部署后

一旦将集群部署到生　境后，就需要有一些工具及最佳　践　来保　集群　行在最佳状　。本章将探　配置、日志　、索引性能　化以及集群　。

更　置

Elasticsearch　里很多　置都是　的，可以通　API　修改。需要　制重　点（或者集群）的配置修改都要　力避免。而且　然通　静　配置　也可以完成　些　更，我　建　是用 API 来　。

集群更新 API 有　工作模式：

（*Transient*）
　些　更在集群重　之前一直会生效。一旦整个集群重　，　些配置就被清除。

永久（*Persistent*）
　些　更会永久存在直到被　式修改。即使全集群重　它　也会存活下来并覆　掉静　配置文件里的　。

　或永久配置需要在 JSON 体里分　指定：

```
PUT /_cluster/settings
{
    "persistent" : {
        "discovery.zen.minimum_master_nodes" : 2 ①
    },
    "transient" : {
        "indices.store.throttle.max_bytes_per_sec" : "50mb" ②
    }
}
```

① 个永久　置会在全集群重　存活下来。

② 个　置会在第一次全集群重　后被移除。

可以　更新的　置的完整清　，　{ref}/cluster-update-settings.html[online reference docs]。

日志

Elasticsearch 会　出很多日志，都放在 `ES_HOME/logs` 目　下。　的日志　等　是 `INFO` 。它提供了　度的信息，但是又　好了不至于　的日志太　大。

当　的　候，特　是　点　相　的　（因　个　常依　于各式　于繁　的　配置），提高日志　等　到 `DEBUG` 是很有　助的。

　可以　修改 `logging.yml` 文件然后重　的　点——但是　做即繁　会　致不必要的宕机　。作替代，　可以通　`cluster-settings` API 更新日志　，就像我　前面　学　的那　。

要　　个更新，　　感　趣的日志器，然后在前面　上 logger. 。　根日志器　可以用 logger._root 来表示。

我　高　点　的日志　　：

```
PUT /_cluster/settings
{
    "transient" : {
        "logger.discovery" : "DEBUG"
    }
}
```

置失效，Elasticsearch 将　始　出 discovery 模　的 DEBUG　　的日志。

TIP　避免使用 TRACE 。　个　非常的　，　到日志反而不再有用了。

慢日志

有　一个日志叫　　慢日志　　。　个日志的目的是捕　那些超　指定　　的　和索引　求。个日志用来追踪由用　生的很慢的　求很有用。

　情况，慢日志是不　的。要　它，需要定　具体　作（query，fetch　　是　index），期望的事件　等　（ WARN 、 DEBUG 等），以及　　。

是一个索引　的　置，也就是　可以独立　用　个索引：

```
PUT /my_index/_settings
{
    "index.search.slowlog.threshold.query.warn" : "10s", ①
    "index.search.slowlog.threshold.fetch.debug": "500ms", ②
    "index.indexing.slowlog.threshold.index.info": "5s" ③
}
```

① 慢于 10 秒　出一个 WARN 日志。

② 取慢于 500　秒　出一个 DEBUG 日志。

③ 索引慢于 5 秒　出一个 INFO 日志。

也可以在　　elasticsearch.yml　　文件里定　些　。没有　　置的索引会自　承在静配置文件里配置的参数。

一旦　　置　了，　可以和其他日志器一　切　日志　等　：

```
PUT /_cluster/settings
{
    "transient" : {
        "logger.index.search.slowlog" : "DEBUG", ①
        "logger.index.indexing.slowlog" : "WARN" ②
    }
}
```

① 置搜索慢日志 DEBUG 。

② 置索引慢日志 WARN 。

索引性能技巧

如果 是在一个索引 很重的 境，比如索引的是基 施日志， 可能 意 牲一些搜索性能 取更快的索引速率。在 些 景里，搜索常常是很少 的操作，而且一般是由 公司内部的人 起的。他 也 意 一个搜索等上几秒 ，而不像普通消 者，要求一个搜索必 秒 返回。

基于 特殊的 景，我 可以有几 衡 法来提高 的索引性能。

> 些技巧 用于 **Elasticsearch 1.3** 及以后的版本
>
> 本 是 最新几个版本的 Elasticsearch 写的， 然大多数内容在更老的版本也也有效。
>
> 不 ，本 提及的技巧， 只 1.3 及以后版本。 版本后有不少性能提升和故障修 是直接影 到索引的。事 上，有些建 在老版本上反而会因 故障或性能 陷而 降低 性能。

