|  |
| --- |
| 网易 |
| kubernetes |
|  |
|  |
| **方应** |
|  |

|  |
| --- |
|  |

**kubernetes分析**

# kubernetes简介

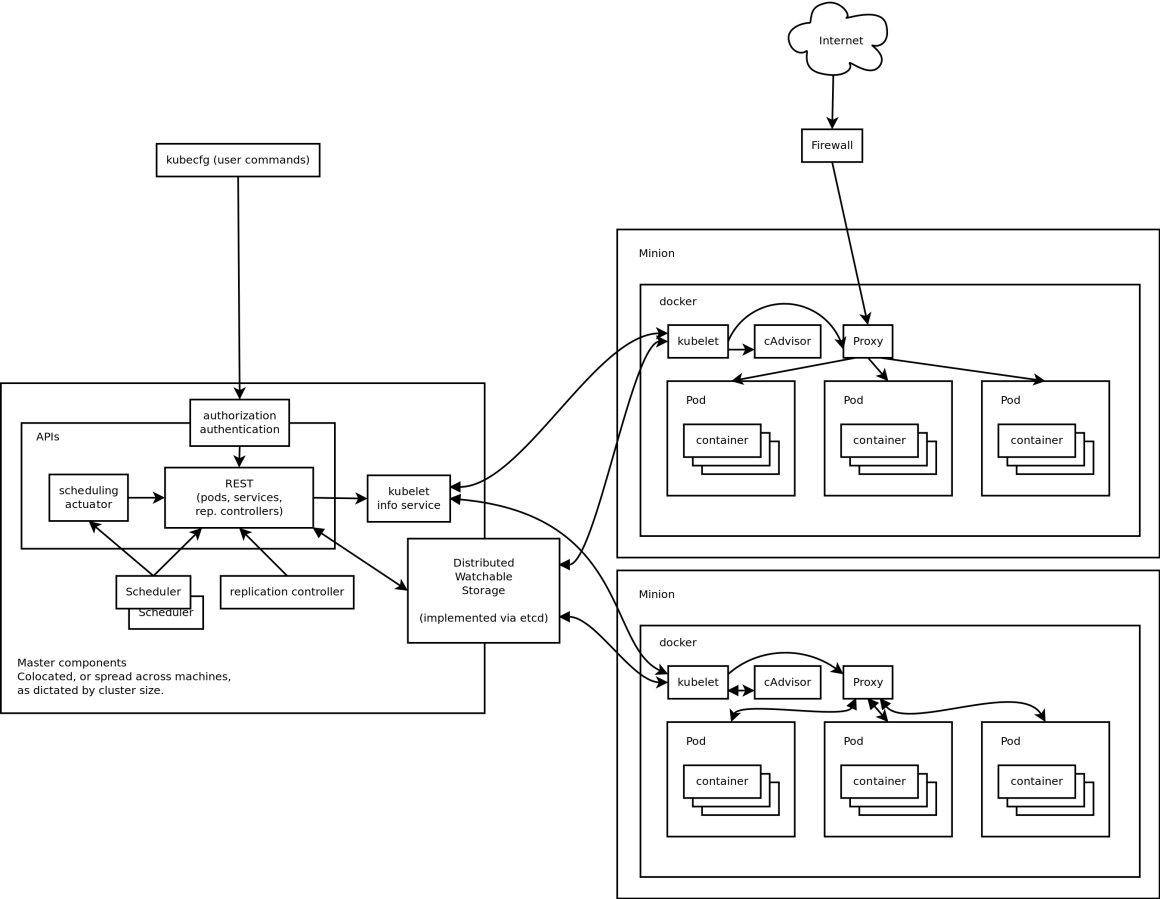
* 1. kubernetes概念
  2. kubernetes功能
     1. kubernetes优劣势

表1-1 xxx对比

|  |  |  |  |
| --- | --- | --- | --- |
|  | kubernetes | Coreos&fleet |  |
| 对比项1 |  |  |  |
| 对比项2 |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

* + 1. 适用场景
  1. kubernetes系统结构

kubernetes主要分为etcd、master和minion，etcd作为key value存储，master上运行apiserver、controller-manager和scheduler，minion上运行kubelet（agent）、proxy和cadvisor。各个组件功能详见实现章节，整个系统结构如图所示。



# kubernetes使用

* 1. kubernetes环境搭建
     1. 系统依赖

kubernetes依赖来源于其对docker的依赖，内核版本至少2.6.32以上，docker对于发行版的支持，可参照docker官方说明。本文示例运行在debian7，内核版本3.10上。

列举依赖项。

在kubernetes集群内运行的docker镜像可以为docker hub上的镜像也可以为本地镜像。

docker镜像库：[***https://registry.hub.docker.com/***](https://registry.hub.docker.com/)

私有镜像库：[***http://registry.hz.netease.com/***](https://registry.hub.docker.com/)

这里列举下kubernetes用到的资源地址：

etcd：***https://github.com/coreos/etcd/***

kubernetes：***https://github.com/GoogleCloudPlatform/kubernetes/***

cadvisor：***https://github.com/google/cadvisor***

docker：通过源安装

* + 1. kubernetes安装

kubernetes安装主要分成3部分：etcd集群、master集群和minions。

本文为了方便以9台云主机为例搭建一个kubernetes集群，云主机机器分配如下所示：

|  |  |
| --- | --- |
| ip | 角色 |
| 10.180.164.21 | Kubernetes master 1 |
| 10.180.164.25 | Kubernetes master 2 |
| 10.180.164.28 | Kubernetes master 3 |
| 10.180.164.24 | Etcd node 1 |
| 10.180.164.25 | Etcd node 2 |
| 10.180.164.27 | Etcd node 3 |
| 10.180.164.23 | Apiserver nginx |
| 10.180.64.8 | Kubernetes minion1 |
| 10.180.64.9 | Kubernetes minion2 |

* + - 1. etcd集群

本次示例中以3台云主机作为etcd node，采用public etcd discovery。

运行方式：docker

镜像 地址：

**root@commonqa-etcd-cluster-3:~# docker images**

**REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE**

**registry.hz.netease.com/kubernetes/etcd 2.0.13 007cfe2b42b5 About an hour ago 91.67 MB**

安装过程：

1、从git下载脚本，安装过程所需的脚本都在这里

https://git.hz.netease.com/nas/scripts-nce/tree/master/v2.0

改变脚本执行权限，在master上先执行etcd-initial-2.0.sh

**root@cnsdev-paas-master:~#** ./ etcd-initial-2.0.sh

这个脚本的工作是初始化etcd的环境和一些配置文件。

2、脚本执行成功之后需要手动修改一下/etc/default/etcd文件，

**root@commonqa-master-cluster-1:~/fangying/script/master# cat /etc/default/etcd**

**# etcd configure**

**#ETCD\_NAME="-name etcd-3"**

**#ETCD\_NODE\_IP="10.180.164.26"**

**#ETCD\_DISCOVERY\_TOKEN="https://discovery.etcd.io/e4c3d88c6e08b952c2ed390600130aba"**

ETCD\_NAME是你希望本机在etcd集群中的名字，也可以不设置；ETCD\_NODE\_IP是用来提供服务的ip地址，对于有多个ip的机器，自己选择合适的ip地址，这里选择的是etcd的机房网ip，这里也可以不配置，在运行docker的时候通过环境变量传入也可以，名称和这里一致；ETCD\_DISCOVERY\_TOKEN是etcd集群用于服务发现的地址，这个地址有两种方式获取，这里使用的是etcd提供的公共服务，执行以下命令就可获取

**curl https://discovery.etcd.io/new?size=3**

3是etcd集群的数量

3、填好配置之后执行etcd-start-2.0.sh，这个脚本有一个参数new/exist，new代表创建一个新的集群，exist表示运行一个老的集群

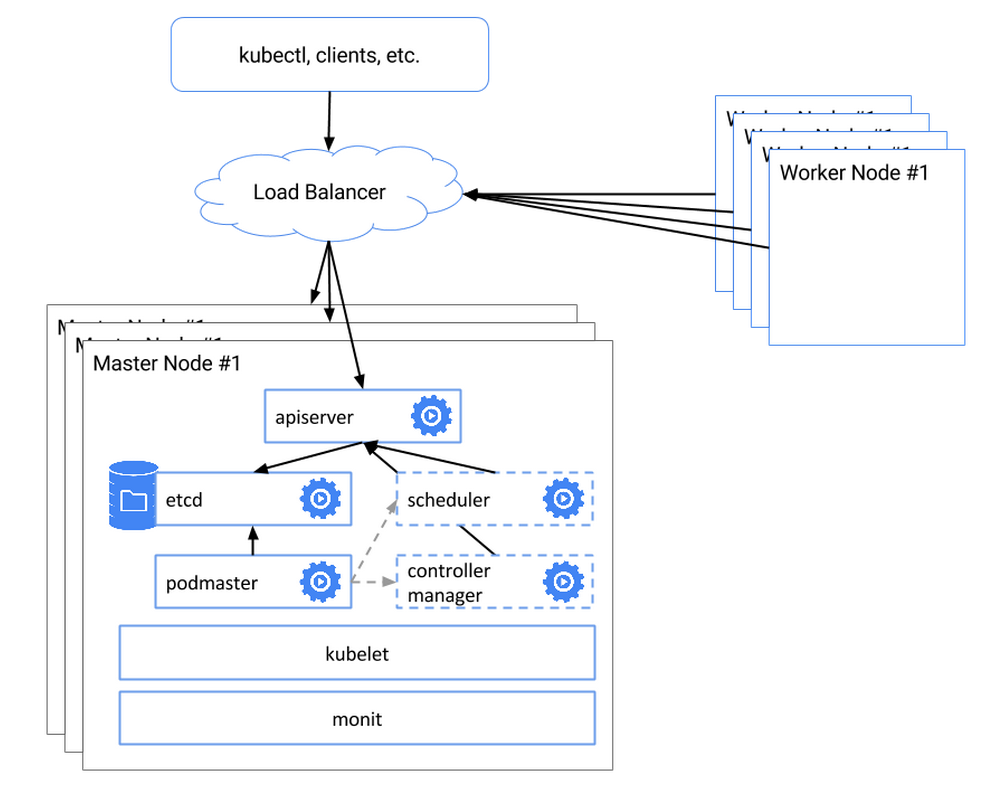
**root@cnsdev-paas-master:~#** ./ etcd-start-2.0.sh new

