远程科研项目总结报告

项目名称：大数据远程课程

学校：上海大学

学院：计算机工程与科学学院

专业：计算机科学与技术

姓名:王梓坤

实习起止时间：2019.08.07-2019.09.04

目录

1.项目研究背景

1.1 大数据技术

大数据（big data）是指无法在一定时间范围内用常规软件工具进行捕捉、管理和处理的数据集合，是需要新处理模式才能具有更强的决策力、洞察发现力和流程优化能力的海量、高增长率和多样化的信息资产。大数据的 5V 特点是：Volume（大量）、Velocity（高速）、Variety（多样）、Value（价值）、Veracity（真实性）。

大数据技术的战略意义不在于掌握庞大的数据信息，而在于对这些含有意义的数据进行专业化处理。换而言之，如果把大数据比作一种产业，那么这种产业实现盈利的关键，在于提高对数据的“加工能力”，通过“加工”实现数据的“增值”。从技术上看，大数据与云计算的关系就像一枚硬币的正反面一样密不可分。大数据必然无法用单台的计算机进行处理，必须采用分布式架构。它的特色在于对海量数据进行分布式数据挖掘。但它必须依托云计算的分布式处理、分布式数据库和云存储、虚拟化技术。

随着云时代的来临，大数据（Big data）也吸引了越来越多的关注。大数据（Big data）通常用来形容一个公司创造的大量非结构化数据和半结构化数据，因为这些数据在下载到关系型数据库用于分析时会花费过多时间和金钱。大数据分析常和云计算联系到一起，因为实时的大型数据集分析需要像 MapReduce 一样的框架来向数十、数百或甚至数千的电脑分配工作。因此，大数据需要特殊的技术，以有效地处理大量的容忍经过时间内的数据。适用于大数据的技术，包括大规模并行处理（MPP）数据库、数据挖掘、分布式文件系统、分布式数据库、云计算平台、互联网和可扩展的存储系统。

大数据分析的六个基本方面：

1. Analytic Visualizations（可视化分析）
2. Data Mining Algorithms（数据挖掘算法）
3. Predictive Analytic Capabilities（预测性分析能力）
4. Semantic Engines（语义引擎）
5.Data Quality and Master Data Management（数据质量和数据管理）
6.数据存储，数据仓库

1.2 项目简介

将 MNIST 应用部署到容器里面，用户通过 curl -XPOST 命令提交带有手写体的数字图片，通过程序先将图片识别出来，然后将识别出的数字返回给用户。对 MNIST 中用户每次提交的图片、识别的文字和时间戳信息，都要记录到 Cassandra 以内进行存储。

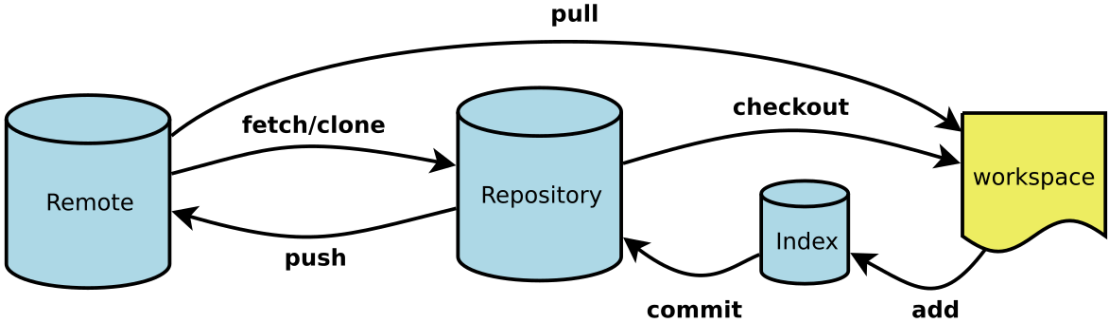
2.知识点回顾

2.1 Github

GitHub 是一个面向开源及私有软件项目的托管平台，因为只支持 git 作为唯一的版本库格式进行托管，故名 GitHub。GitHub 于 2008 年 4 月 10 日正式上线，除了 Git 代码仓库托管及基本的 Web 管理界面以外，还提供了订阅、讨论组、文本渲染、在线文件编辑器、协作图谱（报表）、代码片段分享（Gist）等功能。目前，其注册用户已经超过 350 万，托管版本数量也是非常之多，其中不乏知名开源项目 Ruby on Rails、jQuery、python 等。

