kubernetes 配置调优部分

DaoCloud Enterprise

Exported on Aug 23, 2019

Table of Contents

1 相比社区的配置 DCE 主要做了什么修改 3

2 DCE 的 kubernetes 相关配置怎么查看和修改？ 4

2.1 查看方法 4

2.2 修改方法 6

3 配置介绍 7

3.1 组件名称：kube-apiserver 7

3.2 组件名称: kubelet 8

3.3 组件名称: kube-proxy 10

3.4 组件名称: kube-scheduler 11

3.5 背景 11

3.6 原理 11

3.6.1 设置组件的高可用性 11

3.6.2 Leader 状态信息 12

3.6.3 Leader 竞争过程 13

3.7 组件部署 14

4 社区参考 15

5 历史记录 16

# 相比社区的配置 DCE 主要做了什么修改

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K8S 组建本身 | kubelet  apiserver  controller manager  scheduler | kubelet 部分修复了 cgroup 泄漏问题。 | 可以进行定制化配置 docker exec -it dce\_kube\_controller\_1 cat /dce/supervisord\_conf.yaml  K8S 配置，支持 RBAC，配置event限制。 |  |
| 资源管理 | QoS | 配置资源限制和预留 | 超配比例限制方法：CPU 支持超配；内存不超配，防止 OOM kill。 | [kustomize](https://dwiki.daocloud.io/download/attachments/33558227/kustomize%20%E5%88%9D%E4%BD%93%E9%AA%8C.key?api=v2&modificationDate=1556275117743&version=1)  CRD |

DCE 架构图： [DCE架构&数据流](https://dwiki.daocloud.io/pages/viewpage.action?pageId=26746539) 包括详细的组件关系。

