金山云容器解决方案

北京金山云网络技术有限公司

概述	3
适用场景	
创建容器集群	4
创建 VPC 及子网	4
创建安全组	5
创建公网 NAT	6
创建云服务器	7
登录云服务器	
通过 ssh 访问云服务器	9
安装 docker-engine	11
创建容器	
将容器端口暴露到公网	12
安装集群管理系统	14
创建 Swarm	14
创建服务	
将服务暴露到公网	
使用 KS3 存储 docker 镜像	19
搭建企业级的镜像仓库	
安装 Docker	
安装 docker-compose	
下载安装包	
配置 Harbor	20
配置后端存储到 KS3	21
配置 HTTPS	22
安装并启动	
测试	23
管理 Harbor 生命周期	
基于 Docker 的持续集成、持续交付	
搭建基础服务	
代码仓库(构建 gitlab 服务)	
镜像仓库(构建 Docker 私有仓库服务)	26
代码构建(构建 Jenkins 服务)	26
容器集群(构建 Docker Swarm 集群)	
Demo: 基于 CI、CD 的 Flask 项目	
Gitlab 上新建项目	27
本地开发	
配置 Jenkins	
触发 CI、CD	
访问测试	

概述

本文指导用户使用金山云产品,构建用于生产环境的 docker 系统,加速系统开发、测试、 部署,提高软件交付质量。一个完整的 docker 工作流(如下图),通常包含构建(build)、 分发(ship)、运行(run)等几个环节,用户可以根据需要,选择其中的一个或者多个环节。



- Build
 - 使用 jenkins 作为 CI、CD 的引擎
- Ship
 - 提供 docker registry 的 KS3 驱动,结合 harbor 等开源系统,用户可以在金山云构建 低成本,高可靠的企业级镜像存储中心
- Run
 - 用户可以选择云主机或者云物理主机(国内独家)搭建容器集群
 - 用户可以选择 mesos、kubernetes、swarm 等作为容器集群管理系统
 - 金山云强大的网络组件(VPC/LB/NAT),提供安全、可靠、高性能的容器网络
 - 用户可选择 RDS、KCS、KS3 等服务,存储持久化数据,免去自己搭建维护数据 库、缓存的工作

适用场景

- CI/CD(持续集成/持续交付)
- Devops
- 自动化运维
- 混合云管理
- 微服务系统构建

创建容器集群

推荐您在 VPC 机房(目前有北京 6 区和上海 2 区)购买云服务器或者云物理主机,搭建容器集群。金山云 VPC 提供高效、安全、可靠的网络服务,并且便于您构建混合云。关于 VPC 的介绍,请参考 <u>VPC 产品文档</u>。

创建 VPC 及子网

在购买云服务器之前,您应该至少拥有一个 VPC 和一个子网。请参考 VPC <u>快速入门</u>中的"创 建 VPC"和"创建子网"章节。

 创建一个名称为"docker"的 VPC,以及一个名为"docker-vnet-1"子网,您可以自定 义网段,或者使用默认值。这里同时创建了一个类型为"终端子网"的子网 2,在使用 金山云 RDS/KCS 等服务时,需要用到"终端子网"。

VPC信息					
名称:	docker				
网段:	10.0.0.0	/	16		
	掩码范围:8-23				
子网1信息					
名称:	docker-vnet-1				
类型:	普通子网				
网段:	10.0.0.0	/	24		
	掩码范围:16-29				
DHCP地址范围:	10.0.0.2-10.0.0.253				
	DHCP地址范围:最多支	特绑定25	52个云服务器		
网关IP:	10.0.0.1				
DNS1 ·	198 18 96 10				

docker-endpoint
终端子网 用于连接RDS或创建私网负载均衡等服务
10.0.1.0 / 24
掩码范围:22-24
如不肯写野注为"Kee Nat"
如小姐与新队为 Ksc_Nat
内网
1000Mbps

创建安全组

您可以使用 VPC 的默认安全组 DefaultSG,或者参考<u>快速入门</u>中的"创建安全组"创建一个新的安全组。

● 创建一个名称为"docker"的安全组

←返回	网络 > VPC > docker(10.0.0.0/16) -		北京6区(VPC) 上海2区(VPC)
• 子网	◆新建 ⑤ 复制安全组 茴 删除		
 • 頭田 • 互联网网关 		安全组规则数量	绑定云服务器数量
• 隧道网关	DefaultSG	1	0
 · 対等连接 · 安全组 (防火衛) 	创建安全组	×	
ACL	名称: docker		
H	创建	取消	

创建公网 NAT

如果您的容器集群有访问公网的需求(如访问 docker hub 拉取 docker 镜像,或者您部署在 容器集群中的服务需要访问其他公网服务),推荐您使用 VPC 的 NAT 功能。NAT 功能使 您容器集群中的机器,共享外网访问能力。请参考<u>快速入门</u>中的"创建 NAT"章节,创建 一个 BGP 类型的 NAT。

● 新建的 VPC 中,会默认创建一个类型为"金山云内网"的 NAT。需要先把这个 NAT 删掉,才能创建公网类型的 NAT。

← 返回	网络 > VPC > docker(10.0.0	0/16) -				北京6区(VPC) 上海2区(VPC)
• 子网	NAT 云服务	器 端口映射	物理机				
 路由 互联网网关 	◆新建 □ 续费 0	j 调整带宽 👘 删除					
 隧道网关 	■ 名称 映射范	围 类型	IP数量 带宽(M	bps)	计费方式 绑定	定子网数量	到期时间
 对等连接 	Ksc_Nat VPC	金山云内网	1 1000		- 1		a.
• 安全组 (防火墙)							
 ACL 	夕称: Ksc Nat						×
	送店 法最	索计 ID送细信自	继完 子网信自				
Ξ	NATID: 96991+70-14	8-4447-8727-acaf1ad06740	APAL 3 PSTHOL	Allapation -	2016-10-21 20:20:20		
	映射范围: VPC			BSAMPSIPS:	1		
	带宽: 1000 Mbps			绑定子网数量:	1		
删除NAT	定删除以下1条NAT?						×
名称	映射范围	裡 I	数量 带宽((Mbps)	计费方式	#定子网数	ل
Ksc_Nat	VPC 😫	全山云内网 1	1000		(22)	1	
					副	除	取消

● 创建一个链路类型为"BGP"的NAT

← 返回	新建NAT O返回NAT列表	
 子网 	映射范围: ○ VPC ● 子网	配置详情
 路由 	VPC类型的NAT只能创建一个,并且VPC类型NAT和子网类型NAT不能同时创建	映射范围: 子网
• 互联网网关	编陈谢田: BGP	链路类型: BGP
• 隧道网关		名称: docker-public 计典方式: 句任句日
• 对等连接	计费方式: 包午包月	带宽值: 1Mbps
• 安全组 (防火墙)	名称: docker-public	购买时长: 1个月
ACL		购大数里: 1个
	带宽值: 1 1Mbps 200Mbps Mbps	ett. ×25
	购买时长; - 1 + 月	立即购买
	绑定子网: ● 子网名称 子网网段	
	odcker-vnet-1 10.0.0.0/24	

