前言

在优化Kubernetes编排的CockroachDB集群之前，你需要：

* 阅读[Kubernetes部署](http://doc.cockroachchina.baidu.com/#deploy/orchestrated-deployment/kubernetes-single-cluster-deployment/)文档，熟悉Kubernetes相关内容以及CockroachDB在上面部署的大致流程。
* 验证CockroachDB集群在相同硬件、不使用Kubernetes编排系统的情况下满足业务负载的需求，你可能发现[调整负载](https://www.cockroachlabs.com/docs/stable/performance-best-practices-overview.html)或是使用[不同的机器配置](http://doc.cockroachchina.baidu.com/#deploy/production-checklist/#hardware)就能够满足需求。比起后期针对Kubernetes花费时间进行调优，前期适度调整可能是更好的选择。

性能因子

在Kebernetes上运行CockroachDB，有很多独立的因素都可能会影响性能。有些性能因子比其他更重要或是更容易调优，可以针对性地选择合适的方案。如果在Kubernetes中已有一个正在运行的CockroachDB集群，并且需要在运行的同时进行调整，则可能需要额外的工作 - 维护和测试。

以下部分将展示如何修改先前提供的Kubernetes配置用的YAML文件里的参数。可以在Github下载最新的版本，一个用于[安全模式下运行CockroachDB](https://github.com/cockroachdb/cockroach/blob/master/cloud/kubernetes/cockroachdb-statefulset-secure.yaml)，一个用于[非安全模式下运行CockroachDB](https://github.com/cockroachdb/cockroach/blob/master/cloud/kubernetes/cockroachdb-statefulset.yaml)。

CockroachDB版本

考虑到CockroachDB处于开发进程中，持续发布新的版本，且每个版本都能带来性能上的提升，我们强烈推荐用户尝试使用最新版本并及时升级系统的CockroachDB数据库版本。

客户端负载

客户端负载是影响数据库性能的最重要的因素，尝试根据[SQL performance best practices](https://www.cockroachlabs.com/docs/stable/performance-best-practices-overview.html)文档去优化负载，加速应用。

机器质量

提升单个机器的能力能够直接地获得更高的性能，更加具体的建议可以详见[hardware recommandations](http://doc.cockroachchina.baidu.com/#deploy/production-checklist/#hardware)。使用更多CPU的机器能够很好地提升系统的整体吞吐量。此外，由于Kubernetes在集群的每台机器上运行一组进程，因此比起使用更多的小机器，使用少数的大机器往往能获得更高的收益。

磁盘类型

CockroachDB会使用大量磁盘空间，因此更快的磁盘设备很容易提升集群的性能。先前提供的配置文件未指定磁盘类型，大多数情况下Kubernetes会自动配置默认磁盘类型。在公有云环境（AWS、GCP、Azure）中，这意味着将使用未针对数据库工作负载进行优化的慢速磁盘（例如，GCE上使用HDDs，AWS上使用未配置IOPS的SSDs）。然而，我们强烈[推荐使用SSDs](http://doc.cockroachchina.baidu.com/#deploy/production-checklist/#hardware)以获取最好的性能。

创建不同的磁盘类型

Kubernetes通过其[StorageClass API对象](https://kubernetes.io/docs/concepts/storage/storage-classes/)，开放卷配置（volume provisioner）用的磁盘类型。每个云环境都有自己默认的StorageClass，用户可以很容易修改默认配置或在你可以获取卷的时候创建一个新的命名类。具体而言，选择[Kubernetes documentation]提供的卷配置类型，获取对应的YAML样本文件并修改为需要的卷配置类型，再执行kubectl create -f <your-storage-class-file.yaml>。

例如，针对Google Compute Engine或是Google Kubernetes Engine调整StorageClass使用pd-ssd硬盘类型，如下：

apiVersion: storage.k8s.io/v1

**kind**: StorageClass

metadata:

**name**: <your-ssd-**class**-**name**>

provisioner: kubernetes.io/gce-pd

parameters:

**type**: pd-ssd

用户如果选择使用新的硬盘类型，可以通过修改YAML文件的相应配置，或是修改该类型作为默认配置两种方式来设置。你可能还需要修改Kubernetes存储类的其他参数，例如针对AWS提供了IOPS卷类型的io1创建StorageClass，需要调整iopsPerGB参数。

