The Ultimate Guide to Switching Careers to Big Data

Upgrading Your Skills For the Big Data Revolution

The definitive guide to joining a Big Data team

Jesse Anderson

The Ultimate Guide to Switching Careers to Big Data

Upgrading Your Skills For the Big Data Revolution

Jesse Anderson

Version 1.1.d15957e

Contents

CHAPTER 1

Introduction

About This Book

This book is for individuals who want to change careers to Big Data. You want to join a data engineering or Big Data team. You're seeing that Big Data skills are in high demand and you want to become in high demand yourself — but you're lost. You're lost in this sea of weird names and don't know where to start so you don't waste your time.

This book is for everyone who wants to join a data engineering team or work with Big Data. This includes:

- Programmers who want to become Data Engineers
- Managers who want to lead data engineering teams
- Data Analysts and Business Intelligence professionals who want start using Big Data and eventually become Data Scientists
- Data Warehouse Engineers, DBAs, and similar SQL-like focused roles that want to become part of a data engineering team
- Operations specialists who want to get into the Big Data side of operations

This book focuses on what you need to do to get a job. It calls out some of the technologies to focus on, but doesn't go too deep. I don't show the actual code that you'll need to learn, that's another product altogether.

I want to you to come away from this guide with a very good understanding of what it's going to take to switch careers to Big Data. From there, you can make an educated decision whether this is the right path for you.

This is an ultimate guide. That means it goes into every question I get. No section or chapter is too verbose, but everything is covered. I don't want you to have to email me asking about a subject. It's all here.

About Big Data

First there's the definition of Big Data itself. The most orthodox definition is the 3 V's, 4 V's, or 5 V's. The actual number of V's depends on which company is trying to sell you something. These V's generally mean:

- Storing massive datasets
- Large user bases
- Data made up with a variety of types
- Running computationally complex algorithms on large datasets

The direct definition of the 3 V's is volume, variety, and velocity.

Personally, I don't like this definition. It's far too difficult to quantify and measure. This definition makes everything become Big Data and the term gets misused.

I prefer the definition of *can't*. If you are experiencing Big Data problems, you will start to say *can't*. You *can't* do something because:

- The processing will take too long
- There aren't enough resources or memory to process the data
- There is a technical limitation, like the RDBMS, that prevents you from doing it

This is what Big Data aims to solve. I like to tell people that Big Data puts the limitation on your imagination instead of on technical limitations.

This opens up some really interesting possibilities. Now that those small data constraints are lifted, you can go as deep as you want to. This includes processing all customer data from the beginning of time to see what their lifetime interactions are.

This allows your data and results to become the lifeblood of the organization. With data, you'll be interacting with all parts of your organization.

A Little About Me

Let me tell you a little bit about myself. I've seen, helped, and mentored thousands of people and hundreds of companies as they've gone through the process of learning Big Data. I know the things that work, and more importantly, I know the things that don't. I'm going to tell you the exact skills that every person needs, depending on which position they're looking for. If you follow the strategies and

plans I outline in this book, you'll be way ahead of the others trying to get into Big Data who've failed.

When other people interview for the same job, you'll be far ahead of them. You'll understand the fundamental differences between small data and Big Data jobs. When others just focus on a single technology, you'll give employers what they're looking for with your ability to show knowledge of all the right technologies. I'll show you what to do because I've interfaced with hundreds of companies and taught thousands of students. I've helped these students get their data engineering and Big Data dream jobs.

Warnings and Success Stories

This book is the result of years of carefully observing teams, individuals, and entire companies. I've also spent years teaching at hundreds of companies and to thousands of students. These companies span the world and are in all kinds of industries. My work starts with my very first interaction with teams and continues as I follow up with those teams to see the final outcome of their project. Other times, I start with students who are absolute beginners to Big Data and help them get their first data engineering job. From there, I analyze what went right, what went wrong, and why.

Throughout this book, I'm going to share some of these interactions and stories. They'll look something like this:

A cautionary story

Learn from this person's mistakes

These will be cautionary tales from the past. Learn from and avoid these mistakes.

Follow this example

Learn from this person's success

These will be success stories from the past. Learn from and imitate their success.

My story

Here's a story that will give you more color on the subject.

These will be stories or comments from my experience in the field teaching individuals. They will give you more background or color on a topic.

In Their Own Words

Here's a story that comes directly from a person.

These will be stories or comments from former students or someone who's been kind enough to share their own experiences with me. These are people who've shared the same journey as you. They will give you their own point of view on a topic.

Who Should Read This

This book is primarily written for individuals. These are people like:

- Programmers and Software Engineers
- Managers
- Data Analysts
- Business Intelligence Analysts
- Data Warehouse Engineers
- DBAs
- ETL and SQL Developers
- Operations and Administrative Specialists
- Enterprise, Data, and Software Architects
- Others with a general desire to get into Big Data

It will help you understand why some people succeed at getting a Big Data position while many others fail.

Navigating the Book Chapters

I highly recommend you read the entire book from start to finish to understand every concept and point. Without the entire background and all of the concepts, you may not understand fully why I recommend a technique or make a specific suggestion.

Here are the chapters and what we'll be covering in each chapter:

- Chapter 2 shares the benefits of getting into Big Data. There are many good reasons to start your Big Data journey and I'll share them.
- Chapter 3 tells you the skills that you'll need to have before you can join a data engineering team.
- Chapter 4 will talk about what you should know about Big Data before you start going down that path. These are things every person should know before they start a Big Data journey.
- Chapter 5 tells you the exact changes you'll need to make depending on your current position and skill set.
- Chapter 6 covers how to get the skills and knowledge to join a data engineering team or start working with Big Data. I talk about any of the extenuating circumstances you may have such as industry, country, or education.
- Chapter 7 shares the secrets to getting a Big Data job. These are the secrets I share with my students to stand out and get the best jobs.
- Chapter 8 goes step by step through the questions you should answer before starting to learn Big Data and gives you a checklist to verify you're ready to go.

Conventions Used in This Book

A *DBA* (Database Administrator) is someone who maintains a database and often writes SQL queries. For the purposes of this book, I'm going to group several different titles that write SQL queries together for simplicity's sake. In addition to DBA, these titles would be Data Warehouse Engineer, SQL Developer, and ETL Developer.

I will use the terms *Software Engineer* and *programmer* interchangeably. These are individuals who have programming or software engineering skills. They are the team members responsible for writing the project's code.

With all of the housekeeping out of the way, let's get started and learn how to switch careers and get into Big Data!

CHAPTER 2

The Benefits of Getting Into Big Data

There are some great reasons to switch careers and get into Big Data. These are some of the outward reasons people start in Big Data.

Pay and Career Advancement

Big Data is one of the leading areas where companies are increasing their investment and resources. My students looked at other people advancing their careers by getting into Big Data and thought that they can do it too. They looked at the other teams in the company and saw the data engineering team had open positions. Whereas, their team didn't have any open positions or was laying people off.

Pay and career advancement are some of the biggest motivators for people. You can make more, sometimes substantially more, with Big Data skills. People are taking their stagnating or disappearing career path and finding an expanding career path in Big Data.

How Much Higher?

In my previous career path there was a cap on how much I could progress, and the salary could reach maybe $100,000 to $130,000 — in data it's 60% higher and I'm no longer stuck on that other track. — Robert H.

Want to start interfacing with the CxO on a regular basis? A data engineering team frequently interacts with the CxOs and VPs of the company. They're creating the data products that these people consume and they're vital. My students advance faster because they're interfacing with the top people in the company and they're making a direct impact on the company's bottom line.

Interesting Work

Let's face it, most enterprise software is downright boring. There's only so many times you can write a CRM or the same select query. It just gets old after a while when you're doing the same thing over and over again. An enterprise software developer's day is more like *Groundhog Day* than *Independence Day*.

Working with data is different. Yes, there is some drudgery, but much of the job gives you the freedom to experiment. I find that good candidates for being a Data Engineer are people that are bored with the routine and want to start working on something that doesn't have a deterministic ending. You get to analyze, create the data pipelines, and consume the data pipelines that give you cool insights into what's happening within the organization.

Burning Out

My transition was due to being burnt out at work. I was a software support engineer for a company that made software tools that encrypted apps, obfuscated keys and other tools that kept people from ripping off licenses, etc. The work was very technical, but really demanding. I wanted more 'creative time'. — Stephan W.

High Demand/Low Supply

If you've ever taken an economics class, you know that the best position in a market is when there is a high demand and low supply for an item. In Big Data, there is a very high demand for qualified people who know Big Data technologies. However, there is a low supply of these people (we'll talk more about why the supply is so low in Chapter 4 "What You Should Know About Big Data").

This inequality in the market means several good things for you:

- You will have fewer people competing with you for the same job
- Pay for the positions will go up as companies compete for qualified people
- There is a decent barrier to entry for new people as not everyone can just up and learn Big Data
- People tend to get promoted quickly as their data engineering teams are newly established

How Can You Improve Your Skills and Be in Demand?

You've read the good things about Big Data. Now there's a gap between your current skills and the skills that will get you a Big Data job.

I see this gap all the time because I teach Big Data. I've taught thousands of students who are in your shoes right now. I've seen some students succeed and some students fail. My successful students are not unique snowflakes with tons of money or classes or special circumstances that have allowed them to be successful at learning Big Data. These students have taken an open and honest look at themselves and asked the following:

- Do I have a desire to learn Big Data?
- Do I have some of the prerequisite skills?
- Do I have the time to dedicate to learning?
- Do I have an expert (or experts) to guide me through the experience?

You will need all four of those points, in their entirety. If you're missing one of those points, it will take forever to learn Big Data and you'll give up. I've seen this many times when people talk to me at conferences or email me. They lack one of the essential points and never make any progress.

I want you to really look over these points so you don't waste your time pursuing something you can't fully realize. Let's go through them in more detail.

A Desire to Learn Big Data

You will have to put in the effort to learn Big Data. Just hoping and trying to passively learn isn't going to get you anywhere. I see this in the comments section of Big Data videos. "I sat there, learned a little passively, and wasn't challenged. This was easy." Six months later, this person still hasn't switched careers.

Some of the Prerequisite Skills

Depending on the position, you will need some skills in your toolbox. In Chapter 5 "Switching Careers," I go through these skills and positions in more depth. These required skills can range from Linux to programming skills. It all depends on the position you're seeking.

The Time to Dedicate to Learning

Let's say for whatever reason a person doesn't have the time to dedicate to learning. They've been misguided by their previous experience with small data technologies. They think they can get to an intermediate, or maybe even an advanced level, in a week or two. Those sorts of timelines don't carry over into Big Data. They will miss out on job opportunities because they aren't willing to put in the time and effort necessary to switch careers.

An Expert to Guide You Through the Experience

Learning Big Data isn't easy and it's even harder without someone who is a recognized expert to learn from. You probably can't look through a class or syllabus and spot the signs of a massive waste of time. You will need an expert to guide you through a complex Big Data landscape. Spend the extra time and money to find the right person to learn from.

Industries Using Big Data

Virtually every industry is using Big Data. Some organizations and industries have more data than others. Still others have been using Big Data for a longer period of time than others and some organizations are just starting out with Big Data.

Let's talk about how a few industries are using Big Data and where you'd fit into their team with a Big Data background. All of these industries are looking for qualified people and are having difficulty finding them.

IoT

The Internet of Things (IoT) is an exciting usage of Big Data. There are two general things most IoT companies need.

IoT companies need to ingest or acquire data — and this ingestion needs to happen very fast. This is because so many devices are sending in data at all times and the company is careful that important data doesn't get lost.

Next, they need to analyze that data in some way. This analysis could happen as the data comes in (real-time), later on as a file (batch), or both. The actual analysis will be driven by the use case.

A data engineering team is responsible for both the ingestion and analysis of incoming data. They may work with other parts of the organization to write the analysis or understand the sort of data they're working with.

Finance

Financial organizations make extensive use of Big Data. These organizations were some of the first with Big Data problems. They have all sorts of data that needs to be processed. This could be doing end-of-day reports and calculations. It could be providing the data for trading and predicting when to trade based on large amounts of input data.

Working at a financial organization requires a great deal of domain knowledge. If you're able to mix your in-depth domain knowledge with the Big Data technical knowledge, you'll be in high demand.

Financial Companies

I spend a good portion of my time teaching at financial companies. There are a few reasons for this. They have very specific Big Data needs and they can't just go out and hire new people. Usually, it's more time efficient to train their existing staff on Big Data because they already know the existing systems.

Financial organizations may not be the most exciting places, but they pay well and they're stable.

Social

Have you ever wondered why social media companies are worth so much despite having a product that is free? The answer is that companies like Facebook, Twitter, and LinkedIn are interested in your data. By making wise use of this data, they can use this data to market products to you.

