

学习下 Kubernetes - 生产级别的容器编排系统

官网: <https://kubernetes.io/zh/>

Kubernetes 是用于自动部署, 扩展和管理容器化应用程序的开源系统。

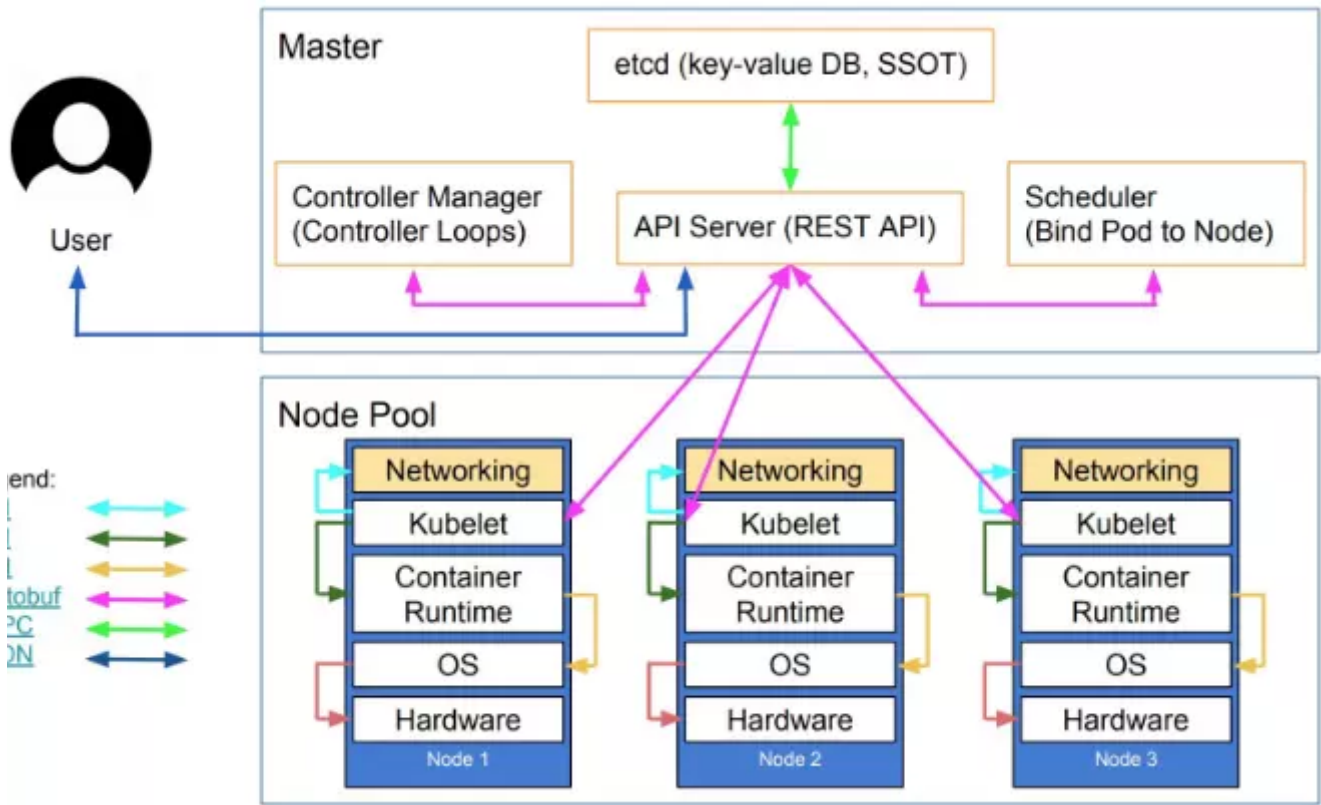
Kubernetes 特性:

- **自动包装。** 根据资源需求和其他约束自动放置容器, 同时不会牺牲可用性, 混合关键和最大努力的工作负载, 以提高资源利用率并节省更多资源。
- **横向缩放。** 使用简单的命令或 UI, 或者根据 CPU 的使用情况自动调整应用程序副本数。
- **自动部署和回滚。** Kubernetes 逐渐部署对应用程序或其配置的更改, 同时监视应用程序运行状况, 以确保它不会同时终止所有实例。如果出现问题, Kubernetes 会为您恢复更改, 利用日益增长的部署解决方案的生态系统。
- **存储编排。** 自动安装您所选择的存储系统, 无论是本地存储, 如公有云提供商 [GCP](#) 或 [AWS](#), 还是网络存储系统 NFS, iSCSI, Gluster, Ceph, Cinder, 或 Flocker。
- **自我修复。** 重新启动失败的容器, 在节点不可用时, 替换和重新编排节点上的容器, 终止不对用户定义的健康检查做出响应的容器, 并且不会在客户端准备投放之前将其通告给客户端。
- **服务发现和负载均衡。** 不需要修改您的应用程序来使用不熟悉的服务发现机制, Kubernetes 为容器提供了自己的 IP 地址和一组容器的单个 DNS 名称, 并可以在它们之间进行负载均衡。
- **密钥和配置管理。** 部署和更新密钥和应用程序配置, 不会重新编译您的镜像, 不会在堆栈配置中暴露密钥(secrets)。
- **批处理。** 除了服务之外, Kubernetes 还可以管理您的批处理和 CI 工作负载, 如果需要, 替换出现故障的容器。

Kubernetes 的核心概念

参考: [Kubernetes 组件](#)

一个典型的 Kubernetes 架构图如下：



一个典型的 Kubernetes 架构图

Master 组件

Master 组件提供的集群控制。 Master 组件对集群做出全局性决策(例如：调度)，以及检测和响应集群事件(副本控制器的replicas字段不满足时，启动新的副本)。