作为开源代码库以及版本控制系统，Github 拥有超过 900 万开发者用户。随着越来越多的应用程序转移到了云上，Github 已经成为了管理软件开发以及发现已有代码的首选方法。作为一个分布式的版本控制系统，在 Git 中并不存在主库这样的概念，每一份复制出的库都可以独立使用，任何两个库之间的不一致之处都可以进行合并。

GitHub 可以托管各种 git 库，并提供一个 web 界面，但它与国外的 SourceForge、Google Code 或中国的 coding 的服务不同，GitHub 的独特卖点在于从另外一个项目进行分支的简易性。为一个项目贡献代码非常简单：首先点击项目站点的“fork”的按钮，然后将代码检出并将修改加入到刚才分出的代码库中，最后通过内建的“pull request”机制向项目负责人申请代码合并。已经有人将 GitHub 称为代码玩家的 MySpace。在 GitHub 进行分支就像在 Myspace 进行交友一样，在社会关系图的节点中不断的连线。在 GitHub 中，用户可以十分轻易地找到海量的开源代码。



下面列举一些常用的 Git 命令：

```
# 在当前目录新建一个 Git 代码库
$ git init
# 新建一个目录，将其初始化为 Git 代码库
$ git init [project-name]
# 下载一个项目和它的整个代码历史
$ git clone [url]
# 显示当前的 Git 配置
$ git config --list
# 编辑 Git 配置文件
$ git config -e [--global]
# 设置提交代码时的用户信息
$ git config [--global] user.name "[name]"
$ git config [--global] user.email "[email address]"
```

```
# 添加指定文件到暂存区
$ git add [file1] [file2] ...
# 添加指定目录到暂存区，包括子目录
$ git add [dir]
# 添加当前目录的所有文件到暂存区
$ git add .
# 添加每个变化前，都会要求确认
# 对于同一个文件的多处变化，可以实现分次提交
$ git add -p
# 删除工作区文件，并且将这次删除放入暂存区
$ git rm [file1] [file2] ...
# 停止追踪指定文件，但该文件会保留在工作区
$ git rm --cached [file]
# 改名文件，并且将这个改名放入暂存区
$ git mv [file-original] [file-renamed]
# 提交暂存区到仓库区
$ git commit -m [message]
# 提交暂存区的指定文件到仓库区
$ git commit [file1] [file2] ... -m [message]
# 提交工作区自上次 commit 之后的变化，直接到仓库区
$ git commit -a
# 提交时显示所有 diff 信息
$ git commit -v
# 使用一次新的 commit，替代上一次提交
# 如果代码没有任何新变化，则用来改写上一次 commit 的提交信息
$ git commit --amend -m [message]
# 重做上一次 commit，并包括指定文件的新变化
$ git commit --amend [file1] [file2] ...
# 列出所有本地分支
$ git branch
# 列出所有远程分支
$ git branch -r
# 列出所有本地分支和远程分支
$ git branch -a
# 新建一个分支，但依然停留在当前分支
$ git branch [branch-name]
# 新建一个分支，并切换到该分支
$ git checkout -b [branch]
# 新建一个分支，指向指定 commit
$ git branch [branch] [commit]
# 新建一个分支，与指定的远程分支建立追踪关系
$ git branch --track [branch] [remote-branch]
# 切换到指定分支，并更新工作区
$ git checkout [branch-name]
```

```
# 切换到上一个分支
$ git checkout -
# 建立追踪关系，在现有分支与指定的远程分支之间
$ git branch --set-upstream [branch] [remote-branch]
# 合并指定分支到当前分支
$ git merge [branch]
# 选择一个 commit，合并进当前分支
$ git cherry-pick [commit]
# 删除分支
$ git branch -d [branch-name]
# 删除远程分支
$ git push origin --delete [branch-name]
$ git branch -dr [remote/branch]
```

2.2 Docker 容器技术

Docker 容器技术首先就离不开容器这个概念，那么什么是容器呢?