修改CockroachDB使用的磁盘类型

用户可以修改YAML文件指定集群使用新的StorageClass，而不是默认的配置。具体到CockroachDB StatefulSet配置，用户需要是添加一行关于VolumeClaimTemplates的配置。

* 原配置：

volumeClaimTemplates:

- metadata:

name: datadir

spec:

accessModes:

- "ReadWriteOnce"

resources:

requests:

storage: 1Gi

* 新配置，在spec添加storageClassName属性：

volumeClaimTemplates:

- metadata:

name: datadir

spec:

accessModes:

- "ReadWriteOnce"

storageClassName: <your-ssd-class-name>

resources:

requests:

storage: 1Gi

使用kubectl create -f指定新YAML配置文件，Kubernetes则会使用新的StorageClass创建卷。

修改默认磁盘类型

如果需要将新StorgaeClass作为集群所有卷的默认配置，用户需要执行一系列的命令进行配置，需要获取新的StorageClass名字，再删除当前默认的配置，最后添加新名字作为默认配置。

kubectl get storageclasses

NAME PROVISIONER

ssd kubernetes.io/gce-pd

standard (default) kubernetes.io/gce-pd

kubectl patch storageclass standard -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'

storageclass "standard" patched

kubectl patch storageclass ssd -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'

storageclass "ssd" patched

请注意，如果运行的是旧版本的Kubernetes，则可能需要使用beta版本的类型，而不是上述的内容。具体地在Kubernetes v1.8版本，你需要使用的是storageclass.beta.kubernetes.io/is-default-class。用户需要执行kubectl describe storageclass来确认该Kubernetes版本支持的磁盘类型并进行适当调整。

磁盘大小

一些云提供商提供的磁盘（特别地包括GCP的所有磁盘，以及AWS的io1类型磁盘），其IOPS大小与磁盘的大小直接相关。这种情况下，增加磁盘大小可以显著地提高CockroachDB的性能，并降低磁盘写满的风险。

在创建CockroachDB集群之前修改CockroachDB YAML文件里的VolumeClaimTemplate属性，申请更多的空间。

* 原配置，为每个集群节点申请1GiB的磁盘空间：

volumeClaimTemplates:

- metadata:

name: datadir

spec:

accessModes:

- "ReadWriteOnce"

resources:

requests:

storage: 1Gi

* 新配置，为每个集群节点申请1TiB的磁盘空间：

volumeClaimTemplates:

- metadata:

name: datadir

spec:

accessModes:

- "ReadWriteOnce"

resources:

requests:

storage: 1024Gi

由于[GCE磁盘的IOPS大小与磁盘大小几乎线性相关](https://cloud.google.com/compute/docs/disks/performance#type_comparison)，1TiB的磁盘几乎是1GiB磁盘的1024倍IOPS性能，新配置下CockroachDB在面对磁盘瓶颈型负载时，性能将会有很大的性能提升。

本地磁盘

到目前为止，我们一直假定用户是使用默认配置的远程磁盘，在StatefulSet中运行CockroachDB。但是使用本地磁盘往往能获得比远程盘更好的性能，例如，在AWS上使用SSD实例存储卷而不是EBS卷；在GCE上使用本地SSDs而不是持久化磁盘。

StatefulSet以往一直不支持使用本地磁盘，但是[在Kubernetes v1.10版本中添加了使用“本地”持久卷的支持（beta版本）](https://kubernetes.io/docs/concepts/storage/volumes/#local)。我们不推荐在该功能成熟之前将其应用到实际业务，但它是一个值得关注的潜在性能增长点。

另一种可能是选择Daemonset运行CockroachDB而不是StatefulSet，详见：[Running in a DaemonSet](http://doc.cockroachchina.baidu.com/#deploy/orchestrated-deployment/kubernetes-performance-optimization/#running-in-a-daemonset)

值得注意的是，使用本地磁盘相比于使用云服务商提供的网络附加磁盘，将会面对磁盘故障所带来的更高风险，因为后者往往会在底层完成数据冗余的工作。因此，使用本地磁盘的时候，我们需要相应地将冗余副本数从默认的3调整到5，详见：[configure replication zones](http://doc.cockroachchina.baidu.com/#deploy/configure-replication-zones/)

