部署

如果 按照 中 做到了 一 ，希望 已 学到了一 件 于 Elasticsearch 的事情并且准 把 的集群部署到生 境。 一章不是在生 中 行集群的 尽指南，但是它涵 了集群上 之前需要考 的 事 。

主要包括三个方面：

- 后勤方面的考 ，如硬件和部署策略的建
- 更 合于生 境的配置更改
- 部署后的考 ，例如安全，最大限度的索引性能和

硬件

按照正常的流程， 可能已 在自己的 本 或集群上使用了 Elasticsearch。 但是当要部署 Elasticsearch 到生 境 ，有一些建 是 需要考 的。 里没有什 必 要遵守的准 ，Elasticsearch 被用于在 多的机器上 理各 任 。基于我 在生 境使用 Elasticsearch 集群的 ， 些建 可以 提供一个好的起点。

内存

如果有一 源是最先被耗尽的，它可能是内存。排序和聚合都很耗内存，所以有足 的堆空 来 付它 是很重要的。即使堆空 是比 小的 候， 也能 操作系 文件 存提供 外的内存。因 Lucene 使用的 多数据 是基于磁 的格式，Elasticsearch 利用操作系 存能 生很大效果。

64 GB 内存的机器是非常理想的， 但是32 GB 和16 GB 机器也是很常 的。少于8 GB 会 得其反（ 最 需要很多很多的小机器），大于64 GB 的机器也会有 ， 我 将在 堆内存:大小和交 中 。

CPUs

大多数 Elasticsearch 部署往往 CPU 要求不高。因此，相 其它 源，具体配置多少个（CPU ）不是那 。 具有多个内核的 代 理器，常 的集群使用 到八个核的机器。

如果 要在更快的 CPUs 和更多的核心之 ， 更多的核心更好。多个内核提供的 外并 微快一点点的 率。

硬

硬 所有的集群都很重要， 大量写入的集群更是加倍重要（例如那些存 日志数据的）。硬 是服 器上最慢的子系 ， 意味着那些写入量很大的集群很容易 硬 和，使得它成 集群的瓶 。

如果 担得起 SSD，它将 超出任何旋 介 （注：机械硬 ，磁 等）。 基于 SSD 的 点， 和索引性能都有提升。如果 担得起，SSD 是一个好的 。

的 I/O 度程序

如果 正在使用 SSDs， 保 的系 I/O 度程序是配置正 的。 当 向硬 写数据，I/O 度程序决定何 把数据 送到硬 。 大多数 *nix 行版下的 度程序都叫做 cfq（完全公平 列）。

度程序分配 片 到 个 程。并且 化 些到硬 的 多 列的 。但它是 旋 介 化的： 机械硬 的固有特性意味着它写入数据到基于物理布局的硬 会更高效。

SSD 来 是低效的，尽管 里没有 及到机械硬 。但是，deadline 或者 noop 被使用。deadline 度程序基于写入等待 行 化， noop 只是一个 的 FIFO 列。

个 的更改可以 来 著的影 。 是使用正 的 度程序，我 看到了500倍的写入能力提升。

如果 使用旋 介 ， 取尽可能快的硬 （高性能服 器硬 ，15k RPM 器）。

使用 RAID 0 是提高硬 速度的有效途径， 机械硬 和 SSD 来 都是如此。没有必要使用 像或其它 RAID 体，因 高可用已 通 replicas 内建于 Elasticsearch 之中。

最后，避免使用 附加存 （NAS）。人 常声称他 的 NAS 解决方案比本地 器更快更可 。除却 些声称， 我 从没看到 NAS 能配得上它的大肆宣 。NAS 常常很慢， 露出更大的延 和更 的平均延 方差，而且它是 点故障的。

快速可 的 然 分布式系 的性能是很重要的。 低延 能 助 保 点 能容易的通 ，大 能 助分片移 和恢 。 代数据中心 （1 GbE, 10 GbE） 大多数集群都是足 的。

即使数据中心 近在咫尺，也要避免集群跨越多个数据中心。 要避免集群跨越大的地理距 。

Elasticsearch 假定所有 点都是平等的—并不会因 有一半的 点在150ms 外的 一数据中心而有所不同。更大的延 会加重分布式系 中的 而且使得 和排 更困 。

和 NAS 的争 似， 个人都声称他 的数据中心 的 路都是健壮和低延 的。 是真的—直到它不是 （ 失 究是会 生的， 可以相信它）。 从我 的 来看，理跨数据中心集群的麻 事是根本不 得的。

取真正的高配机器在今天是可能的：成百 GB 的 RAM 和几十个 CPU 核心。 反之，在云平台上串起成千的小虚 机也是可能的，例如 EC2。 方式是最好的?