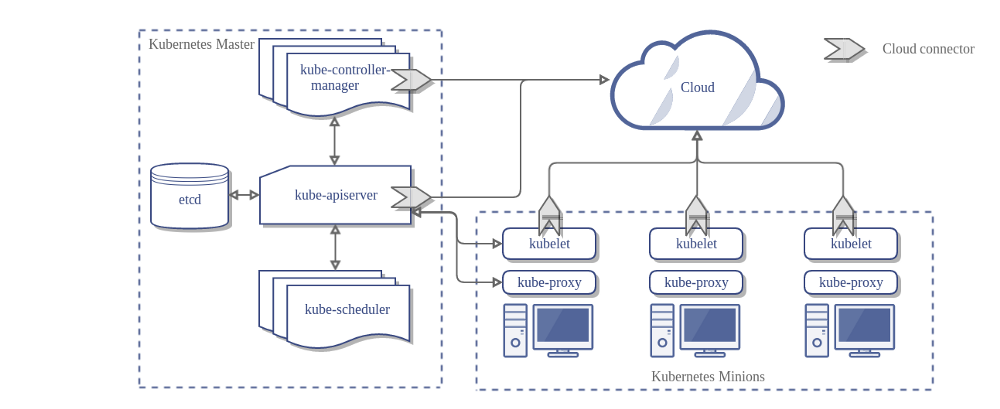


Figure 1 Pre CCM Kube Arch

<https://kubernetes.io/docs/concepts/architecture/cloud-controller/>

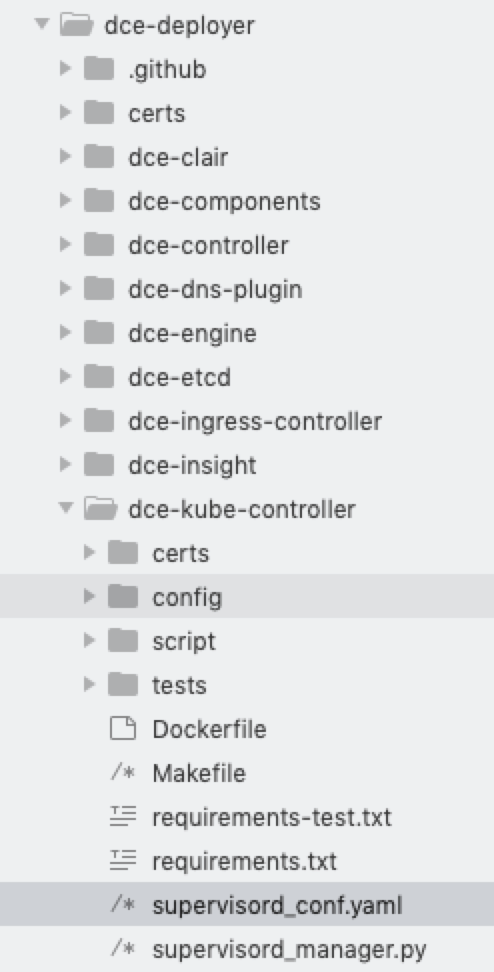
# DCE 的 kubernetes 相关配置怎么查看和修改？

## 查看方法

DCE 控制节点

* docker exec -it dce\_kube\_controller\_1 cat /etc/supervisord.conf

DCE 代码：

* dce-deployer/dce-kube-controller
* 
* 如下主要配置是 证书、端口，

|  |
| --- |
| common:  deamon\_options:  autorestart: true  startsecs: 0  stderr\_logfile: /dev/stderr  stderr\_logfile\_maxbytes: 0  stdout\_logfile: /dev/stdout  stdout\_logfile\_maxbytes: 0    kube-apiserver:  command: /usr/local/bin/kube-apiserver  command\_options\_required:  - --etcd-servers  - --advertise-address  - --service-cluster-ip-range  - --service-node-port-range  command\_options\_immutable:  -v: 2  --allow-privileged: true  --etcd-cafile: /etc/ssl/private/client/ca.pem  --etcd-certfile: /etc/ssl/private/client/client-cert.pem  --etcd-keyfile: /etc/ssl/private/client/client-key.pem  --insecure-port: 8080  --insecure-bind-address: 0.0.0.0  --secure-port: 16443  --bind-address: 0.0.0.0  --service-account-key-file: /etc/ssl/private/kube-controller/ca-key.pem  --tls-private-key-file: /etc/ssl/private/kube-controller/peer-key.pem  --tls-cert-file: /etc/ssl/private/kube-controller/peer-cert.pem  --client-ca-file: /etc/ssl/private/kube-controller/ca.pem  --kubelet-client-certificate: /etc/ssl/private/kubelet/server.crt  --kubelet-client-key: /etc/ssl/private/kubelet/server.key  --requestheader-client-ca-file: /etc/ssl/private/kube-front-proxy/front-proxy-ca.pem  --proxy-client-cert-file: /etc/ssl/private/client/front-proxy-client-cert.pem  --proxy-client-key-file: /etc/ssl/private/client/front-proxy-client-key.pem  --authorization-mode: RBAC  --feature-gates: ExperimentalCriticalPodAnnotation=true  command\_options\_default:  --kubelet-preferred-address-types: InternalIP,ExternalIP,Hostname  --runtime-config: batch/v2alpha1=true  --admission-control: NamespaceLifecycle,LimitRanger,ServiceAccount,PersistentVolumeLabel,DefaultStorageClass,ResourceQuota,DefaultTolerationSeconds  --requestheader-username-headers: X-Remote-User  --requestheader-group-headers: X-Remote-Group  --requestheader-extra-headers-prefix: X-Remote-Extra  --requestheader-allowed-names: dce-front-proxy-client      kube-controller-manager:  command: /usr/local/bin/kube-controller-manager  command\_options\_required:  - --cluster-cidr  command\_options\_immutable:  -v: 2  --address: 0.0.0.0  --leader-elect: true  --master: http://127.0.0.1:8080  --use-service-account-credentials: true  --root-ca-file: /etc/ssl/private/kube-controller/ca.pem  --service-account-private-key-file: /etc/ssl/private/kube-controller/ca-key.pem  --cluster-signing-key-file: /etc/ssl/private/kube-controller/ca-key.pem  --cluster-signing-cert-file: /etc/ssl/private/kube-controller/ca.pem  --feature-gates: ExperimentalCriticalPodAnnotation=true  command\_options\_default:  --controllers: "\*,bootstrapsigner,tokencleaner"  --allocate-node-cidrs: true  --horizontal-pod-autoscaler-use-rest-clients: true  --node-monitor-grace-period: 40s  --node-monitor-period: 5s  --pod-eviction-timeout: 5m0s  --terminated-pod-gc-threshold: 1000      kube-scheduler:  command: /usr/local/bin/kube-scheduler  command\_options\_immutable:  -v: 2  --address: 0.0.0.0  --leader-elect: true  --master: http://127.0.0.1:8080  --feature-gates: ExperimentalCriticalPodAnnotation=true      kube-dashboard:  command: /opt/kube-dashboard/dashboard  command\_options\_immutable:  -v: 2  --apiserver-host: http://127.0.0.1:8080  --insecure-bind-address: 0.0.0.0  --insecure-port: 9090  # deamon\_options: |