一句话概括容器：容器就是将软件打包成标准化单元，以用于开发、交付和部署。容器镜像是轻量的、可执行的独立软件包 ，包含软件运行所需的所有内容：代码、运行时环境、系统工具、系统库和设置。容器化软件适用于基于 Linux 和 Windows 的应用，在任何环境中都能够始终如一地运行。容器赋予了软件独立性，使其免受外在环境差异（例如，开发和预演环境的差异）的影响，从而有助于减少团队间在相同基础设施上运行不同软件时的冲突。

接下来，再介绍一下 Docker：Docker 是一款世界领先的软件容器平台。Docker 使用 Google 公司推出的 Go 语言进行开发实现，基于 Linux 内核的 cgroup，namespace，以及 AUFS 类的 UnionFS 等技术，对进程进行封装隔离，属于操作系统层面的虚拟化技术。由于隔离的进程独立于宿主和其它的隔离的进程，因此也称其为容器。Docke 最初实现是基于 LXC。Docker 能够自动执行重复性任务，例如搭建和配置开发环境，从而解放了开发人员以便他们专注在真正重要的事情上：构建杰出的软件。用户可以方便地创建和使用容器，把自己的应用放入容器。容器还可以进行版本管理、复制、分享、修改，就像管理普通的代码一样。

Docker 容器的三个特点：

1. 轻量，在一台机器上运行的多个 Docker 容器可以共享这台机器的操作系统内核；它们能够迅速启动，只需占用很少的计算和内存资源。镜像是通过文件系统层进行构造的，并共享一些公共文件。这样就能尽量降低磁盘用量，并能更快地下载镜像。
2. 标准，Docker 容器基于开放式标准，能够在所有主流 Linux 版本、Microsoft Windows 以及包括 VM、裸机服务器和云在内的任何基础设施上运行。
3. 安全，Docker 赋予应用的隔离性不仅限于彼此隔离，还独立于底层的基础设施。Docker 默认提供最强的隔离，因此应用出现问题，也只是单个容器的问题，而不会波及到整台机器。

Docker 的优势在于：

1. Docker 的镜像提供了除内核外完整的运行时环境，确保了应用运行环境一致性，从而不会再出现“这段代码在我机器上没问题啊”这类问题；——一致的运行环境
2. 可以做到秒级、甚至毫秒级的启动时间。大大的节约了开发、测试、部署的时间。——更快速的启动时间

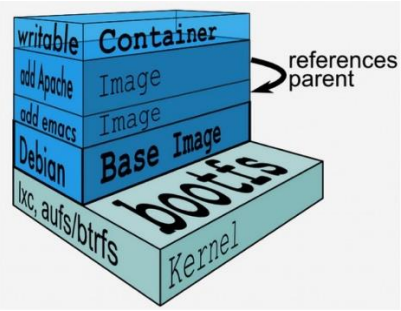
3. 避免公用的服务器，资源会容易受到其他用户的影响。——隔离性

4. 善于处理集中爆发的服务器使用压力；——弹性伸缩，快速扩展

5. 可以很轻易的将在一个平台上运行的应用，迁移到另一个平台上，而不用担心运行环境的变化导致应用无法正常运行的情况。——迁移方便

6. 使用 Docker 可以通过定制应用镜像来实现持续集成、持续交付、部署。——持续交付和部署

Docker 包括三个基本概念：镜像（Image），容器（Container），仓库（Repository）。



Docker 镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

镜像构建时，会一层层构建，前一层是后一层的基础。每一层构建完就不会再发生改变，后一层上的任何改变只发生在自己这一层。比如，删除前一层文件的操作，实际不是真的删除前一层的文件，而是仅在当前层标记为该文件已删除。在最终容器运行的时候，虽然不会看到这个文件，但是实际上该文件会一直跟随镜像。因此，在构建镜像的时候，需要额外小心，每一层尽量只包含该层需要添加的东西，任何额外的东西应该在该层构建结束前清理掉。分层存储的特征还使得镜像的复用、定制变的更为容易。甚至可以用之前构建好的镜像作为基础层，然后进一步添加新的层，以定制自己所需的内容，构建新的镜像。