通常， 中配或者高配机器更好。避免使用低配机器， 因 不会希望去管理 有上千个 点的集群，而且在 些低配机器上 行 Elasticsearch 的 也是 著的。

与此同 ，避免使用真正的高配机器。它 通常会 致 源使用不均衡（例如，所有的内存都被使用，但 CPU 却没有）而且在 机上 行多个 点 ，会 加 度。

Java 虚　机

　　始　　行最新版本的　Java　虚　机（　JVM　），　除非　Elasticsearch　　站上　有　明。Elasticsearch，　特　是 Lucene，是一个高要求的　件。Lucene 的　元　　和集成　　常暴露出 JVM 本身的 bug。　些 bug 的　　从　微的麻　到　重段　　，所以，最好尽可能的使用最新版本的 JVM。

Java 8　烈　先　　于 Java 7。不再支持 Java 6。Oracle 或者 OpenJDK 是可以接受的，它　在性能和　定性也差不多。

如果　的　用程序是用　Java　　写并正在使用　　客　端（注：Transport　Client，下同）或　点客　端（注：Node　Client，下同），　　保　行　　用程序的 JVM 和服　器的 JVM 是完全一　的。 在 Elasticsearch 的几个地方，使用 Java 的本地序列化（ IP 地址、　常等等）。不幸的是，Oracle 的 JVM 在几个小版本之　有修改序列化格式，从而　致奇怪的　　。　　情况很少　，但最佳　践是客　端和服　器使用相同版本 JVM。

> **不要　整 JVM　置**
>
> JVM　暴露出几十个（甚至数百）的　置、参数和配置。　它　允　　行微　　JVM　几乎是　一个方面。　当遇到一个旋　，要打　它是人的本性。我　　求　　制　个本性，而　不要　去　整 JVM 参数。Elasticsearch 是　　的　件，并且我　根据多年的　　使用情况　整了当前 JVM　置。它很容易　始　　旋　，并　生　以衡量的、未知的影　，并最　使集群　入一个　慢的、不　定的混乱的效果。当　　集群　，第一　往往是去除所有的自定　配置。多数情况下，　此就可以恢　定和性能。

Transport Client 与 Node Client

如果　使用的是　　Java，　可能想知道何　使用　　客　端（注：Transport　　Client，下同）与　点客　端（注：Node　Client，下同）。　在　的　　所述，　　客　端作　一个集群和　用程序之　的通信　。它知道 API 并能自　　在　点之　　，　　嗅探集群等等。但它是集群 外部的 ，和 REST 客端　似。

　一方面，　点客　端，　　上是一个集群中的　点（但不保存数据，不能成　主　点）。因　它是一个　点，它知道整个集群状　（所有　点　留，分片分布在　些　点，等等）。　　意味着它可以　行　APIs 但少了一个　　点。

　里有　个客　端案例的使用情况：

- 如果要将　用程序和　Elasticsearch　集群　行解　，　　客　端是一个理想的　　。例如，如果　的　用程序需要快速的　建和　　到集群的　接，　　客　端比　点客　端”　”，因　它不是一个集群的一部分。

  　似地，如果　需要　建成千上万的　接，　不想有成千上万　点加入集群。　　客　端（　　TC　）将是一个更好的　　。

- 　一方面，如果　只需要有少数的、　期持久的　象　接到集群，客　端　点可以更高效，因　它知道集群的布局。但是它会使　的　用程序和集群　合在一起，所以从防火　的角度，它可能会　成　　。

配置管理

如果 已 使用配置管理（ Puppet，Chef，Ansible）， 可以跳 此提示。

如果 没有使用配置管理工具，那 注意了！通 `parallel-ssh` 管理少量服 器 在可能正常工作，但伴随着集群的 它将成 一 梦。 在不犯 的情况下手 30 个配置文件几乎是不可能的。