资源请求和限制

当Kubernetes直接或间接通过其他资源类型（例如StatefulSet）运行pod时，用户可以为每个pod里的容器保留一定的CPU和/或内存，或者限制每个容器的CPU和/或内存。根据Kubernetes集群的使用方式，执行上述操作可能会产生不同的影响。详见[Kubernetes documentation](https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/)。

资源请求

资源请求允许用户为容器保留一定的CPU或内存资源。如果在CockroachDB YAML文件添加资源请求，Kubernetes会将每个CockroachDB pod安排到具有足够可用资源的节点上，并确保使用适当的Linux容器原语保证pod的资源使用。如果在Kubernetes集群中运行其他工作负载，强烈建议设置资源请求以保证良好的性能。如果不设置必要的资源请求，CockroachDB可能会缺乏CPU周期，或是在OOM的时候比起不太重要的资源被优先杀死。

查看Kubernetes节点能够使用的资源情况，可以执行：

kubectl describe nodes

Name: gke-perf-**default**-pool-aafee20c-k4t8

[...]

Capacity:

cpu: 4

**memory**: 15393536Ki

pods: 110

Allocatable:

cpu: 3920m

**memory**: 12694272Ki

pods: 110

[...]

Non-terminated Pods: (2 **in** total)

Namespace Name CPU Requests CPU Limits Memory Requests Memory Limits

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

kube-**system** kube-dns-778977457c-kqtlr 260m (6%) 0 (0%) 110Mi (0%) 170Mi (1%)

kube-**system** kube-proxy-gke-perf-**default**-pool-aafee20c-k4t8 100m (2%) 0 (0%) 0 (0%) 0 (0%)

Allocated resources:

(Total limits may be over 100 **percent**, i.e., overcommitted.)

CPU Requests CPU Limits Memory Requests Memory Limits

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

360m (9%) 0 (0%) 110Mi (0%) 170Mi (1%)

该命令将为集群中的每个节点输出大量信息。如上所示，可以看到每个节点上有多少“可分配”资源，和其他pods已使用了多少资源。 “可分配”资源指的是Kubernetes愿意为运行在一台机器上的pod提供的CPU和内存资源。其与节点的“容量”之间的差异是前者为Kubernetes的管理过程管理，后者为操作系统管理。 上述内容中“3920m”中的“m”代表“milli-CPUs”，即“千分之一CPU”。

Kubernetes在kube-system命名空间中运行少量pod，这些pod是集群基础结构的一部分。 考虑到其中一些进程对于Kubernetes集群的健康状况至关重要，这可能使得为CockroachDB保留节点上所有可分配资源变得困难。因此如果在集群中的每个节点上运行CockroachDB，则必须为这些进程预留资源。 如果只在集群中的部分节点上运行CockroachDB，则CockroachDB可以使用除了预留给一定会运行在集群中每个节点上的某些pods（例如kube-proxy或fluentd的日志代理）以外的所有“可分配”资源。

在v1.10或更早版本的Kubernetes的实际应用中，很难真正耗尽所有可分配空间，因为用户必须通过删除的手段手动地抢占已经在CockroachDB的节点上运行的kube-systempod 。 而在Kubernetes的未来版本中，当[pod优先级](https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/)功能从alpha升级到beta时，上述手动操作将变得更加容易实现。 如果该功能被广泛地使用，用户可以将CockroachDB pods设置为更高的优先级，Kubernetes调度程序为高优先级pods抢占资源并将kube-system pods重新安排到其他计算机上。

如果需要CockroachDB预留大量的CPU和内存资源，用户必须在CockroachDB YAML文件中配置资源请求，如下：

* 原文件：

containers:

- name: cockroachdb

image: cockroachdb/cockroach:v2.0.5

imagePullPolicy: IfNotPresent

ports:

- containerPort: 26257

name: grpc

- containerPort: 8080

name: http

* 新文件：

containers:

- name: cockroachdb

image: cockroachdb/cockroach:v2.0.5

imagePullPolicy: IfNotPresent

resources:

requests:

cpu: "3500m"

memory: "12300Mi"

ports:

- containerPort: 26257

name: grpc

- containerPort: 8080

name: http

创建StatefulSet时，用户需要检查以确保所有CockroachDB pods都已成功编排。 如果用户发现pod处于pending状态，需要执行kubectl describe pod <podname>并检查Event以获取有关pending状态的信息。 用户可能需要通过在其上执行kubectl delete pod来手动抢占一个或多个节点上的pods，以便为CockroachDB pods腾出空间。 由于删除的pods是由更高级别的Kubernetes对象（如Deployment或StatefulSet）创建的，它们将在另一个节点上被安全地重新创建。

资源限制

资源限制在概念上类似于资源请求，但服务于不同的目的。前者允许用户将pod的资源使用限制在不超过预设阀值的范围内。 资源限制有几个不同的用途：

* 提供更可预测的性能表现。pod在其所在机器上不允许突破限制地使用资源，在负载低的时候（例如流量暂停期间）不能比负载高的时候（同一时间段同一机器上其他pod充分利用其预留资源）获得更多的资源。
* Kubernetes v1.8及更低版本在运行时提供了服务质量（Quality of Service）管理，资源限制能够减少机器在超售时pod出现抢占的可能，[保证服务质量](https://github.com/kubernetes/community/blob/master/contributors/design-proposals/node/resource-qos.md)。
* 内存限制设定了容器可用的内存大小，有助于指定CockroachDB--cache和--max-sql-memory标志的百分比数值，具体可参考先前提供的默认的配置文件。

资源限制的配置类似于资源请求，用户可以在以上配置了资源请求的CockroachDB YAML文件的基础上，配置资源限制，如下：

containers:

- name: cockroachdb

image: cockroachdb/cockroach:v2.0.5

imagePullPolicy: IfNotPresent

resources:

requests:

cpu: "3500m"

memory: "12300Mi"

limits:

cpu: "3500m"

memory: "12300Mi"

ports:

- containerPort: 26257

name: grpc

- containerPort: 8080

name: http

该配置下pods将被限制为仅使用为其保留的资源，并保证除非在非常特殊的情况下，资源不会被其他pods抢占。 通常来说，pods不会在资源未充分利用的Kubernetes集群上获得更好的性能，但会在运行其他工作负载时提供更可预测的性能表现。

**WARNING:** 虽然强烈建议设置内存限制，但考虑到[设置CPU限制可能会影响Kubernetes目前实现的尾部延迟](https://github.com/kubernetes/kubernetes/issues/51135)，我们建议不要设置CPU限制。除非用户在设置Kubernetes集群时明确启用了非默认[静态CPU管理策略](https://kubernetes.io/docs/tasks/administer-cluster/cpu-management-policies/#static-policy)，或者用户只设置与请求对等的整数数值的（非小数）CPU限制和内存限制。

默认资源请求和限制

大多数情况下在安装Kubernetes的时候，会自动配置在dafault命名空间下的[LimitRange](https://kubernetes.io/docs/tasks/administer-cluster/cpu-default-namespace/)变量，通常配置为100m或是十分之一的CPU，用户可执行下述命令查看默认配置：

kubectl describe limitranges

Name: limits

Namespace: default

Type Resource M**in** Max Default Request Default Limit Max Limit/Request Ratio

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

Container cpu - - 100m - -

运行在同一机器的其他pods

如上面关于资源请求和限制的章节所提到的，即使用户没有创建任何pod，Kubernetes集群的机器上总会运行除了CockroachDB之外的其他pod。 用户可以通过如下命令随时查看：