That's the very basic description of their business model, but how do they do it from a technical perspective? They take their Big Data and process it to understand who you are and what you like. These companies are at the forefront of Big Data and often create their own Big Data technologies to handle their

use cases. A few examples of these are Presto from Facebook and Apache Storm from Twitter.

These companies want people with the latest skills. If you know the latest cutting edge technology, a social media company can make faster and better analysis about their users.

Marketing and eCommerce

Marketing and eCommerce companies share a common goal. Both types of organizations use data to sell to their customers. They have vast quantities of data about their customers. Most companies will track every online interaction with a site. The real value comes in analyzing those interactions.

Which pages or products did you visit before you chose the product you bought? Given a category, which product is bought most often? Given a product, which products are the most similar to it? These might sound like easy questions to answer, but at the scales of Big Data you have a more difficult problem. These interactions are spread on 100s of web servers' log files and on many different systems.

These companies have 50-200 million active customers. They'll be dealing with 10's to 250 million different products. Your knowledge of Big Data will help them answer these questions, no matter what the scale.

Government and Non-profits

Big Data is at all levels of government. It isn't just a federal or state/provincial problem. The datasets are entire countries. Some government organizations deal with data from the entire world.

Your knowledge of Big Data helps the government find value in a sea of data. Better yet, you'll be able to join other datasets together to process them all at once. Some of the biggest insights for government use cases come from taking data from different silos and processing them together.

Not Just Military and Spies

Usually the military and spy agencies get the most press about their Big Data efforts, but all parts of government use Big Data. One of my friends processed commercial and consumer data for consumer protection and consumer confidence. There are all different use cases.

Others

These are just a few of the high points of the industry usage. If your industry isn't here, don't worry. I didn't want this section to become an exhaustive list of every single industry.

I do want to share some common use cases that exist in almost every industry.

It's very common to take logging data and process it. This could be logs from your custom application, Apache web logs, or system logs. Once you have a large enough infrastructure, it's difficult to know what's running and what's about to encounter problems. These logging use cases bring all data together so it can be processed.

Most companies have a website. They'll often want to know more information about what's happening than their off-the-shelf software can provide. Or, they'll want to know something custom to their use case that's happening. Sometimes, they'll want deeper analytics into what their users did over large periods of time. Either way, they'll need Big Data and custom analytics.

Other companies need a customer service backend that operates at large scale. These backends are for companies with many customers. For each of those customers, there are all sorts of data points and the data may be coming from many different sources. The Big Data solution allows the company to combine all of the data in a single place so the customer service representatives have a complete view of the customer. This leads to happier customers and enables companies to process this data for other purposes.

CHAPTER 3

What Are the Skills Needed on a Data Engineering Team?

Before I can explain the exact skills you need, I need to give you some definitions. These definitions will help you understand how a Big Data team or data engineering team is set up and works. I'll also talk about how a data engineering team interacts with other parts of the organization.

If you are a team lead, manager, VP, or CxO and want to learn more about how to run data engineering teams, I've written an entire book on the subject titled *Data Engineering Teams*. Please visit http://tiny.bdi.io/detbook for more information about the book.

Don't worry, I'll give specific information about positions, titles, and what specific skills each member of the team needs in Chapter 5 "Switching Careers".

What Is a Data Engineer?

Let me start off by giving the title Data Engineer a formal definition. A Data Engineer is someone with specialized skills in creating software solutions around data. Their skills are predominantly based around Hadoop, Spark, and the open source Big Data ecosystem projects. Data Engineers come from a Software Engineering background and program in Java, Scala, or Python.

A Data Engineer has realized the need to go from being a general Software Engineer and specialize in Big Data as a Data Engineer. This is because Big Data is changing and they need to keep up with the changes. Also, there is a copious amount of knowledge that a Data Engineer needs to know and there isn't enough time to keep up with Big Data and other general software topics.

What Is a Data Engineering Team?

Next, we need to define a data engineering team, talk about where it lives in the organization, and how the team interacts with the rest of the organization.

I call the data engineering team the hub of the wheel for data. It shows how a data engineering team becomes an essential part of the business process.

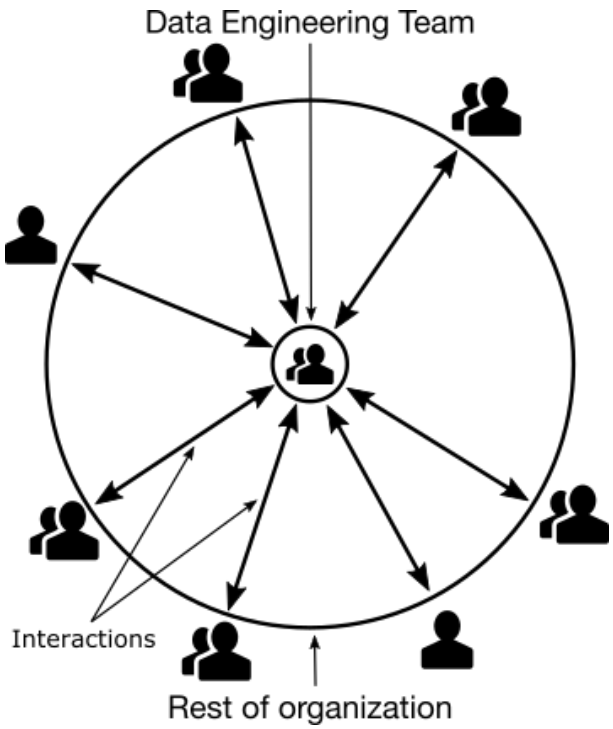

Figure 3.1: The data engineering team as the hub of data pipeline information for the organization

Being the hub in the wheel means the data engineering team needs to understand the whole data pipeline, and need to disseminate this information to other teams. The data engineering team will also need to help other teams know what data is available and what that data format is. Finally, they'll have to review code or write the code for other teams.

People often confuse data engineering teams and Data Engineers as being one in the same. They aren't. A data engineering team isn't made up of a single type of person or title. Rather, there are required skills that every data engineering

team needs. The sorts of titles on a data engineering team are Data Engineer, Data Architect, DBA (or variation thereof like Data Warehouse Engineer), and DevOps Engineer.

This multidisciplinary approach is required because the team doesn't just handle data. They aren't programmers behind the scenes that no one interacts with. The team interacts with virtually every part of the company, and the team's products become the company's lifeblood. This isn't your average backend software; this will be how your company improves its bottom line and starts making money from its data.

These results require people who aren't just ordinary programmers. They're programmers who have cross-trained in other fields and skills. The data engineering team also has some members or skillsets that aren't normally in an orthodox software engineering team.

The team is almost entirely Big Data focused. This is where the team's specialty is consistent. Everyone on the data engineering team needs to understand how Big Data systems are created and how they work.

That isn't to say that a data engineering team can't or won't create small data solutions. Some small data technologies will be part of a Big Data solution. Using the data engineering team for small data is entirely possible, but it is generally a waste of their specialty.

Qualified Data Engineers

I often talk about qualified Data Engineers. This means that they have shown their abilities in at least one real-world Big Data deployment. Their code is running in production, and they've learned from this experience. These engineers also know 10 to 30 different Big Data technologies.

A qualified Data Engineer's value is to know the right tool for the job. They understand the subtle differences in use cases and between technologies, and they can create data pipelines. These peope are the ones you rely on to make the difficult technology decisions.

Data Scientists and Data Science Teams

I will briefly define a Data Scientist and data science team, before I talk about their relationship to data engineering teams.

A Data Scientist is someone with a math and probability background who also knows how to program. They often know Big Data technologies in order to run their algorithms at scale.

A data science team is multidisciplinary, just like a data engineering team. The team has the variety of skills needed to prevent any gaps. It's unusual to have a single person with all of these skills and you'll usually need several different people.

A Data Engineer is different from a Data Scientist in that a Data Engineer is a much better programmer and distributed systems expert than a Data Scientist. A Data Scientist is more skilled at the math, analysis, and probabilities than a Data Engineer. That isn't to say there isn't some crossover, but my experience is that Data Scientists usually lack the hardcore engineering skills to create Big Data solutions. Conversely, Data Engineers lack the math backgrounds to do advanced analysis. Hence, the teams are more complementary than heavily overlapping.

Multidisciplinary

Data engineering teams are multidisciplinary in nature. In contrast to other teams, not everyone on the data engineering team will have the title Data Engineer. This is because the skills that are needed on a data engineering team aren't always found in just Data Engineers.

The team will be predominantly made up of Data Engineers, but some companies will embed a few non-Data Engineer positions. These titles include:

- Data Warehouse Engineer
- DevOps Engineer
- Data Scientist
- Business Intelligence or Data Analyst
- DBA

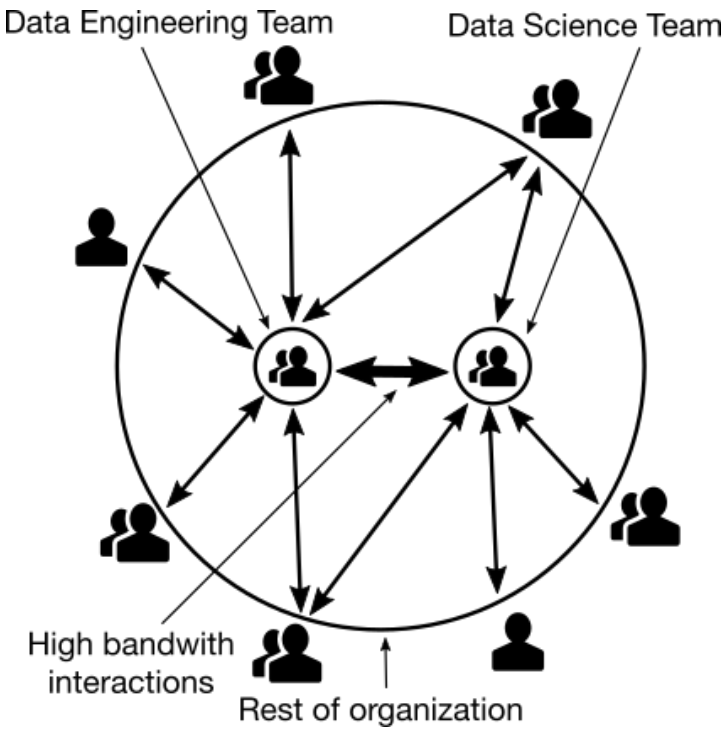

Figure 3.2: The data engineering team interacting with the data science team as the hubs of data for the organization

Where Do You Fit In These Teams?

Looking at this list of titles and positions on a data engineering team, you may not see your title. Does that mean you shouldn't be doing Big Data? No, the data engineering team creates the data pipelines. These pipelines are then consumed by the rest of the organization.

These other teams will need Big Data skills, though not at the technical level as their Data Engineer counterparts, to process or analyze the data. You or your team may exist on the outside of the hub making extensive usage of the data created inside the hub.

Data Engineering Team Skills

Now that we've defined some terms and helped you understand how you'll be interacting with the team, let's talk about the specific skills that are needed on a data engineering team. Every person on the team should have at least one of these skills, and ideally several.

The skills needed on a data engineering team are:

- Distributed systems
- Programming
- Analysis
- Visual communication
- Verbal communication
- Project veteran
- Schema
- Domain knowledge

Distributed Systems

Big Data is a subspecialty of distributed systems. Distributed systems are hard. You're taking many different computers and making them work together. This requires systems to be designed a different way — you actually have to design how data moves around those computers.

Having taught distributed systems for many years, I know this is something that takes people time to understand. It takes time and effort to get right.

Common titles with this skill: Software Architect, Software Engineer

Programming

This is the skill for someone who actually writes the code. They are tasked with taking the use case and writing the code that executes it on the Big Data framework.

The actual code for Big Data frameworks isn't difficult. Usually the biggest difficulty is keeping all of the different APIs straight; programmers will need to know 10 to 30 different technologies.

I also look to the programmers to give the team its engineering fundamentals. They're the ones expecting continuous integration, unit tests, and engineering processes. Sometimes data engineering teams forget that they are still doing engineering and operate as if they've forgotten their fundamentals.

Common titles with this skill: Software Engineer

Analysis

A data engineering team produces data analysis as a product. This analysis can range from simple counts and sums all the way up to more complex products. The actual bar or skill level can vary dramatically on data engineering teams; it will depend entirely on the use case and organization. The quickest way to judge the skill level needed for the analysis is to look at the complexity of the data products. Are they equations that most programmers wouldn't understand or are they relatively straightforward?

Other times, a data product is a simple report that's given to another business unit. This could be done with SQL queries.

Very advanced analysis is often the purview of a Data Scientist and may not be directly part of the data engineering team.