科学的 性能

性能 永 是 的，所以在 的方法里已 要尽可能的科学。随机 弄旋 以及写入 可不是做性能 的好 法。如果有太多 可能 ，我 就无法判断到底 一 有最好的 效果 。合理的 方法如下：

1. 在 个 点上， 个分片，无副本的 景 性能。
2. 在 100% 配置的情况下 性能 果， 就有了一个 比基 。
3. 保性能 行足 的 （30 分 以上） 可以 估 期性能，而不是短期的峰 或延 。一些事件（比如段合并，GC）不会立刻 生，所以性能概况会随着 而改 的。
4. 始在基 上逐一修改 。 格 它 ，如果性能提升可以接受，保留 个配置 ， 始下一 。

使用批量 求并 整其大小

而易 的， 化性能 使用批量 求。批量的大小 取决于 的数据、分析和集群配置，不 次批量数据 5–15 MB 大是个不 的起始点。注意 里 的是物理字 数大小。文 数 批量大小来 不是一个好指 。比如 ，如果 次批量索引 1000 个文 ， 住下面的事 ：

- 1000 个 1 KB 大小的文 加起来是 1 MB 大。
- 1000 个 100 KB 大小的文 加起来是 100 MB 大。

可是完完全全不一 的批量大小了。批量 求需要在 点上加 内存，所以批量 求的物理大小比文 数重要得多。

从 5–15 MB 始 批量 求大小， 慢 加 个数字，直到 看不到性能提升 止。然后 始 加 的批量写入的并 度（多 程等等 法）。

用 Marvel 以及 如 `iostat` 、 `top` 和 `ps` 等工具 控 的 点， 察 源什 候 到瓶 。如果 始收到 `EsRejectedExecutionException` ， 的集群没 法再 了：至少有一 源到瓶了。或者 少并 数，或者提供更多的受限 源（比如从机械磁 成 SSD），或者添加更多 点。

NOTE	写数据的 候，要 保批量 求是 往 的全部数据 点的。不要把所有 求都 个 点，因 个 点会需要在 理的 候把所有批量 求都存在内存里。

存

磁 在 代服 器上通常都是瓶 。Elasticsearch 重度使用磁 ， 的磁 能 理的 吐量越大， 的 点就越 定。 里有一些 化磁 I/O 的技巧：

- 使用 SSD。就像其他地方提 的，他 比机械磁 秀多了。
- 使用 RAID 0。条 化 RAID 会提高磁 I/O，代 然就是当一 硬 故障整个就故障了。不要使用 像或者奇偶校 RAID 因 副本已 提供了 个功能。
- 外，使用多 硬 ，并允 Elasticsearch 通 多个 `path.data` 目 配置把数据条 化分配到它上面。
- 不要使用 程挂 的存 ，比如 NFS 或者 SMB/CIFS。 个引入的延 性能来 完全是背道而 的。
- 如果 用的是 EC2，当心 EBS。即便是基于 SSD 的 EBS，通常也比本地 例的存 要慢。

段和合并

段合并的 算量 大，而且 要吃掉大量磁 I/O。合并在后台定期操作，因 他 可能要很 才能完成，尤其是比 大的段。 个通常来 都没 ，因 大 模段合并的概率是很小的。

不 有 候合并会 累写入速率。如果 个真的 生了，Elasticsearch 会自 限制索引 求到 个程里。 个可以防止出 段爆炸 ，即数以百 的段在被合并之前就生成出来。如果 Elasticsearch 合并 累索引了，它会会 一个声明有 `now throttling indexing` 的 `INFO` 信息。

Elasticsearch 置在 比 保守：不希望搜索性能被后台合并影 。不 有 候（尤其是 SSD，或者日志 景）限流 太低了。

是 20 MB/s， 机械磁 是个不 的 置。如果 用的是 SSD，可以考 提高到 100–200 MB/s。 的系 个 合 ：

```
PUT /_cluster/settings
{
    "persistent" : {
        "indices.store.throttle.max_bytes_per_sec" : "100mb"
    }
}
```

如果 在做批量 入，完全不在意搜索， 可以 底 掉合并限流。 的索引速度 到 磁 允 的限：

```
PUT /_cluster/settings
{
    "transient" : {
        "indices.store.throttle.type" : "none" ①
    }
}
```