## 修改方法

1. docker exec -it dce\_kube\_controller\_1 bash 进入容器   
2. vi /dce/supervisord\_conf.yaml 修改 --node-monitor-grace-period   
3. exit 退出容器   
4. docker restart dce\_kube\_controller\_1 重启容器

# 配置介绍

## 组件名称：kube-apiserver

|  |  |  |
| --- | --- | --- |
| 组件参数 | 功能描述 | 使用建议 |
| –max-requests-inflight | "–max-requests-inflight"设置了apiserver在一定时间内并行处理non-mutating request的最大数量，当数量超过这个数值时，超过部分的request会被拒绝处理。当设置为0时，表示apiserver没有单位时间内请求数量的限制，该参数使用400作为默认值。该参数可以很好地控制kube-apiserver的内存消耗，而API server在处理大量request时对CPU并没有很高要求。 | 当该参数值过低时，系统会发生大量request-limit-exceed错误。当该参数值过高时，kube-apiserver会因试图并行处理过多request而内存不够（OOM）发生故障。总体而言，25～30个Pod并行处理15个请求是足够的。 |
| –max-mutating-requests-inflight | “–max-mutating-requests-inflight"设置了apiserver在一定时间内并行处理mutating request的最大数量，当数量超过这个数值时，超过部分的request会被拒绝处理。当设置为0时，表示apiserver没有单位时间内请求数量的限制，该参数使用200作为默认值。该参数旨在限制单位时间内处理mutating requests的最大数量。因为mutating request通常需要比non-mutating request处理更多的业务逻辑，所以mutating request需要消耗更多内存资源，”–max-mutating-requests-inflight"的值也要比"–max-requests-inflight"的值更低。性能测试在决定特定业务场景中"–max-requests-inflight"和"–max-mutating-requests-inflight"两个值的比例时是有必要的。根据社区中的部分反馈，1比3是一个合理的方案，参考[47899](https://github.com/kubernetes/kubernetes/issues/47899), [38093](https://github.com/kubernetes/kubernetes/pull/38093)。 |  |
| –enable-garbage-collector | “–enable-garbage-collector"设置了是否开启garbage collector，该选项必须和kube-controller-manager的”–enable-garbage-collector"参数设置一致，该参数使用true作为默认值。 | Kubernetes的garbage collector是用来删除曾经有owner，但是现在没有owner的对象的。举个例子，一个ReplicaSet是一组Pod的owner，当这个ReplicaSet被删除后，Kubernetes的garbage collector负责处理掉这个ReplicaSet控制下的Pod。如果这些Pod在业务中需要被保留，则"–enable-garbage-collector"应被设置为false，其他情况则应设置为true。 |
| –target-ram-mb | "–target-ram-mb"设置了apiserver的内存限制。 | 根据这篇 [性能测试](http://blog.kubernetes.io/2016/03/1000-nodes-and-beyond-updates-to-Kubernetes-performance-and-scalability-in-12.html)，32C120G可以运行2000个Node和60000个Pod，相当于60M／Node和30Pod／Node。通常而言，每20～30个Pod使用60M是比较合理的。 |
| –enable-logs-handler | "–enable-logs-handler"设置了是否为apiserver安装logs handler，该参数使用true作为默认值。 | 如果不处理apiserver的log是有必要的，则将该参数设置为false。其他情况下，设置为true或者使用默认值。 |