这个脚本会检测docker服务的配置和状态，然后运行docker启动etcd，etcd容器将会和host共享网络。

etcd更多的配置已经打入镜像，但是提供两种方式进行修改

* 通过/etc/default/etcd配置文件修改
* 通过参数传递给etcd-start-2.0.sh
  + - 1. master节点

先放一张master的架构图，便于从全局上了解master的运行架构。



运行方式：docker

镜像地址：

root@commonqa-master-cluster-1:~/fangying/script/master# docker images

REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE

registry.hz.netease.com/kubernetes/apiserver 1.0.2 c81b6c008acd 18 hours ago 124.9 MB

registry.hz.netease.com/kubernetes/podmaster 1.1.3 d7c0e808f12b 20 hours ago 91.56 MB

registry.hz.netease.com/kubernetes/scheduler 1.0.1 629e91811e48 23 hours ago 102.9 MB

registry.hz.netease.com/kubernetes/controller 1.0.1 06bf0cadb72a 23 hours ago 118.3 MB

registry.hz.netease.com/kubernetes/pause 0.8.0 e3682f2c46b1 27 hours ago 758.5 kB

monit是一个检测进程运行状态的工具，master运行架构可以如下表述：

* 运行kubelet，kubelet可以通过json文件来启动pod，在pod里可以运行master的各个组件
* apiserver运行在pod里，直接有kubelet来运行

podmaster是一个类似于HA的东西，由kubelet直接运行，podmaster每个master上都有一个

* podmaster运行controller和scheduler，由于controller和scheduler在集群里只有一个实例，所以podmaster会保证其高可用，某个master节点的controller挂掉，另外master节点的podmaster将会拉起本节点的controller接替工作

1、执行master-initial-2.0.sh初始化master环境

**root@cnsdev-paas-master:***~#*./master-initial-2.0.sh

这个脚本负责安装docker service、kubelet service，拷贝apiserver、controller、scheduler和podmaster用的的json文件。

2、手动修改/etc/default/kubernetes文件

**root@commonqa-master-cluster-1:~/fangying/script/master# cat /etc/default/kubernetes**

**# kubernetes config**

**ETCD\_NODES\_LIST="10.180.164.24,10.180.164.26,10.180.164.27"**

**PROTECT\_LAN\_IP=10.180.164.23**

ETCD\_NODES\_LIST是已经创建的etcd集群的几点ip列表，由于etcd集群大小和ip是不固定的，所以需要手动修改，这里没有用到saltstack或者pupet之类的配置工具，所以采用手动修改。

PROTECT\_LAN\_IP这个名字和含义有点不符合，其代表的是apiserver的nginx代理服务器的ip地址，nginx对外服务地址，apiserver采用负载均衡方式，流量由nginx进行转发，这里就是nginx的云主机的机房ip **10.180.164.23**

3、最后一步就是执行master-start-2.0.sh

**root@cnsdev-paas-master:***~#*./master-start-2.0.sh

这个脚本启动docker服务和kubelet服务，然后apiserver、podmaster、controller和scheduler就duang的一下都起来了，这里采用了3个节点的集群，综合效果是3个apiserver容器分别运行在每个master节点上，每个master节点还运行了2个podmaster容器，分别监控着集群的controller和scheduler服务是否正常。controller和scheduler整个集群分别只有1个容器，至于运行在那个master节点上那就看哪个master节点的podmaster比较腻害了。

* + - 1. nginx负载均衡

运行方式：docker

镜像地址：

nginx负责对apiserver的请求进行负载均衡，安装过程也很简单

1、执行nginx-initial-2.0.sh

**root@cnsdev-paas-master:***~#*./master-start-2.0.sh

2、修改/etc/hosts文件，由于暂时没有使用dns，所以在该文件中加入如下信息

**10.180.164.25 kube-apiserver-1**

**10.180.164.28 kube-apiserver-2**

**10.180.164.21 kube-apiserver-3**

nginx和host共享网络，所以修改host的hosts文件，nginx即可根据配置文件将请求分流道各个apiserver上

3、执行nginx-start-2.0.sh

**root@cnsdev-paas-master:***~#*./master-start-2.0.sh

所有访问kubernetes master的请求都访问nginx的ip：8080端口

* + - 1. minion节点

minion节点的安装和老的版本基本相同，可以参考老的部署文档。

* 1. kubernetes配置文件

通过docker的方式，可用性和部署集群更加方便，采用的部分默认配置可以通过修改/etc/default/xxx对应的文件来修改配置，下面说明一些默认的配置。这些配置文件都在其对应的docker镜像中，运行时通过共享host文件的方式将host上对应的配置文件的值覆盖默认的值。

* + 1. etcd配置文件

etcd配置文件

ETCD\_NAME="-name etcd-1"

etcd节点名称，默认名称为default。

ETCD\_PEER\_ADDRESS="-initial-advertise-peer-urls http://hostip:7001"

etcd集群之间node的通讯地址，一般指定7001或者2380端口

ETCD\_CLIENT\_ADDRESS="-advertise-client-urls http://hostip:4001"

etcd node对外服务的地址，一般指定4001或者2379端口

ETCD\_DATA\_DIR="-data-dir /home/data/etcd"

etcd存储数据的目录，docker运行时将host的/home/data/etcd/映射到了容器的相应目录下

ETCD\_LISTEN\_PEER\_ADDRESS="-listen-peer-urls http://0.0.0.0:2380"

etcd node监听的地址，如果为0.0.0.0将会监听所有接口，这里配置为[http://0.0.0.0:](http://0.0.0.0:7001/)2380。

ETCD\_LISTEN\_CLIENT\_ADDRESS="-listen-client-urls http://0.0.0.0:2379"

对外服务监听地址，配置为[http://0.0.0.0:](http://0.0.0.0:4001/)2379。

ETCD\_CLUSTER\_STATE="-initial-cluster-state new"

etcd集群状态，new表示新建一个集群，existing表示已经存在。

ETCD\_ARGS=""

需要额外添加的参数，可以自己添加，etcd的所有参数可以通过etcd -h查看。

* + 1. kubernetes集群配置文件

common：

KUBE\_LOGTOSTDERR="--logtostderr=true"

表示错误日志记录到文件还是输出到stderr。

KUBE\_LOG\_LEVEL="--v=0"

日志等级。

KUBE\_ALLOW\_PRIV="--allow\_privileged=false"

允许运行特权容器。

* + 1. apiserver配置文件

apiserver：

KUBE\_API\_ADDRESS="--address=0.0.0.0"

监听的接口，如果配置为127.0.0.1则只监听localhost，配置为0.0.0.0会监听所有接口，这里配置为0.0.0.0。

KUBE\_API\_PORT="--port=8080"

apiserver的监听端口，默认8080，不用修改。

KUBELET\_PORT="--kubelet\_port=10250"

minion上kubelet监听的端口，默认10250，无需修改

KUBE\_SERVICE\_ADDRESSES="--portal\_net=10.254.0.0/16"

kubernetes可以分配的ip的范围，kubernetes启动的每一个pod以及serveice都会分配一个ip地址，将从这个范围分配。

KUBE\_API\_ARGS=""

需要额外添加的配置项，简单地启用一个集群无需配置。

* + 1. controller配置文件

controller-manager：

KUBE\_CONTROLLER\_MANAGER\_ARGS=""

需要额外添加的参数

* + 1. scheduler配置文件

schedule：

如果需要额外参数可以自行添加，这里暂时不添加新的参数。

* + 1. kubelet配置文件

kubelet：

KUBELET\_ADDRESS="--address=10.180.64.8"

minion监听的地址，每个minion根据实际的ip配置，这里minion1上为10.180.64.8，minion2上为10.180.64.9。

KUBELET\_PORT="--port=10250"

监听端口，不要修改，如果修改，同时需要修改master上配置文件中涉及的配置项。

KUBELET\_HOSTNAME="--hostname\_override=10.180.64.8"

kubernetes看到的minion的名称，使用kubecfg list minions时看到的将是这个名称而不是hostname，设置和ip地址一样便于识别。

KUBELET\_ARGS=""

额外增加的参数

* + 1. proxy配置文件

proxy：

如有额外参数自行配置，这里不需要添加。

* 1. kubernetes启动

所有组件都运行之后，到master上检测下状态。

查看下集群状况

[**root@cnsdev-paas-master**](mailto:root@cnsdev-paas-master)**:**~# kubectl get node