kubectl get pods --all-namespaces

NAMESPACE NAME READY STATUS RESTARTS AGE

kube-system event-exporter-v0.1.7-5c4d9556cf-6v7lf 2/2 Running 0 2m

kube-system fluentd-gcp-v2.0.9-6rvmk 2/2 Running 0 2m

kube-system fluentd-gcp-v2.0.9-m2xgp 2/2 Running 0 2m

kube-system fluentd-gcp-v2.0.9-sfgps 2/2 Running 0 2m

kube-system fluentd-gcp-v2.0.9-szwwn 2/2 Running 0 2m

kube-system heapster-v1.4.3-968544ffd-5tsb8 3/3 Running 0 1m

kube-system kube-dns-778977457c-4s7vv 3/3 Running 0 1m

kube-system kube-dns-778977457c-ls6fq 3/3 Running 0 2m

kube-system kube-dns-autoscaler-7db47cb9b7-x2cc4 1/1 Running 0 2m

kube-system kube-proxy-gke-test-default-pool-828d39a7-dbn0 1/1 Running 0 2m

kube-system kube-proxy-gke-test-default-pool-828d39a7-nr06 1/1 Running 0 2m

kube-system kube-proxy-gke-test-default-pool-828d39a7-rc4m 1/1 Running 0 2m

kube-system kube-proxy-gke-test-default-pool-828d39a7-trd1 1/1 Running 0 2m

kube-system kubernetes-dashboard-768854d6dc-v7ng8 1/1 Running 0 2m

kube-system l7-default-backend-6497bcdb4d-2kbh4 1/1 Running 0 2m

[集群插件](https://github.com/kubernetes/kubernetes/tree/master/cluster/addons)提供了一系列的基础功能。例如：管理集群中服务的DNS条目，为Kubernetes仪表板UI显示提供支持；或是从集群中运行的所有pod中收集日志或指标。 如果用户不希望让这些功能占用集群的资源，可以通过适当地配置Kubernetes集群来阻止其中一些功能运行。 例如，在GKE上，您可以通过运行以下命令创建一个具有最小插件集的集群：

gcloud container clusters create <your-cluster-name> --no-enable-cloud-logging --no-enable-cloud-monitoring --addons=""

但是，一些类似kube-proxy和kube-dns的功能对于一个Kubernetes集群来说是必要的，意味着在集群中始终运行一些不属于用户创建的pods，因此了解CockroachDB必须与其他进程共享机器资源的事实及其可能带来的影响是十分重要的。 与CockroachDB pod在同一台计算机上的进程越多，其性能可能越差且越不可预测。 为了防止这种情况，强烈建议在CockroachDB pod上配置资源请求项，以提供某种程度上的CPU和内存资源隔离。

然而即使设置了资源请求，仍然存在争用共享资源（如网络I/O）或在[特殊情况](https://hackernoon.com/another-reason-why-your-docker-containers-may-be-slow-d37207dec27f)下内部内核数据结构的争用。 由于这些原因，加上每台机器上会运行的Kubernetes基础进程，在Kubernetes上运行的CockroachDB无法达到与直接在专用机器上运行相同的性能水平。在合理配置Kubernetes后，这种性能差异能够缩小到较小的水平。

如果设置资源请求的情况下用户仍然无法满足需求，可以尝试使用下文提到的专用节点。

同一台机器上运行CockroachDB以外的客户端应用程序

在与CockroachDB相同的机器上运行客户端应用程序（例如基准测试应用程序）会带来比同一台机器上运行Kubernetes系统pod更糟糕的影响。当应用程序获得比平时更多负载时，CockroachDB进程也会响应更多负载，这个过程中很可能会导致资源竞争。 避免这种情况的最佳方法是设置资源请求和限制，但如果用户因某些原因不愿意或无法执行此操作，可以在客户端应用程序上设置[反关联性调度策略](https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity)。在配置文件中设置pod的反关联策略：

* 原配置

spec:

containers:

* 新配置

spec:

affinity:

podAntiAffinity:

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 100

podAffinityTerm:

labelSelector:

matchExpressions:

- key: app

operator: In

values:

- loadgen

topologyKey: kubernetes.io/hostname

- weight: 99

podAffinityTerm:

labelSelector:

matchExpressions:

- key: app

operator: In

values:

- cockroachdb

topologyKey: kubernetes.io/hostname

containers:

该配置项首先将loadgenpod安排在彼此不同的节点上，这对loadgenpod自身的容错很重要。作为次要优先级，它将尝试把pod安排在尚未运行CockroachDBpod的节点上。这将保证负载生成器和CockroachDB集群的容错和性能两者之间的最好的平衡。

网络

[Kubernetes运行需要申请大量网络资源](https://kubernetes.io/docs/concepts/cluster-administration/networking/)，以便为集群中的每个pod提供可路由的IP地址和隔离的Linux网络命名空间，以满足其他要求。受篇幅所限无法解释更多细节，这些细节本身在很大程度上取决于用户为集群设置网络的具体方式。但通常来说Docker和Kubernetes的虚拟网络会给高吞吐量的分布式应用程序（如CockroachDB）在整体性能上带来损失。