创建云服务器

参考云服务器<u>购买指导</u>。

•

● 根据需要选择云服务器配置,如 CPU、内存、数据盘、购买时长,购买数量,镜像推荐使用 ubuntu-14.04,为了搭建容器集群,购买数量一般最少为3个。

计集方式 每年5日	云服务器 北京6区(VPC)	价格: ¥711.00
	计费方式: 包年包月	
数据中心类型: 🔘 VPC数据中心	配置: 2核 4GB (IO优化型主机)	
数据中心: 北京6区(VPC) 上海2区(VPC)	数据盘: 1块本地SSD系统盘(20GB) 1块本地SSD数据盘(50GB)	
云服务器类型: 10优化型主机 ⑦	镜像: ubuntu-14.04	
	购买量: 1个月 x 3台	
CPU: 1核 2核 4核 8核 16核 24核 32核	免费开通服务器安全 ⑦	
内符: 2GB 4GB 8GB		息计:¥ 711.00
锦像关型: 💿 标准销像 🔿 自定义镜像	上一步下一步	购买
鎬像: -Q ubuntu-14.04 ▼		
系统盘: 本地SSD鱼 v 免费履送(Linux 20GB/Windows 50GB)		
数据曲: 本地SSD曲 → 50 G8 ⑦		
购买时长: 1 1 2 3 4 5 6 7 8 9 11 1年 11 2年 11 3年		
<u> 約买数量:</u> - 3 +		
选择弹性 IP 页面,选择"稍后购买"		
#定時世1P: 〇 购买新的時世1P	云服务器 北京6区(VPC)	价格:¥711.00
稿后可以在"控制台>网络>弹性IP"购买	计费方式: 包年包月	
	配置: 2核 4GB (IO优化型主机)	
	数据盘: 1块本地SSD系统盘(20GB) 1块本地SSD数据盘(50GB)	
	镜像: ubuntu-14.04	
	购买量: 1个月 x 3台	
	◎ 免费开通服务器安全 ?	

Bit: ¥711.00

下一步 购买

上一步

关联VPC ·	docker	_	alitevpc	云服务器	北京6区(VPC)	价格: ¥711.00
	docker	•		计费方式:	包年包月	
关联子网:	docker-vnet-1	*		配置:	2核 4GB (IO优化型主机)	
服务器IP:	自动分配(最大可绑定252台主机)	~		数据盘:	1块本地SSD系统盘(20GB 1块本地SSD数据盘(50GB	
				镜像:	ubuntu-14.04	
关联安全组:	docker	•		购买量:	1个月 x 3台	
				◎ 免费开	風服务器安全 ?	
						Bit: ¥711.00
				E-	步 下 步	购买

● 设置基本信息页面,设置服务器名称,密码等信息后,点击"购买"

服务器名称:	docker	取消后缀	云服务器	北京6区(VPC)	价格: ¥711.00
	可以通过添加后缀,为多台服务器创建	不同的名称	计费方式:	包年包月	
			配置:	2核 4GB (IO优化型主机)	
后缀起始值:	1 示意: docker-1,	docker-2,	数据盘:	1块本地SSD系统盘(20GB) 1块本地SSD数据盘(50GB)	
标签(可空):	标签分类 ▼ 标签名称 ▼	创建新标签	镜像:	ubuntu-14.04	
: <u>站出</u> : : 穴淑员墅曾	● 开启 ○ 关闭 ⑦ ubuntu		购买量:	1个月 x 3台 画服务器安全 ⑦	
管理员密码:		3			息计: ¥711.00
确认密码:			E-	步 购买	

● 购买并成功支付后,可在云服务器控制台看到您购买的主机

计算资	源 > 云服务器	> 实例					北京6区(VPC)	上海2区(VPC)
+	新建实例	資 续费 ①) 开启 (1) 关闭	① 重启	更多	输入名称	或内网IP Q	C @ 🔅
	名称/ID	↓ 状态 (全部)	Ŋ 〒 │ IP地址	\$ 配置	到期时间	创建时间	镜像名称	操作
	docker-1	运行中	10.0.0.2 🖪	2核4G	2016-11-30 23:59:59	2016-10-31 21:17:52	Ubuntu-14.04.3- x86_64_2016081 91455	连接实例 更多 ▼
	docker-3	运行中	10.0.0.4 🔁	2核4G 📕	2016-11- <mark>3</mark> 0 23:59:59	201 <mark>6-10-31</mark> 21:17:52	Ubuntu-14.04.3- x86_64_2016081 91455	连接实例 更多 ▼
	docker-2	运行中	10.0.0.3 🗖	2核4G 📙	2016-11- <mark>3</mark> 0 23:59:59	201 <mark>6-1</mark> 0-31 21:17:52	Ubuntu-14.04.3- x86_64_2016081 91455	连接实例 更多 ▼

登录云服务器

● 在云服务器列表页,点击"连接实例",可通过 VNC 控制台登录主机

算资源 > 云服务器	> 实例					北京6区(VPC)	上海2区(VPC)
◆ 新建实例	□ 续费 (○ 开)	自 (1) 关闭	 ① 重启 	更多	输入名称	或内MIP Q	0 B 🔅
□ 名称/ID	≑ │ 状态 (全部) 💌	IP地址	• 配置	到期时间	创建时间 🚽	镜像名称	操作
docker-1	运行中	10.0.0.2 🔀	2核4G	2016-11-30 23:59:59	2016-10-31 21:17:52	Ubuntu-14.04.3- x86_64_2016081 91455	连接实例 更多 ▼
docker-3	运行中	10.0.0.4 🔀	2核4G 📕	2016-11-30 23:59:59	2016-10-31 21:17:52	Ubuntu-14.04.3- x86_64_2016081 91455	连接实例 更多 ▼
docker-2	运行中	10.0.0.3 内	2核4G 📕	2016-11-30 23:59:59	2016-10-31 21:17:52	Ubuntu-14.04.3- x86_64_2016081 91455	连接实例 更多 ▼

● 输入用户名和密码登录后,检测外网是否联通



如果无法 ping 通外网,请检查是否正确配置了公网 NAT。

通过 ssh 访问云服务器

如果您习惯 ssh 方式登录云服务器,我们推荐使用负载均衡(LB)将云服务器的 22 端口暴露到公网。另外,您还需要在安全组中放行 22 端口。参考负载均衡快速入门的"创建负载均衡"以及 VPC 快速入门的"创建安全组规则"。

• 创建一个名称为"docker-lb"的负载均衡,网络类型选择"公网"

