The Data Engineering Cookbook

How to master the plumbing of data science

Andreas Kretz

December 2, 2018

v0.1

Contents

1 Introduction

What do you actually need to learn to become an awesome data engineer? Look no further, you find it here.

How to use this document: This is not a training! It's a collection of skills, that I value highly in my daily work as a data engineer. It's intended to be a starting point for you to find the topics to look into.

This project is a work in progress! Over the next weeks I am going to share with you my thoughts on why each topic is important. I also try to include links to useful resources.

How to find out what is new? You will always find the newest version on my Patreon **https://www.patreon.com/plumbersofds**

Help make this collection awesome! Join the discussion on Patreon or write me an email to andreaskayy@gmail.com. Tell me your thoughts, what you value, you think should be included, or where I am wrong.

– Andreas

2 What is Data Science?

2.1 Data Scientist

2.2 Data Engineer

2.3 Data Analyst

3 The Basic Skills

It's the combination of multiple skills that is important. The technique is called talent stacking.

I talked about why Companies Badly Need Data Engineers

I talked about talent stacks for data engineers in this podcast:

4 Learn to Write Code

Why this is important: Without coding you cannot do much in data engineering. I cannot count the number of times I needed a quick Java hack.

The possibilities are endless:

- Writing or quickly getting some data out of a SQL DB
- Testing to produce messages to a Kafka topic
- Understanding Source code of a Java Webservice
- Reading counter statistics out of a HBase key value store

So, which language do I recommend then?

I highly recommend Java. It's everywhere!

When you are getting into data processing with Spark you should use Scala. But, after learning Java this is easy to do.

Also Python is a great choice. It is super versatile.

Personally however, I am not that big into Python. But I am going to look into it

Where to Learn? There's a Java Course on Udemy you could look at: https://www.udemy.com/java-programming-tutorial-for-beginners

4.1 Coding Basics

- OOP Object oriented programming
- What are Unit tests to make sure what you code is working
- Functional Programming
- How to use build managment tools like Maven
- Resilliant testing (?)

I talked about the importance of learning by doing in this podcast:

4.2 Learn To Use GitHub

Why this is important: One of the major problems with coding is to keep track of changes. It is also almost impossible to maintain a program you have multiple versions of.

Another is the topic of collaboration and documentation. Which is super Important.

Let's say you work on a Spark application and your colleges need to make changes while you are on holiday. Without some code management they are in huge trouble:

Where is the code? What have you changed last? Where is the documentation? How do we mark what we have changed?

But if you put your code on GitHub your colleges can find your code. They can understand it through your documentation (please also have in-line comments)

Developers can pull your code, make a new branch and do the changes. After your holiday you can inspect what they have done and merge it with your original code. and you end up having only one application

Where to learn: Check out the GitHub Guides page where you can learn all the basics: https://guides.github.com/introduction/flow/

This great GitHub commands cheat sheet saved my butt multiple times: https://www.atlassian.com/git/
git-cheatsheet

Pull, Push, Branching, Forking

Also talked about it in this podcast:

4.3 Agile Development

4.3.1 Scrum

4.3.2 OKR

I talked about this in this Podcast: https://anchor.fm/andreaskayy/embed/episodes/Agile-Development-Is-Important-But-Please-Dont-Do-Scrum–PoDS-041-e2e2j4

5 Computer Science Basics

5.1 Learn how a Computer Works

5.1.1 CPU,RAM,GPU,HDD

5.1.2 Differences between PCs and Servers

I talked about computer hardware and GPU processing in this podcast: https://anchor.fm/andreaskayy/
the-hardware-and-the-GPU-is-super-important–PoDS-030-e23rig

5.2 Computer Networking

5.2.1 ISO/OSI Model

5.2.2 IP Subnetting

5.2.3 Switch, Level 3 Switch

5.2.4 Router

5.2.5 Firewalls

I talked about Network Infrastructure and Techniques in this podcast: https://anchor.fm/andreaskayy/e
Networking-Infrastructure-and-Linux-031-PoDS-e242bh