配置管理工具通 自 化更改配置的 程保持集群的一致性。 可能需要一点 来建立和学 ，但它本身 ，随着 的推移会有 厚的回 。

重要配置的修改

Elasticsearch 已 有了 很好 的 ，特 是 及到性能相 的配置或者 。 如果 有疑 ，最好就不要 它。我 已 目 了数十个因 的 置而 致 的集群， 因 它的管理者 改 一个配置或者 就可以 来 100 倍的提升。

NOTE	整 文章，所有的配置 都同等重要，和描述 序无 ， 所有的配置 ，并 用到 的集群中。

其它数据 可能需要 ，但 得来 ，Elasticsearch 不需要。 如果 遇到了性能 ，解决方法通常是更好的数据布局或者更多的 点。 在 Elasticsearch 中很少有“神奇的配置 ”，如果存在，我 也已 化了！

外，有些 上的 配置在生 境中是 整的。 些 整可能会 的工作更加 松，又或者因 没 法 定一个 （它取决于 的集群布局）。

指定名字

Elasticsearch 的集群名字叫 `elasticsearch` 。 最好 的生 境的集群改个名字，改名字的目的很 ， 就是防止某人的 本 加入了集群 意外。 修改成 `elasticsearch_production` 会很省心。

可以在 的 `elasticsearch.yml` 文件中修改：

```
cluster.name: elasticsearch_production
```

同 ，最好也修改 的 点名字。就像 在可能 的那 ， Elasticsearch 会在 的 点 的 候随机 它指定一个名字。 可能会 得 很有趣，但是当凌晨 3 点 的 候， 在 回 台物理机是 Tagak the Leopard Lord 的 候， 就不 得有趣了。

更重要的是， 些名字是在 的 候 生的， 次 点， 它都会得到一个新的名字。 会使日志 得很混乱，因 所有 点的名称都是不断 化的。

可能会 得 ，我 建 个 点 置一个有意 的、清楚的、描述性的名字，同 可以在 `elasticsearch.yml` 中配置：

```
node.name: elasticsearch_005_data
```

路径

情况下，Elasticsearch 会把 件、日志以及 最重要的数据放在安装目 下。 会 来不幸的事故，如果 重新安装 Elasticsearch 的 候不小心把安装目 覆 了。如果 不小心， 就可能把的全部数据 掉了。

不要笑， 情况，我 很多次了。

最好的 就是把 的数据目 配置到安装目 以外的地方， 同 也可以 移 的 件和日志目 。

可以更改如下：

```
path.data: /path/to/data1,/path/to/data2 ①

# Path to log files:
path.logs: /path/to/logs

# Path to where plugins are installed:
path.plugins: /path/to/plugins
```

① 注意： 可以通 逗号分隔指定多个目 。

数据可以保存到多个不同的目 ， 如果将 个目 分 挂 不同的硬 ， 可是一个 且高效 一个磁 列（ RAID 0 ）的 法。Elasticsearch 会自 把条 化（注：RAID 0 又称 Stripe（条化），在磁 列中,数据是以条 的方式 穿在磁 列所有硬 中的） 数据分隔到不同的目 ，以便提高性能。

WARNING

多个数据路径的安全性和性能

如同任何磁 列（ RAID 0 ）的配置，只有 一的数据拷 保存到硬器。如果 失去了一个硬 器， 肯定 会失去 算机上的一部分数据。 气好的 的副本在集群的其他地方，可以用来恢 数据和最近的 。

Elasticsearch 将全部的条 化分片放到 个 器来保 最小程度的数据失。 意味着 分片 0 将完全被放置在 个 器上。 Elasticsearch 没有一个条 化的分片跨越在多个 器，因 一个 器的 失会破坏整个分片。

性能 生的影 是：如果 添加多个 器来提高一个 独索引的性能，可能助不大，因 大多数 点只有一个分片和 一个 的器。多个数据路径只是 助如果 有 多索引／分片在 个 点上。

多个数据路径是一个非常方便的功能，但到 来，Elasticsearch 并不是 磁 列（software RAID ）的 件。如果 需要更高 的、 健的、 活的配置， 我 建 使用 磁 列（software RAID ）的 件，而不是多个数据路径的功能。

最小主 点数

`minimum_master_nodes` 定 的集群的 定 其 重要。 当 的集群中有 个 masters（注：主点）的 候， 个配置有助于防止 裂，一 个主 点同 存在于一个集群的 象。