Master 组件可以在集群中的任何节点上运行。然而，为了简单起见，设置脚本通常会启动同一个虚拟机上所有 Master 组件，并且不会在此虚拟机上运行用户容器。

API 服务器

kube-apiserver 对外暴露了Kubernetes API。 它是的 Kubernetes 前端控制层。它被设计为水平扩展，即通过部署更多实例来缩放。

etcd

etcd 用于 Kubernetes 的后端存储。 所有集群数据都存储在此处，始终为您的 Kubernetes 集群的 etcd 数据提供备份计划。

kube-controller-manager

[kube-controller-manager](#) 运行控制器，它们是处理集群中常规任务的后台线程。逻辑上，每个控制器是一个单独的进程，但为了降低复杂性，它们都被编译成独立的可执行文件，并在单个进程中运行。

这些控制器包括:

- **节点控制器:** 当节点移除时，负责注意和响应。
- **副本控制器:** 负责维护系统中每个副本控制器对象正确数量的 Pod。
- **端点控制器:** 填充 端点(Endpoints) 对象(即连接 Services & Pods)。
- **服务帐户和令牌控制器:** 为新的命名空间创建默认帐户和 API 访问令牌。

云控制器管理器-(cloud-controller-manager)

cloud-controller-manager 是用于与底层云提供商交互的控制器。

以下控制器具有云提供商依赖关系:

- **节点控制器:** 用于检查云提供商以确定节点是否在云中停止响应后被删除。
- **路由控制器:** 用于在底层云基础架构中设置路由。
- **服务控制器:** 用于创建，更新和删除云提供商负载均衡器。
- **数据卷控制器:** 用于创建，附加和装载卷，并与云提供商进行交互以协调卷。

调度器 - (kube-scheduler)

[kube-scheduler](#) 监视没有分配节点的新创建的 Pod，选择一个节点供他们运行。

插件(addons)

插件是实现集群功能的 Pod 和 Service。

Pods 可以通过 Deployments, ReplicationControllers 管理。插件对象本身是受命名空间限制的，被创建于 `kube-system` 命名空间。

DNS

虽然其他插件并不是必需的，但所有 Kubernetes 集群都应该具有[Cluster DNS](#)，许多示例依赖于它。

Cluster DNS 是一个 DNS 服务器，和您部署环境中的其他 DNS 服务器一起工作，为 Kubernetes 服务提供 DNS 记录。

Kubernetes 启动的容器自动将 DNS 服务器包含在 DNS 搜索中。

用户界面

控制面板 Dashboard 提供了集群状态的只读概述。

容器资源监控

容器资源监控 将关于容器的一些常见的时间序列度量值保存到一个集中的数据库中，并提供用于浏览这些数据的界面。

集群层面日志

集群层面日志 机制负责将容器的日志数据保存到一个集中的日志存储中，该存储能够提供搜索和浏览接口。

节点组件

节点组件在每个节点上运行，维护运行的 Pod 并提供 Kubernetes 运行时环境。

kubelet

kubelet 是主要的节点代理，它监测已分配给其节点的 Pod(通过 apiserver 或通过本地配置文件)，提供如下功能：

- 挂载 Pod 所需要的数据卷(Volume)。
- 下载 Pod 的 secrets。
- 通过 Docker 运行(或通过 rkt)运行 Pod 的容器。
- 周期性的对容器生命周期进行探测。
- 如果需要，通过创建 镜像 Pod (Mirror Pod) 将 Pod 的状态报告回系统的其余部分。
- 将节点的状态报告回系统的其余部分。

kube-proxy

kube-proxy 通过维护主机上的网络规则并执行连接转发，实现了Kubernetes服务抽象。

docker

Docker 用于运行容器。

rkt

支持 rkt 运行容器作为 Docker 的试验性替代方案。

supervisord

supervisord 是一个轻量级的进程监控系统，可以用来保证 kubelet 和 docker 运行。

fluentd

fluentd 是一个守护进程，它有助于提供[集群层面日志](#) 集群层面的日志。

在 AWS 上使用 Kubernetes 方式1：使用 Amazon EKS

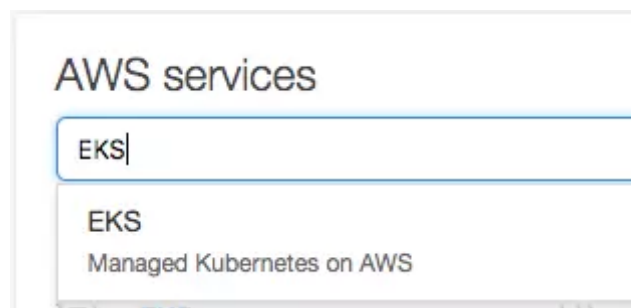
参见：[什么是 Amazon EKS?](#)

Amazon Elastic Container Service for Kubernetes (Amazon EKS) 是一项托管服务，让您在 AWS 上轻松运行 Kubernetes，**而无需支持或维护您自己的 Kubernetes 控制层面。**

Amazon EKS 运行最新版本的开源 Kubernetes 软件，因此您可以使用 Kubernetes 社区的所有现有插件和工具。在 Amazon EKS 上运行的应用程序与在任何标准 Kubernetes 环境中运行的应用程序完全兼容，无论此类环境是在本地数据中心还是在公有云中运行都是如此。这意味着，您可以轻松地将任何标准 Kubernetes 应用程序迁移到 Amazon EKS，而无需进行任何代码修改。

Amazon EKS 入门是很轻松的：

- 首先，在 AWS 管理控制台中或使用 AWS CLI 或 AWS 开发工具包之一创建一个 Amazon EKS 集群。



- 然后，启动向此 Amazon EKS 集群注册的工作线程节点。我们为您提供了一个可自动配置您的节点的 AWS CloudFormation 模板。
- 在集群准备就绪时，可以将常用 Kubernetes 工具（如 **kubectl**）配置为与集群通信。
- 像在任何其他 Kubernetes 环境中一样在 Amazon EKS 集群上部署和管理应用程序。

在 AWS 上使用 Kubernetes 方式2：手动来操作

登陆到一台 EC2 主机：`ssh -i "XiangSecret.pem" ec2-user@52.14.52.46`

关于 AWS EC2 的创建，参见：[AWS 第一个 NodeJS 应用程序](#)

