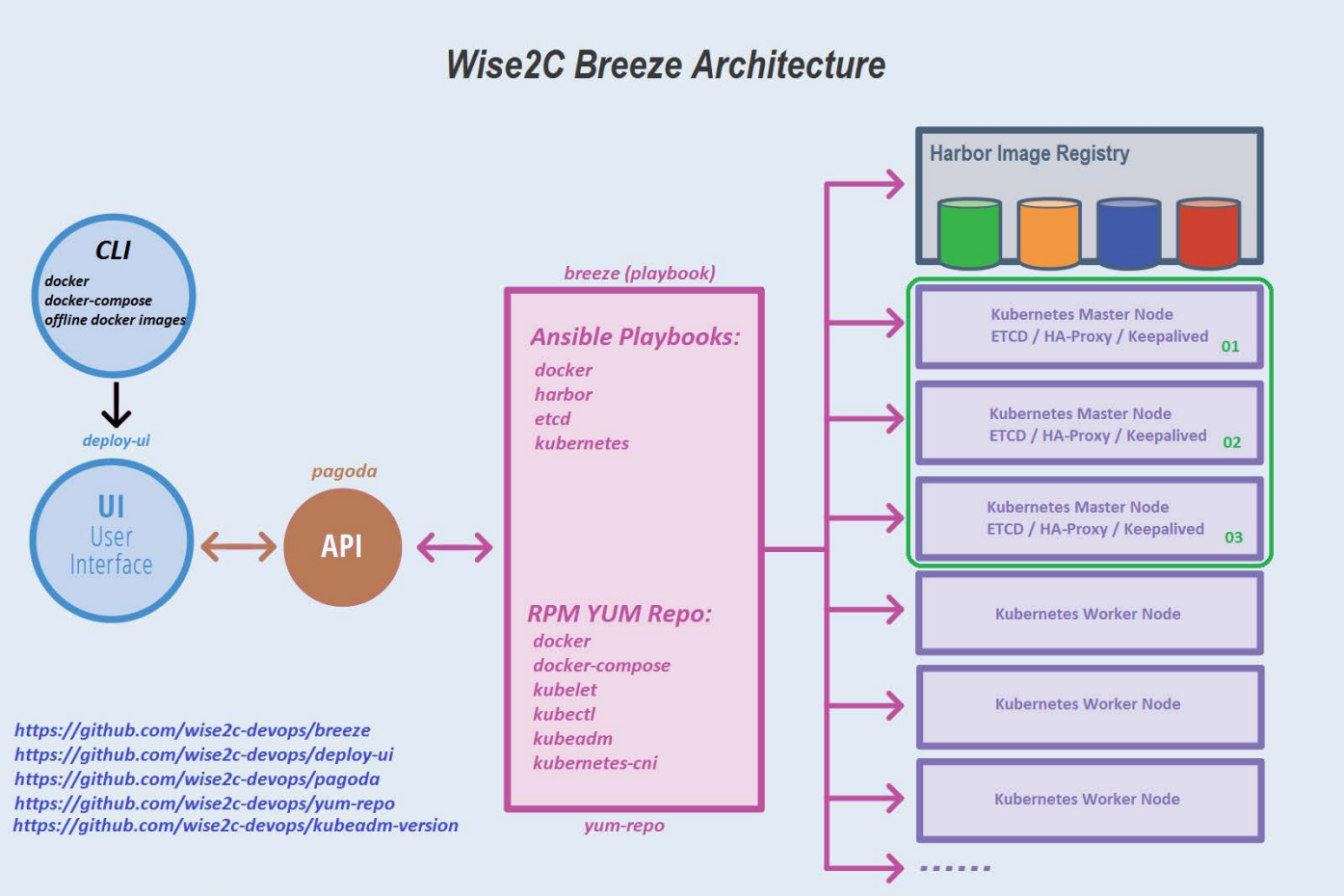
# 使用睿云智合开源 Breeze 工具部署 Kubernetes 1.12.1 高可用集群

Breeze 项目是深圳睿云智合所开源的 Kubernetes 图形化部署工具，大大简化了 Kubernetes 部署的步骤，其最大亮点在于支持全离线环境的部署，且不需要翻墙获取 Google 的相应资源包，尤其适合某些不便访问互联网的服务器场景。（项目地址 https://github.com/wise2c-devops/breeze ）

Breeze 开源工具由以下子项目构成：

1. **playbook（breeze）** 该项目由不同的 ansible playbook 构成，分别是 docker、etcd、registry、kubernetes
2. **yum-repo** 该项目用于为安装过程中提供离线的 yum repo 源，包含了 docker、kubelet、kubectl、kubeadm、kubernetes-cni、docker-compose 等 rpm 包库，除此之外我们还包括了可能会用到的 ceph 及 nfs 相关 rpm
3. **deploy-ui** 用户前端 UI，采用 vue.js 框架实现
4. **pagoda** 实现了对 ansible 脚本调用的 API 集
5. **kubeadm-version** 输出 kubernetes 组件镜像版本信息
6. **haproxy** 用于安装负载均衡的镜像及启动脚本
7. **keepalived** 为负载均衡实现统一入口虚 IP 的组件镜像及启动脚本

Breeze 软件架构简图：




用户通过 Breeze 工具，只需要一台安装有 Docker 及 docker-compose 命令的服务器，连接互联网下载一个对应 Kubernetes 版本的 docker-compose.yaml 文件即可将部署程序运行出来，对部署机而已，只需能有普通访问互联网的能力即可，无需翻墙，因为我们已经将所有 Kubernetes 所需要的 docker 镜像以及 rpm 包内置于 docker image 里了。

如果需要离线安装，也是极其容易的，只需要将 docker-compose.yaml 文件里涉及的 docker 镜像保存下来，到了无网环境预先使用 docker load 命令载入，再运行 docker-compose up -d 命令即可无网运行部署程序。所有被部署的集群角色服务器，完全无需连入互联网。

该项目开源，用户可以很方便的 fork 到自己的 git 账号结合 travis 自动构建出任意 Kubernetes 版本的安装工具。

在我们的实验环境中准备了六台服务器，配置与角色如下（如果需要增加 Minion/Worker 节点请自行准备即可）：

| 主机名 | IP 地址 | 角色 | OS | 组件 |
|---|---|---|---|---|
| deploy | 192.168.9.10 | Breeze Deploy | CentOS 7.5 x64 | docker / docker-compose / Breeze |
| k8s01 | 192.168.9.11 | K8S Master | CentOS 7.5 x64 | K8S Master / etcd / HAProxy / Keepalived |
| k8s02 | 192.168.9.12 | K8S Master | CentOS 7.5 x64 | K8S Master / etcd / HAProxy / Keepalived |
| k8s03 | 192.168.9.13 | K8S Master | CentOS 7.5 x64 | K8S Master / etcd / HAProxy / Keepalived |
| k8s04 | 192.168.9.14 | K8S Minion Node | CentOS 7.5 x64 | K8S Worker |
| registry | 192.168.9.20 | Harbor | CentOS 7.5 x64 | Harbor 1.6.0 |
|  | 192.168.9.30 | VIP |  | HA 虚 IP 地址在 3 台 K8S Master 浮动 |

步骤：

一、准备部署主机（deploy / 192.168.9.10）

（1）以标准 Minimal 方式安装 CentOS 7.5 (1804) x64 之后，登录 shell 环境，执行以下命令开放防火墙：

```
setenforce 0
sed --follow-symlinks -i "s/SELINUX=enforcing/SELINUX=disabled/g" /etc/selinux/config
firewall-cmd --set-default-zone=trusted
firewall-cmd --complete-reload
```

（2）安装 docker-compose 命令

```
curl -L https://github.com/docker/compose/releases/download/1.21.2/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose
```