镜像（Image）和容器（Container）的关系，就像是面向对象程序设计中的类和实例一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等 。容器的实质是进程，但与直接在宿主执行的进程不同，容器进程运行于属于自己的独立的命名空间。前面讲过镜像使用的是分层存储，容器也是如此。容器存储层的生存周期和容器一样，容器消亡时，容器存储层也随之消亡。因此，任何保存于容器存储层的信息都会随容器删除而丢失。

一个 Docker Registry 中可以包含多个仓库（Repository）；每个仓库可以包含多个标签（Tag）；每个标签对应一个镜像。所以说：镜像仓库是 Docker 用来集中存放镜像文件的地方类似于我们之前常用的代码仓库。Docker Registry 公开服务是开放给用户使用、允许用户管理镜像的 Registry 服务。一般这类公开服务允许用户免费上传、下载公开的镜像，并可能提供收费服务供用户管理私有镜像。

本项目常用的 docker 命令：

```
docker build -t friendlyhello .  # Create image using this directory's Dockerfile
docker run -p 4000:80 friendlyhello  # Run "friendlyhello" mapping port 4000 to 80
docker run -d -p 4000:80 friendlyhello         # Same thing, but in detached mode
docker container ls                                # List all running containers
docker container ls -a             # List all containers, even those not running
docker container stop <hash>           # Gracefully stop the specified container
docker container kill <hash>         # Force shutdown of the specified container
```

```
docker container rm <hash>              # Remove specified container from this machine
docker container rm $(docker container ls -a -q)          # Remove all containers
docker image ls -a                                        # List all images on this machine
docker image rm <image id>                # Remove specified image from this machine
docker image rm $(docker image ls -a -q)     # Remove all images from this machine
docker login                  # Log in this CLI session using your Docker credentials
docker tag <image> username/repository:tag    # Tag <image> for upload to registry
docker push username/repository:tag                  # Upload tagged image to registry
docker run username/repository:tag                           # Run image from a registry
```

2.3 RESTful API

REST 设计原则：

1. 客户端-服务器：通过将用户 UI 与数据存储分开，我们可以简化服务器组件来提高跨多个平台的用户界面的可移植性并提高可伸缩性。 它可以比表现成前后端分离的思想。
2. 无状态：从客户端到服务器的每个请求都必须包含理解请求所需的所有信息，并且不能利用服务器上任何存储的上下文。 这表示你应该尽可能的避免使用 session，由客户端自己标识会话状态。（token）
3. 规范接口：REST 接口约束定义：资源识别；请求动作；响应信息；它表示通过 uri 标出你要操作的资源，通过请求动作（http method）标识要执行的操作，通过返回的状态码来表示这次请求的执行结果。
4. 可缓存： 缓存约束要求将对请求的响应中的数据隐式或显式标记为可缓存或不可缓存。如果响应是可缓存的，则客户端缓存有权重用该响应数据以用于以后的等效请求。 它表示 get 请求响应头中应该表示有是否可缓存的头（Cache-Control）

四种基本操作：

GET：查询操作

POST：新增操作

PUT：修改操作

DELETE：删除操作

2.4 Cassandra 存储技术

Cassandra 是一个高可靠的大规模分布式存储系统。高度可伸缩的、一致的、分布式的结构化 key-value 存储方案，集 Google BigTable 的数据模型与 Amazon Dynamo 的完全分布式的架构于一身。Cassandra 使用了 Google BigTable 的数据模型，与面向行的传统的关系型数据库不同，这是一种面向列的数据库，列被组织成为列族（Column Family），在数据库中增加一列非常方便。Cassandra 的数据会写入多个节点，来保证数据的可靠性。适用于有节点、网络失效，以及多数据中心的场景。