准备工作

确保已成功安装 awscli

可以通过 `aws` 来确认：

```
[ec2-user@ip-172-31-38-187 ~]$ aws
usage: aws [options] <command> <subcommand> [<subcommand> ...] [parameters]
To see help text, you can run:

aws help
aws <command> help
aws <command> <subcommand> help
aws: error: too few arguments
[ec2-user@ip-172-31-38-187 ~]$
```

awscli

安装 kubectl

什么是 `kubectl`？`kubectl` 是 Kubernetes 命令行工具，它通过 Kubernetes API 来管理部署。

参见：[安装 kubectl](#)

```
curl -o kubectl https://amazon-eks.s3-us-west-2.amazonaws.com/1.10.3/2018-07-26/bin/linux/amd64/
chmod +x ./kubectl
mkdir $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```

可以通过 `kubectl version --short --client` 查看 `kubectrl` 版本：

```
[ec2-user@ip-172-31-38-187 ~]$ kubectl version --short --client
Client Version: v1.10.3
```

查看 kubectr 版本

PS：如果碰到权限问题，请使用 `sudo`

安装 kops

什么是 `kops`？<https://github.com/kubernetes/kops>

kops 是官方推荐的工具，用来在 AWS 生产环境中，快速地部署 Kubernetes 集群。

We like to think of it as `kubect1` for clusters.

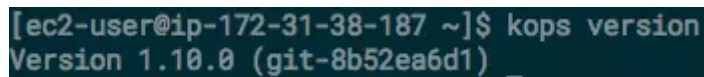
`kops` helps you create, destroy, upgrade and maintain production-grade, highly available, Kubernetes clusters from the command line. AWS (Amazon Web Services) is currently officially supported.

参见：

- [Installing Kubernetes on AWS with kops](#)
- [使用 kops 在 AWS 部署 Kubernetes 集群](#)

```
wget https://github.com/kubernetes/kops/releases/download/1.10.0/kops-linux-amd64
chmod +x kops-linux-amd64
mv kops-linux-amd64 /usr/local/bin/kops
```

可以通过 `kops version` 查看 `kops` 版本：



```
[ec2-user@ip-172-31-38-187 ~]$ kops version
Version 1.10.0 (git-8b52ea6d1)
```

查看 kops 版本

创建 IAM 用户

为了使用 `kops` 部署集群，还需要为 `kops` 创建一个 IAM 用户 `kops`，并分配相应的权限。包括：

- AmazonEC2FullAccess
- AmazonRoute53FullAccess
- AmazonS3FullAccess
- IAMFullAccess
- AmazonVPCFullAccess


```
aws iam create-group --group-name kops
aws iam attach-group-policy --policy-arn arn:aws:iam::aws:policy/AmazonEC2FullAccess --group-name kops
aws iam attach-group-policy --policy-arn arn:aws:iam::aws:policy/AmazonRoute53FullAccess --group-name kops
aws iam attach-group-policy --policy-arn arn:aws:iam::aws:policy/AmazonS3FullAccess --group-name kops
aws iam attach-group-policy --policy-arn arn:aws:iam::aws:policy/IAMFullAccess --group-name kops
aws iam attach-group-policy --policy-arn arn:aws:iam::aws:policy/AmazonVPCFullAccess --group-name kops
aws iam create-user --user-name kops
aws iam add-user-to-group --user-name kops --group-name kops
```

为 **kops** 用户创建密钥:

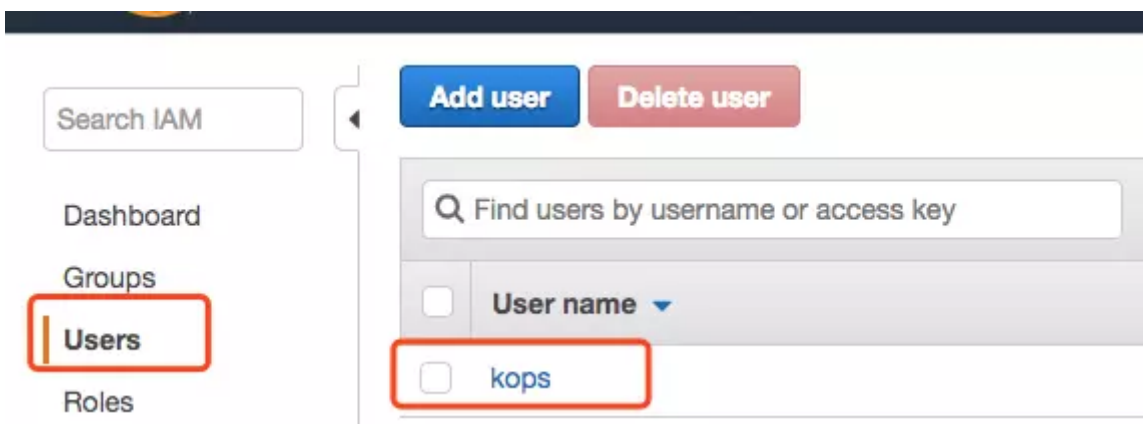
```
aws iam create-access-key --user-name kops
```

上面的命令会返回 **kops** 用户的 **AccessKeyId** 和 **SecretAccessKey**。

```
[ec2-user@ip-172-31-38-187 ~]$ aws iam create-access-key --user-name kops
{
  "AccessKey": {
    "UserName": "kops",
    "Status": "Active",
    "CreateDate": "2018-11-13T08:39:57Z",
    "SecretAccessKey": "AKIAIHOVBNH5T4EBN",
    "AccessKeyId": "AKIAIHOVBNH5T4EBN"
  }
}
```

image.png

创建后，我们可以在 AWS 控制台的 UI 上看到用户：**kops**：



在 AWS 控制台的 UI 上看到用户：kops

接着通过 **aws configure** 更新 **awscli** 的配置，让它使用新创建的 **kops** 用户的密钥：