## 组件名称: kubelet

|  |  |  |
| --- | --- | --- |
| 组件参数 | 默认值 | 说明 |
| –experimental-allocatable-ignore-eviction | false | （1）设置为true时，当计算节点可分配资源总量时，**–eviction-hard参数将被忽略**。此时，节点可分配资源总量 [Allocatable] = [Node Capacity] - [Kube-Reserved] - [System-Reserved]。（2）设置为false时，当计算节点可分配资源总量时，在原先的基础上还要减去–eviction-hard参数所设置的值。此时，节点可分配资源总量 [Allocatable] = [Node Capacity] - [Kube-Reserved] - [System-Reserved] - [Hard-Eviction-Threshold]。（&）[Node Capacity] 表示节点的实际资源量；[Kube-Reserved] 表示为Kubernetes系统组件预留的资源量，可通过–kube-reserved参数设置；[System-Reserved] 表示为系统预留的资源量，可通过–system-reserved参数设置。  [😢 eviction policy 驱逐策略 debug 分析 (kubernetes, kubelet)](https://dwiki.daocloud.io/pages/viewpage.action?pageId=13065808)  [为什么镜像或容器会被频繁删除](https://dwiki.daocloud.io/pages/viewpage.action?pageId=18485144) |
| –cgroup-root | ’ ’ , 表示将使用容器运行时的默认值 | 为pods设置的root cgroup。 |
| –experimental-mounter-path | ’ ’，空 | （1）设置为空时，表示使用默认的挂载命令。（2）设置为绝对路径时，表示使用指定路径上的挂载命令。同时，其会将--expermental-check-node-capabilities-before-mount参数重置为false，因此，当执行挂载操作时，并不会去检查节点是否具备挂载此种类型数据卷的必要组件，如二进制可执行文件。 |
| –experimental-check-node-capabilities-before-mount | false | (1) 设置为false时，在执行挂载操作前，不会去检查本节点是否具有挂载某种类型数据卷的必要组件，如二进制可执行文件等等。当挂载操作失败之后，会重新再执行此挂载操作。（2）设置为true时，在执行挂载操作前，会根据要挂载的数据卷类型，确定数据卷插件，并检查本节点具有此数据卷插件的必要的依赖性组件。当检查结果是不具备时，此挂载操作立即失败，对于此挂载操作，不会进行重试。（&）--expermental-mounter-path参数取值，会影响本参数的最终取值。 |
| –enable-debugging-handlers | true | 设置为true表示提供远程访问本节点容器的日志、进入容器执行命令等相关的REST服务。在安全性方面，还需后序调研。 |
| –eviction-hard | memory.available<100Mi,nodefs.available<10%,nodes.inodesFree<5% | 表示触发Pod Eviction操作的一组硬门限设置。当节点资源达至这个下限时，出于节点稳定性的考虑，kubelet会**立刻**执行Pod Eviction操作，释放资源，维护节点的稳定性。 |
| –feature-gates |  | （1）ExperimentalCriticalPodAnnotation 默认值为false；设置为true时，表示当CriticalPod被调度器调度到本节点后，如果因为节点资源不足而无法运行时，kubelet会启用重调度机制，按qos等级删除一部分pod，释放资源来使CriticalPod可以在本节点上运行。另外，CriticalPod不会被kubelet的Pod Eviction机制驱逐出去。  为什么要配置？详情参考 [Events 巨大的时候 Contrller Crash 研究报告](https://dwiki.daocloud.io/pages/viewpage.action?pageId=26744750) |