① 置限流 型 none 底 合并限流。等 完成了 入， 得改回 merge 重新打 限流。

如果 使用的是机械磁 而非 SSD， 需要添加下面 个配置到 的 elasticsearch.yml 里：

```
index.merge.scheduler.max_thread_count: 1
```

机械磁 在并 I/O 支持方面比 差，所以我 需要降低 个索引并 磁 的 程数。 个 置允 max_thread_count + 2 个 程同 行磁 操作，也就是 置 1 允 三个 程。

于 SSD， 可以忽略 个 置， 是 Math.min(3, Runtime.getRuntime().availableProcessors() / 2)， SSD 来 行的很好。

最后， 可以 加 index.translog.flush_threshold_size 置，从 的 512 MB 到更大一些的 ，比如 1 GB。 可以在一次清空触 的 候在事 日志里 累出更大的段。而通 建更大的段，清空的 率 低，大段合并的 率也 低。 一切合起来 致更少的磁 I/O 和更好的索引速率。当然，会需要 量 的 heap 内存用以 累更大的 冲空 ， 整 个 置的 候 住 点。

其他

最后， 有一些其他 得考 的 西需要 住：

- 如果 的搜索 果不需要近 的准 度，考 把 个索引的 index.refresh_interval改到 30s。如果 是在做大批量 入， 入期 可以通 置 个 -1 掉刷新。 忘 在完工的 候重新 它。
- 如果 在做大批量 入，考 通 置 index.number_of_replicas: 0 副本。文 在 制的 候，整个文 内容都被 往副本 点，然后逐字的把索引 程重 一遍。 意味着 个副本也会 行分析、索引以及可能的合并 程。

  相反，如果 的索引是零副本，然后在写入完成后再 副本，恢 程本 上只是一个字 到字 的 。相比重 索引 程， 个算是相当高效的了。
- 如果 没有 个文 自 ID，使用 Elasticsearch 的自 ID 功能。 个 避免版本 做了化，因 自 生成的 ID 是唯一的。
- 如果 在使用自己的 ID， 使用一 Lucene 友好的 ID。包括零填充序列 ID、UUID-1 和 秒； 些 ID 都是有一致的， 良好的序列模式。相反的，像 UUID-4 的 ID，本 上是随机的， 比很低，会明 慢 Lucene。

推　分片分配

正如我　在　[_scale_horizontally]　　，　Elasticsearch　将自　在可用　点　　行分片均衡，包括新　点的加入和　有　点的　　。

理　上来　，　个是理想的行　，我　想要提　副本分片来尽快恢　　失的主分片。　　我　同　也希望保　源在整个集群的均衡，用以避免　点。

然而，在　践中，立即的再均衡所造成的　　会比其解决的更多。　例来　，考　到以下情形：

1. Node（　点）19 在　　中失　了（某个家　　到了　源　）
2. Master　立即注意到了　个　点的　　，它决定在集群内提　其他　有　Node　19　上面的主分片　　的副本分片　主分片
3. 在副本被提　　主分片以后，master　　点　始　行恢　操作来重建　失的副本。集群中的　点之　互相拷　分片数据，　　力　　，集群状　　。
4. 由于目前集群　于非平衡状　，　个　程　有可能会触　小　模的分片移　。其他不相　的分片将在　点　　移来　到一个最佳的平衡状

与此同　，那个　到　源　的倒　管理　，把服　器　好　源　　行了重　，　在　点　　Node　　19 又重新加入到了集群。不幸的是，　个　点被告知当前的数据已　没有用了，　　数据已　在其他　点上重新分配了。所以 Node 19 把本地的数据　行　除，然后重新　始恢　集群的其他分片（然后　又　致了一个新的再平衡）

如果　一切听起来是不必要的且　　大，那就　了。是的，不　前提是　知道　个　点会很快回来。如果　点 Node 19 真的　了，上面的流程　　正是我　想要　生的。

　了解决　　瞬　中断的　　，Elasticsearch　　可以推　分片的分配。　可以的集群在重新分配之前有　　去　　个　点是否会再次重新加入。

修改　　延

　　情况，集群会等待一分　来　看　点是否会重新加入，如果　个　点在此期　重新加入，重新加入的点会保持其　有的分片数据，不会触　新的分片分配。

通　修改参数 `delayed_timeout`，　　等待　　可以全局　置也可以在索引　　行修改:

```
PUT /_all/_settings ①
{
  "settings": {
    "index.unassigned.node_left.delayed_timeout": "5m" ②
  }
}
```

① 通　使用 `_all` 索引名，我　可以　集群里面的所有的索引使用　个参数

②　　被修改成了 5 分

　个配置是　　的，可以在　行　　行修改。如果　希望分片立即分配而不想等待，　可以　置参数：`delayed_timeout: 0`.