NAME LABELS STATUS

10.180.164.44 kubernetes.io/hostname=10.180.164.44 Ready

查看当前集群的pod

[**root@cnsdev-paas-master**](mailto:root@cnsdev-paas-master)**:~#** kubectl get pod   
NAME READY STATUS RESTARTS AGE

kube-apiserver-commonqa-master-cluster-1 1/1 Running 1 19h

kube-apiserver-commonqa-master-cluster-2 1/1 Running 0 4h

kube-apiserver-kube-master-1 1/1 Running 0 22h

kube-controller-manager-commonqa-master-cluster-1 1/1 Running 1 19h

kube-scheduler-commonqa-master-cluster-2 1/1 Running 0 3h

scheduler-master-commonqa-master-cluster-1 2/2 Running 2 19h

scheduler-master-commonqa-master-cluster-2 2/2 Running 4 3h

testpod1 1/1 Running 0 19h

启动集群时，master节点的kubelet配置了nginx的ip地址，所以查看pod看到了master自己的pod列表

* 1. kubernetes接口
     1. kubectl

表3-1 kubectl总览

|  |  |
| --- | --- |
| 名称 | 功能 |
| kubectl proxy [flags] |  |
| kubectl get [(-o|--output=)json|yaml|...] <resource> [<id>] [flags] |  |
| kubectl describe <resource> <id> [flags] |  |
| kubectl create -f filename [flags] |  |
| kubectl createall [-d directory] [-f filename] [flags] |  |
| kubectl update -f filename [flags] |  |
| kubectl delete ([-f filename] | (<resource> [(<id> | -l <label>)] [flags] |  |
| kubectl log [-f] <pod> [<container>] [flags] |  |
| namespace [<namespace>] |  |

* + - 1. kubectl proxy

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | proxy | 功能 | 创建一个到apiserver的本地代理 |
| 命令形式 | kubectl proxy [--port=PORT] [--www=static-dir] [--www-prefix=prefix] [--api-prefix=prefix] [flags] | | |
| 参数解析 | | | |
|  | | | |
| 样例解析 | | | |
| 例子：  解释： | | | |

* + - 1. kubectl get

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | get | 功能 | **获取资源信息，包括**pods (po), replication controllers (rc), services (se), minions (mi), or events (ev) |
| 命令形式 | kubectl get [(-o|--output=)json|yaml|template|wide|...] (RESOURCE [NAME] | RESOURCE/NAME ...) [flags] | | |
| 参数解析 | | | |
| *resource：pods (po), replication controllers (rc), services (se), minions (mi), or events (ev)等*  *resource/name：resource类型加上对应object的名称*  *[--all-namespace]：默认false，指定该参数将会列出所有namespace中满足条件的资源信息*  *[-L，--label-columns]：逗号分隔的label列表，每个label会有一列在信息的最后，每个pod会针对该label显示是否带有该label*  *[--no-headers]：如果使用default，不会显示前面的标题行*  *[-o，--output]：输出的格式，json、yaml、template、templatefile、wide*  *[--output-version]：输出的api-version*  *[-l，--selector]：用于筛选的label*  *[-t、--template]：template string或者templatefile 路径*  *[-w，--watch]：获取信息之后继续watch变化*  *[--watch-only]：不列出信息直接watch，只在有变更的时候列出信息* | | | |
| 样例解析 | | | |
| 例子：*kubectl get pods*  解释：列出所有的*pods，*风格为普通ls类  例子：*kubectl get pods e473c35e-961d-11e4-bc28-fa163e8b5289*  解释：列出id为*e473c35e-961d-11e4-bc28-fa163e8b5289*的pods*，*风格为普通ls类，如果为replicationController，那么id为其NAME栏所示  例子：*kubectl get -o json pod e473c35e-961d-11e4-bc28-fa163e8b5289*  解释：列出指定id的pod的详细信息*，*风格为json格式  例子：*kubectl get pod -L name=wordpress-pod,name=wordpress-pod1*  解释： | | | |

* + - 1. kubectl describe

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | describe | 功能 | **调用多个api将resource的各种信息详细列出来** |
| 命令形式 | kubectl describe (RESOURCE NAME\_PREFIX | RESOURCE/NAME) [flags] | | |
| 参数解析 | | | |
| *resource：资源名称，详见get*  *[name\_prefix]：资源的前缀，比如同一个rc创建的pod都是具有相同前缀的，用这个前缀就可以删除这些pod*  *[resource/name]：资源类型加名字的方式*    *[flags]：*  *-l，--selector："" 用于筛选的label* | | | |
| 样例解析 | | | |
| 例子：kubectl describe pod/scheduler-master-commonqa-master-cluster-2 或者  kubectl describe pod scheduler-master-commonqa-master-cluster-2  解释：列出指定pod scheduler-master-commonqa-master-cluster-2的各种信息  例子：kubectl describe pod wordpress  解释：列出名称前缀是wordpress的所有pod的详细信息  例子：kubectl describe pod -l name=wordpress-pod  解释：列出label为name=wordpress-pod的所有pod的详细信息 | | | |
|  | | | |

* + - 1. kubectl create

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | create | 功能 | **根据指定文件或者标准输入创建一个其描述的资源** |
| 命令形式 | kubectl create -f filename [flags] | | |
| 参数解析 | | | |
| *filename：描述资源各种信息的文件，格式为json或者yaml*  *[flags]：* | | | |
| 样例解析 | | | |
| 例子：kubectl create -f tomcat-pod.json  解释：根据tomca-pod.json创建相关资源，资源类型为文件里面描述的类型 | | | |

* + - 1. kubectl replace

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | replace | 功能 |  |
| 命令形式 | kubectl replace -f FILENAME [flags] | | |
| 参数解析 | | | |
| *filename：创建的资源的名称*  *[--cascade]：默认false，当使用force时，将会删除本资源管理的其他资源，比如replace rc时，其创建的pod将被删除*  *[--force]：默认false，强制替换，先删除再创建*  *[--grace-period]：默认-1，force使用时有效，资源强制回收前的等待时间*  *[--timeout]：默认0，force使用时有效，放弃delete的时间，为0表示根据资源的多少决定时间* | | | |
| 样例解析 | | | |
| 例子：kubectl replace -f pod.json  解释：用pod.json中的资源替换相同名称的资源  例子：kubectl replace --force -f pod.json  解释：强制替换 | | | |

* + - 1. kubectl patch

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | patch | 功能 | 为资源新增加配置项 |
| 命令形式 | kubectl patch RESOURCE NAME -p PATCH [flags] | | |
| 参数解析 | | | |
| *resource：资源的类型*  *name：资源名称*  *-p，--patch：需要对资源json文件添加的配置项，采用json格式* | | | |
| 样例解析 | | | |
| 例子：kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'  解释：设置node为不可调度状态，新增加unschedulable状态 | | | |

* + - 1. kubectl delete

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | delete | 功能 | 删除资源 |
| 命令形式 | kubectl delete ([-f filename] | (<resource> [(<id> | -l <label> --all)] [flags] | | |
| 参数解析 | | | |
| *[--all]：false，选择所有资源*    *[-f，--filename=[]]：文件、目录以及url，要删除的文件、存文件目录、url描述的资源*  *[--cascade]:true 层级式删除当前资源管理的资源*  *[--grace-period]:-1 资源优雅结束的时间，负数被忽略*  *[--ignore-not-found]：false 将没有找到相关资源作为成功的删除*  *[-l,--selector]:"" 用于过滤资源的label*  *[--timeout]:0 放弃当前操作前的等待时间，0值代表根据资源大小来评估* | | | |
| 样例解析 | | | |
| 例子：kubectl delete -f tomcat-pod.json  解释：通过创建时的json或者yaml文件删除  例子：kubectl delete pods -l name=tomcat  解释：通过label删除，label为创建时指定  例子：kubectl delete pod e473c35e-961d-11e4-bc28-fa163e8b5289  解释：通过资源的id删除 | | | |

* + - 1. kubectl logs

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | logs | 功能 |  |
| 命令形式 | kubectl logs [-f] [-p] POD [-c CONTAINER] [flags] | | |
| 参数解析 | | | |
| *pod：pod的id*  *[-c，--container]：pod里面运行的container名字*  *[-f，--follow]：是否以流式输出*  *[--interactive]：默认false，交互式*  *[-p，--previous]：输出以前的container的日志* | | | |
| 样例解析 | | | |
| 例子：kubectl logs 123456-7890 ruby-container  解释：  例子：kubectl logs -p 123456-7890 ruby-container  解释：  例子：kubectl logs -f 123456-7890 ruby-container  解释： | | | |

* + - 1. kubectl namespace

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | namespace | 功能 |  |
| 命令形式 | kubectl namespace [namespace] [flags] | | |
| 参数解析 | | | |
|  | | | |
| 样例解析 | | | |
| 例子：  解释： | | | |

* + - 1. kubectl config

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | config | 功能 |  |
| 命令形式 | kubectl config SUBCOMMAND [flags] | | |
| 参数解析 | | | |
| *view:查看一个config*  *--flatten：false*  *--merge：true*  *--minify：false*  *--no-headers：false*  *-o，--output："" 输出格式json|yaml|templatefile|wide*  *--raw：false*  *-t，--template："" template文件，和-o组合，格式为templatefile使用*  *set-cluster:设置cluster entry*  *--api-version*  *--certificate-authority*  *--embed-certs*  *--insecure-skip-tls-verify*  *--server*  *set-credentials:设置user entry*  *--client-certificate*  *--client-key*  *--embed-certs*  *--password*  *--token*  *--username*  *set-context:设置context entry*  *--cluster*  *--namespace 使用的namespace*  *--user*  *set:在kubeconfig文件中设置某个值，后跟一个dns风格的name和value值*  *unset:取消某个值*  *use-context:设置当前的context*    *[--kubeconfig=""]使用某个特定的kubeconfig*    *[可选项2]* | | | |
| 样例解析 | | | |
| 例子：kubectl config view  解释：查看当前的context  例子：kubectl config set-context dev --namespace=development  解释：设置一个dev环境，使用development namespace  例子：kubectl config use-context dev  解释：切换到dev环境，你将看不到其他namespace的resource  例子：kubectl config set contexts.default.namespace default  解释：设置default context使用的namespace为default | | | |