Common titles with this skill: Software Engineer, Data Analyst, Business Intelligence Analyst, DBA

Visual Communication

A data engineering team needs to communicate its data products visually. This is often the best way to show what's happening with data, especially vast amounts of it, so others can readily use it. You'll often have to show data over time and with animation. This function combines programming and visualization.

A team member with visual communication skills will help you tell a graphic story with your data. They can show the data not just in a logical way, but with the right aesthetics too.

Common titles with this skill: Software Engineer, Business Intelligence Analyst, UX Engineer, UI Engineer, Graphic Artist

Verbal Communication

The data engineering team is the hub in the wheel where many spokes of the organization come in. You need people on the team who can communicate verbally with the other parts of your organization.

Your verbal communicator is responsible for helping other teams be successful in using the Big Data platform or data products. They'll also need to speak to these teams about what data is available. Other data engineering teams will operate like internal solutions consultants.

This skill can mean the difference between increasing internal usage of the cluster and the work going to waste.

Common titles with this skill: Software Architect, Software Engineer, Technical Manager

Project Veteran

A project veteran is someone who has worked with Big Data and has had their solution in production for a while. This person is ideally someone who has extensive experience in distributed systems or, at the very least, extensive multithreading experience. This person brings a great deal of experience to the team.

The project veteran is the person that holds the team back from bad ideas. They have the experience to know when something is technically feasible, but a bad idea in the real world. They will give the team some footing or long-term viewpoint on distributed systems. This translates into better design that saves the team money and time once things are in production.

Common titles with this skill: Senior Software Architect, Senior Software Engineer, Senior Technical Manager

Schema

The schema skill is another odd skill for a data engineering team, because it's often missing. Members with this skill help teams lay out data. They're responsible for creating the data definitions and designing its representation when it is stored, retrieved, transmitted, or received.

The importance of this skill really manifests as data pipelines mature. I tell my classes that this is the skill that makes or breaks you as your data pipelines become more complex. When you have 1 PB of data saved on Hadoop, you can't rewrite it any time a new field is added. This skill helps you look at the data you have and the data you need to define what your data looks like.

Often, teams will choose a small data format like JSON or XML. Having 1 PB of XML, for example, means that 25% to 50% of the information is just the overhead of tags. It also means that data has to be serialized and deserialized every time it needs to be used.

The schema skill goes beyond simple data modeling. Practitioners will understand the difference between saving data as a string and a binary integer. They also advocate for binary formats like Avro or Protobuf. They know to do this because data usage grows as other groups in a company hear about its existence and capabilities. A format like Avro will keep the data engineering team from having to type-check everyone's code for correctness.

Common titles with this skill: DBA, Software Architect, Software Engineer

Domain Knowledge

Some jobs and companies aren't technology focused; they're actually domain expertise focused. These jobs focus 80% of their effort on understanding and implementing the domain. They focus the other 20% on getting the technology right. Domain-focused jobs are especially prevalent in finance, health care, consumer products, and other similar companies.

Domain knowledge needs to be part of the data engineering team. This person will need to know how the whole system works throughout the entire company. They'll need to deeply understand the domain for which you're creating data products; these data products will need to reflect this domain and be usable within it.

Common titles with this skill: DBA, Software Architect, Software Engineer, Technical Manager, The Graybeard, Project Manager

CHAPTER 4

What You Should Know About Big Data

What Putting in Your Own Lawn Has to Do With Big Data

I'm going to tell you a story about the time I put in my own lawn and sprinkler system. Trust me, it all relates to Big Data.

What's a Lawn and Sprinkler System?

For the folks around the world who don't know what lawns and sprinkler systems are, let me explain. A lawn is something (most) Americans want. It's an area of grass that you spend your weekends mowing and caring for. It doesn't rain enough rain where I live to keep a lawn green and yet my homeowner's association requires it to be green. I had to install a system that sprays water on the lawn.

See also:

- American obsession with lawns
- Homeowner associations
- Sprinkler systems

I'm a programmer, not a landscaper. I read a book and checked a few sites on how to install sprinkler systems and lay sod. Within a short amount of time I could go from an amateur to an intermediate level and get things done. I didn't have to spend a copious amount of time learning; the majority of the time was spent on the backbreaking labor.

As long as the sprinkler system generally worked, I was happy. Any mistakes that I made during the installation would be localized to a small area and under ground. If I had to fix something, I would simply dig up that small part of the

Figure 4.1: My backyard after putting the sod down

yard, fix the issue, and put it back to normal.

With these skills, I could do 99% percent of lawns. I could have helped other friends install their lawns. The same sorts of principles will apply.

Small Data

Working with small data has given you a similar comfort level. You didn't need an expert-level knowledge of every technology to use it right. You could implement things the same way even in different circumstances.

You have a software stack, be it Linux-Apache-MySQL-PHP or Linux-Tomcat-MySQL-Java. This same software stack can handle 99% of whatever is thrown at it.

A Common Frustration

This is one of the most common reasons people get frustrated learning Big Data. You don't have a have a simple software stack for everything. You'll need to know and understand more technologies at a much deeper level.

If there was a problem, the problem just had to be metaphorically dug up and fixed in one place. There weren't far reaching effects to the system and the system was relatively uncomplicated.

Big Data

Working with Big Data will get you outside of your comfort zone. You'll need to know 10-30 different technologies to create a data pipeline. This knowledge will need to range from cursory to expert level.

Instead of having a standard stack, you'll be focused on the use case. Every decision from technology choices to implementation will be predicated on your use case.

When you make a mistake, you'll have to metaphorically dig up the entire back-yard. The system is integrated and data pipelines have far reaching effects on the system. Likewise, the mistakes are exaggerated too. Mistakes with Big Data can take weeks or months to fix instead of hours or days.

There is a really big difference in how Big Data systems are built and updated.

Why Is Big Data So Much More Complicated?

Let's start off with a diagram that shows a sample architecture for a mobile product, with a database back end. Figure 4.1 illustrates a run-of-the-mill mobile application that uses a web service to store data in a database.

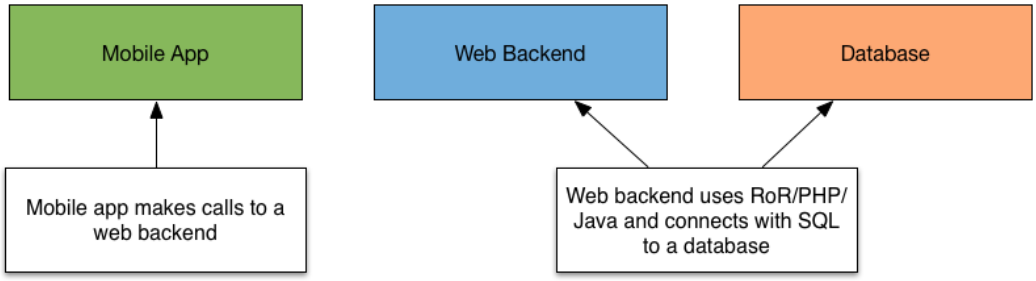

Figure 4.1: Simple architecture diagram for a mobile application, with a web backend.

Let's contrast the simple architecture for a mobile app with Figure 4.2, which shows a starter Hadoop solution.

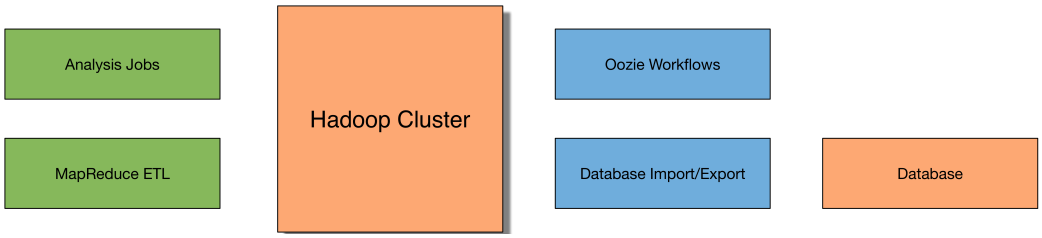

Figure 4.2: Simple architecture diagram for a Hadoop project.

As you can see, Hadoop weighs in at double the number of boxes in our diagram. Why? What's the big deal? It's that a "simple" Hadoop solution actually isn't very simple at all. You might think of this more as a starting point or, to put it another way, as "crawling" with Hadoop. The "Hello World!" of a Hadoop solution is more complicated than other domain's intermediate to advanced setups. Just look at the source code for a "Hello World!" in Big Data to see it's not so simple.

Now, let's look at a complicated mobile project's architecture diagram in Figure 4.3.

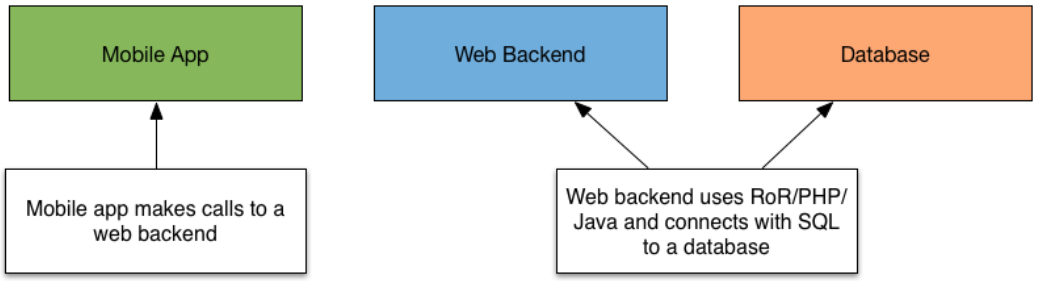

Figure 4.3: Complex architecture diagram for a mobile application with a web backend (yes, it's the same as the simple one).

Note: That's the same diagram as the simple mobile architecture. A more complex mobile solution usually requires more code or more web service calls, but no additional technologies are added to the mix.

Let's contrast a simple Big Data/Hadoop solution with a complex Big Data/Hadoop solution in Figure 4.4.

Yes, that's a lot of boxes, representing various types of components you may need when dealing with a complex Big Data solution. This is what I call the "running" phase of a Big Data project.

You might think I'm exaggerating the number of technologies to make a point; I'm not. I've taught at companies where this is their basic architectural stack. I've also taught at companies with twice as much complexity as this.

Instead of just looking at boxes, let's consider how many days of training it would take between a complex mobile and Big Data solution, assuming you already know Java and SQL. Based on my experience, a complex mobile course would take four to five days, compared to a complex Big Data course, which would take 18 to 20 days, and this estimate assumes you can grok the distributed systems side of all of this training. In my experience teaching courses, a Data Engineer can learn mobile, but a Mobile Engineer has a very difficult time learning data engineering.

You'll see me say that Data Engineers need to know 10 to 30 different technologies in order to choose the right tool for the job. Data engineering is hard because we're taking 10 complex systems, for example, and making them work together at a large scale. There are about 10 shown in Figure 4.4. To make the right decision in choosing, for example, a NoSQL cluster, you'll need to have

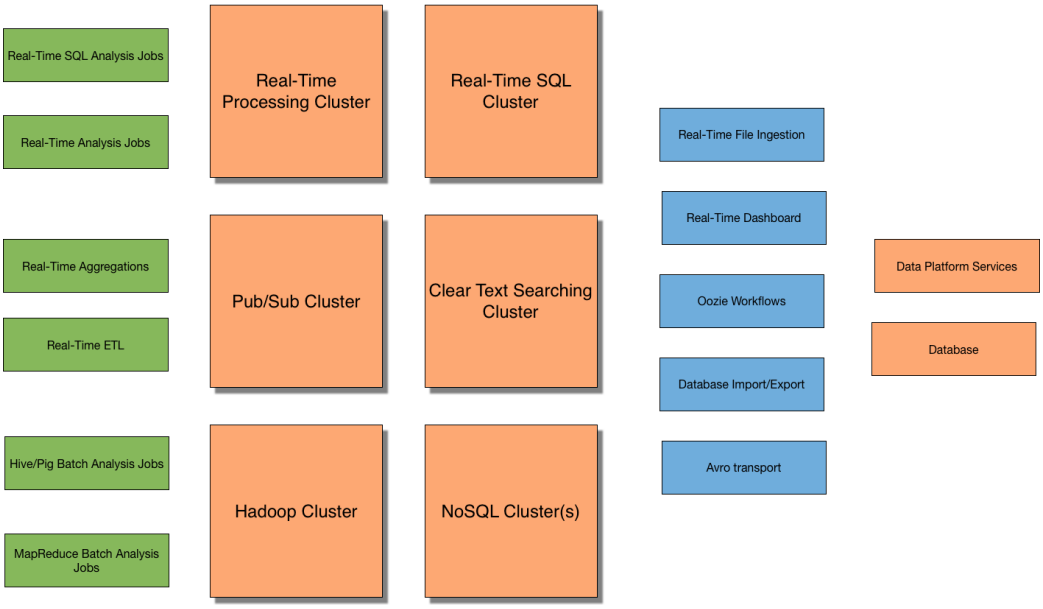

Figure 4.4: More complex architecture diagram for a Hadoop project with real-time components.

learned the pros and cons of five to 10 different NoSQL technologies. From that list, you can narrow it down to two to three for a more in-depth look.

