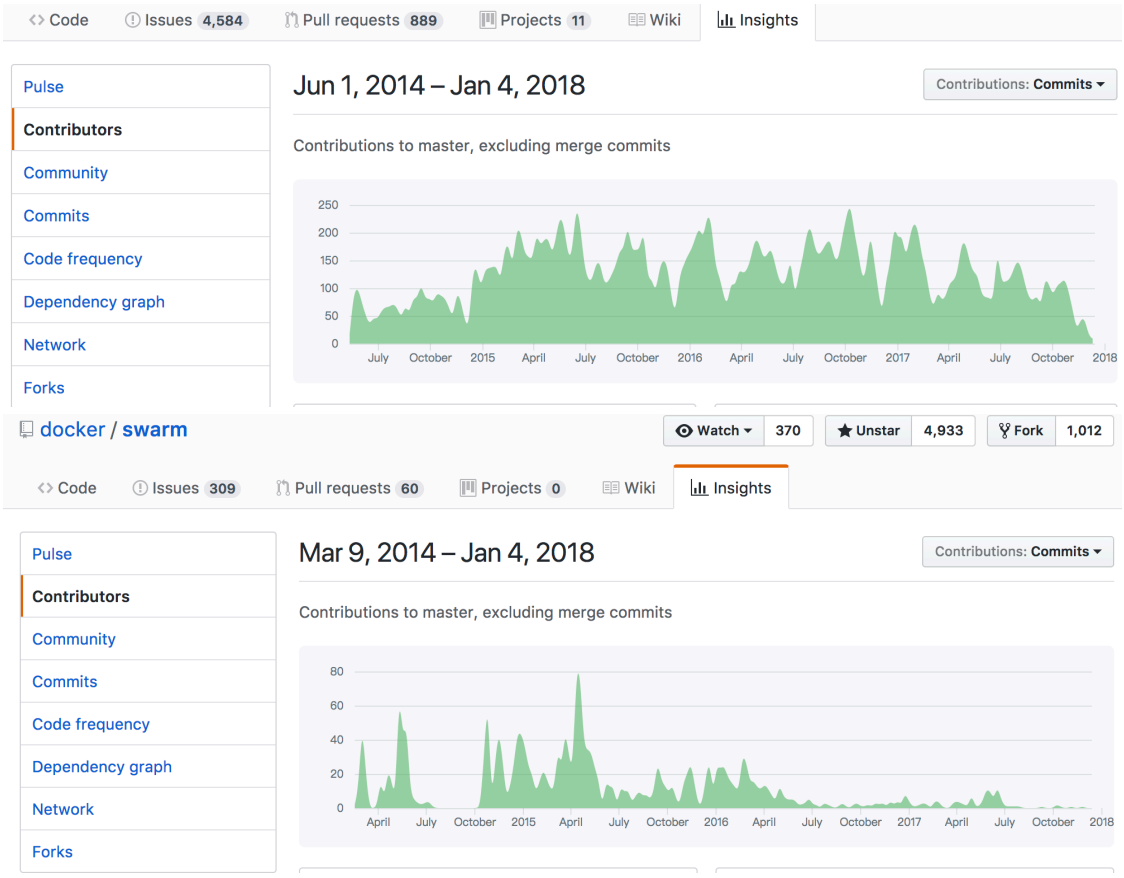
Feel K8s

前言

k8s vs swarm



> Kubernetes处于快速生长期，项目代码频繁重构

2017 k8s 回顾

> - 集群规模： 3000节点 -> 5000节点
> - 调度能力大增强
> - 存储增强
> - 安全性增强

What is K8s

- 敏捷的应用程序创建和部署：

与虚拟机映像使用相比，提高了容器映像创建的便捷性和效率。

- 持续开发，集成和部署： 提供可靠，频繁的容器映像构建和部署，快速简单的回滚（由于映像不可变）。
- Dev和Ops分离问题： 在构建/发布时间而不是部署时间创建应用程序容器映像，从而将应用程序与基础架构分离。
- 开发，测试和生产过程中的环境一致性： 在笔记本电脑上运行和云中一样。
- 云和操作系统的可移植性： 运行在Ubuntu，RHEL，CoreOS，本地，谷歌Kubernetes引擎，以及其他任何地方。
- 以应用程序为中心的管理： 提高在虚拟硬件上运行操作系统的抽象级别，以使用逻辑资源在操作系统上运行应用程序。
- 松散耦合的，分布式的，弹性的，解放的微服务： 应用程序被分解为更小的，独立的部分，可以被动态地部署和管理，而不是在一台大的单一用途的机器上运行的胖胖的单片堆栈。
- 资源隔离：可预测的应用程序性能。
- 资源利用：高效率和密度。