（3）安装 docker

```
yum install docker
```

（4）建立部署主机到其它所有服务器的 ssh 免密登录途径

a）生成秘钥，执行：

```
ssh-keygen
```

b）针对目标服务器做 ssh 免密登录，依次执行：

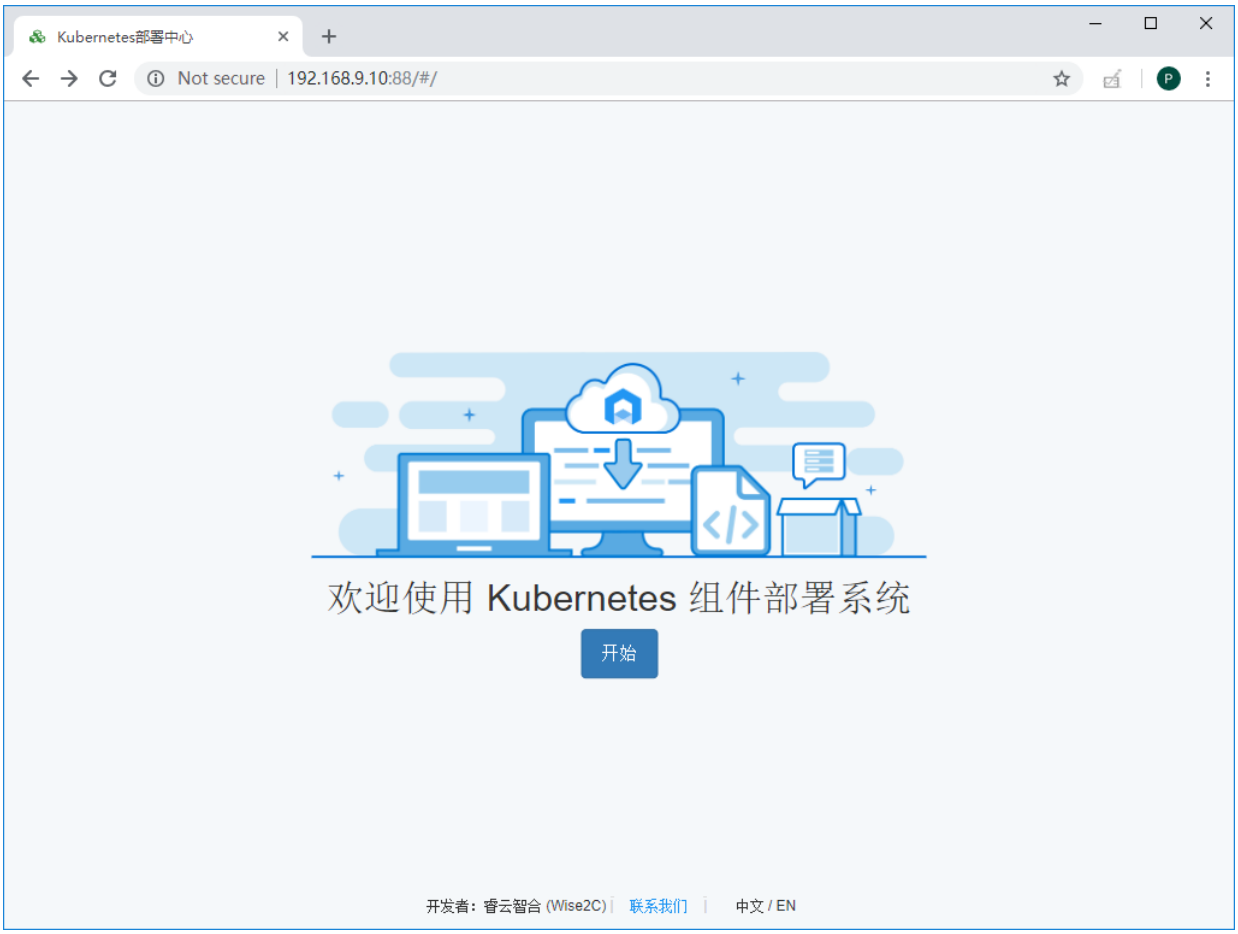
```
ssh-copy-id 192.168.9.11
ssh-copy-id 192.168.9.12
ssh-copy-id 192.168.9.13
ssh-copy-id 192.168.9.14
ssh-copy-id 192.168.9.20
```

二、获取针对 K8S 某个具体版本的 Breeze 资源文件并启动部署工具，例如此次实验针对刚刚发布的 K8S v1.12.1

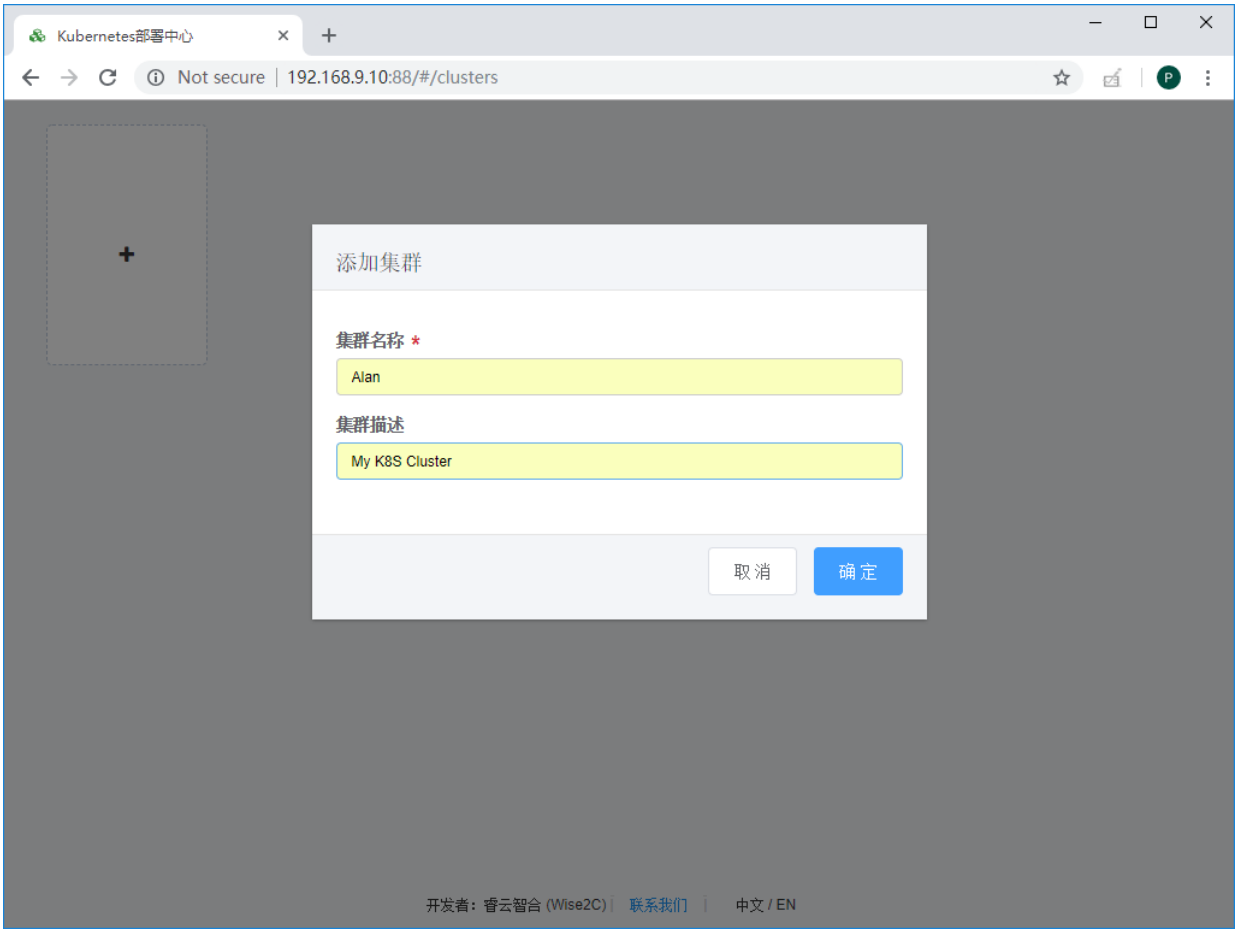
```
curl -L https://raw.githubusercontent.com/wise2ck8s/breeze/v1.12.1/docker-compose.yml -o docker-compose.yml
docker-compose up -d
```