5.3 Security and Privacy

5.3.1 SSL Public & Private Key Certificates

5.3.2 What is a certificate authority

5.3.3 JAva Web Tokens

5.3.4 GDPR regulations

5.3.5 Privacy by design

5.4 Linux

5.4.1 OS Basics

5.4.2 Shell scripting

5.4.3 Cron jobs

5.4.4 Packet management

Linux Tips are the second part of this podcast: https://anchor.fm/andreaskayy/embed/episodes/IT-Networking-Infrastructure-and-Linux-031-PoDS-e242bh

6 Data Science Platform

6.1 Security Zone Design

6.1.1 How to secure a multi layered application

(UI in different zone then SQL DB)

6.1.2 Cluster security with Kerberos

I talked about security zone design and lambda architecture in this podcast: https://anchor.fm/andreask
to-Design-Security-Zones-and-Lambda-Architecture–PoDS-032-e248q2

6.2 Lambda Architecture

6.2.1 Stream and Batch processing

6.2.2 My Big Data Platform Blueprint

Some time ago I have created a simple and modular big data platform blueprint for myself. It is based on what I have seen in the field and read in tech blogs all over the internet.

Today I am going to share it with you.

Why do I believe it will be super useful to you?

Because, unlike other blueprints it is not focused on technology. It is based on four common big data platform design patterns.

Following my blueprint will allow you to create the big data platform that fits exactly your needs. Building the perfect platform will allow data scientists to discover new insights.

It will enable you to perfectly handle big data and allow you to make data driven decisions.

THE BLUEPRINT The blueprint is focused on the four key areas: Ingest, store, analyse and display.

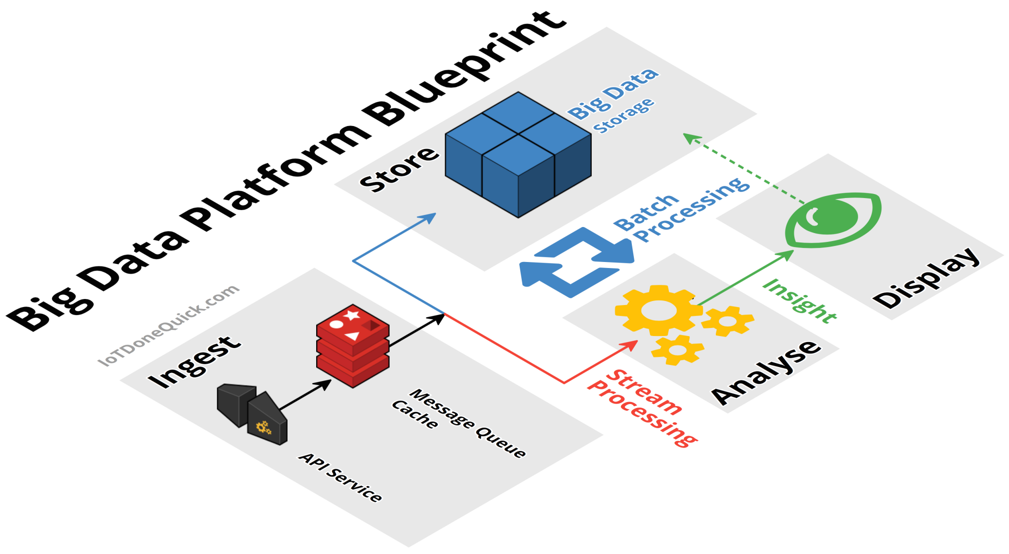

Figure 1: Platfrom Blueprint

Having the platform split like this turns it it a modular platform with loosely coupled interfaces.

Why is it so important to have a modular platform?

If you have a platform that is not modular you end up with something that is fixed or hard to modify. This means you can not adjust the platform to changing requirements of the company.

Because of modularity it is possible to switch out every component, if you need it.

Now, lets talk more about each key area.

INGEST Ingestion is all about getting the data in from the source and making it available to later stages. Sources can be everything form tweets, server logs to IoT sensor data like from cars.