NOTE	延　分配不会阻止副本被提　　主分片。集群　是会　行必要的提　来　集群回到　yellow 状　。　失副本的重建是唯一被延　的　程。

自　取消分片　移

如果　点在超　之后再回来，且集群　没有完成分片的移　，会　生什　事情　？在　　情形下，Elasticsearch　　会　　机器磁　上的分片数据和当前集群中的活　主分片的数据是不是一　—如果　者匹配，　　明没有　来新的文　，包括　除和修改—那　　master　将会取消正在　行的再平衡并恢　机器磁　上的数据。

之所以　　做是因　本地磁　的恢　永　要比　　要快，并且我　保　了他　的分片数据是一　的，　个　程可以　是双　。

如果分片已　　生了分　（比如：　点　　之后又索引了新的文　），那　恢　　程会　　按照正常流程　行。重新加入的　点会　除本地的、　　的数据，然后重新　取一　新的。

重

有一天　会需要做一次集群的　　重　——保持集群在　和可操作，但是逐一把　点下　。

常　的原因：Elasticsearch　　版本升　，或者服　器自身的一些　　操作（比如操作系　升　或者硬件相　）。不管　　情况，都要有一　特　的方法来完成一次　　重　。

正常情况下，Elasticsearch　　希望　的数据被完全的　制和均衡的分布。如果　手　　了一个　点，集群会立刻　　点的　失并　始再平衡。如果　点的　　是短期工作的　，　一点就很　人了，因　大型分片的再平衡需要花　相当的　　（想想　　制　　1TB　　的数据——即便在高速　上也是不一般的事情了）。

我　需要的是，告　　Elasticsearch　　推　再平衡，因　　外部因子影　下的集群状　，我　自己更了解。操作流程如下：

1. 可能的　，停止索引新的数据。　然不是　次都能真的做到，但是　一　可以　助提高恢　速度。
2. 禁止分片分配。　一　阻止　　Elasticsearch　　再平衡　失的分片，直到　告　它可以　行了。如果　知道　　口会很短，　个主意棒　了。　可以像下面　　禁止分配：

   ```
   PUT /_cluster/settings
   {
       "transient" : {
           "cluster.routing.allocation.enable" : "none"
       }
   }
   ```

3. 　　个　点。
4. 　行　　/升　。
5. 重　　点，然后　　它加入到集群了。
6. 用如下命令重　分片分配：

```
PUT /_cluster/settings
{
    "transient" : {
        "cluster.routing.allocation.enable" : "all"
    }
}
```

分片再平衡会花一些　　。一直等到集群　成　色 状　后再　　。

7. 重　第 2 到 6　操作剩余　点。
8. 到　　可以安全的恢　索引了（如果　之前停止了的　），不　等待集群完全均衡后再恢　索引，也会有助于提高　理速度。

的集群

使用无　个存　数据的　件，定期　　的数据都是很重要的。Elasticsearch　　副本提供了高可性；它　　可以容忍零星的　点　失而不会中断服　。

但是，副本并不提供　　性故障的保　。　　情况，　需要的是　集群真正的　——在某些　西出　的　候有一个完整的拷　。

要　　的集群，　可以使用　　snapshot　　API。　个会拿到　集群里当前的状和数据然后保存到一个共享　里。　个　　程是"智能"的。　的第一个快照会是一个数据的完整拷　，但是所有后　的快照会保留的是已存快照和新数据之　的差　。随着　不　的　数据　行快照，　也在量的添加和　除。　意味着后　　会相当快速，因　它　只　很小的数据量。