## 组件名称: kube-proxy

|  |  |  |
| --- | --- | --- |
| 组件参数 | 默认值 | 说明 |
| –iptables-min-sync-period | 0 | 表示iptables规则的最小同步周期。（1）设置为非0时，kube-proxy使用令牌桶算法实现流控控制，避免频繁刷新。例如，当--iptables-min-sync-period=10s, kube-proxy以每秒0.1个令牌的速度，向令牌桶中放入令牌，桶的容量为2。（2）设置为0时，kube-proxy则不启用流控控制。 |
| –iptables-sync-period | 30s | 表示iptables规则的最大同步周期。 |
| —feature-gates= ExperimentalCriticalPodAnnotation=true |  | 关于ExperimentalCriticalPodAnnotation=true这个参数，kube-proxy中并没有在使用。如果确实想用CriticalPod，为保险起见，最好还是设置上。 |

## 组件名称: kube-scheduler

|  |  |  |  |
| --- | --- | --- | --- |
| 组件参数 | 参数作用 | 使用建议 | 理由 |
| –feature-gates=ExperimentalCriticalPodAnnotation=true | ExperimentalCriticalPodAnnotation 是 alpha: v1.5 的参数，确保[保证调度](https://kubernetes.io/docs/tasks/administer-cluster/guaranteed-scheduling-critical-addon-pods/)的 Pod 被标记（annotation）为scheduler.alpha.kubernetes.io/critical-pod，并且防止被 Node 干掉。这个特性对 QOS 为 BestEffort 的 Pod无效 | 可以使用 | 根据需求开启 alpha 特性 |

kube-controller-manager 与 kube-scheduler 是保证集群正常运行和合理调度的重要组件。

## 背景

kube-controller-manager 作为集群内部的管理中心，负责集群内的Node、Pod 副本、服务端点（Endpoint）、命名空间（Namespace）、服务账号（ServiceAccount）、资源定额（ResourceQuota）等的管理，当某个Node意外挂掉时，kube-controller-manager 会及时发现此故障并执行自动化修复流程，确保集群始终处于预期的工作状态。  
kube-scheduler 在整个系统中承担了"承上启下"的重要功能，“承上” 是指它负责接收kube-controller-manager 创建的新Pod，为其安排一个落脚的"家"——目录Node；“启下” 是指安置工作完成之后，目标Node上的kubelet服务进程接管后继工作，负责Pod生命周期中的"下半生"。kube-scheduler 的作用是将待调度的Pod按照特定的调度算法和调度策略绑定到集群中的某个结合的Node上，并将绑定信息写入etcd中。  
本文就 kube-controller-manager 与 kube-scheduler 组件在 kubernetes 集群的作用进行了简单的说明，但并不会就其工作原理的细节进行深入研究。本文的重点在于，从源码层次阐述 kube-controller-manager 与 kube-scheduler 组件的高可用的实现原理，以及为部署 kube-controller-manager 与 kube-scheduler 组件提供一些参考性建议。

## 原理

在进入正题之前，首先需要声明的一点是， kube-controller-manager 与 kube-scheduler 组件实现高可用所使用的机制完全相同，对其提供接口参数完全一样。因此，完全没有必要同时讲解kube-controller-manager 与 kube-scheduler 。下面，仅以 kube-controller-manager 为例，进行深入讲解。同时，并作约定性说明，讲解中所涉及到集群操作均是基于kargo部署的。

### 设置组件的高可用性

通过设置 kube-controller-manager 组件的启动参数--leader-elect=true，即可实现 kube-controller-manager 组件的高可用性。

