

你也可以通过点击服务下拉菜单的编辑按钮来增加或者减少服务内容器的数量。在编辑服务的弹出框内，你可以通过滑动条来修改容器数量。

修改

在这里可修改的参数是有限的，因为容器在创建之后是不可变的。这也包括重启容器，你停止和启动的都是同一个容器。你所能修改的都是Rancher存储的一些参数，而不是Docker容器本身的参数。如果你想要修改其他参数，你可以通过升级或者克隆这个服务来进行修改。

你可以点击服务下拉菜单中的编辑按钮，来查看你可以修改的参数。你可以修改服务名称，服务描述和服务中容器的数量。如果你在创建服务的时候，忘了增加相关连接。你可以在编辑页面设置连接。

对服务来说，大多数参数都不能被修改，因为容器在创建之后是不可变的。为了摆脱这个限制，你可以克隆一个服务。克隆会创建一个和该服务全部参数都相同的新的服务，你可以在点击创建之前修改你想要更新的参数。

克隆

你可以克隆任何服务，克隆的服务包含原服务的全部配置。但是其他服务里指向到原服务的连接并不会被克隆。你需要通过修改那些服务，把指向原服务的连接指向克隆出来的服务上。

示例：

服务A连接到了服务B。如果克隆服务B，得到服务C。这时服务A并不会连接到服务C。让服务A建立与服务C的连接的唯一方法就是修改服务A，添加指向服务C的连接。

停止

你可以停止某个服务，也可以一键停止应用内全部的服务。如果你想要停掉某个服务，可以点击服务下拉列表里的停止按钮。如果你想要停掉应用里的全部服务，可以点击应用下拉菜单里的停止服务按钮。

删除

你可以单独删除服务也可以删除整改应用。当你选择删除某个服务的时候，这个服务中的容器将会先被停止，然后被从主机上删除。这可能会稍微有些延迟，因为Rancher会先清理主机上的容器，然后才会在UI上显示容器已删除。

服务

- Cattle对服务采用标准Docker Compose术语，并将基本服务定义为从同一Docker镜像创建的一个或多个容器。一旦服务（消费者）链接到同一个应用中的另一个服务（生产者）相关的DNS记录 会被自动创建，“消费”服务的容器可以发现这些容器。在Rancher创建服务的其他好处包括：
- 服务高可用性（HA）：Rancher会不断监控服务中的容器状态，并主动管理以确保所需的服务实例规模。当健康的容器小于（或者多于）正常服务所需容器规模，主机不可用，容器故障或者不能满足健康检查就会被触发。
- 健康检查: Rancher通过在主机上运行healthcheck的基础设施服务，从而实现了容器和服务的分布式健康检查系统。这个healthcheck服务内部使用HAProxy来检查应用程序的运行状况。当在单个容器或服务上启用健康检查时，Rancher将监控每个容器。

用户界面中的服务选项

在以下示例中，我们假设你已经创建了一个应用, 设置了你的主机，并准备好开始构建应用程序来。

我们将在添加服务的过程中了解一些服务的选项，最后将介绍如何创建一个连接到Mongo数据库的LetsChat应用程序。

在应用中，你可以通过单击添加服务按钮添加服务。也可以在应用列表中添加服务 ，每个单个应用都可以看到添加服务按钮。

在数量部分，你可以使用滑块来指定要为服务启动的容器的数量。或者，你可以选择总是在每台主机上运行一个此容器的实例。使用此选项时，你的服务将被部署到该环境中的任何主机上。如果你在调度选项卡中创建了调度规则，则Rancher将仅在符合调度规则的主机上启动容器。

你还需要输入名称，如果需要，还可以输入服务描述。

为服务设置所需的镜像。你可以使用DockerHub上的任何镜像，以及已添加到你的环境中的任何镜像仓库。镜像名称的语法与docker run命令中使用的语法相同。

镜像名称的语法。默认情况下，我们从Dockerhub中拉取。如果没有指定标签，我们将拉取标签为tag的镜像。

[registry-name]/[namespace]/[imagename]:[version]

在镜像名称下方，有一个复选框创建前总是拉取镜像。默认情况下，这是被勾选的。选择此选项后，每次在主机上启动容器的时候，都将始终尝试拉取镜像，即使该镜像已被缓存在了该主机上。

选项

Rancher努力与Docker保持一致，我们的目标是，支持任何docker run所支持的选项。端口映射和服务链接显示在主页面上，但所有其他选项都在不同的选项卡中。

默认情况下，服务中的所有容器都以分离模式运行，例如：docker run命令中的-d。

端口映射

当配置了映射端口后，你可以通过主机上的公共端口访问容器暴露的端口。在端口映射部分中，需要设置暴露在主机上的端口。该端口将流量指向你设置的私有端口。私有端口通常是容器上暴露的端口（例如：镜像的Dockerfile中的EXPOSE）。当你映射一个端口时，Rancher将会在启动容器之前检查主机是否有端口冲突。

当使用端口映射时，如果服务的容器规模大于具有可用端口的主机数量时，你的服务将被阻塞在正在激活状态。如果你查看服务的详细信息，你将可以看到Error状态的容器，这表明容器由于无法在主机上找到未被占用的端口而失败。该服务将继续尝试，如果发现有主机/端口可用，则该服务将在该主机上启动一个容器。

> 注意： 当在Rancher中暴露端口时，它只会显示创建时暴露端口。如果端口映射有任何改变，它不会在docker ps中更新，因为Rancher通过管理iptable规则，来实现端口动态变更的。

随机端口映射

如果你想要利用Rancher的随机端口映射，公共端口可以留空，你只需要定义私有端口。

链接服务

如果你的环境中已经创建了其他服务，则可以将已有服务链接到你正在创建的服务。正在创建的服务中的所有容器都会链接到目标服务中的所有容器。链接就像docker run命令中的--link功能一样。

链接是基于Rancher内部DNS的附加功能，当你不需要按服务名称解析服务时，可以使用链接。

RANCHER 选项

除了提供docker run支持的所有选项之外，Rancher还通过UI提供了额外选项。

健康检查

如果Rancher中主机不能正常工作来（例如：处于reconnecting或inactive状态），你需要配置健康检查，以使Rancher将服务中的容器调度到其他的主机上。

> 注意： 健康检查仅适用于托管网络的服务。如果你选择任何其他网络，则不能被监察到。

在健康检查选项卡中，你可以选择检查服务的TCP连接或HTTP响应。

阅读有关Rancher如何处理健康检查的更多详细信息。

标签/调度

在标签选项卡中，Rancher允许将任何标签添加到服务的容器中。标签在创建调度规则时非常有用。在调度选项卡中，你可以使用主机标签，容器/服务标签，和容器/服务名称来创建你服务需要的调度规则。

阅读有关标签与调度的更多细节。

在UI中添加服务

首先，我们通过设置数量为1个容器的服务来创建我们的数据库，给它设置名称database，并使用mongo:latest镜像。不需要其他的配置，点击创建。该服务将立即启动。

现在我们已经启动了我们的数据库服务，我们将把web服务添加到我们的应用中。这一次，我们将服务规模设置为2个容器，创建一个名称为web并使用sdelements/lets-chat作为镜像的服务。我们没有暴露Web服务中的任何端口，因为我们将添加负载均衡来实现服务访问。我们已经创建了数据库服务，我们将在服务链接的目标服务选择数据库服务，在名称中填写mongo。点击创建，我们的LetsChat应用程序已准备好了，我们马上可以用负载均衡服务来暴露端口了。

RANCHER COMPOSE 中的服务选项

阅读更多关于配置Rancher Compose的细节。

Rancher Compose工具的工作方式和Docker Compose一样，并支持V1和V2版本的docker-compose.yml文件。要启用Rancher支持的功能，你还可以使用扩展或重写了docker-compose.yml的rancher-compose.yml文档。例如，rancher-compose.yml文档包含了服务的scale和healthcheck。

如果你不熟悉Docker Compose或Rancher Compose，我们建议你使用UI来启动你的服务。你可以通过单击应用的下拉列表中的查看配置来查看整个应用的配置（例如：与你的应用等效的docker-compose.yml文件和rancher-compose.yml文件）。

链接服务

在Rancher中，环境中的所有服务都是可以通过DNS解析的，因此不需要明确设置服务链接，除非你希望使用特定的别名进行DNS解析。

> 注意： 我们目前不支持将从服务与主服务相关联，反之亦然。阅读更多关于Rancher内部DNS工作原理。

应用中的服务都是可以通过服务名称service_name来解析的，当然，你也可以通过链接来使用其他名称进行解析。

例子 DOCKER-COMPOSE.YML

```
version: '2'
services:
  web:
    labels:
      io.rancher.container.pull_image: always
    tty: true
    image: sdelements/lets-chat
    links:
    - database:mongo
    stdin_open: true
  database:
    labels:
      io.rancher.container.pull_image: always
    tty: true
    image: mongo
    stdin_open: true
```

在这个例子中，mongo可以解析为database。如果没有链接，web服务需要通过服务名称database来解析数据库服务。

对于不同应用中的服务，可以使用service_name.stack_name对服务进行解析。如果你希望使用特定别名进行DNS解析，则可以在docker-compose.yml中使用external_links。

例子 DOCKER-COMPOSE.YML

```
version: '2'
services:
  web:
    image: sdelements/lets-chat
    external_links:
    - alldbs/db1:mongo
```

在此示例中，alldbs应用中的db1服务将链接到web服务。在web服务中，mongo将可解析为db1。没有外部链接时，db1.alldbs将可解析为db1。

> 注意： 跨应用的服务发现受环境的限制（特意设计的）。不支持应用的跨环境发现。

使用 RANCHER COMPOSE 添加服务

阅读更多关于配置Rancher Compose的详情.

我们将创建与上面通过UI创建的相同示例。首先，你将需要创建一个docker-compose.yml文件和一个rancher-compose.yml文件。使用Rancher Compose，我们可以一次启动应用程序中的所有服务。如果没有rancher-compose.yml文件，则所有服务将以1个容器的规模启动。

例子 DOCKER-COMPOSE.YML

```
version: '2'
services:
  web:
    labels:
      io.rancher.container.pull_image: always
    tty: true
    image: sdelements/lets-chat
    links:
    - database:mongo
    stdin_open: true
  database:
    labels:
      io.rancher.container.pull_image: always
    tty: true
    image: mongo
    stdin_open: true
```

例子 RANCHER-COMPOSE.YML

```
# 你想要拓展的效果服务
version: '2'
services:
  web:
    scale: 2
  database:
    scale: 1
```

创建文件后，可以将服务部署到Rancher Server。

```
#创建并启动一个没有环境变量的服务并选择一个应用
#如果没有提供应用名称，应用的名称将是命令运行的文件夹名称
#如果该应用没有存在于Rancher中，它将会被创建
$ rancher-compose --url URL_of_Rancher --access-key <username_of_environment_api_key> --secret-key <password_of_environment_ap
i_key> -p LetsChatApp up -d

#创建并运行一个已经设置好环境变量的服务
$ rancher-compose -p LetsChatApp up -d
```

从服务

Rancher支持通过使用从服务的概念对服务进行分组，从而使一组服务可以同时进行调度和扩缩容。通常创建具有一个或多个从服务的服务，来支持容器之间共享卷（即--volumes_from）和网络（即--net=container）。

你可能希望你的服务的使用volumes_from和net去连接其他服务。为了实现这一点，你需要在服务直接建立一个从属关系。通过从属关系，Rancher可以将这些服务作为一个单元进行扩容和调度。例如：B是A的从服务，Rancher始终将A和B作为一对进行部署，服务的数量规模将始终保持一致。

如果你有多个服务总需要部署在同一主机上，你也可以通过定义从属关系来实现它。

当给一个服务定义一个从服务时，你不需要链接该服务，因为从服务会自动被DNS解析到。

当在服务中使用负载均衡时，而该服务又拥有从服务的时候，你需要使用主服务作为负载均衡器的目标。从服务不能成为目标。 了解更多关于Rancher内部DNS的详情。

在UI中添加从服务

要设置一个从服务，你可以点击+添加从容器按钮，按钮位于页面的数量那部分。第一个服务被认为是主服务，后面每个附加的从服务都是辅助服务。

通过RANCHER COMPOSE添加从服务

要设置sidekick关系，请向其中一个服务添加标签。标签的键是io.rancher.sidekicks，该值是从服务。如果你要将多个服务添加为从服务，可以用逗号分隔。例：io.rancher.sidekicks: sidekick1, sidekick2, sidekick3

主服务

无论哪个服务包含sidekick标签都被认为是主服务，而各个sidekicks被视为从服务。主服务的数量将用作sidekick标签中所有从服务的数量。如果你的所有服务中的数量不同，则主服务的数量将用于所有服务。

当使用负载均衡器指向带有从服务的服务时，你只能指向主服务，从服务不能成为目标。

RANCHER COMPOSE里面的从容器例子:

例子docker-compose.yml

```
version: '2'
services:
  test:
    tty: true
    image: ubuntu:14.04.2
    stdin_open: true
    volumes_from:
    - test-data
    labels:
      io.rancher.sidekicks: test-data
  test-data:
    tty: true
    command:
    - cat
    image: ubuntu:14.04.2
    stdin_open: true
```

例子 rancher-compose.yml

```
version: '2'
services:
  test:
    scale: 2
  test-data:
    scale: 2
```

RANCHER COMPOSE里面的从服务例子:多服务使用来自同一个服务VOLUMES_FROM

如果你有多个服务，他们将使用相同的容器去做一个volumes_from，你可以添加第二个服务作为主服务的从服务，并使用相同的数据容器。由于只有主服务可以作为负载均衡的目标，请确保选择了正确的服务作为主服务（即，具有sidekick标签的服务）。 示例 docker-compose.yml

```
version: '2'
services:
  test-data:
    tty: true
    command:
    - cat
    image: ubuntu:14.04.2
    stdin_open: true
  test1:
    tty: true
    image: ubuntu:14.04.2
    stdin_open: true
    labels:
      io.rancher.sidekicks: test-data, test2
    volumes_from:
    - test-data
  test2:
    tty: true
    image: ubuntu:14.04.2
    stdin_open: true
    volumes_from:
    - test-data
```

卷

概览

持久化卷是有状态应用中非常重要的一部分。Rancher使你在多主机环境下使用卷变得非常容易。

术语

卷插件和卷驱动同时在Docker和Rancher中使用。他们代表的是同一个东西: 一个Docker卷插件 可以给一个Docker容器提供本地卷或者共享的持久化卷的支持。Rancher卷插件(驱动)目前是以Docker卷插件的形式实现的。并且可以通过docker volume命令行来进行操作，但是这取决于存储技术。卷可以被环境中的某个主机，某些主机或者全部主机访问。Rancher对跨主机使用共享卷的复杂过程进行了封装。例如：rancher-nfs, rancher-ebs 和 pxd (portworx)。

存储驱动是关于容器和镜像是如何在Docker主机上被存储的。例如：aufs, btrfs, zfs, devicemapper等。这些驱动在Rancher的管控范围之外。Rancher UI混合了这个术语，但实际上指的是卷插件和卷驱动。更多关于存储驱动的插件信息信息，请查看这里。

管理卷

在这一部分，你将会了解如何创建一个可以被容器之间共享的持久化卷。在这里我们将专门使用Rancher命令行。

> 注意： UI可以用来管理除了由local卷驱动创建的卷。

创建卷

你可以通过rancher volume create命令创建一个卷。

```
$ rancher volume create --driver local app-data
```

这将创建一个新的名为app-data的本地卷。名称必须由字母数字开头，后面可以接a-z0-9，_ (下划线), . (点) 或者- (横杠)。

--driver参数用来指定使用哪个卷驱动。Docker提供了一个local卷驱动。使用这个驱动的卷会将数据保存在主机的文件系统上，并且同一台主机上的任何容器都可以访问该数据。当使用local卷驱动时，其他主机上的任何容器都无法共享这个数据卷。

列出卷

你可以列出环境中的卷。

```
$ rancher volume ls
```

如果你创建了一个app-data卷，你可能想知道为什么这个卷没有被列出来。你可以通过添加--all或者-a参数，来查看inactive的卷。

```
$ rancher volume ls --all
```

删除卷

你可以通过rancher volume rm命令删除一个卷。

```
$ rancher volume rm app-data
```

卷状态

卷有七个不同的状态：inactive, activating, active, deactivating, detached, removing 和 removed。

一个刚刚建好的卷的状态是inactive，直到有容器尝试挂载这个卷。

当容器创建时，关联的卷进入activating状态。一旦容器进入了running阶段，它等容器就会进入active状态。如果容器去挂载一个已经是active状态的卷，这并不会对该卷的状态产生影响。

当全部挂载了这个卷的容器都被开始删除了，这个卷进入deactivating状态。一旦容器被删除成功，卷进入detached状态。

当卷被标记为删除时，它进入一个removing状态。一旦数据被从主机上删除成功，它进入removed状态。被删除的卷不会出现在列出卷的结果里。但是它们将会继续在Rancher API里存在一段时间，这是为了调试和审计的目的。

卷作用域

卷可以有不同的作用域。这指的是Rancher管理卷的不同级别。

目前，你可以通过Rancher Compose文件来创建不同类型的卷。有作用域的卷必须定义在docker-compose.yml文件中最顶层的volumes部分。默认情况下，将创建应用栈级别的卷，但是你可以通过修改其值来创建不同作用域的卷。

如果最顶层的定义被遗漏了，卷的行为将会有所不同。请查看更多详情。

通过UI你只能创建环境级别的卷。

应用级别的卷

应用级别的卷是由创建它的应用来管理的。主要的好处是这种卷的生命周期是其应用生命周期的一部分，将由Rancher自动管理。

应用级别的存储卷，应用中的服务如果引用了相同的卷都将共享同一个存储卷。不在该应用内的服务则无法使用此存储卷。

Rancher中，应用级别的存储卷的命名规则为使用应用名称为前缀来表明该卷为此应用独有，且会以随机生成的一组数字结尾以确保没有重名。在引用该存储卷时，你依然可以使用原来的卷名称。比如，如果你在应用 stackA中创建了一个名为foo 的卷， 你的主机上显示的卷名称将为stackA*foo*，但在你的服务中，你依然可以使用foo。

创建示例

下面的例子中，将会创建应用级别的卷data。

> 注意： 因为在文件中最顶层的volumes部分不存在其他配置值，所以这个卷的级别为应用级。

```
version: '2'
services:
  redis:
    image: redis:3.0.7
    volumes:
    - data:/data
volumes:
  data:
    driver: local
```

在上面的例子中，我们特意指定了卷驱动为local。默认情况下，卷驱动的值就是local。最简洁的定义data的方法是设置一个空值{}。请看下面的例子。

```
volumes:
  data: {}
```

在通过Rancher命令行创建应用之后，你可以列出卷来确认data卷已经创建成功。这个卷的名称将为*data*。

> 注意： 应用级别的卷可以被其他应用挂载。应用级别的卷并不是一种安全的机制。仅是为了管理卷的生命周期。

环境级别的卷

环境级别的卷可能需要被环境中的全部容器共享。Rancher会把容器调度到卷所在的主机，从而保证容器可以挂载这个卷。

环境级别的卷并不能在某个环境中的全部的主机上自动共享。你需要借助一个共享驱动（例如：rancher-nfs）来实现跨主机共享。这意味着一个由local卷驱动创建的环境级别卷只能在一台主机上被访问到，所以使用该卷的容器都会被调度到这台主机上。

你在创建一个使用环境级别卷的服务之前，Rancher需要你先创建这个环境级别的卷。你可以使用任何配置好的卷驱动来创建卷。

环境级别卷的主要好处是，你可以轻松的在不同的服务应用之间共享数据。尤其是当这些应用和服务有着不同的生命周期需要被独立管理。用户可以对卷进行完全的管控

共享的环境级别卷示例

首选，创建一个环境级别的卷从而使其他应用共享这个卷。

```
$ rancher volume create --driver local redis-data-external
```

为了创建一个环境级别的卷，在最顶层的volume部分，你需要添加external: true。

```
version: '2'
services:
  redis:
    image: redis:3.0.7
    volumes:
    - redis-data-external:/data
volumes:
  redis-data-external:
    driver: local
    external: true  # 如果没有这个定义，将会创建一个应用级别的卷。
```

> 注意： 如果在volume中没有定义external: true，这个卷将会被创建为一个应用级别的卷.

在通过Rancher命令行创建应用之后，你可以列出卷来确认redis-data-external卷已经创建成功并且状态为active。

> 注意： 对一个服务进行扩容和缩容的时候，将会挂载或卸载同一个共享的卷。

任何新的应用都可以挂载同一个redis-data-external卷。最简单的方法就是复制compose文件中最顶层的volume部分。

```
volumes:
  redis-data-external:
    driver: local
    external: true
```

V1与V2版本的COMPOSE对比

直到这里，我们讨论的一直是Docker V2 Compose的卷。如果你没有在V2 compose文件中的最顶层定义volumes，它将会按照Docker V1 Compose的方式来处理卷。

你也可能用到了Docker V1 Compose。在V1 compose文件中，不支持最顶层的volume部分。所以这时卷只能被定义在服务里。Rancher会把V1的卷定义直接传递给Docker。所以，卷不会被自动删除同时也无法确保可以正常调度到卷所在的主机。为了解决这个在V1卷下的调度问题，你需要使用调度标签。

> 注意： 请尽可能不要使用V1版本的Compose。

V1版本的COMPOSE示例

注意这里没有volumes部分；这个配置在V1中不存在。

```
etcd:
  image: rancher/etcd:v2.3.7-11
  volumes:
  - etcd:/pdata
```

V1和V2版本的COMPOSE文件

Docker compose的V2版本是V1版本的超集；两个格式都可以被使用。我们先创建一个环境级别的卷。

```
$ rancher volume create --driver local etcd_backup
```

这个例子中，etcd_backup是一个V2的环境级别的卷，etcd是一个V1的卷。因为没有定义volume，这隐式的将etcd设置为了V1的卷。

```
version: '2'
services:
  etcd:
    image: rancher/etcd:v2.3.7-11
    environment:
      EMBEDDED_BACKUPS: 'true'
      BACKUP_PERIOD: 5s
      BACKUP_RETENTION: 15s
    volumes:
    - etcd:/pdata
    - etcd_backup:/data-backup
volumes:
  etcd_backup:
    driver: local
    external: true
```

最后，如果你定义了一个空的volumes，这个卷将会被视为一个V1卷。这等同于yaml中完全没有volumes这部分。

```
version: '2'
volumes: {}
```

负载均衡

Rancher支持多种负载均衡驱动，通过在它之上建立代理规则，可以将网络及应用流量分发至指定的容器中。负载均衡的目标服务中的容器都会被Rancher自动注册为负载均衡的目标。在Rancher中，将负载均衡加入到应用中是一件非常容易的事情。

默认情况下，Rancher提供一个基于HAProxy的托管的负载均衡，它可以被手动扩容至多台主机。在接下来的例子中将会涉及到负载均衡中不同的配置项，这些配置项主要以HAProxy为参考。我们计划支持除HAProxy以外的其他负载均衡驱动，但这些配置项都会是相同的。

我们使用round robin算法分发流量至目标服务。这个算法可在自定义HAProxy.cfg中进行自定义。 另外，你可以配置负载均衡来将流量分发至与负载均衡容器处于相同主机的目标容器。通过给负载均衡设置一个特定的标签，能够将负载均衡的目标限定在同一台主机中的目标容器（例如 io.rancher.lb_service.target=only-local），或者优先转发至同一台主机中的目标容器(例如： io.rancher.lb_service.target=prefer-local)。

我们将会查看在UI和Rancher Compose中的负载均衡的配置项，并且给出UI和Rancher Compose的用例。

如何在UI上新增一个负载均衡

我们将为我们在添加服务部分中创建的“letschat”应用新增一个负载均衡.

首先，我们从添加一个负载均衡服务开始，点击”添加服务“旁边的下拉图标，找到添加负载均衡并点击它。

进入添加页面后，容器数量默认是1，填入“名称”，如“LetsChatLB”。

端口规则下，访问选择默认的公开，协议选择默认的HTTP。请求端口填入80，目标选择letschat服务， 并且端口填入8080。

点击创建。

现在，让我们来实际感受一下负载均衡。在应用视图下， 有一个连接到80端口的超链接，这是负载均衡中的源端口。如果你点击它，将会在你的浏览器中自动新开一个页签，并指向负载均衡服务所在的主机。请求将会被重定向到其中一个”LetsChat“容器。如果你刷新浏览器，负载均衡服务会把新的请求重定向到“LetsChat”服务下的其他容器中。

页面上的负载均衡选项

Rancher提供一个基于HAProxy的容器来重定向流量至目标服务。

> 注意： 负载均衡只会在那些使用托管网络的服务中生效，其他网络模式都不会生效。

点击添加服务旁边的下拉图标，找到添加负载均衡并点击它。

你能使用滑块选择数量，就是负载均衡使用多少个容器。或者，你可以选择总是在每台主机上运行一个此容器的实例。使用这一个选项, 你的负载均衡容器数量将会随着你环境下的主机数量增减而增减。如果你在调度部分设定了调度规则，Rancher将会在满足规则的主机上启动负载均衡。如果你的环境下新增了一台不满足调度规则的主机，负载均衡容器不会在这一台主机中启动。

> 注意： 负载均衡容器的扩缩容不能超过环境下主机的数量，否则会造成端口冲突，负载容器服务将会被阻碍在activating状态。它会不断去尝试寻找可用的主机并开启端口，直到你修改它的数量或者添加主机.

你需要提供负载均衡的名称，如果有需要的话，你可以添加描述。

接下来，你可以定义负载均衡的端口规则。有两种规则类型可供创建。用于目标为特定的已存在的服务的服务规则和用于匹配一定选择规则的选择器规则。

当创建了多条服务和选择器规则的时候，请求头和路径规则将会自顶向下按显示在UI上的顺序匹配。

服务规则

服务规则指的是端口指向目标容器的规则。

在访问选项栏中，你可以决定这个负载均衡端口是否可以被公网访问（就是说是否可以从主机以外访问）或者仅仅在环境内部访问。默认情况下，Rancher假定你希望被公网访问，但是如果你希望仅仅在环境内部被访问，你可以选择内部。

选择协议选项栏。获取更多关于我们协议选项的信息。如果你选择了需要SSL终端（如 https or tls），你将需要在SSL终端标签页中新增你的认证信息。

接下来，你可以针对流量的来源填写请求头信息, 端口 和 路径。

> 注意： 42 端口不能被用作负载均衡的源端口，因为它被用于健康检查。

请求头信息 / 路径

请求头信息是HTTP请求头中的host的属性。请求路径可以是一段特殊的路径。你可以任意设置其中一个或者两者都设置。

例子:

```
domain1.com -> Service1
domain2.com -> Service2

domain3.com -> Service1
domain3.com/admin -> Service2
```

通配符

当基于HOST值设置路由时，Rancher支持通配符。所支持的语法如下。

```
*.domain.com -> hdr_end(host) -i .domain.com
domain.com.* -> hdr_beg(host) -i domain.com.
```

目标服务和端口

每一个服务规则，你都可以选择你想要的目标服务。这些服务列表是基于该环境下所有的服务清单的。每一个服务，你还能选择与之配套的端口。服务上的私有端口通常就是镜像所暴露的端口。

选择器规则

在选择器规则中，你需要填写一个选择器标签而不是特定的服务。选择器基于服务的标签来选择目标服务。当负载均衡被创建的时候，选择器规则将会针对环境下现有的任意一个服务来计算看是否为可匹配的服务。后面新增的服务或者对标签进行修改都会拿来与选择器标签进行匹配。

对于每一个源端口，你都可以添加相应的请求头信息或路径。选择器标签是基于目标的，你能指定一个特定的端口接收转发到服务上的流量。服务上的私有端口通常就是镜像所暴露的端口。

例子: 2 选择器规则

1. 源端口: 100; 选择器: foo=bar; 端口: 80
2. 源端口: 200; 选择器: foo1=bar1; 端口: 80
3. 服务A有一个 foo=bar 标签，它将会匹配第一条规则. 任何指向100的流量都会被转发到服务A。
4. 服务B有一个foo1=bar 标签，它将会匹配第二条规则. 任何指向200的流量都会被转发到服务B。
5. 服务C有foo=bar和foo1=bar1两个标签，它将会匹配两条规则. 任何指向200和100的流量都会被转发到服务C.

> 注意： 目前，如果你想要将一条选择器规则应用于多个主机名 / 路径上，你需要使用Rancher Compose在目标服务上去设置主机名 / 路径。

SSL会话终止

SSL会话终止标签提供了添加用于https和tls协议证书的能力。在证书下拉框中，你可以为负载均衡选择主证书。

添加证书前，请阅读如何添加证书.

为负载均衡添加多个证书是可以实现的。这样相应的证书会基于请求主机名(查看 服务器名称指示)展示给客户端。这个功能可能在那些不支持SNI(它会获取主要证书)的老客户端上失效。对于现代客户端，我们会在能匹配到的列表中提供证书，如果没有匹配成功，就会提供主证书。

负载均衡的会话粘性

你可以点击选择负载均衡的会话粘性。会话粘性就是你的cookie策略。

Rancher支持以下两种选项：

- 无: 这个选项意味着不会设置cookie策略
- 创建新的Cookie: 这个选项意味着在你的应用之外会创建cookie。这个cookie是由负载均衡设置在请求与响应中的。这就是会话粘性策略。

自定义HAPROXY.CFG

由于Rancher基于HAProxy来搭建负载均衡，所以你能自定义HAproxy的配置。你在这里定义的配置都会被添加到Rancher生成的配置的最后面。

自定义HAPROXY配置的例子

```
global
    maxconn 4096
    maxpipes 1024

defaults
    log global
    mode    tcp
    option  tcplog

frontend 80
    balance leastconn

frontend 90
    balance roundrobin

backend mystack_foo
    cookie my_cookie insert indirect nocache postonly
    server $IP <server parameters>

backend customUUID
    server $IP <server parameters>
```

标签 / 调度负载均衡

我们支持向负载均衡添加标签并且调度负载均衡在哪启动。点击这里查看更多关于标签和调度的信息。

用RANCHER COMPOSE 添加负载均衡

在这，我们将一步步为我们之前在创建服务章节创建的”letschat”应用设置一个负载均衡。

点击这里查看更多关于如何配置一个Rancher Compose。

> 注意：：在我们的例子中，我们会使用作为负载均衡镜像的标签。每一个Rancher版本都有特定的，被负载均衡所支持的lb-service-haproxy版本。

我们将会建立一个和我们上面在UI中所使用到的例子一样范例。首先你需要创建一个docker-compose.yml文件和一个rancher-compose.yml文件。使用Rancher Compose，我们可以启动一个负载均衡

EXAMPLE DOCKER-COMPOSE.YML

```
version: '2'
services:
  letschatlb:
    ports:
    - 80
    image: rancher/lb-service-haproxy:<version>
```

EXAMPLE RANCHER-COMPOSE.YML

```
version: '2'
services:
  letschatlb:
    scale: 1
    lb_config:
      port_rules:
      - source_port: 80
        target_port: 8080
        service: letschat
    health_check:
      port: 42
      interval: 2000
      unhealthy_threshold: 3
      healthy_threshold: 2
      response_timeout: 2000
```

RANCHER COMPOSE 中的负载均衡配置

Rancher 提供一个基于HAProxy的容器来做负载均衡。

> 注意： 负载均衡仅仅在使用托管网络的服务中生效。其他的网络选择都不会生效。

负载均衡可以像其他任何一个服务一样被调度。点击这里获取更多关于在Rancher Compose中使用负载均衡的例子。

负载均衡由暴露在主机上的端口和负载均衡配置组成，这些配置包括针对不同目标服务的特定端口规则，自定义配置和会话粘性策略。

当与含有从容器的服务一起使用的时候，你需要将主服务作为目标服务，就是那些含有sidekick标签的服务。

源端口

当创建一个负载均衡的时候，你可以将任意一个你想要的端口暴露在主机上。这些端口都可以被用做负载均衡的源端口。如果你想要一个内部的负载均衡，就不要暴露任何端口在负载均衡上，只需要在负载均衡配置中添加端口规则。

> 注意： 42 端口 不能被用作负载均衡的源端口，因为它被用于健康检查。

EXAMPLE DOCKER-COMPOSE.YML

```
version: '2'
services:
  lb1:
    image: rancher/lb-service-haproxy:<version>
    # Any ports listed will be exposed on the host that is running the load balancer
    # To direct traffic to specific service, a port rule will need to be added.
    ports:
    - 80
    - 81
    - 90
```

LOAD BALANCER CONFIGURATION

所有负载均衡的配置项都被定义在rancher-compose.yml的lb_config字段中

```
version: '2'
services:
  lb1:
    scale: 1
    # All load balancer options are configured in this key
    lb_config:
      port_rules:
      - source_port: 80
        target_port: 80
        service: web1
    health_check:
      port: 42
      interval: 2000
      unhealthy_threshold: 3
      healthy_threshold: 2
      response_timeout: 2000
  web1:
    scale: 2
```

端口规则

端口规则是定义在rancher-compose.yml中的。因为端口规则是单独定义的，会有许多不同的端口指向同一个服务。默认情况下，Rancher将会优先使用那些基于特定的优先级顺序的端口。如果你想要改变这些优先级顺序，你需要设定特定的优先级规则。

默认优先级顺序

1. 没有通配符和URL的主机名
2. 没有通配符的主机名
3. 有通配符和URL的主机名
4. 有通配符的主机名
5. URL
6. 默认(没有主机名，没有URL)

源端口

源端口是值暴露在主机上的某个端口（也就是定义在docker-compose.yml中的端口）。

如果你想要创建一个内部负载均衡，那么源端口酒不需要与docker-compose.yml中定义的任意一个匹配。

目标端口

目标端口是服务内部端口。这个端口就是用于启动你容器的镜像所暴露的端口。

协议

Rancher的负载均衡支持多种协议类型。

- http - 默认情况下，如果没有设置任何协议，负载均衡就会使用http。HAProxy 不会对流量做任何解析，仅仅是转发。
- tcp - HAProxy 不会对流量做任何解析，仅仅是转发。
- https - 需要设置SSL会话终结。流量将会被HAProxy使用证书解密，这个证书必须在设定负载均衡之前被添加入Rancher。被流量负载均衡所转发的流量是没有加密的。
- tls - 需要设置SSL会话终结。流量将会被HAProxy使用证书解密，这个证书必须在设定负载均衡之前被添加入Rancher。被流量负载均衡所转发的流量是没有加密的。
- sni - 流量从负载均衡到服务都是被加密的。多个证书将会被提供给负载均衡,这样负载均衡就能将合适的证书基于主机名展示给客户端。 点击服务器名称指示）查看更多详情。
- udp - Rancher 的HAProxy不支持.