如果 的集群 生了 裂，那 的集群就会 在 失数据的危 中，因 主 点被 是 个集群的最高

治者，它决定了什　候新的索引可以　建，分片是如何移　的等等。如果　有　个　masters　点，的数据的完整性将得不到保　，因　有　个　点　他　有集群的控制　。

个配置就是告　Elasticsearch 当没有足　master 候　点的　候，就不要　行 master　点　，等 master 候　点足　了才　行　。

此　置　始　被配置　master　候　点的法定个数（大多数个）。法定个数就是　(　master　候点个数 / 2) + 1。　里有几个例子：

- 如果　有 10 个　点（能保存数据，同　能成　master），法定数就是 6。
- 如果　有 3 个候　master　点，和 100 个 data　点，法定数就是 2，　只要数数那些可以做 master 的　点数就可以了。
- 如果　有　个　点，　遇到　了。法定数当然是　2　，但是　意味着如果有一个　点挂掉，　整个集群就不可用了。　置成　1　可以保　集群的功能，但是就无法保　集群　裂了，像的情况，　最好至少保　有 3 个　点。

可以在　的 elasticsearch.yml 文件中　配置：

```
discovery.zen.minimum_master_nodes: 2
```

但是由于　ELasticsearch　是　的，　可以很容易的添加和　除　点，　但是　会改　个法定个数。　不得不修改　一个索引　点的配置并且重　的整个集群只是　了　配置生效，　将是非常痛苦的一件事情。

基于　个原因，　minimum_master_nodes　(　有一些其它配置）允　通　API　用的方式　行配置。当　的集群在　行的　候，　可以　修改配置：

```
PUT /_cluster/settings
{
    "persistent" : {
        "discovery.zen.minimum_master_nodes" : 2
    }
}
```

将成　一个永久的配置，并且无　配置　里配置的如何，　个将　先生效。当　添加和　除　master 点的　候，　需要更改　个配置。

集群恢　方面的配置

当　集群重　，几个配置　影　的分片恢　的表　。首先，我　需要明白如果什　也没配置将会　生什　。

想象一下假　有　10　个　点，　个　点只保存一个分片，　个分片是一个主分片或者是一个副本分片，或者　有一个有　5　个主分片／1　个副本分片的索引。有　需要　整个集群做　（比如，　了安装一个新的　程序），　当　重　的集群，恰巧出　了 5 个　点已　，　有 5 个　没　的　景。

假　其它　5　个　点出　，或者他　根本没有收到立即重　的命令。不管什　原因，　有　5　个　点在

上， 五个 点会相互通信， 出一个 master，从而形成一个集群。 他 注意到数据不再均 分布，因 有 5 个 点在集群中 失了，所以他 之 会立即 分片 制。

最后， 的其它 5 个 点打 加入了集群。 些 点会 它 的数据正在被 制到其他 点，所以他 除本地数据（因 数据要 是多余的，要 是 的）。 然后整个集群重新 行平衡，因 集群的大小已 从 5 成了 10。

在整个 程中， 的 点会消耗磁 和 ，来回移 数据，因 没有更好的 法。 于有 TB 数据的大集群， 无用的数据 需要 很 。如果等待所有的 点重 好了，整个集群再上 ，所有的本地的数据都不需要移 。

在我 知道 的所在了，我 可以修改一些 置来 解它。 首先我 要 ELasticsearch 一个 格的限制：

```
gateway.recover_after_nodes: 8
```

将阻止 Elasticsearch 在存在至少 8 个 点（数据 点或者 master 点）之前 行数据恢 。 个 的 定取决于个人喜好：整个集群提供服 之前 希望有多少个 点在 ？ 情况下，我 置 8，意味着至少要有 8 个 点， 集群才可用。

在我 要告 Elasticsearch 集群中 有多少个 点，以及我 意 些 点等待多 ：

```
gateway.expected_nodes: 10
gateway.recover_after_time: 5m
```