* + - 1. kubectl rolling-update

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | rolling-update | 功能 |  |
| 命令形式 | kubectl rolling-update OLD\_CONTROLLER\_NAME ([NEW\_CONTROLLER\_NAME] --image=NEW\_CONTAINER\_IMAGE | -f NEW\_CONTROLLER\_SPEC) [flags] | | |
| 参数解析 | | | |
| *old\_controller\_name：需要升级的controller的名称*  *new\_controller\_image：升级后的pod运行的镜像*  new\_controller\_spec：升级后的controller的json文件（或者其他支持格式）  （要么指定名称加镜像，要么指定用于升级的json文件）  *[new\_controller\_name]：升级后的controller名称*  *[--deployment-label-key]：默认"deployment"，用于区分两个不同的controller的key，只有指定--images时有效*  *[--dry-run]：生成需要改变的地方，并不实际运行*  *[--image]：用于升级的镜像，不能喝-f同用*  *[--no-headers]：不显示标题栏*  *[-o, --output]：输出格式*  *[--output-version]：输出使用的version*  *[--poll-interval]：update之后间隔时间再拉状态*  *[--rollback]：默认false，终止并回滚*  *[-t, --template]：不解释，同上*  *[--timeout]：update的时间上限*  *[--update-period]：更新pod之间的等待时间* | | | |
| 样例解析 | | | |
| 例子：  解释： | | | |

* + - 1. kubectl scale

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | scale | 功能 |  |
| 命令形式 | kubectl scale [--resource-version=version] [--current-replicas=count] --replicas=COUNT RESOURCE NAME [flags] | | |
| 参数解析 | | | |
| *--replicas：需要扩展到的数量*  *resource：资源类型*  *name：资源实例名称*  *[--current-replicas]：前置条件，满足当前副本数量为该值时才执行*  *[--resource-version]：前置条件，同上* | | | |
| 样例解析 | | | |
| 例子：kubectl scale --replicas=3 replicationcontrollers foo  解释：  例子：kubectl scale --current-replicas=2 --replicas=3 replicationcontrollers foo  解释： | | | |

* + - 1. kubectl exec

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | exec | 功能 | 在容器内执行命令 |
| 命令形式 | kubectl exec POD -c CONTAINER -- COMMAND [args...] [flags] | | |
| 参数解析 | | | |
| *POD*  *Pod的名称*  *CONTAINER*  *Pod内container的名称*  *COMMAND*  *要执行的命令*  *[args]*  *Command的参数*  *[flags]*  *分为局部参数和全局，局部参数主要有*  *-c --container="" 指定container*  *-p --pod="" 指定pod*  *-i --stdin=false 传递stdin到container*  *-t --tty=false stdin是否为tty* | | | |
| 样例解析 | | | |
| 例子：kubectl exec nginx date  解释：在pod nginx里面的第一个容器里运行date指令  例子：kubectl exec nginx -c nginx-client date  解释：在pod nginx里面的nginx-client容器里运行date指令  例子：kubectl exec nginx -c nginx-client -i -t -- bash -il  解释：在pod nginx里面的nginx-client容器进行指令交互，后面有参数要用--分开容器内参数和kubectl参数 | | | |

* + - 1. kubectl port-forward

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | port-forward | 功能 |  |
| 命令形式 | kubectl port-forward -p POD [LOCAL\_PORT:]REMOTE\_PORT [...[LOCAL\_PORT\_N:]REMOTE\_PORT\_N] [flags] | | |
| 参数解析 | | | |
| *POD：pod的名称*  *remote\_port：需要重定向到pod的端口*  *[local\_port]：本地需要暴露的端口* | | | |
| 样例解析 | | | |
| 例子：kubectl port-forward -p mypod 5000 6000  解释：重定向本地5000到pod5000，本地6000到pod6000  例子：kubectl port-forward -p mypod 8888:5000  解释：重定向本地8888到pod5000  例子：kubectl port-forward -p mypod :5000  解释：本地随机端口到pod5000  例子：kubectl port-forward -p mypod 0:5000  解释：本地随机端口到pod5000 | | | |

* + - 1. kubectl run

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | run | 功能 | 生成一个rc用参数填充然后运行 |
| 命令形式 | kubectl run NAME --image=image [--port=port] [--replicas=replicas] [--dry-run=bool] [--overrides=inline-json] [flags] | | |
| 参数解析 | | | |
| *name：新生成的rc的名称，也是pod的前缀*  *image：使用的镜像*  *[--dry-run]：*  *[--generator]：使用默认值*  *[--hostport]：暴露的host端口*  *[-l, --labels]：pod的label*  *[--no-headers]：*  *[-o, --output]：*  *[--output-version]：*  *[--overrides]：用于覆盖默认生成的json文件中的相关部分，为json格式*  *[--port]：暴露的container端口*  *[-r, --replicas]：副本数*  *[-t, --template]：* | | | |
| 样例解析 | | | |
| 例子：kubectl run nginx --image=nginx  解释：  例子：kubectl run nginx --image=nginx --replicas=5  解释：  例子：kubectl run nginx --image=nginx --dry-run  解释：  例子：kubectl run nginx --image=nginx --overrides='{ "apiVersion": "v1", "spec": { ... } }'  解释： | | | |

* + - 1. kubectl stop

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | stop | 功能 |  |
| 命令形式 | namespace [<namespace>] | | |
| 参数解析 | | | |
| *必须项*    *[可选项1]*    *[可选项2]* | | | |
| 样例解析 | | | |
| 例子：  解释： | | | |

* + - 1. kubectl expose

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | expose | 功能 | **为rc创建服务** |
| 命令形式 | kubectl expose RESOURCE NAME --port=port [--protocol=TCP|UDP] [--target-port=number-or-name] [--name=name] [--public-ip=ip] [--type=type] | | |
| 参数解析 | | | |
| *resource：资源类型，通常为rc或者service，pod也可以*  *name：资源名称，rc为需要暴露的rc的名称，service为service的名称*  *--port：实际监听的端口*    *[--protocol]：后端支持的协议类型，tcp或者udp*  *[--target-port]：要暴露的service的port，用户访问的port*  *[--name]：新生成service的名称*  *[--public-ip]：分配给service的ip，将会替代默认生成的ip*  *[--type]：service的类型，分为clusterIP、nodePort和LoadBalancer*  *[--create-external-load-balancer]：默认false，创建一个外部loadBanlancer*  *[--dry-run]：默认false，只是打印生成的json或者其他格式文件，并不会创建，和-o选项配合使用能获得详细的service配置信息*  *[--generator]：默认service/v2，采用哪个版本的api来生成service，service/v1 port那么默认为default*  *[-l，--labels]：生成的service的labels*  *[--name]：生成的service的name*  *[--no-headers]：默认false，如果没有-o，设置为true将不会打印最上面的那一排标题*  *[-o，--output]：输出的格式，json、yaml、template、templatefile、wide*  *[--output-version]：用指定版本的version输出生成的object*  *[--overrides]：用一个内嵌的json覆盖生成的object的json*  *[--selector]：用于新生成的service的selector*  *[-t，--template]：当指定-o template的时候，指定template string或者templatefile path* | | | |
| 样例解析 | | | |
| 例子：kubectl expose rc nginx --port=80 --target-port=8000  解释：  例子：kubectl expose service nginx --port=443 --target-port=8443 --name=nginx-https  解释：  例子：kubectl expose rc streamer --port=4100 --protocol=udp --name=video-stream  解释： | | | |

* + - 1. kubectl label

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | label | 功能 |  |
| 命令形式 | kubectl label [--overwrite] RESOURCE NAME KEY\_1=VAL\_1 ... KEY\_N=VAL\_N [--resource-version=version] [flags] | | |
| 参数解析 | | | |
| *resource：资源类型*  *name：资源的实例名称*    *[--all]：针对该namespace的所有指定类型的资源*  *[--no-headers]：不显示前面的标题栏*  *[-o，--output]：指定输出格式，同上*  *[--output-version]：输出采用的apiversion*  *[--overwrite]：默认false，覆盖已经存在的label*  *[--resource-version]：当前resourceversion和指定的一致时，更新才生效，只对单个对象有效*  *[-l，--selector]：用于筛选的label*  *[-t，--template]：template string或者templatefile路径* | | | |
| 样例解析 | | | |
| 例子：kubectl label pods foo unhealthy=true  解释：  例子：kubectl label --overwrite pods foo status=unhealthy  解释：  例子：kubectl label pods --all status=unhealthy  解释：  例子：kubectl label pods foo status=unhealthy --resource-version=1  解释：  例子：kubectl label pods foo bar-  解释： | | | |

* + - 1. kubectl cluster-info

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | cluster-info | 功能 | **获取集群信息** |
| 命令形式 | Kubectl cluster-info | | |
| 参数解析 | | | |
| *必须项*    *[可选项1]*    *[可选项2]* | | | |
| 样例解析 | | | |
| 例子：  解释： | | | |

* + - 1. kubectl api-version

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | api-version | 功能 |  |
| 命令形式 | namespace [<namespace>] | | |
| 参数解析 | | | |
| *必须项*    *[可选项1]*    *[可选项2]* | | | |
| 样例解析 | | | |
| 例子：  解释： | | | |

* 1. 重要概念
     1. limitrange

limitrange用于限制pod和其中container可以使用的资源上下限，和某个namespace关联，和ResourceQuota配合对namespace进行资源限制。