```
ec2-user@ip-172-31-38-187 ~]$ aws configure
Access Key ID [*****DIKQ]: 
Secret Access Key [*****al1V]: 
Default region name [us-east-2]: 
Default output format [None]:
```

更新 awscli 的配置

同时还需要将 **kops** 用户的密钥导出到命令行的环境变量：其实是写入 `~/.bashrc` 文件

```
echo 'export AWS_ACCESS_KEY_ID=...' >> ~/.bashrc
echo 'export AWS_SECRET_ACCESS_KEY=...' >> ~/.bashrc
echo 'export AWS_REGION=us-east-2' >> ~/.bashrc
```

最后是生成 SSH 密钥：生成的密钥位置 `/home/ec2-user/.ssh/id_rsa.pub`

```
ssh-keygen
```

配置 S3 Bucket

需要创建一个 S3 bucket，**用于存储集群的数据**，例如，这里我们将这个 bucket 命名为 `xiang.k8s.local-state`，同时写到环境变量 `KOPS_STATE_STORE`。

(PS：这里用 `xiang` 是因为我的名字里有 `xiang`，不代表你也一定要这么用)

```
aws s3api create-bucket --bucket xiang.k8s.local-state --create-bucket-configuration LocationCon
echo 'export KOPS_STATE_STORE=s3://xiang.k8s.local-state' >> ~/.bashrc
```

创建后，我们可以在 AWS 控制台的 UI 上看到 S3 bucket： `xiang.k8s.local-state`：

在 AWS 控制台的 UI 上看到S3 bucket: xiang.k8s.local-state

创建集群

在上面的准备工作完成后，我们可以开始创建 Kubernetes 集群。

创建集群的配置文件

下面的命令会创建集群的配置文件，**并不会真正地创建集群**：

```
kops create cluster \  
  --name=xiang.k8s.local \  
  --zones=us-east-2b \  
  --master-size="t2.micro" \  
  --node-size="t2.micro" \  
  --ssh-public-key=~/.ssh/id_rsa.pub"
```

注意：

- `xiang.k8s.local` 是什么呢？为了让 `kops` 创建基于 gossip 的集群，集群的命名需要使用 `.k8s.local` 作为后缀，例如，这里我们将集群命名为 `xiang.k8s.local`
- `t2.micro` 是什么呢？它是一种 EC2 实例的类型
 - 具体的类型可以参见：<https://aws.amazon.com/cn/ec2/pricing/on-demand/>
 - 我现在拥有的两个 EC2 实例就是 `t2.micro` 类型的：

Instance Type	Availability Zone	Instance State
t2.micro	us-east-2c	running
t2.micro	us-east-2b	running

我现在拥有的两个 EC2 实例就是 t2.micro 类型的

- `~/.ssh/id_rsa.pub` 其实就是上面我们通过 `ssh-keygen` 来创建的

在创建集群之前，可以检查集群的配置文件是否正确：

```
kops edit cluster xiang.k8s.local
```

创建集群

如果确认没问题，就可以使用下面的命令创建集群：

```
kops update cluster xiang.k8s.local --yes
```

创建集群之后，需要一段时间等待集群的初始化，我们可以通过 `kops validate cluster` 来检查集群的创建状态：

```
[ec2-user@ip-172-31-38-187 ~]$ kops validate cluster
Using cluster from kubectl context: xiang.k8s.local




Validating cluster xiang.k8s.local

INSTANCE GROUPS
NAME                ROLE      MACHINETYPE  MIN  MAX  SUBNETS
master-us-east-2b   Master    t2.micro      1    1    us-east-2b
nodes                Node      t2.micro      2    2    us-east-2b

NODE STATUS
NAME                                                         ROLE      READY
ip-172-20-44-223.us-east-2.compute.internal node      True
ip-172-20-54-238.us-east-2.compute.internal master    True
ip-172-20-59-124.us-east-2.compute.internal node      True

Your cluster xiang.k8s.local is ready
```

等待集群起来之后，可以在 AWS 控制台的 UI 上看到新创建的 EC2 实例：

Name	Instance ID	Instance Type	Availability Zone	Instance State
master-us-east-2b.masters.xiang.k8s.local	i-06bd5b2a22b75ff80	t2.micro	us-east-2b	 running
nodes.xiang.k8s.local	i-03fd68d3cb0dee74e	t2.micro	us-east-2b	 running
nodes.xiang.k8s.local	i-09d6a3335c4e26ae6	t2.micro	us-east-2b	 running

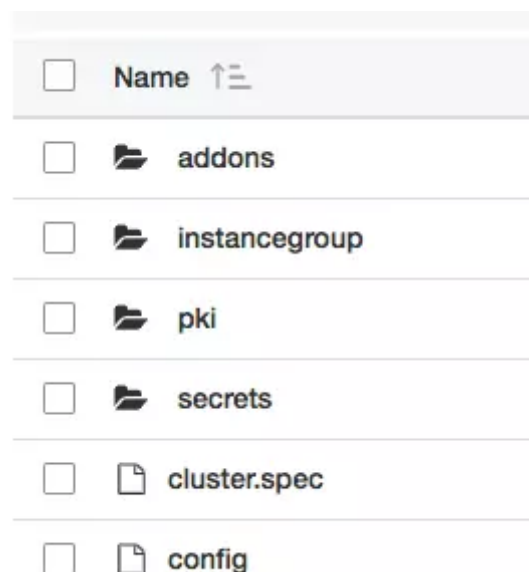
可以在 AWS 控制台的 UI 上看到产生的集群，包括一个 master，两个 node

每一个实例对应一个 Kubernetes 中的节点 Node，其中一个的角色为 master，另外两个的角色为 node 具体信息如下：

- **EC2 实例名称：** `master-us-east-2b.masters.xiang.k8s.local`

- 公有 IP: 3.16.188.170
 - 公有 DNS: ec2-3-16-188-170.us-east-2.compute.amazonaws.com
 - 私有 IP: 172.20.54.238
 - **对应的 Kubernetes 节点名称:** ip-172-20-54-238.us-east-2.compute.internal (其实这就是私有 DNS)
-
- **EC2 实例名称:** nodes.xiang.k8s.local
-
- 公有 IP: 3.16.181.109
 - 公有 DNS: ec2-3-16-181-109.us-east-2.compute.amazonaws.com
 - 私有 IP: 172.20.44.223
 - **对应的 Kubernetes 节点名称:** ip-172-20-44-223.us-east-2.compute.internal (其实这就是私有 DNS)
-
- **EC2 实例名称:** nodes.xiang.k8s.local
-
- 公有 IP: 18.223.182.144
 - 公有 DNS: ec2-18-223-182-144.us-east-2.compute.amazonaws.com
 - 私有 IP: 172.20.59.124
 - **对应的 Kubernetes 节点名称:** ip-172-20-59-124.us-east-2.compute.internal (其实这就是私有 DNS)

我们也可以在 AWS 控制台的 UI 上看到 `xiang.k8s.local-state` 这个 S3 Bucket 包含对应的文件, 用于存储集群的数据:



xiang.k8s.local-state 这个 S3 Bucket 包含对应的文件

操作集群

在上述步骤结束之后，`kops` 会配置好 Kubernetes 的命令行工具 `kubectl`，配置的内容包括 Kubernetes 集群的 API endpoint 端口和身份认证信息。此后，我们就可以使用 `kubectl` 来操作 Kubernetes 集群了。

使用 `kubectl get nodes` 来查看节点信息