意味着 Elasticsearch 会采取如下操作：

- 等待集群至少存在 8 个 点
- 等待 5 分 ，或者10 个 点上 后，才 行数据恢 ， 取决于 个条件先 到。

三个 置可以在集群重 的 候避免 多的分片交 。 可能会 数据恢 从数个小 短 几秒 。

注意： 些配置只能 置在 `config/elasticsearch.yml` 文件中或者是在命令行里（它 不能更新）它 只在整个集群重 的 候有 性作用。

最好使用 播代替 播

Elasticsearch 被配置 使用 播 ，以防止 点无意中加入集群。只有在同一台机器上 行的 点才会自 成集群。

然 播 然 作 件提供， 但它 永 不被使用在生 境了，否在 得到的 果就是一个 点意外的加入到了 的生 境， 是因 他 收到了一个 的 播信号。 于 播 本身 并没有 ， 播会 致一些愚蠢的 ，并且 致集群 的脆弱（比如，一个 工程 正在 鼓 ，而没有告 ， 会 所有的 点突然 不了 方了）。

使用 播， 可以 Elasticsearch 提供一些它 去 接的 点列表。 当一个 点 系到 播列表中的成 ，它就会得到整个集群所有 点的状 ，然后它会 系 master 点，并加入集群。

意味着 的 播列表不需要包含 的集群中的所有 点， 它只是需要足 的 点，当一个新 点系上其中一个并且 上 就可以了。如果 使用 master 候 点作 播列表，只要列出三个就可以了。 个配置在 elasticsearch.yml 文件中：

```
discovery.zen.ping.unicast.hosts: ["host1", "host2:port"]
```

于 Elasticsearch 点 的 信息， 参 Zen Discovery Elasticsearch 文献。

不要触 些配置！

在 Elasticsearch 中有一些 点，人 可能不可避免的会 到。 我 理解的，所有的 整就是 了 化，但是 些 整， 真的不需要理会它。因 它 常会被乱用，从而造成系 的不 定或者糟 的性能，甚至 者都有可能。

回收器

里已 要介 了 [garbage_collector_primer]，JVM 使用一个 回收器来 放不再使用的内存。 篇内容的 是上一篇的一个延 ，但是因 重要，所以 得 独拿出来作 一 。

不要更改 的 回收器！

Elasticsearch 的 回收器（ GC ）是 CMS。 个 回收器可以和 用并行 理，以便它可以最小化停 。然而，它有 个 stop-the-world 段， 理大内存也有点吃力。

尽管有 些 点，它 是目前 于像 Elasticsearch 低延 需求 件的最佳 回收器。官方建 使用 CMS。

在有一款新的 回收器，叫 G1 回收器（ G1GC ）。 款新的 GC 被 ，旨在比 CMS 更小的停 ，以及 大内存的 理能力。 它的原理是把内存分成 多区域，并且 些区域最有可能需要回收内存。通 先收集 些区域（ *garbage first* ）， 生更小的 停 ，从而能 更大的内存。

听起来很棒！ 憾的是，G1GC 是太新了， 常 新的 bugs。 些 通常是段（ segfault ） 型，便造成硬 的崩 。 Lucene 的 套件 回收算法要求 格，看起来 些 陷 G1GC 并没有很好地解决。

我 很希望在将来某一天推 使用 G1GC，但是 于 在，它 不能足 定的 足 Elasticsearch 和 Lucene 的要求。

程池

多人 喜 整 程池。 无 什 原因，人 都 加 程数无法抵抗。索引太多了？ 加 程！搜索太多了？ 加 程！ 点空 率低于 95%？ 加 程！

Elasticsearch 的 程 置已 是很合理的了。 于所有的 程池（除了 搜索 ）， 程个数是根据 CPU 核心数 置的。 如果 有 8 个核， 可以同 行的只有 8 个 程，只分配 8 个 程 任何特定的 程池是有道理的。

搜索 程池 置的大一点，配置 `int ( ( 核心数 * 3 ) / 2 ) + 1`。

可能会 某些 程可能会阻塞（如磁 上的 I／O 操作），所以 才想加大 程的。 于 Elasticsearch 来 并不是一个 ：因 大多数 I／O 的操作是由 Lucene 程管理的，而不是 Elasticsearch。

此外， 程池通 彼此之 的工作配合。 不必再因 它正在等待磁 写操作而担心 程阻塞，因 程早已把 个工作交 外的 程池，并且 行了 。

最后， 的 理器的 算能力是有限的， 有更多的 程会 致 的 理器 繁切 程上下文。 一个 理器同 只能 行一个 程。所以当它需要切 到其它不同的 程的 候，它会存 当前的状 （寄存器等等），然后加 外一个 程。 如果幸 的 ， 个切 生在同一个核心，如果不幸的 ， 个切 可能 生在不同的核心， 就需要在内核 上 行 。