During this period, you might compare, for example, HBase and Cassandra. Is HBase the right one or is Cassandra? That comes down to you knowing what you're doing. Do you need ACID-ity? There are a plethora of questions you'd need to ask to choose one. Don't get me started on choosing a real-time processing system, which requires knowledge of and comparison among Kafka, Spark, Flink, Storm, Heron, Flume, and the list goes on.

Distributed Systems Are Hard

Distributed systems frameworks like Hadoop and Spark make it easy, right? Well, yes and no.

Yes, distributed systems frameworks make it easy to focus on the task at hand. You're not thinking about how and where to spin up a task or threading. You're not worried about how to make a network connection, serialize some data, and then deserialize it again. Distributed systems frameworks allow you to focus on

what you want to do rather than coordinating all of the pieces to do it.

No, distributed systems frameworks don't make everything easy. They make it even more important to know the weaknesses and strengths of the underlying system. They assume that you know and understand the unwritten rules and design decisions they made. One of those assumptions is that you already know distributed systems or can learn the fundamentals quickly.

I think Kafka is a great example of how, in making a system distributed, you add 10x the complexity. Think of your favorite messaging system, such as RabbitMQ. Now, imagine you added 10x the complexity to it. This complexity isn't added through some kind of over-engineering, or my PhD thesis would fit nicely here. It's simply the result of making it distributed across several different machines. Making a system distributed, fault tolerant, and scalable adds the 10x complexity.

How Long Will It Take to Learn?

Given this level of complexity and the sheer number of technologies you need to learn. It's going to take a while to learn everything you need.

In my own experience, it took me 4 months before I really felt comfortable with all of the things I needed to know. Keep in mind that I had a decent background in distributed systems before moving over to Big Data. This background helped a great deal, but didn't completely alleviate the complexity of learning. Honestly, nothing prepared me for the sheer number of technologies I had to learn.

In my experience teaching, the average person is going to take 6-12 months to feel comfortable with everything. They'll take about 1-4 months to learn enough about the various frameworks to be productive.

Ability Gap

There's another crucial part to how long it will take you to learn Big Data technologies. Instead of a question of when, it's a question of if.

As a trainer, I've lost count of the number of students and companies I've taught. One thing is common throughout my teaching: there is an ability gap. Some people simply won't understand Big Data concepts on their best day.

Most industry analysts usually talk about skills gaps when referring to a new technology. They believe it's a matter of an individual simply learning and

eventually mastering that technology. I believe that too, except when it comes to Big Data.

Big Data isn't your average industry shift. Just like all of the shifts before it, it's revolutionary. Unlike the previous shifts, the level of complexity is vastly greater.

This manifests itself in the individuals trying to learn Big Data. I'll talk to people where I've figured out they have no chance of understanding the Big Data concept I'm discussing. They've simply hit their ability gap. They'll keep asking the same question over and over in a vain attempt to understand.

Yes, this is harsh

The technical bar for Data Engineers is pretty high. This ability gap is specific to Data Engineers where they'll have to create these data pipelines.

It isn't something that other members of the data engineering team or consumers of data pipelines will have to deal with as much. These positions can rely on the Data Engineers to help them with the really difficult parts.

Changes You'll Need to Make

Oddly enough, not every change you'll need to make will be purely technical. Some changes will be around your mindset and perception of data. The biggest change is around scale. When thinking about scale, I encourage students to think in terms of:

- 100 billion rows or events
- Processing 1 PB of data
- Jobs take 10 hours to complete

I'll talk about each one of these thoughts in turn.

When you are processing some data, in real-time or batch, you need to imagine that you're processing 100 billion rows. These can be 100 billion rows of whatever you want them to be. This affects how you think about reduces or the equivalent of reduces in your technology of choice. If you reduce inefficiently or when you don't have to, you'll experience scaling issues.

When you are thinking about amounts of data, think in terms of 1 PB. Although you may have substantially less data stored, you'll want to make sure your processing is thought about in those terms. As you're writing a program to process 100 GB, you'll want to make sure that same code can scale to 1 PB. One common manifestation for this problem is to cache data in memory for an algorithm. While that may work at 100 GB, it probably will get an out of memory error at 1 PB.

When you are thinking about long-running processes, I encourage teams to think of them running for 10 hours to complete. This thought has quite a few manifestations.

The biggest manifestation is about coding defensively. If you have an exception at 9.5 hours into a 10-hour job, you have two problems: to find and fix the error, and to rerun the 10-hour job. A job should be coded to check assumptions, whenever possible, to avoid an exception from exiting a job. A common example of these issues is when a team is dealing with string-based formats, like JSON and XML, and then expecting a certain format. This could be casting a string to a number. If you don't check that string beforehand with a regular expression, you could find yourself with an exception.

Given the 10-hour job considerations, the team needs to decide what to do about data that doesn't fit the expected input. Most frameworks won't handle data errors by themselves. This is something the team has to solve in code. Some common options are to skip the data and move on, log the error and the data, or to die immediately. Each of these decisions is very use case dependent. If losing data or not processing every single piece of data is the end of the world, you'll end up having to fix any bad data manually.

Every engineering decision needs to be made through these lenses. I teach this to every team, even if their data isn't at these levels. If they are truly experiencing Big Data problems, they will hit these levels eventually.

Not Just Beginner Skills

Big Data is complex. Organizations who are hiring for teams have been bitten by hiring people who've hit their ability gap. They hired someone with beginner skills thinking that they'd be able to progress over time. That person didn't progress due to their ability gap and the organization is forced to hire another person. Suffice it to say, the organization doesn't repeat that mistake.

Now, organizations require intermediate to advanced skills before hiring new people on the team. This is to verify that the new person doesn't have an ability gap beforehand.

Organizations simply cannot afford to hire people with the wrong skills and ability.

Many people haven't internalized this fact — they continue to think their beginner skills will get them a job. They fail interview after interview or they never get the call for the interview in the first place.

A person wanting to work with Big Data will need intermediate to advanced skills. It's important to note that just memorizing interview questions is another route people take. A competent interviewer will see this quickly.

The Hiring Manager Conversation

"Pass" he said while on the phone.

"Pass" he said again into the phone.

Very curious, I waited until he finished with his phone call. He's the hiring manager for a large company who is looking for Data Engineers. He was telling their recruiter to pass (not hire) on the job candidates they had just interviewed.

Why was he passing on these candidates? "They weren't qualified enough," he said. "They either didn't have the experience or breadth of knowledge to do the job. It's better to not even hire them."

You may be reading this and start to worry about a chicken and egg scenario. How will you get the skills without getting a job? In this case, people are equating skill with experience. You can gain and demonstrate skills without having on the job experience (see Chapter 7 "How Do You Get a Job"). Companies do realize that data engineering is new and not everyone will have job experience. They do expect that you have spent the time beforehand getting the skills through your own learning.

Understanding the distinction betwen *skill* and *experience* is the difference between getting a job and wasting countless hours applying and interviewing.

Which Technology Should You Learn?

People who are new to Big Data fall into a trap from their time in small data. They think they just need to know a stack or one technology. So, they look through the marketing materials and the blogosphere for what people are recommending. Others go on LinkedIn and ask what the latest technology is that they should learn.

The sad truth is that these people fall into another trap that wastes their time. Their perception is that everything just needs one technology. That differs from the real world at companies and their perception. The reality is that companies use **many** different technologies all working together to create a data pipeline. Even those consuming and analyzing the data pipelines will need to work with several different technologies.

The answer is that Data Engineers will need to know 10-30 different technologies. Their value is that they know the right tool for the job and that saves the organization time and money. This book doesn't go deeply into the technologies themselves, but we go a little deeper in Chapter 6 "How Can You Do It."

Hadoop Is Dead

You'll read that some technology is dead. That leads smart people to always be searching for the technology that isn't dead and they get into an infinite loop searching for what isn't dying. The reality is that every technology is dying; some are simply dying faster than others.

X is Dead

Most of these "technology X is dead" articles are coming from a vendor whose product competes with that "dead" technology. The vendor has a vested interest in furthering the perception that the technology is dead.

If you use this as a reason to not learn a technology, you'll never get anywhere. Trust me, this is a great way to never get anything done — and I see it happen **all** the time.

Hadoop MapReduce

Is there any value to learning MapReduce now? I still teach MapReduce. I find it's a great way to see how things are working in the behind the scenes and with a simpler API than others. The newer technologies like Spark and Flink are more supersets that add more functionality. Understanding MapReduce makes learning Spark and Flink easier.

Many companies have legacy MapReduce code that Data Engineers need to maintain or use. Still other companies haven't made the jump and they're waiting for the next technology after Spark to get adopted.

Future Proofing Your Career

The Big Data landscape is changing constantly. Some technologies are changing faster than others. There is no such thing as future proofing your career by choosing the right technology. The right technology will eventually become the wrong technology. You will need to spend time and effort to keep up with what's happening.

This is part of the reason that Data Engineers must specialize. They can't be a general Software Engineer anymore. They need to specialize in Big Data technologies in order to keep up with sheer number of changes and new technologies coming out every day.

Is Big Data Going to Last?

Since Big Data is changing so much, does that mean it's going to burn out like so many fads before it? Personally, I went down the path of a few fads (ahem Ruby on Rails). I know the pain and wasted time of going down a path of something that showed so much promise and burnt out faster that you can say fad.

I've spent a good amount of time looking at Big Data's life span and thinking about it. I obviously have a dog in this fight, but I did all of my analysis before I went heavily into Big Data. I redid my analysis before I started my own company

focused on Big Data. I'm also at the forefront of Big Data. Almost every week, I'm teaching a company that is either starting their Big Data journey or expanding it.

There is an obvious bias in this data — the companies that stopped or cancelled their Big Data projects aren't bringing me in to train their staff (aka survivor bias). I've found these companies stop for three reasons:

- They thought Big Data is a silver bullet that would cover over the shortcomings of their company
- They didn't have real Big Data problems in the first place, could replace Big Data technologies with small data technologies, and be better off.
- They didn't know how to run a Big Data project and it failed.

If an organization gets past these issues and they have true Big Data needs, they can't do anything else but use Big Data technologies. Think about the companies that have real Big Data needs like Google, Apple, Facebook, etc. They've never said "We don't have Big Data problems anymore. Let's switch back to these small data technologies and fire the teams." That doesn't happen because they can't go back to small data technologies and teams with only small data skills.

Companies like Google have been researching how to make Big Data easier for years. They've made it easier, but they haven't lowered the bar so much that **anyone** can do it. There's still a high technical bar for the Data Engineers who can create these systems. It's somewhat easier for these skilled people to create these systems and they don't have to write these systems from scratch anymore.

Why I Don't Think Big Data Is Going Away

In my view, Big Data isn't going away and will increase in usage because:

- When you have real Big Data problems you can't switch back to small data.
- Other organizations are seeing what's possible and wanting to mimic that success
- The supply of Data Engineers will go up, but won't become saturated because the technical bar is much higher. You're not going to wake up one day and be replaced by a bunch of Web Developers who learned Big Data technologies.
- Data pipelines aren't a one and done project. The project and data pipeline are constantly changing. Organizations will need people to continually update and evolve these pipelines.

CHAPTER 5

Switching Careers

Retooling

A good portion of my students are people who've spent several years in small data. They're looking to retool their careers and switch to Big Data.

In these scenarios, you'll need to focus on what you bring that others don't. You may bring more domain experience — and that's difficult to train a new person on. You'll have more systems and coding experience than others.

Once you add Big Data skills to existing experience, you become an asset to the organization. Most data engineering teams skew towards the senior level titles.

Other teams who aren't part of the data engineering team tend to be evenly distributed on seniority.

When Layoffs Happen

I've had the unfortunate luck of teaching while layoffs were happening. They literally sent the email while I was lecturing. I know the face of teams who've just found out that there are layoffs.

I tell them to focus on the class even more. This Big Data class is how they're going to get their next job. If you've been laid off or are feeling like layoffs are imminent, focus on your learning. The organization can fire you but they can't take your acquired knowledge. Use your existing domain knowledge and your new Big Data skills to get your next job.

Newly Graduated

While data engineering teams skew towards senior people, that doesn't mean that newly graduated people or people who are fresh out of school don't work on data engineering teams. There are fewer of these people and they'll need to improve their skills.

I've had interns and recent graduates people in my classes. They tend to have their Master's degrees in Computer Science. Those with their Bachelor's' degree had a focus on distributed systems and multi-threading.

Other teams who aren't part of the data engineering team tend to be evenly distributed on seniority and have more junior engineers.