其他的负载均衡驱动可能只支持以上的几种。

主机名路由

主机名路由只支持http, https 和 sni，只有http 和 https同时支持路径路由。

服务

服务名就是你的负载均衡的目标。如果服务在同一个应用下，你可以使用服务名。如果服务在不同的应用下，你需要使用<应用名>/<服务名>。

EXAMPLE RANCHER-COMPOSE.YML

```
version: '2'
services:
  lb1:
    scale: 1
    lb_config:
      port_rules:
      - source_port: 81
        target_port: 2368
        # Service in the same stack
        service: ghost
      - source_port: 80
        target_port: 80
        # Target a service in a different stack
        service: differentstack/web1
    health_check:
      port: 42
      interval: 2000
      unhealthy_threshold: 3
      healthy_threshold: 2
      response_timeout: 2000
  ghost:
    scale: 2
```

主机名和路径

Rancher基于HAProxy的负载均衡支持七层路由，可以在端口规则下通过设定指定的主机头和路径来使用它。

EXAMPLE RANCHER-COMPOSE.YML

```
version: '2'
services:
  lb1:
    scale: 1
    lb_config:
      port_rules:
      - source_port: 81
        target_port: 2368
        service: ghost
        protocol: http
        hostname: example.com
        path: /path/a
    health_check:
      port: 42
      interval: 2000
      unhealthy_threshold: 3
      healthy_threshold: 2
      response_timeout: 2000
  ghost:
    scale: 2
```

通配符

当设置基于主机名的路由规则时，Rancher支持通配符。所支持的语法如下。

```
*.domain.com -> hdr_end(host) -i .domain.com
domain.com.* -> hdr_beg(host) -i domain.com.
```

优先级

默认情况下，Rancher 针对同一个服务遵循默认优先级顺序，但是你也可以定制化你自己的优先级规则（数字越小，优先级越高）

EXAMPLE RANCHER-COMPOSE.YML

```
version: '2'
services:
  lb1:
    scale: 1
    lb_config:
      port_rules:
      - source_port: 88
        target_port: 2368
        service: web1
        protocol: http
        hostname: foo.com
        priority: 2
      - source_port: 80
        target_port: 80
        service: web2
        protocol: http
        priority: 1
    health_check:
      port: 42
      interval: 2000
      unhealthy_threshold: 3
      healthy_threshold: 2
      response_timeout: 2000
  web1:
    scale: 2
```

选择器

你可以通过设定选择器来指定多个服务。通过使用选择器，你可以在目标服务上定义服务连接和主机名路由规则，那些标签匹配了选择器的服务将成为负载均衡的目标。

当使用选择器的时候，lb_config可以设定在负载均衡和任意一个匹配选择器的服务上。

在负载均衡器中。选择器标签 设置在selector下的lb_config中。负载均衡的lb_config端口规则不能有服务，并且也不能有目标端口。目标端口是设置在目标服务的端口规则中的。如果你需要使用主机名路由，主机名和路径是设置在目标服务下的。

> 注意: 对于那些在v1版本yaml中使用了的选择器标签字段的负载均衡，这不会被转化成v2版本的负载均衡。因为服务上的端口规则不会更新。

EXAMPLE DOCKER-COMPOSE.YML

```
version: '2'
services:
  lb1:
    image: rancher/lb-service-haproxy:<version>
    ports:
    - 81
  # These services (web1 and web2) will be picked up by the load balancer as a target
  web1:
    image: nginx
    labels:
      foo: bar
  web2:
    image: nginx
    labels:
      foo: bar
```

EXAMPLE RANCHER-COMPOSE.YML

```
version: '2'
services:
  lb1:
    scale: 1
    lb_config:
      port_rules:
      - source_port: 81
        # Target any service that has foo=bar as a label
        selector: foo=bar
        protocol: http
    health_check:
      port: 42
      interval: 2000
      unhealthy_threshold: 3
      healthy_threshold: 2
      response_timeout: 2000
  # web1 and web2 are targeted with the same source port but with the different hostname and path rules
  web1:
    scale: 1
    lb_config:
      port_rules:
      - target_port: 80
        hostname: test.com
  web2:
    scale: 1
    lb_config:
      port_rules:
      - target_port: 80
        hostname: example.com/test
```

后台名称

如果你想要清晰地在负载均衡配置中标明你的后台，你需要使用backend_name。如果你想要为一个某个后台自定义配置参数，这就会用得上。

证书

如果你需要使用https 或者 tls 协议, 你可以使用直接加入Rancher或者挂载在负载均衡容器中的证书。

引用在RANCHER中添加的证书

证书可以在负载均衡容器的lb_config中被引用。

```
version: '2'
services:
  lb:
    scale: 1
    lb_config:
      certs:
      - <certName>
      default_cert: <defaultCertName>
```

将证书挂载进负载均衡容器

仅仅在Compose文件中支持

证书可以作为卷直接挂载进负载均衡容器。证书需要按照特定的目录结构挂载入容器。如果你使用LetsEncrypt客户端生存证书，那么它就已经满足Rancher的要求。否则，你需要手动设置目录结构，使他与LetsEncrypt客户端生成的一致。

Rancher的负载均衡将会检测证书目录来实现更新。任何对证书的新增 / 删除操作都将每30秒同步一次。

所以的证书都位于同一个基础的证书目录下。这个文件名将会作为负载均衡服务的一个标签，用于通知负载均衡证书的所在地。

在这个基础目录下，相同域名的证书被放置在同一个子目录下。文件名就是证书的域名。并且每一个文件夹都需要包含privkey.pem和 fullchain.pem。对于默认证书，可以被放置在任意一个子目录名下，但是下面的文件命名规则必须保持一致。

```
-- certs
  |-- foo.com
  |   |-- privkey.pem
  |   |-- fullchain.pem
  |-- bar.com
  |   |-- privkey.pem
  |   |-- fullchain.pem
  |-- default_cert_dir_optional
  |   |-- privkey.pem
  |   |-- fullchain.pem
...
```

当启动一个负载均衡的时候，你必须用标签声明证书的路径（包括默认证书的路径）。这样以来，负载均衡将忽略设置在lb_config中的证书。

> 注意： 你不能同时使用在Rancher中添加的证书和挂载在负载均衡容器中的证书

```
labels:
  io.rancher.lb_service.cert_dir: <CERTIFICATE_LOCATION>
  io.rancher.lb_service.default_cert_dir: <DEFAULT_CERTIFICATE_LOCATION>
```

证书可以通过绑定主机的挂载目录或者通过命名卷来挂在入负载均衡容器，命名卷可以以我们的storage drivers为驱动。

EXAMPLE DOCKER-COMPOSE.YML

```
version: '2'
services:
  lb:
    image: rancher/lb-service-haproxy:<TAG_BASED_ON_RELEASE>
    volumes:
    - /location/on/hosts:/certs
    ports:
    - 8087:8087/tcp
    labels:
      io.rancher.container.agent.role: environmentAdmin
      io.rancher.container.create_agent: 'true'
      io.rancher.lb_service.cert_dir: /certs
      io.rancher.lb_service.default_cert_dir: /certs/default.com
  myapp:
    image: nginx:latest
    stdin_open: true
    tty: true
```

EXAMPLE RANCHER-COMPOSE.YML

```
version: '2'
services:
  lb:
    scale: 1
    start_on_create: true
    lb_config:
      certs: []
      port_rules:
      - priority: 1
        protocol: https
        service: myapp
        source_port: 8087
        target_port: 80
    health_check:
      healthy_threshold: 2
      response_timeout: 2000
      port: 42
      unhealthy_threshold: 3
      interval: 2000
      strategy: recreate
  myapp:
    scale: 1
    start_on_create: true
```

自定义配置

高阶用户可以在rancher-compose.yml中声明自定义的配置。点击HAProxy配置文档查看更多详情。

EXAMPLE RANCHER-COMPOSE.YML

```
version: '2'
services:
  lb:
    scale: 1
    lb_config:
      config: |-
        global
            maxconn 4096
            maxpipes 1024

        defaults
            log global
            mode    tcp
            option  tcplog

        frontend 80
            balance leastconn

        frontend 90
            balance roundrobin

        backend mystack_foo
            cookie my_cookie insert indirect nocache postonly
            server $$IP <server parameters>

        backend customUUID
  health_check:
    port: 42
    interval: 2000
    unhealthy_threshold: 3
    healthy_threshold: 2
    response_timeout: 2000
```

会话粘性策略

如果你需要使用会话粘性策略，你可以更新rancher-compose.yml中的策略。

EXAMPLE RANCHER-COMPOSE.YML

```
version: '2'
services:
  lb:
    scale: 1
    lb_config:
      stickiness_policy:
        name: <policyName>
        cookie: <cookieInfo>
        domain: <domainName>
        indirect: false
        nocache: false
        postonly: false
        mode: <mode>
  health_check:
    port: 42
    interval: 2000
    unhealthy_threshold: 3
    healthy_threshold: 2
    response_timeout: 2000
```

RANCHER COMPOSE EXAMPLES

LOAD BALANCER EXAMPLE (L7)

EXAMPLE DOCKER-COMPOSE.YML

```
version: '2'
services:
  web:
    image: nginx
  lb:
    image: rancher/lb-service-haproxy
  ports:
  - 80
  - 82
```

EXAMPLE RANCHER-COMPOSE.YML

```
version: '2'
services:
  lb:
    scale: 1
    lb_config:
      port_rules:
      - source_port: 80
        target_port: 8080
        service: web1
        hostname: app.example.com
        path: /foo
      - source_port: 82
        target_port: 8081
        service: web2
        hostname: app.example.com
        path: /foo/bar
  health_check:
    port: 42
    interval: 2000
    unhealthy_threshold: 3
    healthy_threshold: 2
    response_timeout: 2000
```

内部负载均衡例子

设置内部负载均衡不需要列举端口，但是你仍然可以设置端口规则来转发流量。

EXAMPLE DOCKER-COMPOSE.YML

```
version: '2'
services:
  lb:
    image: rancher/lb-service-haproxy
  web:
    image: nginx
```

EXAMPLE RANCHER-COMPOSE.YML

```
version: '2'
services:
  lb:
    scale: 1
    lb_config:
      port_rules:
      - source_port: 80
        target_port: 80
        service: web
    health_check:
      port: 42
      interval: 2000
      unhealthy_threshold: 3
      healthy_threshold: 2
      response_timeout: 2000
  web:
    scale: 1
```

SSL会话终止 EXAMPLE

在rancher-compose.yml中使用的证书必须被加入到Rancher中。

EXAMPLE DOCKER-COMPOSE.YML

```
version: '2'
services:
  lb:
    image: rancher/lb-service-haproxy
    ports:
    - 443
  web:
    image: nginx
```

EXAMPLE RANCHER-COMPOSE.YML

```
version: '2'
services:
  lb:
    scale: 1
    lb_config:
      certs:
      - <certName>
      default_cert: <defaultCertName>
      port_rules:
      - source_port: 443
        target_port: 443
        service: web
        protocol: https
  web:
    scale: 1
```

添加外部服务

你可能会有一些部署在Rancher之外的服务想要整合进Rancher。你可以通过添加一个外部服务的功能将它添加到Rancher集群中。

在UI上添加外部服务

在你的应用上，你可以通过 添加服务 旁边的下拉菜单按钮添加外部服务。选择 外部服务。 或者你在应用层级的页面查看你的应用，同样存在相同的 添加服务 下拉菜单。

你将需要提供一个外部服务的 名称，如果需要的话，提供这个服务的 描述。

添加你想要的目标。你可以选择外部的IP或者域名。最后点击 添加。

外部服务的IP和域名会在服务中呈现。和Rancher的服务一样，你需要去启动一个外部服务。

添加/删除外部服务目标

在任何时候你都可以编辑你外部服务中的服务目标。在外部服务的下拉菜单中点击 编辑，你可以添加或者移除目标。

使用RANCHER COMPOSE添加外部服务

在外部服务中，你可以设置外部IP地址 或者 域名。rancher/external-service 并不是一个真实的镜像，但在 docker-compose.yml 中是必要的。Rancher不会为外部服务创建容器。

DOCKER-COMPOSE.YML例子

```
version: '2'
services:
  db:
    image: rancher/external-service
  redis:
    image: redis
RANCHER-COMPOSE.YML 使用外部IP的例子
version: '2'
services:
  db:
    external_ips:
    - 1.1.1.1
    - 2.2.2.2

  # Override any service to become an external service
  redis:
    image: redis
    external_ips:
    - 1.1.1.1
    - 2.2.2.2
```

RANCHER-COMPOSE.YML 使用域名的例子

```
version: '2'
services:
  db:
    hostname: example.com
```

服务别名

通过添加一个服务别名，可以提供一种别名的方式而不是直接指向服务。

在UI上添加服务别名

在你的应用中，你通过 添加服务 旁边的下拉按钮，并点击服务别名去添加一个服务别名。同样的，如果你在应用层级页面，同样的 添加服务的下拉菜单也会在每个应用页面中。

你需要提供服务别名的 名称，以及填写必要的 描述。名称 将是你选择服务的服务别名。

选择一个或多个你想添加到别名的目标。可用目标列表是当前应用中已经创建的服务。最后点击 创建。

服务别名中生效的服务列表会在服务层级页面显示。和我们的服务一样，你需要启动这个服务别名才能生效。

添加/移除服务

在任何时候你都可以在服务别名中修改目标服务。在服务的下拉菜单中点击 编辑，你可以添加更多的服务到这个别名中，或者移除现有的服务。

通过RANCHER COMPOSE添加服务别名

一个服务别名创建了一个指向服务的指针。在以下的例子中，web[.stack-name.rancher.internal]会被解析为容器web1以及web2的IP地址。rancher/dns-service并不是一个真实的镜像，但是他需要填写在docker-compose.yaml。不会为别名服务创建额外的容器。

EXAMPLE DOCKER-COMPOSE.YML

```
version: '2'
services:
  web:
    image: rancher/dns-service
    links:
    - web1
    - web2

  web1:
    image: nginx

  web2:
    image: nginx
```

健康检查

Cattle环境中，Rancher通过运行一个叫healthcheck的基础设施服务部署了一套健康检查系统，其原理为在每台主机上部署了healthcheck的容器来实现分布式的健康检查。这些容器在内部利用HAProxy来检查应用的健康状态。一旦容器或服务上启用了健康检查，每个容器将最多被三个运行在不同主机上的healthcheck 容器监控起来。只要有一个HAProxy实例认为其状态正常，该容器将被视为正常。如果所有HAProxy实例都认为其状态不正常，该容器将被视为状态异常。

> 注意： 该模式下唯一的例外为你的环境中只有一台主机，这种情况下健康检查将在同一台主机上被执行。

Rancher利用了不同网络位置的主机进行健康检查，这种方式比基于客户端的健康检查更高效。利用HAProxy来进行健康检查，Rancher使用户可以在服务和负载均衡上使用相同的健康检查策略。

> 注意： 健康检查将只能在使用托管网络的服务上生效。如果你选择了其他的网络类型，该服务将不会被监控。

配置

可以通过如下选项来配置健康检查：

检查类型: 有两种检查方式 - TCP连接 (只验证端口是否打开) 以及 HTTP响应2xx/3xx (进行HTTP请求并确保收到正确的回复)。

HTTP 请求: 如果检查类型是 HTTP响应2xx/3xx, 你必须制定一个可以接受查询的URL。 你可以选择方法 (GET, POST, etc) 以及HTTP版本 (HTTP/1.0, HTTP/1.1).

端口: 需要进行检查的端口

初始化超时: 在退出初始化之前等待的毫秒数。

重新初始化超时: 在退出重新初始化之前等待的毫秒数。

检查间隔: 在每次检查之间的时间间隔（毫秒）。

检查超时: 在检查未得到回复并超时之前等待的毫秒数。

健康阈值: 在一个不健康容器被标记为健康之前需要收到的健康检查回复的次数。

不健康阈值: 在健康容器被标记为不健康之前需要收到的健康检查回复的次数。

不健康门槛: T 当容器被认为是不健康时，有3种选择。不进行操作意味着容器将保持不健康状态。重新创建意味着Rancher会破坏不健康的容器并为服务创建一个新的容器。 重新创建，仅当至少X个容器健康时意味着如果有X多个容器健康，不健康的容器将被破坏并重新创建。

在UI中添加HEALTH CHECKS

对于服务或负载均衡，可以通过导航到Health Check选项卡来添加Health check服务。你可以检查服务的TCP连接或HTTP响应，并更改health check配置的默认值。

通过RANCHER COMPOSE添加 HEALTH CHECKS

使用Rancher Compose，health checks能添加在rancher-compose.yml文件中。

在我们的示例中，如果容器发现不健康，我们会显示三种不同策略的健康检查配置。

```
version: '2'
services:
  service1:
    scale: 1
    health_check:
      # Which port to perform the check against
      port: 80
      # For TCP, request_line needs to be '' or not shown
      # TCP Example:
      # request_line: ''
      request_line: GET /healthcheck HTTP/1.0
      # Interval is measured in milliseconds
      interval: 2000
      initializing_timeout: 60000
      reinitializing_timeout: 60000
      unhealthy_threshold: 3
      # Strategy for what to do when unhealthy
      # In this service, no action will occur when a container is found unhealthy
      strategy: none
      healthy_threshold: 2
      # Response timeout is measured in milliseconds
      response_timeout: 2000
  service2:
    scale: 1
    health_check:
      # Which port to perform the check against
      port: 80
      # Interval is measured in milliseconds
      interval: 2000
      initializing_timeout: 60000
      unhealthy_threshold: 3
      # Strategy for what to do when unhealthy
      # In this service, Rancher will recreate any unhealthy containers
      strategy: recreate
      healthy_threshold: 2
      # Response timeout is measured in milliseconds
      response_timeout: 2000
  service3:
    scale: 1
    health_check:
      # Which port to perform the check against
      port: 80
      # Interval is measured in milliseconds
      interval: 2000
      initializing_timeout: 60000
      unhealthy_threshold: 3
      # Strategy for what to do when unhealthy
      # In this service, Rancher will recreate any healthy containers only if there   is at least 1 container
      # that is healthy
      strategy: recreateOnQuorum
      recreate_on_quorum_strategy_config:
        quorum: 1
      healthy_threshold: 2
      # Response timeout is measured in milliseconds
      response_timeout: 2000
```

故障情况

状况	响应
被监测的容器停止响应Health check。	所有的HAProxy实例将检测到故障，并将容器标记为“不健康”。如果容器是服务的一部分，则Rancher通过其服务HA功能将服务恢复到其预定义的规模。
运行启用来健康检查的容器的主机失去了网络连接或该主机上的代理失去了网络链接。	当主机丢失网络连接时，与Rancher服务连接的Rancher代理也会丢失网络连接。 由于代理不可访问，主机被标记为“重新连接”。这样我们知道了Rancher Server无法连接到该主机的主机代理。对Rancher的健康检查是针对容器本身而不是主机完成的; 因此，容器将无法被所有活动的HAProxy实例访问。如果容器是服务的一部分，则Rancher通过其服务HA功能将服务恢复到其预定义的规模。
运行启用了健康检查的容器的主机完全故障。	当主机遇到完全故障（如断电）时，与Rancher服务连接的Rancher代理也会丢失网络连接。 由于代理不可访问，主机被标记为“重新连接”。这样我们知道了Rancher Server无法连接到该主机的主机代理。对Rancher的健康检查是针对容器本身而不是主机完成的; 因此，容器将无法被所有活动的HAProxy实例访问。如果容器是服务的一部分，则Rancher通过其服务HA功能将服务恢复到其预定义的规模。
主机的代理失败，但主机保持在线，容器正在运行，并且正在进行健康检查。	在这种情况下，与Rancher服务连接的Rancher代理也会丢失网络连接。 由于代理不可访问，主机被标记为“重新连接”。这样我们知道了Rancher Server无法连接到该主机的主机代理。对Rancher的健康检查是针对容器本身而不是主机完成的; 因此，容器将无法被所有活动的HAProxy实例访问。如果容器是服务的一部分，则Rancher通过其服务HA功能将服务恢复到其预定义的规模。

根据health check的结果容器会被标记成绿色或红色状态， 根据健康检查的结果，会判断容器是处于绿色或红色状态。如果运行该服务的所有容器处于绿色状态，该服务就处于绿色（或“运行”）的状态。如果运行该服务所有容器都处于红色状态，则服务处于红色（或“停止”）状态。如果Rancher检测到至少一个容器是处于红色状态或正在要变为绿色状态，该服务会处于黄色（或“退化”）的状态。

检测故障所用的时间是通过“间隔”值进行控制的，该值是通过compose或UI创建健康检查时定义的。

> 注意： 故障恢复操作仅在容器状态变为绿色后执行。也就是说，如果服务的启动时间很长，则容器将不会立即重新启动，因为服务需要超过2000ms才能启动。健康检查首先需要在采取任何其他行动之前将容器变绿。

服务调度

在Rancher中，你可以根据严格或宽松的关联与斥关联规则，在特定主机上安排服务。 这些规则可以比较主机上的标签或主机上容器上的标签，以确定容器应该安排在哪个主机上。

默认情况下，Rancher将检测主机上的端口冲突，如果端口不可用，则不会将需要此端口的容器调度到这一主机上。

这个核心调度逻辑内置于Rancher，但Rancher还支持位于我们外部调度器中的其他调度能力，这是我们基础设施服务的一部分。其他调度能力包括：

- 多个IP的主机调度能力
- 基于资源约束的调度能力 (例如 CPU和内存)
- 能够限制在主机上安排哪些服务

标签和调度规则

不管是通过服务或者负载均衡来创建容器时，我们都提供了为容器创建标签的选项，并且可以安排你想要放置容器的主机。对于本节的剩余部分，我们将使用服务这个术语，但这些标签也适用于负载均衡器（即特定类型的服务）

调度规则提供了让Rancher选择要使用哪个主机的灵活性。 在Rancher中，我们使用标签来帮助定义调度规则。 你可以根据需要在容器上创建任意数量的标签。 通过多个调度规则，你可以完全控制容器在哪些主机上创建。 你可以要求在具有特定主机标签，容器标签或名称或特定服务的主机上启动该容器。 这些调度规则可以帮助你创建容器对主机的黑名单和白名单。

在UI中添加标签

对于添加服务，可以在标签选项卡中添加标签。 对于添加负载均衡，也可以在标签选项卡中找到添加标签。

通过向负载均衡添加标签，每个负载均衡容器将接收该标签，该标签是一个键值对。 在Rancher中，我们使用这些容器标签来帮助定义调度规则。 你可以根据需要在负载均衡上创建任意数量的标签。 默认情况下，Rancher已经在每个容器上添加了系统相关的标签。

在UI中调度选项

对于服务和负载均衡，标签可以在调度选项卡中找到。

对于 服务, 我们提供了两个选项来确定在哪里启动容器。

选项1：在指定主机上运行 全部 容器

通过选择此选项，容器/服务将在特定主机上启动。 如果你的主机掉线，则容器也将离线。 如果你从容器页面创建一个容器，即使有端口冲突，容器也将被启动。 如果创建一个数量大于1且服务端口冲突的服务，则你的服务可能会停留在 Activating 状态，直到你编辑正确的服务数量为止。

选项2：为每一个容器自动选择符合调度规则的主机

通过选择此选项，你可以灵活地选择调度规则。 遵循所有规则的任何主机都是可以启动容器的主机。 你可以通过点击 + 按钮添加规则。

对于负载均衡，只有选项2可用，因为端口冲突。 你只能添加调度规则。 点击 调度 选项卡。 你可以通过点击 添加调度规则 按钮添加任意数量的调度规则。

对于每个规则，你可以选择规则的条件。 有四种不同的条件，它们定义了遵守规则的严格程度。 字段确定要应用规则的字段。 键和值是要检查字段的值。 如果你启动了一个服务或负载均衡，Rancher将根据每个主机的负载来扩展容器在适用主机上的分发。 根据所选择的条件将确定适用的主机是什么。

> 注意： 对于添加服务/添加负载均衡，如果你在数量那里选择了 总是在每台主机上运行一个此容器的实例，则只有主机标签将显示为可能的字段。

条件

- 必须 或 不能：Rancher只会使用与字段和值匹配或不匹配的主机。 如果Rancher找不到符合这些条件的所有规则的主机，你的服务可能会停留在 Activating 状态。 该服务将不断尝试找到容器的主机。 要修复此状态，你可以编辑服务的数量或添加/编辑将满足所有这些规则的其他主机。
- 应该或不应该：Rancher将尝试使用匹配字段和值的主机。 在没有匹配所有规则的主机的情况下，Rancher将逐个删除软约束（应该/不应该规则），直到主机满足剩余的约束。

字段

- 主机标签：当选择要用于容器/服务的主机时，Rancher将检查主机上的标签，看它们是否与提供的键/值对匹配。 由于每个主机都可以有一个或多个标签，所以Rancher会将键/值对与主机上的所有标签进行比较。 将主机添加到Rancher时，可以向主机添加标签。 你还可以使用主机下拉菜单中的编辑选项来编辑主机上的标签。 活动主机上的标签列表可从关键字段的下拉列表中找到。
- 容器标签：选择此字段时，Rancher会查找已经具有与键/值对匹配的标签的容器的主机。 由于每个容器都可以有一个或多个标签，所以Rancher会将键/值对与主机中每个容器上的所有标签进行比较。 容器标签位于容器的标签选项中。 在容器启动后，你将无法编辑容器标签。 为了创建具有相同设置的新容器，你可以克隆容器或服务，并在启动之前添加标签。 运行容器上的用户标签列表可从关键字段的下拉列表中找到。
- 服务名称: Rancher将检查主机上是否有一个具有特定名称的服务。 如果在稍后的时间，该服务名称将更改或不活动/已删除，该规则将不再有效。 如果你选择此字段，则该值将需要以应用名称/服务名称的格式。 运行服务的列表可从值字段的下拉列表中获得。
- 容器名称: Rancher会检查一个主机是否有一个具有特定名称的容器。 如果稍后时间，容器有名称更改或不活动/已删除，该规则将不再有效。 运行容器的列表可从值字段的下拉列表中获得。

在RANCHER COMPOSE中添加标签

Rancher根据docker-compose.yml文件中定义的labels来决定如何安排一个服务的容器。 所有带有调度的标签都将在docker-compose.yml文件中使用。 Rancher通过3个主要组件定义了调度规则：条件，字段和值。 条件决定了Rancher遵守规则的严格程度。 字段是要比较的项目。 价值是你在字段上定义的。 在介绍一些例子之前，我们将广泛讨论这些组件。

调度条件

当我们编写我们的调度规则时，我们对每个规则都有条件，这说明了Rancher如何使用规则。亲和条件是当我们试图找到一个符合我们的价值的字段。反亲和条件是当我们试图找到一个不符合我们价值的字段时。

为了区分亲和力和反亲和度，我们在标签名称中添加_ne来表示标签是不符合字段和值。

规则也有硬条件和软条件。

一个硬条件相当于说必须或不能。 Rancher只会使用匹配或不匹配字段和值的主机。如果Rancher找不到符合这些条件的所有规则的主机，你的服务可能会停留在 Activating 状态。该服务将不断尝试找到容器的主机。要修复此状态，你可以编辑服务的数量或添加/编辑一台主机来满足所有这些规则。

一个软条件相当于应该或不应该。 Rancher将尝试使用与该字段和值相匹配的主机。在没有匹配所有规则的主机的情况下，Rancher将逐个删除软约束（应该/不应该规则），直到主机满足剩余的约束。

为了区分 must 和 should 条件，我们将“_soft”添加到我们的标签名称中，以表明标签是应该尝试匹配字段和值。

字段

Rancher能够与主机标签，容器标签，容器名称或服务名称的值进行比较。 标签前缀是Rancher用来定义哪个字段将被评估的用法。

字段	标签前缀
主机标签	io.rancher.scheduler.affinity:host_label
容器标签/服务名称	io.rancher.scheduler.affinity:container_label
容器名称	io.rancher.scheduler.affinity:container

请注意，服务名称中没有特定的前缀。 当Rancher创建服务时，会将系统标签添加到服务的所有容器中，以指示应用和服务名称。

当我们创建标签的关键字时，我们从一个字段前缀（例如io.rancher.scheduler.affinity：host_label）开始，根据我们正在寻找的条件，我们附加我们想要的条件类型。 例如，如果我们希望容器在不能等于（即_ne）主机标签值的主机上启动，则标签键将是io.rancher.scheduler.affinity：host_label_ne。

值

你可以使用这些值来定义要检查的字段。 如果你有两个值要与同一条件和字段进行比较，则需要为标签名称使用一个标签。 对于标签的值，你需要使用逗号分隔列表。 如果有多个具有相同键的标签（例如io.rancher.scheduler.affinity：host_label_ne），则Rancher将使用与标签键一起使用的最后一个值覆盖任何先前的值。

```
labels:
  io.rancher.scheduler.affinity:host_label: key1=value1,key2=value2
```

全局服务

将服务提供到全局服务中相当于在UI中的每个主机上选择总是在每台主机上运行一个此容器的实例。 这意味着将在环境中的任何主机上启动一个容器。 如果将新主机添加到环境中，并且主机满足全局服务的主机要求，则该服务将自动启动。

目前，全局服务只支持使用硬条件的主机标签字段。 这意味着只有在调度时才会遵守与主机标签相关的标签，并且必须或不能等于该值。 任何其他标签类型将被忽略。

例子 DOCKER-COMPOSE.YML

```
version: '2'
services:
  wordpress:
    labels:
      # 使wordpress成为全局服务
      io.rancher.scheduler.global: 'true'
      # 使wordpress只在具有key1 = value1标签的主机上运行容器
      io.rancher.scheduler.affinity:host_label: key1=value1
      # 使wordpress只在没有key2 = value2标签的主机上运行
      io.rancher.scheduler.affinity:host_label_ne: key2=value2
    image: wordpress
    links:
    - db: mysql
    stdin_open: true
```

使用主机标签查找主机

将主机添加到Rancher时，你可以添加主机标签。 调度服务时，你可以利用这些标签来创建规则来选择要部署服务的主机。

使用主机标签的示例

```
labels:
  # 主机必须有`key1 = value1`的标签
  io.rancher.scheduler.affinity:host_label: key1=value1
  # 主机不得有`key2 = value2`的标签
  io.rancher.scheduler.affinity:host_label_ne: key2=value2
  # 主机应该有`key3 = value3`的标签
  io.rancher.scheduler.affinity:host_label_soft: key3=value3
  # 主机应该没有`key4 = value4`的标签
  io.rancher.scheduler.affinity:host_label_soft_ne: key4=value4
```

自动创建的主机标签

Rancher会自动创建与主机的linux内核版本和Docker Engine版本相关的主机标签。

Key	Value	描述
io.rancher.host.linux_kernel_version	主机上的Linux内核版本 (例如3.19)	主机上运行的Linux内核的版本
io.rancher.host.docker_version	主机上的Docker版本（例如1.10.3）	主机上运行的Docker Engine版本
io.rancher.host.provider	云提供商信息	云提供商名称（目前仅适用于AWS）
io.rancher.host.region	云提供商区域	云提供商区域（目前仅适用于AWS）
io.rancher.host.zone	云提供商可用区	云提供商可用区（目前仅适用于AWS）

```
labels:
# 主机必须运行Docker版本1.10.3
io.rancher.scheduler.affinity:host_label: io.rancher.host.docker_version=1.10.3
# 主机必须不运行Docker 1.6版
io.rancher.scheduler.affinity:host_label_ne: io.rancher.host.docker_version=1.6
```

> 注意： Rancher不支持在具有>=特定版本的主机上调度容器的概念。 你可以使用主机调度规则来创建特定的白名单和黑名单，以确定你的服务是否需要特定版本的Docker Engine。