新建负载均衡 ◎返回SLB列表					
	◆ 负载均衡配置	② 绑定弹性IP			
数据中心类型: VPC数据中心			负载均衡		价格:¥ 0
数据中心: 北京6区(VPC) 上海2区(VPC)			数据中心:	北京6区(VPC)	
负载均衡名称: docker-lb			VPC实例: 真实服务器类型:	docker 云服务器	
选择VPC实例: docker			网络类型:	公网	
真实服务器类型: 云服务器				总计:	¥O
网络类型法 公网 私网					-步

● 在弹性 IP 页面,选择"购买新的弹性 IP"

新建合载均衡	G返回SIB列表
DYLATE ON PARTICIPATION	a sector of the

	负载均衡配置	1		◆ 绑定弹性IP	在书时在		Maxo
第三学性1P方式: 目前可用学性IP <u>第天新的学性IP</u> 2 (戦闘典型:) 帯変値: 1Mbps	不绑定弹性IP ▼ 200Mbps	1	Mbps		以事は3個 数据中心: 负戦均衡名称: VPC实例: 真实服务器关型: 网络类型:	北京6区(VPC) docker-lb docker 云服务器 公网	M18.+ 0
计表方式: 包年包月 购买时长: - 1 + 月 1 花1(0个月的线享受一年的服务!				3単性IP 数振中心: 链路类型:	北京6区(VPC) BGP	价格:¥ 25
					带宽值: 计费方式: 购买量:	1 Mbps 包年包月 1个月 X 1个 总计:	¥25

● 支付成功后,在负载均衡列表页可以看到刚创建好的 LB 和弹性 IP,点击"进入负载均衡"可以对 LB 进行配置。

网络 > 负载均衡								北京6区(VPC)	上海2区(VPC)
➡新建负载均衡	⑤ 开启	① 关闭	② 绑定弹性IP	◎ 解绑弹性IP	意 删除				O
负载均衡名称		状态	网络类型	VPC实行	利名	3単性IP	监听器数量	操作	
docker-lb		开启	公网	docker		120.	0	进入负载均	衝

创建一个名为 ssh_1 的监听器,监听 10001 端口,真实服务器(RS)设置为 docker-1 (10.0.0.2),真实服务器的端口为 22。创建成功以后,我们即把 10.0.0.2 的 22 端口映射到了负载均衡器的 10001 端口。

基本信息					健康检查	Ť			
监听器名称: ssh	u_1				儲	康检查:	◉ 开启	◎ 关闭	
协议: TCF	•				健康检查	间隔(s):	5	C	D
监听器端口: 100	001	?			健康的	啊值(次):	5	C	D
转发方式: 轮锚	• •	?			不健康的	喝值(次):	4	C	D
会话保持: • 开	启 🔍 关闭	?							
会话保持超时: 360	00	?							
真实服务器									
可添加的服务器	输入服务器名称或IP (2		已添	加的服务器				
服务器名称	服务器IP				服务器名称	服	务器IP	服务器端口	权重
docker-3	10 <mark>.0.0.4</mark>	端口:	22] [docker-1	10	.0.0.2	22	20
docker-2	10.0.0.3	权重:	20			1			
			添加 »						
			《移除					12	

● 依次创建另外两个监听器,分别把 docker-2(10.0.0.3)和 docker-3(10.0.0.4)的 22端口,映射到负载均衡器(docker-lb)的 10002 和 10003 端口。完成后如图所示。

			創除					
监听器名称	协议	监听器端口	转发方式	监听器状态	会话保持	健康检查	后端实例数	操作
ssh_3	ТСР	10003	轮询	开启	开启	开启	1	删除
ssh_2	ТСР	10002	轮询	开启	开启	开启	1	删除
ssh_1	TCP	10001	轮询	开启	开启	开启	1	删除
最后,	我们在安全	全组中添加	u针对 2	2 端口的入	、站规则。			
1 1	网络 > VPC > d	ocker(10.0.0.0/16)	-					北京6区(VPC) 上海2
2		. entodo (=						
±	一 新建 [1]		豊味					
联网网关					安全组规则数	量	绑定	云服务器数量
道网关	DefaultCC				1		3	
幸/建按 全组 (防火墙)	Derautso				1		U	
:L	安全组: docker							
	详情	入站规则	出站规	[J] 云服务器	官息			
Ξ	+ 创建安全组排	ROJ Ť MA						
	入站规则(从外部	访问云资源)						
	一协议	行为 載	始端口 (?)	/ 结束端口	(?)	源IP		备注
旧建立人们		行为 載	始端口 (?)	结束端口 /(ToT)/~~	(?) - 没有找到亲要的数操	源IP 我~		备注
リ建安全组	bix 规则	行为 趙	始端口 (?)	│ 結束端口 /(ToT)/~/	(?) 没有找到亲要的数据。	〕 源IP 我~		备注
则建安全组		行为 1 起入站	始端口 (7)	結束端口 /(ToT)/~/	(7) 没有找到宗要的数据》 	」 源IP _{我∼}	ping	备注
则建安全组	□ ^{助议} 规则 _{方向:} 行为:	 行为 1 超 入站 允许 	始端口 (7)	結束端口 /(ToT)/~/	(7) 没有找到亲要的数据。]快费	遐ɪp _{我~} 龍方式:	ping	备注
J建安全组	□ ^{助ix} 丸则 方向: 行为: 协议:	 行为 1 起 入站 允许 TCP 	始端口 (7)	结束端口 /(ToT)/~~	(?) - 没有找到疾要的数编》] 快报]) 遐ɪp 乘~ 重方式:	ping ssh http	督注
J建安全组	助ix 助ix 方向: 行为: 协议: 起始端口:	 行为 1 入站 允许 TCP 22 	始端口 (7)	结束端口 /(ToT)/~~	(?) - 没有找到疾要的跋撮即]]]	j 遐ɪ₽ _{我∼} 重方式:	ping ssh http	督注
J建安全组	 助ix 加加 方向: 行为: 协议: 起始端口: 结束端口: 	行为 該 入站 允许 TCP 22 22	给端口 (7)	結束端ロ /(ToT)/~~	(?) - 没有找到亲要的政编印]]]]	j 遯₽ _{我~} 種方式:	ping ssh http https	督注
リ建安全组	 助ix 助ix 方向: 行为: 协议: 起始端口: 结束端口: 网段: 	行为 創 入站 允许 TCP 22 22 0.0.0.0	给端口 (7)	結束端ロ /(ToT)/~~	(7) - 没有找到亲要的数编印]]]]	遐ɪp 我~ 世方式:	ping ssh http https openvpn	备注
则建安全组	□ biù □ biù 方向: 行为: 协议: 起始端口: 结束端口: 网段: 音注:	行为 該 入站 允许 TCP 22 0.0.0.0	给端口 (7)	结束端ロ /(ToT)/~-/ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・	(?) - 没有找到亲要的数编印]]]]	遐ɪp _{我~} 世方式:	ping ssh http https openvpn remote	备注
リ建安全组	□ bix 力向: 方向: 行为: bù议: 起始端口: 结束端口: M段: 音注:	行为 創 入站 允许 TCP 22 0.0.0.0		結束端ロ /(ToT)/~-/	(7) - 没有找到亲美的数编印] 快想]	_我 ~ ^我 ~	ping ssh http https openvpn remote	备注