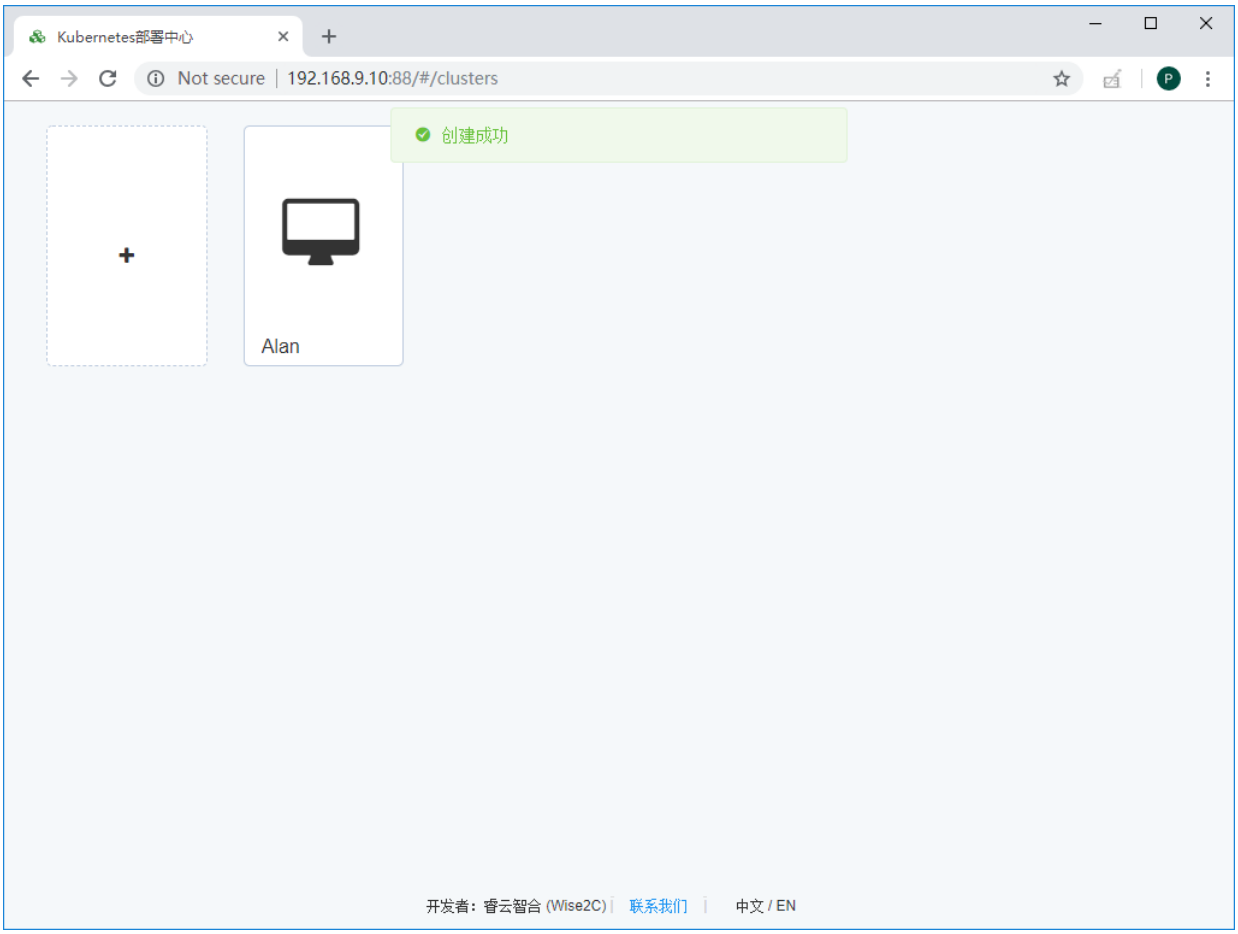
三、访问部署工具的浏览器页面(部署机 IP 及端口 88)，开始部署工作

http://192.168.9.10:88

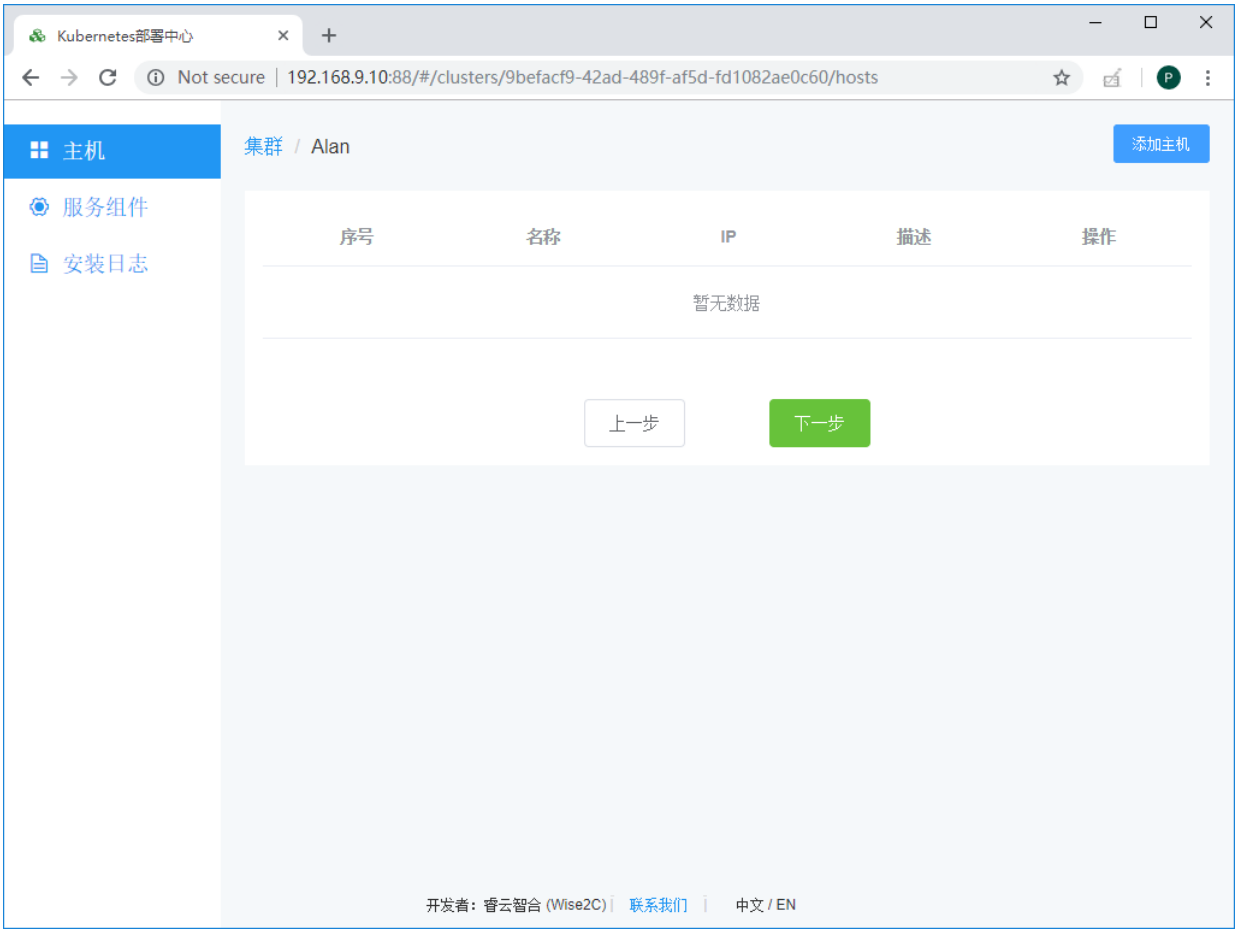



（1）点击开始按钮后，点击+图标开始添加一个集群：

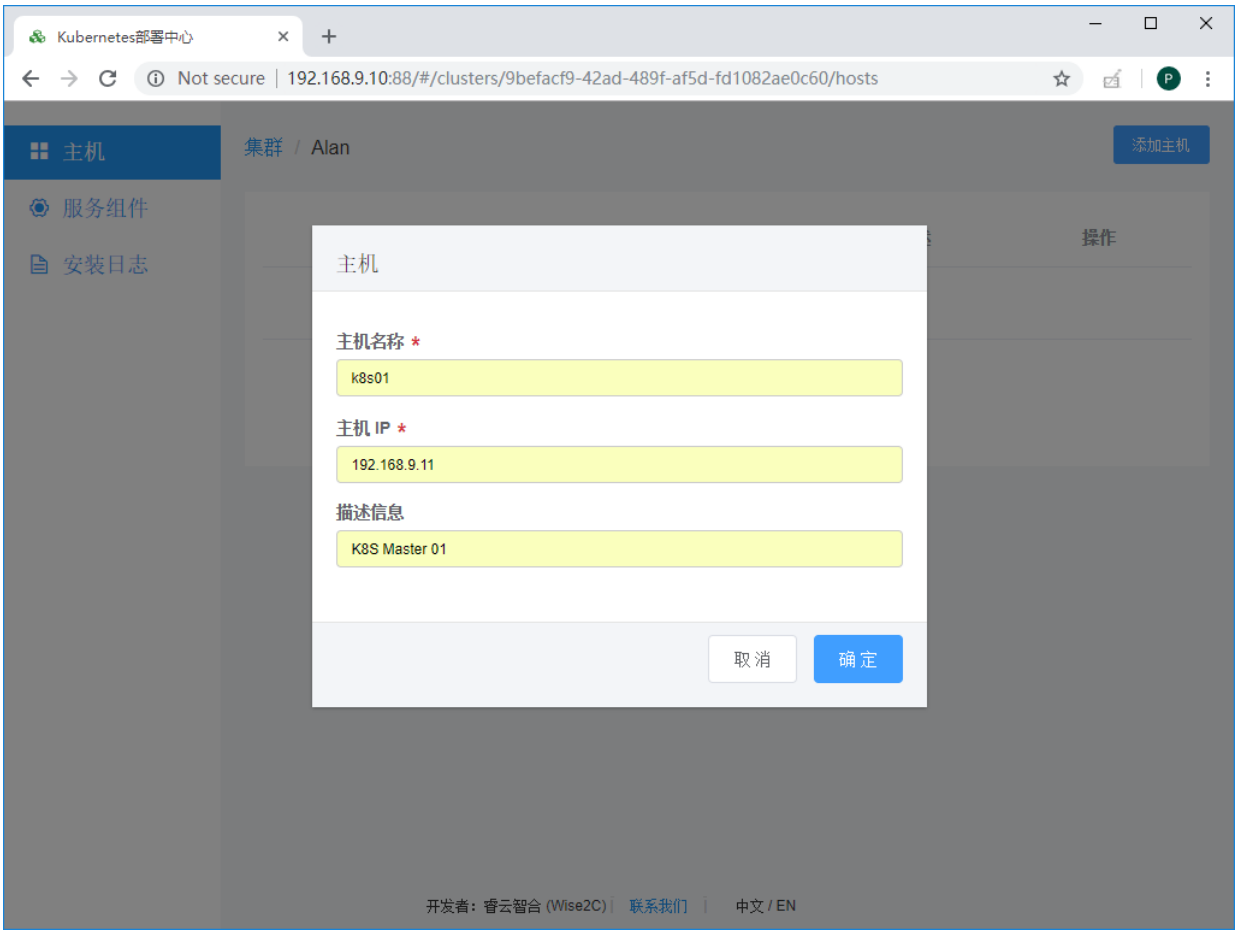




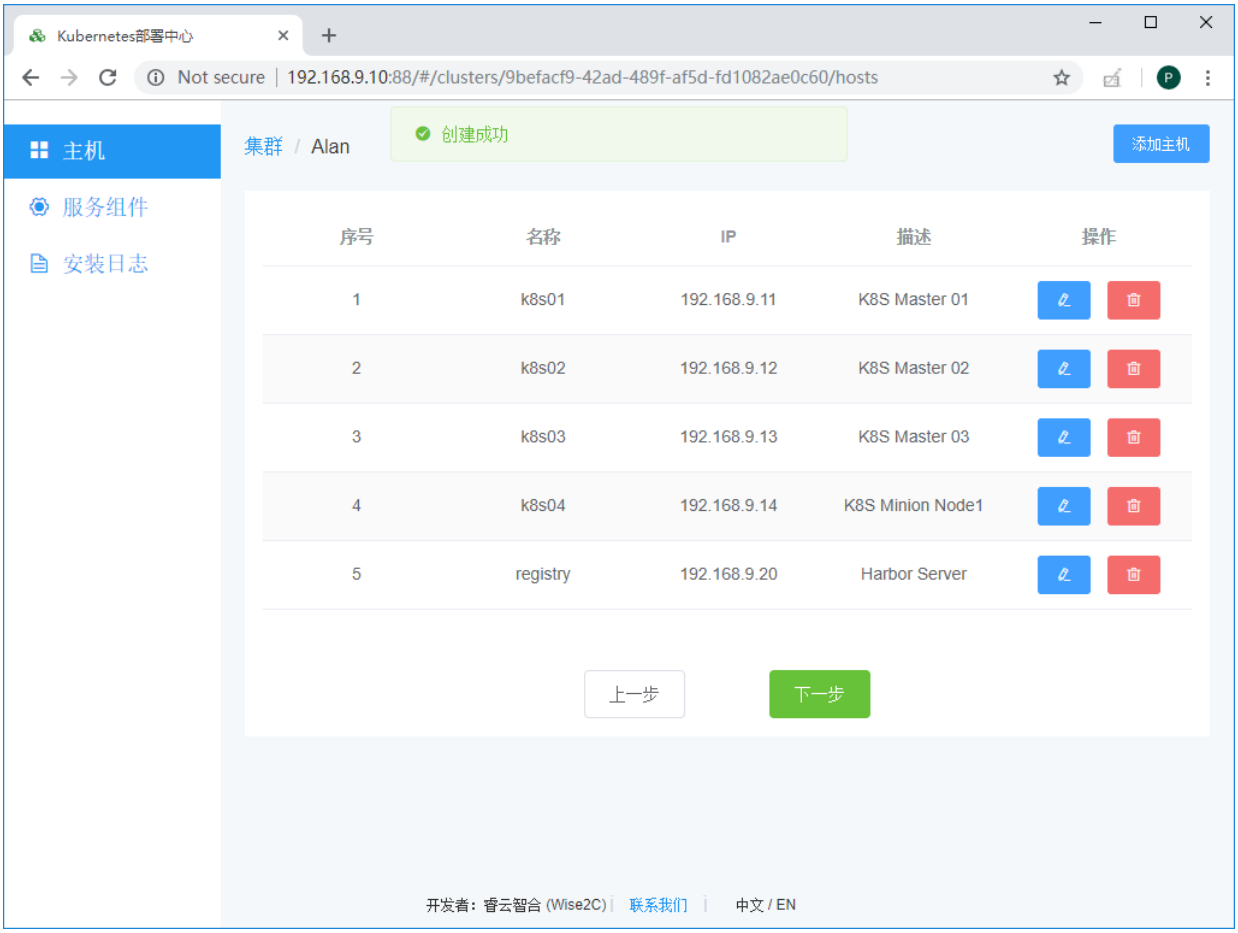

（2）点击该集群图标进入添加主机界面：




点击右上角“添加主机按钮”：

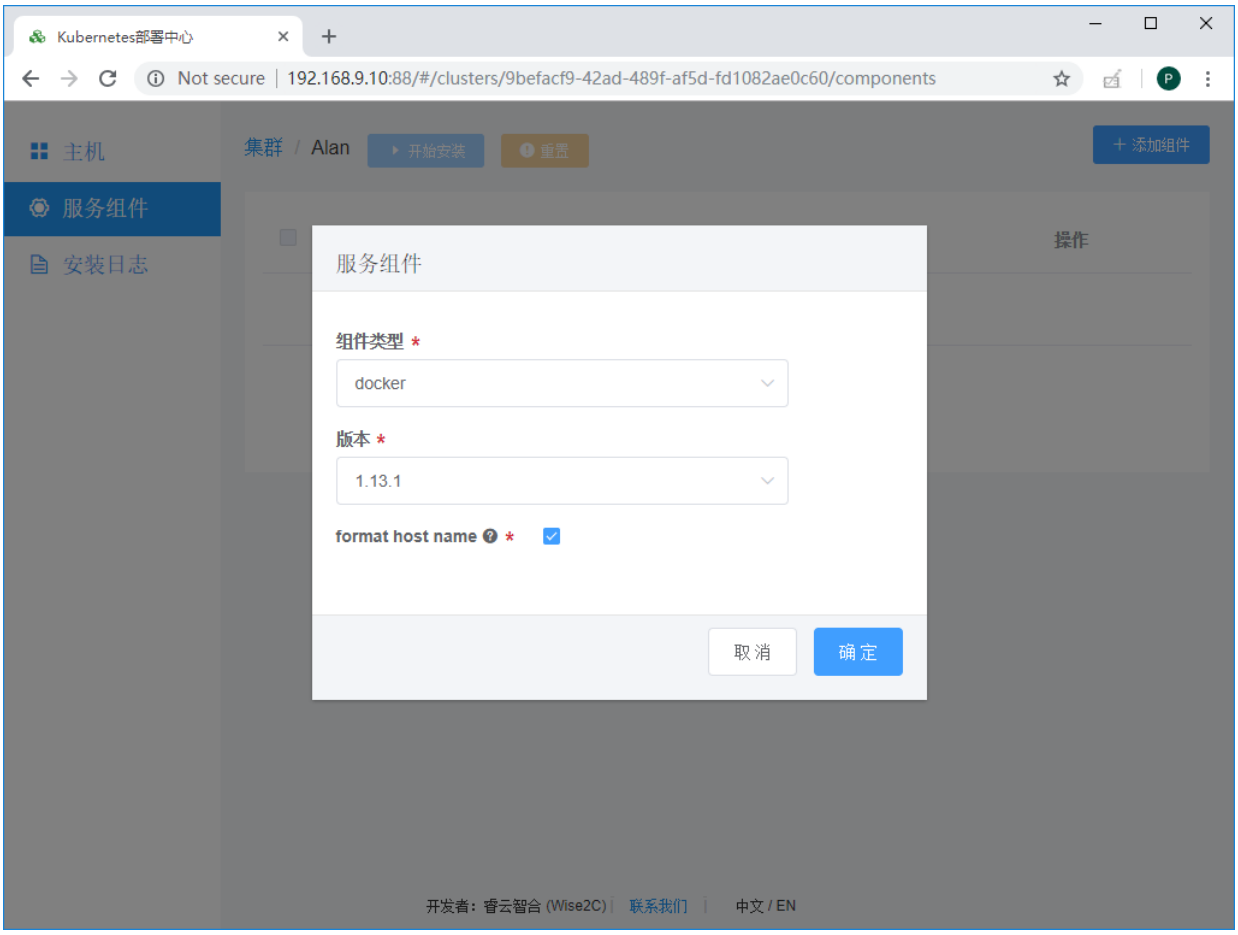