Sources send data to your API Services. The API is going to push the data into a temporary storage.

The temporary storage allows other stages simple and fast access to incoming data.

A great solution is to use messaging queue systems like Apache Kafka, RabbitMQ or AWS Kinesis. Sometimes people also use caches for specialised applications like Redis.

A good practice is that the temporary storage follows the publish, subscribe pattern. This way APIs can publish messages and Analytics can quickly consume them.

STORE This is the typical big data storage where you just store everything. It enables you to analyse the big picture.

Most of the data might seem useless for now, but it is of upmost importance to keep it. Throwing data away is a big no no.

Why not throw something away when it is useless?

Although it seems useless for now, data scientists can work with the data. They might find new ways to analyse the data and generate valuable insight from it.

What kind of systems can be used to store big data?

Systems like Hadoop HDFS, Hbase, Amazon S3 or DynamoDB are a perfect fit to store big data.

ANALYSE The analyse stage is where the actual analytics is done. Analytics, in the form of stream and batch processing.

Streaming data is taken from ingest and fed into analytics. Streaming analyses the “live” data thus, so generates fast results.

As the central and most important stage, analytics also has access to the big data storage. Because of that connection, analytics can take a big chunk of data and analyse it.

This type of analysis is called batch processing. It will deliver you answers for the big questions.

To learn more about stream and batch processing read my blog post: How to Create New and Exciting Big Data Aided Products

The analytics process, batch or streaming, is not a one way process. Analytics also can write data back to the big data storage.

Often times writing data back to the storage makes sense. It allows you to combine previous analytics outputs with the raw data.

Analytics insight can give meaning to the raw data when you combine them. This combination will often times allow you to create even more useful insight.

A wide variety of analytics tools are available. Ranging from MapReduce or AWS Elastic MapReduce to Apache Spark and AWS lambda.

DISPLAY Displaying data is as important as ingesting, storing and analysing it. People need to be able to make data driven decisions.

This is why it is important to have a good visual presentation of the data. Sometimes you have a lot of different use cases or projects using the platform.

It might not be possible for you to build the perfect UI that fits everyone. What you should do in this case is enable others to build the perfect UI themselves.

How to do that? By creating APIs to access the data and making them available to developers.

Either way, UI or API the trick is to give the display stage direct access to the data in the big data cluster. This kind of access will allow the developers to use analytics results as well as raw data to build the the perfect application.

6.2.3 Three methods of streaming

In stream processing sometimes it is ok to drop messages, other times it is not. Sometimes it is fine to process a message multiple times, other times that needs to be avoided like hell.

Today's topic are the different methods of streaming: At most once, at least once and exactly once.

What this means and why it is so important to keep them in mind when creating a solution. That is what you will find out in this article.

At Least Once At least once, means a message gets processed in the system once or multiple times. So with at least once it's not possible that a message gets into the system and is not getting processed.

It's not getting dropped or lost somewhere in the system.

One example where at least once processing can be used is when you think about a fleet management of cars. You get GPS data from cars and that GPS data is transmitted with a timestamp and the GPS coordinates.

It's important that you get the GPS data at least once, so you know where the car is. If you're processing this data multiple times, it always has the the timestamp with it.

Because of that it does not matter that it gets processed multiple times, because of the timestamp. Or that it would be stored multiple times, because it would just override the existing one.

At Most Once The second streaming method is at most once. At most once means that it's okay to drop some information, to drop some messages.

But it's important that a message is only only processed once as a maximum.

A example for this is event processing. Some event is happening and that event is not important enough, so it can be dropped. It doesn't have any consequences when it gets dropped.

But when that event happens it's important that it does not get processed multiple times. Then it would look as if the event happend five or six times instead of only one.

Think about engine misfires. If it happens once, no big deal. But if the system tells you it happens a lot you will think you have a problem with your engine.

Exactly Once Another thing is exactly once, this means it's not okay to drop data, it's not okay to lose data and it's also not okay to process one message what data said multiple times

A example for this is for instance banking. When you think about credit card transactions it's not okay to drop a transaction.