要使用　个功能，　必　首先　建一个保存数据的　。有多个　　型可以供　　：

- 共享文件系　，比如 NAS
- Amazon S3
- HDFS (Hadoop 分布式文件系　)
- Azure Cloud

建

我部署一个共享文件系　　：

```
PUT _snapshot/my_backup ①
{
    "type": "fs", ②
    "settings": {
        "location": "/mount/backups/my_backup" ③
    }
}
```

①　我　的　　取一个名字，在本例它叫 my_backup 。

② 我 指定 的 型 是一个共享文件系 。

③ 最后，我 提供一个已挂 的 作 目的地址。

注意：共享文件系 路径必 保集群所有 点都可以 到。

会在挂 点 建 和所需的元数据。 有一些其他的配置 可能想要配置的， 些取决于 的 点、的性能状况和 位置：

`max_snapshot_bytes_per_sec`
当快照数据 入 ， 个参数控制 个 程的限流情况。 是 秒 `20mb` 。

`max_restore_bytes_per_sec`
当从 恢 数据 ， 个参数控制什 候恢 程会被限流以保障 的 不会被占 。 是 秒 `20mb`。

假 我 有一个非常快的 ，而且 外的流量也很 OK，那我 可以 加 些 ：

```
POST _snapshot/my_backup/ ①
{
    "type": "fs",
    "settings": {
        "location": "/mount/backups/my_backup",
        "max_snapshot_bytes_per_sec" : "50mb", ②
        "max_restore_bytes_per_sec" : "50mb"
    }
}
```

① 注意我 用的是 `POST` 而不是 `PUT` 。 会更新已有 的 置。

② 然后添加我 的新 置。

快照所有打 的索引

一个 可以包含多个快照。 个快照跟一系列索引相 （比如所有索引，一部分索引，或者 个索引）。当 建快照的 候， 指定 感 趣的索引然后 快照取一个唯一的名字。

我 从最基 的快照命令 始：

```
PUT _snapshot/my_backup/snapshot_1
```

个会 所有打 的索引到 `my_backup` 下一个命名 `snapshot_1` 的快照里。 个用会立刻返回，然后快照会在后台 行。

TIP

通常 会希望 的快照作 后台 程 行，不 有 候 会希望在 的脚本中一直等待到完成 。 可以通 添加一个 wait_for_completion :

```
PUT _snapshot/my_backup/snapshot_1?wait_for_completion=true
```

个会阻塞 用直到快照完成。注意大型快照会花很 才返回。

快照指定索引

行 是 所有打 的索引。不 如果 在用 Marvel， 不是真的想要把所有 断相 的 .marvel 索引也 起来。可能 就 根没那 大空 所有数据。

情况下， 可以在快照 的集群的 候指定 些索引：

```
PUT _snapshot/my_backup/snapshot_2
{
    "indices": "index_1,index_2"
}
```

个快照命令 在只会 index1 和 index2 了。

列出快照相 的信息

一旦 始在 的 里 起快照了， 可能就慢慢忘 里面各自的 了——特 是快照按照 分命名的 候（比如， backup_2014_10_28 ）。

要 得 个快照的信息，直接 和快照名 起一个 GET 求：

```
GET _snapshot/my_backup/snapshot_2
```

个会返回一个小 ，包括快照相 的各 信息：

```
{
   "snapshots": [
      {
         "snapshot": "snapshot_1",
         "indices": [
            ".marvel_2014_28_10",
            "index1",
            "index2"
         ],
         "state": "SUCCESS",
         "start_time": "2014-09-02T13:01:43.115Z",
         "start_time_in_millis": 1409662903115,
         "end_time": "2014-09-02T13:01:43.439Z",
         "end_time_in_millis": 1409662903439,
         "duration_in_millis": 324,
         "failures": [],
         "shards": {
            "total": 10,
            "failed": 0,
            "successful": 10
         }
      }
   ]
}
```

要 取一个 中所有快照的完整列表，使用 _all 占位符替 掉具体的快照名称：

```
GET _snapshot/my_backup/_all
```

除快照

最后，我 需要一个命令来 除所有不再有用的旧快照。 只要 /快照名称 一个 的 DELETE HTTP 用：

```
DELETE _snapshot/my_backup/snapshot_2
```

用 API 除快照很重要，而不能用其他机制（比如手 除，或者用 S3 上的自 清除工具）。因 快照是 量的，有可能很多快照依 于 去的段。delete API 知道 些数据 在被更多近期快照使用，然后会只 除不再被使用的段。