* + - 1. json文件

{

"apiVersion": "v1",

"kind": "LimitRange",

"metadata": {

"name": "mylimits"

},

"spec": {

"limits": [

{

"max": {

"cpu": "2",

"memory": "1Gi"

},

"min": {

"cpu": "250m",

"memory": "6Mi"

},

"type": "Pod"

},

{

"default": {

"cpu": "250m",

"memory": "100Mi"

},

"max": {

"cpu": "2",

"memory": "1Gi"

},

"min": {

"cpu": "250m",

"memory": "6Mi"

},

"type": "Container"

}

]

}

}

分为pod和container两种，每一个分为最小、最大和默认值。

* + - 1. 使用

首先需要创建namespace

root@commonqa-master-cluster-1:~/fangying/resource/v1.0/limitRange# kubectl get namespace

NAME LABELS STATUS

default <none> Active

development name=development Active

production name=production Active

然后创建limitrange

root@commonqa-master-cluster-1:~/fangying/resource/v1.0/limitRange# kubectl create -f limits.json --namespace=development

limitranges/mylimits

查看一下创建出来的limitrange

root@commonqa-master-cluster-1:~/fangying/resource/v1.0/limitRange# kubectl describe limits mylimits --namespace=development

Name: mylimits

Namespace: development

Type Resource Min Max Default

---- -------- --- --- ---

Pod cpu 250m 2 -

Pod memory 6Mi 1Gi -

Container memory 6Mi 1Gi 100Mi

Container cpu 250m 2 250m

创建一个pod使用刚创建的limitrange

{

"apiVersion": "v1",

"kind": "Pod",

"metadata": {

"labels": {

"name": "valid-pod"

},

"name": "valid-pod"

},

"spec": {

"containers": [

{

"image": "gcr.io/google\_containers/serve\_hostname",

"name": "kubernetes-serve-hostname",

"resources": {

"limits": {

"cpu": "1",

"memory": "512Mi"

}

}

}

]

}

}

上面的1个cpu和512M内存位于针对container的限制范围之内，所以创建会成功，如果pod不指定resources.limits，那么默认的250m cpu和2G内存将会被采用。如果超过指定的大小，创建会失败。如果没有创建limits，pod也没有指定使用的资源，请求也会被拒绝。

* + 1. pod健康检测
       1. json文件

分为两种类型的健康检测，都需要在pod的相关位置加入配置。

执行命令检测：

{

"apiVersion": "v1",

"kind": "Pod",

"metadata": {

"labels": {

"test": "liveness"

},

"name": "liveness-exec"

},

"spec": {

"containers": [

{

"args": [

"/bin/sh",

"-c",

"echo ok > /tmp/health; sleep 10; rm -rf /tmp/health; sleep 600"

],

"image": "gcr.io/google\_containers/busybox",

"livenessProbe": {

"exec": {

"command": [

"cat",

"/tmp/health"

]

},

"initialDelaySeconds": 15,

"timeoutSeconds": 1

},

"name": "liveness"

}

]

}

}

http检测：

{

"apiVersion": "v1",

"kind": "Pod",

"metadata": {

"labels": {

"test": "liveness"

},

"name": "liveness-http"

},

"spec": {

"containers": [

{

"args": [

"/server"

],

"image": "gcr.io/google\_containers/liveness",

"livenessProbe": {

"httpGet": {

"path": "/healthz",

"port": 8080

},

"initialDelaySeconds": 15,

"timeoutSeconds": 1

},

"name": "liveness"

}

]

}

}

* + 1. persistent volume

在pod里面要使用存储，有多种资源可以使用，共享宿主机的目录、挂载一块本地盘或者云盘。要使用存储首先需要声明存储PersistentVolume，然后描述需求PersistentVolumeClaim，后台会根据需求绑定存储资源到pod

* + - 1. json文件

声明持有的存储资源，这个种类比较多，可以是host目录或者某个云商的存储，host：

{

"apiVersion": "v1",

"kind": "PersistentVolume",

"metadata": {

"labels": {

"type": "local"

},

"name": "pv0002"

},

"spec": {

"accessModes": [

"ReadWriteOnce"

],

"capacity": {

"storage": "8Gi"

},

"hostPath": {

"path": "/tmp/data02"

},

"persistentVolumeReclaimPolicy": "Recycle"

}

}

gce：

{

"apiVersion": "v1",

"kind": "PersistentVolume",

"metadata": {

"name": "pv0003"

},

"spec": {

"accessModes": [

"ReadWriteOnce",

"ReadOnlyMany"

],

"capacity": {

"storage": "10Gi"

},

"gcePersistentDisk": {

"fsType": "ext4",

"pdName": "abc123"

}

}

}

使用的时候先声明需求

{

"kind": "PersistentVolumeClaim",

"apiVersion": "v1",

"metadata": {

"name": "myclaim-3"

}, "spec": {

"accessModes": [

"ReadWriteOnce",

"ReadOnlyMany"

],

"resources": {

"requests": {

"storage": "10G"

}

}

}

}

* + - 1. 使用

使用的时候引用申明的需求

{

"apiVersion": "v1",

"kind": "Pod",

"metadata": {

"labels": {

"name": "frontendhttp"

},

"name": "mypod"

},

"spec": {

"containers": [

{

"image": "nginx",

"name": "myfrontend",

"ports": [

{

"containerPort": 80,

"name": "http-server"

}

],

"volumeMounts": [

{

"mountPath": "/var/www/html",

"name": "mypd"

}

]

}

],

"volumes": [

{

"name": "mypd",

"persistentVolumeClaim": {

"claimName": "myclaim-3"

}

}

]

}

}

* + 1. ResourceQuota

ResourceQuota用于对一个namespace限制总的资源使用量，包括cpu、memory、pod数、service数量、secret数量、存储使用量、rc数量

* + - 1. json文件

{

"apiVersion": "v1",

"kind": "ResourceQuota",

"metadata": {

"name": "quota"

},

"spec": {

"hard": {

"cpu": "20",

"memory": "1Gi",

"persistentvolumeclaims": "10",

"pods": "10",

"replicationcontrollers": "20",

"resourcequotas": "1",

"secrets": "10",

"services": "5"

}

}

}

* + - 1. 使用

对一个namespace应用ResourceQuota

root@commonqa-master-cluster-1:~/fangying/resource/v1.0/resourcequota# kubectl create -f quota.json --namespace=development

resourcequotas/quota

root@commonqa-master-cluster-1:~/fangying/resource/v1.0/resourcequota# kubectl describe quota quota --namespace=development

Name: quota

Namespace: development

Resource Used Hard

-------- ---- ----

cpu 0 20

memory 0 1Gi

persistentvolumeclaims 0 10

pods 0 10

replicationcontrollers 0 20

resourcequotas 1 1

secrets 0 10

services 0 5

* + 1. secret

secret用于提供pod比较私密的数据

* + - 1. json文件

{

"apiVersion": "v1",

"data": {

"data-1": "dmFsdWUtMQ0K",

"data-2": "dmFsdWUtMg0KDQo="

},

"kind": "Secret",

"metadata": {

"name": "test-secret"

},

# Opaque是默认类型，kubernetes.io/service-account-token 是service account访问api的token

"type":"Opaque"

}

正确使用secret的pod json

{

"apiVersion": "v1",

"kind": "Pod",

"metadata": {

"name": "secret-test-pod"

},

"spec": {

"containers": [

{

"command": [

"/mt",

"--file\_content=/etc/secret-volume/data-1"

],

"image": "kubernetes/mounttest:0.1",

"name": "test-container",

"volumeMounts": [

{

"mountPath": "/etc/secret-volume",

"name": "secret-volume"

}

]

}

],

"restartPolicy": "Never",

"volumes": [

{

"name": "secret-volume",

"secret": {

"secretName": "test-secret"

}

}

]

}

}

secret会放到容器内部，是通过一个volume挂载进去，所以要声明volume的挂载点和volume的类型为secret

* + - 1. 使用

首先创建一个secret

root@commonqa-master-cluster-1:~/fangying/resource/v1.0/secret# kubectl create -f secret.json --namespace=development

secrets/test-secret

查看secret的详情

root@commonqa-master-cluster-1:~/fangying/resource/v1.0/secret# kubectl describe secret test-secret --namespace=development

Name: test-secret

Namespace: development

Labels: <none>

Annotations: <none>

Type: Opaque

Data

====

data-2: 11 bytes

data-1: 9 bytes

创建pod然后查看

root@commonqa-master-cluster-1:~/fangying/resource/v1.0/secret# kubectl get pod --namespace=development

NAME READY STATUS RESTARTS AGE

secret-test-pod 0/1 ExitCode:0 0 5m

* 1. json模板
  2. 场景分析

# kubernetes 机制详解

本章主要详细描述各部件的工作原理。

* 1. 结构图

整个系统的结构图如图xxx。

* 1. apiserver
  2. controller
  3. scheduler

调度的时候将会考虑5种情况：

1、检查是否有端口冲突

2、检查资源是否足够

3、检查存储是否冲突

4、检查nodeSelector的值

5、检查是否指定了hostname

计算优先级时有三步计算：

1、根据node的资源利用率

2、相同服务的pod在同一个node上的数量

3、公平计算

* 1. kubelet

# kubernetes 开发

本章节描述paas基于kubernetes开发所需信息。

* 1. events
     1. pod events
        1. create pod

root@commonqa-etcd1:~# kubectl get ev -o json

{

"kind": "List",

"apiVersion": "v1",

"metadata": {},

"items": [

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1a91003c946e",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1a91003c946e",