反复依次添加完整个集群的 5 台服务器：

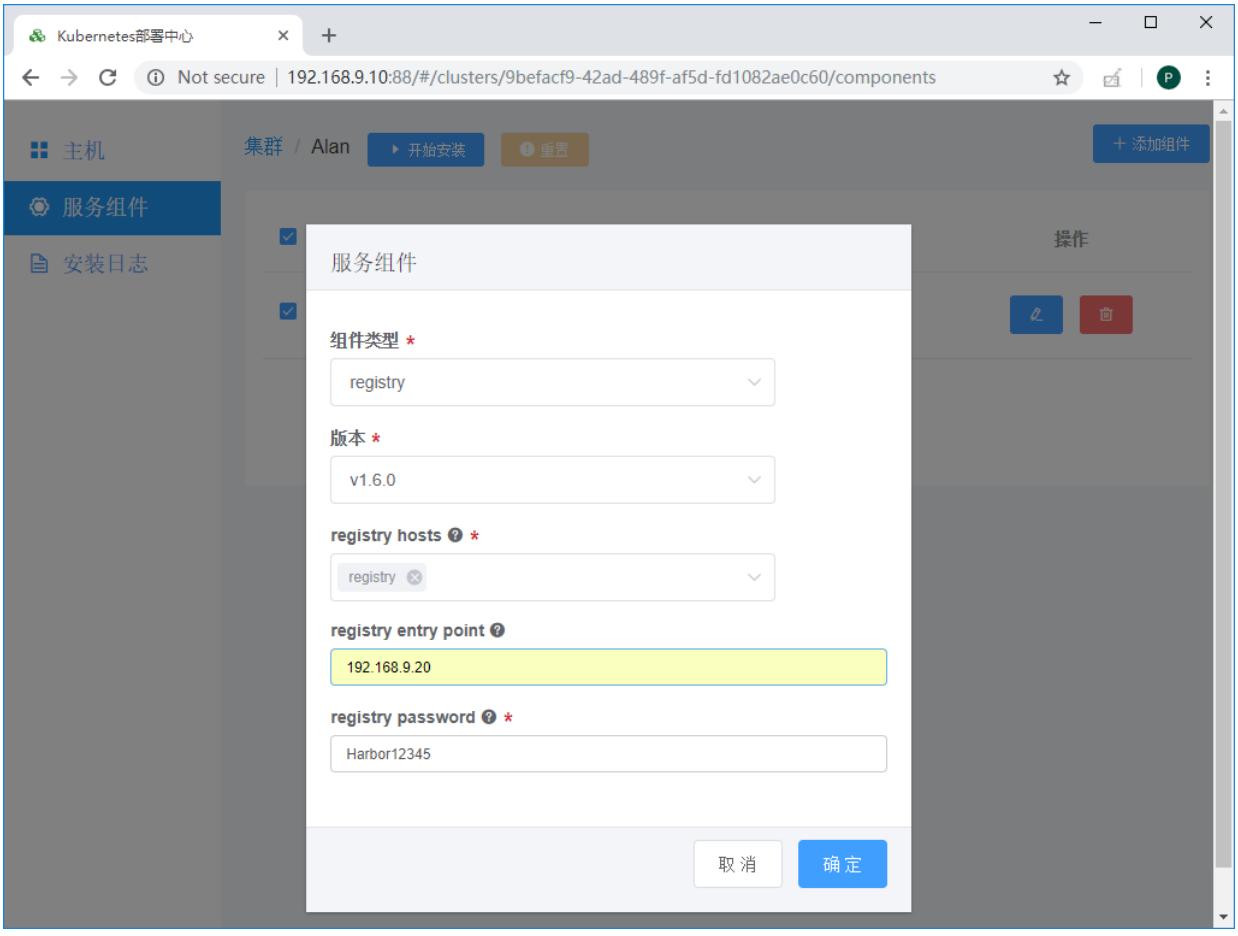



点击下一步进行服务组件定义

（3）点击右上角“添加组件”按钮添加服务组件，选择 docker，因为所有主机都需要安装，因此无需选择服务器：

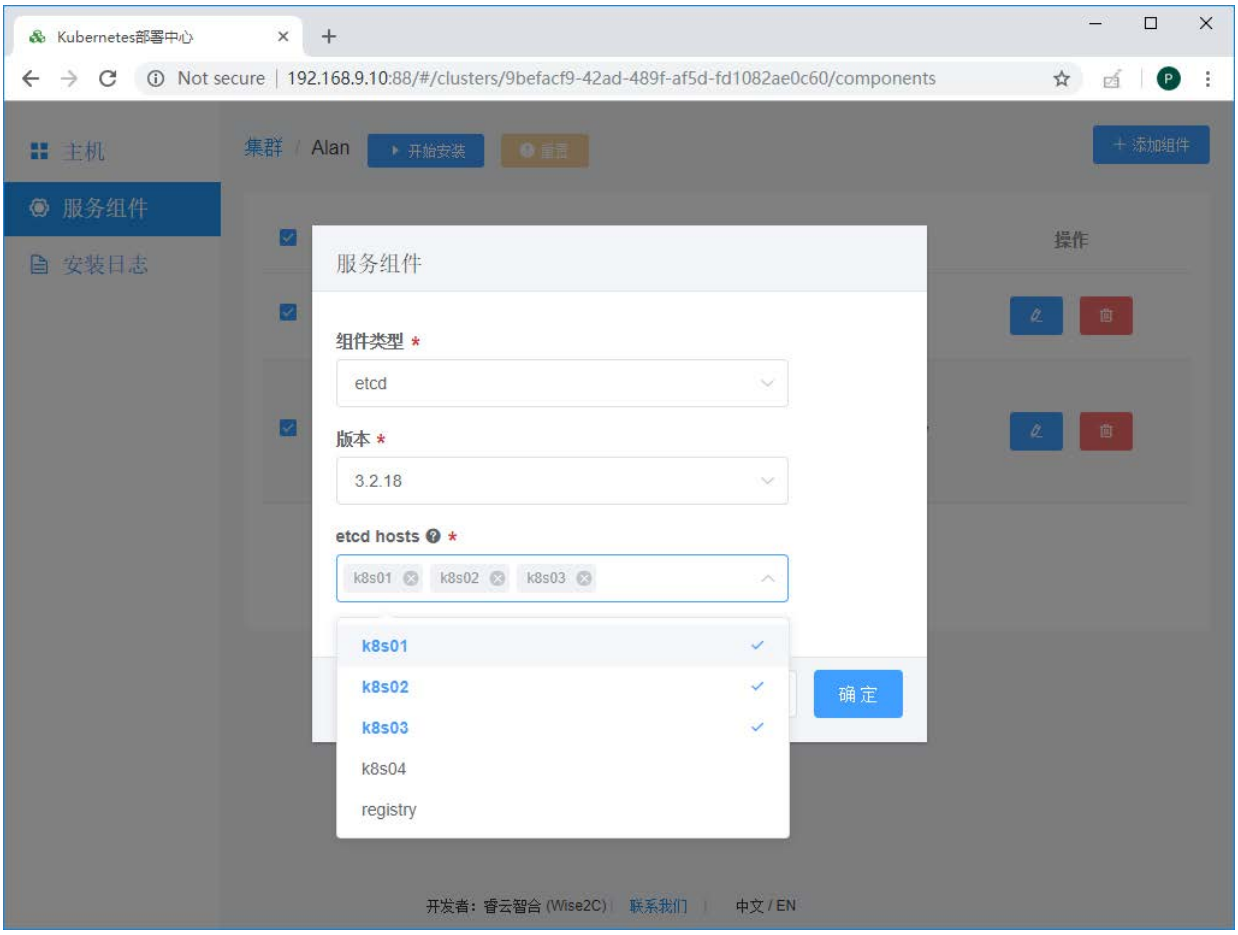



再添加镜像仓库组件

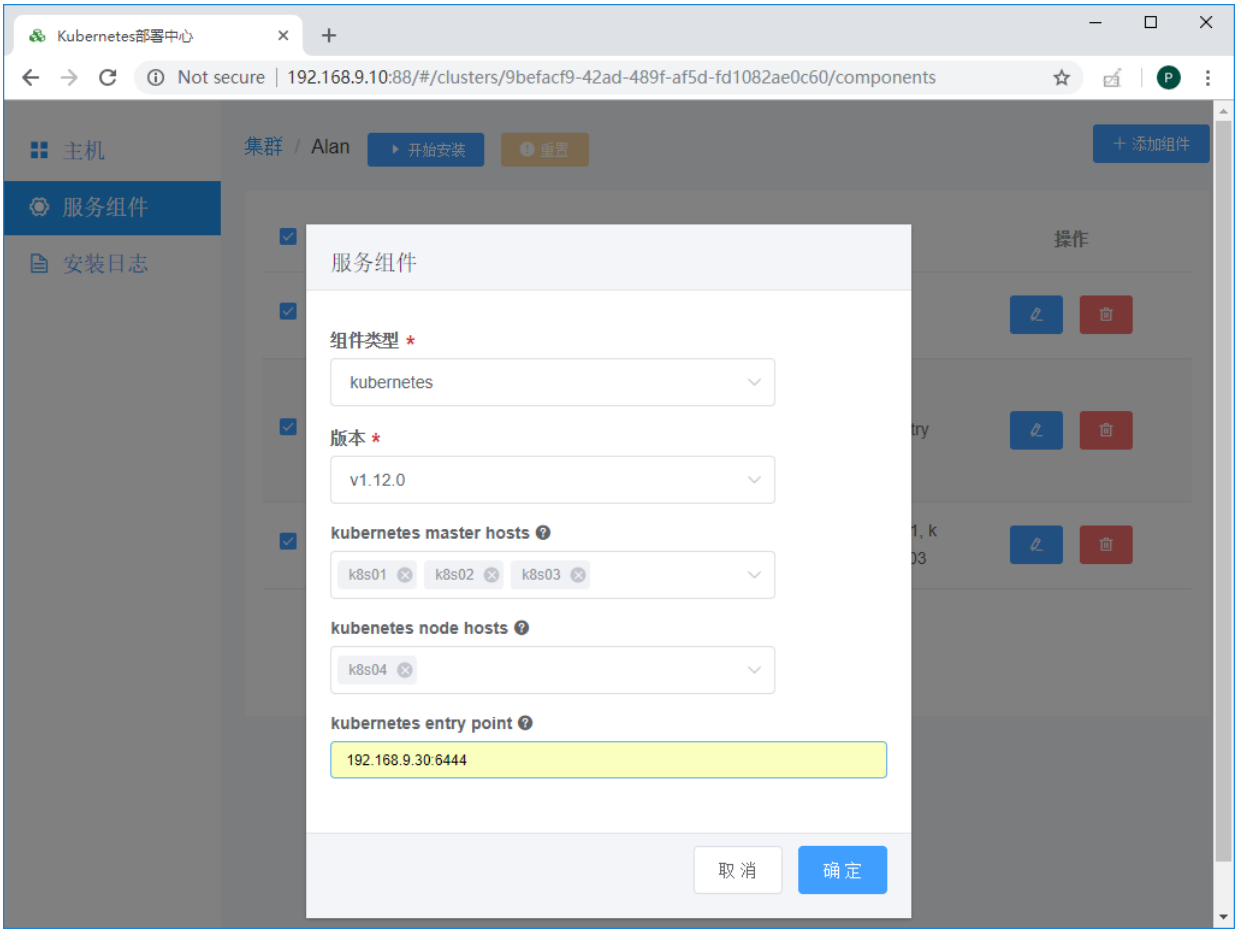



备注：registry entry point 默认就填写 Harbor 服务器的 IP 地址，有些环节可能使用域名则填写域名