现在,可以通过 LB(IP 地址为 120.x.x.x)的 10001、10002、10003 端口,以 ssh 协议访问 三台云服务器。

安装 docker-engine

参考 docker 官方的<u>安装教程</u>,在 3 台云服务器上安装 docker-engine。建议选择 1.12 以上版本,以便获取 swarm mode 等特性。

-	\rightarrow $+$ \rightarrow $+$ $-$	1 1 1 ·	~ ~		1 1 11-1-
		利子 docker vergio	n 命公		docker EV A
•			$\Pi H \downarrow \checkmark)$	刑机头衣门	

root@vm10-0-0- Client:	2:~# docker version
Version:	1.12.3
API version:	1.24
Git commit:	bb644ec
Built:	Wed Oct 26 21:44:32 2016
OS/Arch:	linux/amd64
Server:	
Version:	1.12.3
API version:	1.24
Go version:	go1.6.3
Git commit:	6b644ec
Built:	Wed Oct 26 21:44:32 2016
0S/Arch:	linux/amd64

创建容器

我们以 nginx 镜像为例,在云服务器 docker-1(IP 地址为 10.0.0.2)创建一个 web 容器。

● 运行 docker run 命令,创建一个名为 web 的容器,将容器的 80 端口映射到主机的 80 端口。

root@vm10-0-0-2:~# docker run -d -p 80:80 --name web nginx 43731c27248c7e27d9271f561d508b5211e8d9dee02189127223cc95b029da60

● 运行 docker ps 命令,确定容器已经在运行

root@vm10-0-0-2:~# CONTAINER ID	docker ps IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES 43731c27248c	nginx	"nginx -g 'daemon off"	6 minutes ago	Up 6 minutes	0.0.0.0:80->80/tcp, 443/tcp

通过主机的 80 端口, 访问该容器提供的服务

将容器端口暴露到公网

推荐使用负载均衡,将容器服务的端口暴露到公网。在"通过 ssh 访问云服务器"一节,我

们已经介绍了基本步骤。您可以购买一个新的负载均衡,或者使用之前创建好的负载均衡 (docker-lb)。

● 在负载均衡 docker-lb 的配置页面,创建一个新的监听器 web_80,监听 80 端口,真实 服务器设置为 docker-1 的 80 端口

网络 > 负载均衡 > (locker-lb (弹性IP: 12	0.9					北京6	区(VPC) 上海2区(VPC)
➡创建监听器	⑦ 开启 ① 关闭	※ 修改 🏥	删除					O
监听器名称	协议	监听器端口	转发方式	监听器状态	会话保持	健康检查	后端实例数	操作
ssh_3	TCP	10003	轮询	开启	开启	开启	1	删除
ssh_2	TCP	10002	轮询	开启	开启	开启	1	删除
ssh_1	TCP	10001	轮询	开启	开启	开启	1	删除
监听器名称: we	b_80				健康检查	: • 开启	◎ 关闭	
协议: TCI	5	-			健康检查间隔(s)	: 5		?
监听器端口: 80		(?)	1		健康阀值(次)	: 5		(?)
		0			了唐南河店小厅			
转反方式:轮印	0	• (?			个健康肉值(次)	4		(7)
会话保持: • 开	启 🔍 关闭	?						
会话保持超时: 36	00	?	ří.					
docker-3	10.4 10.4	0.0.4 0.0.3 十畝 九口 左十 5	 満口: 80 权重: 20 添加 < 務報 <		docker-1	10.0.02	80	20
				чн ј∕ ∖ ≄µ)	50,73			
子网	MIRE > VPC > do	LKEI(10.0.0/16) -					A DOPU	L'astr(ALC)
路由	● 新建 ⑨	え制安全组 前	111)除					
互联网网关					安全组规则数量		「绑定云服务器	敗量
隧道网关					2		3	
安全组 (防火墙)	Delaurisa						U	
ACL	安全组: docker	1-44 ALE MA			=			>
	详情	入站规则	出站规则	云服务器	言思			
-	+ 创建安全组规	創意删除						
	入站规则(从外部说	问云资源)						
	协议	行为 起如	台端口 (?)	结束端口	(?)	源IP	备注	

22

0.0.0.0/0

ТСР

接受

22

创建安全组规则

:回己	入站	•	快速方式;	ping
行为:	允许	Ŧ		ssh
协议:	ТСР	¥	Г	http
起始端口:	80		L	
结束端口:	80			https
网段:	0.0.0	/ 0		openvpn
备注:		(选埴)		remote
				ID
				创建 取消
现在,我们可以;	通过负载均衡的	的 IP 地址,在浏	览器中访问邻	容器的 web 服务
Welcome to nginx! X	• inter- • •	inters. F Bangh.	 Region. 	· • • • • • • • • • • • • • • • • • • •
- C 0 120.3				
		Welcome to	o nginx!	
		If you see this page, the working. Further configu	nginx web server is ration is required.	s successfully installed and
		For online documentation Commercial support is av	n and support pleas vailable at nginx.cor	e refer to <u>nginx.org</u> . m.

Х

安装集群管理系统

通过上面的步骤,我们已经安装了 docker engine,并且可以创建容器。但是,单个容器的作 用是有限的。在一个实际的系统中,往往需要管理大量容器,并且组织容器之间的关系以便 构建分布式服务,这时,要借助容器集群管理系统。

目前,主流的容器集群管理系统有 Kubernetes、Mesos、Swarm 等。您可以选择并安装自己 熟悉的系统。如果您之前没有了解过这些系统,推荐您使用 Swarm。Swarm 是 docker 官方 的集群管理工具,并且在 docker 1.12 版本以后,Swarm 已经内置在 docker engine 里,安装 和配置都比较简单。

创建 Swarm

参考官方的教程: <u>https://docs.docker.com/engine/swarm/swarm-tutorial/</u>

• 我们的 3 台云服务器,设定其中一台为管理节点(manager),另外两台为工作节点(worker)

节点名称	节点 IP 地址	角色
docker-1	10.0.0.2	manager
docker-2	10.0.0.3	worker
docker-3	10.0.0.4	worker