When dropped your payment is not going through. It's also not okay to have a transaction processed multiple times, because then you are paying multiple times.

Check The Tools! All of this sounds very simple and logical. What kind of processing is done has to be a requirement for your use case.

It needs to be thought about in the design process, because not every tool is supporting all three methods. Very often you need to code your application very differently based on the streaming method.

Especially exactly once is very hard to do.

So, the tool of data processing needs to be chosen based on if you need exactly once, at least once or if you need at most once.

6.3 Big Data

6.3.1 What is big data and where is the difference to data science and data analytics?

I talked about the difference in this podcast: https://anchor.fm/andreaskayy/embed/episodes/BI-vs-Data-Science-vs-Big-Data-e199hq

6.3.2 The 4Vs of Big data:

It is a complete misconception. Volume is only one part of the often called four V's of big data: Volume, velocity, variety and veracity.

Volume is about the size. How much data you have.

Velocity is about the speed of how fast the data is getting to you.

How much data is in a specific time needs to get processed or is coming into the system. This is where the whole concept of streaming data and real-time processing comes in to play.

Variety is the third one. It means, that the data is very different. That you have very different types of data structures.

Like CSV files, PDFs that you have stuff in XML. That you have JSON logfiles, or that you have data in some kind of a key value store.

It's about the variety of data types from different sources that you basically want to join together. All to make an analysis based on that data.

Veracity is fourth and this is a very very difficult one. The issue with big data is, that it is very unreliable.

You cannot really trust the data. Especially when you're coming from the IoT, the Internet of Things side. Devices use sensors for measurement of temperature, pressure, acceleration and so on.

You cannot always be hundred percent sure that the actual measurement is right.

When you have data that is from for instance SAP and it contains data that is created by hand you also have problems. As you know we humans are bad at inputting stuff.

Everybody articulates different. We make mistakes, down to the spelling and that can be a very difficult issue for analytics.

I talked about the 4Vs in this podcast: https://anchor.fm/andreaskayy/embed/episodes/4-Vs-Of-Big-Data-Are-Enough-e1h2ra

6.3.3 Why Big Data?

What I always emphasize is the four V's are quite nice. They give you a general direction.

There is a much more important issue: Catastrophic Success.

What I mean by catastrophic success is, that your project, your startup or your platform has more growth that you anticipated. Exponential growth is what everybody is looking for.

Because with exponential growth there is the money. It starts small and gets very big very fast. The classic hockey stick curve:

1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384....BOOM!

Think about it. It starts small and quite slow, but gets very big very fast.

You get a lot of users or customers who are paying money to use your service, the platform or whatever. If you have a system that is not equipped to scale and process the data the whole system breaks down.

That's catastrophic success. You are so successfull and grow so fast that you cannot fulfill the demand anymore. And so you fail and it's all over.

It's now like you just can make that up while you go. That you can foresee in a few months or weeks the current system doesn't work anymore.

Planning is Everything It's all happens very very fast and you cannot react anymore. There's a necessary type of planning and analyzing the potential of your business case necessary.

Then you need to decide if you actually have big data or not.

You need to decide if you use big data tools. This means when you conceptualize the whole infrastructure it might look ridiculous to actually focus on big data tools.

But in the long run it will help you a lot. Good planning will get a lot of problems out of the way, especially if you think about streaming data and real-time analytics.

The Problem With ETL A typical old-school platform deployment would look like the picture below. Devices use a data API to upload data that gets stored in a SQL database. An external analytics tool is querying data and uploading the results back to the SQL db. Users then use the user interface to display data stored in the database.

Figure 2: ETL Issues

Now, when the front end queries data from the SQL database the following three steps happen:

The database extracts all the needed rows from the storage. Extracted data gets transformed, for instance sorted by timestamp or something a lot more complex.

The extracted and transformed data gets loaded to the destination (the user interface) for chart creation With exploding amounts of stored data the ETL process starts being a real problem.