用容器标签查找主机

向Rancher添加容器或服务时，可以添加容器标签。 这些标签可以用于你希望规则与之进行比较的字段。 提醒：如果将全局服务设置为true，则无法使用。

> 注意： 如果容器标签有多个值，Rancher会查看主机上所有容器上的所有标签，以检查容器标签。 多个值不需要在主机上的同一容器上。

使用容器标签的示例

```
labels:
  # 主机必须有一个标签为`key1=value1`的容器
  io.rancher.scheduler.affinity:container_label: key1=value1
  # 主机不能有一个标签为`key2=value2`的容器
  io.rancher.scheduler.affinity:container_label_ne: key2=value2
  #主机应该有一个标签为`key3=value3`的容器
  io.rancher.scheduler.affinity:container_label_soft: key3=value3
  # 主机应该没有一个标签为`key4=value4`的容器
  io.rancher.scheduler.affinity:container_label_soft_ne: key4=value4
```

服务名称

当Rancher Compose启动服务的容器时，它也会自动创建多个容器标签。 因为检查一个特定的容器标签正在寻找一个key=value，所以我们可以使用这些系统标签作为我们规则的关键。 以下是在Rancher启动服务时在容器上创建的系统标签：

标签	值
io.rancher.stack.name	$${stack_name}
io.rancher.stack_service.name	$${stack_name}/$${service_name}

> 注意： 使用io.rancher.stack_service.name时，该值必须为应用名称/服务名称的格式。

宏$$ {stack_name}和$$ {service_name}也可以在任何其他标签中的docker-compose.yml文件中使用，并在服务启动时进行评估。

使用服务名称的示例

```
labels:
  # Host必须有一个服务名称为`value1`的容器
  io.rancher.scheduler.affinity:container_label: io.rancher.stack_service.name=stackname/servicename
```

查找具有容器名称的主机

向Rancher添加容器时，可以给每个容器一个名称。 你可以使用此名称作为希望规则进行比较的字段。 提醒：如果将全局服务设置为true，则无法使用。

使用容器名称的示例

```
labels:
  # 主机必须有一个名为`value1`的容器
  io.rancher.scheduler.affinity:container: value1
  # 主机不能有一个名称为`value2`的容器
  io.rancher.scheduler.affinity:container_ne: value2
  # 主机应该有一个名称为`value3`的容器
  io.rancher.scheduler.affinity:container_soft: value3
  # 主机应该没有一个名为`value4`的容器
  io.rancher.scheduler.affinity:container_soft_ne: value4
```

示例

示例1:

典型的调度策略可能是尝试在不同的可用主机之间部署服务的容器。 实现这一点的一个方法是使用反相关性规则来关联自身:

```
labels:
  io.rancher.scheduler.affinity:container_label_ne: io.rancher.stack_service.name=$${stack_name}/$${service_name}
```

由于这是一个很强的反相关性规则，如果比例大于可用主机数量，我们可能会遇到问题。 在这种情况下，我们可能需要使用软反相关性规则，以便调度程序仍然允许将容器部署到已经具有该容器的主机。 基本上，这是一个软规则，所以如果没有更好的选择存在，它可以被忽略。 yaml labels: io.rancher.scheduler.affinity:container_label_soft_ne: io.rancher.stack_service.name=$${stack_name}/$${service_name}

示例2:

另一个例子可能是将所有容器部署在同一个主机上，而不考虑哪个主机。 在这种情况下，可以使用对其自身的软亲合力。

```
labels:
  io.rancher.scheduler.affinity:container_label_soft: io.rancher.stack_service.name=$${stack_name}/$${service_name}
```

如果选择了自己的硬约束规则，则第一个容器的部署将失败，因为目前没有运行该服务的主机。

调度标签表

标签	值	描述
io.rancher.scheduler.global	true	将此服务指定为全局服务
io.rancher.scheduler.affinity:host_label	key1=value1,key2=value2, etc…	容器必须部署到具有标签key1=value1和key2 = value2的主机上
io.rancher.scheduler.affinity:host_label_soft	key1=value1,key2=value2	容器应该被部署到具有标签key1=value1和key2=value2”的主机
io.rancher.scheduler.affinity:host_label_ne	key1=value1,key2=value2	容器不能被部署到具有标签key1=value1或key2=value2的主机
io.rancher.scheduler.affinity:host_label_soft_ne	key1=value1,key2=value2	容器不应该被部署到具有标签key1=value1或key2=value2的主机
io.rancher.scheduler.affinity:container_label	key1=value1,key2=value2	容器必须部署到具有标签key1=value1和key2=value2的容器的主机上。 注意：这些标签不必在同一个容器上。 可以在同一主机内的不同容器上。
io.rancher.scheduler.affinity:container_label_soft	key1=value1,key2=value2	容器应该部署到具有运行标签key1=value1和key2=value2的主机上
io.rancher.scheduler.affinity:container_label_ne	key1=value1,key2=value2	容器不能部署到具有运行标签key1=value1或key2=value2的容器的主机上
io.rancher.scheduler.affinity:container_label_soft_ne	key1=value1,key2=value2	容器不应该被部署到具有标签key1=value1或key2=value2的容器的主机上
io.rancher.scheduler.affinity:container	container_name1,container_name2	容器必须部署到具有名称为container_name1和container_name2运行的容器的主机上
io.rancher.scheduler.affinity:container_soft	container_name1,container_name2	容器应该被部署到具有名称为container_name1和container_name2运行的容器的主机上
io.rancher.scheduler.affinity:container_ne	container_name1,container_name2	容器不能部署到具有名称为container_name1或container_name2运行的容器的主机上
io.rancher.scheduler.affinity:container_soft_ne	container_name1,container_name2	容器不应该被部署到具有容器名称为container_name1或container_name2运行的主机

升级服务

在部署服务之后, 你可能想要通过修改服务来升级应用。例如，镜像已经被更新了，你想要部署服务的新版本。由于Docker容器是不可变的，为了修改服务，你需要销毁旧的容器并部署新的容器。Rancher提供了两种升级服务的方法。推荐的方式是服务内升级，这种方式会停掉旧的容器并且在这个服务内启动新的容器。 RancherUI仅支持服务内升级。你也可以通过Rancher Compose命令行进行服务内升级。 另一种升级方式为替换式升级。这种升级方式会删除旧的服务并创建一个新的服务，只有通过Rancher Compose命令行才能进行替换式升级。

> 注意： 如果你是想对你的服务进行扩容，你可以修改服务页面的数量参数，或者也可以用过Rancher Compose命令行来进行服务扩容。 rancher-compose scale <服务名>=<新的容器数量>.

服务内升级

通过UI进行升级

任何服务都可以通过UI来进行升级。和部署服务类似，你之前选择的全部Docker参数都会被保留，你可以修改这些参数的值。额外还有一些升级专用的选项：

- 批量大小：服务中的容器升级会被分成几批，批量大小代表每次你想要停掉的旧容器数量和启动的新容器数量。例如一共有10个容器要升级，当批量大小为2时，每次会升级2个容器，停掉2个旧的，启动2个新的，分5批完成升级。
- 批量间隔：每次批量升级间隔的时间，例如10个容器，每次升级5个，批量时间为5秒。在完成5个容器升级后，在等待5秒后，会升级剩下的5个容器。
- 启动行为：默认情况下，旧的容器会先停止，然后再启动新的容器。你可以选择先启动新的容器，再停止旧的容器。

在配置好了新的服务参数和升级专用参数以后，点击升级。

当全部旧的容器被停掉，新的容器启动成功之后，服务将会变成 Upgraded 状态。在这个阶段，你应该去测试你的新服务来确保服务可以正常工作。然后，可以通过点击升级完成来完成升级。如果你的服务出现了异常，你可以点击回滚来回退到之前的服务。

通过RANCHER COMPOSE命令行进行服务升级

通过Rancher Compose命令行进行服务内升级时，现有服务会按照docker-compose.yml中的内容进行升级，服务中旧的容器会被删除。升级相关的参数需要传递给rancher-compose up命令。如果命令加上--upgrade参数，docker-compose.yml内定义的服务如果发生了改变，这个服务会被按照文件中的配置进行升级。和在UI上进行升级操作一样，服务内升级分为两个步骤，需要用户确认来完成升级或者执行回滚。

第一步： 进行升级

升级的时候，你可以通过docker-compose.yml文件升级整个应用栈或者升级应用内指定的服务。

```
# 升级应用栈内的全部有变化的服务
$ rancher-compose up --upgrade
# 升级应用栈内指定的服务 (例如仅升级service1和service2)
$ rancher-compose up --upgrade service1 service2
# 强制触发服务升级，即使在docker-comopse.yml文件中未被改变的服务，也将被升级。
$ rancher-compose up --force-upgrade
```

升级选项

选项	描述
--pull, -p	升级前，在每台运行该容器的主机上执行docker pull操作，尝试获取新镜像
-d	在后台运行升级
--upgrade, -u, --recreate	仅升级在docker-compose.yml中配置有变化的服务
--force-upgrade, --force-recreate	不论服务的配置是否有变化，全部进行升级
--confirm-upgrade, -c	升级完成后自动确认升级成功，删除旧的容器
--rollback, -r	回滚到之前的版本
--batch-size “2”	每批升级的容器个数
--interval “1000”	每批升级的时间间隔

拉取镜像

在升级时，你可能需要在部署容器之前执行docker pull，因为主机上可能已经有了该镜像的缓存。你可以通过--pull参数，在部署容器之前拉取最新的镜像。

```
# 在升级时，强制每台主机在部署容器之前，执行docker pull更新镜像
$ rancher-compose up --upgrade --pull
```

批量大小

在默认的情况下，每批升级服务的2个容器。你可以通过--batch-size参数来设置每批所更新的镜像数量。

```
# 升级服务时每次会启动3个新的容器
# 直到新的容器达到设置的数量
$ rancher-compose up --upgrade --batch-size 3
```

批量间隔

在默认的情况下，每次新的容器启动和旧的容器停止之间有2秒的时间间隔。你可以通过--interval来覆盖这个时间间隔，参数后面跟着的时间间隔是以毫秒为单位的。

```
# 将服务的升级间隔设置为30秒。
# service1和service2的新容器启动和他们的旧容器停止之间为30s
$ rancher-compose up --upgrade service1 service2 --interval "30000"
```

在停止旧的容器前启动新的容器

在默认的情况下，服务内升级会先停掉旧的容器，再启动新的容器。如果想要先启动新的容器，再停止旧的容器，你需要在rancher-compose.yml文件中写入如下内容。

```
version: '2'
services:
  myservice:
    upgrade_strategy:
      start_first: true
```

```
# 通过上面的rancher-compose.yml配置，会先启动myservice服务中的新容器，然后再停掉旧容器。
$ rancher-compose up --upgrade myservice
```

第二步：确认升级

在你验证该服务升级成了并且可以正常工作了之后，你需要在Rancher里确认升级成功。这种设计是因为有时候你可能想要回滚你的服务。当你点击完成升级后，就不能再进行回滚操作了

```
# 下面的命令可以确认升级成功，不需要在UI上点击完成升级。
$ rancher-compose up --upgrade --confirm-upgrade
```

回滚

在升级过程完成之后，你的服务可能发生了问题不能正常工作。Rancher支持回滚功能，可以把服务回滚到升级之前的状态。回滚操作只能在点击完成升级之前进行。

```
# 回滚到之前的版本
$ rancher-compose up --upgrade --rollback
```

替换式升级

替换式升级会通过创建一个新的服务来替换旧的服务，而不是在同一个服务内停止旧的容器并启动新的容器。只有Rancher Compose命令行才支持这种升级方式，UI上不可以。替换式升级操作非常简单：

```
$ rancher-compose upgrade service1 service2
```

service2是你想要在Rancher里启动的新服务的名字。service1是你想要在Rancher里停止并替换的服务。当service2被部署之后，service1里面的容器会被删除，但是服务本身并不会从Rancher里删除，只是容器的数量会变为0。

这两个服务名都需要被定义在docker-compose.yml里面。service1只需要把服务名在yml文件中定义即可，Rancher Compose会用这个名字来找到相应的服务。service2则需要在yml文件中定义全部需要的配置，Rancher Compose命令行会根据文件中的配置来部署service2。

示例 DOCKER-COMPOSE.YML

```
version: '2'
services:
  service1:
  # 这里不需要额外的参数，因为service1已经在运行中了。
  service2:
    image: wordpress
    links:
    - db:mysql
```

默认情况下，全部的指向service1的负载均衡或者是服务连接都会被自动更新并指向service2。如果你不想创建这些连接，你可以通过设置来禁止连接创建。

> 注意： 升级服务时并不需要rancher-compose.yml文件。 在默认情况下，新服务中的容器数量和旧服务中的容器数量相同。你可以通过传递--scale参数来设置容器的数量。

升级过程中的容器数量

直到新的服务中的容器数量和旧的服务中的容器数量达到你设定的数量时，旧服务中的容器才会被删除。

示例:

```
$ rancher-compose upgrade service1 service2 --scale 5
```

service1中有两个容器，你想要把它升级为service2，service2中想要启动5个容器。

service1	service2	备注
2	0	service1在运行中，并且有两个容器。
2	2	service2每批启动两个容器(默认批量数量)。
2	4	service2第二批时，再启动两个容器。
1	4	在上一步，新旧容器的数量总和为6个，已经超过目标数量的5个。这时会停掉service1里的一个容器，使新旧容器数量为5个。
1	5	service2再启动一个容器，达到目标的5个容器。
0	5	service1删除最后一个容器。

UPGRADE命令的参数

upgrade命令，可以接受几个参数来自定义升级行为。

参数	描述
--batch-size “2”	每批升级的容器个数
--scale “-1”	最终要运行的容器数量
--interval “2000”	每批升级的时间间隔
--update-links	更新指定服务的连接
--wait, -w	等待升级完成
--pull, -p	升级前，在每台运行该容器的主机上执行docker pull操作，尝试获取新镜像
--cleanup, -c	在升级完毕后，删除旧的服务

批量大小

在默认情况下，升级的时候每次启动2个新服务的容器。你可以通过--batch-size参数来设置每批启动的容器个数。

```
# 升级的过程中，每批将会启动3个service2的容器，直到service2的容器数量达到设定的值。
$ rancher-compose upgrade service1 service2 --batch-size 3
```

最终数量

在默认情况下，新服务的容器数量和旧服务之前运行的容器数量相同。你可以通过传递--scale参数，来设置新服务所运行容器的数量。

```
#设置service2升级后的容器数量为8个
$ rancher-compose upgrade service1 service2 --scale 8
```

> 注意： 旧服务中的容器并不根据批量大小的数量来删除。当新旧服务中的容器数量和，达到设置的最终数量时，旧服务中的容器将会被停止并删除。

批量间隔

在默认的情况下，每次新的容器启动和旧的容器停止之间有2秒的时间间隔。你可以通过--interval来覆盖这个时间间隔，参数后面跟着的时间间隔是以毫秒为单位的。

```
# 将服务的升级间隔设置为30秒。
# service1和service2的新容器启动和他们的旧容器停止之间为30s
$ rancher-compose upgrade service1 service2 --interval "30000"
```

更新连接

在默认情况下，全部指向旧服务的连接会被设置到新服务上。如果你不想让那些服务连接到新的服务的话，你可以通过--update-links="false"参数来禁止这些连接的创建。

```
# 不把指向service1中的连接设置到service2上
$ rancher-compose upgrade service1 service2 --update-links="false"
```

等待升级完毕

在默认情况下，Rancher Compose命令行在向Rancher发出升级命令后就会立刻退出。退出的时候升级可能还没执行完毕。通过传递--wait或者-w到upgrade命令，Rancher Compose命令行将在旧的容器被停止且新容器启动后才退出。

```
#等待升级完成
$ rancher-compose upgrade service1 service2 --wait
```

拉取新镜像

在升级时，你可能想在部署容器之前执行docker pull，因为主机上可能已经有了该镜像的缓存。你可以通过--pull或者-p参数，在部署容器之前拉取最新的镜像。

```
# 在启动容器时，先执行docker pull获取最新镜像
$ rancher-compose upgrade service1 service2 --pull
```

清除旧的服务 在默认情况下，升级完成后旧的服务不会被删除，只是旧服务中容器的数量为0。如果你觉得并不需要回滚，也不需要保留旧的服务配置。你可以通过传递--cleanup或者-c参数到upgrade命令。这个参数会同时应用--wait参数，因为Rancher Compose命令行需要等待升级完成，然后才能删掉旧的服务。

```
# 升级完成后删除service1
$ rancher-compose upgrade service1 service2 --cleanup
```

Rancher Cattle环境中的内部DNS服务

在Rancher中，我们拥有自己的内部DNS服务，允许同一个环境中的任何服务都可以解析环境中的任何其他服务.

应用中的所有服务都可以通过<服务名称>解析，并且不需要在服务之间设置服务链接。 创建服务时，你可以定义服务链接以将服务链接在一起。 对于任何不同应用的服务，你可以通过<服务名称>.<应用名称>而不是<服务名称>来解析。 如果你想以不同的名称解析服务，你可以设置服务链接，以便服务可以由服务别名解析.

通过链接设置服务别名

在UI中，添加服务时，展开服务链接部分，选择服务，并提供别名.

如果你使用Rancher Compose添加服务，docker-compose.yml将使用links或external_links指令.

```
version: '2'
services:
  service1:
    image: wordpress
    # 如果其他服务在同一个应用中
    links:
    # <service_name>:<service_alias>
    - service2:mysql
    # 如果另一个服务是不同的应用
    external_links:
    # <stackname>/<service_name>:<service_alias>
    - Default/service3:mysql
```

从容器和服务连接

在启动服务时，你可能需要指定只在同一台主机上一起启动服务。 具体的用例包括尝试使用另一个服务中的volume_from或net时。 当添加一个从容器时，这些服务可以通过他们的名字自动地相互解析。 我们目前不支持通过从容器中的links/external_links来创建服务别名。

当添加一个从容器时，总是有一个主服务和从容器。 它们一起被认为是单个启动配置。 此启动配置将作为一组容器部署到主机上，1个来自主服务器，另一个从每个从容器中定义。 在启动配置的任何服务中，你可以按其名称解析主服务和从容器。 对于启动配置之外的任何服务，主服务可以通过名称解析，但是从容器只能通过<从容器名称>.<主服务名称>来解析。

容器名称

所有容器都可以通过其名称来全局解析，因为每个服务的容器名称在每个环境中都是唯一的。 没有必要附加服务名称或应用名称。

例子

PING同一应用中的服务

如果你执行一个容器的命令行，你可以通过服务名称ping同一应用中的其他服务.

在我们的例子中，有一个名为stackA的应用，有两个服务，foo和bar.

在执行foo服务中的一个容器之后，你可以ping通bar服务.

```
$ ping bar
PING bar.stacka.rancher.internal (10.42.x.x) 58(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.63 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.13 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=1.07 ms
```

PING不同应用中的服务

对于不同应用的服务，你可以使用<服务名称>.<应用名称>在不同的应用中ping服务.

在这个例子中，我们有一个名为stackA的应用，它包含一个名为foo的服务，我们有另一个名为stackB的应用，它包含一个名为bar的服务.

如果我们执行foo服务中的一个容器，你可以用bar.stackb来ping.

```
$ ping bar.stackb
PING bar.stackb (10.42.x.x) 56(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.43 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.15 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=1.27 ms
```

PING服务中的从容器

取决于你从哪个服务ping，你可以通过<从容器名称>或<从容器名称>.<主服务名称>来访问从容器服务。

在我们的例子中，我们有一个名为stackA的应用，它包含一个名为foo的服务，它有一个从容器bar和一个名为hello的服务。 我们也有一个应用叫stackB，它包含一个服务world。

如果我们执行foo服务中的一个容器，你可以直接用`bar’命令ping它。

```
# 在`foo`服务中的一个容器中，`bar`是一个从容器。
$ ping bar
PING bar.foo.stacka.rancher.internal (10.42.x.x) 56(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=64 time=0.060 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=64 time=0.111 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=64 time=0.114 ms
```

如果我们执行在同一个应用中的hello服务的一个容器，你可以通过foo来pingfoo服务和bar.foo来pingbar服务.

```
# 在`hello`服务中的一个容器内部，这不是服务/从容器的一部分
# Ping主服务(i.e. foo)
$ ping foo
PING foo.stacka.rancher.internal (10.42.x.x) 56(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.04 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.40 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=1.07 ms
# Ping从容器(i.e. bar)
$ ping bar.foo
PING bar.foo (10.42.x.x) 56(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.01 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.12 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=1.05 ms
```

如果我们执行world服务中的一个容器，它是不同的应用，你可以通过foo.stacka来pingfoo服务和bar.foo.stacka来ping从容器bar.

```
# 在`world`服务中的一个容器内，它们位于不同的应用中
# Ping另一个应用`stacka`中的主服务(i.e. foo)
$ ping foo.stacka
PING foo.stacka (10.42.x.x) 56(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.13 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.05 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=1.29 ms
# Ping另一个应用`stacka`中的从容器(i.e. bar)
$ ping bar.foo.stacka
PING bar.foo.stacka (10.42.x.x) 56(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.23 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.00 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=0.994 ms
```

PING容器名称

从同一个环境下的任何一个容器中，无论它们是否在相同的应用或服务中，你都可以用容器名称来ping其他容器，

在我们的示例中，我们有一个名为stackA的应用，它包含一个名为foo的服务。 我们还有另一个应用叫stackB，它包含一个名为“bar”的服务。 容器的名称是 `<stack_name>-<service_name>-<number>` 。

如果我们执行foo服务中的一个容器，你可以通过ping来访问bar服务中的相关容器.

```
$ ping stackB-bar-1
PING stackB-bar-1.rancher.internal (10.42.x.x): 56 data bytes
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.994 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.090 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=1.100 ms
```

外部DNS服务

在应用商店中，Rancher提供了多种的DNS服务并且这些服务可以监听rancher-metadata的事件，并根据metadata的更变生成DNS记录。我们会以Route53作为例子说明外部DNS是如何工作的，但Rancher还有其他由其他DNS服务商提的供社区版服务。

最佳实践

- 在每个你启动的Rancher环境中，应该有且只有1个 route53 服务的实例。
- 多个Rancher实例不应该共享同一个 hosted zone。

需要配置AWS IAM权限

下面的IAM权限是Route53 DNS所需要的最小权限。 请确保你设置的主机AWS安全密钥(Access Key ID / Secret Access Key)或者主机IAM权限至少被配置了如下权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "route53:GetHostedZone",
                "route53:GetHostedZoneCount",
                "route53:ListHostedZonesByName",
                "route53:ListResourceRecordSets"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "route53:ChangeResourceRecordSets"
            ],
            "Resource": [
                "arn:aws:route53:::hostedzone/<HOSTED_ZONE_ID>"
            ]
        }
    ]
}
```

> 注意： 当使用这个JSON文档在AWS中创建自定义IAM规则时，请使用Route53所在区域或者通配符(‘*‘)来替换。

启动 ROUTE53 服务

在 应用商店 标签页中，你可以选择 Route53 DNS Stack。

为你的应用栈填写 名称，并填写必要的 描述。

在 配置选项 中，你需要提供一下的信息：

名称	值
AWS Access Key	访问AWS API的Access key
AWS Secret Key	访问AWS API的Secret key
AWS Region	在AWS中的区域名称。我们建议你填写一个与你服务器相近的区域。他会转换成Rancher Route53发送DNS更新请求的地址。
Hosted Zone	Route53 hosted zone。这个必须在你的Route53实例上预创建。

在完成表单后，点击 创建。一个带有 route53 服务的应用栈将会被创建，你只需要启动这个服务。

使用ROUTE53的服务

route53服务只会为在Host上映射端口的服务生成DNS记录，每一个Rancher生成的DNS记录，他使用一下的格式为服务创建一个fqdn：

```
fqdn=<serviceName>.<stackName>.<environmentName>.<yourHostedZoneName>
```

在AWS的Route 53中，他会以name=fqdn，value=[服务所在的host的ip地址列表]的一个记录集合呈现。Rancher route53 服务只会管理以.结尾的记录集合。当前TTL的时间为300秒。

一旦DNS记录被设置在AWS的Route 53上，生成的fqdn会返回到Rancher，并会设置 服务.fqdn 字段。你可以在服务下拉菜单的在API中查看 并查询 fqdn 找到fqdn字段。

当在浏览器使用fqdn，他会被指向到服务中的其中一个容器。如果服务中容器的IP地址发生了变化，这些变化会在AWS的Route 53服务同步更新。由于用户一直在使用fqdn，所有这些更变对用户是透明的。

> 注意： 在 route53 服务被启动后，任何已经部署并使用了host端口的服务都会获得一个fqdn。

删除 ROUTE53 服务

当 route53 从Rancher被移除，在AWS Route 53服务中的记录并不会被移除。这些记录需要在AWS中手工移除。

为外部DNS使用特定的IP

在默认下，Rancher DNS选择注册在Rancher Server中的主机IP去暴露服务。其中会有一个应用场景是主机在一个私有网络中，但主机将需要使用外部DNS在公网中暴露服务。你需要在启动外部DNS服务前添加一个主机标签，来指定在外部DNS中使用的IP地址。

在启动外部服务前，需要添加以下标签到主机上。标签的值需要是Rancher的Route53 DNS服务上要用到的IP。如果这个标签没有设置在主机上，Rancher的Route53服务将会默认使用主机注册在Rancher上的IP的地址。

```
io.rancher.host.external_dns_ip=<IP_TO_BE_USED_FOR_EXTERNAL_DNS>
```

Rancher Compose

Rancher Compose是一个多主机版本的Docker Compose。它运行于Rancher UI里属于一个环境多个主机的应用里。Rancher Compose启动的容器会被部署在满足调度规则的同一环境中的任意主机里。如果没有设置调度规则，那么这些服务容器会被调度至最少容器运行的主机上运行。这些被Rancher Compose启动的容器的运行效果是和在UI上启动的效果是一致的.

Rancher Compose工具的工作方式是跟Docker Compose的工作方式是相似的，并且兼容版本V1和V2的 docker-compose.yml 文件。为了启用Rancher的特性，你需要额外一份rancher-compose.yml文件，这份文件扩展并覆盖了docker-compose.yml文件。例如，服务缩放和健康检查这些功能就会在rancher-compose.yml中体现。

在阅读这份Rancher Compose文档之前，我们希望你已经懂得 Docker Compose 了。如果你还不认识 Docker Compose，请先阅读 Docker Compose文档。

安装

Rancher Compose的可执行文件下载链接可以在UI的右下角中找到，我们为你提供了Windows, Mac 以及 Linux 版本供你使用。

另外，你也可以到Rancher Compose的发布页找到可执行二进制文件的下载链接。

为 RANCHER COMPOSE 设置 RANCHER SERVER

为了让Rancher Compose可以在Rancher实例中启动服务，你需要设置一些环境变量或者在Rancher Compose命令中送一些参数。必要的环境变量分别是 RANCHER_URL, RANCHER_ACCESS_KEY, 以及 RANCHER_SECRET_KEY。 access key和secret key是一个环境API Keys, 可以在API -> 高级选项菜单中创建得到。

> 注意： 默认情况下，在API菜单下创建的是账号API Keys, 所以你需要在高级选项中创建环境API Keys.

```
# Set the url that Rancher is on
$ export RANCHER_URL=http://server_ip:8080/
# Set the access key, i.e. username
$ export RANCHER_ACCESS_KEY=<username_of_environment_api_key>
# Set the secret key, i.e. password
$ export RANCHER_SECRET_KEY=<password_of_environment_api_key>
```

如果你不想设置环境变量，那么你需要在Rancher Compose 命令中手动送入这些变量：

```
$ rancher-compose --url http://server_ip:8080 --access-key <username_of_environment_api_key> --secret-key <password_of_environ
ment_api_key> up
```

现在你可以使用Rancher Compose 配合docker-compose.yml文件来启动服务了。这些服务会在环境API keys对应的环境中启动服务的。

就像Docker Compose，你可以在命令后面加上服务名称来选择启动全部或者仅启动指定某些docker-compose.yml中服务

```
$ rancher-compose up servicename1 servicename2
$ rancher-compose stop servicename2
```

调试 RANCHER COMPOSE

你可以设置环境变量RANCHER_CLIENT_DEBUG的值为true来让Rancher Compose输出所有被执行的CLI命令。

```
# Print verbose messages for all CLI calls
$ export RANCHER_CLIENT_DEBUG=true
```

如果你不需要所有的 CLI 命令信息，你可以在命令后上--debug来指定输出哪些可视化CLI命令。

```
$ rancher-compose --debug up -d
```

删除服务或容器

在缺省情况下，Rancher Compose不会删除任何服务或者容器。这意味着如果你在一行命令里执行两次 up 命令，那么第二个 up 命令不会起任何作用。这是因为第一个 up 命令会创建出所有东西后让他们自己运行。即使你没有在 up 中使用 -d 参数，Rancher Compose 也不会删除你任何服务。为了删除服务，你只能使用 rm 命令。

构建

构建docker镜像可以有两种方法。第一种方法是通过给build命令一个git或者http URL参数来利用远程资源构建，另一种方法则是让 build 利用本地目录，那么会上传构建上下文到 S3 并在需要时在各个节点执行

为了可以基于S3来创建，你需要设置 AWS 认证。我们提供了一个说明怎样利用在Rancher Compose 里使用S3详细例子供你参考

指令与参数

Rancher Compose 工具的工作方式是跟 Docker Compose 的工作方式是相似的，并且支持版本V1的 docker-compose.yml 文件。为了启用 Rancher 的特性，你需要额外一份rancher-compose.yml文件，这份文件扩展并覆盖了docker-compose.yml文件。例如，服务缩放和健康检查这些特性就会在rancher-compose.yml中体现。

RANCHER-COMPOSE 命令

Rancher Compose 支持所有 Docker Compose 支持的命令。

Name	Description
create	创建所有服务但不启动
up	启动所有服务
start	启动服务
logs	输出服务日志
restart	重启服务
stop, down	停止服务
scale	缩放服务
rm	删除服务
pull	拉取所有服务的镜像
upgrade	服务之间进行滚动升级
help, h	输出命令列表或者指定命令的帮助列表

RANCHER COMPOSE 选项

无论何时你使用 Rancher Compose 命令，这些不同的选项你都可以使用

Name	Description
--verbose, --debug	-
--file, -f [–file option –file option]	指定一个compose 文件 (默认: docker-compose.yml) [$COMPOSE_FILE]
--project-name, -p	指定一个项目名称 (默认: directory name)
--url	执行 Rancher API接口 URL [$RANCHER_URL]
--access-key	指定 Rancher API access key [$RANCHER_ACCESS_KEY]
--secret-key	指定 Rancher API secret key [$RANCHER_SECRET_KEY]
--rancher-file, -r	指定一个 Rancher Compose 文件 (默认: rancher-compose.yml)
--env-file, -e	指定一个环境变量配置文件
--help, -h	输出帮助文本
--version, -v	输出 Rancher Compose 版本

例子

准备开始后，你需要创建一个 docker-compose.yml 文件和一个可选的 rancher-compose.yml 文件，如果没有 rancher-compose.yml 文件，那么所有服务默认只分配1个容器

样例文件 DOCKER-COMPOSE.YML

```
version: '2'
services:
  web:
    image: nginx
  db:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: test
```

样例文件 RANCHER-COMPOSE.YML

```
# Reference the service that you want to extend
version: '2'
services:
  web:
    scale: 2
  db:
    scale: 1
```

当你的这些文件创建好后，你就可以启动这些服务到 Rancher 服务了

```
# Creating and starting services without environment variables and selecting a stack
# If the stack does not exist in Rancher, it will be created in Rancher
$ rancher-compose --url URL_of_Rancher --access-key <username_of_environment_api_key> --secret-key <password_of_environment_api_key> -p stack1 up

# Creating and starting services with environment variables already set
$ rancher-compose -p stack1 up

# To change the scale of an existing service
$ rancher-compose -p stack1 scale web=3

# To launch a specific service in the docker-compose.yml
$ rancher-compose -p stack1 up web
```

> 注意： 如果你没有传入 -p ，应用名就是你执行Rancher Compose命令所在的文件夹名称。

使用 --ENV-FILE 选项

当你运行 Rancher Compose 命令时，可以使用--env-file 选项传入一个环境变量配置文件。

样例 SECRETS 文件

```
MYSQL_ROOT_PASSWORD=test
```

样例文件 DOCKER-COMPOSE.YML

```
version: '2'
services:
  db:
    image: mysql
    environment:
    # Just like Docker Compose, if there is only a key, Rancher Compose will resolve to
    # the values on the machine or the file passed in using --env-file
      MYSQL_ROOT_PASSWORD:
```

你可以启动服务时传入 secrets 文件

```
$ rancher-compose --env-file secrets up -d
```

在传入一个文件并一个环境变量只含一个key，Rancher Compose 将从这个文件或者从运行 Rancher Compose 命令的机器中的系统环境变量中提取这个值。当在文件和系统环境变量中同时存在同一个变量时，Rancher Compose 使用文件中的值。

命令选项

UP命令

Name	Description
--pull, -p	升级前先在各个已有这个镜像的主机拉取最新镜像
-d	不要阻塞或输出日志
--upgrade, -u, --recreate	当服务改变时升级
--force-upgrade, --force-recreate	强制升级服务，不管服务是否改变
--confirm-upgrade, -c	确认升级成功并删除老容器
--rollback, -r	回滚到上一个已部署的版本
--batch-size "2"	每次升级多少个容器
--interval "1000"	升级间隔

当你运行 Rancher Compose 的 up 命令时，在所有任务完成后进程会继续运行。如果你希望任务完成后进程退出，那么你需要传入 -d 选项，防止阻塞和输出日志。

```
# If you do not use the -d flag, Rancher Compose will continue to run until you Ctrl+C to quit
$ rancher-compose up