个上下文的切 ，会 CPU 周期 来管理 度的 ；在 代的 CPUs 上， 估 高 30 µs。也就是 程会被堵塞超 30 µs，如果 个 用于 程的 行， 有可能早就 束了。

人 常稀里糊 的 置 程池的 。8 个核的 CPU，我 遇到 有人配了 60、100 甚至 1000 个 程。 些 置只会 CPU 工作效率更低。

所以，下次 不要 整 程池的 程数。如果 真 想 整 ， 一定要 注 的 CPU 核心数，最多 置成核心数的 倍，再多了都是浪 。

堆内存:大小和交

Elasticsearch 安装后 置的堆内存是 1 GB。 于任何一个 部署来 ， 个 置都太小了。如果 正在使用 些 堆内存配置， 的集群可能会出 。

里有 方式修改 Elasticsearch 的堆内存。最 的一个方法就是指定 `ES_HEAP_SIZE` 境 量。服 程在 候会 取 个 量，并相 的 置堆的大小。比如， 可以用下面的命令 置它：

```
export ES_HEAP_SIZE=10g
```

此外， 也可以通 命令行参数的形式，在程序 的 候把内存大小 它，如果 得 更 的 ：

```
./bin/elasticsearch -Xmx10g -Xms10g ①
```

① 保堆内存最小 （ `Xms` ）与最大 （ `Xmx` ）的大小是相同的，防止程序在 行 改 堆内存大小，是一个很耗系 源的 程。

通常来 ， 置 `ES_HEAP_SIZE` 境 量，比直接写 `-Xmx -Xms` 更好一点。

把 的内存的（少于）一半 Lucene

一个常 的 是 Elasticsearch 分配的内存 太 大了。假 有一个 64 GB 内存的机器， 天 ，我要把 64 GB 内存全都 Elasticsearch。因 越多越好 ！

当然，内存 于 Elasticsearch 来 是重要的，它可以被 多内存数据 使用来提供更快的操作。但是 到 里， 有 外一个内存消耗大 非堆内存 （off-heap）：Lucene。

Lucene 被 可以利用操作系 底 机制来 存内存数据 。 Lucene 的段是分 存 到 个文件中的。因 段是不可 的， 些文件也都不会 化， 是 存友好的，同 操作系 也会把 些段文件 存起来，以便更快的 。

Lucene 的性能取决于和操作系 的相互作用。如果 把所有的内存都分配 Elasticsearch 的堆内存，那将不会有剩余的内存交 Lucene。 将 重地影 全文 索的性能。

准的建 是把 50% 的可用内存作 Elasticsearch 的堆内存，保留剩下的 50%。当然它也不会被浪 ，Lucene 会很 意利用起余下的内存。

如果 不需要 分 字符串做聚合 算（例如，不需要 fielddata ）可以考 降低堆内存。堆内存越小，Elasticsearch（更快的 GC）和 Lucene（更多的内存用于 存）的性能越好。

不要超 32 GB！

里有 外一个原因不分配大内存 Elasticsearch。事 上， JVM 在内存小于 32 GB 的 候会采用一个内存 象指 技 。

在 Java 中，所有的 象都分配在堆上，并通 一个指 行引用。 普通 象指 （OOP）指向 些 象，通常 CPU 字 的大小：32 位或 64 位，取决于 的 理器。指 引用的就是 个 OOP 的字 位置。

于 32 位的系 ，意味着堆内存大小最大 4 GB。 于 64 位的系 ， 可以使用更大的内存，但是 64 位的指 意味着更大的浪 ，因 的指 本身大了。更糟 的是， 更大的指 在主内存和各 存（例如 LLC，L1 等）之 移 数据的 候，会占用更多的 。

Java 使用一个叫作 内存指 （compressed oops）的技 来解决 个 。 它的指 不再表示 象在内存中的精 位置，而是表示 偏移量 。 意味着 32 位的指 可以引用 40 个 象 ， 而不是 40 个字 。最 ， 也就是 堆内存 到 32 GB 的物理内存，也可以用 32 位的指 表示。

