1.**安装环境**（Windows和Linux：课件中我整理了图文安装教程，如果遇到问题可以自行百度，难度不大，实在不行也可以问我，这里就不演示步骤了）

(1) 安装ES

1. 安装六字箴言：
   1) JDK->依赖
   2) 下载->elastic.co
   3) 启动->./elasticsearch -d
   4) 验证->http://localhost:9200/
2. 开发模式和生产模式
   1) 开发模式：默认配置（未配置发现设置），用于学习阶段
   2) 生产模式：会触发ES的引导检查，学习阶段不建议修改集群相关的配置。

(2) 安装Kibana（从版本6.0.0开始，Kibana仅支持64位操作系统。）

1. **下载**：http://elastic.co
2. **启动**：依然是开箱即用
   Linux：./kibana
   Windows：.\kibana.bat
3. **验证**：localhost:5601

(3) **安装Head插件（选装）**：

1. **介绍**：提供可视化的操作页面对ElasticSearch搜索引擎进行各种设置和数据检索功能，可以很直观的查看集群的健康状况，索引分配情况，还可以管理索引和集群以及提供方便快捷的搜索功能等等。
2. **下载**：https://github.com/mobz/elasticsearch-head同时课件中也提供了安装包。
3. **安装**：依赖于node和grunt管理工具
4. **启动**：npm run start
5. **验证**：http://localhost:9100/

2.**集群健康值**：

(1) 健康值检查

1. _cat/health
2. _cluster/health

(2) 健康值状态

1. Green：所有Primary和Replica均为active，集群健康
2. Yellow：至少一个Replica不可用，但是所有Primary均为active，数据仍然是可以保证完整性的。
3. Red：至少有一个Primary为不可用状态，数据不完整，集群不可用。

3.基于XX系统的CRUD

1.创建索引：PUT /product?pretty

2.查询索引：GET _cat/indices?v

3.删除索引：DELETE /product?pretty

4.插入数据：

```
PUT /index/_doc/id
{
  Json数据
}
```

5.更新数据

1) 全量替换
2) 指定字段更新

6.删除数据 DELETE /index/type/id

ES分布式文档系统

1.ES如何实现高可用（生产环境均为一台机器一个节点）

(2) ES在分配单个索引的分片时会将每个分片尽可能分配到更多的节点上。但是，实际情况取决于集群拥有的分片和索引的数量以及它们的大小，不一定总是能均匀地分布。

(3) ES不允许Primary和它的Replica放在同一个节点中，并且同一个节点不接受完全相同的两个Replica

(4) 同一个节点允许多个索引的分片同时存在。

2.容错机制

(1) 啥叫容错?

1. 傻X的代码你能看懂，牛X的代码你也能看懂
2. 只能看懂自己的代码，容错性低
3. PS：**各种情况（支持的情况越多，容错性越好）**下，都能保证work 正常运行
4. 换到咱们ES上就是，就是在局部出错异常的情况下，保证服务正常运行并且有自行恢复能力。

(2) ES-node

1.

1) Master：主节点，每个集群都有且只有一个
   a. 尽量避免Master节点 node.data = true
2) voting：投票节点
   a. Node.voting_only = true（仅投票节点，即使配置了data.master = true，也不会参选, 但是仍然可以作为数据节点）。
3) coordinating：协调节点

每一个节点都隐式的是一个协调节点，如果同时设置了data.master = false和data.data=false，那么此节点将成为仅协调节点。

4) Master-eligible node（候选节点）：
5) Data node（数据节点）：
6) Ingest node：
7) Machine learning node（机器学习节点）：

1. 两个配置：node.master和node.data
   1) node.master = true node.data = true
   这是ES节点默认配置，既作为候选节点又作为数据节点，这样的节点一旦被选举为Master，压力是比较大的，通常来说Master节点应该只承担较为轻量级的任务，比如创建删除索引，分片均衡等。
   2) node.master = true node.data = false
   只作为候选节点，不作为数据节点，可参选Master节点，当选后成为真正的Master节点。
   3) node.master = false node.data = false
   既不当候选节点，也不作为数据节点，那就是仅协调节点，负责负载均衡
   4) node.master=false node.data=true
   不作为候选节点，但是作为数据节点，这样的节点主要负责数据存储和查询服务。

(3) 图解容错机制

1. **第一步：Master选举（假如宕机节点是Master）**
   1) 脑裂：可能会产生多个Master节点
   2) 解决：discovery.zen.minimum_master_nodes=N/2+1
2. 第二步：Replica容错，新的（或者原有）Master节点会将丢失的Primary对应的某个副本提升为Primary
3. 第三步：Master节点会尝试重启故障机
4. 第四步：数据同步，Master会将宕机期间丢失的数据同步到重启机器对应的分片上去

3、总结（如何提高ES分布式系统的可用性以及性能最大化）：

（1）每台节点的Shard数量越少，每个shard分配的CPU、内存和IO资源越多，单个Shard的性能越好，当一台机器一个Shard时，单个Shard性能最好。

（2）稳定的Master节点对于群集健康非常重要！理论上讲，应该尽可能的减轻Master节点的压力，分片数量越多，Master节点维护管理shard的任务越重，并且节点可能就要承担更多的数据转发任务，可增加“仅协调”节点来缓解Master节点和Data节点的压力，但是在集群中添加过多的仅协调节点会增加整个集群的负担，因为选择的主节点必须等待每个节点的集群状态更新确认。

（3）反过来说，如果相同资源分配相同的前提下，shard数量越少，单个shard的体积越大，查询性能越低，速度越慢，这个取舍应根据实际集群状况和结合应用场景等因素综合考虑。

（4）数据节点和Master节点一定要分开，集群规模越大，这样做的意义也就越大。

（5）数据节点处理与数据相关的操作，例如CRUD，搜索和聚合。这些操作是I / O，内存和CPU密集型的，所以他们需要更高配置的服务器以及更

高的带宽，并且集群的性能冗余非常重要。

（6）由于仅投票节不参与Master竞选，所以和真正的Master节点相比，它需要的内存和CPU较少。但是，所有候选节点以及仅投票节点都可能是数据节点，所以他们都需要快速稳定低延迟的网络。

（7）高可用性（HA）群集至少需要三个主节点，其中**至少两个不是仅投票**节点。即使其中一个节点发生故障，这样的群集也将能够选举一个主节点。生产环境最好设置3台仅Master候选节点（node.master = true node.data = true）

（8）为确保群集仍然可用，集群**不能同时停止投票配置中的一半或更多节点**。只要有一半以上的投票节点可用，群集仍可以正常工作。这意味着，如果存在三个或四个主节点合格的节点，则群集可以容忍其中一个节点不可用。如果有两个或更少的主机资格节点，则它们必须都保持可用