• 在管理节点上,执行 swarm init 命令,创建一个新的 swarm

root@vm10-0-0-2:~# docker swarm initadvertise-addr 10.0.0.2 Swarm initialized: current node (dqxpss15jjq510817dxn3webi) is now a manager.							
To add a worker to this swarm, run the following command:							
docker swarm join \ token SWMTKN-1-194rumn9pqng50jn2sygvza7mw7io7vh4jahx0izz1v8ulue8i-4dep8r9yyf0w45y69xb6jru8 10.0.0.2:2377							
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.							
● 在两个工作节点上,执行 swarm join 命令,将工作节点加入集群							
root@vm10-0-0-3:~# docker swarm join \ > -token SMMTKN-1-194rumn9pqng50jn2sygvza7mw7io7vh4jahx0izz1v8ulue8i-4dep8r9yyf0w45y69xb6jru8q \ > 10.0.0.2:2377 This node joined a swarm as a worker.							
root@vm10-0-0-4:~# docker swarm join \ > -token SMMTKN-1-194rumn9pqng50jn2sygvza7mw7io7vh4jahx0izz1v8ulue8i-4dep8r9yyf0w45y69xb6jru8q \ > 10.0.0.2:2377 This node joined a swarm as a worker.							
● 在管理节点上,执行 node ls 命令,确认集群创建成功并且包含 3 个节点							
root@vm10-0-0-2:~# docker node ls ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS 7jfsome23nixu0ohhb068k6r8 vm10-0-0-3.ksc.com Ready Active 7qw3g03wm16kgy3bnz1sivxvf vm10-0-0-4.ksc.com Ready Active dqxpss15jjg510817dxn3webi * vm10-0-0-2.ksc.com Ready Active Leader							

创建服务

参考官方教程: <u>https://docs.docker.com/engine/swarm/services/</u>

 在管理节点上,执行 service create 命令,创建一个名为 my_web 的服务,副本数设置为 3,公开端口(publish port)为 8080

root@vm10-0-0-2:~# docker service create --name my_web --replicas 3 --publish 8080:80 nginx 21f8cesimbalzd77fwn9qyad1

• 在管理节点上,执行 service ps 命令,可以看到 3 个容器被调度到了集群中运行

root@vm10-0-0-2:~# docker	service ps	my_web			
ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
8nxgdcg46a1539nba1xh3bs9u	my_web.1	nginx	∨m10-0-0-4.ksc.com	Running	Running 3 minutes ago
3deo9do99yx0cz1hpyxmw8waa	my_web.2	nginx	∨m10-0-0-3.ksc.com	Running	Running 2 minutes ago
Opeaxhy3v2uutg7dg2jotyszg	my_web.3	nginx	∨m10-0-0-3.ksc.com	Running	Running 2 minutes ago

● 在集群的任意节点上,访问节点的 8080 端口,可以访问该服务。



将服务暴露到公网

使用负载均衡把 my_web 发布到公网。注意,前面创建的负载均衡 docker-lb,其 80 端口已 经被占用。如果继续使用这个负载均衡,只能选择其他端口(下面的示例,选择 8080 端口)。 或者,购买一个新的负载均衡发布到 80 端口。

● 在负载均衡 docker-lb 的配置页面,创建一个新的监听器 web_8080,监听 8080 端口, 真实服务器设置为 docker-1、docker-2、docker-3 的 8080 端口

网络 > 负载均衡 >	docker-lb (弹性IP: 120.	- (北京6区(VPC) 上海2区(VPC)
+ 创建监听器	⑤ 开启 ① 关闭	※修改 前期除						0
监听器名称	协议	监听器端口	转发方式	监听器状态	会话保持	健康检查	后端实例数	操作
web_80	ТСР	80	轮询	开启	开启	开启	1	删除
ssh_3	ТСР	10003	轮询	开启	开启	开启	1	趣味
ssh_2	ТСР	10002	轮询	开启	开启	开启	1	删除
ssh_1	ТСР	10001	轮询	开启	开启	开启	1	删除

基本信息			健康检查		
监听器名称:	web_8080		健康检查:	 开启 关闭 	
协议:	тср 👻		健康检查间隔(s):	5	?
监听器端口:	8080	?	健康阀值(次):	5	?
转发方式:	轮询 🔻	(?)	不健康阀值(次):	4	?
会话保持:	 开启 关闭 	?			
会话保持超时:	3600	?			
真实服务器					
可添加的服务器	输入服务器名称或IP	Q	已添加的服务器		
服务器名	你 服务器IP	39	服务器名称 服务	5器IP 間	跟务器端口 权重
		端口: 8080	docker-3 10.	0.0.4 8	080 20

端口:	8080	docker-3	10.0.0.4	8080	20
权重:	20	docker-1	10.0.0.2	8080	20
	添加 »	docker-2	10.0.0.3	8080	20
	《 移除				

• 在 VPC 安全组中, 增加针对 8080 端口的入站规则

← 返回	网络 > VPC >	docker(10.0.0.0	0/16) -				北京6区(VP	C) 上海2区(VPC)
• 子网	+新建	🕤 复制安全组	<u>─</u>					
 路由 互联网网关 	名称				安全组规则	数量	绑定云服务器数量	
• 隧道网关	docker				3		3	
◎ 对等连接	DefaultSG				1		0	
 安全组(防火墙) ACL 	安全组: dock 详情	er 入站	规则 出站规则	云服务器信息				×
Ξ	▲ 创建安全线 入站规则(从外	11規则 11前向云资源)	lê.					
	一协议	行为	起始端口 (?)	结束端](?)	源IP	备注	
	ТСР	接受	80	80		0.0.0/0		
	ТСР	接受	22	22		0.0.0/0		

.

行为:	分许	•		
11/1	7047		ssh	
协议:	ТСР	٣	http	
起始端口:	8080			
结束端口:	8080		https	
网段:	0.0.0.0	/ 0	openvpn	
备注:		(选填)	remote	
			IP	
				_
			创建	



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to <u>nginx.org</u>. Commercial support is available at <u>nginx.com</u>.

Thank you for using nginx.