网络方案

如果用户不使用托管的Kubernetes服务，则通常必须在创建Kubernetes集群时选择设置网络的方式。 有[很多解决方案](https://kubernetes.io/docs/concepts/cluster-administration/networking/#how-to-achieve-this)，它们具有明显不同的性能特征和功能。 我们没有可推荐的任何网络软件或配置方式，但必须指出比起在Kubernetes之外运行CockroachDB，用户在网络方案上的选择会对性能产生有重大的影响。

使用主机网络

如果用户满足于集群的网络设置或者不想将网络结构复杂化，Kubernetes确实提供了一个可选方案，即hostNetwork设置，可以避免网络性能开销。它允许用户绕过网络抽象层将pods直接运行在主机网络上。当然，这有许多缺点。例如，在同一台计算机上使用hostNetwork的两个pod不能使用相同的端口，且如果该机器可以在公网上访问到，也会产生严重的安全隐患。 即使如此，如果用户想尝试了解使用主机网络会对当前的工作负载有什么影响，只需要在CockroachDB YAML配置文件做如下更改：

* 原文件：

spec:

affinity:

* 新文件：

spec:

hostNetwork: true

dnsPolicy: ClusterFirstWithHostNet

affinity:

hostNetwork：true行代表Kubernetes将把pod放入主机网络的命名空间，使用其IP地址、主机名和整个网络协议栈。 dnsPolicy：ClusterFirstWithHostNet行代表Kubernetes配置pod仍然使用集群的DNS基础设备用于服务发现。

使用主机网络不会有明显的提升，所以请谨慎使用。 在已有的测试中，针对GKE上运行在[自定义kv负载生成器](https://hub.docker.com/r/cockroachdb/loadgen-kv/)的3个节点组成的集群，数据库吞吐量提高了6％。

DeamonSet运行

在先前的所有示例中，我们一直在使用标准的CockroachDB StatefulSet配置文件或者对它进行略微调整。 另一种替代方法是从使用StatefulSet进行编排切换到使用DaemonSet进行编排。[DaemonSet](https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/)是一种Kubernetes类型，用户在合适的场景下使用DeamonSet在所有节点上各运行一个pod。

它主要带来的好处如下：在隔离上是更自然的的抽象，存在于[专用节点](http://doc.cockroachchina.baidu.com/#deploy/orchestrated-deployment/kubernetes-performance-optimization/#dedicated-nodes)上。用户将CockroachDB进程与节点一对一地耦合，因此能够自然地[使用主机网络](http://doc.cockroachchina.baidu.com/#deploy/orchestrated-deployment/kubernetes-performance-optimization/#using-the-hosts-network)进行配对，并且它允许用户使用本地硬盘，不同于StatefulSets提供的使用本地磁盘的beta级别的支持。 DeamonSet最大的问题是它限制了Kubernetes帮助集群从节点故障中恢复的能力。DeamonSet在所有匹配的节点运行CockroachDB pod以后，一旦发生节点故障，它无法创建新的pod来替换故障节点上的pod，类似于CockroachDB直接在一组物理机器上运行，物理机器只能由人工操作来手动替换。

配置DaemonSet需要比StatefulSet多一些工作，我们将以[DaemonSet配置文件](https://github.com/cockroachdb/cockroach/blob/master/cloud/kubernetes/performance/cockroachdb-daemonset-insecure.yaml)为基础进行配置：

* 我们需要使用节点标签或是节点taints去选择需要运行CockroachDB的节点，否则Kubernetes集群的所有节点都会运行CockroachDB。在选择或创建节点后，请在DaemonSet YAML文件中适当配置相关[专用节点](http://doc.cockroachchina.baidu.com/#deploy/orchestrated-deployment/kubernetes-performance-optimization/#dedicated-nodes)部分。
* 在YAML文件中使用CockroachDB--join的标志去配置地址。配置文件默认[使用主机网络](http://doc.cockroachchina.baidu.com/#deploy/orchestrated-deployment/kubernetes-performance-optimization/#using-the-hosts-network)，因此我们需要使用主机的IP地址或hostname作为连接地址。 选出其中的两个或三个以包含并替换提供的文件中的列表(10.128.0.4,10.128.0.5,10.128.0.3)。 值得注意的是如果用户需要更新--join标签，将不需要的计算机及时从Kubernetes群集中删除，否则新的CockroachDB实例将无法加入群集。
* 从主机中选择要存储CockroachDB数据的目录，并将配置文件中的path：/ tmp / cockroach-data行替换为所需目录。
* 按照上述步骤或是其他自定义修改步骤配置完以后，用户可以执行命令启动DaemonSet：

kubectl create -f cockroachdb-daemonset.yaml

daemonset "cockroachdb" created

* 选择其中一个pod的名字来初始化集群，执行：

kubectl exec -it <pod-name> -- ./cockroach init --insecure

Cluster successfully initialized

专用节点

考虑到Kubernetes集群可能由异构硬件组成，用户希望CockroachDB仅在某些计算机上运行，从而尽可能多地使用某些机器的性能。用户可能还需要确保在这些机器上没有运行除CockroachDB之外的任何其他服务。