特点：

1. 列表数据结构

在混合模式可以将超级列添加到 5 维的分布式 Key-Value 存储系统。

2. 真正的可扩展性

Cassandra 是纯粹意义上的水平扩展。为给集群添加更多容量，可以增加动态添加节点即可。你不必重启任何进程，改变应用查询，或手动迁移任何数据。

3. 分布式写操作

你以在任何地方任何时间集中读或写任何数据。并且不会有任何单点失败。

项目涉及命令:

#在 docker 中启动 cassandra

docker run --name some-cassandra --network some-network -d cassandra:tag

#用 cql 语言链接到 cassandra 数据库

docker run -it --network some-network --rm cassandra cqlsh some-cassandra

CQL 语言是一种类似于 SQL 的数据库编程语言，用于非关系型数据库 cassandra。

2.5 编程模型 MapReduce

Hadoop MapReduce 是一个软件框架，基于该框架能够容易地编写应用程序，这些应用程序能够运行在由上千个商用机器组成的大集群上，并以一种可靠的，具有容错能力的方式并行地处理上 TB 级别的海量数据集。

MapReduce 的思想就是“分而治之”。

（1）Mapper 负责“分”，即把复杂的任务分解为若干个“简单的任务”来处理。“简单的任务”包含三层含义：一是数据或计算的规模相对原任务要大大缩小；二是就近计算原则，即任务会分配到存放着所需数据的节点上进行计算；三是这些小任务可以并行计算，彼此间几乎没有依赖关系。

（2）Reducer 负责对 map 阶段的结果进行汇总。至于需要多少个 Reducer，用户可以根据具体问题，通过在 mapred-site.xml 配置文件里设置参数 mapred.reduce.tasks 的值，缺省值为 1。

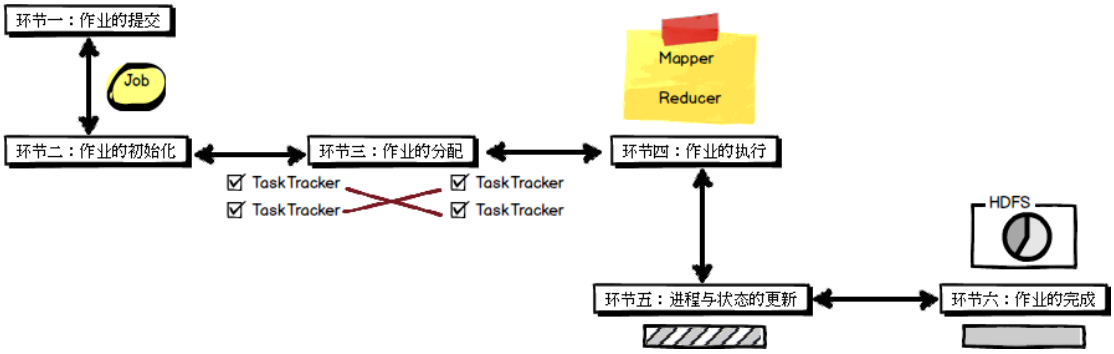
MapReduce 的整个工作过程包含如下 4 个独立的实体:

实体一：客户端，用来提交 MapReduce 作业。

实体二：JobTracker，用来协调作业的运行。

实体三：TaskTracker，用来处理作业划分后的任务。

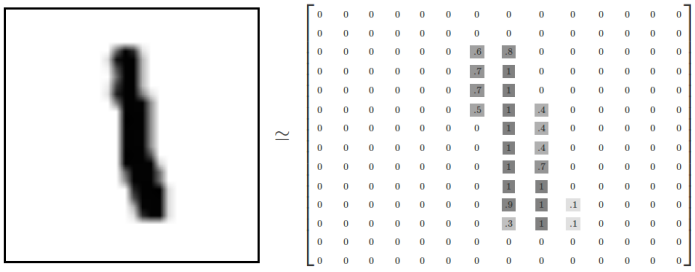
实体四：HDFS，用来在其它实体间共享作业文件。



2.6 MNIST

MNIST 是一个入门级的计算机视觉数据集，它包含各种手写数字图片，它也包含每一张图片对应的标签，告诉我们这个是数字几。数据集被分成两部分：60000 行的训练数据集（mnist.train）和 10000 行的测试数据集（mnist.test）。这样的切分很重要，在机器学习模型设计时必须有一个单独的测试数据集不用于训练而是用来评估这个模型的性能，从而更加容易把设计的模型推广到其他数据集上（泛化）。