# 截选自 /etc/kubernetes/manifests/kube-controller-manager.manifest 文件  
 command:  
 - /hyperkube  
 - controller-manager  
 - --master=http://127.0.0.1:8080  
 - --leader-elect=true  
 - --service-account-private-key-file=/etc/kubernetes/ssl/apiserver-key.pem  
 - --root-ca-file=/etc/kubernetes/ssl/ca.pem  
 - --cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem  
 - --cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem  
 - --enable-hostpath-provisioner=false  
 - --node-monitor-grace-period=40s  
 - --node-monitor-period=5s  
 - --pod-eviction-timeout=5m0s  
 - --v=2

除--leader-elect参数外，kube-controller-manager 还提供了下述参数，供集群管理员对 kube-controller-manager 高可用机制，进行更加细粒化的调整。

|  |  |
| --- | --- |
| 参数名与默认值 | 参数说明 |
| --leader-elect-lease-duration=15s | leading 的有效期 |
| --leader-elect-renew-deadline=10s | 在停止 leading 角色之前，再次申请延长 leading 有效期这个动作的时间间隔。 |
| --leader-elect-retry-period=2s | 申请成为 leading 角色的时间间隔 |
| --leader-elect-resource-lock=endpoints | 成为 leading 角色的凭据 |

通常，情况下不需要更改上述配置。另一个，需要注意的一点是，kubernetes/pkg/leaderelection/leaderelection.go 文件中给出的信息。

|  |
| --- |
| DISCLAIMER: this is an alpha API. This library will likely change significantly or even be removed entirely in subsequent releases. Depend on this API at your own risk. |

### Leader 状态信息

在具有n个kube-controller-manager副本的kuberentes集群中，如何判定当前哪一个kube-controller-manager 副本是正处于 leader 状态呢？所有的 kube-controller-manager 副本都会以一种在一定范围内随机时间间隔，周期性的向 kube-apiserver 请求获取当前的leader的状态信息，并将此信息保存在本地。

kubectl -n kube-system describe ep kube-controller-manager

通过上面的shell命令，可以查看 kubernetes 集群中 当前 leader 的状态信息。

Name: kube-controller-manager  
Namespace: kube-system  
Labels: <none>  
Annotations: control-plane.alpha.kubernetes.io/leader={"holderIdentity":"192-168-1-215","leaseDurationSeconds":15,"acquireTime":"2017-06-07T18:55:01Z","renewTime":"2  
017-06-08T14:14:48Z","leaderTransitions":0}  
Subsets:  
Events: <none>

control-plane.alpha.kubernetes.io/leader 中存放着当前 kube-controller-manager 的leader状态信息。

|  |  |
| --- | --- |
| 状态信息参数 | 参数说明 |
| holderIdentity | kube-controller-manager 副本的唯一标识，它是kube-controller-manager 副本的hostname，绝大数情况下与podname相同。 |
| leaseDurationSeconds | kube-controller-manager 的启动参数 --leader-elect-lease-duration 一致 |
| acquireTime | kube-controller-manager 副本成为 leader的初始时刻 |
| renewTime | 正处于leader状态的kube-controller-manager 副本申请延长leader有效时间这个动作的时刻 |
| leaderTransitions | 发生leader状态改变的次数，从0开始计数 |

Leader的状态信息在 etcd 集群当中。通过下述命令即可查看。

export ETCDCTL\_API=3  
etcdctl --endpoints=https://192.168.1.215:2379,https://192.168.1.122:2379,https://192.168.2.220:2379 --cert=/etc/ssl/etcd/ssl/node-node1.pem --k  
ey=/etc/ssl/etcd/ssl/node-node1-key.pem --cacert=/etc/ssl/etcd/ssl/ca.pem get /registry/services/endpoints/kube-system/kube-controller-manager