一旦 越 那个神奇的 ~32 GB 的 界，指 就会切回普通 象的指 。 个 象的指 都 了，就会使用更多的 CPU 内存 ，也就是 上失去了更多的内存。事 上，当内存到 40–50 GB 的 候，有效内存才相当于使用内存 象指 技 候的 32 GB 内存。

段描述的意思就是 ：即便 有足 的内存，也尽量不要 超 32 GB。因 它浪 了内存，降低了 CPU 的性能， 要 GC 大内存。

到底需要低于 32 GB多少，来 置我的 JVM？

憾的是， 需要看情况。 切的 分要根据 JVMs 和操作系 而定。 如果 想保 其安全可 ， 置堆内存 31 GB 是一个安全的 。 外， 可以在 的 JVM 置里添加 -XX:+PrintFlagsFinal 用来 JVM 的 界 ， 并且 UseCompressedOops 的 是否 true。 于 自己使用的 JVM 和操作系 ， 将 到最合 的堆内存 界 。

例如，我 在一台安装 Java 1.7 的 MacOSX 上 ，可以看到指 在被禁用之前，最大堆内存大 是在 32600 mb（~31.83 gb）：

```
$ JAVA_HOME=`/usr/libexec/java_home -v 1.7` java -Xmx32600m -XX:+PrintFlagsFinal 2>
/dev/null | grep UseCompressedOops
     bool UseCompressedOops   := true
$ JAVA_HOME=`/usr/libexec/java_home -v 1.7` java -Xmx32766m -XX:+PrintFlagsFinal 2>
/dev/null | grep UseCompressedOops
     bool UseCompressedOops   = false
```

相比之下，同一台机器安装 Java 1.8，可以看到指 在被禁用之前，最大堆内存大 是在 32766 mb（~31.99 gb）：

```
$ JAVA_HOME=`/usr/libexec/java_home -v 1.8` java -Xmx32766m -XX:+PrintFlagsFinal 2>
/dev/null | grep UseCompressedOops
     bool UseCompressedOops   := true
$ JAVA_HOME=`/usr/libexec/java_home -v 1.8` java -Xmx32767m -XX:+PrintFlagsFinal 2>
/dev/null | grep UseCompressedOops
     bool UseCompressedOops   = false
```

个例子告 我 ，影 内存指 使用的 界 ， 是会根据 JVM 的不同而 化的。 所以从其他地方取的例子，需要 慎使用，要 操作系 配置和 JVM。

如果使用的是 Elasticsearch v2.2.0， 日志其 会告 JVM 是否正在使用内存指 。会看到像 的日志消息：

```
[2015-12-16 13:53:33,417][INFO ][env] [Illyana Rasputin] heap size [989.8mb],
compressed ordinary object pointers [true]
```

表明内存指 正在被使用。如果没有，日志消息会 示 `[false]`。

我有一个 1 TB 内存的机器！

个 32 GB 的分割 是很重要的。那如果 的机器有很大的内存 ？ 一台有着 512–768 GB内存的服 器愈 常 。

首先，我 建 避免使用 的高配机器（参考 硬件）。

但是如果 已 有了 的机器， 有三个可 ：

- 主要做全文 索 ？考 Elasticsearch 4 - 32 GB 的内存， Lucene 通 操作系 文件存来利用余下的内存。那些内存都会用来 存 segments， 来 速的全文 索。
- 需要更多的排序和聚合？而且大部分的聚合 算是在数字、日期、地理点和 非分字符串上？ 很幸 ， 的聚合 算将在内存友好的 doc values 上完成！ Elasticsearch 4 到 32 GB 的内存，其余部分 操作系 存内存中的 doc values。
- 在 分 字符串做大量的排序和聚合（例如， 或者 SigTerms，等等）不幸的是，意味着 需要 fielddata，意味着 需要堆空 。考 在 个机器上 行 个或多个点，而不是 有大量 RAM 的一个 点。 然要 持 50% 原 。

假 有个机器有 128 GB 的内存， 可以 建 个 点， 个 点内存分配不超 32 GB。也就是 不超 64 GB 内存 ES 的堆内存，剩下的超 64 GB 的内存 Lucene。