```
[ec2-user@ip-172-31-38-187 ~]$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
ip-172-20-44-223.us-east-2.compute.internal	Ready	node	2h	v1.10.6
ip-172-20-54-238.us-east-2.compute.internal	Ready	master	2h	v1.10.6
ip-172-20-59-124.us-east-2.compute.internal	Ready	node	2h	v1.10.6

使用 `kubectl cluster-info` 来查看集群信息

```
[ec2-user@ip-172-31-38-187 ~]$ kubectl cluster-info
Kubernetes master is running at https://api-xiang-k8s-local-ips5k0-118730418.us-east-2.elb.amazo
KubeDNS is running at https://api-xiang-k8s-local-ips5k0-118730418.us-east-2.elb.amazonaws.com/a

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

`api-xiang-k8s-local-ips5k0-118730418.us-east-2.elb.amazonaws.com` 这又是个什么呢？好像刚刚创建

的三个 EC2 的实例都不叫这个域名。

我们打开 AWS 控制台，可以看到自动创建了一个 **负载均衡器** `api-xiang-k8s-local-ips5k0`，它提供了这个域名：



获取 kube 和 admin 这两个账户的密码

我们可以通过下面的命令获取 `kube` 和 `admin` 这两个账户的密码:

```
kops get secrets kube --type secret -oplaintext
kops get secrets admin --type secret -oplaintext
```

```
ec2-user@ip-172-31-38-187 ~]$ kops get secrets kube --type secret -oplaintext
sing cluster from kubect1 context: xiang.k8s.local
5t1[REDACTED]
ec2-user@ip-172-31-38-187 ~]$ kops get secrets admin --type secret -oplaintext
sing cluster from kubect1 context: xiang.k8s.local
xYB[REDACTED]rRgt4iooCpgc
ec2-user@ip-172-31-38-187 ~]$
```

在设置之后, 可以通过 `kubect1 config view` 看到用户名和密码:

```
c2-user@ip-172-31-38-187 ~]$ kubectl config view
```

kubectl config view 看到用户名和密码

登陆到 Master 和 Node 看看？

我们已经知道了 Master 和 Node 的 DNS 和 IP 地址。

我们通过如下命令登陆到 Master：

```
ssh admin@3.16.188.170 (公有 IP 或者 公有 DNS)
```

我们可以看到 Master 上已经安装好了 Docker 和 `kubectl`：

```
admin@ip-172-20-54-238:~$ kubectl version --short --client
Client Version: v1.10.6
admin@ip-172-20-54-238:~$ docker --version
Docker version 17.03.2-ce, build f5ec1e2
```

我们通过如下命令分别登陆到两个 Node，同样，Node 上已经安装好了 Docker 和 `kubectl`。

```
ssh admin@3.16.181.109 (公有 IP 或者 公有 DNS)
ssh admin@18.223.182.144 (公有 IP 或者 公有 DNS)
```

部署 Kubernetes 的控制面板 Dashboard

通过下面的命令到部署 Kubernetes 的控制面板 Dashboard：

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recomm
```

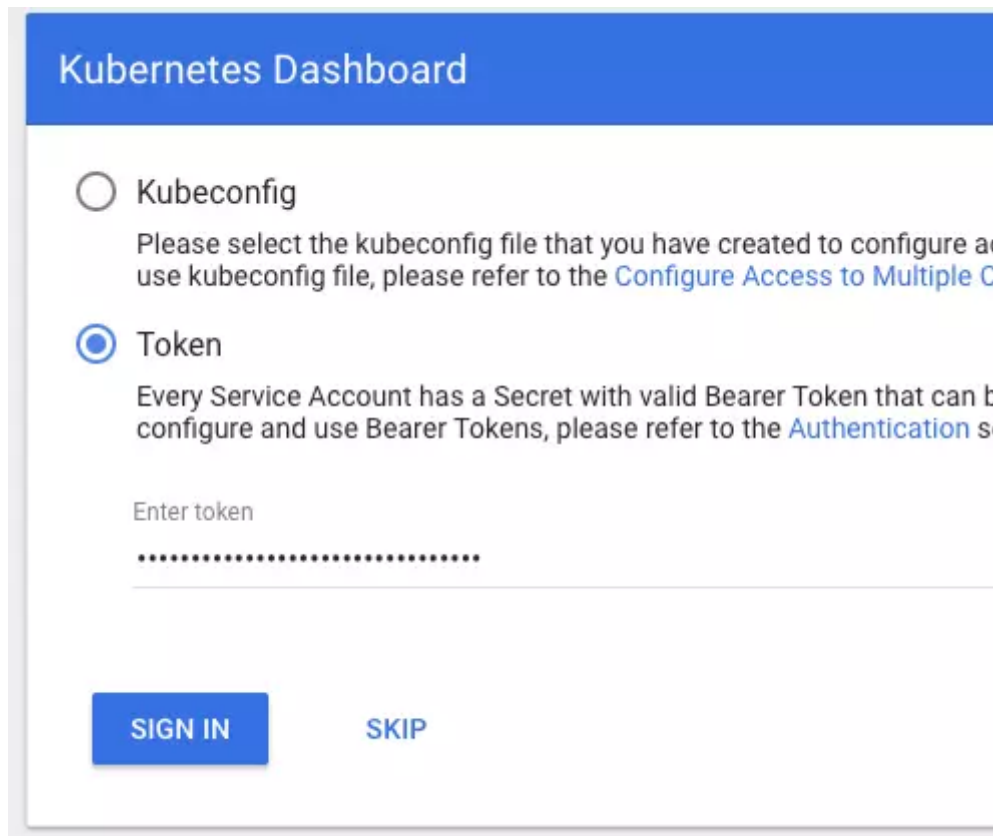
部署成功后，我们可以通过如下的网址访问 Dashboard：

```
https://<master-ip>:<apiserver-port>/api/v1/namespaces/kube-system/services/https:kubernetes-das
```