节点标签

节点标签和节点选择器可以在Kubernetes中指定一个pod运行在哪些节点上，标记节点可以通过执行kubectl label node命令指定一个节点的名称，替换响应的键值对：

kubectl **label** node <node-name> <key>=<value>

一些Kubernetes安装工具允许用户自动为对应的节点添加标签，例如创建[GKE Node Pool](https://cloud.google.com/kubernetes-engine/docs/concepts/node-pools)时，可以在执行gcloud container node-pools create命令的时候使用--node-labels标签来指定节点标签。

在给节点打标签以后，用户可以使用[NodeSelector](https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#nodeselector)去控制一个pod应该安排在哪些节点上，如下修改DaemonSet的配置文件：

* 原文件：

spec:

hostNetwork: true

containers:

* 新文件：

spec:

nodeSelector:

<key>: <value>

hostNetwork: true

containers:

节点taints

此外，如果想要保证CockroachDB是一组机器上唯一运行的的pod，用户可以使用一组名为[Taints和Tolerations](https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/)的拓展功能，来指定Kubernetes不要为这些机器安排其他pod。 类似于设置节点标签和节点选择器的方式来设置：

kubectl taint node <node-name> <key>=<value>:NoSchedule

一些Kubernetes安装工具也允许用户自动为对应的节点添加taints，例如创建[GKE Node Pool](https://cloud.google.com/kubernetes-engine/docs/concepts/node-pools)时，可以在执行gcloud container node-pools create命令时使用--node-taints标签来指定节点taints。

在给只运行CockroachDB的节点标记上合适的Taints以后，需要修改CockroachDB配置文件对应的Toleration部分，例如修改DaemonSet的配置文件：

* 原文件

spec:

hostNetwork: true

containers:

* 新文件：

spec:

tolerations:

- **key**: <**key**>

**operator**: "Equal"

value: <value>

effect: "NoSchedule"

hostNetwork: true

containers:

值得注意的是，上述设置只会阻止非CockroachDB的pod在这些计算机上运行，并不会阻止CockroachDB的pod在其他计算机上运行，因此在大多数情况下，用户需要将相应的节点标签和节点选择器与taints配对使用以创建真正的专用节点，对应的配置文件如下所示：

spec:

tolerations:

- key: <key>

operator: "Equal"

value: <value>

effect: "NoSchedule"

nodeSelector:

<key>: <value>

hostNetwork: true

containers:

调整已有的CockroachDB集群

Kubernetes可以轻松修改现有关资源使用的部分配置。 例如，用户很容易修改CPU和内存的资源请求，或者将节点添加到集群或升级到新的CockroachDB Docker镜像。 而其他一些配置修改则非常困难且容易出错，例如从StatefulSet更改为DaemonSet。 用户若要更新资源的配置，以下是几个常见的命令：

* 应用自定义调整后的配置文件

kubectl apply -f <your-file>.yaml

* 以文本编辑器打开StatefulSetYAML配置文件并做对应的修改

kubectl edit statefulset cockroachdb

* 以文本编辑器打开DaemonSetYAML配置文件并做对应的修改

kubectl edit daemonset cockroachdb

* 若想要执行一个单行命令，可以构造适当的JSON并执行

kubectl patch statefulset cockroachdb --type='json' -p='[{"op": "replace", "path": "/spec/template/spec/containers/0/image", "value":"cockroachdb/cockroach:VERSION"}]

**TIPS:** 查看更多Kubernetes对应版本的更新内容，可参考[官方文档](https://kubernetes.io/docs/concepts/cluster-administration/manage-deployment/#in-place-updates-of-resources)，或是使用kubectl <command> --help查看该命令的更多信息。