# Use the -d flag for rancher-compose to exit after running
$ rancher-compose up -d
```

阅读更多关于 利用Rancher Compose升级服务.

START命令

Name	Description
-d	防止阻塞或输出日志

如果你希望任务完成后进程退出，那么你需要传入 -d 选项，防止阻塞和输出日志。

LOGS命令

Name	Description
--follow	持续输出日志

RESTART命令

Name	Description
--batch-size "1"	每次重启多少个容器
--interval "0"	重启间隔

缺省情况下，Rancher Compose 会顺序地逐个重启服务。你可以设置批量大小和重启间隔。

STOP与SCALE

Name	Description
--timeout, -t "10"	指定停止超时秒数

```
# To change the scale of an existing service
$ rancher-compose -p stack1 scale service1=3
```

RM 命令

Name	Description
--force, -f	允许删除所有服务
-v	同时移除关联的容易

当移除服务时，Rancher Compose 仅移除在 docker-compose.yml 文件中出现的服务。如果有其他的服务在Rancher 的 stack 里，他们不会被移除，因为 Rancher Compose 不知道他们的存在。

所以 stack 不会被移除，因为 Rancher Compose 不知道stack 里是否还有其他容器。

缺省情况下，附加到容器的卷不会被移除。你可以通过 docker volume ls 查看所有的卷。

PULL命令

Name	Description
--cached, -c	只更新存在该镜像缓存的主机，不要拉取新的

```
# Pulls new images for all services located in the docker-compose.yml file on ALL hosts in the environment
$ rancher-compose pull

# Pulls new images for all services located in docker-compose.yml file on hosts that already have the image
$ rancher-compose pull --cached
```

> 注意： 不同于 docker-compose pull, 你不可以指定拉取哪些服务的镜像，Rancher Compose 会拉取所有在 docker-compose.yml 里的服务镜像。

UPGRADE 命令

你可以使用 Rancher Compose 升级在 Rancher 里的服务。请阅读更多关于在何时和怎样更新你的服务.

删除服务 / 容器

默认情况下，Rancher Compose 不会删除任何东西。 这意味着如果你在一行里有两个 up 命令，第二个 up 是不会做任何事情的。这是因为第一个 up 会创建所有东西并保持运行。甚至你没有传 -d 给 up，Rancher Compose 也不会删除你的服务。要删除服务，你只能使用 rm 。

环境插值

在使用 Rancher Compose 时，docker-compose.yml 和 rancher-compose.yml 文件中可以使用运行 Rancher Compose 的机器中的环境变量。 这个特性只在 Rancher Compose 命令中有效，在 Rancher UI 中是没有这个特性的。

怎么使用

在 docker-compose.yml 和 rancher-compose.yml 文件中，你可以引用你机器中的环境变量。如果没有该环境变量，它的值会被替换为空字符串，请注意的是 Rancher Compose 不会自动去除 : 两侧的空字符来适配相近的镜像。例如 `<imagename>` : 是一个非法的镜像名称，不能部署出容器。它需要用户自己来保证环境变量在该机器上是存在并有效的。

例子

在我们运行 Rancher Compose 的机器上有一个这样的环境变量，IMAGE_TAG=14.04 。

```
# Image tag is set as environment variable
$ env | grep IMAGE
IMAGE_TAG=14.04
# Run Rancher Compose
$ rancher-compose up
```

样例文件 docker-compose.yml

```
version: '2'
services:
  ubuntu:
    tty: true
    image: ubuntu:$IMAGE_TAG
    stdin_open: true
```

在 Rancher 里，一个 ubuntu 服务会使用镜像 ubuntu:14.04 部署。

环境插值格式

Rancher Compose 支持和 Docker Compose 一样的格式。

```
version: '2'
services:
  web:
    # unbracketed name
    image: "$IMAGE"

    # bracketed name
    command: "${COMMAND}"

    # array element
    ports:
    - "${HOST_PORT}:8000"

    # dictionary item value
    labels:
      mylabel: "${LABEL_VALUE}"

    # unset value - this will expand to "host-"
    hostname: "host-${UNSET_VALUE}"

    # escaped interpolation - this will expand to "${ESCAPED}"
    command: "$${ESCAPED}"
```

利用 AWS S3 构建

构建 docker 镜像可以有两种方法。第一种方法是通过给 build 命令一个 git 或者 http URL参数来利用远程资源构建，另一种方法则是让 build 利用本地目录，那么会上传构建上下文到 S3 并在需要时在各个节点执行

前置条件

- Docker
- Rancher Compose
- AWS 账户
- Rancher Server 和1台主机

在我们这个例子里，我们会在docker-compose.yml里定义我们的应用，并且把这个文件放在composetest下。这个compose文件会定义个web服务，它会打开5000端口并映射到主机上，还会链接redis服务，这样可以让在web中运行的服务可以通过redis这个主机名来访问redis容器

```
version: '2'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    links:
      - redis

  redis:
    image: redis
```

我们还会添加一个 rancher-compose.yml 到同一个 composetest 目录下来使用 Rancher的缩放能力。缺省情况下，如果没有rancher-compose.yml文件或者服务在rancher-compose.yml中没有定义，那么容器数量默认为1个。

```
version: '2'
services:
  web:
    scale: 3
```

当提供给 Rancher Compose 的这些文件准备好后，下一步就是实现这个程序并按照步骤来构建它。

使用docker-compose文档中的例子，我们会创建一个名为app.py的文件。这个应用会访问一个名为redis的主机，这个主机会运行 redis KV 存储服务。它会递增redis 中的键为 hits 的键值，然后取回这个值。

```
from flask import Flask
from redis import Redis

app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    redis.incr('hits')
    return 'Hello World! I have been seen %s times.' % redis.get('hits')

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

这个应用会依赖两个库，所以我们同时会创建一个名为 requirements.txt 的文件。

```
flask
redis
```

现在我们会在Dockerfile文件中定义应用的构建步骤。在Dockerfile文件里的指令会定义出要怎么构建出这个应用容器。

```
FROM python:2.7
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD python app.py
```

因为你已经运行着Rancher Server了，所以你只需要配置好你 AWS 认证信息，然后用你的 Rancher Server URL 和API key来运行 Rancher Compose 。

```
# Set up your AWS credentials
$ aws configure
AWS Access Key ID []: AWS_ACCESS_KEY
AWS Secret Access Key []: AWS_SECRET_KEY
Default region name []: NOT_NEEDED_FOR_S3
Default output format [None]:
# Run rancher-compose in your composetest directory where all the files are created
$ rancher-compose --url URL_of_Rancher --access-key username_of_API_key --secret-key password_of_API_key up
```

根据上面的指令，这个 web 容器会在一台 Rancher Server 管理的主机上运行起来。rancher-compose 会先上传当前目录到 S3，而你可以到 S3的 UI 上检索到这个目录。当镜像上传成功后，它会下载这个些文件到主机上构建起一个容器。

问题解答

如果你在利用S3构建时出现了一些问题，你可以先在本机测试一下是否可以构建并运行。在你运行rancher-compose的同一目录下，使用下面的命令来校验是否在 docker 中可以正常工作。

```
# Test building locally to see if works
$ docker build -t test .
# Test running the newly built image
$ docker run test
```

密文 - 实验性的

Rancher支持创建密文并在容器中使用该密文（在容器中使用该密文需要部署应用商店里的Rancher Secrets服务）。Rancher通过对接加密后台来保障密文的安全。加密后台可以使用本地的AES密钥或者使用Vault Transit

加密后台设置

默认情况下，Rancher Server会使用本地的AES256密钥来对密文进行加密。加密的密文存储在MySQL数据库里。

使用VAULT TRANSIT

如果不想使用本地密钥加密，你可以通过配置Vault Transit来进行密文加密。

在RANCHER中配置VAULT TRANSIT

在安装Rancher Server之前，需要进行如下Vault Transit相关的配置。

1. 在要运行Rancher Server的主机上安装Vault transit后台。
2. 通过Vault命令行或者API，创建一个叫rancher的加密密钥。
3. 通过Vault命令行或者API，创建一个Vault访问口令，这个访问口令可以通过rancher加密密钥进行加密和解密。
    - 这个访问口令必须具有一个给Rancher Server使用的安全策略，来限制Rancher Server的访问权限。下面列表中的就是之前创建的rancher加密密钥

```
path "transit/random/*" {
 capabilities = ["create", "update"]
}

path "transit/hmac/*" {
 capabilities = ["create", "update"]
}

path "transit/encrypt/rancher" {
 capabilities = ["create", "update"]
}

path "transit/decrypt/rancher" {
 capabilities = ["create", "update"]
}

path "transit/verify/rancher/*" {
 capabilities = ["create", "update", "read"]
}

path "transit/keys/*" {
 capabilities = ["deny"]
}

path "sys/*" {
 capabilities = ["deny"]
}
```

4. 启动Rancher Server，并加入相关环境变量来连接Vault。

```
$ docker run -d --restart=unless-stopped -p 8080:8080 \
-e VAULT_ADDR=https://<VAULT_SERVER> -e VAULT_TOKEN=<TOKEN_FOR_VAULT_ACCCESS> rancher/server
```

> 注意： 请检查运行的Rancher Server版本是否是你想要的。

5. 在Rancher服务启动成功之后，你需要修改Rancher中的service-backend设置。在系统管理 -> 系统设置 -> 高级设置中，找到secrets.backend。它的默认值是localkey，你可以把它修改为vault。

> 注意： 目前Rancher不支持对不同加密后台之间进行切换。

创建密文

你可以在每个Rancher环境里创建密文。这也意味着，密文名称在环境中是唯一的。同一个环境下的任何容器都可以通过配置来共享密文。例如，一个数据库的密码db_password可以被用在数据库容器里，也可以被用在Wordpress容器里。一旦这个密文被创建了，这个密文的密文值就不能被修改了。如果你需要修改一个现有的密文，唯一的方法就是删除这个密文，然后再创建一个新密文。新密文被创建后，使用这个密文的服务需要重新部署。这样容器才能使用新的密文值。

通过RANCHER命令行创建密文

在命令行当中有两种方法来创建密文。一种是在标准输入中（stdin）输入密文值，另一种是给命令行传递含有密文的文件名称。

通过标准输入（STDIN）创建密文

```
$ rancher secrets create name-of-secret - <<< secret-value
```

通过传递密文所在的文件名称来创建密文

```
$ echo secret-value > file-with-secret
$ rancher secrets create name-of-secret file-with-secret
```

通过UI创建密文

点击基础架构 -> 密文。点击添加密文。输入名称和密文值然后点击保存。

删除密文

> 备注： 目前Rancher命令行不支持删除密文。

你可以在UI里把密文从Rancher中删除，但是这并不会在已使用该密文的容器中删除该密文文件。如果一台主机上运行着使用该密文的容器，Rancher也不会在该主机上删除该密文文件。

在RANCHER中启用密文

为了在容器中使用密文，你要先部署Rancher Secrets服务。你可以把这个服务加到环境模版中，在添加该服务之后部署的新环境里都会含有Rancher Secrets服务。你也可以直接通过应用商店部署该服务。如果你想在现有的环境中部署Rancher Secrets服务，你可以通过应用商店 -> 官方认证，然后搜索Rancher Secrets找到Rancher Secrets服务。如果不部署Rancher Secrets服务的话，你仅仅可以创建密文，但是不能在你的容器里使用这些密文。

向服务 / 容器中添加密文

当密文被添加到容器中时，密文会被写到一个tmpfs卷中。你可以在容器里和主机上访问这个卷。

- 在使用该密文的容器中：这个卷被挂载在/run/secrets/.
- 在运行使用该密文的容器所在的主机上：这个卷被挂载在/var/lib/rancher/volumes/rancher-secrets/.

通过RANCHER命令行添加密文到服务中

> 注意： 密文是在compose文件版本3中被引入的。由于Rancher不支持compose文件版本，所以我们在版本2中加入了密文功能。

你可以在docker-compose.yml里，通过配置服务的secrets值来指定一个或者多个密文。密文文件的名称与在Rancher中加入的密文名称相同。在默认情况下，将使用用户ID0和组ID0创建该密文文件，文件权限为0444。在secrets里将external设置为true确保Rancher知道该密文已经被创建成功了。

基础示例DOCKER-COMPOSE.YML

```
version: '2'
services:
  web:
    image: sdelements/lets-chat
    stdin_open: true
    secrets:
    - name-of-secret
    labels:
      io.rancher.container.pull_image: always
secrets:
  name-of-secret:
    external: true
```

如果你想要修改密文的默认配置，你可以用target来修改文件名，uid和gid来设置用户ID和组ID，mode来修改文件权限。

修改密文文件配置示例DOCKER-COMPOSE.YML

```
version: '2'
services:
  web:
    image: sdelements/lets-chat
    stdin_open: true
    secrets:
    - source: name-of-secret
      target: different-target-filename
      uid: "1"
      gid: "1"
      mode: 0400
    labels:
      io.rancher.container.pull_image: always
secrets:
  name-of-secret:
    external: true
```

Racnher可以在创建应用的时候创建密文。你可以通过指定file参数，使Rancher在创建应用并启动服务之前创建密文。该密文值来自你指定的文件内容。

指定多个密文并且在启动服务前创建密文的示例DOCKER-COMPOSE.YML

```
version: '2'
services:
  web:
    image: sdelements/lets-chat
    stdin_open: true
    secrets:
    - name-of-secret
    - another-name-of-secret
    labels:
      io.rancher.container.pull_image: always
secrets:
  name-of-secret:
    external: true
  another-name-of-secret:
    file: ./another-secret
```

通过RANCHER UI添加密文到服务中

你可以在创建服务/容器页面的密文页里，向服务/容器中添加密文。

1. 点击添加密文
2. 下拉列表中会列出，已经加入到Rancher中的全部可用密文。你可以选择一个你想要使用的密文。
3. （可选操作）默认情况下，挂载到容器内的密文文件的名称为密文名。你可以在映射名称栏，给容器中的密文文件设置

一个不同的文件名。

4. （可选操作）如果你想要修改默认的文件所有者和文件权限。你可以点击自定义文件所有者及权限链接来更新配置。你可以修改用户ID，组ID和文件权限。用户ID的默认值为0，组ID的默认值为0，文件权限的默认值为0444。
5. 点击 创建.

DOCKER HUB镜像

Docker在很多自己的官方镜像中都支持通过文件来传递密文。你可以添加以_FILE结尾的环境变量名并且以/run/secrets/NAME>为值的环境变量，来达到这一效果。当在容器启动时，文件中的密文值将会被赋给去掉_FILE的环境变量里。

例如，当你部署一个MySQL容器的时候，你可以配置如下环境变量。

```
-e MYSQL_ROOT_PASSWORD_FILE=/run/secrets/db_password
```

MYSQL_ROOT_PASSWORD环境变量的值，就是你所指定这个文件的内容。这个文件就是我们在Rancher中添加的密文。这样你就可以很方便的从环境变量中获取在Rancher中配置的密文，而不用自己去读取密文文件。但并不是所有镜像都支持这个功能。

已知的安全隐患

被入侵的RANCHER SERVER容器

存储在Rancher中的密文和存储在CI系统（如Travis CI和Drone）中的密文安全程度是一样。由于加密密钥直接存储在Rancher Server容器中，所以如果Rancher Server容器被入侵，全部的密文都能被黑客获取到。Rancher将在以后的版本中努力降低这种情况的安全隐患。

> 注意： 如果你使用Vault进行加密，你需要创建一个策略来限制Rancher Server所用的token的访问权限。

被入侵的主机

如果一台主机被入侵了，这台主机上所运行的容器中使用到的全部密文，都可以被读取。 但是黑客获取不到其他主机上的额外密文。

容器访问

如果一个用户可以exec进入到容器中，该用户可以通过容器中挂载的卷查看到密文值。可以通过如下方式访问容器：

- UI点击”执行命令行”
- Rancher命令行工具
- Docker原生命令

Webhooks

在 Rancher 中，你可以创建接收器钩子。 这些钩子提供了一个可以在Rancher 中触发事件的 URL。比如，接收器钩子可以和监控系统整合来增加或减少服务的容器数量。 在 API -> Webhooks 页面， 你可以查看或创建一个接收钩子。

添加接收器钩子

要创建一个接收器钩子，导航到，API -> Webhooks，点击 添加接收器

- 填写接收器 名称 以方便识别。
- 选择你要创建的接收器 类型。
- 基于接收器的类型确定接收器事件。

点击 创建。创建成功后，就可以在新创建接收器钩子旁边看到相应的URL。

使用接收器钩子

要使用触发 URL，你需要先发一个 POST 请求到这个 URL。 向这个 URL POST 请求不需要在验证头和 body 信息。

接收器钩子的类型

- 服务扩缩容
- 主机数量增减
- 基于 DockerHub 标签的更新来更新一个服务

服务扩缩容

要扩缩容一个服务，你必须先配置你的 webhook：

- 扩大 / 缩小一个服务(即，添加或移除一个服务中的容器)
- 在环境中选择服务
- 一次投放 / 移除多少容器
- 服务的最大 / 最小容器数量

一个用接收器钩子来自动扩缩服容务的示例

使用接收器钩子来扩缩容服务，你可以通过整合外界服务来实现自动扩缩容。 在这个示例中，我们使用 Prometheus (普罗米修斯) 来监控服务，通过报警管理程序来发送 POST 请求到触发 URL.

安装 PROMETHEUS

Rancher 应用商店 提供了 Prometheus 监控服务，在 应用商店 中可以找到这个服务。选中Prometheus 然后启动应用商店入口。在 Prometheus 应用中找到一个名为 prometheus 的服务，这个服务暴露了 9090 端口。在容器中找到 /etc/prom-conf。Prometheus 的配置文件prometheus.yml 就在 /etc/prom-conf 目录。为了添加警报， 单独创建一个警报文件，在 prometheus.yml 中提供文件的路径。 比如，如果你创建了一个名为 rules.conf 的警报文件，把它加入到 prometheus.yml，在 prometheus.yml 末尾加入如下两行:

```
rule_files:
  - rules.conf
```

rules.conf 可以有多个报警配置，下面就是一个报警的配置

```
/ETC/PROM-CONF/RULES.CONF 中的警报配置例子
ALERT CpuUsageSpike
IF rate(container_cpu_user_seconds_total{container_label_io_rancher_container_name="Demo-testTarget-1"}[30s]) * 100 > 70
LABELS {
  severity="critical",
  action="up"
}
ANNOTATIONS {
  summary = "ADDITIONAL CONTAINERS NEEDED",
  description = "CPU usage is above 70%"
}
```

加入报警配置后，重启服务。

添加报警管理程序

要调用接受器钩子， 报警管理程序需要先启用。 你可以把它加入到 Prometheus 应用. 在 Prometheus 应用中点击 添加服务。用 prom/alertmanager 添加服务。添加服务的时记得映射端口9093:9093。服务启动后，在容器中执行命令，更新 etc/alertmanager/config.yml。 在 etc/alertmanager/config.yml 中添加 webhook 的 URL 。这样，当警报被触发时报警管理程序就会向这个 URL 发送 POST 请求。在 etc/alertmanager/config.yml 添加 URL 信息后需要重启服务。

示例 ETC/ALERTMANAGER/CONFIG.YML

```
route:
  repeat_interval: 5h
  routes:
  - match:
      action: up
    receiver: "webhook-receiver-up"
  - match:
      action: down
    receiver: "webhook-receiver-down"
receivers:
- name: "webhook-receiver-up"
  webhook_configs:
  - url: <WEBHOOK_URL>
    send_resolved: true
- name: "webhook-receiver-down"
  webhook_configs:
  - url: <WEBHOOK_URL>
    send_resolved: true
```

自动扩缩容

Prometheus 和警报管理程序随警报钩子更新后，重启服务器，以确保配置处于最新的激活状态。对于已经添加了警报的服务，服务会自动根据创建的更新器钩子自动扩容或缩容。

主机弹性伸缩

Rancher 可以通过克隆用 Rancherc 创建的， 并且已经存在的主机来增加主机的数量。(即 Docker Machine)。这意味这通过 自定义命令 添加的主机不能进行伸缩。

使用 主机上的标签， 主机可以被分组到一起组成一个弹性伸缩组。我们推荐在主机上使用唯一的标签来方便区分弹性伸缩组。任何标签相同的主机，不管它是如何被添加到Rancher的，都会被当作是同一个弹性伸缩组的一部分。创建 webhook 时, 主机上不要求有标签，但是当在弹性伸缩组中使用webhook时，至少要有一个主机带有标签，这样 webhook 才能有一个可以克隆的主机。总之， Rancher 会选择一台在弹性伸缩组中的可克隆主机

要弹性伸缩主机，你必须配置你的 webhook：

- 扩增 / 减少主机(即，添加或移除主机)
- 添加一个主机选择器标签。这个标签是用来把主机分组成一个弹性伸缩组的标签。
- 选择单次要伸缩的主机数量。
- 选择主机数量伸缩的上下限。添加主机时，主机数量不能超过上线， 减少时不能低于下限。

- 如果你创建了一个 webhook 来缩小主机的数量，你可以选者移除主机的优先顺序。

主机弹性伸缩注意事件

- 主机标签: 标签被用把主机划分为不同的弹性伸缩组。因为这些标签是由用户添加的，在选择，添加，编辑，标签时必须要非常小心。任何添加在主机上的标签都会自动地把这台主机到添加到一个弹性伸缩组。如果这台主机是可克隆的，它可能会被用于克隆出更多主机。任何主机标签的移除都会自动地把相应的主机从弹性伸缩组移除，这台主机也将不再能够被 webhook 服务克隆或移除。
- 自定义主机: 任何类型的主机都可以被添加到弹性伸缩组中，你只需要在主机上添加一个标签。Rancher 不能用这些主机来克隆或创建出更多主机。
- 主机克隆: 因为主机扩增既是主机克隆，所有配置，包括资源分配，Docker 引擎等都会在新主机被复制。Rancher 总是会用克隆最旧的主机。
- 处于错误状态的主机: 任何处于 Error 状态的主机都不会被添加到弹性伸缩组中.
- 移除主机的顺序: 从 Rancher 中删除主机时，Rancher会根据主机的状态，按以下的顺序删除弹性伸缩组中的主机 (Inactive， Deactivating，Reconnecting 或 Disconnected)，最后才会删除处于 active 状态的主机

基于 DOCKER HUB WEBHOOKS 升级服务

利用 Docker Hub 的 webhooks, 你可以加入 Rancher 的接收器钩子。这样，每当push一个镜像，一个 POST 请求就会被发送到 Rancher 来触发这个触发器钩子。使用这种 webhooks 组合, 你可以实现自治。 这样，每当在 Docker Hub push一个 image:tag，所有使用了匹配这个镜像版本的服务都会自动被升级。你需要用一个选择器标签来选择匹配的服务，然后再升级选中的服务。标签应该在服务创建时添加。如果服务没有标签，你需要在Rancher中升级服，然后添加供 webhook 使用的标签。

为了升级服务，你必须配置自己的 webhook：

- 选择要升级的标签
- 选择标签来找到要升级的服务
- 确定单次要升级的容器数量(即，批量大小)
- 确定在升级期间启动下一个容器的秒数(即，批量间歇)
- 选择是否新容器应该在旧容器停止前启动。

创建接受器钩子后，你需要在你的 Docker Hub webhook 中使用 触发 URL。当Docker Hub 触发自己的 webhook, 被 Rancher 触发器钩子选中的服务会被升级。Rancher 触发器钩子默认需要 Docker Hub webhook 提供的特定信息。同时使用 Rancher’s 接受器钩子和其它webhook，POST 请求中需要包含以下字段:

```
{
    "push_data": {
        "tag": <pushedTag>
    },
    "repository": {
        "repo_name": <image>
    }
}
```

标签

Rancher在服务/容器和主机上使用标签来帮助管理Rancher的不同功能。

RANCHER COMPOSE标签使用指南

标签用于帮助Rancher启动服务并利用Rancher的功能。下列的标签索引用于帮助用户使用Rancher Compose来创建服务。 这些标签在UI上有对应关系，不需要额外添加到服务上。

Key	Value
io.rancher.sidekicks	服务名称
io.rancher.loadbalancer.target.SERVICE_NAME	REQUEST_HOST:SOURCE_PORT/REQUEST_PATH=TARGET_PORT
io.rancher.container.dns	true
io.rancher.container.hostname_override	容器名称
io.rancher.container.start_once	true
io.rancher.container.pull_image	always
io.rancher.container.requested_ip	IP于10.42.0.0/16的地址空间
io.rancher.container.dns.priority	service_last
io.rancher.service.selector.container	Selector Label Values
io.rancher.service.selector.link	Selector Label Values
io.rancher.scheduler.global	true
io.rancher.scheduler.affinity:host_label	主机标签的Key Value配对
io.rancher.scheduler.affinity:container_label	容器标签的Key Value配对
io.rancher.scheduler.affinity:container	容器名称
io.rancher.lb_service.target	Target Service Label Values

> 注意： 对于以io.rancher.scheduler.affinity为前缀的标签，根据你想要匹配的方式（即相等或不相等，hard或soft规则）会有轻微的变化。 更多细节可以在这里找到这里.

选择器标签

使用 选择器标签（即io.rancher.service.selector.link, io.rancher.service.selector.container），Rancher可以通过标签识别服务/容器，并将它们自动链接到服务。 将在以下两种情况下进行评估。 情景1是将 选择器标签 添加到服务时。 在情景1中，对所有现有标签进行评估，以查看它们是否与 选择器标签 匹配。 情景2是服务已经有 选择器标签 时。 在情景2中，检查添加到Rancher的任何新服务/容器，看看它是否符合链接条件。 选择器标签 可以由多个要求组成，以逗号分隔。 如果有多个要求，则必须满足所有要求，因此逗号分隔符作为 **AND** 逻辑运算符。

```
# 其中一个容器标签必须具有一个等于`foo1`的键，并且值等于`bar1`
foo1 = bar1
# 其中一个容器标签必须具有一个等于`foo2'的键，值不等于`bar2`
foo2 != bar2
#其中一个容器标签必须有一个等于`foo3`的键，标签的值不重要
foo3
#其中一个容器标签必须有一个等于`foo4`的键，值等于`bar1`或`bar2`
foo4 in (bar1, bar2)
#其中一个容器标签必须有一个等于`foo5'的键和'bar3`或`bar4`以外的值
foo5 notin (bar3, bar4)
```

> 注意： 如果标签有中包含逗号的标签，则选择器将无法与标签匹配，因为 选择器标签 可以匹配任何没有关联值的键。 示例：io.rancher.service.selector.link: foo=bar1,bar2的标签将转换为任何服务必须具有一个标签为foo的键值，并且值等于bar1 和另一个带有等于bar2的标签。 它不会选择一个键等于foo，并且值等于bar1，bar2的标签的服务。

逗号分隔列表的示例

```
service1:
  labels:
    # 添加选择器标签来接收其他服务
    io.rancher.service.selector.link: hello != world, hello1 in (world1, world2), foo = bar
```

在此示例中，将链接到service1的服务需要满足以下所有条件：

- 具有键等于hello并且值不等于world的标签
- 具有键等于“hello1”但值可以等于world1或world2的标签
- 具有键等于foo和值等于bar的标签

以下示例，service2在部署时会自动链接到service1。

```
service2:
   labels:
      hello: test
      hello1: world2
      foo: bar
```

服务上的系统标签

除了Rancher Compose可以使用的标签之外，Rancher在启动服务时还会创建一系列系统标签。

Key	描述
io.rancher.stack.name/io.rancher.project.name	根据应用名称创建
io.rancher.stack_service.name/io.rancher.project_service.name	根据服务名称创建
io.rancher.service.deployment.unit	根据部署的从容器服务创建
io.rancher.service.launch.config	基于从容器服务的配置创建。
io.rancher.service.requested.host.id	根据该服务安排在哪个主机上创建

主机标签

主机标签 可以在主机注册期间添加到主机，创建后可通过编辑在主机中添加。

Key	Value	描述
io.rancher.host.external_dns_ip	用于外部DNS的IP, 例如： a.b.c.d	用于外部DNS服务，并需要对DNS记录进行编程使用主机IP以外的IP

自动创建的主机标签

Rancher会自动创建与主机的linux内核版本和Docker Engine版本相关的主机标签。 这些标签可以用于调度.

Key	Value	描述
io.rancher.host.linux_kernel_version	主机上的Linux内核版本 (例如3.19)	主机上运行的Linux内核的版本
io.rancher.host.docker_version	主机上的Docker版本（例如1.10.3）	主机上运行的Docker Engine版本
io.rancher.host.provider	云提供商信息	云提供商名称（目前仅适用于AWS）
io.rancher.host.region	云提供商区域	云提供商区域（目前仅适用于AWS）
io.rancher.host.zone	云提供商可用区	云提供商可用区（目前仅适用于AWS）

本地DOCKER标签

Key	Value	描述
io.rancher.container.network	true	将此标签添加到docker run命令中，以将Rancher网络添加到容器中

目标服务标签

负载均衡可以配置为将流量优先分发于同负载均衡为同一主机的目标容器。 根据标签的值，负载均衡将被配置为将流量定向到指定的容器，或者将流量的优先级设置为某些指定的容器。 默认情况下，负载平衡器以Round-robin算法将流量分发到目标服务下的所有容器。

Key	Value	描述
io.rancher.lb_service.target	only-local	只能将流量转发到与负载均衡为相同主机的容器上。 如果同一主机上没有目标服务的容器，则不会将流量转发到该服务。
io.rancher.lb_service.target	prefer-local	将流量优先于同负载均衡容器为同一主机上的容器。 如果在同一主机上没有目标服务的容器，则流量将被路由到其他拥有目标服务容器的宿主机上。

Kubernetes

要在Rancher中部署Kubernetes，你首先需要创建一个新的环境 ，创建环境时需要使用一个设置了Kubernetes容器编排引擎的环境模板。

设置KUBERNETES

Kubernetes可以在创建或者编辑环境模板时设置。如果你启动了一个Cattle环境, 你可以从应用商店 -> 官方认证中启动Kubernetes。 如果选择Catalog方式, 你可以跳过下列步骤1。

1. 在环境模板中编辑Kubernetes设置, 在环境模板的Orchestration一栏下点击Edit Config。你可以在创建一个新的环境模板时编辑设置或者编辑一个已有的环境模板的设置。
2. 确认模板中的Kubernetes版本是你需要的版本。
3. 选择cloud providers, backups, add-ons等的设置选项。
4. 点击设置保存设置到环境模板并点击启动按钮从应用商店中启动Kubernetes。

> 注意： 我们建议从正确的设置中启动Kubernetes环境。如果你希望修改一个已有的Kuberentes部署的设置，你可以点击已经是最新版本按钮升级Kubernetes部署到新的设置。

创建一个KUBERNETES环境

在环境菜单的下拉列表中, 点击管理环境。要创建一个新的环境，点击添加环境, 输入名称, 描述 (可选)等信息, 选择你希望使用的设置了Kubernetes作为容器编排引擎的环境模板。 如果启用了访问控制, 你可以添加成员并设置他们的成员角色。任何被添加到成员列表的用户都将能够访问你的环境。

在Kubernetes环境创建之后，你可以通过UI界面左上角环境菜单的环境下拉列表切换到这个环境，你也可以在环境管理页面中点击对应环境右侧的切换到此环境按钮切换到对应环境。

> 注意： 由于Rancher支持多种容器编排框架，目前暂不支持在已有服务运行的环境中切换容器编排框架。

启动KUBERNETES

创建Kubernetes环境之后, 直到你添加至少一台主机到这个环境之前，基础设施服务将不会启动。添加主机的操作步骤在各种容器编排框架下都是相同的。一旦第一台主机被添加，Rancher将会自动开始基础设施服务的部署包括Kubernetes相关服务(也就是：master, kubelet, etcd, proxy等)。你可以通过访问 Kubernetes -> 基础设施标签查看部署进度。

KUBERNETES环境主机需求

所需端口

- 被用为运行Kubernetes节点的主机需要开通10250和10255端口给kubectl。为了可以访问到暴露的服务，用作NodePort的端口也需要被开通。默认的NodePort是从30000到32767之间的TCP端口。
- 对于重叠平面部署: 至少1 CPU, 2GB RAM。资源需求根据应用负载不同而不同。
- 对于分隔平面部署: 此种部署类型至少需要5台主机。
- 数据平面: 添加至少3台具有1 CPU, 大于等于1.5GB RAM, 大于等于20GB DISK的主机。添加主机时, 参考设置主机标签中的步骤为主机设置etcd=true标签。
- 编排平面: 添加至少1台由于大于等于1 CPU且大于等于2GB RAM的主机。添加主机时, 参考设置主机标签中的步骤为主机设置orchestration=true标签。你可以仅使用一台主机部署, 但这样将牺牲高可用性。当仅有的一台主机发生故障时，一些K8s功能例如API、在pods发生故障时进行重新调度等，将不会正常工作，直到一台新的主机被加入。
- 计算平面: 添加一台或多台主机。 添加主机时, 参考设置主机标签中的步骤为主机设置compute=true标签。

> 注意： 只有Rancher管理员和环境所有者有权限查看基础设施服务.