在这里 `<master-ip>` 为 `api-xiang-k8s-local-ips5k0-118730418.us-east-2.elb.amazonaws.com` (通过 `kubectl cluster-info` 命令获得), `<apiserver-port>` 为 `443`, 通过如下的网址访问 Dashboard:

<https://api-xiang-k8s-local-ips5k0-118730418.us-east-2.elb.amazonaws.com:443/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/>

如果弹出一个登陆框, 用户名输入 `kube`, 密码为上述步骤中为 `kube` 用户设置的密码。接下来可能需要第二次 Login, 选择 Token, 密码为上述步骤中为 `admin` 用户设置的密码:

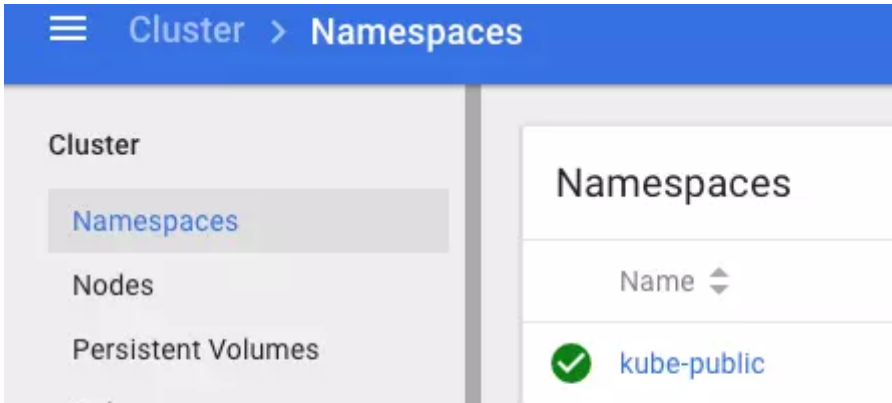
The image shows the Kubernetes Dashboard login interface. At the top is a blue header with the text "Kubernetes Dashboard". Below the header, there are two radio button options. The first option is "Kubeconfig" with an unselected radio button. Below it, the text says "Please select the kubeconfig file that you have created to configure access to the cluster. For more information, see the documentation on kubeconfig files, please refer to the Configure Access to Multiple Clusters page." The second option is "Token" with a selected radio button. Below it, the text says "Every Service Account has a Secret with valid Bearer Token that can be used to access the API server. For more information, see the Authentication and Authorization page." Below the "Token" option, there is a label "Enter token" followed by a text input field containing a series of dots. At the bottom of the form, there are two buttons: "SIGN IN" in a blue box and "SKIP" in a light blue box.