使用 KS3 存储 docker 镜像

Docker Registry 是构建 docker 镜像仓库的核心组件, docker 官方提供了多种存储驱动, 使得用户可以把镜像存储到本地磁盘或者亚马逊 S3 等云储存服务上。

KS3(Kingsoft Standard Storage Service)是金山云提供的无限制、多备份、分布式的低成本存储空间服务,解决存储扩容、数据可靠安全以及分布式访问等相关复杂问题。

如果您想构建私有镜像仓库并且使用金山云对象存储(KS3)作为后端存储,请使用我们<u>定</u>制的 registry 镜像: ksyun/registry。详细的配置,请参考<u>配置文档</u>。

搭建企业级的镜像仓库

我们推荐您使用 Harbor,结合金山云 KS3 存储,构建企业级镜像仓库服务。

Harbor 是 VMware 中国研发中心开发的一款镜像仓库管理工具,提供了企业级用户需要的 用户管理、权限管理、镜像复制、LDAP/AD 支持等功能。

本节介绍了如何使用 KS3 + Harbor 搭建企业级镜像仓库。

安装 Docker

如果您还没有安装 docker,执行以下命令安装 Docker curl -fsSL https://get.docker.io | bash

安装 docker-compose

默认的官方文档 安装命令如下:

curl -L https://github.com/docker/compose/releases/download/1.6.2/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

下载安装包

Harbor 发行版的安装包可以在 Github 下载,有在线(online)安装和离线(offline)安装两种安装方式选择。 以在线安装示例,通过 wget 命令下载,使用 tar 命令将其解压:

wget https://github.com/vmware/harbor/releases/download/0.4.5/harbor-online-installer-0.4.5.tgz tar -zxvf harbor-online-installer-0.4.5.tgz

配置 Harbor

Harbor 的常用配置参数主要存放在 harbor.cfg, 直接编辑 harbor.cfg 即可:

vim harbor.ctg	
# 其配置信息如下	
hostname = reg.yourdomain.com	# Harbor 服务器域名
ui_url_protocol = https	# UI 组件访问协议
email_server = smtp.mydomain.com	# email 服务器地址

```
# email 端口
email_server_port = 25
                                             # email 账号
email username = sample admin@mydomain.com
                                           # email 密码
email_password = abc
email_from = admin <sample_admin@mydomain.com> # email 发件人
email_ssl = false
                                          # 是否启用 SSL
harbor_admin_password = Harbor12345
                                           # Harbor 初始化管理员(admin)密码
auth_mode = db_auth
                                           # 权限管理模型(db_auth/ldap_auth)
                                            # ldap 地址
ldap_url = ldaps://ldap.mydomain.com
ldap_basedn = uid=%s,ou=people,dc=mydomain,dc=com # ldap 权限模型
db_password = root123
                                           # 数据库 管理员密码
self_registration = on
                                           # 是否打开自动注册
                                           # 是否启用压缩 js
use_compressed_js = on
max_job_workers = 3
                                           # 最大任务数
                                           # token 超时
token_expiration = 30
verify_remote_cert = on
                                           # 是否验证远程证书
                                           # 是否启用自定义证书
customize_crt = on
# 以下为自定义证书信息
crt_country = CN
crt_state = State
crt_location = CN
crt_organization = organization
crt_organizationalunit = organizational unit
crt_commonname = example.com
crt_email = example@example.com
```

配置后端存储到 KS3

在默认情况下,Harbor的镜像存储在本地磁盘/data/registry目录下。但是在生产环境中,出于高可用、高吞吐、安全可靠等因素,我们可能需要将镜像数据存储在 KS3、Ceph 等。

金山云提供了基于 Docker registry v2 封装了 KS3 驱动的官方镜像 ksyun/registry。该镜像提供了 Docker registry v2 和 KS3 的完美对接,通过该镜像,用户可以很方便将镜像仓库数据存储到 KS3 上。

使用 KS3 作为 Harbor 镜像仓库的后端存储, 你需要两步配置:

首先,修改 harbor/docker-compose 文件,修改 registry 模块,以支持 KS3 驱动的官方镜像。 regsitry:

```
image: ksyun/registry:latest # 修改为支持 KS3 驱动的镜像
container_name: registry
restart: always
volumes:
    # 此处注释掉 harbor 默认挂在本地目录 vlomues 的存储方式
```

```
# - /data/registry:/storage
- ./common/config/registry/:/etc/registry/
environment:
- GODEBUG=netdns=cgo
command:
["serve", "/etc/registry/config.yml"]
depends_on:
- log
logging:
driver: "syslog"
options:
syslog-address: "tcp://127.0.0.1:1514"
tag: "registry"
....
```

其次, 配置 template/registry/config.yml 文件, 修改 storage 模块, 为 KS3 的配置数据。

```
storage:
 cache: inmemory
 ks3:
  accesskey: "your ks3 accesskey" # KS3 的访问私钥
                                  # KS3 的访问秘钥
  secretkey: "your ks3 secretkey"
                                 #默认值 false,是否使用内网加速, true 代表使用内网加速, (高速,
  internal: false
不计流量)
  region: ks3-cn-beijing
                                  # 区域设置
  bucket : "your bucket name"
                                  # KS3 存储桶的名字
                                  # 默认值 false,是否对镜像数据加密
  encrypt: false
  secure: false
                                  # 默认值 false,是否使用 https (ssl)
                                  # 块大小
  chunksize: 5242880
  storage_page: /
                                  #存储路径
```

更多 KS3 存储配置,参见 Docker Registry KS3 存储驱动配置: https://github.com/softlns/distribution/blob/master/docs/storage-drivers/ks3.md。

配置 HTTPS

```
创建 CA 证书
```

```
openssl req \
```

```
-newkey rsa:4096 -nodes -sha256 -keyout ca.key \
-x509 -days 365 -out ca.crt
```

证书签名

```
openssl req \
    -newkey rsa:4096 -nodes -sha256 -keyout yourdomain.com.key \
    -out yourdomain.com.csr
```

初始化 CA 信息

mkdir demoCA
cd demoCA
touch index.txt
echo '01' > serial
cd ..

配置 Nginx

```
# 复制证书
cp registry.mritd.me.crt config/nginx/cert
cp ca/registry.mritd.me.key config/nginx/cert
# 备份配置
mv config/nginx/nginx.conf config/nginx/nginx.conf.bak
# 使用模板文件
mv config/nginx/nginx.https.conf config/nginx/nginx.conf
# 修改 nginx.conf 配置
server {
    listen 443 ssl;
    server_name harbordomain.com;
    ...
server {
    listen 80;
```

server_name harbordomain.com;
rewrite ^/(.*) https://\$server_name:443/\$1 permanent;

安装并启动

配置完成后,执行 Harbor 提供的安装脚本(install.sh),来生成私有配置,在线安装版本会到 Docker Hub 拉取镜像,并启动 Harbor 服务。

sudo ./install.sh

测试

访问 Harbor UI

在浏览器中打开 http://reg.yourdomain.com/ , 以 admin 账号(初始账号/密码: admin/Harbor12345)登录,可以查看 Harbor 的控制面板。

测试 docker client 登录、推拉镜像

在控制面板(Harbor UI)中创建一个项目 myproject, docker client 登录认证,并测试推送私 有镜像:

docker login reg.yourdomain.com

docker push reg.yourdomain.com/myproject/myrepo:mytag

管理 Harbor 生命周期

你可以通过 docker-compose 工具来管理 Harbor 的生命周期,常用命令如下:

```
# 停止 Harbor
$ sudo docker-compose stop
Stopping harbor_proxy_1 ... done
Stopping harbor_ui_1 ... done
Stopping harbor_registry_1 ... done
Stopping harbor_mysql_1 ... done
Stopping harbor_log_1 ... done
Stopping harbor_jobservice_1 ... done
```