继续添加 etcd 组件，这里我们将其合并部署于 k8s master 节点，也可以挑选单独的主机进行部署：



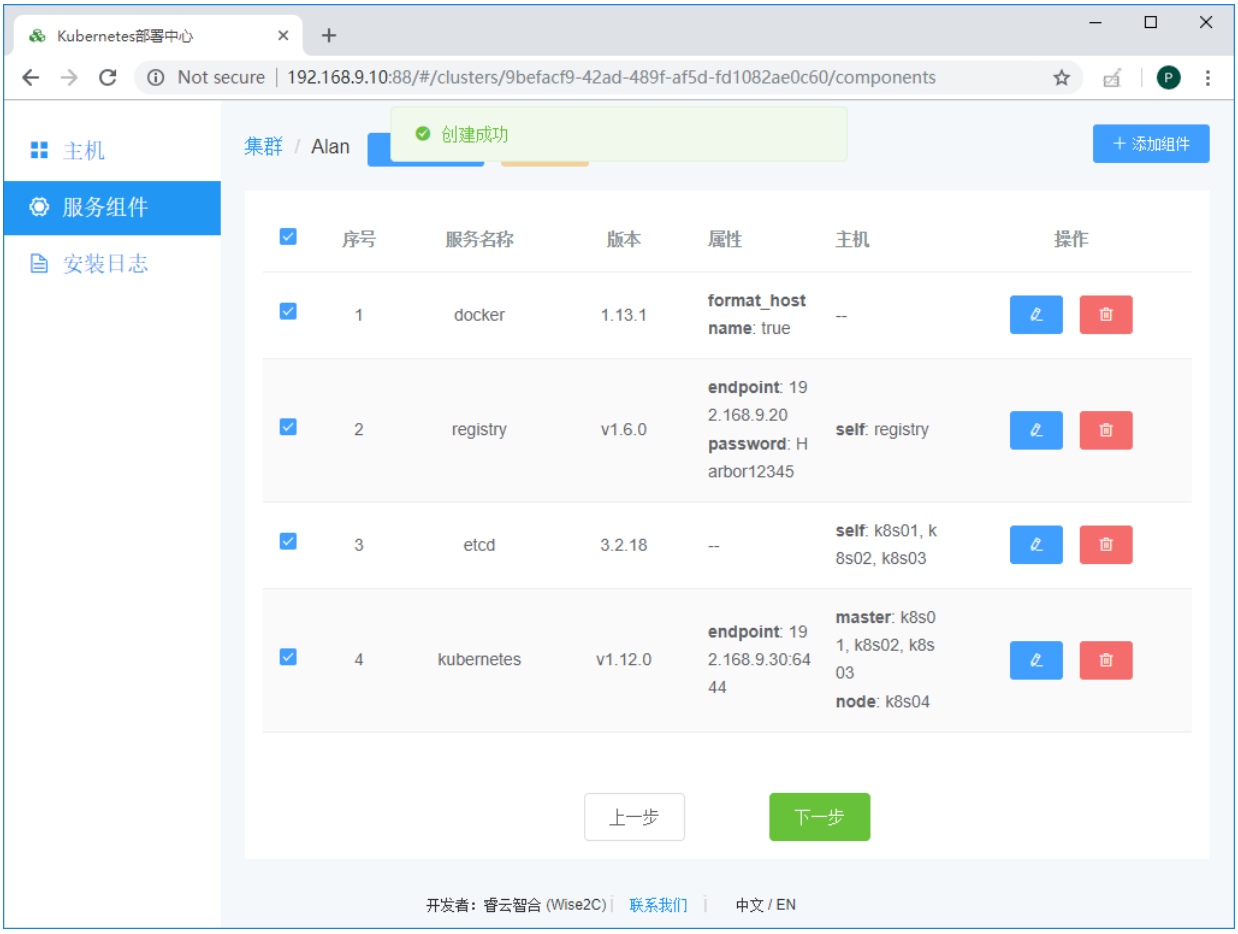

最后添加 k8s 组件，这里分为 master 和 minion nodes：




备注：这里 kubernetes entry point 是为了 HA 场景，比如此次试验我们在每一个 k8s master 节点同时各部署了

haproxy 和 keepalived 组件，其产生的虚 IP 是 192.168.9.30，端口是 6444，那么我们在这里应该填写为 192.168.9.30:6444，如果您只安装一个 master，那么可以填写为 master 的入口，例如 192.168.9.11:6443