请根据实际情况，修改etcdctl 的相应参数。

### Leader 竞争过程

在讲解 Leader选举过程之前，读者需要首先掌握[lease lock](http://www.cnblogs.com/lijiawei/archive/2013/01/17/2865422.html)的相关知识。kubernetes/pkg/leaderelection包内利用lease lock相关概念实现了Leader的选择过程。

所有的 kube-controller-manager 副本，在开始正式工作之前，都会通过竞争，尝试使自己成为leader。这是因为只有取得leader身份的 kube-controller-manager 副本，才有资格开始工作，而那些没有取得leader身份的副本，只能像傻瓜一样不停地去尝试使自己成为leader，而留给傻瓜们的机会却并不多。

副本之间的竞争规则其实简单，用一种通俗的方式来讲就是，只有勤劳且勇敢的副本才会被持续授予以Leader的身份。

勇敢是副本成为Leader的前提条件。所谓勇敢是指，你必须是所有副本中第一个向 kube-apiserver 提出自己要当Leader要求的副本，否则，你就是懒惰的，是不能leader的。但是，纵使副本成功申请到leader身份，并不意味着一劳永逸。这是因为Leader的身份是有期限的。

勤劳是决定副本持续成为Leader的前提条件。所谓勤劳是指，当副本已经成为leader时，在这个身份失效前或者在其他副本成为Leader前，副本必须向kube-apiserver发送延长Leader身份有效期的请求。一旦请求失败，副本最终只能死亡，从而开始了一轮新的竞争过程，这也就是上面我所讲到的傻瓜们的机会。

## 组件部署

关于实现 kube-controller-manager 与 kube-scheduler 组件的高可用需要注意的地方有两点：一是设置"–leader-elect=true"，二是要保证同一组件的不同副本之间的hostname（podname）的唯一性。  
self-hosted kubernetes 集群可以很容易地对kube-controller-manager 与 kube-scheduler 组件弹性伸缩。因为 这两个组件的部署方式是deployment。

# 社区参考

<https://github.com/kubernetes-sigs/kubespray/blob/7cf8ad4dc732f5f09ee59a6b1d9b8c620a972b9d/roles/kubernetes/master/defaults/main/main.yml>

<https://github.com/kubernetes-sigs/kubespray/blob/f6a63d88a7fac7c7229750a14358e3cf1b575984/roles/kubernetes/master/templates/kubeadm-config.v1beta2.yaml.j2>

[[调研] 各个 kubernetes 解决方案里 kube 组件的参数使用](https://dwiki.daocloud.io/pages/viewpage.action?pageId=12003608)

# 历史记录

[3.0.1-26483和3.0.5-27697 版本配置对比](https://dwiki.daocloud.io/pages/viewpage.action?pageId=40545104)

**配置差异对比**

1.DAEMON\_ARGS=-v=4 改为 DAMON-ARGS=-v=2

3.0.0-3.0.2 kubelt 日志级别默认为 4， debug 是误开。  
hotfix 3.0.2（2018-10-25） <https://dwiki.daocloud.io/pages/viewpage.action?pageId=19827593>  
3.0.2 hotfix 和 3.0.3之后的版本，日志级别恢复到info， 也就是 -v 2.  
3.0.3 同期升级 kubelet 从 1.10.5 到 1.10.11。

调整日志级别原因  
日志设置为 debug 有可能会塞满磁盘，引发驱逐。在企业客户，引发过一个小事故。因此进行了这次日志级别的调整。

2.增加--feature-gates ExperimentalCriticalPodAnnotation=true --anonymous-auth=false --client-ca-file=/etc/daocloud/dce/certs/ca.crt

详情参考：[浦发 DCE 3.0.1 升级到 3.0.5 遇到的 Prometheus 证书问题](https://dwiki.daocloud.io/pages/viewpage.action?pageId=36680375)

端口差异对比：仅 端口增加证书认证，其他未改变**ExperimentalCriticalPodAnnotation**