启动 harbor

```
$ sudo docker-compose start
Starting harbor_log_1
Starting harbor_mysql_1
Starting harbor_registry_1
Starting harbor_ui_1
Starting harbor_proxy_1
Starting harbor_jobservice_1
```

更新 Harbor 配置

\$ sudo docker-compose down

\$ vim harbor.cfg

\$ sudo install.sh

基于 Docker 的持续集成、持续交付

本节介绍了,如何通过 gitlab、jenkins 等基础服务来搭建持续集成、持续部署流水线。

搭建基础服务

CI、CD 涉及的基础服务有:

- 代码仓库(以 Gitlab 示例)
- 镜像仓库(以搭建私有镜像仓库示例)
- 代码构建(以 Jenkins 示例)
- 容器集群(以 swarm 示例)

代码仓库(构建 gitlab 服务)

GitLab 是一个利用 Ruby on Rails 开发的开源应用程序,实现一个自托管的 Git 项目仓库,可通过 Web 界面进行访问公开的或者私人项目;类似于 Github。

选一台主机以 docker 化的方式来搭建 Gitlab 服务。(已有私有 Gitlab 仓库或使用 Github,可跳过此步骤)

我们选择 Docker Hub 上的 sameersbn/gitlab 镜像来快速构建 Gitlab 服务。根据 sameersbn/docker-gitlab 给出的文档,我们需要启动三个容器组件:

```
# 启动 postgresql 容器
docker run --name gitlab-postgresql -d \
   --env 'DB_NAME=gitlabhq_production' \
   --env 'DB_USER=gitlab' \
   --env 'DB_PASS=password' \
   --env 'DB_EXTENSION=pg_trgm' \
   --volume /srv/docker/gitlab/postgresql:/var/lib/postgresql \
   sameersbn/postgresql:9.5-3
# 启动 redis 容器
docker run --name gitlab-redis -d ∖
   --volume /srv/docker/gitlab/redis:/var/lib/redis \
   sameersbn/redis:latest
# 启动 gitlab 容器
docker run --name gitlab -d ∖
   --link gitlab-postgresql:postgresql --link gitlab-redis:redisio \
   --publish 10022:22 --publish 10080:80 \
   --env 'GITLAB_PORT=10080' \
```

```
--env 'GITLAB_SSH_PORT=10022' \
```

```
--env 'GITLAB_SECRETS_DB_KEY_BASE=long-and-random-alpha-numeric-string' \
--env 'GITLAB_SECRETS_SECRET_KEY_BASE=long-and-random-alpha-numeric-string' \
--env 'GITLAB_SECRETS_OTP_KEY_BASE=long-and-random-alpha-numeric-string' \
--volume /srv/docker/gitlab/gitlab:/home/git/data \
sameersbn/gitlab:latest
```

测试:

执行 docker ps 命令查看容器运行状态。

浏览器访问 http://host:port, 查看 gitlab 网页能够正常打开,并设置 root 账号初始密码。

镜像仓库(构建 Docker 私有仓库服务)

在 CI、CD 中, 需要一个 Docker 镜像仓库来存放每次构建的镜像。从镜像的安全、可靠、 访问速度等因素考虑, 搭建一个私有的镜像仓库, 对企业级开发和实践是很有必要的, 推荐 使用 KS3 <u>搭建高可用的企业级私有镜像仓库</u>。

如果仅为了测试,可以使用 registry 镜像搭建一个简单的仓库:

docker run -d --restart=always --name registry \
-v /mnt/docker/registry:/var/lib/registry -p 15000:5000 registry:2

测试:

执行 docker ps 命令查看容器运行状态。 测试镜像的推送/拉取(docker push/docker pull)

注: docker 私有镜像仓库 拉取/推送 失败,提示不能使用 http 连接。解决方法:

1、设置 https 访问,参见企业级镜像仓库的搭建。

2、在 docker 启动参数里添加--insecure-registry ip:port, 然后重启 docker。

代码构建(构建 Jenkins 服务)

Jenkins 是基于 Java 开发的一种开源持续集成工具,具有开源,支持多平台和插件扩展,安装简单,界面化管理等特点。Jenkins 使用 job 来描述每一步工作,节点是用来执行项目的环境。Master 节点是 Jenkins job 的默认执行环境,也是 Jenkins 应用本身的安装环境。

由于 Jenkins 本身版本比较老,插件多,但许多插件缺少维护更新,加上本身环境比较陈旧, 且各工程之间插件依赖关系难以管理;因此建议使用 Docker 来搭建 Jenkins 服务。

在 Jenkins 容器中调用 docker 的方式有三种:

1、通常在单节点下,可将节点 docker 挂载到 docker 容器中,需添加如下参数:
 -v /var/run/docker.sock:/var/run/docker.sock -v \$(which docker):/usr/bin/docker

2、通常在集群中,可以使用 jenkins 的 docker 集群,连接到 Docker server REST API (http://master-ip:2375),需安装一下插件:

* Docker plugin

* Docker Commons Plugin

* docker-build-step

3、通常在 jenkins 的集群中,插件的方式并不是很方便,仍然需要在 shell 中执行 docker 命 令,这时候可以使用 DinD (Docker- in-Docker)的方式来构建 jenkins 服务,参见 Github 项 目 <u>https://github.com/jpetazzo/dind</u>。

这里我们使用单节点的方式,使用 Docker Hub 上金山云的 ksyun/jenkins 镜像来构建 jenkins 服务:

```
docker run docker run -d -p 8080:8080 \
    --name jenkins --restart=always
    -v /mnt/jenkins_home:/var/jenkins_home \
    -v /var/run/docker.sock:/var/run/docker.sock
    --env 'DOCKER_SOCKET=/var/run/docker.sock' \
    --env 'DOCKER_GROUP=dockerhost' \
    --env 'DOCKER_HOST=unix:///var/run/docker.sock' \
    --env 'JENKINS_USER=jenkins' ksyun/jenkins
```

测试: 执行 docker ps 命令查看容器运行状态。 在浏览器,打开 http://ip:port,进入 jenkins。

容器集群(构建 Docker Swarm 集群)

详见创建 Swarm

Demo: 基于 CI、CD 的 Flask 项目

以上,我们搭建了基于 Docker 的 CI、CD 的基础服务,下面我们创建一个 Flask 项目的 Demo 来测试下:

- 在 gitlab 创建 python-hello-world 项目仓库
- clone 项目,本地开发
- 配置 jenkins
- 触发构建

Gitlab 上新建项目

在 gitlab 创建 python-hello-world 项目仓库。

		Р	
	pyth	on-hello-world 🛛	
 1			
 0	HIIP -	http://localnost/root/pvthon-	

本地开发

编写 Flask 应用 创建 app.py from flask import Flask app = Flask(__name__) @app.route('/') def hello_world(): return "Hello World!
 version: v1 author: lain" if __name__ == '__main__': app.run(host="0.0.0", port=5000)