General Advice on Switching

Reading this book will put you ahead of others. You'll actually know going in what you're up against. You'll have a much better idea of how much effort and work it will be given your existing skills.

Setting up your expectations is a major reason I wrote this book. Unrealistic expectations is a major reason I see people fail at learning Big Data. They come from their small data background and think Big Data will follow suit. **It doesn't**. Learning Big Data will be difficult and time consuming.

People who are new to Big Data think I'm exaggerating when I say that Data Engineers need to know 10-30 different technologies. This is how people don't get jobs. They know 1 technology and no one will hire them; their perception doesn't match reality. You will need to know multiple technologies because a data pipeline is made up of multiple technologies all working together.

Those who aren't part of a data engineering team but still interact with data pipelines have a lower technical bar. You will still need to know various technologies, but not at the depth a Data Engineer needs to know them.

Now, I'll give specific advice and recommendations to different positions and skills.

Programmers and Software Engineers

Programmers and Software Engineers generally become Data Engineers. These Data Engineers take their existing knowledge of programming and augment it with Big Data.

I put architects in this section too. In my experience, architects need to do some level of coding with Big Data. They may help create the PoC, but not do the majority of coding. I've found that architects trying to do Big Data pipelines without a coding background don't understand the technology tradeoffs enough to make the correct decision.

Some Software Engineers have a math or statistics background. These people are often good candidates to become Data Scientists. I've had students who've joined data science teams to improve the data science team's programming abilities, but still know the math behind the algorithms.

Data Science Team's Engineering Skills

Most data science team's software engineering skills vary from fair to absent. They're people without engineering backgrounds doing engineering. A programmer with a good engineering background is an asset to the team.

Required Skills

First and foremost, you will need programming skills. You should have at least intermediate programming skills. People who are brand new to programming struggle as Data Engineers.

You should have a general understanding of Linux. You'll need to be relatively familiar with using a Linux command line to issue commands. Most work can be done in the GUI, but some Hadoop-specific interactions are on the command line.

More helpful, but not required, is a background in distributed systems. Big Data frameworks like Hadoop are distributed systems. These frameworks make it easier to work with distributed systems, but don't completely mask all of the complexity. At some point in your journey, you'll need to learn these concepts to

really master Big Data frameworks.

Another possibility is to build on your existing multi-threading skills. If you have done cross thread and concurrency work, some of Big Data's concepts will be familiar. With Big Data, your doing concurrency on many processes spread out on many nodes instead of threads in a single process.

Which Languages?

The majority of Hadoop and Hadoop ecosystem is written in Java. You should have an intermediate to advanced level of Java knowledge. You need to understand concepts like generics, inheritance, and abstract classes.

In Java 8, the language introduced Lambda functions. Most of the new Big Data frameworks are moving to using Lambda functions throughout. It's well worth your time to learn them as they improve your Big Data code significantly.

Hadoop and the its ecosystem support other languages to varying extents.

Scala is a popular language. Since Scala is a JVM language, you can use it with any other JVM-based framework. Some Big Data frameworks are written in Scala. Apache Spark is one of those technologies.

In my experience, Data Engineers and data engineering teams are 95% Java-based. Scala is more popular with data science teams because of its dynamic nature.

Python is another popular language with Data Engineers and Data Scientists. Support for Python is improving in the various frameworks out there. Apache Spark and Apache Beam have native Python support. Python is usually supported as a quasi-first class citizen. Everything gets added and tested in Java/Scala first and then ported to Python. This means that Python will lag in support and not have access to everything.

Other languages, will work to varying degrees of effectiveness and gotchas. If you mostly program in a language that I didn't mention, check to see if your target industry or company is using that language. Otherwise, you may want to learn one of the predominant languages above.

SQL Skills

I'm seeing a push to add more SQL support in Big Data frameworks. That's because it's just easier to express some things in SQL. Joins are one great example.

To that end, I highly recommend that Data Engineers learn SQL if they don't already know it.

Cloud vs Open Source/On Prem

As you're choosing a path, you'll have to decide between learning Cloud-specific technologies and open source technologies.

Cloud-specific technologies will limit your job search to companies that are either currently using the cloud or will be using the cloud in the future. They have the benefit that you won't have to deal with or learn any of the operational sides.

Keys to Getting the Job

- Programming skills
- In-depth knowledge of the technologies to know the right tool for the job
- An awesome personal project
- A desire and interest in data

How I See People Fail

- Using the wrong materials to learn by cheaping out on learning (I'll use YouTube to learn this)
- Underestimating the complexity (I don't believe Jesse that it's really that hard)
- Underestimating or not allocating the time to learn
- A lack luster or Hello World-level personal project

How I See People Succeed

- Saving time with the right learning materials
- Having an awesome personal project with a beginning, middle, and end that you can demonstrate to the interviewer
- Showing an interest in the company's data and results during the interview
- Taking the time and effort to learn these technologies thoroughly

Managers

Why would a manager need to change anything for Big Data? Isn't it all the same thing as a small data project?

This is how I've seen so many Big Data projects fail. The management team doesn't internalize that Big Data projects need to be run and resourced differently. To give you an idea of how differently, I've written an entire book just on how teams should be skilled and how projects should be run called *Data Engineering Teams*. **Every** manager should read that book.

A small data manager is usually becoming a Data Engineering Team Manager.

Required Skills

Managers don't need to know the technology at the same depth as their Data Engineers. That said, managers need at least a cursory knowledge of the technology behind the scenes. Managers that don't understand the basics, won't understand what the Data Engineers are talking about otherwise.

A technical background is helpful, but not required for this position.

Keys to Getting the Job

- Knowing the skills that need to be on a data engineering team
- Knowing how to run a Big Data project
- A cursory understanding of the technologies you'll be using
- Reading my *Data Engineering Teams* book

How I See People Fail

- Thinking that running a Big Data project is just like running a small data project
- Trying to get to the same knowledge level as the Data Engineers
- Not gaining at least a cursory level of understanding
- Not listening to their Data Engineers on technical decisions

How I See People Succeed

- Truly internalizing how Big Data is different

- Knowing the skills on a data engineering team and knowing how to do a skills gap analysis (covered in *Data Engineering Teams*)
- Having concrete ways that the team can be improved
- Not being afraid to make tough choices on the team (some team members may have an ability gap)

Data Analysts and Business Intelligence

Data Analysts and Business Intelligence Analysts are usually consumers of a data pipeline. As such, they're usually not members of the data engineering team.

Sometimes, Data Analysts and Business Intelligence Analysts are part of the data engineering team. I've seen this happen when the company is small or when an organization's analyst team isn't technically proficient enough to write/code what they need.

The Difference Between a Data Scientist and Analyst

I'm often asked what the difference between a Data Scientist and Data Analysts/Business Intelligence Analysts. Both of those positions have backgrounds in math.

They differ in two key areas.

First, a Data Scientist has better programming skills than an analyst. These usually range from intermediate to those of a Data Engineer. They predominantly use Python and Scala. They know how to use Big Data frameworks to run their code and models at scale.

Second, a Data Scientist will know how to apply their statistical background to problems such as machine learning.

It is a common progression for Analysts to improve their programming skills and learning Big Data frameworks to become Data Scientists.

Required skills

Whether they're part of a data engineering team or not, Data Analysts and Business Intelligence Analysts need to have the technical skills to use the Big Data frameworks. The technical bar is lower and the Data Engineers have created the data pipeline. You will need to have enough technical knowledge to run your analysis.

A common question is if Analysts need to learn to program. I highly suggest that Analysts learn to program. This will set you apart from other people going for the same position.

Python is a language that's commonly used by Analysts. It has expanding support in the Big Data ecosystem.

R is another popular language with analysts. Its support in the Big Data ecosystem is emerging. Most projects will have little to no support of R.

SQL Skills

I'm seeing a push to add more SQL support in Big Data frameworks. That's because it's just easier to express some things in SQL. It's also easier for Analysts to query the data themselves. I highly recommend that Analysts learn SQL if they don't already know it.

Keys to Getting the Job

- Understanding the Big Data technologies you'll be using
- Having either SQL or programming skills
- A personal project that shows your analytic and technical skills
- A true interest in data and finding newfound insights in data

How I See People Fail

- Not understanding the technology behind the scenes
- Thinking that the analysis/math is the only hard part (the technology is just as hard)
- Not learning the Big Data technologies aimed at Analysts
- A personal project that doesn't show insights that you'd expect a good Analyst to find

How I See People Succeed

- Having SQL or programming skills (ideally both)
- A personal project that shows a true understanding of the domain and the insights you found
- A good knowledge of the Big Data technologies for Analysts like Apache Impala, Hive, Spark SQL, etc
- Being able to compliment a data engineering team with your analytics background

DBAs and SQL-focused Positions

In this section I discuss the positions that primarily focus on SQL. These are titles like:

- DBA
- Data Warehouse Engineer
- SQL Developer
- ETL Developer

For ease of reading, I'll collectively refer to these titles as DBAs.

Of all the careers, DBAs are facing the biggest crises from Big Data. They're faced with a changing landscape of technologies. Data used to be their purview and now they're finding a brand new team and title emerging, data engineering team and Data Engineer.

DBAs are faced with a difficult decision. They'll need to dramatically increase their technical skills to become part of a data engineering team. By learning to program and learning the Big Data frameworks, I've seen some become Data Engineers. I've seen some DBAs put the time and effort into their technical skills and handle the analysis part of a data engineering team.

Other DBAs will go into more of an operations role outside the data engineering team.

All of this boils down to your programming skills. Programming skills are the determining factor when a DBA is figuring out to go into operations or join a data engineering team.

The Traditional DBA

The traditional DBA role as we know it is gradually decreasing. The title isn't going away completely, but we're going to see a gradual decrease in the size of tradition DBA teams. Part of the reason for sizable DBA teams was that we were using the wrong tool for the job.

Doing a full table scan brings the RDBMS to its knees because we didn't have a better choice. With Hadoop and Spark, we have systems that are purpose built to do the equivalent of full table scans with ease.

This isn't to say that DBAs or RDBMSs are going away. Rather, their use for the wrong jobs will diminish. That will, in turn, reduce the number of DBAs required for the job; instead of a team of 10 DBAs there will be 5 DBAs on the team. Those 5 fired/laid off DBAs are faced with the dilemma of what to do.

I strongly encourage all DBAs to start learning Big Data frameworks now.

Required Skills

You should have advanced SQL skills. There are technologies in the Hadoop ecosystem that use SQL. However, these technologies aren't enough to create a full-fledged data pipeline. Being limited to SQL as your only means of querying is inherently limiting. It leads to data pipelines that don't really accomplish their goals.

I highly suggest you learn how to program. DBAs often get in an infinite loop figuring out which programming language to learn. For those coming from a PL/SQL background, Python is a general recommendation. If you're wanting to learn the most widely used language, Java is my recommendation. Learning to program isn't the memorization of APIs; it's the application of an API to solve a problem.

I find that DBAs fill the schema skill on a data engineering team the best. DBAs have spent their careers dealing with the manifestations and requirements of

schemas. Programmers, on the other hand, don't understand schema as often.

It's critical that DBAs learn the Big Data frameworks. They may not need to learn the systems as deeply as a Data Engineer, but they still need to understand the systems. These new systems aren't RDBMS with a few differences; they're completely different systems with no RDBMS corollary.

In the Big Data space, DBAs will often gravitate to the NoSQL databases. I've worked with DBA teams who've designed and implemented NoSQL databases without really learning the systems. The projects tend to fail because they're implemented like a RDBMS and that's a recipe for disaster.

There are some WYSIWYG-style (What You See Is What You Get) frameworks for Big Data. They layer on top of the existing Big Data frameworks and they vary in maturity and price.

WYSIWYGs

I don't have enough data points to comment one way or another. My initial data is showing that you still need to understand the systems underneath. A WYSIWYG helps you connect and do some pre-canned processing. It doesn't help you design or create the data pipeline.

It also remains to be seen how often you have to write some of your own code to get things done. The examples I've seen left the DBA with weird and inefficient work arounds to get things done.

Keys to Getting the Job

- Filling the data engineering team's schema and domain knowledge requirements
- Learning Hadoop and the Hadoop ecosystem
- Understanding and learning about the NoSQL ecosystem
- Learning to program

How I See People Fail

- Thinking Data Warehousing is the same level of complexity as a Big Data pipeline
- Not increasing their knowledge of these new Big Data systems
- Thinking that just RDBMS knowledge will get them on a data engineering team
- That every problem can be solved with SQL

How I See People Succeed

- Changing their mindset about Big Data
- Getting serious about coding
- Showing your coding skills with an awesome personal project
- Deeply understanding NoSQL and when a team should be using it

Operations

Operations are usually the maintainers of the clusters. They're tasked with keeping the various processes running and maintaining the health of the nodes. They serve as the first line of troubleshooting when something goes wrong in the cluster.