当添加主机到Kubernetes环境时，主机名将作为运行kubectl get nodes时返回的Kubernetes节点的唯一标识。

使用KUBERNETES

一旦部署完成，你就可以开始通过以下方式创建或者管理你的Kubernetes应用:

RANCHER应用商店

Rancher支持提供一个基于Kubernetes模板的应用商店。要使用一个模板，点击应用商店标签。选择你希望部署的模板并点击查看详情。检查并编辑应用栈名称、描述以及设置选项并点击启动。

如果你希望添加自己的应用模板到Kubernetes应用商店，你可以把编排文件添加到Rancher catalog中的kubernetes-templates目录下。

KUBECTL

要设置你自己的kubectl客户端访问新创建的Kubernetes集群，点击Kubernetes -> kubectl。点击生成配置文件按钮以生成需要的kube/config_file配置文件，你可以下载并添加此文件到你的本地目录。

配置文件中提供了本地主机所需要的相关信息，你通过kubectl创建的所有对象都将被显示在Rancher中。

KUBECTL VIA SHELL

Rancher提供了一个易用的shell界面访问一个托管的kubectl实例，可以通过这个kubectl管理Kubernetes集群和应用。

在KUBERNETES中添加私有镜像仓库

通过在Kubernetes环境中添加私有镜像仓库，环境中的Kubernetes服务能够使用这些私有镜像仓库。

Kubernetes - 弹性平面（Resiliency Planes）

对生产环境的部署，最佳实践是每一个平面运行在专门的物理或虚拟主机上。对开发环境，可以用多租户去简化管理和减少开销。

数据平面

数据平面由一个或多个 etcd 容器组成。Etcd是一个分布式可靠的键-值存储，它存储了所有Kubernetes状态。可以认为数据平面是有状态的，也就是说组成数据平面的软件维护着应用状态。

编排平面

编排平面由无状态的组件组成，它们控制Kubernetes分发。

计算平面

计算平面由 Kubernetes pods组成。

计划

在安装之前，考虑你特定的用例是很重要的。Rancher提供了两种不同的部署类型。

如果你在寻找一种快速启动Kubernetes的方式以试验我们的Kubernetes，我们建议通过 重叠的平面的方式启动Kubernetes。这是默认Kubernetes模版的默认配置。

针对生产环境，Rancher建议通过 分隔的平面的方式启动Kubernetes。

安装

重叠平面（OVERLAPPING PLANES）

默认情况，Kubernetes设置为用重叠平面的方式做部署。所有平面可以重叠，所有服务可以运行在一个主机上。服务会被随机地调度。增加至少三台主机以使数据平面（亦即 etcd）有复原能力。

1. 创建一个 Kubernetes 环境。
2. 增加1个或多个主机，主机至少有 1 CPU， 2GB 内存。资源需求依据工作负荷有所区别。

分隔平面（SEPARATED PLANES）

这种部署方式允许用户分割开不通类型的平面，使每一种平面运行在特定、专用的主机上。它可以提供数据平面的恢复能力，保证计算平面的性能。你需要在增加主机之前 配置 Kubernetes。在配置 Kubernetes 的时候，在 Plane Isolation 选项中选择 required。

> 注意： 如果你想要从重叠平面升级Kubernetes到分隔平面，请 阅读如何升级Kubernetes以正确地处理改变。

增加带标签（LABEL）的主机

所有加入到Kubernetes环境的主机必须打好标签，这样 Rancher 就可以根据平面类型去调度不同的服务。在这种部署类型的情况下，最少需要5台主机。

1. 数据平面： 增加三个或以上的主机，主机需要有 >=1的CPU，>=1.5GB的内存，>=20GB的磁盘存储空间。在加入主机的时候，给主机打上标签 etcd=true。
2. 编排平面 增加2个或以上的主机，主机需要有 >=1 的CPU和 >=2GB 的内存。在加入主机的时候， 给主机打上标签 orchestration=true。你可以只用一台主机，但是在这台主机故障的时候，直到新的用于编排的主机加入之前，K8s API将会不可用。

3. 计算平面 增加一个或以上的主机。在加入主机的时候， 给主机打上标签 compute=true。

> 注意： 主机标签可以加到已有的主机上，从而将这台主机添加到某一种平面中，但我们不支持通过修改标签把一种平面类型的主机转到另一种平面类型。如果想改变一台主机的平面类型，你可以删除旧的平面类型标签，接着删除主机上已有的所有服务，接着再增加或更新为新的平面类型标签，最后通过 升级 Kubernetes来根据新的平面标签重新平衡容器。

Kubernetes - Cloud Providers

在Kubernetes中, 有一个cloud providers的概念, cloud provider是Kubernetes的一个模块，提供接口用于管理负载均衡、节点(也就是主机)以及网络路由。

目前, Rancher在 设置Kubernetes 时支持以下两种类型的cloud provider。 你可以选择使用哪种cloud provider。

RANCHER

- 节点: 支持任何可以被加入Rancher的主机。
- 负载均衡: 启动Rancher的负载均衡, 使用HAproxy和rancher/lb-service-haproxy镜像作为负载均衡服务。 默认情况下, 负载均衡将请求以轮询方式发送给pods。

默认情况下, Kubernetes的cloud proivder被设置为rancher。

AWS

- 节点: 仅支持以自定义主机方式添加AWS主机。
- 负载均衡: 启动一个AWS Elastic Load Balancer (ELB)作为负载均衡服务。 同时，你仍然可以通过使用ingress对象来创建Rancher负载均衡。
- 持久化卷(PV): 能够使用AWS Elastic Block Stores (EBS)用于persistent volumes.

添加主机

在设置Kubernetes以aws cloud provider运行后，任何加入环境的主机都必须是一个AWS EC2实例并且至少具有以下IAM策略：

```
{
  "Effect": "Allow",
  "Action": "ec2:Describe*",
  "Resource": "*"
}
```

为了在Kubernetes中使用Elastic Load Balancers (ELBs)和EBS, 主机需要拥有一个具备合适权限的IAM角色。

IAM角色策略示例:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:Describe*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "ec2:AttachVolume",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "ec2:DetachVolume",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": ["ec2:*"],
      "Resource": ["*"]
    },
    {
      "Effect": "Allow",
      "Action": ["elasticloadbalancing:*"],
      "Resource": ["*"]
    }
  ]
}
```

ELASTIC LOAD BALANCER (ELB)作为一个KUBERNETES服务

在设置Kubernetes中选择aws作为cloud provider并确保主机拥有配置ELB的相应IAM策略后，你可以开始创建负载均衡。

LB.YML文件示例

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-lb
  labels:
    app: nginx
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
  - name: http
    port: 80
    protocol: TCP
```

通过使用kubectl客户端, 让我们在Kubernetes中启动我们的负载均衡服务。 记住, 你可以通过为本地主机配置kubectl在本机使用kubectl客户端或者通过UI界面中的Kubernetes -> kubectl 命令行界面使用kubectl客户端。

```
$ kubectl create -f lb.yml
service "nginx-lb" created
$ kubectl describe services nginx-lb
Name:            nginx-lb
Namespace:        default
Labels:            app=nginx
Selector:        name=nginx
Type:            LoadBalancer
IP:            10.43.137.5
LoadBalancer Ingress:    a4c7d4290f48011e690470275ac52fef-1158549671.us-west-2.elb.amazonaws.com
Port:            http    80/TCP
NodePort:        http    32166/TCP
Endpoints:        <none>
Session Affinity:    None
Events:
  FirstSeen    LastSeen    Count    From            SubObjectPath    Type        Reason            Message
  ---------    --------    -----    ----            -------------    --------    ------            -------
  17s        17s        1    {service-controller }            Normal        CreatingLoadBalancer    Creating load balancer
  14s        14s        1    {service-controller }            Normal        CreatedLoadBalancer    Created load balancer
```

使用EBS卷

在设置Kubernetes使用aws作为cloud provider并确保主机拥有配置EBS的相应IAM策略后，你可以开始使用EBS卷.

在Rancher中搭配Kubernetes使用私有仓库

如果你在 离线环境 运行Rancher，或者Rancher不能够访问DockerHub (亦即 docker.io) 以及Google容器仓库 (亦即 gcr.io)，那么Pod的基础容器镜像和Kubernetes的插件将无法正常安装。你需要 配置 Kubernetes 来指定一个私有仓库以安装Kubernetes的插件以及Pod的基础容器镜像。

对私有仓库的要求

Rancher 期望私有仓库能映射（mirror）DockerHub (亦即 docker.io) 和 Google 容器仓库 (亦即 gcr.io).

POD 基础容器镜像

在 配置 KubernetesRancher使用一个 Pod 基础容器镜像。每一个Pod会用它来共享 network/ipc 命名空间。

```
# 配置为k8s模版中的默认值
Image: gcr.io/google_containers/pause-amd64:3.0
```

KUBERNETES 插件

镜像的 namespace/name:tag 需要和 Rancher 插件模版中的镜像保持一致

下列是目前支持的插件的镜像列表，你需要 查看 Github 仓库找具体的版本号 然后复制具体版本号到每个镜像。

HELM的镜像

```
# 位于 helm/tiller-deploy.yaml
image: <$PRIVATE_REGISTRY>/kubernetes-helm/tiller:<VERSION>
```

DASHBOARD的镜像

```
# 位于 dashboard/dashboard-controller.yaml
image: <$PRIVATE_REGISTRY>/google_containers/kubernetes-dashboard-amd64:<VERSION>
```

HEAPSTER的镜像

```
# 位于 heapster/heapster-controller.yaml
image: <$PRIVATE_REGISTRY>/google_containers/heapster:<VERSION>

# 位于 heapster/influx-grafana-controller.yaml
image: <$PRIVATE_REGISTRY>/kubernetes/heapster_influxdb:<VERSION>
image: <$PRIVATE_REGISTRY>/google_containers/heapster_grafana:<VERSION>
```

Rancher Kubernetes RBAC集成

从Rancher v1.6.3开始支持

在默认情况下，K8S所在的Rancher环境里的所有用户都可以访问K8S内的资源。启用Kubernetes RBAC可以使得环境的所有者能够限制环境中的某些用户去访问K8S的资源。

认证模块集成在了Rancher的访问控制选项，这意味着Rancher所支持的全部外部认证系统都可以用在K8S的RBAC里。

在使用Kubernetes RBAC权限之前，你需要了解命名空间对定义。了解更多命名空间的内容，请参考Kubernetes 文档.

启用KUBERNETES RBAC

为了启用Kubernetes RBAC，你需要确保正确配置了Kubernetes并且选择了启用Kubernetes RBAC。如果你已经部署了Kubernetes基础设施服务，你可以点击已经为最新版本来进行Kubernetes相关配置的修改。

默认角色

环境的所有者被授予了环境中Kubernetes集群对管理员权限。拥有管理员权限意味着用户可以对任何Kubernetes资源进行读写操作。同时，也可以通过创建角色给Kubernetes环境中的其他用户创建权限。

> 重要： 在默认情况下，任何非环境所有者对用户对Kubernetes集群中任何的资源都没有权限，除非这些用户被赋予了相关权限后，才可以访问Kubernetes集群中的相应资源。

从环境中删除一个用户会移除掉该用户在Kubernetes集群中拥有的所有访问权限。给一个环境所有者降权会移除该用户对Kubernetes集群的管理员权限。类似的，如果将一个用户升级为所有者，这个用户将会拥有Kubernetes集群的管理员权限。

已知问题

Helm和Tiller目前不支持RBAC角色。任何可以访问它们其中之一的用户拥有Kubernetes集群的管理员权限。

向用户 / 组中添加新的角色

为了给非所有者用户提供访问Kubernetes集群资源的权限，所有者需要创建一个新的角色并且把这个角色应用到这些用户上。有两种Kubernetes资源类型RoleBindings和ClusterRoleBindings。RoleBindings可以将权限应用于环境中的某个命名空间里。ClusterRoleBindings可以把权限应用于全局。（例如，某个环境里的全部的命名空间）。

kubectl apply命令可以用来应用角色绑定。你也可以使用kubectl apply命令，通过简单的从资源文件中添加或者删除权限来实现权限的更新。

组名称规则

当将组应用到角色绑定时，你的Rancher访问控制将会决定了组名称的语法规则。

认证方式	组名称规则
Active Directory	ldap_group:
Azure AD	azuread_group:
Github Organizations	github_org:
Github Teams	github_team:
OpenLDAP	openldap_group:
Shibboleth	shibboleth_group:

> 注意： 目前，在kubernetes RBAC里，你可以把权限应用于特定的组，但是你必须把组内用户的个体添加到环境当中。目前，kubernetes RBAC不支持将整个组添加到一个环境当中。

角色

用户可以在特定的命名空间中被授予特定的权限。Kubernetes内嵌了3种方便的角色。

- view - 对命名空间下的大多数资源拥有只读权限。
- edit - 对命名空间下的大多数资源拥有读写权限。
- admin - 拥有edit角色的全部权限，同时可以创建新的角色和角色绑定。

自定义角色相对于内嵌的角色（admin, edit 和 view）来说，允许更具体的管控。更多有关如何构建特定角色的信息，请参考Kubernetes RBAC文档。

示例

在特定命名空间里给两个用户授予修改权限

在这个例子里，我们给两个用户developer1和developer2添加权限。从而使这两个用户可以对dev命名空间下几乎全部的资源进行读写操作。我们使用如下的角色绑定把edit角色授予这两个用户。

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: edit-dev
  namespace: dev # 指定你想要把权限应用于哪个命名空间
subjects:
  - kind: User
    name: developer1
  - kind: User
    name: developer2
roleRef:
  kind: ClusterRole
  name: edit # 指定你想要把哪个角色应用于这两个用户
  apiGroup: rbac.authorization.k8s.io
```

在特定命名空间里给一个用户授予只读权限

在这个例子里，我们给一个用户developer2添加权限。从而使这个用户可以对qa命名空间下几乎全部的资源进行只读操作。我们使用如下的角色绑定把view角色授予这个用户。

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: view-qa
  namespace: qa # 指定你想要把权限应用于哪个命名空间
subjects:
  - kind: User
    name: developer2
roleRef:
  kind: ClusterRole
  name: view # 指定你想要把哪个角色应用于用户
  apiGroup: rbac.authorization.k8s.io
```

在特定命名空间里给一个组授予只读权限

在这个例子里，我们给一个Github团队mycompany-research中全部的成员添加权限。从而使这个团队可以对dev命名空间下几乎全部的资源进行只读操作。我们使用如下的角色绑定把view角色授予这个团队。

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: view-dev-group
  namespace: dev # 指定你想要把权限应用于哪个命名空间
subjects:
  - kind: Group # 指定这是一个组而不是独立用户
    name: github_team:mycompany-research
roleRef:
  kind: ClusterRole
  name: view # 指定你想要把哪个角色应用于用户
  apiGroup: rbac.authorization.k8s.io
```

任何mycompany-research团队中的成员都拥有这个集群中的大多数资源的只读权限。

> 注意： 目前，在kubernetes RBAC里，你可以把权限应用于特定的组，但是你必须把组内用户的个体添加到环境当中。目前，kubernetes RBAC不支持将整个组添加到一个环境当中。

给全部用户授予列出命名空间的权限

在Kubernetes Dashboard里切换名命名空间对某些用户来说比较困难。例如这个用户没有列出全部命名空间的权限。下面的集群角色和关联的绑定可以使全部的用户拥有列出命名空间的权限。

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: default-reader
rules:
  - apiGroups: [""]
    resources:
      - namespaces
    verbs: ["get", "watch", "list"]
  - nonResourceURLs: ["*"]
    verbs: ["get", "watch", "list"]

---

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: default-reader-binding
subjects:
  - kind: Group
    name: system:authenticated
roleRef:
  kind: ClusterRole
  name: default-reader
  apiGroup: rbac.authorization.k8s.io
```

你可以根据自己的需要，将下面的资源也添加到default-reader当中。

- componentstatuses
- events
- endpoints
- namespaces
- nodes
- persistentvolumes
- resourcequotas
- services

Kubernetes - 灾难恢复

Rancher在三个不同的主机上运行多达三个 etcd 实例。如果运行 etcd 的多数主机出了故障，按以下步骤做灾难恢复：

1. 在 Kubernetes -> Infrastructure Stacks页面中，展开 Kubernetes 栈。点击 etcd 服务。查找一个处于 running 状态的 kubernetes-etcd 容器。通过 Execute Shell 执行进这个容器中，运行 etcdctl cluster-health。 如果输出的最后一行显示 cluster is healthy，那么这会儿没有灾难发生并且etcd集群也正常。 如果最后行显示 cluster is unhealthy，记下这个 kubernetes-etcd 容器。这是你唯一的幸存者。使用这个容器做扩展（scale up）可以取代其余的出故障的容器。
2. 删除处于 Disconnected 状态的主机。确认其中没有主机在运行你的幸存者容器。
3. 执行进入上述幸存者容器，在shell中运行 disaster。容器会自动重启，并且etcd会自愈成单节点的集群。系统功能会恢复。
4. 添加新的主机直到有至少拥有先前运行etcd的主机的数量。旧的主机会包含etcd数据卷和备份，这将引起问题。如果你不能添加新主机，你可以先清理主机然后再重新添加这台主机。我们建议运行至少三个主机。如果你正在使用 分隔平面，别忘了给你的主机加上 etcd=true 标签。 随主机的加入并运行 etcd 服务，etcd会扩展回集群的数量。在大部分情况，一切都会自动恢复。如果新的 / 死掉的容器在 initializing 状态卡住超过三分钟，执行进这些容器并运行 delete。 在任何情况都不要在你的幸存者容器中运行 delete 命令。

Kubernetes - 备份

默认情况下，Kubernetes中的备份是激活的。网络存储的延迟和大小都应该在备份中被考虑到。 每个备份50MB是一个比较好的对存储需求的估计。比如，每15分钟创建一个备份，保留一天的策略会存储最多96个备份数据，需要大约5GB的存储。如果没有从任意时刻及时恢复到前一个保存点的意图，可以保留更少的历史备份数。

KUBERNETES 配置

在 配置 Kubernetes的时候，你可以选择是否激活备份。

如果备份被激活，你可以指明备份创建周期和备份保留周期。

备份周期的时间设置必须是十进制数字序列，每个可以带额外的分数以及单位后缀，例如 300ms, 1.5h 或者 2h45m。有效的时间单位有 ns， us 或 µs， ms， s， m 以及 h。

备份创建周期 表明了备份创建的速率，不推荐短于 30s 的创建周期。

备份保留周期 表明历史备份删除的速率。留存时间超过该周期的备份会在下一次成功备份之后过期。

磁盘中存储的最大备份数量满足如下等式 ceiling(保留周期 / 创建周期)。 例如， 5m 的创建周期和 4h 的保留周期最多会存储 ceiling(4h / 5m) 个备份，亦即 48 个备份。对备份大小的保守估计是 50MB，因此挂载的网络存储应该有至少 2.4GB 空闲空间。备份大小会依据使用情况有所区别。

如果备份被禁用， 备份创建周期 和 备份保留周期 会被忽略。

设置远程备份

目前，备份存储在host上的一个静态位置： /var/etcd/backups。你需要在所有运行etcd服务的host的该目录挂载网络存储。网络存储必须在Kubernetes启动之前设置好。

从备份恢复

如果所有运行 etcd 服务的主机都出故障，遵循以下步骤：

1. 通过从Kubernetes -> Infrastructures Stacks 删除 Kubernetes 应用栈，将你的环境中的调度类型修改为 Cattle。Pods会保持完整及可用。
2. 删除 disconnected 的主机并增加新的主机。如果你选择了 弹性控制面板resiliency planes，你需要加入带etcd=true标签的主机。
3. 对将会运行 etcd 服务的每台主机，挂载包含备份的网络存储，这应该作为 配置远程备份的部分被创建。然后执行以下命令： 在命令行中运行:

```
* target=<NAME_OF_BACKUP>
* docker volume rm etcd
* docker volume create --name etcd
* docker run -d -v etcd:/data --name etcd-restore busybox
* docker cp /var/etcd/backups/$target etcd-restore:/data/data.current
* docker rm etcd-restore
```

> 注意： 你必须以一个对远程备份有读权限的用户登录。否则 docker cp 命令会默默失败。

1. 通过应用商店（catalog）启动Kubernetes。请确保你 配置好 Kubernetes。 Kubernetes 的基础设施应用会启动并且你的Pods会恢复一致。你的备份可能会导致一个与当前存在的部署不同的拓扑结构，Pods可能会被删除 / 重建。

Kubernetes中的持久化存储

Rancher能够通过Kubernetes原生资源对象创建拥有持久化存储的服务。在Kubernetes中, 持久化存储通过API资源对象管理, 其中包括PersistentVolume和PersistentVolumeClaim。Kubernetes中的存储组件支持多种后端存储(例如：NFS、EBS等), 存储具有独立于pod的生命周期。根据你希望使用的持久化卷的类型，你可能需要设置Kubernetes环境.

以下是如何在Rancher的Kubernetes环境中使用NFS和EBS的示例。

PERSISTENT VOLUMES - NFS

在Kubernetes中使用NFS卷时，文件系统(也就是NFS)将被挂载在pod中。NFS允许多个pod同时进行写操作，这些pod使用相同的persistent volume claim。通过使用NFS卷，相同的数据可以在多个pod之间共享。

NFS设置

你需要有一台正常运行的NFS服务器并设置了共享目录。在下面的示例中，我们假定/nfs被设置为共享目录。

创建PERSISTENT VOLUMES (PV)和PERSISTENT VOLUME CLAIMS (PVC)

在Kubernetes模板中，kind需要被设置为PersistentVolume并使用nfs资源。 服务器将使用 `<IP_OF_NFS_SERVER>` 中设置的地址以及path所指定的路径作为共享目录(在我们的示例yaml文件中也就是/nfs目录)。

示例pv-nfs.yml文件

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: <IP_OF_NFS_SERVER>
    # Exported path of your NFS server
    path: "/nfs"
```

通过kubectl客户端, 我们可以使用下面的命令在Kubernetes中创建persistent volume。记住, 你可以通过为本地主机配置kubectl在本机使用kubectl客户端或者通过UI界面中的Kubernetes -> kubectl 命令行界面使用kubectl客户端。

```
$ kubectl create -f pv-nfs.yml
```

创建persistent volume之后, 你可以创建persistent volume claim，用于请求创建的persistent volume资源。

示例pvc-nfs.yml文件

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Mi
```

通过kubectl客户端, 我们可以使用以下命令在Kubernetes中创建persistent volume claim。记住，你可以通过为本地主机配置kubectl在本机使用kubectl客户端或者通过UI界面中的Kubernetes -> kubectl 命令行界面使用kubectl客户端。

```
$ kubectl create -f pvc-nfs.yml
```

在创建了persistent volume和persistent volume claim之后, 你可以通过以下命令确认persistent volume已绑定。

```
$ kubectl get pv,pvc

NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS    CLAIM         REASON    AGE
pv/nfs    1Mi        RWX           Retain          Bound     default/nfs             8s

NAME      STATUS    VOLUME    CAPACITY   ACCESSMODES   AGE
pvc/nfs   Bound     nfs       1Mi        RWX           5s
```

创建一个POD使用PERSISTENT VOLUME CLAIM

现在persistent volume claim已经被创建, 我们可以创建使用这个persistent volume claim的pods。这些pods将会访问相同的数据。

示例rc-nfs.yml文件

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: rc-nfs-test
spec:
  replicas: 2
  selector:
    app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
          - name: nginx
            containerPort: 80
        volumeMounts:
            - name: nfs
              mountPath: "/usr/share/nginx/html"
      volumes:
      - name: nfs
        persistentVolumeClaim:
          claimName: nfs
```

通过kubectl客户端, 我们可以使用以下命令在Kubernetes中创建replication controller和两个pods。记住，你可以通过为本地主机配置kubectl在本机使用kubectl客户端或者通过UI界面中的Kubernetes -> kubectl 命令行界面使用kubectl客户端。

```
$ kubectl create -f rc-nfs-test.yml
service "nginx-service" created
replicationcontroller "nginx" created

$ kubectl get pods
NAME           READY     STATUS    RESTARTS   AGE
rc-nfs-test-42785   1/1       Running   0          10s
rc-nfs-test-rt3ld   1/1       Running   0          10s
```

我们可以测试两个pods都可以访问同一个persistent volume以及NFS服务端产生的修改。示例如下，我们可以在NFS服务器端的/nfs目录下创建一个文件。

```
$ echo "NFS Works!" > /nfs/index.html
```

在NFS服务器端创建文件后，我们可以在两个pods中查看这个文件。

```
$ kubectl exec rc-nfs-test-42785 cat /usr/share/nginx/html/index.html
NFS Works!
$ kubectl exec rc-nfs-test-rt3ld cat /usr/share/nginx/html/index.html
NFS Works!
```

PERSISTENT VOLUMES - EBS

要在Kubernetes中使用EBS作为persistent volume，你需要对Kubernetes进行一些设置。

1. 参考AWS cloud provider选项文档中的步骤设置Kubernetes环境.
2. 所有主机必须是AWS EC2实例并且拥有正确的IAM策略。

在Kubernetes中使用EBS数据卷的操作可以分为两种: 静态初始化, 和通过storage classes动态初始化。

静态初始化

在Kubernetes环境中使用persistent volume之前，需要预先在Rancher主机所在的AWS区域和可用区中创建EBS卷。为了使用persistent volume, 你需要先创建persistent volume资源对象。

示例pv-ebs.yml文件

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ebs
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  awsElasticBlockStore:
    fsType: ext4
    # The EBS volume ID
    volumeID: <VOLUME_ID_IN_EBS>
```

通过kubectl客户端, 我们可以使用以下命令在Kubernetes中创建persistent volume。 文件中的需要被替换为相应的EBS volume id。 记住，你可以通过为本地主机配置kubectl在本机使用kubectl客户端或者通过UI界面中的Kubernetes -> kubectl 命令行界面使用kubectl客户端。

```
$ kubectl create -f pv-ebs.yml
```

创建persistent volume之后, 你可以创建persistent volume claim，用于请求创建的persistent volume资源。

```
Example pvc-ebs.yml

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvs-ebs
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      release: "stable"
```

在创建了persistent volume和persistent volume claim之后, 你可以通过以下命令确认persistent volume已绑定。

$ kubectl get pv,pvc 创建一个POD使用PERSISTENT VOLUME CLAIM 创建persistent volume claim之后, 你只需要创建一个pod来使用它。

示例pod-ebs.yml文件

```
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: ebs
  volumes:
    - name: ebs
      persistentVolumeClaim:
        claimName: pvc-ebs
```

在pod使用了相应的卷之后，AWS中卷的状态应当从可用变为使用中。

```
$ kubectl get pods,pv,pvc
NAME       READY     STATUS    RESTARTS   AGE
po/mypod   1/1       Running   0          3m

NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS    CLAIM             REASON    AGE
pv/ebs    1Gi        RWO           Recycle         Bound     default/myclaim             3m

NAME          STATUS    VOLUME    CAPACITY   ACCESSMODES   AGE
pvc/pvc-ebs   Bound     pv1       1Gi        RWO           3m
```

动态创建

在Kubernetes中使用EBS的另外一种方法是动态创建, 这种方式使用StorageClass自动创建并挂载数据卷到pods中。在我们的示例中, storage class将指定AWS作为storage provider并使用gp2类型以及us-west-2a可用区。

示例storage-class.yml文件

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  zone: us-west-2a
```

创建POD使用STORAGE CLASSES

你可以在任何pod中使用storage class从而使Kubernetes自动创建新的卷并挂载到pod中。

```
{
  "kind": "PersistentVolumeClaim",
  "apiVersion": "v1",
  "metadata": {
    "name": "claim2",
    "annotations": {
      "volume.beta.kubernetes.io/storage-class": "standard"
    }
  },
  "spec": {
    "accessModes": [
      "ReadWriteOnce"
    ],
    "resources": {
      "requests": {
        "storage": "1Gi"
      }
    }
  }
}
```

通过在pod中使用这种请求方式，你将看到一个新的pv被创建并自动和pvc绑定:

```
$ kubectl get pv,pvc,pods
NAME                                          CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS    CLAIM             REASON    A
GE
pv/pvc-36fcf5dd-f476-11e6-b547-0275ac92095a   1Gi        RWO           Delete          Bound     default/claim2              1
m

NAME          STATUS    VOLUME                                     CAPACITY   ACCESSMODES   AGE
pvc/claim2    Bound     pvc-36fcf5dd-f476-11e6-b547-0275ac92095a   1Gi        RWO           1m

NAME       READY     STATUS    RESTARTS   AGE
po/nginx   1/1       Running   0          29s
```

Kubernetes 插件

Rancher 会自动安装好 Kubernetes 插件以提高用户使用Kubernetes的体验. 如果你想要关闭这一功能, 你需要 配置 Kubernetes 来禁用插件的自动安装.

- Helm - Kubernetes的软件包管理工具
- Dashboard - Kubernetes的Web仪表板
- SkyDNS - Kubernetes的DNS服务器

HELM

Helm是一个用来提高Kubernetes中应用安装及管理效率的工具。 它通过将复杂应用打包为Charts的方式来帮助你在Kubernetes中运行应用。一个Chart是描述Kubernetes资源的一组文件，可以用来部署简单的Pods，也可以用来部署复杂的应用（例如一个完整的web应用栈）。

Helm包括两个部分，一个叫Tiller的服务端和一个叫Helm的客户端。Tiller由Rancher自动在kube-system 命名空间启动. Helm客户端安装在集成的kubectl CLI中.

在RANCHER中使用HELM 在Rancher中安装Kubernetes时，你需要配置你的 Kubernetes 为启用插件，这样Helm可以自动安装好。等Kubernetes安装完成后，你可以通过Rancher在UI提供的Shell直接使用Helm或者配置你的工作站来使用helm.

在RANCHER的UI中使用 HELM

Rancher提供对一个kubectl实例直接的shell访问，可以用它来管理Kubernetes集群和应用。要想使用这个shell，点击 Kubernetes -> CLI. 这个shell中自动安装好了Helm客户端，可以直接使用Helm的命令。

在你的工作站使用 HELM

在Rancher中安装好Kubernetes之后，通过 Kubernetes -> **CLI** 菜单下生成一个配置文件，你可以在你的工作站配置 kubectl。

要在你的工作站使用Helm，你需要根据要求安装Helm工具。请参考官方的 Helm安装文档 来安装 Helm.

在你的工作站，验证可以通过你安装的Helm客户端和Tiller通信：

```
$ helm init
$HELM_HOME has been configured at $HOME/.helm.
Warning: Tiller is already installed in the cluster. (Use --client-only to suppress this message.)
Happy Helming!

$ helm version
Client: &version.Version{SemVer:"v2.1.3", GitCommit:"5cbc48fb305ca4bf68c26eb8d2a7eb363227e973", GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.1.3", GitCommit:"5cbc48fb305ca4bf68c26eb8d2a7eb363227e973", GitTreeState:"clean"}
```

升级 HELM

每个Rancher版本都有一个特定的版本，基于上游Kubernetes附加组件的当前状态，但是您可以根据您的需求随时更新helm组件。

Helm有两个版本的组件，客户端（ Helm ）和服务器（ tiller ）。为了获得最好的结果，建议运行相同版本的客户端和服务器。升级客户端可以通过在本地系统上下载一个新的二进制文件来完成。然后，客户端可以通过运行以下操作来将服务器组件升级到匹配的版本：

```
$ helm init --upgrade

Tiller (the helm server side component) has been upgraded to the current version.
Happy Helming!
NOTE The CLI built into the Rancher UI has its own copy of the helm client, so upgrading the server without upgrading this cli
ent may break functionality with the UI based CLI until Rancher releases an updated version.
```

使用 HELM

如同其他包管理工具，在使用Helm的时候我们应该确认Charts更到最新。

```
> helm repo update

Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "stable" chart repository
Update Complete. â Happy Helming!â
Kubernetes有自己的官方Helm charts，可以直接拿来使用。接下来我们通过一个例子来演示如何在Kubernetes安装一个Wordpress Chart。
```

> 注意： 因为Helm中的Charts在持续更新，我们在这的版本不一定跟最新的版本相匹配。你应该安装最新的版本。

首先，我们通过使用 helm search 来查找可用的 Charts.