Login

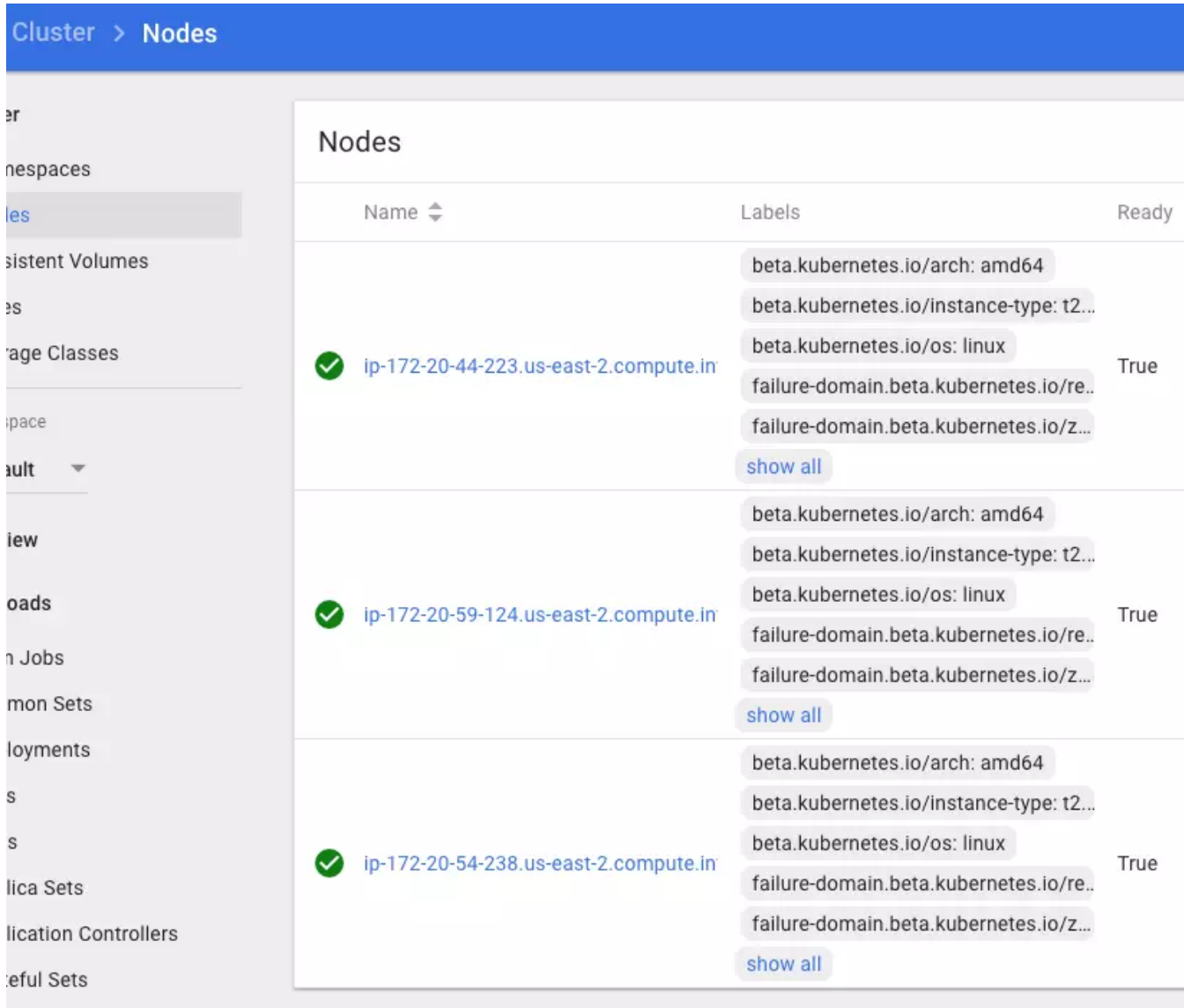
操作 Kubernetes 的控制面板 Dashboard

具体玩法, 参见: [Kubernetes Web UI \(Dashboard\)](#)

可以看到目前有三个 Namespace 和 三个 Node:



三个 Namespace



三个 Node

销毁集群

在销毁集群之前，需要先确认一下 `kops` 会删除哪些资源：

```
kops delete cluster --name xiang.k8s.local
```

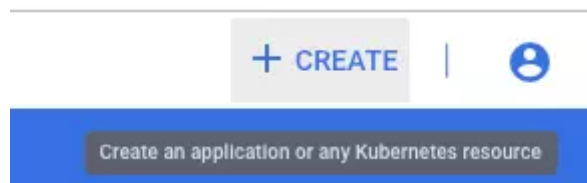
如果确认没问题，就可以真正删除集群：

```
kops delete cluster --name xiang.k8s.local --yes
```

发布一个容器化应用到 Kubernetes

部分参考：[Kubernetes: How to deploy a containerized app in Kubernetes using Kubernetes dashboard](#)

我们回到 Kubernetes Dashboard，点击右上角的 CREATE：



在上一篇文章 [AWS EC2 上安装 Docker 的示例](#) 中，我创建了一个 Docker 的镜像：

679435956173.dkr.ecr.us-east-2.amazonaws.com/hello-world，内容很简单，就是启动一个 Apache 服务器，发布一个 index.html 页面。

现在我们要将它通过 Kubernetes 部署到容器中并运行。

选择 CREATE AN APP：

- App name 可以自由输入，这里我们填写 xiang-k8s-hello-world
- 在 Container image 中填入我们之前创建的并推送到 Amazon Elastic Container Registry 的 Docker 镜像：679435956173.dkr.ecr.us-east-2.amazonaws.com/hello-world
- Number of pods 填入 1，表示现在只需要一台主机
- Service 选择 External
- 端口填入 9999 9999，将容器上公开的端口 9999 映射到主机系统上的端口 9999
- 最后点击 DEPLOY

CREATE FROM TEXT INPUT

CREATE FROM FILE

CREATE AN APP

App name *

xiang-k8s-hello-world

21 / 24

Container image *

679435956173.dkr.ecr.us-east-2.amazonaws.com/hello-world

Number of pods *

1

Service *

External

Port *

9999

Target port *

9999

Protocol *

TCP

Port

Target port

Protocol *

TCP

发布一个容器化应用到 Kubernetes

随后，点击 **Overview** 可以看到 Deployment, Replica Set 和 Pod 被成功创建，并且正在运行：

aces

it Volumes

Classes

is

Sets

ents

sets

on Controllers

Sets

and Load Balancing

t

Storage

Workloads Statuses

100.00%

Deployments

100.00%

Pods

100.00%

Replica Sets

Deployments

Name	Labels	Pods	Age	Images
xiang-k8s-hello-world	k8s-app: xiang-k8s-hello-world	1 / 1	18 minutes	679435956173.dkr.ecr.us-east-2.amazonaws.com/?

Pods

Name	Node	Status	Restarts	Age
xiang-k8s-hello-world-5bc8b7fc8f-xgm67	ip-172-20-59-124.us-east-2.compute.internal	Running	0	18 minutes

Replica Sets

Name	Labels	Pods	Age	Images
xiang-k8s-hello-world-5bc8b7fc8f	k8s-app: xiang-k8s-hello-world pod-template-hash: 1674639749	1 / 1	18 minutes	679435956173.dkr.ecr.us-east-2.amazonaws.com/?