每一个 MNIST 数据单元有两部分组成：一张包含手写数字的图片和一个对应的标签。我们把这些图片设为“xs”，把这些标签设为“ys”。训练数据集和测试数据集都包含 xs 和 ys，比如训练数据集的图片是 mnist.train.images ，训练数据集的标签是 mnist.train.labels。每一张图片包含 28X28 个像素点。我们可以用一个数字数组来表示这张图片：



2.7 Cloud Native

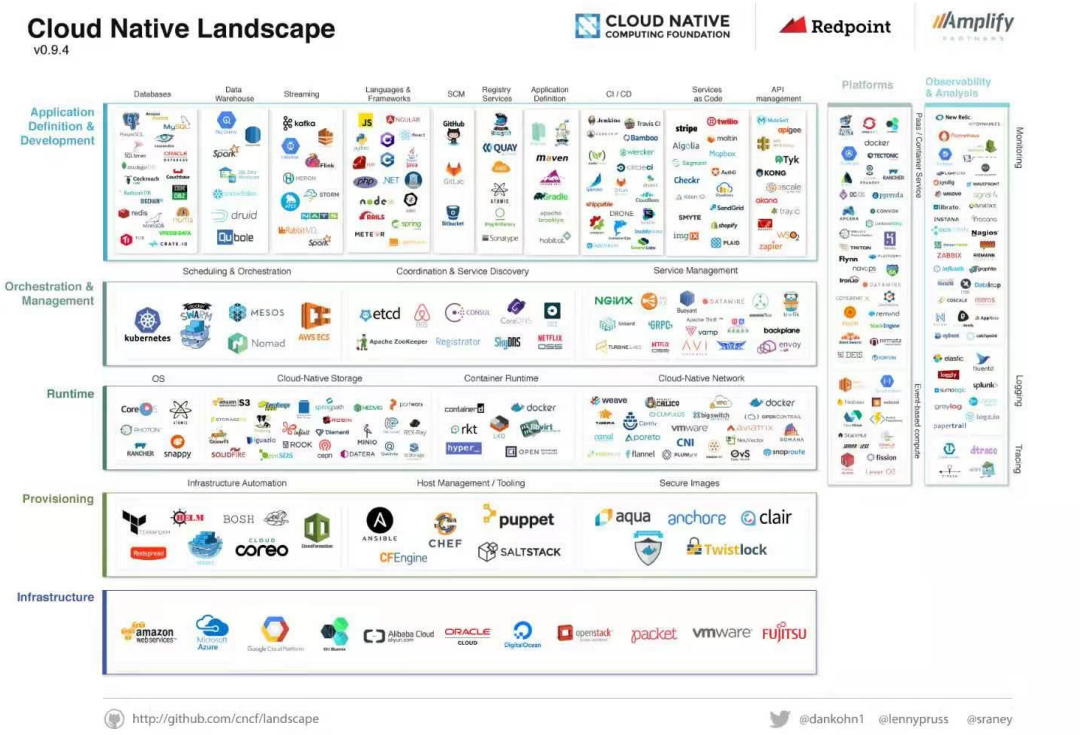
Cloud Native （国内译为“云原生”），最早是 Matt Stine 提出的一个概念。与微服务一样，Cloud Native 并不是一种具体的技术，而是一类思想的集合，包括 DevOps、持续交付（Continuous Delivery）、微服务（MicroServices）、敏捷基础设施（Agile Infrastructure）、康威定律（Conways Law）等，以及根据商业能力对公司进行重组。Cloud Native 既包含技术（微服务，敏捷基础设施），也包含管理（DevOps，持续交付，康威定律，重组等）。所以，Cloud Native 也可以说是一系列 Cloud 技术、企业管理方法的集合。