如果 一 ， 需要配置 `cluster.routing.allocation.same_shard.host: true` 。 会防止同一个分片（shard）的主副本存在同一个物理机上（因 如果存在一个机器上，副本的高可用性就没有了）。

Swapping 是性能的 墓

是 而易 的，但是 是有必要 的更清楚一点：内存交 到磁 服 器性能来 是 致命的。想想看：一个内存操作必 能 被快速 行。

如果内存交 到磁 上，一个 100 微秒的操作可能 成 10 秒。 再想想那 多 10 微秒的操作 延累加起来。不 看出 swapping 于性能是多 可怕。

最好的 法就是在 的操作系 中完全禁用 swap。 可以 禁用：

```
sudo swapoff -a
```

如果需要永久禁用， 可能需要修改 `/etc/fstab` 文件， 要参考 的操作系 相 文 。

如果 并不打算完全禁用 swap，也可以 降低 `swappiness` 的 。 个 决定操作系 交 内存的 率。 可以 防正常情况下 生交 ，但 允 操作系 在 急情况下 生交 。

于大部分Linux操作系 ，可以在 `sysctl` 中 配置：

```
vm.swappiness = 1 ①
```

① `swappiness` 置 `1` 比 置 `0` 要好，因 在一些内核版本 `swappiness` 置 `0` 会触 系 OOM-

killer（注：Linux 内核的 Out of Memory（OOM）killer 机制）。

最后，如果上面的方法都不合 ， 需要打 配置文件中的 `mlockall` 。 它的作用就是允 JVM 住内存，禁止操作系 交 出去。在 的 `elasticsearch.yml` 文件中， 置如下：

```
bootstrap.mlockall: true
```

文件描述符和 MMap

Lucene 使用了 大量的 文件。 同 ，Elasticsearch 在 点和 HTTP 客 端之 行通信也使用了大量的套接字（注：sockets）。 所有 一切都需要足 的文件描述符。

可悲的是， 多 代的 Linux 行版本， 个 程 允 一个微不足道的 1024 文件描述符。 一个小的 Elasticsearch 点来 在是太 低 了，更不用 一个 理数以百 索引的 点。

加 的文件描述符， 置一个很大的 ，如 64,000。 个 程困 得 人 火，它高度依 于 的特定操作系 和分布。 参考 操作系 文 来 定如何最好地修改允 的文件描述符数量。

一旦 已 改 了它， Elasticsearch，以 保它的真的起作用并且有足 的文件描述符：

```
GET /_nodes/process

{
   "cluster_name": "elasticsearch__zach",
   "nodes": {
      "TGn9iO2_QQKb0kavcLbnDw": {
         "name": "Zach",
         "transport_address": "inet[/192.168.1.131:9300]",
         "host": "zacharys-air",
         "ip": "192.168.1.131",
         "version": "2.0.0-SNAPSHOT",
         "build": "612f461",
         "http_address": "inet[/192.168.1.131:9200]",
         "process": {
            "refresh_interval_in_millis": 1000,
            "id": 19808,
            "max_file_descriptors": 64000, ①
            "mlockall": true
         }
      }
   }
}
```

① `max_file_descriptors` 字段 示 Elasticsearch 程可以 的可用文件描述符数量。

Elasticsearch 各 文件混合使用了 NioFs（ 注：非阻塞文件系 ）和 MMapFs （注：内存映射文件系 ）。 保 配置的最大映射数量，以便有足 的虚 内存可用于 mmapped 文件。 可以 置：

```
sysctl -w vm.max_map_count=262144
```

或者　可以在 `/etc/sysctl.conf` 通　修改 `vm.max_map_count` 永久　置它。

在生　之前，重温　个列表

在　　入生　之前，　可能　　了本　。本章中　及的　　非常好，一般是可以知道的，但　　是，正　部署到生　　境之前需要重温　个列表。

一些　　会　　地阻止　(如：可用的文件描述符太少)。因　他　很快　　出来，　些都是容易　　的。其他的一些　　，如　裂和内存　置，只有在糟　的事情　生之后才可　。在　一点上，解决　法往往是凌乱和繁　的。

在　　生　之前　，通　　当配置集群来主　阻止　些情况　生，是更好的　　。所以如果　想要从整本　的一个部分折角（或保存　　），本章将是一个很好的　　。在部署到生　　境的前一周，　　地里　出的列表，并　　所有的建　。