Even if you're doing everything in the cloud, I still suggest teams have at least one operations person. They may not be part of the data engineering team, but they would be assigned cluster maintenance as their primary task.

Sometimes, I see operations people who are part of a data engineering team. This happens when the team or organization is doing a DevOps model. In DevOps, I find that the team members are more on the operations spectrum than on the development side.

Doing both operations and development puts DevOps Engineers in a difficult situation. They have to know both the operational parts and API parts of the system. This can be a massive undertaking.

Required Skills

You will need to learn Big Data frameworks from an administrator point of view. These frameworks require daemons to be running. There are a plethora

of configuration properties you will need to know and tune. Often, operations is tasked with the actual running of the jobs and queries that the Big Data framework processes.

Hadoop runs on Linux. The majority of clusters run on RHEL/CentOS or Ubuntu. The Big Data nature of things will stress things in ways you might not have seen before. It will expose weird problems you only see at scale. To diagnose and fix these issues, you'll need to be very good with Linux, especially from the command line. Most of the computers in your Hadoop cluster will be sitting in a data center's rack or in the cloud.

Some of the Hadoop companies like Cloudera and Hortonworks are making cluster administration easier with web-based GUIs. These will help in detecting and monitoring Hadoop clusters. Despite these programs, you'll still need to know how to troubleshoot a computer with a Linux command line.

If you're planning on administrating an enterprise cluster, you'll probably be dealing with security. This includes everything from authentication with Kerberos, to line encryption, to at rest encryption. It's the administrator's job to set all of this up and keep things secure. Security is becoming a key part of operations as hacks expose entire data pipelines.

Some Big Data technologies highly benefit from a DevOps model. Apache HBase, for example, is one of those technologies. In order to really be successful with HBase, the team needs to have equal parts operations and programming knowledge.

SQL Skills?

Operations teams don't need SQL skills, but it definitely helps. You don't need to know every operation and feature, but I suggest you have some basic `SELECT` statements.

Cloud vs Open Source/On Prem

Cloud does not eliminate operations. It reduces the number of people you need for operations. I would never let 90% of the developers I've taught near a production system.

As you decide to learn cloud technologies or open source technologies, remember that they're inherently limiting. Between the options, learning open source

gives you the most possibilities.

Keys to Getting the Job

- Knowing how to set up a cluster from scratch
- Knowing how to troubleshoot issues whether they're hardware, custom software, or the framework itself
- In-depth knowledge of Linux
- Knowledge of the operations on a wide breadth of Big Data technologies

How I See People Fail

- Thinking that managing a Big Data framework is easy
- Failing to understand the various issues of running a 100+ node cluster
- Thinking that maintaining a cluster just means making sure the hardware and network are working
- Thinking that cloud means no operations

How I See People Succeed

- Having excellent troubleshooting skills
- Showing demonstrable mastery of Linux, especially from the command line
- Being able to operate a cluster both from the Web GUIs and command lines
- Providing excellent first level support so that only the right things get escalated

What if You're Not on this List?

In the previous list of titles and positions, I tried to capture 80-90% of what I've encountered in the real world. That means that I didn't mention everyone. I'll try to give some general suggestions here.

I've taught a number of Physicists. They have a background in data and doing large scale data processing. Often, they'll become Data Scientists after improving their programming skills and learning the Big Data frameworks.

Other times, people don't come from a computer science background. I've had a few Electrical Engineers learn programming and then learn Big Data

technologies. They've said some of the electrical engineering concepts are very similar to the Big Data concepts.

A few people have a general math background. They're going for an Analyst or Data Science position. Once again, they'll need to improve/learn programming and then learn Big Data.

Keys to Getting the Job

- Clearly identifying which position they're going after
- Figuring out what skills they have now and what skills they'll need eventually
- Learning and applying the skills they need to get a job
- Continuing to learn and improve those skills

How I See People Fail

- Thinking that programming is writing out equations in code
- Thinking that the technical skills don't matter
- Failing to learn to code well before moving on to more complicated problems
- Taking an honest look at your abilities and skills when self-evaluating

How I See People Succeed

- Taking enough time to get to an intermediate-level programming skill
- Finding the best learning resources to guide them
- Being honest about your skill level during an interview
- Having a personal project that showcases your skills and how your different point of view will compliment a data science or data enginering team

CHAPTER 6

How Can You Do It?

General Learning

There's a common misconception that because the Big Data frameworks are open source, everything else is free and open source. When it comes to learning, the best resources aren't free. I find that people who try to go the free route eventually stop learning because they don't progress and just stop trying.

Your choice of learning materials directly affects the amount of time and wasted effort you'll have. There are no fact checkers on the internet. I find that many of the "Learn Fast" or "Learn Cheap" sites are only cheap and light on the learning. They waste your time with either completely wrong or really bad materials.

The goal of this guide is to get you a job doing Big Data. Learning about Big Data for pleasure is one thing. Learning enough to get a job on a data engineering team or on a team using Big Data is another level all together.

Learning Big Data to get a job means that you can't just learn the basics. You will need to find a learning resource that teaches the foundational concepts **and** the advanced materials. Companies don't hire people with an introductory knowledge; they want people with advanced skills, but not necessarily advanced experience.

Will These Materials Get You a Job?

If there are testimonials or comments for the materials, read through them. Do they say "great video" or "thanks!"? These are materials that will waste your time. If the testimonials say "I got a job," those are the materials that will get you a job too.

If a learning resource doesn't go deep enough for you to get a job, you're wasting your time. I see this scenario too often. Someone is trying to go the free or cheap

route to learning and getting a job. The problem is that they spend all of their time trying to find and understand poorly written and instructed videos. If they had just spent the money up front for world class materials, they would have been much better off.

Let's go through an example. Say you could make $20,000 per year more with Big Data skills. If you go the free route and it takes 9 months, you will only make an extra $5,000 and lose out on $15,000. Those free materials will likely only give you a beginner-level knowledge and your job search will be very difficult to impossible. You run a real risk of failing to get a job. If you went a premium route and it takes 2 months, you will make an extra $16,600 and lose out on $3,400. Those premium materials cost $5,000 and give you an advanced-level knowledge and making your job search much easier. After paying for the class, you will have made an extra $11,600 and saved 7 months of your time.

How Much Is Your Time Worth?

One of the biggest mistakes I made during my career was not valuing my time correctly. I would spend hours and hours toiling with free resources because they were free and I was too cheap to pay for them. I would have saved so much time and earned significantly more had I used better resources.

Let's talk about the most common ways of learning Big Data.

Books and Blogs

Books are one of the most common ways to learn Big Data frameworks. This is how I personally learned some of the Big Data frameworks. I had a considerable background in distributed systems and network programming that helped me. I wasn't learning the concepts from scratch; I was using the books as more of a reference guide than anything.

I think that's where most books have a problem with Big Data. Few books really teach the concepts. Most are there to serve as a reference guide and to show advanced pieces that aren't used as often. The people who can pick up a book and learn Big Data are few and far between.

Blogs cover a wide range of topics. I find that blogs are best for a single concept

or a deep dive into a specific use case. That's how I write my blogs. They'll dive deeply into something specific. In order for you to make any sense of the blog, you'll need a deeper foundation. Without this foundation, these blogs don't make sense.

After getting a solid foundation, these blogs serve as a great tool to learn about new features and use cases.

Books and Blogs I Like

Once you have your fundamentals, here are some blogs that I like:

- http://www.jesse-anderson.com/
- https://www.confluent.io/blog/
- https://blog.cloudera.com/
- https://hortonworks.com/blog/

From O'Reilly, I like the entire *The Definitive Guide* series. These aren't great books for learning the fundamentals, but they're great for references and going deeper into a specific topic. The authors are top notch too.

For learning languages, I like Pragmatic Programmers books on languages. They're to the point and have great voice.

Twitter is another great way to keep up on what's happening. Most of the discussion about what's happening is on Twitter. I generally digest all of this on my @jessetanderson account.

Classes

There are a plethora of virtual and in-person classes. The quality of these classes varies dramatically. A class can be broken down into two parts: the course materials and the delivery/instruction.

I've seen some course materials that were just copying and pastes of the Apache documentation (which is terrible). I've seen course materials that were woefully

out-of-date. Still other materials will never compile or are so simple you'll be left with a Hello World level knowledge.

The instructor has a great deal to do with the quality of a class. The instructor should specialize in Big Data — otherwise, they won't really know what's happening and what's coming. The instructor should be able to answer your questions, whether they are code or conceptual.

An average person won't be able to select the best training. Just know that there's a *big* difference in quality between a $2,000 per person class and a $3,000 per person class. The cutting corners extends to the course materials and the instructor.

Virtual Classes

I only specialize in Big Data courses. You can see my courses at http://tiny.bdi.io/courses.

Boot Camps and Intensives

I've taught boot camps and interacted with people who've come out of bootcamps. The major issue is being jobless for a while.

I taught a bootcamp that was 2 months long. My students came out of it with jobs as Data Engineers and Data Scientists. That's not to say these people didn't take a big risk quitting their jobs and having to find a new one.

In Their Own Words

I waffled for a while because I realized I would have to quit my job and look for another after the immersive. In the past, that search took 6 months. I finally took time to talk about this with some friends and decided to take the class. The journey was interesting. After the class, I kept studying. — Stephan W.

The upside to these intensives is the level of interaction you get. As opposed to short term classes, you will have months of time to interact with an instructor.

The downside to these intensives is the cost. It's expensive to have that much access to an instructor. These courses start at $20,000 or so.

Online Courses

Online courses vary greatly in quality. Before you take the plunge, you'll want to be very certain that you've chosen the right one for you. There are all sorts of companies seeking to cash in on the Big Data market. That means they'll put anything out there.

Look at the testimonials and comments for the product. Do the people say that they got a job or that they liked the course? When someone says that they liked the course, that means they learned passively and didn't get a job. If they say they got a job, they were actually learned and applied the materials to accomplish their goals. There is a big difference in price and quality between a passive learning course and a course that teaches you how to do a new job.

Shameless Plug

I have an entirely online course called *Professional Data Engineering*. It is about 8 weeks long and guides you through the technologies you need to know. To make sure people people are completely happy with it, I give a 60 day unconditional money back guarantee.

You can purchase it at http://tiny.jesse-anderson.com/pdesales.

MOOC

Massive Online Open Courses (MOOC) are massive online classes. They can have 1000s of students. These courses vary even more in quality. Since they're attracting so many students, they can offer super cheap prices.

These classes focus on introductory materials. Any interaction with the instructor is limited or nonexistent. This leads many students to get stuck and never progress in their learning. Since they're so cheap, there's also an 'I'll do it later' trap that people fall into.

YouTube/Free

This is a route I see many people taking and I don't understand why. I have a YouTube channel at http://tiny.jesse-anderson.com/youtube where I show videos of Big Data concepts. My channel gives you some concepts in 5-10 minute videos.

I've went through many of the other Big Data videos on YouTube. They cover some coverage of basic concepts. Most of these videos are total waste of time because they're either incorrect technically or too high level.

If you're starting out with Big Data, learning basic concepts won't get you a job. You're going to need more substantial learning materials.

No Really, Does It Work?

One of my goals for this guide is to keep you from wasting your time and getting discouraged. I think a YouTube-centric route is exactly that. In preparation for writing this section, I went through the people who'd responded to surveys or emailed me directly saying that they're using YouTube as their primary learning resource. I compared their titles on LinkedIn to see if they've actually switched to a Big Data role. None of these people accomplished their goal.

This isn't to to say YouTube is a bad resource. I have some advanced videos on my channel, but without a solid foundation, you're not going to understand what I'm talking about. A YouTube-centric approach doesn't give you a solid enough foundation to keep building.

Learning to Program

In chapter 5 "Switching Careers," I talked about the need for some titles and positions to either learn to program or advance their programming skills.

To be honest, Big Data won't challenge your syntax and knowledge of the language. In Java, for example, you're not going to be using the `synchronized` keyword, but you might be using the `transient` keyword.

You're not going to be making extensive use of design patterns or heavy lifting with a repository pattern. You will be making extensive use of system architectural patterns.

For Data Engineers, you should have intermediate to advanced programming skills. You should know at least one compiled language like Java and one dynamic language like Scala or Python.

For other positions that need programming skills, you should have beginner to intermediate skills.

Which Technologies to Learn?

You've read that people interacting with data pipelines need to know 10-30 different technologies. Here is just a small example of what you should know:

- Apache Hadoop
- Apache Spark
- Apache Hive
- Apache HBase
- Apache Impala
- Apache Kafka
- Apache Crunch
- Hue
- Apache Oozie
- Apache NiFi
- Apache Flink
- Apache Apex
- Apache Storm
- Heron
- Apache Beam
- Apache Cassandra

Just Apache?