```
> helm search
NAME                    VERSION DESCRIPTION
stable/drupal           0.3.4   One of the most versatile open source content m...
stable/jenkins          0.1.1   A Jenkins Helm chart for Kubernetes.
stable/mariadb          0.5.2   Chart for MariaDB
stable/mysql            0.1.1   Chart for MySQL
stable/redmine          0.3.3   A flexible project management web application.
stable/wordpress        0.3.1   Web publishing platform for building blogs and ...
```

你可以立即从Helm的仓库中安装Chart，不过我们先获取Wordpress Chart，检查看看部署这一Chart可用的选项。

```
$ helm fetch stable/wordpress
$ tar xzvf wordpress-*.tgz
$ cd wordpress
```

你可以通过values.yaml文件来查看对Wordpress Chart可用的选项。这个文件包括了Chart中使用到的所有变量。用你喜欢的文本编辑器打开这个文件，你可以看到如下所示多个配置。

```
image: bitnami/wordpress:4.7-r0
imagePullPolicy: IfNotPresent
wordpressUsername: user
# wordpressPassword:
wordpressEmail: user@example.com
wordpressFirstName: FirstName
wordpressLastName: LastName
wordpressBlogName: User's Blog!
....
```

在Wordpress的这个文件中，你会看到持久存储默认是激活的。默认配置使用一个叫default的storage class 来动态提供持久存储卷。想要在Rancher的Kubernetes中使用动态供给持久存储，请阅读关于 在 Rancher 中使用持久存储的文档。

如果在用例中持久存储不是必须的，我们可以在安装这个Chart的时候禁用持久存储。

```
$ helm install --name wordpress --set mariadb.persistence.enabled=false,persistence.enabled=false stable/wordpress
NAME:   wordpress
LAST DEPLOYED: Fri Apr 21 16:46:18 2017
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME                TYPE      DATA      AGE
wordpress-mariadb   Opaque    2         2s
wordpress-wordpress   Opaque    2         2s

==> v1/ConfigMap
NAME                DATA      AGE
wordpress-mariadb   1         2s

==> v1/Service
NAME                  CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
wordpress-wordpress   10.43.218.155   <pending>     80:32247/TCP,443:31795/TCP   2s
wordpress-mariadb   10.43.57.189   <none>    3306/TCP   2s

==> extensions/Deployment
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
wordpress-wordpress   1         1         1            0           2s
wordpress-mariadb   1         1         1         0         2s


NOTES:
1. Get the WordPress URL:

  NOTE: It may take a few minutes for the LoadBalancer IP to be available.
        Watch the status with: 'kubectl get svc --namespace default -w wordpress-wordpress'

  export SERVICE_IP=$(kubectl get svc --namespace default wordpress-wordpress -o jsonpath='{.status.loadBalancer.ingress[0].ip
}')
  echo http://$SERVICE_IP/admin

2. Login with the following credentials to see your blog

  echo Username: user
  echo Password: $(kubectl get secret --namespace default wordpress-wordpress -o jsonpath="{.data.wordpress-password}" | base6
4 --decode)
```

你会注意到有一个 NOTES 的小节来帮助你使用安装的Wordpress Chart。这个 NOTES 提供的信息包括如何获取WordPress URL以及如何用默认的认证去登陆。

```
$ export SERVICE_IP=$(kubectl get svc --namespace default wordpress-wordpress -o jsonpath='{.status.loadBalancer.ingress[0].ip
}')
$ echo http://$SERVICE_IP/admin
http://x.x.x.x/admin

$ echo Username: user
$ echo Password: $(kubectl get secret --namespace default wordpress-wordpress -o jsonpath="{.data.wordpress-password}" | base6
4 --decode)
Username: user
Password: 58wYgIT06m
```

通过访问展示出来的URL，你可以开始使用在Kubernetes中搭好的Wordpress并用提供的认证去登陆。

配合 HELM CHART 使用持久存储

如果你在Rancher中 配置了持久存储，你可以在Kubernetes中创建 storage classes。在下一个例子中，我们会使用同一个 Wordpress Chart并选择使用AWS上的持久存储。

如下几个先决条件需要配置好：

- Kubernetes 设置为使用 aws 作为 云提供商cloud provider.
- 所有的 hosts主机 通过AWS EC2 加到Rancher，并设置了正确的IAM策略。
- 需要创建一个名为 default的 storage class 并配置为使用AWS 存储卷。

在正确启动Kubernetes，添加完storage class之后，你可以部署这个Wordpress Chart并使用你的持久存储。

```
$ helm install --name wordpress stable/wordpress
NAME:   wordpress
LAST DEPLOYED: Fri Apr 21 17:12:35 2017
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME                TYPE      DATA      AGE
wordpress-mariadb   Opaque    2         2s
wordpress-wordpress   Opaque    2         2s

==> v1/ConfigMap
NAME                DATA      AGE
wordpress-mariadb   1         2s

==> v1/Service
NAME                CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
wordpress-mariadb   10.43.101.232   <none>        3306/TCP   2s
wordpress-wordpress   10.43.250.75   <pending>   80:30296/TCP,443:30094/TCP   2s

==> extensions/Deployment
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
wordpress-wordpress   1         1         1            0           2s
wordpress-mariadb   1         1         1         0         2s

==> v1/PersistentVolumeClaim
NAME                            STATUS    VOLUME                                     CAPACITY   ACCESSMODES   AGE
wordpress-wordpress-wordpress   Bound     pvc-f396de3d-26a4-11e7-9213-02ee7a4cff8e   8Gi        RWO           2s
wordpress-wordpress-apache   Bound     pvc-f3986989-26a4-11e7-9213-02ee7a4cff8e   1Gi       RWO       2s
wordpress-mariadb   Bound     pvc-f399feb7-26a4-11e7-9213-02ee7a4cff8e   8Gi       RWO       2s


NOTES:
1. Get the WordPress URL:

  NOTE: It may take a few minutes for the LoadBalancer IP to be available.
        Watch the status with: 'kubectl get svc --namespace default -w wordpress-wordpress'

  export SERVICE_IP=$(kubectl get svc --namespace default wordpress-wordpress -o jsonpath='{.status.loadBalancer.ingress[0].ip
}')
  echo http://$SERVICE_IP/admin

2. Login with the following credentials to see your blog

  echo Username: user
  echo Password: $(kubectl get secret --namespace default wordpress-wordpress -o jsonpath="{.data.wordpress-password}" | base6
4 --decode)
```

你可以看到使用持久存储的Chart中的新资源被创建。

```
==> v1/PersistentVolumeClaim
NAME                            STATUS    VOLUME                                     CAPACITY   ACCESSMODES   AGE
wordpress-wordpress-wordpress   Bound     pvc-f396de3d-26a4-11e7-9213-02ee7a4cff8e   8Gi        RWO           2s
wordpress-wordpress-apache   Bound     pvc-f3986989-26a4-11e7-9213-02ee7a4cff8e   1Gi       RWO       2s
wordpress-mariadb   Bound     pvc-f399feb7-26a4-11e7-9213-02ee7a4cff8e   8Gi       RWO       2s
```

你可以验证persistent volumes 已经被创建。

```
bash $ kubectl get pv NAME CAPACITY ACCESSMODES RECLAIMPOLICY STATUS CLAIM REASON AGE pvc-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 8Gi RWO Delete Bound default/wordpress-wordpress-wordpress 4m pvc-yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy 1Gi RWO Delete Bound d
efault/wordpress-wordpress-apache 4m pvc-zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz 8Gi RWO Delete Bound default/wordpress-mariadb 4
m
```

在这个例子中，我们选择了AWS作为 云提供商cloud provider，它也会在AWS上创建一个ELB来用作你的Wordpress Chart的负载均衡器。

```
$ export SERVICE_IP=$(kubectl get svc --namespace default wordpress-wordpress -o jsonpath='{.status.loadBalancer.ingress[0].ho
stname}')
$ echo http://$SERVICE_IP/admin
http://xxxxxxxxxxxxx.us-west-2.elb.amazonaws.com/admin
```

SKYDNS

在Rancher中，每个服务（service）会被赋予一个名字。其他服务可以使用DNS服务名来和一个服务通信。 DNS服务名为 ..svc.cluster.local.

使用上述Helm例子中启动的Wordpress应用，你可以获得Wordpress服务的名字和命名空间（namespace）。

```
> kubectl get services

NAME                  CLUSTER-IP      EXTERNAL-IP        PORT(S)                      AGE
kubernetes            10.43.0.1       <none>             443/TCP                      22d
wordpress-mariadb     10.43.101.232   <none>             3306/TCP                     1h
wordpress-wordpress   10.43.250.75    xxxxxxxx...   80:30296/TCP,443:30094/TCP   1h

> kubectl describe services/wordpress-wordpress
Name:           wordpress-wordpress
Namespace:      default
Labels:         app=wordpress-wordpress
                  chart=wordpress-0.4.2
                  heritage=Tiller
                  release=wordpress
Selector:       app=wordpress-wordpress
Type:           LoadBalancer
IP:         10.43.250.75
LoadBalancer Ingress:   xxxxxxxxxxxxxxxxxxx.elb.amazonaws.com
Port:           http    80/TCP
NodePort:       http    30296/TCP
Endpoints:      10.42.122.207:80
Port:           https   443/TCP
NodePort:       https   30094/TCP
Endpoints:      10.42.122.207:443
Session Affinity:   None
No events.
```

这个Wordpress应用被命名为 wordpress。在这个例子中，DNS服务名为 wordpress-wordpress.default.svc.cluster.local。

Ingress Support

在开始使用 Kubernetes Ingress资源之前，你需要准备一个Kubernetes环境 。并且建议在本地电脑上设置kubectl ，以便更容易地将Kubernetes的资源发布到Rancher中。或者，你可以使用Rancher UI提供的shell来启动资源。

Kubernetes Ingress资源可以支持你选择的任何负载均衡器类型，因此，为了利用Rancher的负载均衡功能，我们引入了Rancher Ingress控制器的概念。ingress控制器是ingress-controller服务的一部分，它做为Kubernetes系统栈的组件被部署。

ingress控制器管理着Rancher负载均衡器的创建/迁移/更新。每个负载均衡器的创建/删除/更新都是基于 Kubernetes ingress 资源。

如果一个ingress被更新或服务端点（Service endpoint）被更改，ingress控制器将更新相应的Rancher负载均衡器，使其与ingress的变化相对应。

同理，如果ingress被移除，Rancher负载均衡器也会被移除。如果一个ingress的后端服务发生了变化(例如，当复制控制器被放大或缩小或重新创建一个pod时)，Rancher负载均衡器也将相应地更新。ingress控制器确保了Rancher负载均衡器与Kubernetes的ingress和后端服务相匹配。

目前的局限性

- Ingress资源只能通过kubectl工具添加

RANCHER INGRESS 控制器

Rancher ingress控制器利用Rancher中现有的负载均衡功能，将Kubernetes ingress的内容转换到Rancher的负载均衡器。

ingress controller 功能:

- 监听Kubernetes服务器事件;
- 部署负载平衡器，并将其与Ingress中定义的路由规则进行适配;
- 通过配置Ingress Address 字段来做为负载均衡器的公共接入地址。

在RANCHER中创建INGRESS资源

设置一个NGINX服务示例 在配置任何ingress之前, 需要在Kubernetes中创建服务. 首先在Kubernetes环境中添加一个服务和复制控制器（Replication Controller）。

这里添加单个的nginx服务到Kubernetes中。

示例：nginx-service.yml:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  labels:
    k8s-app: nginx-service
spec:
  ports:
  - port: 90
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    k8s-app: nginx-service
---
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx-service
spec:
  replicas: 1
  selector:
    k8s-app: nginx-service
  template:
    metadata:
      labels:
        k8s-app: nginx-service
    spec:
      terminationGracePeriodSeconds: 60
      containers:
      - name: nginx-service
        image: nginx:latest
        ports:
        - containerPort: 80
```

让我们使用kubectl将nginx服务发布到Kubernetes。请记住, 你也可以为本地机器配置kubectl ，或者在Kubernetes/kubectl的UI中使用shell。

```
$ kubectl create -f nginx-service.yml
service "nginx-service" created
replicationcontroller "nginx-service" created
```

配置单个INGRESS资源

你可以为单个服务设置单一的ingress资源。

示例：simple-ingress.yml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: simplelb
spec:
  backend:
    serviceName: nginx-service
    servicePort: 90
```

让我们使用kubectl创建ingress。在创建ingress后, ingress 控制器将触发创建一个负载均衡器并且可以在Kubernetes -> System 标签栏中看到创建的 kubernetes-ingress-lbs 应用栈 .

默认情况下，负载均衡器服务只有一个实例被部署。 通过kubectl, 可以看到 ingress被创建, 但通过UI只能看到负载均衡器应用. ingress控制器已经完成了所有ingress转换到Rancher负载均衡器的请求。

```
$ kubectl create -f simple-ingress.yml
ingress "simplelb" created
$ kubectl get ingress
NAME          RULE      BACKEND            ADDRESS          AGE
simplelb      -         nginx-service:80   1.2.3.4          5m
```

ingress中的address将是负载均衡器服务启动的公共接入地址。如果负载均衡器被移动到不同的主机并得到不同的公共端点（public endpoint），则ingress地址将被更新。

你可以在缺省端口80 (例如：http://1.2.3.4:80)或直接访问地址 (例如：http://1.2.3.4)来访问你的应用。

配置多个服务

如果你希望ingress做为多个服务的入口点，那么需要使用主机名路由规则来配置它，它允许将host/path 的路由添加到服务。

让我们添加多个服务到 Kubernetes中.

示例：multiple-nginx-services.yml:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service-1
  labels:
    k8s-app: nginx-service-1
spec:
  ports:
  - port: 90
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    k8s-app: nginx-service-1
---
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx-service-1
spec:
  replicas: 1
  selector:
    k8s-app: nginx-service-1
  template:
    metadata:
      labels:
        k8s-app: nginx-service-1
    spec:
      terminationGracePeriodSeconds: 60
      containers:
      - name: nginx-service-1
        image: nginx:latest
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service-2
  labels:
    k8s-app: nginx-service-2
spec:
  ports:
  - port: 90
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    k8s-app: nginx-service-2
---
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx-service-2
spec:
  replicas: 1
  selector:
    k8s-app: nginx-service-2
  template:
    metadata:
      labels:
        k8s-app: nginx-service-2
    spec:
      terminationGracePeriodSeconds: 60
      containers:
      - name: nginx-service-2
        image: nginx:latest
        ports:
        - containerPort: 80
```

让我们用kubectl将我们的服务和复制控制器发布到Kubernetes中，请记住，你可以为本地机器配置kubectl，也可以在Kubernetes-kubectl的UI中使用shell。

```
$ kubectl create -f multiple-nginx-services.yml
service "nginx-service-1" created
replicationcontroller "nginx-service-1" created
service "nginx-service-2" created
replicationcontroller "nginx-service-2" created
```

基于主机路由配置 INGRESS 资源

示例：host-based-ingress.yml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: host-based-ingress
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          serviceName: nginx-service-1
          servicePort: 90
  - host: bar.foo.com
    http:
      paths:
      - backend:
          serviceName: nginx-service-2
          servicePort: 90
```

让我们使用kubectl创建ingress。在创建ingress后, ingress 控制器 将触发创建一个负载均衡器并且可以在Kubernetes -> System 标签栏中看到创建的 kubernetes-ingress-lbs 应用栈 .

默认情况下，负载均衡器服务只有一个实例被部署。 通过ingress，负载均衡器将有一个以主机名命名的路由规则被创建。

通过kubectl, 可以看到 ingress 被创建, 但通过UI只能看到负载均衡器应用. ingress控制器已经完成了所有ingress转换到Rancher负载均衡器的请求。

```
$ kubectl create -f host-based-ingress.yml
ingress "host-based-ingress" created
$ kubectl get ingress
NAME                    RULE            BACKEND             ADDRESS     AGE
host-based-ingress      -                                   1.2.3.4     20m
                        foo.bar.com
                                        nginx-service-1:80
                        foo1.bar.com
                                        nginx-service-2:80
```

与单个ingress负载均衡器类似，这个负载均衡器将在Kubernetes/System选项卡中位于Kubernetes-lbs的应用栈中。

ingress中的地址将是负载均衡器服务启动的公共接入地址。.如果负载均衡器被移动到不同的主机并得到不同的公共端点，则ingress地址将被更新

你可以在缺省端口80 (例如：http://1.2.3.4:80)或直接访问地址 (例如：http://1.2.3.4)。来访问你的应用。

使用基于路径的路由配置INGRESS资源

如果有多个想要做负载均衡的服务，那么可以向ingress添加基于路径的路由。

示例：path-based-ingress.yml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: path-based-ingress
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: nginx-service-1
          servicePort: 90
      - path: /bar
        backend:
          serviceName: nginx-service-2
          servicePort: 90
```

让我们使用kubectl创建ingress。在创建ingress后, ingress 控制器 将触发创建一个负载均衡器并且可以在Kubernetes -> System 标签栏中看到创建的 kubernetes-ingress-lbs 应用栈。默认情况下，负载均衡器服务只有一个实例被部署。负载均衡器将拥有从ingress创建的主机名路由规则。

通过kubectl, 可以看到 ingress 被创建, 但是UI中只能看到负载均衡器应用. ingress控制器已经完成了所有在ingress中的转换到Rancher负载均衡器的请求。

```
$ kubectl create -f path-based-ingress.yml
ingress "path-based-ingress" created
$ kubectl get ingress
NAME                   RULE            BACKEND             ADDRESS           AGE
path-based-ingress     -                                   1.2.3.4           15s
                       foo.bar.com
                       /foo            nginx-service1:80
                       /bar            nginx-service2:80
```

与单个的ingress负载均衡器类似，这个负载均衡器将在Kubernetes-System选项卡中位于Kubernetes-lbs的堆栈中。

ingress中的address将是负载均衡器服务启动的公共接入地址。如果负载均衡器被移动到不同的主机并得到不同的公共端点，则ingress地址将被更新。

要访问你的应用程序，可以在缺省端口80 (例如：http://1.2.3.4:80)或地址直接访问地址(例如：http://1.2.3.4)。

KUBERNETES INGRESS的负载均衡器选项

默认情况下，Kubernetes ingress将在默认80/443端口上使用http/https部署1个负载均衡器实例。Rancher可支持多组负载均衡，用户可以根据需要自行制定端口。扩展ingress后，Kubernetes 配置的地址将被复制到所有负载均衡器。

示例:

- 多个负载均衡器使用不同的端口
- 使用 TLS
- 禁用HTTP
- 自定义HAProxy
- 所有主机运行负载均衡器
- 特定主机运行负载均衡器

> 注意： 如果增加ingress实例数量，你需要确保在Kubernetes环境中至少有同等数量的主机可用。

设置单个服务

在建立任何一个ingress之前，需要在Kubernetes中创建服务。首先在Kubernetes环境中添加一个服务和复制控制器。

这里向Kubernetes添加一个单个的nginx服务。

示例：nginx-service.yml:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  labels:
    k8s-app: nginx-service
spec:
  ports:
  - port: 90
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    k8s-app: nginx-service
---
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx-service
spec:
  replicas: 1
  selector:
    k8s-app: nginx-service
  template:
    metadata:
      labels:
        k8s-app: nginx-service
    spec:
      terminationGracePeriodSeconds: 60
      containers:
      - name: nginx-service
        image: nginx:latest
        ports:
        - containerPort: 80
```

使用kubectl，让我们将nginx服务发布到Kubernetes。请记住，你可以为本地机器配置kubectl，也可以在Kubernetes-kubectl的UI中使用shell。

```
$ kubectl create -f nginx-service.yml
service "nginx-service" created
replicationcontroller "nginx-service" created
```

示例：两个负载均衡器使用交替的端口

你可以在两个不同的主机上运行2个负载均衡器启动一个ingress，它使用一个99端口代替默认的80端口。为了让这个ingress正常工作，你的kubernetes环境至少需要两台拥有99端口的主机。

示例：scaled-ingress.yml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: scaledlb
  annotations:
    # Scaling to 2 load balancer instances
    scale: "2"
    # Using a different port (not 80)
    http.port: "99"
spec:
  backend:
    serviceName: nginx-service
    servicePort: 90
```

让我们使用kubectl创建ingress。在创建ingress之后，ingress控制器将触发在Kubernetes-System选项卡中创建的负载均衡器服务，并在Kubernetes-ingss-lbs堆栈中可见。由于在ingress中scale 设置为2，所以在负载均衡器服务中会有两个负载均衡器实例被部署。。

在kubectl中，你可以看到ingress创建，但是UI只显示负载均衡器。ingress控制器已经完成了在ingress转化到Rancher负载均衡器的所有请求。因为有两个负载均衡器，需要在ingress中设置两个地址。

```
$ kubectl create -f scaled-ingress.yml
ingress "host-based-ingress" created
$ kubectl get ingress
NAME       RULE      BACKEND            ADDRESS             AGE
simplelb   -         nginx-service:90   1.2.3.4,5.6.7.8     41s
```

示例：使用TLS

如果你想要在Kubernetes中使用TLS，那么你需要将证书添加到Rancher中。在Rancher中添加的证书可以用于为TLS终端提供一个安全的ingress。

假设我们添加了一个名为foo的证书

示例：tls-ingress.yml 使用这个 foo 证书。

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: tlslb
  annotations:
    https.port: "444"
spec:
  tls:
   - secretName: foo
  backend:
    serviceName: nginx-service
    servicePort: 90
```

让我们使用kubectl创建ingress。在创建ingress之后，ingress控制器将触发在Kubernetes-System选项卡中创建的负载均衡器服务，并在Kubernetes-ingss-lbs堆栈中可见。默认情况下，负载均衡器服务只有一个负载均衡器的实例被部署。

在kubectl中，你可以看到ingress被创建，但是UI只显示负载均衡器应用。ingress控制器已经完成了在ingress中转换到Rancher负载均衡器所有请求的。

禁用HTTP

默认情况下，即使使用了TLS，端口80也是可以访问的。为了阻止80端口，你可以添加额外的参数（annotation） (allow.http: “false”)做为ingress模板的一部分。

示例：tls-ingress.yml and 禁用80 端口

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: tlslb
  annotations:
    https.port: "444"
    # Added to block HTTP
    allow.http: "false"
spec:
  tls:
   - secretName: foo
  backend:
    serviceName: nginx-service
    servicePort: 90
```

示例：用户定制

在Rancher中，我们的负载均衡器运行的是HAProxy软件。如果你想要定制负载均衡器配置文件的global 和defaults 部分，可以通过ingress注释(annotations)来配置它们。

示例：custom-ingress.yml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: customlb
  annotations:
    # Customizing HAProxy in the load balancer
    config: "defaults\nbalance source\ntimeout server 70000\nglobal\nmaxconnrate 60"
spec:
  backend:
    serviceName: nginx-service
    servicePort: 80
```

在配置中，默认和全局关键字标识可自定义的部分，后面应该紧跟着有新的换行。 这些部分中的每个参数都应该跟着一条新的换行。

示例:在所有主机上运行负载均衡器 在当前环境中，可以让负载均衡器在所有主机上运行。这些全局负载均衡器可以使用注释（annotations）进行调度，i.e. io.Rancher.scheduler.global: "true".

示例：global-ingress.yml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: globallb
  annotations:
    # Create load balancers on every host in the environment
    io.Rancher.scheduler.global: "true"
spec:
  backend:
    serviceName: nginx-service
    servicePort: 80
```

示例：在特定主机上运行负载均衡器

在当前环境中，你可以将负载均衡器安排到某个特定主机上。为了将负载均衡器安排到特定的主机上，你需要向主机添加标签。主机上的标签是一个键值对，比如你可以给主机设置标签为foo=bar。主机添加标签之后，你需要使用参数annotation，（如Rancher.scheduler.affinity.host_label: “foo=bar”`）以使负载均衡器容器安排到标记的主机上。

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: scheduledlb
  annotations:
    # Search for a host that has label foo=bar and schedule the load balancer on that host.
    io.Rancher.scheduler.affinity.host_label: "foo=bar"
spec:
  backend:
    serviceName: nginx-service
    servicePort: 80
```

例：将流量导向同一主机上的不同容器

你可以配置负载均衡器，将流量路由到与负载均衡器容器同一主机上的服务容器。如果主机上没有目标服务的容器，那么负载均衡器不会将任何流量路由到目标服务的其他容器，就像它们在其他主机上一样。为了配置负载均衡器，你需要使用annotation参数i.e. io.Rancher.lb_service.target: "only-local".

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: onlylocallb
  annotations:
    # Direct traffic to only containers that are on the same host as the load balancer container
    io.Rancher.lb_service.target: "only-local"
spec:
  backend:
    serviceName: nginx-service
    servicePort: 80
```

示例：在同一主机上对容器的流量进行优先级排序

对于一个多实例服务，你可以配置负载均衡器，使流量优先分配到与负载均衡器容器相同主机的服务容器中。如果主机上没有目标服务的容器，则负载均衡器将流量引导到目标服务所在的其他主机上。为了配置负载均衡器，你将使用一个annotation， i.e. io.Rancher.lb_service.target: "prefer-local".

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: preferlocallb
  annotations:
    # Prioritize traffic to containers that are on the same host as the load balancer container
    io.Rancher.lb_service.target: "prefer-local"
spec:
  backend:
    serviceName: nginx-service
    servicePort: 80
```

示例：粘滞会话

可以为Rancher负载均衡器中运行的HAProxy软件的配置粘性策略。

例如，你可以配置负载均衡器使来自相同源的流量路由到相同的容器。

为了配置负载均衡器,你需要使用一条包括粘性政策的annotation参数 (i.e. io.Rancher.stickiness.policy)。

可以设置一个HAProxy软件能够识别的关键参数 \n做为界限值。

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test
  annotations:
    # Configure stickiness policy
    io.Rancher.stickiness.policy: "name: testname\n cookie: cookie123\ndomain: test.domain"
spec:
  backend:
    serviceName: nginx-service
    servicePort: 80
```

升级Kubernetes

如果希望升级一个已有的Kubernetes部署为平面隔离部署, 请参考迁移部分的说明。

整个迁移流程由两个阶段组成。

1. 确保你的环境中拥有足够数量具备合适标签的主机。
2. 升级Kubernetes基础设施应用栈。

主机与标签

确认环境中拥有足够数量具备各平面标签的主机。你可以添加新的主机或使用已有主机。

1. 计算平面主机: 对于全部已存在的运行有kubelet和proxy容器的主机, 添加标签compute=true。 这些是当你运行kubectl get node命令时显示的已经注册到Kubernetes的节点。此步骤非常重要 ，如果主机没有这个标签，运行在主机上的Kubernetes pods将变为孤儿pods。如果环境中有运行kubelet和proxy容器的主机，你可以按照从计算平面移除主机中的步骤操作将其移除。你也可以增加更多的主机并按照为主机设置标签中的步骤为主机设置compute=true标签。
2. 数据平面主机: 拥有至少3台具有1 CPU, >=1.5GB RAM, >=20GB DISK的主机。如果你有一个已经运行的Kubernetes环境中包含etcd服务运行，请确保运行etcd服务的主机具有etcd=true标签。
3. 编排平面主机: 添加至少2台具有大于等于1 CPU且大于等于2GB RAM的主机。添加主机时，按照为主机设置标签中的步骤为主机设置orchestration=true标签。 你也可以仅使用一台主机进行部署，但这样将牺牲高可用性。当仅有的一台主机发生故障时，一些K8s功能例如API、在pods发生故障时进行重新调度等，将不会正常工作，直到一台新的主机被加入。

升级KUBERNETES

1. 在Kubernetes -> 基础设施应用下, 找到Kubernetes基础设施应用栈。点击有可用更新或已经是最新版本按钮。
2. 确认Kubernetes模板版本是你希望使用的版本。
3. 将Plane Isolation设置为required。
4. 选择cloud providers, backups, add-ons等的设置选项。
5. 点击升级以启动新版本的Kubernetes基础设施应用。
6. 在Kubernetes应用栈下的所有服务完成升级并显示为已升级状态后, 点击已升级: 完成升级按钮。

将运行PODS的主机从计算平面删除

警告: 如果你计划从计算平面删除任何主机，不属于任何replication controller或类似对象的裸pod将 不会 被重新调度。这是正常行为。

计算平面的主机运行kubelet和proxy容器。

1. 如果主机具有compute=true标签, 从主机上移除此标签。此操作将阻止kubelet和proxy容器被删除后重新被调度到这些主机。
2. 通过shell或者远程客户端使用kubectl工具, 运行kubectl delete node 。 你可以从Rancher用户界面或通过kubectl工具运行kubectl get node获取主机名 (也就是：)。在进行下一步可选操作之前，请等待所有pods被删除。
3. 可选: 如果你希望从主机中删除kubelet和proxy容器，你可以对kubelet和proxy服务进行一次升级，升级过程中无需修改任何设置。 在进行升级操作之前，确保主机上的所有pods都已经被删除。

删除 Kubernetes

如果你选择从你的环境删除 Kubernetes 编排，并想继续使用你的环境和主机，你会需要清理你的主机。

清理 PODS

在通过 Kubernetes -> Infrastructure Stacks 删除Kubernetes栈之前，你需要删除你的Pods。你可以使用Kubernetes来删除所有的结点然后等待。

```
$ kubectl delete node --all
```

清理持久化数据

在删除了 Kubernetes 基础设施服务之后，持久化数据依然会保留在主机上。

清理主机

对每台跑过 etcd 服务的主机，有几样东西需要通过执行进每台主机做清理：

- 一个命名存储卷 etcd: 通过执行 docker volume rm etcd 来删除该卷。
- 默认情况下， 备份 是被激活的，备份数据存储在 /var/etcd/backups。 通过执行 rm -r /var/etcd/backups/* 删除备份。

清理持久化数据的场景

- 当一台主机需要被重新添加到另一个Kubertes运行的环境时。
- 当Kubernetes基础设施被删除并在同一组主机上通过应用商店（catalog）重建时。

> 注意： 如果你正要清理持久化数据，你需要在删除Kubernetes基础设施之前先删掉 pods。否则Pods会在主机上留存。

不做清理的后果

如果不做清理，来自先前保存的 etcd 存储卷会被新的 Kubernetes 基础设施使用。这样会导致诸如以下的严重问题：

- 你可能不想重建所有先前存在的Kubernetes应用，但当Kubernetes API service连到原来保存的etcd数据时这一情况就会发生。
- 重要事项： 每个新的Kubernetes API service 会从Rancher获得一个新的证书。这会导致所有先前存在etcd存储卷中的 Kubernetes 密钥过期。这意味着所有需要和Kuvernetes API service通信的应用（例如：dashboard， helm，heapster以及其他需要这种通信的应用）都会出故障。解决办法是要使用旧的密钥集合重建应用的Pods。

Mesos

在Rancher中部署Mesos，你首先需要添加一个新的环境，这个环境需要环境模版中设定容器编排引擎是 Mesos。

创建一个MESOS环境

在左上角的环境的下拉菜单中，点击环境管理。通过点击添加环境去创建一个新的环境，需要填写名称，描述（可选），并选择Mesos作为编排引擎的环境模版。如果启用了访问控制，你可以在环境中编辑成员并选择他们的成员角色。所有被添加到成员列表的用户都能访问你的环境。

在创建Mesos环境后，你可以在左上角环境的下拉菜单中切换到你的环境，或者在环境管理页面中，在对应环境的下拉选项中点击 切换到此环境 。

> 注意： Rancher支持多种容器编排引擎框架，但Rancher目前不支持在已有运行服务的环境里切换容器编排引擎。

启动 MESOS

在Mesos环境被创建后，在你添加一台主机到这个环境之前，基础设施服务是不会启动的。Mesos服务需要至少3个主机节点。添加主机的过程与其他编排引擎相同。一旦第一个主机被添加成功，Rancher将会自动启动基础设施服务，包括Mesos的组件（例如mesos-master，mesos-agent以及zookeeper）。你可以在 Mesos页面看到部署的过程。

> 注意： 只有Rancher的管理员或者环境的所有者才能够看到基础设施服务。

使用 MESOS

当安装成功后，你可以通过以下的方式开始创建或者管理你的Mesos应用：

MESOS UI

你可以通过点击Mesos UI去管理Mesos。它会打开一个新的页面，你可以在这个不同的UI中管理Mesos。任何在该UI上创建的framework同样会在Rancher反映出来。

RANCHE应用商店

Rancher支持Mesos框架下的应用商店。通过点击 Mesos -> 启动一个framework按钮或者直接点击 应用商店 标签去选择使用一个framework。选择你想启动的framework并且点击查看详情。查看并且编辑Stack名称，Stack描述以及配置选项，最后点击 启动。

如果你想添加你自己的Mesos应用模版，你可以把他们添加到你的私有Rancher 应用商店并把你的模版放入mesos-templates的目录。

Swarm

在Rancher中部署Swarm，你首先需要添加一个新的环境。这个环境需要使用编排引擎为Swarm的环境模版进行创建。

需求

- Docker 1.13 以上
- 端口 2377 以及 2378 需要能在主机之间相互访问。

创建一个 SWARM 环境

在左上角的环境的下拉菜单中，点击环境管理。通过点击添加环境去创建一个新的环境，需要填写名称，描述（可选），并选择Swarm作为编排引擎的环境模版。如果启用了访问控制，你可以在环境中编辑成员并选择他们的成员角色。所有被添加到成员列表的用户都能访问你的环境。

在创建Swarm环境后，你可以在左上角环境的下拉菜单中切换到你的环境，或者在环境管理页面中，在对应环境的下拉选项中点击切换到此环境。

> 注意： Rancher支持多种容器编排引擎框架，但Rancher目前不支持在已有运行服务的环境里切换容器编排引擎。

启动 SWARM

在Swarm环境被创建后，在你添加一台主机到这个环境之前，基础设施服务是不会启动的。Swarm服务需要至少3个主机节点。添加主机的过程与其他编排引擎相同。一旦第一个主机被添加成功，Rancher将会自动启动基础设施服务，包括Swarm的组件（例如swarm与swarm-agent）。你可以在Swarm页面看到部署的过程。

> 注意： 请不要手动对docker daemon使用docker swarm命令。这可能与Rancher管理Swarm集群产生冲突。
>
> 注意： 只有Rancher的管理员或者环境的所有者才能够看到基础设施服务。

在命令行中使用CLI

在Rancher的UI里，你可以方便的连接到能够运行docker或者docker-compose命令的容器中。通过这个命令行你可以控制Swarm集群。

Windows (实验性)

在Rancher中部署Windows，你首先需要添加一个新的环境。这个环境需要使用编排引擎为Windows的环境模版进行创建。

目前Rancher只支持在特定主机上创建容器。大多数在Cattle和Rancher UI上有的特性目前都不支持Windows(如 服务发现, 健康检查, 元数据, DNS, 负载均衡)。

> 注意： Rancher中有一个默认的Windows环境模版。如果你想创建你自己的Windows环境模版，你需要禁用所有其它的基础设施服务，因为这些服务目前都不兼容Windows。

创建一个 WINDOWS 环境

在左上角的环境的下拉菜单中，点击环境管理。通过点击添加环境去创建一个新的环境，需要填写名称，描述（可选），并选择Windows作为编排引擎的环境模版。如果启用了访问控制，你可以在环境中编辑成员并选择他们的成员角色。所有被添加到成员列表的用户都能访问你的环境。

在创建Windows环境后，你可以在左上角环境的下拉菜单中切换到你的环境，或者在环境管理页面中，在对应环境的下拉选项中点击切换到此环境。

> 注意： Rancher支持多种容器编排引擎框架，但Rancher目前不支持在已有运行服务的环境里切换容器编排引擎。

添加 WINDOWS 主机

在Rancher中添加一个Windows主机，你需要先有一个运行了Docker的Windows Server 2016主机。

在基础架构->主机->添加主机页面，你可以按照指示用自动生产的自定义命令启动Rancher Agent。

在主机上，Rancher的二进制客户端会被下载到C:/Program Files/rancher目录，你可以在C:/ProgramData/rancher/agent.log找到客户端日志。

移除 WINDOWS 主机

作为一个Rancher中的主机，Rancher客户端已经被安装并且注册在了主机上。你必须在Windows主机上删除已经存在的Rancher客户端服务，你可以在 powershell 中运行如下命令来删除客户端。删除客户端后你可以在 Windows 环境中重用这个主机

```
& 'C:\Program Files\rancher\agent.exe' -unregister-service
```

WINDOWS 中的网络

我们默认支持NAT和透明网络.