Deployment, Replica Set 和 Pod 被成功创建

具体信息如下：

- 创建的 Deployment，即 Service： **xiang-k8s-hello-world**

- 创建的 Replica Set: `xiang-k8s-hello-world-5bc8b7fc8f`
- 创建的 Pod: `xiang-k8s-hello-world-5bc8b7fc8f-xgm67`

我们点击进入 Pod `xiang-k8s-hello-world-5bc8b7fc8f-xgm67` 的详情页面，可以看到它对应的 Node 是 `ip-172-20-59-124.us-east-2.compute.internal`（这是一个私有 DNS）：

Network

Node: `ip-172-20-59-124.us-east-2.compute.internal`
IP: `100.96.1.3`

Pod 对应的 Node

我们也可以点击进入 Node `ip-172-20-59-124.us-east-2.compute.internal` 的详情页面，可以看到它对应的 IP 和域名：

```
ip-172-20-59-124.us-east-2.compute.internal
beta.kubernetes.io/arch: amd64  beta.kubernetes.io/instance-type: t2.micro  beta.kubernetes.io/os: linux  failure-domain.beta.kubernetes.io/region: us-east-2  failure-domain.beta.kubern
show all
node.alpha.kubernetes.io/ttl: 0  volumes.kubernetes.io/controller-managed-attach-detach: true
Time: 2018-11-14T04:39 UTC
IS: InternalIP: 172.20.59.124  ExternalIP: 18.223.182.144  InternalDNS: ip-172-20-59-124.us-east-2.compute.internal  ExternalDNS: ec2-18-223-182-144.us-east-2.compute.amazonaws.com
Hostname: ip-172-20-59-124.us-east-2.compute.internal
```

Node 对应的 IP 和域名

可以看出，这就是我们在上面所创建的三个 EC2 实例中的第三个实例，我们的应用 `xiang-k8s-hello-world` 就是部署在这个实例的 Docker 容器里。

- InternalIP: `172.20.59.124`
- ExternalIP: `18.223.182.144`
- InternalDNS: `ip-172-20-59-124.us-east-2.compute.internal`
- ExternalDNS: `ec2-18-223-182-144.us-east-2.compute.amazonaws.com`
- Hostname: `ip-172-20-59-124.us-east-2.compute.internal`

回到 Pod 的详情页面，我们也可以看到 Event 的历史，包括容器的创建，获取镜像等等：

vents

Message	Source
Successfully assigned xiang-k8s-hello-world-5bc8b7fc8f-xgm67 to ip-172-20-59-124.us-east-2.compute.internal	default-scheduler
MountVolume.SetUp succeeded for volume "default-token-btwg9"	kubelet ip-172-20-59-124.us-east-2.compute.internal
pulling image "679435956173.dkr.ecr.us-east-2.amazonaws.com/hello-world"	kubelet ip-172-20-59-124.us-east-2.compute.internal
Successfully pulled image "679435956173.dkr.ecr.us-east-2.amazonaws.com/hello-world"	kubelet ip-172-20-59-124.us-east-2.compute.internal
Created container	kubelet ip-172-20-59-124.us-east-2.compute.internal

Event 的历史，包括容器的创建，获取镜像等等

我们也可以点击进入到 Service `xiang-k8s-hello-world` 的详情页面，看到对应的 Internal endpoints 和 External endpoints：

nnection	
ster IP:	100.64.223.195
ernal endpoints:	xiang-k8s-hello-world:9999 TCP xiang-k8s-hello-world:30009 TCP
ernal endpoints:	ad7cddd5fe8b711e8a334061f750d5e6-1358820683.us-east-2.elb.amazonaws.com:9999 

对应的 Internal endpoints 和 External endpoints

我们也可以通过 `kubectl get services xiang-k8s-hello-world` 和 `kubectl describe svc xiang-k8s-hello-world` 命令查看该 Service 的详情：

```
[ec2-user@ip-172-31-38-187 ~]$ kubectl get services xiang-k8s-hello-world
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
xiang-k8s-hello-world             LoadBalancer       100.64.223.195  ad7cddd5fe8b7... 9999:30009/TCP   1d
```

```
[ec2-user@ip-172-31-38-187 ~]$ kubectl describe svc xiang-k8s-hello-world
Name:                               xiang-k8s-hello-world
Namespace:                           default
Labels:                               k8s-app=xiang-k8s-hello-world
Annotations:                           <none>
Selector:                             k8s-app=xiang-k8s-hello-world
Type:                                 LoadBalancer
IP:                                   100.64.223.195
LoadBalancer Ingress:                ad7cddd5fe8b711e8a334061f750d5e6-1358820683.us-east-2.elb.amazonaws.com
Port:                                 tcp-9999-9999-9tlmx 9999/TCP
TargetPort:                           9999/TCP
NodePort:                             tcp-9999-9999-9tlmx 30009/TCP
Endpoints:                             100.96.1.3:9999
Session Affinity:                     None
External Traffic Policy:               Cluster
```

```
Events:
  Type     Reason              Age             From              Message
  ----     -
  Normal   EnsuringLoadBalancer 59m (x2 over 1d) service-controller Ensuring load balancer
  Normal   Type                 59m             service-controller NodePort -> LoadBalancer
  Normal   EnsuredLoadBalancer 59m (x2 over 1d) service-controller Ensured load balancer
```

ad7cddd5fe8b711e8a334061f750d5e6-1358820683.us-east-2.elb.amazonaws.com 又是从哪里来的呢？我们打开 AWS 控制台，可以看到自动创建了一个 **负载均衡器** ad7cddd5fe8b711e8a334061f750d5e6，它提供了这个域名：

网络接口

负载均衡

负载均衡器

示群组

TO SCALING

边配置

to Scaling 组

能管理器服务

行命令

态管理器

置合规性

边化

丁合规性

丁基准

能管理器共享资源

容灾实例

负载均衡器: ad7cddd5fe8b711e8a334061f750d5e6

描述

实例

运行状况检查

侦听器

监控

标签

迁移

基本配置

名称: ad7cddd5fe8b711e8a334061f750d5e6

* DNS 名称: ad7cddd5fe8b711e8a334061f750d5e6-1358820683.us-east-2.elb.amazonaws.com (A 记录)

类型: Classic (立即迁移)

模式: internet-facing

可用区: subnet-01668e0cbc3c6357b - us-east-2b

端口配置

端口配置: 9999 (TCP) 正在转发到 30009 (TCP)

粘性选项对 TCP 协议不可用

小礼物走一走，来简书关注我