"uid": "78c9adf5-447c-11e5-a196-fa163ec2098d",

"resourceVersion": "1561308",

"creationTimestamp": "2015-08-17T01:08:29Z",

"deletionTimestamp": "2015-08-17T02:08:31Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561262"

},

"reason": "scheduled",

"message": "Successfully assigned wordpress to 10.180.164.46",

"source": {

"component": "scheduler"

},

"firstTimestamp": "2015-08-17T01:08:35Z",

"lastTimestamp": "2015-08-17T01:08:35Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1a9dc2d28f5b",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1a9dc2d28f5b",

"uid": "a0e240b2-447c-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1561352",

"creationTimestamp": "2015-08-17T01:09:37Z",

"deletionTimestamp": "2015-08-17T02:09:38Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "implicitly required container POD"

},

"reason": "pulled",

"message": "Pod container image \"registry.hz.netease.com/kubernetes/pause:0.8.0\" already present on machine",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:09:30Z",

"lastTimestamp": "2015-08-17T01:09:30Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1a9dd361f235",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1a9dd361f235",

"uid": "9d5e5b20-447c-11e5-a196-fa163ec2098d",

"resourceVersion": "1561353",

"creationTimestamp": "2015-08-17T01:09:31Z",

"deletionTimestamp": "2015-08-17T02:09:33Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "implicitly required container POD"

},

"reason": "created",

"message": "Created with docker id 034d91e90545",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:09:31Z",

"lastTimestamp": "2015-08-17T01:09:31Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1a9de2471b31",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1a9de2471b31",

"uid": "a132b58b-447c-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1561354",

"creationTimestamp": "2015-08-17T01:09:37Z",

"deletionTimestamp": "2015-08-17T02:09:39Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "implicitly required container POD"

},

"reason": "started",

"message": "Started with docker id 034d91e90545",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:09:31Z",

"lastTimestamp": "2015-08-17T01:09:31Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1a9e063a898d",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1a9e063a898d",

"uid": "9de08924-447c-11e5-a196-fa163ec2098d",

"resourceVersion": "1561356",

"creationTimestamp": "2015-08-17T01:09:32Z",

"deletionTimestamp": "2015-08-17T02:09:34Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "spec.containers{wordpress}"

},

"reason": "created",

"message": "Created with docker id 3a4ff835d313",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:09:31Z",

"lastTimestamp": "2015-08-17T01:09:31Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1a9e0e15818f",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1a9e0e15818f",

"uid": "9df499b9-447c-11e5-a196-fa163ec2098d",

"resourceVersion": "1561357",

"creationTimestamp": "2015-08-17T01:09:32Z",

"deletionTimestamp": "2015-08-17T02:09:34Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "spec.containers{wordpress}"

},

"reason": "started",

"message": "Started with docker id 3a4ff835d313",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:09:32Z",

"lastTimestamp": "2015-08-17T01:09:32Z",

"count": 1

}

]

}

* + - 1. 停掉其中业务container

会重新拉起一个container，产生如下event

{

"kind": "List",

"apiVersion": "v1",

"metadata": {},

"items": [

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1af1dbf7125f",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1af1dbf7125f",

"uid": "747eb5af-447d-11e5-a196-fa163ec2098d",

"resourceVersion": "1561619",

"creationTimestamp": "2015-08-17T01:15:32Z",

"deletionTimestamp": "2015-08-17T02:15:34Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "spec.containers{wordpress}"

},

"reason": "created",

"message": "Created with docker id 7d08615ccfff",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:15:31Z",

"lastTimestamp": "2015-08-17T01:15:31Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1af1e39e94fb",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1af1e39e94fb",

"uid": "7847d57f-447d-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1561621",

"creationTimestamp": "2015-08-17T01:15:38Z",

"deletionTimestamp": "2015-08-17T02:15:40Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "spec.containers{wordpress}"

},

"reason": "started",

"message": "Started with docker id 7d08615ccfff",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:15:32Z",

"lastTimestamp": "2015-08-17T01:15:32Z",

"count": 1

}

]

}

* + - 1. 停掉pause container

会导致整个pod container都会被杀掉，所有pod包含的container会被重新创建，产生如下event

{

"kind": "List",

"apiVersion": "v1",

"metadata": {},

"items": [

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1a9dc2d28f5b",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1a9dc2d28f5b",

"uid": "bff38d22-447d-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1561713",

"creationTimestamp": "2015-08-17T01:17:38Z",

"deletionTimestamp": "2015-08-17T02:17:40Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "implicitly required container POD"

},

"reason": "pulled",

"message": "Pod container image \"registry.hz.netease.com/kubernetes/pause:0.8.0\" already present on machine",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:09:30Z",

"lastTimestamp": "2015-08-17T01:17:32Z",

"count": 2

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1b0de093a113",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1b0de093a113",

"uid": "bfe6fe07-447d-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1561712",

"creationTimestamp": "2015-08-17T01:17:38Z",

"deletionTimestamp": "2015-08-17T02:17:40Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "spec.containers{wordpress}"

},

"reason": "killing",

"message": "Killing with docker id 7d08615ccfff",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:17:32Z",

"lastTimestamp": "2015-08-17T01:17:32Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1b0dee75056f",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1b0dee75056f",

"uid": "bc5fe7c4-447d-11e5-a196-fa163ec2098d",

"resourceVersion": "1561714",

"creationTimestamp": "2015-08-17T01:17:32Z",

"deletionTimestamp": "2015-08-17T02:17:34Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "implicitly required container POD"

},

"reason": "created",

"message": "Created with docker id 95c971e86b52",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:17:32Z",

"lastTimestamp": "2015-08-17T01:17:32Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1b0dfed84255",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1b0dfed84255",

"uid": "c0347aed-447d-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1561715",

"creationTimestamp": "2015-08-17T01:17:39Z",

"deletionTimestamp": "2015-08-17T02:17:40Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "implicitly required container POD"

},

"reason": "started",

"message": "Started with docker id 95c971e86b52",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:17:32Z",

"lastTimestamp": "2015-08-17T01:17:32Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1b0e11a4fc39",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1b0e11a4fc39",

"uid": "bcb63734-447d-11e5-a196-fa163ec2098d",

"resourceVersion": "1561716",

"creationTimestamp": "2015-08-17T01:17:33Z",

"deletionTimestamp": "2015-08-17T02:17:35Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "spec.containers{wordpress}"

},

"reason": "created",

"message": "Created with docker id e9748882012a",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:17:33Z",

"lastTimestamp": "2015-08-17T01:17:33Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1b0e17a8086f",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1b0e17a8086f",

"uid": "c07403ca-447d-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1561717",

"creationTimestamp": "2015-08-17T01:17:39Z",

"deletionTimestamp": "2015-08-17T02:17:41Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "spec.containers{wordpress}"

},

"reason": "started",

"message": "Started with docker id e9748882012a",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:17:33Z",

"lastTimestamp": "2015-08-17T01:17:33Z",

"count": 1

}

]

}

* + - 1. 直接删除pod

pod会被删除，产生如下event

{

"kind": "List",

"apiVersion": "v1",

"metadata": {},

"items": [

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1b3bba66ad8b",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1b3bba66ad8b",

"uid": "3199814a-447e-11e5-a196-fa163ec2098d",

"resourceVersion": "1561865",

"creationTimestamp": "2015-08-17T01:20:49Z",

"deletionTimestamp": "2015-08-17T02:20:51Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "spec.containers{wordpress}"

},

"reason": "killing",

"message": "Killing with docker id e9748882012a",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:20:49Z",

"lastTimestamp": "2015-08-17T01:20:49Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1b3bcb6f1e3d",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1b3bcb6f1e3d",

"uid": "3573a2e6-447e-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1561866",

"creationTimestamp": "2015-08-17T01:20:55Z",

"deletionTimestamp": "2015-08-17T02:20:57Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "5457bad9-447c-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561307",

"fieldPath": "implicitly required container POD"

},

"reason": "killing",

"message": "Killing with docker id 95c971e86b52",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T01:20:49Z",

"lastTimestamp": "2015-08-17T01:20:49Z",

"count": 1

}

]

}

* + - 1. 如果node和master网络中断

如果在master没有感知到的时候恢复，这个时间可能接近5分钟，网络恢复后不会有影响，如果master感知到之后，将会删除etcd的数据。此时node上的docker还存在，一旦网络恢复，kubelet将会删除node上的docker，产生如下event

超时恢复

{

"kind": "List",

"apiVersion": "v1",

"metadata": {},

"items": [

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "10.180.164.46.13f7ce9cbea1a487",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/10.180.164.46.13f7ce9cbea1a487",

"uid": "b7779272-4483-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1563657",

"creationTimestamp": "2015-08-17T02:00:21Z",

"deletionTimestamp": "2015-08-17T03:00:23Z"

},

"involvedObject": {

"kind": "Node",

"name": "10.180.164.46",

"uid": "10.180.164.46"

},

"reason": "NodeReady",

"message": "Node 10.180.164.46 status is now: NodeReady",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-06T07:22:58Z",

"lastTimestamp": "2015-08-17T02:00:14Z",

"count": 8

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "10.180.164.46.13fa65d5829e0015",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/10.180.164.46.13fa65d5829e0015",

"uid": "b04f2004-4480-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1562629",

"creationTimestamp": "2015-08-17T01:38:40Z",

"deletionTimestamp": "2015-08-17T02:38:42Z"

},

"involvedObject": {

"kind": "Node",

"name": "10.180.164.46",

"uid": "10.180.164.46"

},

"reason": "NodeNotReady",

"message": "Node 10.180.164.46 status is now: NodeNotReady",

"source": {

"component": "controllermanager"