Cloud Native 具备有以下特性：以云为基础架构、云服务、无服务、可扩展、高可用、敏捷、云优先等等。

云计算的第一个浪潮是关于成本节约和业务敏捷性，尤其是云计算的基础设施更加廉价。很多企业倾向于使用微服务架构来开发应用。微服务开发快速，职责单一，能够更快速的被客户所采纳。同时，这些应用能够通过快速迭代的方式，得到进化，赢得客户的认可。Cloud Native 可以打通微服务开发、测试、部署、发布的整个流程环节。

云供应商为迎合市场，提供了满足各种场景方案的 API，例如用于定位的 Google Maps，用于社交协作的认证平台等。将所有这些 API 与企业业务的特性和功能混合在一起，可以让他们为客户构建独特的方案。所有这些整合都在 API 层面进行。这意味着，不管是移动应用还是传统的桌面应用都能无缝集成。所以，采用 Cloud Native 所开发的应用都且具备极强的可扩展性。

软件不可能不出故障。传统的企业级开发方式，需要有专职人员来对企业应用进行监控与维护。而在 Cloud Native 架构下，底层的服务或者是 API 都由将部署到云中，等价于将繁重的运维工作转移给了云平台供应商。这意味着客户应用将得到更加专业的看护，同时，也节省了运维成本。因此，云是大势所趋。



3.项目总结

3.1 个人心得体会

起初，我是抱着增加自己的知识储备和丰富科研项目经历的目的，参加到这个大数据远程课程项目中的，但在整个为期一个月的远程课程中，我发现这门远程课程带给我的改变远超出我的预料。

首先，因为是远程课程的缘故，所以我们一周只有三个小时的授课时间，我担心自己学习不到足够多的知识，并且可能会因为课程间隔时间较长，而不能够很好的将课程内容衔接起来。然而实际上，负责授课的张帆老师是一名非常负责且教学经验丰富的老师，在上课时间，他通过形象生动的例子让我们了解到所学习内容的核心，并在课后布置相关知识点的补充说明，这样我们就能够在这一周里的任意空闲时间进行自学，回顾课上内容的同时进一步补充完善相关知识的体系，并应用在最后的项目中。这样的教学方式极大地提高了我的学习效率，我感觉这一个月所学习到的知识，要是放在学校，可能要学两个月。除此之外，张老师为人也十分的亲切，无论我们何时遇到什么样的问题，都能从他那里得到建议，这样良好的沟通也利于我们顺利的完成项目并学到实用的技能知识。

其次，这次远程课程的主题是关于大数据方向的。原本以为专业是计算机科学与技术的我会有一些优势，但张老师上课所讲的内容与我在学校学到的知识完全不一样，例如：容器技术 Docker，编程模型 MapReduce，数据存储技术 Cassander，构建和调用服务的方法等等。老师解释说这些技术都是现在应用范围较广的技术，因此会与课本上讲述的计算机的基础知识有一定区别。老实说，这一点是我收获的最大的惊喜，我通过这次契机，接触到了这

些技术，我就能够有更多的时间去钻研和实践，这无疑会是我之后求职过程中的一大优势。再次感谢张帆老师带我入了这个门。

最后，我认为通过这次的项目，也提高了我写代码的能力，同时我也大致熟悉了完成一个项目的各个阶段所要做的工作。这对我之后的工作学习也有一定的指导作用。还有一点就是，因为授课导师是在 MIT 实验室工作的教授，所以张老师也比较熟悉美国大学的生活和学习，在授课过程中也会给我们一些建议，我相信这些能帮助我之后在美国更好的完成学业。总而言之，我感到十分幸运参加了这次的远程课程，认识了张帆导师和一些志同道合的同学，既学到了知识，又完善提升了自己的专业技能。我想我会永远记住这次宝贵的经历。