设置完成的界面如下：




如果要实现高可用 HA 架构，请提前在部署机准备好以下资源包，详情请参阅：
https://github.com/wise2ck8s/haproxy-k8s
https://github.com/wise2ck8s/keepalived-k8s

（1）haproxy-k8s 镜像与启动脚本
（2）keepalived-k8s 镜像与启动脚本

在部署机上下载两个镜像：
**docker pull wise2c/haproxy-k8s**
**docker pull wise2c/keepalived-k8s**

保存镜像包：
**docker save wise2c/haproxy-k8s wise2c/keepalived-k8s -o /root/k8s-ha.tar**

拷贝镜像包至所有 master 节点：
**scp /root/k8s-ha.tar 192.168.9.11:/root/**
**scp /root/k8s-ha.tar 192.168.9.12:/root/**
**scp /root/k8s-ha.tar 192.168.9.13:/root/**

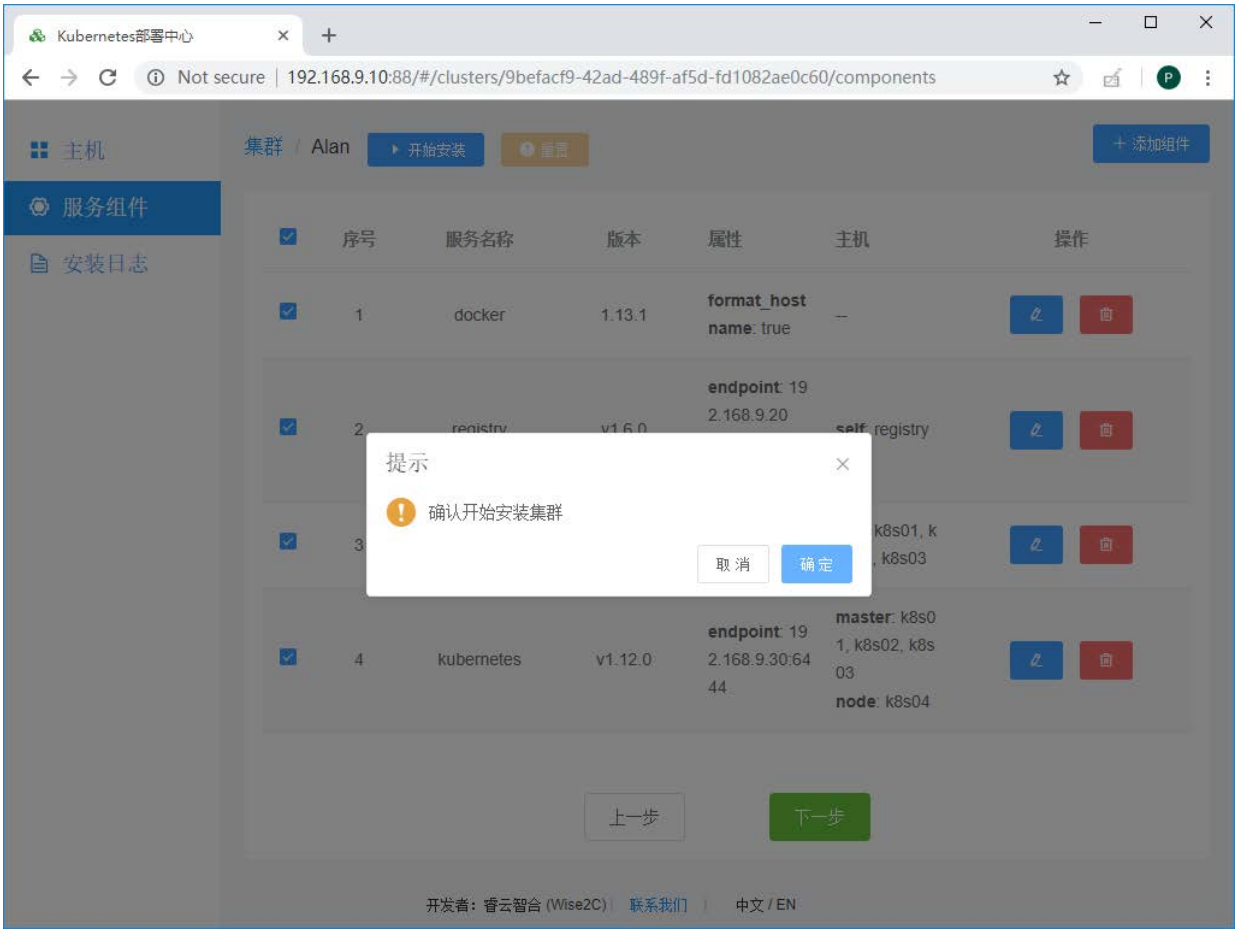
下载启动脚本
**curl -L /root/start-haproxy.sh https://raw.githubusercontent.com/wise2ck8s/haproxy-k8s/master/start-**