You'll notice that this list is made up of mostly Apache projects. You are correct. In my dealings with organizations, they're either using or moving to Apache projects. Even the ones that aren't Apache projects are Apache licensed.

This isn't the exhaustive list. These are just some of the high points that you should know. Also, I'm also not including the technologies that are still in use, but are dying a slow death.

How do you figure out which ones you should know? How do you understand how all of these fit together? How do you make a data pipeline with all of these technologies? You're going to need a guide. This guide will have to be an expert.

Which One Is Better?

Which is better Apache Hadoop or Apache Spark? Which is better Apache HBase or Apache Hive? These questions come from a small data mindset that each technology is interchangeable and does about the same thing. There are several direct answers to these questions:

- The choice of technology is entirely dependent on the use case.
- Some of these technologies are complimentary and don't do the same thing. You may need to use both of them in the same data pipeline.
- You should learn all of them to choose the right tool for the job.

Over Analyzing

A common issue when starting out is to spend all of your time trying to figure out what the next big thing will be? Which one is the single technology that's going to get you a job. You get too focused on the minutiae and never make any progress. The answer is that you'll need to know a little to a lot about **every** one of these technologies. For Data Engineers, it will be a lot. For other teams, it will be a little with a focus on certain technologies.

Can You Do This At Your Current Organization?

Your organization may have Big Data needs. It's critical that you establish that your organization **really** has Big Data needs. I've seen too many people start using Big Data technologies just to get that on their resume. That's not a good idea.

Once you've established the need, you may be able to convince your boss of the need for you to become a Data Engineer. You will still need to make sure you get the skills, but you won't have to get a new job.

Can You Do This Without a Degree?

You don't need a degree in Computer Science or a Master's degree to start using Big Data. I've taught people with all different education levels. They've ranged from university professors to completely self-taught. If you show the skills with a personal project, you can do it.

My Education

This may surprise you, but I'm completely self-taught. I've never attended a single college class. The only time I've spent at universities is to guest lecture on Big Data.

Can You Do This in Your Country?

I've taught and worked with people from many different countries. The majority of countries are experiencing Big Data problems. If you're worried that your country doesn't have Big Data jobs, take some time to look at job postings and ask around. Chances are, you'll be surprised to find that someone or an entire company is doing it.

Countries Accessing My Blog

To help you understand which countries you'll want to double check, I took a look at the countries that have accessed my blog over the past 3 months. This should give you an idea of where interest in Big Data interest lies since my blog is entirely about Big Data.

In North and South America, only Paraguay, Guyana, French Guiana, and Suriname don't have hits. In Europe, Asia, and Australia these countries don't have hits Papua New Guinea, Mongolia, Tajikistan, Kyrgyzstan, Turkmenistan, Yemen, and Greenland. In Africa, about half the countries have hits.

How Diverse Is Big Data?

I find Big Data to be more diverse than other specialties in technology. I really do want to see that improve.

To make that happen, I have a scholarship that is open to African-Americans, Hispanic-Americans, and Native Americans. You must live in the United States. If you meet these qualifications, go to http://tiny.jesse-anderson.com/diversity to apply.

CHAPTER 7

How Do You Get a Job?

Your goal should be to get a job in a data engineering team or a team that's using Big Data. Every chapter, section, and tip in this book is leading you up to this goal.

If you aren't making specific and appreciable progress, you are not going to meet your goal. You have the goal of getting into Big Data, but aren't willing to put in the time, money, and effort into actually doing it.

You're the person who's sending out resumes to Big Data companies and never getting a call back. Or you're getting the interviews, but failing miserably.

What's happening? You have to come in with a full stack. The first person looking at your resume will verify that you have the right frameworks. For example, if you just learn Apache Spark, the person looking at your resume is going to look at that and say "where's the rest of the stack?"

On the other hand, my students have put the work in beforehand. They've learned the technologies that are actually used at companies and in production. They get the jobs.

No Experience Necessary?

Other students try and fail during interviews and chalk it up to a lack of experience. How does someone who's brand new get experience? You have a chicken and egg problem.

The real root of the problem is that you have to show skill and not necessarily experience. You can show skill without experience with an awesome personal project.

Experience, Skill, and Personal Projects

I verify this assertion when I teach at a company. I ask the hiring manager, "if someone has no experience, but they can show skill with a personal project, will you hire them?" The resounding answer is yes they will. The key is that **you** have to show skill if you lack the experience.

Where Do You Fit in the Data Engineering Ecosystem?

Not every person and position will be a fit on the data engineering team. Some positions will be on a team that is consuming data from the data engineering team.

You will need to take a very honest look at where you fit. If you're looking at becoming a Data Engineer, the technical bar will be much higher for you than a person on a different team. Either way, you will need the Big Data skills to use the data.

Once you've figured this out, you can closely target the position. This involves creating a plan to acquire the skills and technologies to get this position.

Personal Project

How do you show skill without experience? You do it with a personal project. You saw several references to personal projects in Chapter 5 "Switching Careers."

A personal project is a project where you show your skills in creating something with the relevant Big Data technologies.

A great personal project takes away all of the "can you do it?" questions because you did it all yourself. This personal project should have the full source code on GitHub (the interviewers may or may not look it over). Another key is to demonstrate this personal project. You should be able to open your laptop and show it fully running. This could be running on a VM on your laptop or perhaps on a cluster in the cloud. If you do go the cloud route, make sure you have internet access during the interview.

Some people try to do a Hello World as their personal project. That doesn't show

any skill and is worse than nothing. It actually shows that maybe you don't really have the skills and can only accomplish Hello World. If you are applying for a senior position, your personal project should show the skills and code of a senior engineer. It should show some creativity and mastery of the technology.

Datasets

One of the more difficult parts of a person project is to find a dataset that interests you. I have a list of unique and interesting datasets on my blog at http://tiny.jesse-anderson.com/datasources.

What Have Personal Projects Done For Me?

Personal projects have been a game changer for me. As a direct result of awesome personal projects, I've gotten job offers and jobs. These aren't just job offers from unknown companies, but from well-known companies like Google, etc.

I've also received extensive notoriety. I've been in the Wall Street Journal, CNN, BBC, NPR, and virtually every tech media.

Why Didn't Your Personal Project Work?

More than likely your project wasn't compelling and didn't show any technical prowess. I've followed up with people who've looked through others' personal projects. The common thread was that they were too simple, boring, or didn't show any creativity. They looked like they were thrown together at the last minute instead of the meticulous planning and execution someone would really put in if they wanted a job.

Should You Get Certified?

People often ask me if they should get certified. I've talked to many hiring managers about this. Managers say they generally distrust certifications. They

all know that they're just a multiple choice test and don't show any true mastery. They also wonder if the person took the test themselves or cheated in some fashion to pass the test.

There is a trend towards certifications that require coding or doing something specific. These are generally considered more prestigious, but chances are the hiring manager won't know anything about them.

Sometimes a certification will get you in the door for an interview. If companies are involved in government contracting, they'll often favor certified people.

Are They Worth It?

In my experience and conversations, most certifications just don't make enough of a difference to be worth it. You'd be better off putting your time, and money into a better personal project that shows true mastery.

Networking

People forget the personal touch. If you want to to be sure that your resume isn't deleted by some person in HR who doesn't know what's going on, then you need to meet the hiring manager in person.

Chances are, the manager or a member of their team attends the local meetups. Attending these meetups gives you the opportunity to meet and interact with these people. They'll see that you have the technical skills and a genuine interest to work at their company. This is something that is difficult to convey in an email or phone call.

Referral Bonus

Companies will give employees who refer a candidate that gets hired a referral bonus. Be sure to ask them if they'll pass on your resume. If you made a good impression, they will pass it on for the referral bonus.

Not So Fast!

At one meetup, the presenter didn't show. The organizers said 'no talk but stay and enjoy the food'. Being the kind of guy I am ... I stood up and said - Not so fast! Please if anyone is hiring or wants to be hired ... meet over here. I got 3 pings and I followed up on one at [company name]. — Stephan W.

Conferences are another great place to meet people. They'll often have job boards filled with positions. Better yet, the hiring managers are often attending. Take the time to talk to them and understand what their company is doing. This is one of the best ways to get a job.

Getting a Job Fast(er)

I've talked about the need to really learn and plan ahead of time so that you can pace yourself. Sometimes, you just get blindsided by a layoff. You'll need to dust yourself off and decide if this is a good time to switch careers over to Big Data.

Here is the advice I've given to others in that same situation. Now that you're not working, you need to spend that time learning and preparing for job seeking. Here's what I'd do:

- Update your resume to focus on the impact your work had on the organization. Your resume shouldn't say things like:
  `Analyzed data with SQL`
  It should say:
  `Found a way to reduce costs by 20% by analyzing our logistics data`
  There's a night and day difference between these two resumes.
- Spend as much time as possible going through your learning materials. Yes, you may have to buy expensive materials to make this jump.
- Practice interviewing with a friend. Have them ask you really hard questions and not just programming questions. They should be questions like "Why do you want to work here?" to "Why did you leave your last position?" Practice the answers to these questions.
- Make a kick ass personal project. Put the full code in GitHub. Make sure that it's easy to pull up with the VM on your laptop so you can show it

actually running. Another option is to sign up for Google Cloud or Amazon Web Service and have it running there. The main points are that you need to show your code and that it works.

- Network at local meetups and conferences. Practice beforehand how you're going to introduce yourself and the position you're looking for.

You'll notice the theme through all of these tips and strategies is to practice beforehand!

CHAPTER 8

What Are You Going to Do?

Questions to Answer Before Starting to Learn Big Data

These are questions you should think about before you start switching careers to Big Data:

- What is your specific goal for learning Big Data? Do you want to switch careers and get a new job? Do you just love to learn new things?
- Where will you be in relation to the data engineering team? Will you be in the data engineering team or using a pipeline created by the data engineering team?
- Honestly, how long will it take you to accomplish this goal of learning Big Data?
- Do you think you're giving yourself enough time?
- What source(s) are you going to use for your learning materials?
- Do these learning materials actually give you the Big Data skills you need? Did others say the learning materials helped them accomplish their goal?
- How are you going to stand out from the crowd?
- What do you think would make for an awesome personal project?
- What meetups and conferences happen in your area to network?

Your Checklist For Starting to Learn Big Data

These are specific items that you should know the answer to **before** you start switching careers to Big Data:

1. **Have a clear and attainable goal for switching to Big Data**
   Example: You are going to switch careers to Big Data and get a job as a Data Engineer.
2. **Identified the skills you need to acquire**
   Example: You need to improve your programming skills from an intermediate level to an advanced level.

3. **Identified the technologies you need to learn**
   Example: You are learning Apache Hadoop, Spark, Kafka, etc.
4. **Purchased the materials that will teach you the skills and technologies**
   Example: You have purchased Professional Data Engineering from http://tiny.jesse-anderson.com/pdesales
5. **Created a realistic and maintainable schedule to learn**
   Example: You are going to take 10-20 hours a week for the next 2 months to learn.
6. **Started thinking about your personal project**
   Example: You're looking for interesting datasets to see how they could be augmented. You're looking for ways to wow a company with your Big Data mastery.
7. **Started to network with others in the Big Data space**
   Example: There is a local meetup that you are going to attend to get a feel for who is hiring and what they're looking for.

Parting Advice

By reading this book, you've already done more preparation than most people. My goal for this book was to help you decide if switching careers to Big Data is right for you, show you the steps to switch, and how to get a job once you're ready.

In Chapter 2, I outlined the 4 ways I've seen people succeed:

- A desire to learn Big Data
- Some of the prerequisite skills
- The time to dedicate to learning
- An expert (or experts) to guide through the experience

You can switch careers to Big Data if you follow these 4 steps. Don't waste your limited time on things that get you nowhere. Do make a plan and stick to it. Most of all, make an awesome personal project and blow your interviewers away. Always keep up to date because this field is changing rapidly.

Best of luck on your Big Data journey.

About the Author

Jesse Anderson is a Data Engineer, Creative Engineer and Managing Director of Big Data Institute.

He trains at companies ranging from startups to Fortune 100 companies on Big Data. This includes training on current edge technologies and Big Data management techniques. He's mentored hundreds of companies on their Big Data journeys. He has taught thousands of students the skills to become Data Engineers.

He is widely regarded as an expert in the field and his novel teaching practices. Jesse is published on O'Reilly and Pragmatic Programmers. He has been covered in prestigious publications such as The Wall Street Journal, CNN, BBC, NPR, Engadget, and Wired.

CHAPTER A

Appendix: Are Your Programming Skills Ready for Big Data?

This appendix is just for those of you who will be programming. This is especially relevant for the Data Engineers and Data Scientists. You will need to make sure that you're ready to program.