> 从以主机为中心的基础设施转移到以集装箱为中心的基础设施

K8s 满足生产中运行的应用程序的一些常见需求

- 协同定位帮助程序进程，为复合应用程序提供便利，并保留每个容器一个应用程序模型
- 安装存储系统
- 秘密分发
- 检查应用程序健康
- 复制应用程序实例
- 使用水平荚Autoscaling
- 命名和发现

- 平衡负载
- 滚动更新
- 监视资源
- 访问和摄取日志
- 调试应用程序
- 提供认证和授权

K8s 不提供的

- 不限制支持的应用程序的类型。 它并不指定应用程序框架（例如Wildfly），限制支持的语言运行时（例如，Java，Python，Ruby）集合，仅迎合12个因子的应用程序，也不区分应用程序和服务。Kubernetes旨在支持各种各样的工作负载，包括无状态，有状态和数据处理工作负载。如果一个应用程序可以在一个容器中运行，它应该在Kubernetes上运行。

- 不提供作为内置服务的中间件（如消息总线），数据处理框架（例如Spark），数据库（例如mysql）或集群存储系统（例如Ceph）。这种应用程序运行在Kubernetes上。

- 没有点击式部署服务市场。

- 不部署源代码，不建立你的应用程序。持续集成（CI）工作流程是一个区域，不同的用户和项目有自己的需求和偏好，所以它支持在Kubernetes上分层CI工作流程，但并不指定分层应该如何工作。

- 允许用户选择他们的日志，监控和警报系统。（它提供了一些整合作为概念验证。）

- 不提供或强制执行全面的应用程序配置语言/系统（例如jsonnet）。

- 不提供或不采用任何全面的机器配置，维护，管理或自我修复系统。

OverView

预备

单机安装

k8s包管理器

docker 安装 k8s

minikube 安装

Docker 安装与重用

启动k8s mini

```
minikube start --docker-env HTTP_PROXY=http://proxy-ip:port \
    --docker-env HTTPS_PROXY=http://proxy-ip:port \
    --network-plugin=cni \
    --host-only-cidr 172.17.17.1/24 \
    --extra-config=kubelet.ClusterCIDR=192.168.0.0/16 \
    --extra-config=proxy.ClusterCIDR=192.168.0.0/16 \
    --extra-config=controller-manager.ClusterCIDR=192.168.0.0/16
```

启动报错后，启用日志

```
// First Step
minikube start --vm-driver=hyperkit  --v=10 --alsologtostderr
// ReCreateCluster √
minikube delete
minikube start --vm-driver=hyperkit --v=10 --alsologstostderr
minikube dashboard
kubectl config use-context minikube
```

```
curl -O -L https://docs.projectcalico.org/v2.6/getting-started/kubernetes/installation/hosted/kubeadm/1.6/calico.yaml
sed -i -e '/nodeSelector/d' calico.yaml
sed -i -e '/node-role.kubernetes.io\/master: ""/d' calico.yaml
sed -i -e 's/10\.96\.232/10.0.0/' calico.yaml
kubectl apply -f calico.yaml
```

> 安装脚本

OverView

理解 Kubernetes 对象

在 Kubernetes 系统中，Kubernetes 对象 是持久化的条目.