haproxy.sh
注意修改上述脚本中的 IP 地址与您实际场景一致：
MasterIP1=192.168.9.11
MasterIP2=192.168.9.12
MasterIP3=192.168.9.13

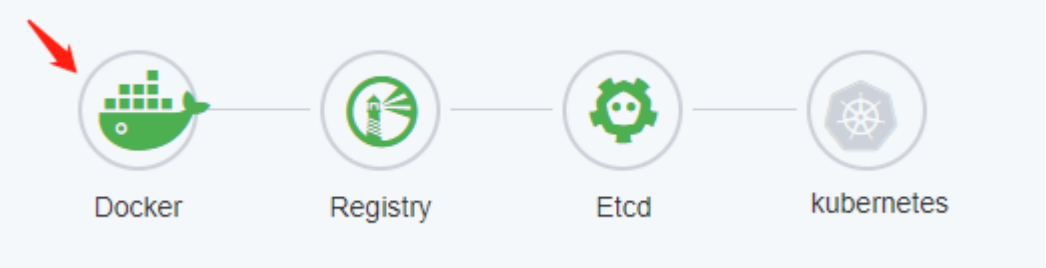
curl -L /root/start-keepalived.sh https://raw.githubusercontent.com/wise2ck8s/keepalived-k8s/master/start-keepalived.sh
注意修改上述脚本中的 VIP 地址和网卡名与您实际场景一致：
VIRTUAL_IP=192.168.9.30
INTERFACE=ens33

拷贝脚本至所有 master 节点：
chmod +x /root/start-haproxy.sh /root/start-keepalived.sh
scp –p /root/start-haproxy.sh 192.168.9.11:/root/
scp –p /root/start-haproxy.sh 192.168.9.12:/root/
scp –p /root/start-haproxy.sh 192.168.9.13:/root/
scp –p /root/start-keepalived.sh 192.168.9.11:/root/
scp –p /root/start-keepalived.sh 192.168.9.12:/root/
scp –p /root/start-keepalived.sh 192.168.9.13:/root/

四、点击下一步，执行部署流程：




在接下来的部署过程中，屏幕会有日志及图标颜色的动态变化：




当你看见 Docker 图标颜色变为绿色的时候，表示所有节点的 docker 已经能正常运行，此时可以不等后续部署过程结束，立刻去所有 k8s master 节点进行 HA 组件的启用：

**docker load -i /root/k8s-ha.tar**

**/root/start-haproxy.sh**

**/root/start-keepalived.sh**

然后耐心等待最后部署界面所有组件颜色变为绿色即可结束 K8S 高可用集群的部署工作。

验证：

```
[root@k8s01 ~]# kubectl get cs
NAME                 STATUS    MESSAGE             ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-2               Healthy   {"health": "true"}
etcd-1               Healthy   {"health": "true"}
etcd-0               Healthy   {"health": "true"}
[root@k8s01 ~]#
[root@k8s01 ~]# kubectl get csr
NAME                                             AGE   REQUESTOR                 CONDITION
csr-5wvkm                                        12m   system:node:k8s03         Approved,Issued
csr-7ntcr                                        13m   system:node:k8s02         Approved,Issued
csr-m8l57                                        13m   system:node:k8s01         Approved,Issued
node-csr-T4olwY7stBrPANRcuY-in-qiB_yoj6D7J6UMY4xqUWQ   12m   system:bootstrap:904250   Approved,Issued
[root@k8s01 ~]#
[root@k8s01 ~]# kubectl -n kube-system get pods -o wide
NAME                                    READY   STATUS    RESTARTS   AGE   IP              NODE    NOMINATED NODE
coredns-64fcf865cf-jbz4l                1/1     Running   0          13m   10.244.1.3      k8s02   <none>
coredns-64fcf865cf-mfqrt                1/1     Running   0          13m   10.244.1.2      k8s02   <none>
kube-apiserver-k8s01                    1/1     Running   0          12m   192.168.9.11    k8s01   <none>
kube-apiserver-k8s02                    1/1     Running   0          12m   192.168.9.12    k8s02   <none>
kube-apiserver-k8s03                    1/1     Running   0          12m   192.168.9.13    k8s03   <none>
kube-controller-manager-k8s01           1/1     Running   0          12m   192.168.9.11    k8s01   <none>
kube-controller-manager-k8s02           1/1     Running   0          12m   192.168.9.12    k8s02   <none>
kube-controller-manager-k8s03           1/1     Running   0          12m   192.168.9.13    k8s03   <none>
kube-flannel-ds-4cvnc                   1/1     Running   0          12m   192.168.9.13    k8s03   <none>
kube-flannel-ds-p9tnv                   1/1     Running   0          12m   192.168.9.11    k8s01   <none>
kube-flannel-ds-snfgp                   1/1     Running   0          12m   192.168.9.14    k8s04   <none>
kube-flannel-ds-zd5j6                   1/1     Running   0          12m   192.168.9.12    k8s02   <none>
kube-proxy-4gks8                        1/1     Running   0          12m   192.168.9.14    k8s04   <none>
kube-proxy-k8grt                        1/1     Running   0          13m   192.168.9.11    k8s01   <none>
kube-proxy-nm9n9                        1/1     Running   0          13m   192.168.9.12    k8s02   <none>
kube-proxy-qsr5g                        1/1     Running   0          13m   192.168.9.13    k8s03   <none>
kube-scheduler-k8s01                    1/1     Running   0          12m   192.168.9.11    k8s01   <none>
kube-scheduler-k8s02                    1/1     Running   0          12m   192.168.9.12    k8s02   <none>
kube-scheduler-k8s03                    1/1     Running   0          12m   192.168.9.13    k8s03   <none>
kubernetes-dashboard-559685d478-46rdh   1/1     Running   0          12m   10.244.0.2      k8s01   <none>
[root@k8s01 ~]#
[root@k8s01 ~]# kubectl get nodes -o wide
NAME    STATUS   ROLES    AGE   VERSION   INTERNAL-IP    EXTERNAL-IP   OS-IMAGE                KERNEL-VERSION          CONTAINER-RUNTIME
k8s01   Ready    master   13m   v1.12.1   192.168.9.11   <none>        CentOS Linux 7 (Core)   3.10.0-862.el7.x86_64   docker://1.13.1
k8s02   Ready    master   13m   v1.12.1   192.168.9.12   <none>        CentOS Linux 7 (Core)   3.10.0-862.el7.x86_64   docker://1.13.1
k8s03   Ready    master   13m   v1.12.1   192.168.9.13   <none>        CentOS Linux 7 (Core)   3.10.0-862.el7.x86_64   docker://1.13.1
k8s04   Ready    <none>   12m   v1.12.1   192.168.9.14   <none>        CentOS Linux 7 (Core)   3.10.0-862.el7.x86_64   docker://1.13.1
[root@k8s01 ~]#
```

Alan Peng

2018 年 10 月 6 日