The question arises — how good does a person's programming skills need to be? This is because programming skills are on a wide spectrum. There are people who are:

- Brand new to programming
- Never programmed before and will have to learn how to program
- Program in a language other than Java/Scala/Python
- Been programming in Java for many years

Another dimension is your role on the team. For example, a Data Engineer will need far better programming skills than a Data Analyst or a Data Scientist.

Usually, people with a solid Java and Scala background will have their programming skills ready. For other people with the a solid non-Java background, they will need to learn enough Java to get by. As you'll see in the next sections, the Java code is not the most complex syntactically. The programming side does give those who are new to programming or have never programmed the most difficulty.

Example Code

To give you an idea of what some Big Data code looks like, here is an example `Mapper` class from my Uno Example.

```
public class CardMapper extends
    Mapper<LongWritable, Text, Text, IntWritable> {
```

```
    private static Pattern inputPattern = Pattern.compile("(.*) (\\d*)");

    @Override
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String inputLine = value.toString();

        Matcher inputMatch = inputPattern.matcher(inputLine);

        // Use regex to throw out Jacks, Queens, Kings, Aces and Jokers
        if (inputMatch.matches()) {
            // Normalize inconsistent case for card suits
            String cardSuit = inputMatch.group(1).toLowerCase();
            int cardValue = Integer.parseInt(inputMatch.group(2));

            context.write(new Text(cardSuit), new IntWritable(cardValue));
        }
    }
}
```

You'll notice a few things about this code.

The class is relatively small and self-contained. This is true for most Big Data code, even production code. This is because the framework, in this case Hadoop MapReduce, is doing all sorts of things behind the scenes for us.

We also see that we're using regular expressions to parse incoming data. It's very common to use regular expressions in processing data. They're so necessary, that I cover them in my Professional Data Engineering course.

You'll also notice this class isn't doing anything exotic. The code and syntax itself isn't going to stress most people's knowledge of Java. The closest thing to exotic is the occasional use of the transient keyword. In this sense, Big Data knowledge of syntax can be intermediate.

As you just saw, the programming side is necessary, but not extremely difficult. You will need to know how to program. I've seen people come from other languages without significant difficulties.

What Is Difficult Then?

There are two main difficulties for the programming side. They are understanding the framework and the algorithms you need to write when creating a distributed system.

The Framework

Looking back at the code above:

- How does the map function get data?
- Where does the key's data come from?
- Where does the value's data come from?
- What happens when you do a write?
- What should you use for your output key and value?

Some of these questions are answered by knowing and understanding what the framework is doing for you. In this example code, Hadoop MapReduce is doing several things for you. What should come in and out of the map function is dependant on what you're trying to do. At its core, you need to have a deep understanding of the framework before you can use it or code for it.

This lack of realization of where the difficulty lies is a common issue for people starting out with Big Data. They think they can use their existing small data background and not make a concerted effort to learn Big Data technologies. They're dead wrong in this thinking and this causes people to fail in switching careers to Big Data. I talk more about these necessary realizations and what to do in my book The Ultimate Guide to Switching Careers to Big Data.

The Algorithms

With Big Data, you're doing things in parallel and across many different computers. This causes you to change the way you process and work with data.

As you saw in the code above, you will need to decide what should come in and out of your map function. But how do you this in a distributed system?

A simple example of the difference can be shown with calculating an average. Let's say we want to calculate the average of these numbers:

```
88 91 38 3 98 79 3 31 23 61
```

On a single computer, that's easy. The answer is to iterate through all 10 values and the answer is 51.5.

Now let's distribute out the data to 3 computers.

Computer 1: `88 91 38`

Computer 2: `3 98 79`

Computer 3: `3 31 23 61`

Now, we run the averages on all 3 computers.

Computer 1: `72.3`

Computer 2: `60`

Computer 3: `29.5`

But we don't have an average of the dataset. We average out the results from all three computers to get 53.94. Now we're off by 2.44. Why? Because an average of averages isn't correct.

In order to distribute out data and run an algorithm in parallel, we need to change the way we'd calculate the averages.

Are your programming skills ready?

The answer comes down to your current programming skill in Java and what your position on the team will be. If you looked over the code and readily understood it, you're probably ready. If you struggled to understand the code, you need to spend some time on your programming skills before embarking on your Big Data career.

Remember that programming is just one piece of the puzzle. You will need to learn and understand the Big Data frameworks. You'll also need to understand how algorithms are done at scale. For this, you'll need materials and help to learn.

CHAPTER B

Appendix: Getting Your Skills Ready

By John Desmond

Note: This post was a guest written by John Desmond and orginally appeared on my site.

My preparation for the course began before I knew about the course, and before I realized that I wanted to specialize in data engineering. When I decided I wanted to learn programming, I hadn't quite decided if I wanted to be a programmer or a data scientist. My initial introduction to programming was through reading the book: Introduction to Statistical Learning in R. I read through the first few chapters of the book and completed the exercises at the end of each chapter. This gave me an introduction to the ways in which programming could be used in a statistical context. Learning from this book allowed me to leave open the option of becoming a programmer or a data scientist, as it taught skills that could be used in both disciplines.

After telling one of my friends who was already working as a data scientist that I was interested in programming, he recommended Learn Python the Hard Way by Zed Shaw. I bought the PDF version of the book for approximately thirty dollars, and I started working on the exercises. I was really happy with the book, and I highly recommend it to beginner programmers. The book starts with the basics of printing and data types, moves on to user input, data structures and control flows, and finishes off with object oriented programming (OOP), virtual machines, unit-testing and porting a game you design to a web application using HTML5, CSS, and Flask (a micro web framework). While completing the book I took the concepts I learned and started applying them to basic projects. I made a flash card app, a gamethat allows the player to practice mental math in the context of playing darts, and a basic website for my band.

I still wasn't entirely sure if I would go the data science or programming route, so I completed Dan Becker's Machine Learning Tutorial and applied what I had

learned from the tutorial to Kaggle's Titanic Data Challenge, which is essentially the hello world for Kaggle Competitions. While completing the tutorial I realized that a lot of my time went into preparing the data to be used for analysis by the given algorithms. At this point I was beginning to realize that I liked the programming and data management aspect of data science more than the analytical facet.

While completing these projects and tutorials, I was working as a data analyst. In my role I was introduced to relational databases, and I was given time to teach myself SQL to create reports for my team. One of the resources I found that helped me practice SQL queries was https://sqlzoo.net/. It starts from basic single column queries, to more advanced topics like recursive queries and subqueries. Some of the tasks at my job were rather repetitive, and so I decided that I would attempt to automate portions of a Standard Operating Procedure that my team had to perform daily. A friend recommended using PowerShell scripts to do so. I took his advice and spent a few days learning the syntax. I then found a script online that I was able to modify. By modifying the script, I was able to automate the saving of particular emails within my inbox to specific folders on my team's shared drive.

As I continued to study in my free time, it became apparent that there were many paths within computer science/software engineering in which one could specialize. After watching a conversation between Sylvester Morgan and John Sonmez in which they discussed the importance of specializing, I found this post by John that lays out some of the potential paths that one can follow within software development. Although data engineering isn't one of the listed specializations, I realized that given my background with databases, Python, and some knowledge of data science, that specializing as a data engineer was something worth striving for.

At this point I started to google how to become a data engineer and found some great material right off of the bat, in particular I found Pranav Dar's Comprehensive List of Resources to Become a Data Engineer. From there I read Part I and Part IIof Robert Chang's A Beginner's Guide to Data Engineering which gives a background of the theory relevant to data engineering and delves into how Robert used data engineering at Airbnb. I continued to search around for advice on what to learn to become a data engineer, and this time I turned to reddit. There I found a few suggestions to pick up the book Designing Data Intensive Applications by Martin Kleppmann and I also found a few references to Jesse

Anderson's Professional Data Engineering course.

I looked up Jesse's course, and found the webpage that provides a free copy of the Ultimate Guide to Switching Careers to Big Data. I read through the guide, and was surprised by the advice that I should find an expert to guide me through the career switch. Throughout the guide Jesse made it clear that intermediate level Java skills were necessary to succeed as a data engineer, and before reading the guide I had yet to learn any Java. I emailed Jesse and he told me that the Java skills mentioned in the guide were a prerequisite to taking his course.

I had just put in my two weeks at my job to begin studying full time, and so I decided to try to teach myself the basics of Java as quickly as possible. I started off with Bryson Payne's Learn Java the Easy Way on Udemy as I had access to the course from a package deal from a website called Infostack. The course was useful as it guides you through downloading the necessary JDKs, Eclipse (an IDE) and Android Studio. I completed the first app on the command line, and quickly realized that I would need to begin more advanced projects in order to get my skills where they needed to be to start the Data Engineering course.

I looked up how I could learn a programming language as quickly as possible, given that I already had an intermediate grasp of another programming language. The advice I heard from the Tech Lead (Patrick Shyu) and John Sonmez, was to re-write projects that you had already created in another programming language into the new language that you wanted to learn. Their point being that this allows you to take the abstract concepts you learned previously and implement them using the new language. I did this with the mental math gameand the RSS feed readerI mentioned earlier.

As I began these projects, I mentioned to some friends who already knew Java that I was starting to learn the language, and they recommended Maven for project dependency management and Spring Boot as a web framework. I utilized Maven for both of my projects and I used Spring Boot as the web framework for my second project. I highly recommend becoming familiar with both of these technologies when starting out with Java, as they greatly simplify the process of creating Java applications. To get started with Maven I used the Command Line approach given here: Maven in Five Minutes (spoiler alert, this may take you longer than 5 minutes). To get started with Spring and Maven I recommend this link and this guide as well.

For the RSS feed reader I set up the HTML using Thymeleaf, the database using

JPA/Hibernate/MariaDB, the web application using Spring Boot and the Java code I used touched on interfaces, inheritance, and threading. Throughout the process of completing the two projects I used a lot of blog posts and occasionally YouTube tutorials to answer some of the questions I had regarding Java. For questions regarding the Spring framework I found myself frequently on the https://www.baeldung.com/ and https://springframework.guru/ websites. For YouTube Java tutorials I recommend the Java Brains channel. For the basics of the Java syntax and for the core abstractions behind the language I recommend the Java tutorials within the Java Documentation website.

I sent Jesse my two projects as an example of what I had learned over the past two and a half weeks, and he recommended that I gain a deeper understanding of threading in Java before beginning the course. I had one asynchronous thread that was running in my Spring Boot application, but I hadn't taken the time to explore threading in detail.

To finish my preparation for the course, I read through and played with the code found in this great tutorial series, Rajeev Singh's Java Concurrency/Multithreading tutorial. The final two posts of the tutorial were especially useful, as they answered some of the questions that I had concerning how to refactor one's code such that multiple threads aren't modifying the same variable or resource simultaneously.

To practice threading, I decided to try my hand at the Producer Consumer problem. I made an application that abstractly cooks and delivers burgers, then checks to see if the burgers have been cooked/delivered. Although my application could have been more efficient, I was able to alter the shared variables within the application in a synchronized manner using two threads and 5 tasks. To ensure that my application was thread safe, I made use of the BlockingQueue interface, the volatile keyword, Atomic Booleans for the burger states and Synchronized Lists to store the orders, i.e. the burgers before they were cooked, and the burgers to be delivered. To schedule the tasks I used the ScheduledExecutorService Interface. This allowed me to create a thread pool, and then schedule each of my tasks periodically. Something I would have done differently now that I have created other threaded programs would be to make a method for each Runnable class, and submit the Runnable class within its given method. This would make the main method of the App class much more readable. Throughout the process, I made continual reference to the tutorial from Rajeev Singh that I mentioned earlier.

If you look back at my earlier commits for the project, you'll see that I was using the synchronized keyword on the Producer and Consumer methods and the AtomicBurger variable set methods, and I was making all of my shared variables volatile. Although the application ran as such, I decided to use Atomic Variables instead of using the synchronized keyword, and to limit my use of the volatile keyword to make my application more efficient. When I instantiated the majority of the objects I needed in the main thread *without* using the volatile keyword, my application ran much faster; the reason being that the volatile keyword disables the compiler's optimization of instructions for the variable it is used to define. While the synchronized keyword and Atomic Variables achieve the same end of ensuring that a shared resource is altered by one thread at a time, Atomic Variables do so by using the compare-and-swap method which in many cases is more efficient than locking the variables. Although the application implementation and design were deliberately simple, the project was still challenging to complete given the complexity of threading.

I am currently a week into the course, and I feel that my preparation for the course has been sufficient; allowing me to comprehend the technical details within the lectures and to complete the assigned exercises. I hope you have been able to learn from the path I've taken to arrive at my current educational locus. If you are new to Java and don't have an intermediate level programming project from another language that you can copy over, here are 13 project ideas for Python developers which can also be implemented using Java.