创建 requirements.txt flask

编写 Dockerfile

From python:2.7
MAINTAINER lain
ADD . /app
WORKDIR ./app
RUN pip install -r requirements.txt
EXPOSE 5000
CMD ["python", "/app/hello.py"]

创建构建部署脚本 build_deploy.sh

#!/bin/bash
#build in jenkins
docker 私有仓库的地址
REG_URL=10.0.0.4:15000
swarm 集群 manage 节点的地址
SWARM_MANAGE_URL=tcp://10.0.0.2:2375
service 名称
SERVICE_NAME=\$JOB_NAME
根据时间生成 tag
TAG=\$REG_URL/\$JOB_NAME:`date +%y%m%d-%H-%M`
使用项目目录下的 Dockerfile 文件打包
docker build -t \$TAG \$WORKSPACE/.

```
docker push $TAG

docker rmi $TAG

# 创建或更新服务

if docker -H $SWARM_MANAGE_URL service ls | grep -i $SERVICE_NAME; then

docker -H $SWARM_MANAGE_URL service update $SERVICE_NAME --image $TAG

else

docker -H $SWARM_MANAGE_URL service create --name $SERVICE_NAME --replicas 3 -p 15000:5000 $TAG

Fi
```

配置 Jenkins

在 Jenkins 上创建一个 python-hello-world 项目,选择自由风格即可:

😰 Jenkins			风查找	
Jenkins 🕨				
쓸 新建		Item名称	python-hello-world	
鵗 用户		● 构建-	一个自由风格的软件项目	
🥏 任务历史			这是Jenkins的主要功能.Jenkins将会结合任何SCM和任何构建系统来构建你的项目, 甚至可以构建软件	以外的系
💥 系统管理		◎ 构建-	一个maven项目	
Credentials		Cutor	构建一个maven项目.Jenkins利用你的POM文件,这样可以大大减轻构建配置.	
🍓 My Views		© Exter	THAI JODD 这个类型的任务允许你记录执行在外部Jenkins的任务,任务甚至运行在远程机器上这可以让Jenkins作: 个文档查看详细内容.	为你所有目
构建队列	-	○ 构建-	一个多配置项目	
队列中没有构建任务			适用于多配置项目,例如多环境测试,平台指定构建,等等.	
构建执行状态	-	ок		
1 空闲				
2 空闲				
殳置 git:				
源码管理				
None				
CVS				
CVS Projectset				
Git Repositories				
repositories	Repository URL	http://192.168.4	40.129:10080/root/python-hello-world.git	
	Credentials	- none -	•	í.
				1
		e Add		(?)
			高级	
			Add Repository Delete Repository	
				52
Branches to build	Branch Specifier	(blank for 'any')	*/mostor	0
			Thiaster	
			Add Branch Delete Branch	
源码库浏览器	(自动)			• 🕐
保存应用	1			

设置构建触发器,这里设置每分钟拉取一次,也可设置 gitlab hook:

构建触发器		
🔲 触发远程构建((例如,使用脚本)	0
Build after other	er projects are built	0
Build periodica	dly	0
Poll SCM		0
日程表	•••••	
		0
	Do you really mean "every minute" when you say "* * * * * "? Perhaps hour	you meant "H * * * *" to poll once per
Ignore post-comm	hit hooks	0

设置构建需要执行的脚本,这里设置成 build_deploy.sh:

Execute	shell	0
Command	<pre>sh \$WORKSPACE/build_deploy.sh</pre>	

触发 CI、CD

推送到代码到 gitlab 的 master 分支,然后可在 jenkins console 查看到构建集成和部署的信息。

-, J. Successfully installed Jinja2-2.8 MarkupSafe-0.23 Werkzeug-0.11.11 click-6.6 flask-0.11.1 itsdangerous-0.24 [91mYou are using pip version 8.1.2, however version 9.0.1 is available. You should consider upgrading via the 'pip install —upgrade pip' command. [Om ---> b3abf3f55f6c Removing intermediate container 7cb13d944ed2 Step 6 : EXPOSE 5000 ---> Running in 6d3141eb9752 ----> 892407930e3b Removing intermediate container 6d3141eb9752 Step 7 : CMD python /app/hello.py ---> Running in f29cb8dd57be ---> e0afe0be0eac Removing intermediate container f29cb8dd57be Successfully built eOafeObeOeac The push refers to a repository [10.0.0.4:5000/python-hello-world] 230952785ef3: Preparing 30a7e9b72alf: Preparing 79ae05a40127: Preparing 160abe12c7c6: Preparing 58f7a855e280: Preparing 787c930753b4: Preparing 9f17712cbaOb: Preparing 223c0d04a137: Preparing fe4c16cbf7a4: Preparing 787c930753b4: Waiting 9f17712cbaOb: Waiting 223c0d04a137: Waiting fe4c16cbf7a4: Waiting 30a7e9b72a1f: Pushed 79ae05a40127: Pushed 230952785ef3: Pushed 58f7a855e280: Pushed 160abe12c7c6: Pushed 223c0d04a137: Pushed fe4c10cbf7a4: Pushed 9f17712cbaOb: Pushed 787c930753b4: Pushed 161118-07-46: digest: sha256:f473d34889b0bbb8e7cbd19cfe7ab7c51027a7e0c7b2af0b1e1e0c44d771d811 size: 2220 Untagged: 10.0.0.4:5000/python-hello-world:161118-07-46 Untagged: 10.0.0.4:5000/python-hello-world@sha256:f473d34889b0bbb8e7cbd19cfe7ab7c51027a7e0c7b2af0b1e1e0c44d771d811 Deleted: sha256:e0afe0be0eac86d317d03aa1f105f268ed114f830b91e1384349cb4305f7bcf0 Deleted: sha256:892407930e3b500ad79c728f89c5d6110652578d0635bf1a6de245d134734ea0 Deleted: sha256:b3abf3f55f6ccdf4d6a9c6119ebb556a8a5388f0813e8361ebf7da77ea2e35ed Deleted: sha256:0d607ce2febc8654a07d2d326195abeeb63974b666c67c93d81b0e60bb510733 1phb6uxst5qfq68kdg0sr5ojg Finished · SUCCESS

访问测试

在 swarm 集群主节点上查看当前 service:



更新代码版本,并推送到代码仓库 gilab:

from flask import Flask

```
app = Flask(__name__)
@app.route('/')
def hello_world():
    return "Hello World! <br/> version: v2 author: lain"
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5000)
```

等待 2 分钟,可以在 jenkins 查看到新的构建是否成功。 在 master 节点查看 service 的容器状态:



访问浏览器:



Hello World!
 version: v2 author: lain