目前，默认的 Windows 环境模版支持名为transparent的透明网络 这个透明网络是在运行 docker network create -d transparent transparent时创建的。

如果你要创建一个名字不是 transparent 的透明网络，你需要创建一个新的环境模版，并把 Windows 设为容器编排平台。选择Windows后，你可以点击 编辑配置 来更改透明网络的名字。你可以用这个环境模版创建一个环境。但在 Rancher UI 中这个透明网络的默认名字依然是 transparent。 因此，你需要把命令更新为 docker network create -d transparent <NEW_NAME_IN_TEMPLATE.

以原生 Docker 命令行的形式使用 Rancher

Rancher整合了原生Docker CLI，所以Rancher可以和其它DevOps和Docker工具同时使用。从高层次上，这意味着如果你在Rancher外启动、停止、或销毁一个容器，Rancher能检测到相应的变化和更新。

DOCKER 事件监控

Rancher通过实时监控所有主机上Docker事件来更新自己的状态。因此，当容器在Rancher外启动、停止、或销毁时(比如，直接在主机上执行docker stop sad_einstein)，Rancher能检测到这些变化和更新，并且相应地更新自己的状态。

> 注意： 目前的一个局限是：我们要等到容器启动(而不是创建)才能把容器导入到Rancher。 运行docker create ubuntu不会使相应的容器出现在Rancher UI，但运行docker start ubuntu或docker run ubuntu会.

你可以在主机上运行docker events来观察Docker事件流。这个事件流就是Rancher正在监听的事件流。

添加 DOCKER 直接启动的容器到 RANCHER 的网络

你可以在Rancher外启动容器，然后把它们添加到Rancher管理的网络中。 这意味着这些容器可以参与夸主机网络。要激活这个功能，创建容器时把io.rancher.container.network标签设为true 。下面是一个例子:

```
$ docker run -l io.rancher.container.network=true -itd ubuntu bash
```

请查阅Rancher中的网络了解更多关于Rancher管理的网络和夸主机网络的详情。

导入已有容器

Rancher支持在注册主机的时候倒入已有的容器。当你用Rancher UI上的自定义命令注册主机时， 任何当前在该主机上的容器都能被检测到，并且会被导入到Rancher。

周期性同步状态

除了实时监控Docker事件之外，Rancher还会周期性地和主机同步状态。每5秒钟主机就会向Rancher报告主机上的所有容器的状态，以保证Rancher中的状态和主机中的状态同步。这能够防止由于网络中断或服务器重启而导致Rancher遗漏某些Docker事件。 用这种方式来保持同步，主机上的容器的状态为单一数据源。比如, 如果Rancher认为一个容器正在运行，但它在主机上实际是停止的，Rancher会把容器的状态更新为停止，但Rancher不会尝试重启容器。

应用商店

Rancher提供了一个应用商店，通过商店中的应用程序模版的可以简化部署复杂应用的过程。进入应用商店页面，你可以看到所有的应用的应用模版。在官方认证应用商店下包含了Rancher认证的应用中的应用模版，在社区共享应用商店下包含了社区贡献的应用的应用模版。Rancher只维护官方认证的 认证 模版。

添加应用商店 CATALOGS

添加一个应用商店只需要添加一个应用商店名称，Git地址和一个分支名称。Git地址需要是git clone可以处理的。分支名称是你应用商店Git仓库的一个分支。如果不提供分支名称，我们默认会使用master。 你无论什么时候添加一个应用商店项目，它都会在你的应用商店中立刻生效。

目前有两类的应用商店可以添加到Rancher。全局应用商店以及环境应用商店。在全局的应用商店中，应用商店的模版会在所有的环境中。在环境应用商店中，应用商店的模版只会在对应的环境中生效。

Rancher的管理员可以在 系统管理 -> 系统设置 添加或者移除全局应用商店。

Rancher环境的用户可以在特定的Rancher环境中通过应用商店页面，右上角的管理去添加或者删除环境应用商店。

如果你在一个代理后方运行Rancher Server，你需要把HTTP代理设置在Rancher Server运行的环境变量中，使得Rancher应用商店可以正常运行。

应用商店中的基础设施服务

在环境模版里的基础设施服务来自Rancher中生效的应用商店中的 infra-templates 文件夹。

在应用商店中也可以看到这些服务。你可以看到全部的基础设施服务，即使这些服务可能和你当前所选择的编排引擎不兼容。建议在创建环境的时候通过环境模版来选择基础设施服务而不是从应用商店中手动部署这些服务。

部署应用模版

你可以按照关键字搜索你需要的模版，也可以通过分类来过滤你想要的模版。当你找到想要部署的应用后，点击启动并填写应用模版所需要的相关信息。

1. 默认下使用最新版本的应用模版，不过需要的话，你也可以选择旧的版本。
2. 输入一个应用的名字，按需填写应用的描述。
3. 填写配置选项，这些都是所选的应用模版需求的配置选项。
4. 点击创建去创建一个基于模版的应用。在创建应用之前，你可以通过点开预览窗口查看用于生成应用k的docker-compose.yml和rancher-compose.yml文件。 在你点击创建后，你的应用会马上被创建，但没有启动任何服务。在该应用的下拉菜单中点击启动服务去启动应用中的所有的服务。

更新应用模版

当一个新版本的应用模版被上传到应用商店后，Rancher会提示你，这个应用有新版本可以更新。当你点击有可用更新时，你可以选择需要升级的版本。请在升级前仔细检查，以发现升级带来的潜在风险。在选择一个版本后，点击保存之前需要检查配置选项。

在所有的服务都被更新后，应用和服务将会处于升级完成状态。如果你确认已经升级成功，最后只需要在应用的下拉菜单中点击完成升级。一旦完成升级，你将不能回退到旧版本。

> 注意： 目前Rancher升级应用时，需要确保应用中的服务都是active或者inactve状态的，如果有服务异常，需要手动停止该服务后再进行升级。

回滚

如果升级过程中出现了问题，并且你需要回退到之前的版本。你可以在应用的下拉菜单中选择回滚。

创建私有应用商店

私有应用商店要遵循应用商店服务指定的格式才可以正常的在Rancher中显示出来。

模板文件夹

应用商店将会根据环境中的调度引擎来显示不同的应用商店模板。

基于不同调度引擎的模板

- Cattle 调度引擎: 界面中的应用模板来自templates文件夹
- Swarm 调度引擎: 界面中的应用模板来自swarm-templates文件夹
- Mesos 调度引擎: 界面中的应用模板来自mesos-templates文件夹

基础设施服务模板

Rancher的基础设施服务可以从环境模板中启用, 这些模板来自于infra-templates文件夹。

这些服务从应用商店菜单中也可以看到, 你可以看到全部的基础设施服务包括那些和当前的编排调度引擎不兼容的服务. 我们建议从环境模板中启用基础设施服务，而不是直接从应用商店中启动。

目录结构

```
-- templates (Or any of templates folder)
  |-- cloudflare
  |   |-- 0
  |   |   |-- docker-compose.yml
  |   |   |-- rancher-compose.yml
  |   |-- 1
  |   |   |-- docker-compose.yml
  |   |   |-- rancher-compose.yml
  |   |-- catalogIcon-cloudflare.svg
  |   |-- config.yml
...
```

你需要创建一个templates文件夹作为根目录。templates文件夹将包含所有你想创建的应用的文件夹。我们建议为应用的文件夹起一个简单明了的名称。

在应用模板的文件夹中 (例如 cloudflare), 将包含该应用模板的各个版本所对应的文件夹。第一个版本为0，后续每个版本加1。比如，第二个版本应该在 1 文件夹中。每增加一个新版本的文件夹，你就可以使用这个新版本的应用模版来升级你的应用了。另外，你也可直接更新0文件夹中的内容并重新部署应用。

> 注意： 应用文件夹名称需要为一个单词，文件名中不能包含空格。针对名字比较长的应用请使用- 连接符。在config.yml中的 name你可以使用空格。

在RANCHER应用商店中展示出的RANCHER CATALOG文件

在应用商店模板的文件夹中，如何展示应用商店模板详细内容取决于两个文件。

- 第一个文件为 config.yml，包含了应用模板的详细信息。

```
name: # 应用商店模板名称
description: |
  # 应用商店模板描述
version: # 应用商店模板对应的版本
category: # 用于模板搜索时的目录
maintainer: # 该模板的维护者
license: # 许可类型
projectURL: # 和模板相关的URL
```

- 另外一个文件为该模板的logo。该文件的前缀必须为 catalogIcon-。

对于每一个应用模板，将至少有以下三个部分组成： config.yml, catalogIcon-entry.svg, 以及 0 文件夹 - 包含该模板的第一个版本配置。

RANCHER 应用商店模板

docker-compose.yml以及rancher-compose.yml为在Rancher中使用Rancher Compose启动服务必须提供的两个文件. 该文件将被保存在版本文件夹中。 (如： 0, 1, 等等)。

docker-compose.yml为一个可以使用 docker-compose up来启动的文件。 该服务遵循docker-compose格式。

rancher-compose.yml将包含帮助你自定义应用模板的其他信息。在catalog部分中，为了应用模板可以被正常使用，有一些选项是必填的。

你也可以创建一个可选的 README.md , 可以为模板提供一些较长的描述以及如何使用他们。

rancher-compose.yml

```
version: '2'
catalog:
  name: # Name of the versioned template of the Catalog Entry
  version: # Version of the versioned template of the Catalog Entry
  description: # Description of the versioned template of the Catalog Entry
  minimum_rancher_version: # The minimum version of Rancher that supports the template, v1.0.1 and 1.0.1 are acceptable inputs
  maximum_rancher_version: # The maximum version of Rancher that supports the template, v1.0.1 and 1.0.1 are acceptable inputs
  upgrade_from: # The previous versions that this template can be upgraded from
  questions: #Used to request user input for configuration options
```

对于 upgrade_from, 有三种值可以使用。

1. 只允许从某一个版本升级： "1.0.0"
2. 可以从高于或低于某一个版本升级： ">=1.0.0", "<=2.0.0"
3. 定义一个区间升级: ">1.0.0 <2.0.0 ||="">3.0.0"

> 注意： 如同例子中的配置，请确保你配置的版本号或版本范围带上双引号。

RANCHER-COMPOSE.YML中的问题部分

应用商店中questions 部分允许用户更改一个服务的一些配置选项。 其答案 将在被服务启动之前被预配置在 docker-compose.yml 中.

每一个配置选项都在rancher-compose.yml的 questions 部分配置.

```
version: '2'
catalog:
  questions:
    - variable: # A single word that is used to pair the question and answer.
      label: # The "question" to be answered.
      description: | # The description of the question to show the user how to answer the question.
      default: # (Optional) A default value that will be pre-populated into the UI
      required: # (Optional) Whether or not an answer is required. By default, it's considered `false`.
      type: # How the questions are formatted and types of response expected
```

类型 type 控制了问题如何在UI中展现以及需要什么样的答案。

合法的格式有:

- string UI中将显示文本框来获取答案，获取到的答案将被设置为字符串型格式。
- int UI中将显示文本框来获取答案，获取到的答案将被设置为整型格式。 UI会在服务启动前对输入进行校验。
- boolean UI中将通过单选按钮获取答案，获取到的答案将被格式化为true 或者 false。 如果用户选择了单选按钮，答案将被格式化为 true。
- password UI中将显示文本框来获取答案，获取到的答案将被设置为字符串型格式。
- service UI中将展示一个下拉框，所有该环境的服务都会显示出来。
- enum UI中将展示一个下拉框，options中的配置将会被展示出来。

```
version: '2'
catalog:
  questions:
    - variable:
      label:
      description: |
      type: enum
      options: # List of options if using type of `enum`
        - Option 1
        - Option 2
```

- multiline 多行文本框会被显示在UI中。

```
version: '2'
catalog:
  questions:
    - variable:
      label:
      description: |
      type: multiline
      default: |
        Each line
        would be shown
        on a separate
        line.
```

- certificate 该环境的所有可用证书都会显示出来。

```
version: '2'
catalog:
  questions:
    - variable:
      label:
      description: |
      type: certificate
```

基于YEOMAN的应用目录生成器

这里有一个基于Yeoman的开源项目, 可以被用于创建一个空的应用商店目录。

Rancher命令行界面

Rancher的命令行界面（CLI）是用来管理Rancher Server的工具。 使用此工具，你可以管理你的环境，主机，应用，服务和容器。

安装

二进制文件可以直接从UI下载。 该链接可以在UI中的页脚右侧找到。 我们有Windows，Mac和Linux的二进制文件。 你还可以查看我们CLI的发布页面 ，你可以从该页面直接下载二进制文件。

配置RANCHER命令行界面

有几种方法可以配置Rancher命令行界面与Rancher进行交互时使用的参数。这些参数包括Rancher URL和帐户API密钥等。帐户的API密钥可以在UI中的API中创建。

参数配置有如下的加载优先级。

1. 在执行rancher config时，你需要设置Rancher URL和API密钥。如果你有多个环境，那么你可以选择一个默认环境。
2. 你可以在环境变量中设置相关参数，这将覆盖rancher config中设置的值。
3. 你可以将参数值直接传递给Rancher命令行，那么这些值将覆盖其他方式配置的参数。

使用RANCHER配置命令

你可以运行rancher config来设置与Rancher Server连接的配置

```
$ rancher config
URL []: http://<server_ip>:8080
Access Key []: <accessKey_of_account_api_key>
Secret Key []:  <secretKey_of_account_api_key>
# If there are more than one environment,
# you will be asked to select which environment to work with
Environments:
[1] Default(1a5)
[2] k8s(1a10)
Select: 1
INFO[0017] Saving config to /Users/<username>/.rancher/cli.json
```

使用环境变量

你可以设置以下环境变量RANCHER_URL，RANCHER_ACCESS_KEY和RANCHER_SECRET_KEY。

```
# Set the url that Rancher is on
$ export RANCHER_URL=http://<server_ip>:8080
# Set the access key, i.e. username
$ export RANCHER_ACCESS_KEY=<accessKey_of_account_api_key>
# Set the secret key, i.e. password
$ export RANCHER_SECRET_KEY=<secretKey_of_account_api_key>
```

如果你的Rancher Server中有多个环境，你还需要设置一个环境变量来选择默认环境，即“RANCHER_ENVIRONMENT”。

```
# Set the environment to use, you can use either environment ID or environment name
$ export RANCHER_ENVIRONMENT=<environment_id>
```

可选参数传递

如果你选择不运行rancher config或设置环境变量，那么你可以传递相同的值作为rancher命令参数选项的一部分。

```
$ rancher --url http://server_ip:8080 --access-key <accessKey_of_account_api_key> --secret-key <secretKey_of_account_api_key>
--env <environment_id> ps
```

使用RANCHER命令行界面调试

当你使用Rancher命令行时，可以将环境变量“RANCHER_CLIENT_DEBUG”设置为“true”，这样当API被调用时，所有的CLI命令将打印出详细信息。

```
# Print verbose messages for all CLI calls
$ export RANCHER_CLIENT_DEBUG=true
```

如果你不想每个CLI命令都打印详细信息，请将环境变量“RANCHER_CLIENT_DEBUG”设置为“false”，然后将--debug传递给指定命令来获取详细消息。

```
$ rancher --debug env create newEnv
```

使用环境变量

如果你使用账户的API密钥，你将能够创建和更新环境。 如果你使用一个环境的API密钥，你将无法创建或更新其他环境，你将只能看到现有的环境。

```
$ rancher env ls
ID        NAME        STATE     CATALOG                           SYSTEM    DETAIL
1e1       zookeeper   healthy   catalog://community:zookeeper:1   false
1e2       Default     healthy                                     false
1e3       App1        healthy                                     false
```

使用指定的主机

有一些命令(比如说rancher docker和rancher ssh)需要选择指定的主机来使用。你可以设置一个环境变量来选择主机，即RANCHER_DOCKER_HOST，或者传递 –host参数来指定主机。

选择主机之前，你可以列出环境中的所有主机。

```
$ rancher hosts
ID        HOSTNAME      STATE     IP
1h1       host-1        active    111.222.333.444
1h2       host-3        active    111.222.333.445
1h3       host-2        active    111.222.333.446
```

现在你可以设置RANCHER_DOCKER_HOST环境变量，或者使用–host参数传入主机ID或主机名来选择不同的主机

```
# Set the host to always select host-1 (1h1)
$ export RANCHER_DOCKER_HOST=1h1
# List the containers running on host-1
$ rancher docker ps
# List the containers running on host-2
$ rancher --host host-2 docker ps
```

使用服务和容器

列出所有的服务

在你选择的环境中，你可以查看在环境中运行的所有服务。

```
$ rancher ps
ID   TYPE                NAME                IMAGE                          STATE     SCALE   ENDPOINTS            DETAIL
1s1  service             zookeeper/zk        rawmind/alpine-zk:3.4.8-4      healthy   3
1s2  service             Default/nginxApp    nginx                          healthy   1
1s4  service             App1/db1            mysql                          healthy   1
1s5  service             App1/wordpress      wordpress                      healthy   4
1s6  loadBalancerService  App1/wordpress-lb                                  healthy   1       111.222.333.444:80
```

列出所有的容器

同样你可以查看环境中的所有容器。

```
$ rancher ps -c
ID      NAME                       IMAGE                                STATE     HOST   IP              DOCKER         DETAIL
1i1     zookeeper_zk_zk-volume_1   rawmind/alpine-volume:0.0.1-1        stopped   1h1                    a92b6d3dad18
1i2     zookeeper_zk_zk-conf_1     rawmind/rancher-zk:0.3.3             stopped   1h1                    2e8085a4b517
1i3     zookeeper_zk_1             rawmind/alpine-zk:3.4.8-4            healthy   1h1    10.42.150.2     e3ef1c6ff70e
1i5     zookeeper_zk_zk-volume_2   rawmind/alpine-volume:0.0.1-1        stopped   1h2                    e716f562e0a4
1i6     zookeeper_zk_zk-conf_2     rawmind/rancher-zk:0.3.3             stopped   1h2                    5cd1cebea5a3
1i7     zookeeper_zk_2             rawmind/alpine-zk:3.4.8-4            healthy   1h2    10.42.88.102    21984a4445d1
1i9     zookeeper_zk_zk-volume_3   rawmind/alpine-volume:0.0.1-1        stopped   1h3                    7c614003f08c
1i10    zookeeper_zk_zk-conf_3     rawmind/rancher-zk:0.3.3             stopped   1h3                    53fb77cd8ae0
1i11    zookeeper_zk_3             rawmind/alpine-zk:3.4.8-4            healthy   1h3    10.42.249.162   84a80eb8e037
1i13    Default_nginxApp_1         nginx                                running   1h1    10.42.107.28    e1195a563280
1i15    App1_db1_1                 mysql                                running   1h3    10.42.116.171   0624e0a7f2fc
1i16    App1_wordpress_1           wordpress                            running   1h1    10.42.66.199    4bb77abebc08
1i17    App1_wordpress-lb_1        rancher/lb-service-haproxy:v0.4.2    healthy   1h2    10.42.199.163   5d3a005278d3
1i18    App1_wordpress_2           wordpress                            running   1h2    10.42.88.114    01ec967c49ac
1i19    App1_wordpress_3           wordpress                            running   1h3    10.42.218.81    3aae3fc6163a
1i20    App1_wordpress_4           wordpress                            running   1h1    10.42.202.31    0b67ef86db22
```

列出指定服务的容器

如果要查看特定服务的容器，可以通过添加服务ID或服务名称列出运行服务的所有容器。

```
$ rancher ps 1s5
ID      NAME               IMAGE       STATE     HOST      IP             DOCKER         DETAIL
1i16    App1_wordpress_1   wordpress   running   1h1       10.42.66.199   4bb77abebc08
1i18    App1_wordpress_2   wordpress   running   1h2       10.42.88.114   01ec967c49ac
1i19    App1_wordpress_3   wordpress   running   1h3       10.42.218.81   3aae3fc6163a
1i20    App1_wordpress_4   wordpress   running   1h1       10.42.202.31   0b67ef86db22
```

使用DOCKER COMPOSE文件启动简单的服务

要开始向Rancher添加服务时，你可以创建一个简单的docker-compose.yml文件，以及可选的rancher-compose.yml文件。 如果没有rancher-compose.yml文件，则所有服务的数量将默认为1。

docker-compose.yml示例

```
version: '2'
services:
  service1:
    image: nginx
```

rancher-compose.yml示例

```
version: '2'
services:
  # Reference the service that you want to extend
  service1:
    scale: 2
```

创建文件后，你可以在Rancher Server中启动对应的服务。

```
# Creating and starting a service without environment variables and selecting a stack
# If no stack is provided, the stack name will be the folder name that the command is running from
# If the stack does not exist in Rancher, it will be created
# Add in -d at the end to not block and log
$ rancher --url URL_of_Rancher --access-key <username_of_account_api_key> --secret-key <password_of_account_api_key> --env Def
ault up -s stack1 -d

# Creating and starting a service with environment variables already set
# Add in -d at the end to not block and log
$ rancher up -s stack1 -d

# To change the scale of an existing service, you can use stackName/serviceName or service ID
$ rancher scale Default/service1=3
$ rancher scale 1s4=5
```

使用RANCHER RUN来启动一个服务

你可以使用Docker CLI添加容器，也可以使用rancher run添加容器到Rancher中。

```
# Services should be stackName/service_name
$ rancher run --name stackA/service1 nginx
```

命令参考

要了解更多的命令行支持，请查看我们的Rancher命令文档.

命令和选项

Rancher CLI 可以用于操作Rancher中的环境、主机、应用、服务和容器。

RANCHER CLI 命令

名字	描述
catalog	操作应用商店
config	设置客户端配置
docker	在主机上运行docker命令
environment, env	操作环境
events, event	展示资源变更事件
exec	在容器上运行命令
export	将应用的yml配置文件导出为tar或者本地文件
hosts, host	操作主机
logs	抓取容器的日志
ps	展示服务 / 容器
restart	重启服务 / 容器
rm	删除服务、容器、应用、主机、卷
run	运行服务
scale	设置一个服务运行的容器数量
ssh	SSH到主机
stacks, stack	操作应用
start, activate	启动服务、容器、主机、应用
stop, deactivate	停止服务、容器、主机、应用
up	启动所有服务
volumes, volume	操作卷
inspect	查看服务、容器、主机、环境、应用、卷的详情
wait	等待服务、容器、主机、应用栈、机器、项目模版
help	展示命令列表或者某个命令的说明

RANCHER CLI 全局参数

当使用rancher时，可以使用不同的全局参数。

名字	描述
--debug	调试日志
--config value, -c value	客户端配置文件 (缺省为 ${HOME}/.rancher/cli.json) [$RANCHER_CLIENT_CONFIG]
--environment value, --env value	环境名字或ID [$RANCHER_ENVIRONMENT]
--url value	指定Rancher API接口链接 [$RANCHER_URL]
--access-key value	指定Rancher API访问密钥 [$RANCHER_ACCESS_KEY]
--secret-key value	指定Rancher API安全密钥 [$RANCHER_SECRET_KEY]
--host value	执行docker命令的主机[$RANCHER_DOCKER_HOST]
--wait, -w	等待资源到达最终状态
--wait-timeout value	等待的超时时间(缺省值: 600秒)
--wait-state value	等待的状态(正常, 健康等)
--help, -h	展示帮助说明
--version, -v	打印版本信息

等待资源

全局的选项如--wait 或 -w 可以用于需逐渐到达最终状态的命令。当编辑Rancher命令的脚本时，使用-w选项可以让脚本等待资源就绪后再执行下一个命令。等待的超时时间默认时十分钟，但如果你想要改变超时时间，可以使用--wait-timeout选项。你还可以使用--wait-state选项来指定资源必须到达某个特定状态后，命令才结束返回。

RANCHER CATALOG 说明

rancher catalog 命令提供了操作应用商店模版的相关操作。

选项

名字	描述
--quiet, -q	只展示IDs
--format value	json格式或自定义格式: `'\{\{.ID\}\} \{\{.Template.Id\}\}'`
--system, -s	展示系统模版

子命令

名字	描述
ls	列出应用商店模版
install	安装应用商店模版
help	展示命令列表或某个命令的说明

RANCHER CATALOG LS

rancher catalog ls 命令列出环境下的所有模版。

选项

名字	描述
--quiet, -q	只展示IDs
--format value	json格式或自定义格式: `'\{\{.ID\}\} \{\{.Template.Id\}\}'`
--system, -s	展示系统模版

```
# 列出所有应用商店模版
$ rancher catalog ls
# 列出运行kubernetes环境中的所有应用商店模版
$ rancher --env k8sEnv catalog ls
# 列出系统应用商店模版
$ rancher catalog ls --system
```

RANCHER CATALOG INSTALL

rancher catalog install命令在你的环境中安装应用商店模版。

选项

名字	描述
-answers value, -a value	模版的参数文件。格式应为yaml或者json，并且确保文件有正确的后缀名。
--name value	创建的应用的名字
--system, -s	安装一个系统模版

```
# 安装一个应用模版
$ rancher catalog install library/route53:v0.6.0-rancher1 --name route53
# 安装一个应用模版并将其标识为系统模版
$ rancher catalog install library/route53:v0.6.0-rancher1 --name route53 --system
```

RANCHER CONFIG 说明

rancher config 命令用于设置你的Rancher Server的配置.

```
$ rancher config
URL []: http://<server_ip>:8080
Access Key []: <accessKey_of_account_api_key>
Secret Key []:  <secretKey_of_account_api_key>
# 如果超过一个环境，你需要指定一个环境
Environments:
[1] Default(1a5)
[2] k8s(1a10)
Select: 1
INFO[0017] Saving config to /Users/<username>/.rancher/cli.json
```

选项

名字	描述
--print	打印当前的配置

如果你想要打印当前配置，可以使用----print。

```
# 显示当前的Rancher的配置
$ rancher config --print
```

RANCHER DOCKER 说明

rancher docker 命令允许你在某台机器上运行任何Docker命令。 使用 $RANCHER_DOCKER_HOST 来运行Docker命令. 使用 --host 或者 --host 来选择其他主机。

```
$ rancher --host 1h1 docker ps
```

选项

名字	描述
--help-docker	显示 docker --help

> 注意： 如果环境变量RANCHER_DOCKER_HOST没有设置，你需要通过--host指定运行Docker命令的主机。

RANCHER ENVIRONMENT 说明

rancher environment命令让你可以操作环境。如果你使用账户API key, 你可以创建和更新环境。如果你使用环境API key ，你不能创建和更新其他环境，只能看到你当前的环境。

选项

名字	描述
--all, -a	显示暂停 / 无效和最近移除的资源
--quiet, -q	只显示IDs
--format value	json 或者自定义格式: `'\{\{.ID\}\} \{\{.Environment.Name\}\}'`

子命令

名字	描述
ls	列出所有环境
create	创建一个环境
templates, template	操作环境模版
rm	删除环境
deactivate	停用环境
activate	启用环境
help	显示命令列表或者某个命令的帮助

RANCHER ENV LS

rancher env ls命令显示Rancher中的所有环境。

选项

名字	描述
--all, -a	显示暂停 / 无效和最近移除的资源
--quiet, -q	只显示IDs
--format value	json 或者自定义格式: `'\{\{.ID\}\} \{\{.Environment.Name\}\}'`

```
$ rancher env ls
ID        NAME      ORCHESTRATION   STATE     CREATED
1a5       Default   Cattle          active    2016-08-15T19:20:46Z
1a6       k8sEnv    Kubernetes      active    2016-08-17T03:25:04Z
# 只列出环境ID
$ rancher env ls -q
1a5
1a6
```

RANCHER ENV CREATE

rancher env create命令用于创建一个新的环境，环境的缺省的编排引擎使用cattle。

选项

名字	描述
--template value, -t	value 创建环境的模版（缺省: “Cattle”）

```
# 创建一个环境
$ rancher env create newCattleEnv
# 创建一个kubernetes 环境
$ rancher env create -t kubernetes newk8sEnv
```

RANCHER ENV TEMPLATE

rancher env template 命令用于导出或者导入环境模版。

选项

名字	描述
--all, -a	显示暂停 / 无效和最近移除的资源
--quiet, -q	只显示IDs
--format value	json 或者自定义格式: `'\{\{.ID\}\} \{\{.ProjectTemplate.Name\}\}'`

子命令

名字	描述
export	将一个环境模版导出到标准输出
import	从一个文件中导入环境模版
help	显示命令列表或者某个命令的帮助

RANCHER ENV RM

rancher env rm命令用于删除环境。可以使用环境名字或者ID来删除。 bash # 使用名字删除环境 $ rancher env rm newk8sEnv # 使用ID删除环境 $ rancher env rm 1a20

RANCHER ENV DEACTIVATE

rancher env deactivate命令停用一个环境。用环境名字或者ID来指定停用的环境。

Rancher Env Activate

rancher env activate 命令启用一个环境。用环境名字或者ID来指定启用的环境。

RANCHER EVENTS 说明

rancher events 命令列出Rancher Server中所有出现的事件。

选项

名字	描述
--format value	json 或者自定义格式: `'\{\{.Name\}\} \{\{.Data.resource.kind\}\}'`
--reconnect,	-r 出错时重连接

RANCHER EXEC 说明

rancher exec 命令可以用于执行进入在Rancher的容器。 用户不需要知道容器在哪个宿主机，只需要知道Rancher中的容器ID（如 1i1, 1i788）。

```
# 执行进入一个容器
$ rancher exec -i -t 1i10
```

选项

名字	描述
--help-docker	显示docker exec --help

在rancher exec 命令找到容器后，它在指定的主机和容器执行 docker exec命令。 可以通过使用--help-docker来显示 docker exec的说明。

```
# 显示docker exec --help
$ rancher exec --help-docker
```

RANCHER EXPORT 说明

rancher export 命令将一个应用的 docker-compose.yml 和 rancher-compose.yml文件导出为tar包。

选项

名字	描述
--file value, -f value	输出到一个指定文件中。使用 - 可以输出到标准输出流
--system, -s	是否导出整个环境，包括系统应用。

```
# 将一个应用中所有服务的docker-compose.yml和 rancher-compose.yml导出为tar包。
$ rancher export mystack > files.tar
$ rancher export -f files.tar mystack
```

RANCHER HOSTS 说明

rancher hosts命令可用于操作环境中的主机。

选项

名字	描述
--all, -a	显示暂停 / 无效和最近移除的资源
--quiet, -q	只显示IDs
--format value	json 或者自定义格式: `'\{\{.ID\}\} \{\{.Host.Hostname\}\}'`

子命令

名字	描述
ls	显示主机列表
create	创建一个主机

RANCHER HOSTS LS

rancher hosts ls 命令列出所有主机。

选项

名字	描述
--all, -a	显示暂停 / 无效和最近移除的资源
--quiet, -q	只显示IDs
--format value	json 或者自定义格式: `'\{\{.ID\}\} \{\{.Host.Hostname\}\}'`

```
$ rancher hosts ls
ID        HOSTNAME      STATE     IP
1h1       host-1        active    111.222.333.444
1h2       host-3        active    111.222.333.555
1h3       host-2        active    111.222.333.666
1h4       host-4        active    111.222.333.777
1h5       host-5        active    111.222.333.888
1h6       host-6        active    111.222.333.999
# 只显示主机ID
$ rancher hosts ls -q
1h1
1h2
1h3
1h4
1h5
1h6
```

RANCHER HOSTS CREATE

rancher hosts create命令用于创建 主机. 当创建主机时，需要调用Docker Machine命令，因此需要提供该命令所需要的选项。

RANCHER LOGS 说明

rancher logs 用于抓取指定容器名或容器ID的容器的日志。

选项

名字	描述
--service, -s	显示服务日志
--sub-log	显示服务副日志
--follow, -f	设置日志继续输出
--tail value	显示日志的最后的几行 (缺省: 100)
--since value	显示自某个时间戳后的日志
--timestamps, -t	显示时间戳

```
# 获取某个容器ID对应容器的最后50行日志
$ rancher logs --tail 50 <ID>
# 使用容器名来查看日志
$ rancher logs -f <stackName>/<serviceName>
```