但是，如果 做了一次人工文件 除， 将会面 重 坏的 ，因 在 除的是可能 在使用中的数据。

控快照 度

wait_for_completion 提供了一个 控的基 形式，但 怕只是 一个中等 模的集群做快照恢 的 候，它都真的不 用。

外 个 API 会 有 快照状 更 的信息。首先 可以 快照 ID 行一个 GET，就像我 之前取一个特定快照的信息 做的那 ：

```
GET _snapshot/my_backup/snapshot_3
```

如果 用 个命令的 候快照 在 行中， 会看到它什 候 始， 行了多久等等信息。不 要注意， 个 API 用的是快照机制相同的 程池。如果 在快照非常大的分片，状 更新的 隔会很大，因 API 在 争相同的 程池 源。

更好的方案是 取 _status API 数据：

```
GET _snapshot/my_backup/snapshot_3/_status
```

_status API 立刻返回，然后 出 的多的 出：

```
{
   "snapshots": [
      {
         "snapshot": "snapshot_3",
         "repository": "my_backup",
         "state": "IN_PROGRESS", ①
         "shards_stats": {
            "initializing": 0,
            "started": 1, ②
            "finalizing": 0,
            "done": 4,
            "failed": 0,
            "total": 5
         },
         "stats": {
            "number_of_files": 5,
            "processed_files": 5,
            "total_size_in_bytes": 1792,
            "processed_size_in_bytes": 1792,
            "start_time_in_millis": 1409663054859,
            "time_in_millis": 64
         },
         "indices": {
            "index_3": {
               "shards_stats": {
                  "initializing": 0,
                  "started": 0,
                  "finalizing": 0,
                  "done": 5,
                  "failed": 0,
                  "total": 5
               },
               "stats": {
```

```
                  "number_of_files": 5,
                  "processed_files": 5,
                  "total_size_in_bytes": 1792,
                  "processed_size_in_bytes": 1792,
                  "start_time_in_millis": 1409663054859,
                  "time_in_millis": 64
               },
               "shards": {
                  "0": {
                     "stage": "DONE",
                     "stats": {
                        "number_of_files": 1,
                        "processed_files": 1,
                        "total_size_in_bytes": 514,
                        "processed_size_in_bytes": 514,
                        "start_time_in_millis": 1409663054862,
                        "time_in_millis": 22
                     }
                  },
                  ...
```

① 一个正在 行的快照会 示 IN_PROGRESS 作 状 。

② 个特定快照有一个分片 在 （ 外四个已 完成）。

包括快照的 体状况，但也包括下 到 个索引和 个分片的 。 个 展示了有 快照 展的非常 的 。分片可以在不同的完成状 ：

INITIALIZING
分片在 集群状 看看自己是否可以被快照。 个一般是非常快的。

STARTED
数据正在被 到 。

FINALIZING
数据 完成；分片 在在 送快照元数据。

DONE
快照完成！

FAILED
快照 理的 候 到了 ， 个分片/索引/快照不可能完成了。 的日志 取更多信息。

取消一个快照

最后， 可能想取消一个快照或恢 。因 它 是 期 行的 程， 行操作的 候一个 或者 就会花很 来解决——而且同 会耗尽有 的 源。

要取消一个快照，在他 行中的 候 的 除快照就可以：

```
DELETE _snapshot/my_backup/snapshot_3
```

个会中断快照 程。然后 除 里 行到一半的快照。

从快照恢

一旦 了数据，恢 它就 了：只要在 希望恢 回集群的快照 ID后面加上 _restore 即可：

```
POST _snapshot/my_backup/snapshot_1/_restore
```

行 是把 个快照里存有的所有索引都恢 。如果 snapshot_1 包括五个索引， 五个都会被恢到我 集群里。和 snapshot API 一 ，我 也可以 希望恢 具体 个索引。

有附加的 用来重命名索引。 个 允 通 模式匹配索引名称，然后通 恢 程提供一个新名称。如果 想在不替 有数据的前提下，恢 老数据来 内容，或者做其他 理， 个 很有用。我 从快照里恢 个索引并提供一个替 的名称：

```
POST /_snapshot/my_backup/snapshot_1/_restore
{
    "indices": "index_1", ①
    "rename_pattern": "index_(.+)", ②
    "rename_replacement": "restored_index_$1" ③
}
```