},

"firstTimestamp": "2015-08-14T17:56:38Z",

"lastTimestamp": "2015-08-17T01:38:40Z",

"count": 2

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1d64e40f06ad",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1d64e40f06ad",

"uid": "bd60ffcc-4483-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1563665",

"creationTimestamp": "2015-08-17T02:00:31Z",

"deletionTimestamp": "2015-08-17T03:00:33Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "6e125743-447e-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561981",

"fieldPath": "spec.containers{wordpress}"

},

"reason": "killing",

"message": "Killing with docker id 960ede219bff",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T02:00:25Z",

"lastTimestamp": "2015-08-17T02:00:25Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress.13fb1d64f219276f",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/wordpress.13fb1d64f219276f",

"uid": "b9d589b2-4483-11e5-a196-fa163ec2098d",

"resourceVersion": "1563668",

"creationTimestamp": "2015-08-17T02:00:25Z",

"deletionTimestamp": "2015-08-17T03:00:27Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "default",

"name": "wordpress",

"uid": "6e125743-447e-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1561981",

"fieldPath": "implicitly required container POD"

},

"reason": "killing",

"message": "Killing with docker id 5f18e7edf56b",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T02:00:25Z",

"lastTimestamp": "2015-08-17T02:00:25Z",

"count": 1

}

]

}

* + - 1. node节点挂掉||直接杀掉kubelet进程

1. kubelet进程快速重启

Mon, 17 Aug 2015 10:32:42 +0800 Mon, 17 Aug 2015 10:32:42 +0800 1 10.180.164.46 Node starting {kubelet 10.180.164.46} Starting kubelet.

Mon, 17 Aug 2015 10:32:43 +0800 Mon, 17 Aug 2015 10:32:43 +0800 1 10.180.164.46 Node NodeReady {kubelet 10.180.164.46} Node 10.180.164.46 status is now: NodeReady

1. kubelet进程超时重启

如果时间较长，master同样会清掉etcd里面的pod数据，此时node节点的docker仍然在运行，一旦重启kubelet进程，kubelet将会干掉node上的docker，只会产生node重启的event，在整个过程中，node下线会被补记（但是这都啥时候了，补记也就只能看看了），pod被干掉都不会产生event

* + 1. rc event
       1. create rc

一个rc一个pod

概要

FIRSTSEEN LASTSEEN COUNT NAME KIND SUBOBJECT REASON SOURCE MESSAGE

Mon, 17 Aug 2015 11:23:20 +0800 Mon, 17 Aug 2015 11:23:20 +0800 1 wordpress-ctrl1 ReplicationController successfulCreate {replication-controller } Created pod: wordpress-ctrl1-n0fir

Mon, 17 Aug 2015 11:23:50 +0800 Mon, 17 Aug 2015 11:23:50 +0800 1 wordpress-ctrl1-n0fir Pod scheduled {scheduler } Successfully assigned wordpress-ctrl1-n0fir to 10.180.164.46

Mon, 17 Aug 2015 11:24:51 +0800 Mon, 17 Aug 2015 11:24:51 +0800 1 wordpress-ctrl1-n0fir Pod implicitly required container POD pulled {kubelet 10.180.164.46} Pod container image "registry.hz.netease.com/kubernetes/pause:0.8.0" already present on machine

Mon, 17 Aug 2015 11:24:51 +0800 Mon, 17 Aug 2015 11:24:51 +0800 1 wordpress-ctrl1-n0fir Pod implicitly required container POD created {kubelet 10.180.164.46} Created with docker id 5c0a6c5211d4

Mon, 17 Aug 2015 11:24:51 +0800 Mon, 17 Aug 2015 11:24:51 +0800 1 wordpress-ctrl1-n0fir Pod implicitly required container POD started {kubelet 10.180.164.46} Started with docker id 5c0a6c5211d4

Mon, 17 Aug 2015 11:24:52 +0800 Mon, 17 Aug 2015 11:24:52 +0800 1 wordpress-ctrl1-n0fir Pod spec.containers{wordpress} created {kubelet 10.180.164.46} Created with docker id 5b1af5e34050

Mon, 17 Aug 2015 11:24:52 +0800 Mon, 17 Aug 2015 11:24:52 +0800 1 wordpress-ctrl1-n0fir Pod spec.containers{wordpress} started {kubelet 10.180.164.46} Started with docker id 5b1af5e34050

events细节

{

"kind": "List",

"apiVersion": "v1",

"metadata": {},

"items": [

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1-n0fir.13fb21f2409a500e",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1-n0fir.13fb21f2409a500e",

"uid": "60f3f976-448f-11e5-a196-fa163ec2098d",

"resourceVersion": "1568827",

"creationTimestamp": "2015-08-17T03:23:50Z",

"deletionTimestamp": "2015-08-17T04:23:52Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "development",

"name": "wordpress-ctrl1-n0fir",

"uid": "52e27da0-448f-11e5-9ee7-fa163ecc02f1",

"apiVersion": "v1",

"resourceVersion": "1568795"

},

"reason": "scheduled",

"message": "Successfully assigned wordpress-ctrl1-n0fir to 10.180.164.46",

"source": {

"component": "scheduler"

},

"firstTimestamp": "2015-08-17T03:23:50Z",

"lastTimestamp": "2015-08-17T03:23:50Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1-n0fir.13fb220072050ee5",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1-n0fir.13fb220072050ee5",

"uid": "855807c5-448f-11e5-a196-fa163ec2098d",

"resourceVersion": "1568878",

"creationTimestamp": "2015-08-17T03:24:51Z",

"deletionTimestamp": "2015-08-17T04:24:53Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "development",

"name": "wordpress-ctrl1-n0fir",

"uid": "52e27da0-448f-11e5-9ee7-fa163ecc02f1",

"apiVersion": "v1",

"resourceVersion": "1568824",

"fieldPath": "implicitly required container POD"

},

"reason": "pulled",

"message": "Pod container image \"registry.hz.netease.com/kubernetes/pause:0.8.0\" already present on machine",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T03:24:51Z",

"lastTimestamp": "2015-08-17T03:24:51Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1-n0fir.13fb2200817bc085",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1-n0fir.13fb2200817bc085",

"uid": "89310884-448f-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1568879",

"creationTimestamp": "2015-08-17T03:24:57Z",

"deletionTimestamp": "2015-08-17T04:24:59Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "development",

"name": "wordpress-ctrl1-n0fir",

"uid": "52e27da0-448f-11e5-9ee7-fa163ecc02f1",

"apiVersion": "v1",

"resourceVersion": "1568824",

"fieldPath": "implicitly required container POD"

},

"reason": "created",

"message": "Created with docker id 5c0a6c5211d4",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T03:24:51Z",

"lastTimestamp": "2015-08-17T03:24:51Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1-n0fir.13fb220092543167",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1-n0fir.13fb220092543167",

"uid": "85aab6b1-448f-11e5-a196-fa163ec2098d",

"resourceVersion": "1568881",

"creationTimestamp": "2015-08-17T03:24:51Z",

"deletionTimestamp": "2015-08-17T04:24:53Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "development",

"name": "wordpress-ctrl1-n0fir",

"uid": "52e27da0-448f-11e5-9ee7-fa163ecc02f1",

"apiVersion": "v1",

"resourceVersion": "1568824",

"fieldPath": "implicitly required container POD"

},

"reason": "started",

"message": "Started with docker id 5c0a6c5211d4",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T03:24:51Z",

"lastTimestamp": "2015-08-17T03:24:51Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1-n0fir.13fb2200a91331cd",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1-n0fir.13fb2200a91331cd",

"uid": "89c17871-448f-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1568883",

"creationTimestamp": "2015-08-17T03:24:58Z",

"deletionTimestamp": "2015-08-17T04:25:00Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "development",

"name": "wordpress-ctrl1-n0fir",

"uid": "52e27da0-448f-11e5-9ee7-fa163ecc02f1",

"apiVersion": "v1",

"resourceVersion": "1568824",

"fieldPath": "spec.containers{wordpress}"

},

"reason": "created",

"message": "Created with docker id 5b1af5e34050",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T03:24:52Z",

"lastTimestamp": "2015-08-17T03:24:52Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1-n0fir.13fb2200b1026bab",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1-n0fir.13fb2200b1026bab",

"uid": "86438aae-448f-11e5-a196-fa163ec2098d",

"resourceVersion": "1568884",

"creationTimestamp": "2015-08-17T03:24:52Z",

"deletionTimestamp": "2015-08-17T04:24:54Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "development",

"name": "wordpress-ctrl1-n0fir",

"uid": "52e27da0-448f-11e5-9ee7-fa163ecc02f1",

"apiVersion": "v1",

"resourceVersion": "1568824",

"fieldPath": "spec.containers{wordpress}"

},

"reason": "started",

"message": "Started with docker id 5b1af5e34050",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T03:24:52Z",

"lastTimestamp": "2015-08-17T03:24:52Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1.13fb21eb682a20c0",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1.13fb21eb682a20c0",

"uid": "4f6da57d-448f-11e5-a196-fa163ec2098d",

"resourceVersion": "1568797",

"creationTimestamp": "2015-08-17T03:23:20Z",

"deletionTimestamp": "2015-08-17T04:23:22Z"

},

"involvedObject": {

"kind": "ReplicationController",

"namespace": "development",

"name": "wordpress-ctrl1",

"uid": "4475e086-448f-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1568778"

},

"reason": "successfulCreate",

"message": "Created pod: wordpress-ctrl1-n0fir",

"source": {

"component": "replication-controller"

},

"firstTimestamp": "2015-08-17T03:23:20Z",

"lastTimestamp": "2015-08-17T03:23:20Z",

"count": 1

}

]