RANCHER PS 说明

rancher ps 命令显示Rancher中的所有服务或者容器。如果不附加任何选项，该命令会返回环境中所有的服务。

选项

名字	描述
--all, -a	显示暂停 / 无效和最近移除的资源
--system, -s	显示系统资源
--containers, -c	显示容器
--quiet, -q	只显示IDs
--format value	json 或者自定义格式: `'\{\{.Service.Id\}\} \{\{.Service.Name\}\} \{\{.Service.LaunchConfig.ImageUuid\}\}'`

```
# 列出所有服务
$ rancher ps
ID        TYPE      NAME           IMAGE     STATE        SCALE     ENDPOINTS   DETAIL
1s1       service   Default/blog   ghost     activating   3                     Waiting for [instance:Default_blog_3]. Instanc
e status: Storage : Downloading
# 列出所有容器
$ rancher ps -c
ID        NAME             IMAGE                          STATE     HOST      DETAIL
1i1       Default_blog_1   ghost                          running   1h1
1i2       Default_blog_2   ghost                          running   1h2
1i3       Default_blog_3   ghost                          running   1h3
```

detail 一列提供了服务的当前状态。

RANCHER RESTART 说明

rancher restart可以用于重启任何主机、服务和容器。

选项

名字	描述
--type value	指定重启的类型 (服务, 容器)
--batch-size value	一次中重启的容器数量 (缺省值: 1)
--interval value	两次重启的间隔时间，单位ms (缺省值: 1000)

```
# 通过服务、容器、主机的ID重启
$ rancher restart <ID>
# 通过服务、容器、主机的名字重启
$ rancher restart <stackName>/<serviceName>
```

> 注意： 服务里中需要包含了应用的名字，以保证指定了正确的服务。

RANCHER RM 说明

rancher rm 命令用于删除资源，比如主机、应用栈、服务、容器或者卷。

选项

名字	描述
--type value	指定删除的特定类型
--stop, -s	在删除前首先暂停资源

```
$ rancher rm <ID>
```

RANCHER RUN 说明

run 命令以1个容器的规模来部署一个服务。当创建服务时，如果想将其置于某个应用栈中， 需要提供--name和stackName/serviceName。如果--name 没有提供，那么新建的服务的名字是Docker提供的容器名，且处于 Default 应用中。

```
$ rancher run --name App2/app nginx
# CLI返回新建服务的ID
1s3
$ rancher -i -t --name serviceA ubuntu:14.04.3
1s4
```

如果要在主机上公开一个端口，那么该主机的端口必须可用。Rancher会自动调度容器到端口可用的主机上。

```
$ rancher -p 2368:2368 --name blog ghost
1s5
```

RANCHER SCALE 说明

当你使用rancher run创建一个服务时，服务的规模缺省是1。可以使用rancher scale命令来扩容某个服务。可以通过名字或者ID来指定服务。

```
$ rancher scale <stackName>/<serviceName>=5 <serviceID>=3
```

RANCHER SSH 说明

rancher ssh 用于ssh到UI创建的某个主机中。它无法ssh通过自定义 命令添加的主机。

```
$ rancher ssh <hostID>
```

RANCHER STACKS 说明

rancher stacks命令可以操作环境中的应用。

选项

名字	描述
--system, -s	显示系统资源
--quiet, -q	只显示ID
--format value	json 或者自定义格式: `'\{\{.ID\}\} \{\{.Stack.Name\}\}'`

命令

名字	描述
ls	列出应用
create	创建一个应用

RANCHER STACKS LS

rancher stacks ls 命令列出指定环境中的应用。

选项

名字	描述
--system, -s	显示系统资源
--quiet, -q	只显示IDs
--format value	json 或者自定义格式: `'\{\{.ID\}\} \{\{.Stack.Name\}\}'`

```
#列出所有应用栈
$ rancher stacks ls
ID        NAME        STATE      CATALOG                           SYSTEM    DETAIL
1e1       zookeeper   healthy    catalog://community:zookeeper:1   false
1e2       Default     degraded                                     false
1e3       App1        healthy                                      false
# 只列出应用栈IDs
$ rancher stacks ls -q
1e1
1e2
1e3
```

RANCHER STACKS CREATE

rancher stacks create 命令用于创建新的应用。应用可以为空的或者从docker-compose.yml 和 rancher-compose.yml文件中创建。

选项

名字	描述
--start	在创建后启动应用
--system, -s	创建一个系统应用
--empty, -e	创建一个空的应用
--quiet, -q	只展示IDs
--docker-compose value, -f value	Docker Compose 文件 (缺省: “docker-compose.yml”)
--rancher-compose value, -r value	Rancher Compose 文件 (缺省: “rancher-compose.yml”)

```
# 创建一个空的应用
$ rancher stacks create NewStack -e
# 从一个docker-compose和rancher-compose文件创建应用
# 以及在创建后运行应用
$ rancher stacks create NewStack -f dc.yml -r rc.yml --start
```

RANCHER START/ACTIVATE 说明

rancher start 或 rancher activate 命令启动指定的资源类型，如主机、服务或容器。

选项

名字	描述
--type value	启动指定的类型 (服务, 容器, 主机, 应用)

```
# 用资源ID来启动
$ rancher start <ID>
# 用资源名字来启动
$ rancher start <stackName>/<serviceName>
```

> 注意： 为了保证指定了正确的服务，服务名中需要包含应用的名字。

RANCHER STOP/DEACTIVATE 说明 rancher stop 或 rancher deactivate 命令用于停止指定的资源类型，如主机、服务和容器。

选项

名字	描述
--type value	停止指定的资源类型 (服务, 容器, 主机, 应用)

```
# 用ID来停止
$ rancher stop <ID>
# 用名字来停止
$ rancher stop <stackName>/<serviceName>
```

注意： 为了保证指定了正确的服务，服务名中需要包含应用的名字。

RANCHER UP 说明

rancher up命令类似于 Docker Compose的up 命令。

选项

名字	描述
--pull, -p	在升级前在所有主机上先拉取镜像
-d	不阻塞和记录日志
--upgrade, -u, --recreate	如果服务发生变更则升级
--force-upgrade, --force-recreate	不管服务有无变更，都进行升级
--confirm-upgrade, -c	确认升级成功和删除旧版本的容器
--rollback, -r	回滚到之前部署的版本
--batch-size value	一次升级的容器数量 (缺省: 2)
--interval value	更新的间隔，单位毫秒 (缺省: 1000)
--rancher-file value	指定一个新的Rancher compose文件 (缺省: rancher-compose.yml)
--env-file value, -e value	指定一个包含了环境变变量的文件。格式应为yaml或者json，并且确保文件有正确的后缀名。
--file value, -f value	指定一个或多个新的compose文件 (缺省: docker-compose.yml) [$COMPOSE_FILE]
--stack value, -s value	指定一个新的项目名字(缺省: 目录名)

```
# 在末尾还上 -d，防止阻塞和记录日志
$ rancher up -s <stackName> -d
```

RANCHER VOLUMES 说明

rancher volumes 命令用于操作卷。

选项

名字	描述
--all, -a	显示暂停 / 无效和最近移除的资源
--quiet, -q	只显示IDs
--format value	json 或者自定义格式: `'\{\{.ID\}\} \{\{.Volume.Name\}\}'`

命令

名字	描述
ls	列出卷
rm	删除一个卷
create	创建一个卷

RANCHER VOLUME LS

rancher volume ls命令列出环境中的所有卷。

选项

名字	描述
--all, -a	显示暂停 / 无效和最近移除的资源
--quiet, -q	只显示IDs
--format value json	或者自定义格式: `'\{\{.ID\}\} \{\{.Volume.Name\}\}'`

```
$ rancher volumes ls
ID        NAME                      STATE      DRIVER       DETAIL
1v1                                 active
1v2                                 active
1v3                                 detached
1v4                                 active
1v5                                 detached
1v6                                 detached
1v7       rancher-agent-state       active     local
```

RANCHER VOLUME RM

rancher volume rm 命令用于删除卷。

```
$ rancher volumes rm <VOLUME_ID>
```

RANCHER VOLUME CREATE

rancher volume create 用于创建卷。

选项

名字	描述
--driver value	指定卷驱动
--opt value	设置驱动特定的 key/value 选项

```
# 使用 Rancher NFS 驱动创建新的卷
$ rancher volume create NewVolume --driver rancher-nfs
```

RANCHER INSPECT 说明

rancher inspect 用于查看资源的详情。

选项

名字	描述
--type value	查看指定的类型 (服务, 容器, 主机)
--links	在资源详情中包含操作和链接的URL
--format value json	或者自定义格式: `'\{\{.kind\}\}' (默认: “json”)`

```
# 用ID来查看详情
$ rancher inspect <ID>
# 用名字来查看详情
$ rancher inspect <stackName>/<serviceName>
```

> 注意： 为了保证指定了正确的服务，服务名中需要包含应用的名字。

RANCHER WAIT 说明

rancher wait 命令用于等待资源完成操作。 它对自动化Rancher命令十分有用，可以在脚本中用于等待某个资源就绪后再执行更多操作。

```
$ rancher start 1i1
$ rancher wait 1i1
```

变量替换

使用rancher up时，可以在docker-compose.yml和rancher-compose.yml文件中使用运行rancher命令的机器中的环境变量。 这仅仅在rancher命令中支持，在Rancher UI中不支持。

如何使用

通过使用docker-compose.yml和rancher-compose.yml文件，你可以引用机器上的环境变量。 如果机器上没有环境变量，它将用空白字符串替换。 Rancher将会提示一个警告，指出哪些环境变量没有设置。 如果使用环境变量作为镜像标签时，请注意，rancher不会从镜像中自动删除：来获取latest镜像。 因为镜像名，比如<镜像名>：是一个无效的镜像名，所以不会部署成功。 用户需要确定机器中所有的环境变量的有效性。

例子

在我们运行rancher的机器上，我们有一个环境变量IMAGE_TAG = 14.04。

```
# Image tag is set as environment variable
$ env | grep IMAGE
IMAGE_TAG=14.04
# Run rancher
$ rancher up
```

例子： docker-compose.yml

```
version: '2'
services:
  ubuntu:
    tty: true
    image: ubuntu:$IMAGE_TAG
    stdin_open: true
```

在Rancher中，一个ubuntu服务将使用ubuntu:14.04镜像来部署。

变量替换格式

Rancher支持与’docker-compose’相同的格式。

```
version: '2'
services:
  web:
    # unbracketed name
    image: "$IMAGE"

    # bracketed name
    command: "${COMMAND}"

    # array element
    ports:
    - "${HOST_PORT}:8000"

    # dictionary item value
    labels:
      mylabel: "${LABEL_VALUE}"

    # unset value - this will expand to "host-"
    hostname: "host-${UNSET_VALUE}"

    # escaped interpolation - this will expand to "${ESCAPED}"
    command: "$${ESCAPED}"
```

模板

在docker-compose.yml里面，Rancher能够支持使用Go模板系统，这样我们可以在docker-compose.yml里面使用逻辑条件语句。

模板可以与Rancher CLI一起使用，也可以与应用商店组合使用，这样可以让你配置你的应用商店模板，也可以让你根据答案来改变你的模板文件。

> 注意： 目前我们只支持对string的比较。

例子

如果你希望能够生成一个在内部暴露端口或者在外部暴露端口的服务，那么你可以设置逻辑条件来实现这样的功能。 在这个例子中，如果public变量设置为ture，那么ports下面的8000端口将对外开放。 否则，这些端口将在expose下开放。在我们的示例中，默认值为true。

docker-compose.yml

```
version: '2'
services:
  web:
    image: nginx
    {{- if eq .Values.PUBLIC "true" }}
    ports:
    - 8000
    {{- else }}
    expose:
    - 8000
    {{- end }}
```

rancher-compose.yml

```
version: '2'
catalog:
  name: Nginx Application
  version: v0.0.1
  questions:
  - variable: PUBLIC
    label: Publish Ports?
    required: true
    default: true
    type: boolean
```

config.yml

```
name: "Nginx Application"
version: v0.0.1
```

应用栈名称替换

从Rancher v1.6.6开始，我们支持在docker-compose.yml文件中替换 {{ .Stack.Name }} 。这样可以在compose文件中使用应用栈名称。

Docker Compose文件可以用于创建新的应用栈，可以通过Rancher命令行或UI来创建。 如下面中的例子，你可以创建一个基于应用栈名称的标签。

示例 DOCKER-COMPOSE.YML

```
version: '2'
services:
  web:
    image:  nginx
    labels:
      stack-name: {{ .Stack.Name }}
```

如果你通过Rancher命令行来创建应用，例如rancher up -s myawesomestack -f docker-compose.yml，那么这个应用将会创建一个带有标签stack-name=myawesomestack的服务。

> 注意： 替换只是发生在应用栈创建时，之后对应用名称的修改无法触发替换。

双括号使用

随着Rancher引入了模板系统，双括号 ({{ or }}) 将被视为模板的一部分。如果你不想将这些字符转换为模板，你可以在包含字符的compose文件的顶部添加上＃notemplating。

```
# notemplating

version: '2'
services:
  web:
    image: nginx
    labels:
      key: "{{`{{ value }}`}}"
```

API文档

支持 Rancher

开发

我们Githb的代码库的Wiki里有所有能帮助你参与Rancher开发的的步骤。

从我们的第一个cowpoke项目开始！

代码库

我们的所有代码都在我们的Github主页。 这些代码库有很多为Rancher所用，下面是其中Rancher用到的主要的代码库的描述。

Rancher： Rancher的主要代码库，它整合了很多其它的代码库.

Cattle：Rancher所开发的主要功能代码库。

UI: Rancher UI代码库。

Rancher Catalog: 这个代码库包含了大多数应用商店 基础设施服务模板。这些模板(在infra-templates目录中)，在环境(environments)中被用做环境模版的一部分。我们欢迎社区参与Rancher开发。

Community Catalog： 这个代码库包含了所有社区贡献的 Rancher 应用商店模板。我们欢迎社区参与Rancher应用商店模板的开发。

Rancher CLI： Rancher 命令行基于此代码库。

Rancher Compose: Rancher Compose命令行基于此代码库。 此代码库和 docker/libcompose 保持同步。

缺陷

如果你发现缺陷，或遇到任何问题，请在Github上提Issue。 虽然我们有很多Rancher相关的代码库，但我们希望你把所有的issue都提交到 Rancher代码库里，以免我们遗漏。

如果你想要更新Rancher的文档，请提交PR到我们的文档代码库或点击编辑此页直接跳转到编辑页面。

Rancher Server的常见问题

我正在运行的RANCHER是什么版本的?

Rancher的版本位于UI的页脚的左侧。 如果你点击该版本，你将可以查看其他组件的详细版本。

我怎么样在代理服务器后运行RANCHER SERVER?

请参照在HTTP代理后方启动Rancher Server.

我在哪能找到 RANCHER SERVER 容器的详细日志？

运行docker logs可以查看在Rancher Server容器的基本日志。要获取更详细的日志，你可以进入到Rancher Server容器内部并查看日志文件。

```
# 进入 Rancher  Server  容器内部
$ docker exec -it <container_id> bash

# 跳转到 Cattle 日志所在的目录下
$ cd /var/lib/cattle/logs/
$ cat cattle-debug.log
```

在这个目录里面会出现cattle-debug.log和cattle-error.log。 如果你长时间使用此Rancher Server，你会发现我们每天都会创建一个新的日志文件。

将RANCHER SERVER的日志复制到主机上。

以下是将Rancher Server日志从容器复制到主机的命令。

```
$ docker cp <container_id>:/var/lib/cattle/logs /local/path
```

如何导出RANCHER SERVER容器的内部数据库？ 你可以通过简单的Docker命令从Rancher Server容器导出数据库。

```
$ docker exec <CONTAINER_ID_OF_SERVER> mysqldump cattle > dump.sql
```

如果RANCHER SERVER的IP改变了会怎么样？

如果更改了Rancher Server的IP地址，你需要用新的IP重新关联主机。

在Rancher中，点击系统管理->系统设置更新 Rancher Server的主机注册地址。注意必须包括Rancher Server暴露的端口号。默认情况下我们建议按照安装手册中使用8080端口。

主机注册更新后，进入基础架构->添加主机->自定义。 添加主机的docker run命令将会更新。 使用更新的命令，在Rancher Server的所有环境中的所有主机上运行该命令。

为什么在日志中看到GO-MACHINE-SERVICE在不断重新启动？ 我该怎么办？

Go-machine-service是一种通过websocket连接到Rancher API服务器的微服务。如果无法连接，则会重新启动并再次尝试。

如果你运行的是单节点的Rancher Server，它将使用你为主机注册地址来连接到Rancher API服务。 检查从Rancher Sever容器内部是否可以访问主机注册地址。

```
$ docker exec -it <rancher-server_container_id> bash
# 在 Rancher-Server 容器内
$ curl -i <Host Registration URL you set in UI>/v1
```

你应该得到一个json响应。 如果认证开启，响应代码应为401。如果认证未打开，则响应代码应为200。

验证Rancher API Server 能够使用这些变量，通过登录go-machine-service容器并使用你提供给容器的参数进行curl命令来验证连接:

```
$ docker exec -it <go-machine-service_container_id> bash
# 在go-machine-service 容器内
$ curl -i -u '<value of CATTLE_ACCESS_KEY>:<value of CATTLE_SECRET_KEY>' <value of CATTLE_URL>
```

你应该得到一个json响应和200个响应代码。

如果curl命令失败，那么在go-machine-service和Rancher API server之间存在连接问题。

如果curl命令没有失败，则问题可能是因为go-machine-service尝试建立websocket连接而不是普通的http连接。 如果在go-machine-service和Rancher API服务器之间有代理或负载平衡，请验证代理是否支持websocket连接。

RANCHER SERVER在运行中变的极慢，怎么去恢复它？

很可能有一些任务由于某些原因而处于僵死状态，如果你能够用界面查看系统管理 -> 系统进程，你将可以看到Running中的内容，如果这些任务长时间运行（并且失败），则Rancher会最终使用太多的内存来跟踪任务。这使得Rancher Server处于了内存不足的状态。

为了使服务器变为可响应状态，你需要添加更多内存。通常4GB的内存就够了。

你需要再次运行Rancher Server命令并且添加一个额外的选项-e JAVA_OPTS="-Xmx4096m"

```
$ docker run -d -p 8080:8080 --restart=unless-stopped -e JAVA_OPTS="-Xmx4096m" rancher/server
```

根据MySQL数据库的设置方式的不同，你可能需要进行升级才能添加该选项。

如果是由于缺少内存而无法看到系统管理 -> 系统进程的话，那么在重启Rancher Server之后，已经有了更多的内存。你现在应该可以看到这个页面了，并可以开始对运行时间最长的进程进行故障分析。

RANCHER SERVER数据库数据增长太快.

Rancher Server会自动清理几个数据库表，以防止数据库增长太快。如果对你来说这些表没有被及时清理，请使用API来更新清理数据的时间间隔。

在默认情况下，产生在2周以前的container_event和service_event表中的数据则数据会被删除。在API中的设置是以秒为单位的(1209600)。API中的设置为events.purge.after.seconds.

默认情况下，process_instance表在1天前产生的数据将会被删除，在API中的设置是以秒为单位的(86400)。API中的设置为process_instance.purge.after.seconds.

为了更新API中的设置，你可以跳转到 `http://<rancher-server-ip>:8080/v1/settings` 页面， 搜索要更新的设置，点击links -> self跳转到你点击的链接去设置，点击侧面的“编辑”更改’值’。 请记住，值是以秒为单位。

为什么RANCHER SERVER升级失败或被冻结？

如果你刚开始运行Rancher并发现它被永久冻结，可能是liquibase数据库上锁了。在启动时，liquibase执行模式迁移。它的竞争条件可能会留下一个锁定条目，这将阻止后续的流程。

如果你刚刚升级，在Rancher Server日志中，MySQL数据库可能存在尚未释放的日志锁定。

```
....liquibase.exception.LockException: Could not acquire change log lock. Currently locked by <container_ID>
```

释放数据库锁

> 注意： 请不要释放数据库锁，除非有相关日志锁的异常。如果是由于数据迁移导致升级时间过长，在这种情况下释放数据库锁，可能会使你遇到其他迁移问题。

如果你已根据升级文档创建了Rancher Server的数据容器，你需要exec到rancher-data容器中升级DATABASECHANGELOGLOCK表并移除锁，如果你没有创建数据容器，你用exec到包含有你数据库的容器中。

```
$ sudo docker exec -it <container_id> mysql
```

一旦进入到 Mysql 数据库, 你就要访问cattle数据库。

```
mysql> use cattle;

#检查表中是否有锁
mysql> select * from DATABASECHANGELOGLOCK;

# 更新移除容器的锁
mysql> update DATABASECHANGELOGLOCK set LOCKED="", LOCKGRANTED=null, LOCKEDBY=null where ID=1;


# 检查锁已被删除
mysql> select * from DATABASECHANGELOGLOCK;
+----+--------+-------------+----------+
| ID | LOCKED | LOCKGRANTED | LOCKEDBY |
+----+--------+-------------+----------+
|  1 |        | NULL        | NULL     |
+----+--------+-------------+----------+
1 row in set (0.00 sec)
```

Rancher Agent的常见问题

我如何在代理服务器后配置主机？ 要在代理服务器后配置主机，你需要配置Docker的守护进程。详细说明参考在代理服务器后添加自定义主机。

RANCHER AGENT无法启动的原因是什么？

添加 --NAME RANCHER-AGENT

如果你从UI中编辑docker run .... rancher/agent...命令并添加--name rancher-agent选项，那么Rancher Agent将启动失败。Rancher Agent在初始运行时会启动3个不同容器。最后启动的名为rancher-agent的容器是Rancher Agent成功连接到Rancher Server所必需的容器。在rancher-agent容器之前，启动的其他两个容器是用来探测主机并且设置状态的。它们会被自动移除。

使用一个克隆的虚拟机。

如果你使用了克隆其他Agent主机的虚拟机并尝试注册它，它将不能工作。在rancher-agent容器的日志中会产生ERROR: Please re-register this agent.字样的日志。Rancher主机的唯一ID保存在/var/lib/rancher/state，因为新添加和虚拟机和被克隆的主机有相同的唯一ID，所以导致无法注册成功。

解决方法是在克隆的VM上运行以下命令： rm -rf /var/lib/rancher/state; docker rm -fv rancher-agent; docker rm -fv rancher-agent-state, 完成后可重新注册。

我在哪里可以找到RANCHER AGENT容器的详细日志?

从v1.6.0起，在rancher-agent容器上运行docker logs将提供agent相关的所有日志。

如何验证你的主机注册地址设置是否正确？ 如果你正面临Rancher Agent和Rancher Server的连接问题，请检查主机设置。当你第一次尝试在UI中添加主机时，你需要设置主机注册的URL，该URL用于建立从主机到Rancher Server的连接。这个URL必须可以从你的主机访问到。为了验证它，你需要登录到主机并执行curl命令：

```
curl -i <Host Registration URL you set in UI>/v1
```

你应该得到一个json响应。 如果开启了认证，响应代码应为401。如果认证未打开，则响应代码应为200。

> 注意： 普通的HTTP请求和websocket连接（ ws:// ）都将被使用。 如果此URL指向代理或负载平衡器，请确保它们可以支持Websocket连接。

主机是如何自动探测IP的？我该怎么去修改主机IP？如果主机IP改变了（因为重启），我该怎么办？

当Agent连接Rancher Server时，它会自动检测Agent的IP。有时，自动探测的IP不是你想要使用的IP，或者选择了docker网桥的IP，如. 172.17.x.x。

或者，你有一个已经注册的主机，当主机重启后获得了一个新的IP, 这个IP将会和Rancher UI中的主机IP不匹配。

你可以重新配置“CATTLE_AGENT_IP”设置，并将主机IP设置为你想要的。

当主机IP地址不正确时，容器将无法访问管理网络。要使主机和所有容器进入管理网络，只需编辑添加自定义主机的命令行，将新的IP指定为环境变量“CATTLE_AGENT_IP”。 在主机上运行编辑后的命令。 不要停止或删除主机上的现有的Rancher Agent容器！

```
$ sudo docker run -d -e CATTLE_AGENT_IP=<NEW_HOST_IP> --privileged \
    -v /var/run/docker.sock:/var/run/docker.sock \
    rancher/agent:v0.8.2 http://SERVER_IP:8080/v1/scripts/xxxx
```

如果我没有使用RANCHER删除主机而是直接删除会发生什么?

如果你的主机直接被删除，Rancher Server会一直显示该主机。 主机会处于Reconnecting状态，然后转到Disconnected状态，这样你才能够从UI中删除这些主机。

如果你有添加了健康检查功能的服务容器部署在Disconnected主机上，它们将会被重新调度到其他主机上。

为什么同一主机在UI中多次出现?

主机不能持久化var/lib/rancher这个文件夹，这是Rancher用来存储用于标识主机的必要信息。请在添加主机前删除这个文件夹。

常见的故障排查与修复方法

请先阅读有关Rancher Server和Rancher Agent的常见问题。

本节假设你能够成功启动Rancher Server并添加主机。

服务/容器

为什么我只能编辑容器的名称？

Docker容器在创建之后就不可更改了。唯一可更改的内容是我们要存储的不属于Docker容器本身的那一部分数据。 无论是停止、启动或是重新启动，它始终在使用相同的容器。如需改变任何内容都需要删除或重新创建一个容器。 你可以克隆，即选择已存在的容器，并基于已有容器的配置提前在添加服务界面中填入所有要设置的内容，如果你忘记填入某项内容，可以通过克隆来改变它之后删除旧的容器。

关联的容器/服务在RANCHER中是如何工作的？

在Docker中，关联的容器（在 docker run中使用--link）会出现在容器的/etc/hosts中。在Rancher中，我们不需要更改容器的/etc/hosts文件，而是通过运行一个内部DNS服务器来关联容器，DNS服务器会返回给我们正确的IP。

求助! 我不能通过RANCHER的界面打开命令行或查看日志。RANCHER是如何去访问容器的命令行和日志的?

Agent主机有可能会暴露在公网上，Agent上接受到的访问容器命令行或者日志的请求是不可信的。Rancher Server中发出的请求包括一个JWT（JSON Web Token)，JWT是由服务器签名并且可由Agent校验的，Agent可以判断出请求是否来自服务器，JWT中包括了有效期限，有效期为5分钟。这个有效期可以防止它被长时间使用。如果JWT被拦截而且没有用SSL时，这一点尤为重要。

如果你运行docker logs -f (rancher-agent名称或ID）。日志会显示令牌过期的信息，随后检查Rancher Server主机和Rancher Agent主机的时钟是否同步。

在哪里可以看到我的服务日志?

在服务的详细页中，我们提供了一个服务日志的页签日志。在日志页签中，列出了和服务相关的所有事件，包括时间戳和事件相关描述，这些日志将会保留24小时。

跨主机通信

如果容器运行在不同主机上，不能够ping通彼此, 可能是由一些常见的问题引起的.

如何检查跨主机通信是否正常?

在应用->基础设施中，检查 healthcheck 应用的状态。如果是active跨主机通信就是正常的。

手动测试，你可以进入任何一个容器中，去ping另一个容器的内部IP。在主机页面中可能会隐藏掉基础设施的容器，如需查看点击“显示系统容器”的复选框。

UI中显示的主机IP是否正确?

有时，Docker网桥的IP地址会被错误的作为了主机IP，而并没有正确的选择真实的主机IP。这个错误的IP通常是172.17.42.1或以172.17.x.x开头的IP。如果是这种情况，在使用docker run命令添加主机时，请用真实主机的IP地址来配置CATTLE_AGENT_IP环境变量。

```
$ sudo docker run -d -e CATTLE_AGENT_IP=<HOST_IP> --privileged \
    -v /var/run/docker.sock:/var/run/docker.sock \
    rancher/agent:v0.8.2 http://SERVER_IP:8080/v1/scripts/xxxx
```

在UBUNTU上运行容器时彼此间不能正常通信。

如果你的系统开启了UFW，请关闭UFW或更改/etc/default/ufw中的策略为：

```
DEFAULT_FORWARD_POLICY="ACCEPT"
```

RANCHER的默认子网（10.42.0.0/16）在我的网络环境中已经被使用或禁止使用，我应该怎么去更改这个子网？

Rancher Overlay网络默认使用的子网是10.42.0.0/16。如果这个子网已经被使用，你将需要更改Rancher网络中使用的默认子网。你要确保基础设施服务里的Network组件中使用着合适的子网。这个子网定义在该服务的rancher - compose.yml文件中的default_network里。

要更改Rancher的IPsec或VXLAN网络驱动，你将需要在环境模版中修改网络基础设施服务的配置。创建新环境模板或编辑现有环境模板时，可以通过单击编辑来配置网络基础结构服务的配置。在编辑页面中，选择配置选项 > 子网输入不同子网，点击配置。在任何新环境中将使用环境模板更新后的子网，编辑已经有的环境模板不会更改现在已有环境的子网。

这个实例是通过升级网络驱动的rancher-compose.yml文件去改变子网为10.32.0.0/16.

```
ipsec:
  network_driver:
    name: Rancher IPsec
    default_network:
      name: ipsec
      host_ports: true
      subnets:
      # After the configuration option is updated, the default subnet address is updated
      - network_address: 10.32.0.0/16
      dns:
      - 169.254.169.250
      dns_search:
      - rancher.internal
    cni_config:
      '10-rancher.conf':
        name: rancher-cni-network
        type: rancher-bridge
        bridge: docker0
        # After the configuration option is updated, the default subnet address is updated
        bridgeSubnet: 10.32.0.0/16
        logToFile: /var/log/rancher-cni.log
        isDebugLevel: false
        isDefaultGateway: true
        hostNat: true
        hairpinMode: true
        mtu: 1500
        linkMTUOverhead: 98
        ipam:
          type: rancher-cni-ipam
          logToFile: /var/log/rancher-cni.log
          isDebugLevel: false
          routes:
          - dst: 169.254.169.250/32
```

> 注意： 随着Rancher通过升级基础服务来更新子网，以前通过API更新子网的方法将不再适用。

DNS

如何查看我的DNS是否配置正确?

如果你想查看Rancher　DNS配置，点击应用 > 基础服务。点击network-services应用，选择metadata，在metadata中，找到名为network-services-metadata-dns-X的容器，通过UI点击执行命令行后，可以进入该容器的命令行，然后执行如下命令。

```
$ cat /etc/rancher-dns/answers.json
```

CENTOS

为什么我的容器无法连接到网络? 如果你在主机上运行一个容器（如：docker run -it ubuntu）该容器不能与互联网或其他主机通信，那可能是遇到了网络问题。

Centos默认设置/proc/sys/net/ipv4/ip_forward为0，这从底层阻断了Docker所有网络。Docker将此值设置为1，但如果在CentOS上运行service restart network，则其将被重新设置为0。

负载均衡

为什么我的负载均衡一直是INITIALIZING状态?

负载均衡器自动对其启用健康检查。 如果负载均衡器处于初始化状态，则很可能主机之间无法进行跨主机通信。

我如何查看负载均衡的配置?

如果要查看负载均衡器的配置，你需要用进入负载均衡器容器内部查找配置文件，你可以在页面选择负载均衡容器的执行命令行

```
$ cat /etc/haproxy/haproxy.cfg
```

该文件将提供负载均衡器的所有配置详细信息。

我在哪能找到HAPROXY的日志?

HAProxy的日志可以在负载均衡器容器内找到。 负载均衡器容器的docker logs只提供与负载均衡器相关的服务的详细信息，但不提供实际的HAProxy日志记录。

```
$ cat /var/log/haproxy
```

高可用

RANCHER COMPOSE EXECUTOR和GO-MACHINE-SERVICE不断重启.

在高可用集群中，如果你正在使用代理服务器后，如果rancher-compose-executor和go-machine-service不断重启，请确保你的代理使用正确的协议。

认证

求助！我打开了访问控制但不能访问RANCHER了，我该如何重置RANCHER禁用访问控制？

如果你的身份认证出现问题（例如你的GitHub身份认证已损坏），则可能无法访问Rancher。 要重新获得对Rancher的访问权限，你需要在数据库中关闭访问控制。 为此，你需要访问运行Rancher Server的主机。

```
$ docker exec -it <rancher_server_container_ID> mysql
```

> 注意： 这个 是具有Rancher数据库的容器。 如果你升级并创建了一个Rancher数据容器，则需要使用Rancher数据容器的ID而不是Rancher Server容器。

访问Cattle数据库。

```
mysql> use cattle;
```

查看setting表。

```
mysql> select * from setting;
```

更改api.security.enabled为false，并清除api.auth.provider.configured的值。此更改将关闭访问控制，任何人都可以使用UI / API访问Rancher Server。

```
mysql> update setting set value="false" where name="api.security.enabled";
mysql> update setting set value="" where name="api.auth.provider.configured";
```

确认更改在setting表中生效。

```
mysql> select * from setting;
```

可能需要约1分钟才能在用户界面中关闭身份认证，然后你可以通过刷新网页来登陆没有访问控制的Rancher Server。

历史文档

- **Rancher v1.5**

- **Rancher v1.4**

- **Rancher v1.3**

- **Rancher v1.2**

- **Rancher v1.1**

- **Rancher v1.0**