描述了如下信息：

什么容器化应用在运行（以及在哪个 Node 上） 可以被应用使用的资源 关于应用如何表现的策略，比如重启策略、升级策略，以及容错策略


```
{
  "kind": "Deployment",
  "apiVersion": "extensions/v1beta1",
  "metadata": {
    "name": "helloworld",
    "namespace": "default",
    "selfLink":
"/apis/extensions/v1beta1/namespaces/default/deployments/helloworld",
    "uid": "31dcd97d-f39a-11e7-80d1-12112dbe6b6b",
    "resourceVersion": "36790",
    "generation": 4,
    "creationTimestamp": "2018-01-07T11:02:04Z",
    "labels": {
      "run": "helloworld"
    },
    "annotations": {
      "deployment.kubernetes.io/revision": "1"
    }
  },
  "spec": {
    "replicas": 4,
    "selector": {
      "matchLabels": {
        "run": "helloworld"
      }
    },
    "template": {
      "metadata": {
        "creationTimestamp": null,
        "labels": {
```

```
          "run": "helloworld"
        }
      },
      "spec": {
        "containers": [
          {
            "name": "helloworld",
            "image": "dockercloud/hello-world",
            "ports": [
              {
                "containerPort": 80,
                "protocol": "TCP"
              }
            ],
            "resources": {},
            "terminationMessagePath": "/dev/termination-log",
            "terminationMessagePolicy": "File",
            "imagePullPolicy": "Always"
          }
        ],
        "restartPolicy": "Always",
        "terminationGracePeriodSeconds": 30,
        "dnsPolicy": "ClusterFirst",
        "securityContext": {},
        "schedulerName": "default-scheduler"
      }
    },
    "strategy": {
      "type": "RollingUpdate",
      "rollingUpdate": {
        "maxUnavailable": 1,
        "maxSurge": 1
      }
    }
  },
  "status": {
    "observedGeneration": 4,
    "replicas": 4,
    "updatedReplicas": 4,
    "readyReplicas": 4,
    "availableReplicas": 4,
    "conditions": [
      {
        "type": "Available",
        "status": "True",
        "lastUpdateTime": "2018-01-07T11:23:03Z",
        "lastTransitionTime": "2018-01-07T11:23:03Z",
        "reason": "MinimumReplicasAvailable",
```

```
        "message": "Deployment has minimum availability."
      }
    ]
  }
}
```

> spec 必须提供，它描述了对象的 期望状态 —— 希望对象所具有的特征。
> status 描述了对象的 实际状态，它是由 Kubernetes 系统提供和更新

Pod

- Inject Info into Application

Service

Service为一组Pod（通过labels来选择）提供一个统一的入口，并为它们提供负载均衡和自动服务发现。比如，可以为前面的nginx-app创建一个service：

- ClusterIP：默认类型，自动分配一个仅cluster内部可以访问的虚拟IP
- NodePort：在ClusterIP基础上为Service在每台机器上绑定一个端口，这样就可以通过<NodeIP>:NodePort来访问该服务
- LoadBalancer：在NodePort的基础上，借助cloud provider创建一个外部的负载均衡器，并将请求转发到<NodeIP>:NodePort
- ExternalName：将服务通过DNS CNAME记录方式转发到指定的域名（通过spec.externlName设定）。需要kube-dns版本在1.7以上。

注意点

- kubectl run 是先创建一个Deployment资源（replicas=1），再由与Deployment关联的ReplicaSet来自动创建Pod，这等价于这样一个配置：

Volume

> 容器中的磁盘文件是短暂的，这在运行在容器中的非平凡的应用程序中存在一些问题。首先，当容器崩溃时，kubelet会重新启动它，但是文件将会丢失 - 容器以干净的状态启动。其次，在一起运行容器时，Pod通常需要在这些容器之间共享文件。Kubernetes Volume抽象解决了这两个问题。

Kubernetes Volume的生命周期与Pod绑定

> 容器挂掉后Kubelet再次重启容器时，Volume的数据依然还在 而Pod删除时，Volume才会清理。数据是否丢失取决于具体的Volume类型，比如emptyDir的数据会丢失，而PV的数据则不会丢

K8s in Action Now

> Config Docker Register Mirror

- bqr1dr1n.mirror.aliyuncs.com
- https://registry.docker-cn.com

How to access my pod

- hostNetwork
- hostPort
- NodePort
- LoadBalancer
- Ingress

本地加载

kubectl run hello-minikube --image=tomcat:8.0 --port=8080 --image-pull-policy=IfNotPresent

---