① 只恢 index_1 索引，忽略快照中存在的其余索引。

② 所提供的模式能匹配上的正在恢 的索引。

③ 然后把它 重命名成替代的模式。

个会恢 index_1 到 及群里，但是重命名成了 restored_index_1 。

TIP

和快照 似， restore 命令也会立刻返回，恢 程会在后台 行。如果 更希望 的 HTTP 用阻塞直到恢 完成，添加 wait_for_completion ：

```
POST _snapshot/my_backup/snapshot_1/_restore?wait_for_completion=true
```

控恢 操作

从 恢 数据借 了 Elasticsearch 里已有的 行恢 机制。在内部 上，从 恢 分片和从一个 点恢 是等 的。

如果 想 控恢 的 度， 可以使用 recovery API。 是一个通用目的的 API，用来展示 集群中移着的分片状 。

个 API 可以 在恢 的指定索引 独 用：

```
GET restored_index_3/_recovery
```

或者 看 集群里所有索引，可能包括跟 的恢 程无 的其他分片移 ：

```
GET /_recovery/
```

出会跟 个 似（注意，根据 集群的活 度， 出可能会 得非常 ！）：

```
{
  "restored_index_3" : {
    "shards" : [ {
      "id" : 0,
      "type" : "snapshot", ①
      "stage" : "index",
      "primary" : true,
      "start_time" : "2014-02-24T12:15:59.716",
      "stop_time" : 0,
      "total_time_in_millis" : 175576,
      "source" : { ②
        "repository" : "my_backup",
        "snapshot" : "snapshot_3",
        "index" : "restored_index_3"
      },
      "target" : {
        "id" : "ryqJ5lO5S4-lSFbGntkEkg",
        "hostname" : "my.fqdn",
        "ip" : "10.0.1.7",
        "name" : "my_es_node"
      },
      "index" : {
        "files" : {
          "total" : 73,
          "reused" : 0,
          "recovered" : 69,
          "percent" : "94.5%" ③
        },
        "bytes" : {
          "total" : 79063092,
          "reused" : 0,
          "recovered" : 68891939,
          "percent" : "87.1%"
        },
        "total_time_in_millis" : 0
      },
      "translog" : {
        "recovered" : 0,
        "total_time_in_millis" : 0
      },
      "start" : {
        "check_index_time" : 0,
        "total_time_in_millis" : 0
      }
    } ]
  }
}
```

① type 字段告　　恢　的本　；　个分片是在从一个快照恢　。

② source 哈希描述了作　恢　来源的特定快照和　　。

③ percent 字段　　恢　的状　有个概念。　个特定分片目前已　恢　了 94% 的文件；它就快完成了。

　出会列出所有目前正在　　恢　的索引，然后列出　些索引里的所有分片。　个分片里会有　　/停止、持　　、恢　百分比、　　字　数等　　。

取消一个恢

要取消一个恢　，　需要　除正在恢　的索引。因　恢　　程其　就是分片恢　，　送一个　　除索引　API 修改集群状　，就可以停止恢　　程。比如：

```
DELETE /restored_index_3
```

如果 restored_index_3 正在恢　中，　个　除命令会停止恢　，同　　除所有已　恢　到集群里的数据。

集群是活着的、呼吸着的生命

一旦　的集群投入生　，　会　　他就　始了他自己的一生。Elasticsearch　　努力工作来保　集群自　自足而且 真就在工作 。不　一个集群也　要有日常照料和投　，比如日常　　和升　。

Elasticsearch　　以非常快的速度　布新版本，　行　　修　和性能　　。保持　的集群采用最新版　是一个好主意。　似的，Lucene　　持　在　　JVM　　自身的新的和令人　　的　　，　意味着　需要尽量保持　的 JVM 是最新的。

　意味着最好是　有一个　准化的、日常的方案来操作　集群的　　重　和升　。升　　是一个日常程序，而不是一个需要好多个小　的精　　下的年度『惨　』。

　似的，　有一个　　是很重要的。　　的集群做　繁的快照——而且通　　行真　恢　的方式定期　　些快照！有些　　做日常　　却从不　　他　的恢　机制，　　直太常　了。通常　会在第一次演　真　恢　的　候　　明　的　陷（比如用　不知道　　挂　　个磁　）。比起在凌晨　　3　　点真的生危机的　候，在日常　　中暴露出　些　　是更好的。