}

* + - 1. 删除rc||stop rc

{

"kind": "List",

"apiVersion": "v1",

"metadata": {},

"items": [

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1-n0fir.13fb2250f2b0fa13",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1-n0fir.13fb2250f2b0fa13",

"uid": "571f792c-4490-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1569208",

"creationTimestamp": "2015-08-17T03:30:43Z",

"deletionTimestamp": "2015-08-17T04:30:44Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "development",

"name": "wordpress-ctrl1-n0fir",

"uid": "52e27da0-448f-11e5-9ee7-fa163ecc02f1",

"apiVersion": "v1",

"resourceVersion": "1568824",

"fieldPath": "spec.containers{wordpress}"

},

"reason": "killing",

"message": "Killing with docker id 5b1af5e34050",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T03:30:36Z",

"lastTimestamp": "2015-08-17T03:30:36Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1-n0fir.13fb225101f451e1",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1-n0fir.13fb225101f451e1",

"uid": "53951cd8-4490-11e5-a196-fa163ec2098d",

"resourceVersion": "1569209",

"creationTimestamp": "2015-08-17T03:30:37Z",

"deletionTimestamp": "2015-08-17T04:30:39Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "development",

"name": "wordpress-ctrl1-n0fir",

"uid": "52e27da0-448f-11e5-9ee7-fa163ecc02f1",

"apiVersion": "v1",

"resourceVersion": "1568824",

"fieldPath": "implicitly required container POD"

},

"reason": "killing",

"message": "Killing with docker id 5c0a6c5211d4",

"source": {

"component": "kubelet",

"host": "10.180.164.46"

},

"firstTimestamp": "2015-08-17T03:30:37Z",

"lastTimestamp": "2015-08-17T03:30:37Z",

"count": 1

}

]

}

* + - 1. 直接删除pod

是删除pod和创建一个新的pod的event集合

* + - 1. kubelet失联

node失联时不会有event产生，但是当node恢复后会补记一条notReady事件（有毛用），pod被干掉没有event产生

概要

FIRSTSEEN LASTSEEN COUNT NAME KIND SUBOBJECT REASON SOURCE MESSAGE

Mon, 17 Aug 2015 12:00:03 +0800 Mon, 17 Aug 2015 12:00:03 +0800 1 10.180.164.46 Node NodeNotReady {controllermanager } Node 10.180.164.46 status is now: NodeNotReady

Mon, 17 Aug 2015 12:15:53 +0800 Mon, 17 Aug 2015 12:15:53 +0800 3 10.180.164.46 Node NodeReady {kubelet 10.180.164.46} Node 10.180.164.46 status is now: NodeReady

Mon, 17 Aug 2015 12:15:53 +0800 Mon, 17 Aug 2015 12:15:53 +0800 1 10.180.164.46 Node starting {kubelet 10.180.164.46} Starting kubelet.

Mon, 17 Aug 2015 12:05:48 +0800 Mon, 17 Aug 2015 12:05:48 +0800 1 wordpress-ctrl1 ReplicationController successfulCreate {replication-controller } Created pod: wordpress-ctrl1-rps4r

Mon, 17 Aug 2015 12:06:49 +0800 Mon, 17 Aug 2015 12:06:49 +0800 1 wordpress-ctrl1-rps4r Pod scheduled {scheduler } Successfully assigned wordpress-ctrl1-rps4r to 10.180.164.37

Mon, 17 Aug 2015 12:08:51 +0800 Mon, 17 Aug 2015 12:08:51 +0800 1 wordpress-ctrl1-rps4r Pod implicitly required container POD created {kubelet 10.180.164.37} Created with docker id b5bf56060a88

Mon, 17 Aug 2015 12:08:51 +0800 Mon, 17 Aug 2015 12:08:51 +0800 1 wordpress-ctrl1-rps4r Pod implicitly required container POD started {kubelet 10.180.164.37} Started with docker id b5bf56060a88

Mon, 17 Aug 2015 12:08:51 +0800 Mon, 17 Aug 2015 12:08:51 +0800 1 wordpress-ctrl1-rps4r Pod implicitly required container POD pulled {kubelet 10.180.164.37} Successfully pulled Pod container image "registry.hz.netease.com/kubernetes/pause:0.8.0"

Mon, 17 Aug 2015 12:10:36 +0800 Mon, 17 Aug 2015 12:10:36 +0800 1 wordpress-ctrl1-rps4r Pod spec.containers{wordpress} pulled {kubelet 10.180.164.37} Successfully pulled image "tutum/wordpress"

Mon, 17 Aug 2015 12:10:37 +0800 Mon, 17 Aug 2015 12:10:37 +0800 1 wordpress-ctrl1-rps4r Pod spec.containers{wordpress} created {kubelet 10.180.164.37} Created with docker id fc96aba9ee11

Mon, 17 Aug 2015 12:10:37 +0800 Mon, 17 Aug 2015 12:10:37 +0800 1 wordpress-ctrl1-rps4r Pod spec.containers{wordpress} started {kubelet 10.180.164.37} Started with docker id fc96aba9ee11

详细

{

"kind": "List",

"apiVersion": "v1",

"metadata": {},

"items": [

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1-rps4r.13fb244abe4462ab",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1-rps4r.13fb244abe4462ab",

"uid": "65e7adc0-4495-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1571243",

"creationTimestamp": "2015-08-17T04:06:55Z",

"deletionTimestamp": "2015-08-17T05:06:57Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "development",

"name": "wordpress-ctrl1-rps4r",

"uid": "4160161b-4495-11e5-9ee7-fa163ecc02f1",

"apiVersion": "v1",

"resourceVersion": "1571176"

},

"reason": "scheduled",

"message": "Successfully assigned wordpress-ctrl1-rps4r to 10.180.164.37",

"source": {

"component": "scheduler"

},

"firstTimestamp": "2015-08-17T04:06:49Z",

"lastTimestamp": "2015-08-17T04:06:49Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1-rps4r.13fb24672b992cdb",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1-rps4r.13fb24672b992cdb",

"uid": "ab29c5fd-4495-11e5-a196-fa163ec2098d",

"resourceVersion": "1571358",

"creationTimestamp": "2015-08-17T04:08:51Z",

"deletionTimestamp": "2015-08-17T05:08:53Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "development",

"name": "wordpress-ctrl1-rps4r",

"uid": "4160161b-4495-11e5-9ee7-fa163ecc02f1",

"apiVersion": "v1",

"resourceVersion": "1571242",

"fieldPath": "implicitly required container POD"

},

"reason": "pulled",

"message": "Successfully pulled Pod container image \"registry.hz.netease.com/kubernetes/pause:0.8.0\"",

"source": {

"component": "kubelet",

"host": "10.180.164.37"

},

"firstTimestamp": "2015-08-17T04:08:51Z",

"lastTimestamp": "2015-08-17T04:08:51Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1-rps4r.13fb246735514058",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1-rps4r.13fb246735514058",

"uid": "aef4e683-4495-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1571359",

"creationTimestamp": "2015-08-17T04:08:58Z",

"deletionTimestamp": "2015-08-17T05:08:59Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "development",

"name": "wordpress-ctrl1-rps4r",

"uid": "4160161b-4495-11e5-9ee7-fa163ecc02f1",

"apiVersion": "v1",

"resourceVersion": "1571242",

"fieldPath": "implicitly required container POD"

},

"reason": "created",

"message": "Created with docker id b5bf56060a88",

"source": {

"component": "kubelet",

"host": "10.180.164.37"

},

"firstTimestamp": "2015-08-17T04:08:51Z",

"lastTimestamp": "2015-08-17T04:08:51Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1-rps4r.13fb2467414959d5",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1-rps4r.13fb2467414959d5",

"uid": "af139007-4495-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1571360",

"creationTimestamp": "2015-08-17T04:08:58Z",

"deletionTimestamp": "2015-08-17T05:08:59Z"

},

"involvedObject": {

"kind": "Pod",

"namespace": "development",

"name": "wordpress-ctrl1-rps4r",

"uid": "4160161b-4495-11e5-9ee7-fa163ecc02f1",

"apiVersion": "v1",

"resourceVersion": "1571242",

"fieldPath": "implicitly required container POD"

},

"reason": "started",

"message": "Started with docker id b5bf56060a88",

"source": {

"component": "kubelet",

"host": "10.180.164.37"

},

"firstTimestamp": "2015-08-17T04:08:51Z",

"lastTimestamp": "2015-08-17T04:08:51Z",

"count": 1

},

{

"kind": "Event",

"apiVersion": "v1",

"metadata": {

"name": "wordpress-ctrl1.13fb243c7f38de02",

"namespace": "development",

"selfLink": "/api/v1/namespaces/development/events/wordpress-ctrl1.13fb243c7f38de02",

"uid": "416f4450-4495-11e5-9ee7-fa163ecc02f1",

"resourceVersion": "1571178",

"creationTimestamp": "2015-08-17T04:05:54Z",

"deletionTimestamp": "2015-08-17T05:05:56Z"

},

"involvedObject": {

"kind": "ReplicationController",

"namespace": "development",

"name": "wordpress-ctrl1",

"uid": "585e4d73-4490-11e5-a196-fa163ec2098d",

"apiVersion": "v1",

"resourceVersion": "1570260"

},

"reason": "successfulCreate",

"message": "Created pod: wordpress-ctrl1-rps4r",

"source": {

"component": "replication-controller"

},

"firstTimestamp": "2015-08-17T04:05:48Z",

"lastTimestamp": "2015-08-17T04:05:48Z",

"count": 1

}

]

}

* 1. xxx

# kubernetes性能测试

# kubernetes问题