Analytics is working with large data sets, for instance whole days, weeks, months or more. Data sets are very big like 100GB or Terabytes. That means Billions or Trillions of rows.

This has the result that the ETL process for large data sets takes longer and longer. Very quickly the ETL performance gets so bad it won't deliver results to analytics anymore.

A traditional solution to overcome these performance issues is trying to increase the performance of the database server. That's what's called scaling up.

Scaling Up To scale up the system and therefore increase ETL speeds administrators resort to more powerful hardware by:

Speeding up the extract performance by adding faster disks to physically read the data faster. Increasing RAM for row caching. What is already in memory does not have to be read by slow disk drives. Using more powerful CPU's for better transform performance (more RAM helps here as well) Increasing or optimising networking performance for faster data delivery to the front end and analytics Scaling up the system is fairly easy.

Figure 3: ETL Issues

But with exponential growth it is obvious that sooner or later (more sooner than later) you will run into the same problems again. At some point you simply cannot scale up anymore because you already have a monster system, or you cannot afford to buy more expensive hardware.

The next step you could take would be scaling out.

Scaling Out Scaling out is the opposite of scaling up. Instead of building bigger systems the goal is to distribute the load between many smaller systems.

The simplest way of scaling out an SQL database is using a storage area network (SAN) to store the data. You can then use up to eight SQL servers, attach them to the SAN and let them handle queries. This way load gets distributed between those eight servers.

Figure 4: ETL Issues

One major downside of this setup is that, because the storage is shared between the sql servers, it can only be used as an read only database. Updates have to be done periodically, for instance once a day. To do updates all SQL servers have to detach from the database. Then, one is attaching the db in read-write mode and refreshing the data. This procedure can take a while if a lot of data needs to be uploaded.

This Link to a Microsoft MSDN page has more options of scaling out an SQL database for you.

I deliberately don't want to get into details about possible scaling out solutions. The point I am trying to make is that while it is possible to scale out SQL databases it is very complicated.

There is no perfect solution. Every option has its up- and downsides. One common major issue is the administrative effort that you need to take to implement and maintain a scaled out solution.

Please Don't go Big Data If you don't run into scaling issues please, do not use big data tools!

Big data is a expensive thing. A Hadoop cluster for instance needs at least five servers to work properly. More is better.

Believe me this stuff costs a lot of money.

Especially when you are talking the maintenance and development on top big daat tools into account.

If you don't need it it's making absolutely no sense at all!

On the other side: If you really need big data tools they will save your ass :)

6.3.4 What are the tools associated?

6.4 What is the difference between a Data Warehouse and a Data Lake

6.5 Hadoop Platfroms

When people talk about big data, one of the first things come to mind is Hadoop. Google's search for Hadoop returns about 28 million results.

It seems like you need Hadoop to do big data. Today I am going to shed light onto why Hadoop is so trendy.

You will see that Hadoop has evolved from a platform into an ecosystem. It's design allows a lot of Apache projects and 3rd party tools to benefit from Hadoop.

I will conclude with my opinion on, if you need to learn Hadoop and if Hadoop is the right technology for everybody.

WHAT IS HADOOP? Hadoop is a platform for distributed storing and analyzing of very large data sets.

Hadoop has four main modules: Hadoop common, HDFS, MapReduce and YARN. The way these modules are woven together is what makes Hadoop so successful.

The Hadoop common libraries and functions are working in the background. That's why I will not go further into them. They are mainly there to support Hadoop's modules.

6.5.1 What makes Hadoop so popular?

Storing and analyzing data as large as you want is nice. But what makes Hadoop so popular?

Hadoop's core functionality is the driver of Hadoop's adoption. Many Apache side projects use it's core functions.

Because of all those side projects Hadoop has turned more into an ecosystem. An ecosystem for storing and processing big data.

To better visualize this eco system I have drawn you the following graphic. It shows some projects of the Hadoop ecosystem who are closely connected with the Hadoop.

It is not a complete list. There are many more tools that even I don't know. Maybe I am drawing a complete map in the future.

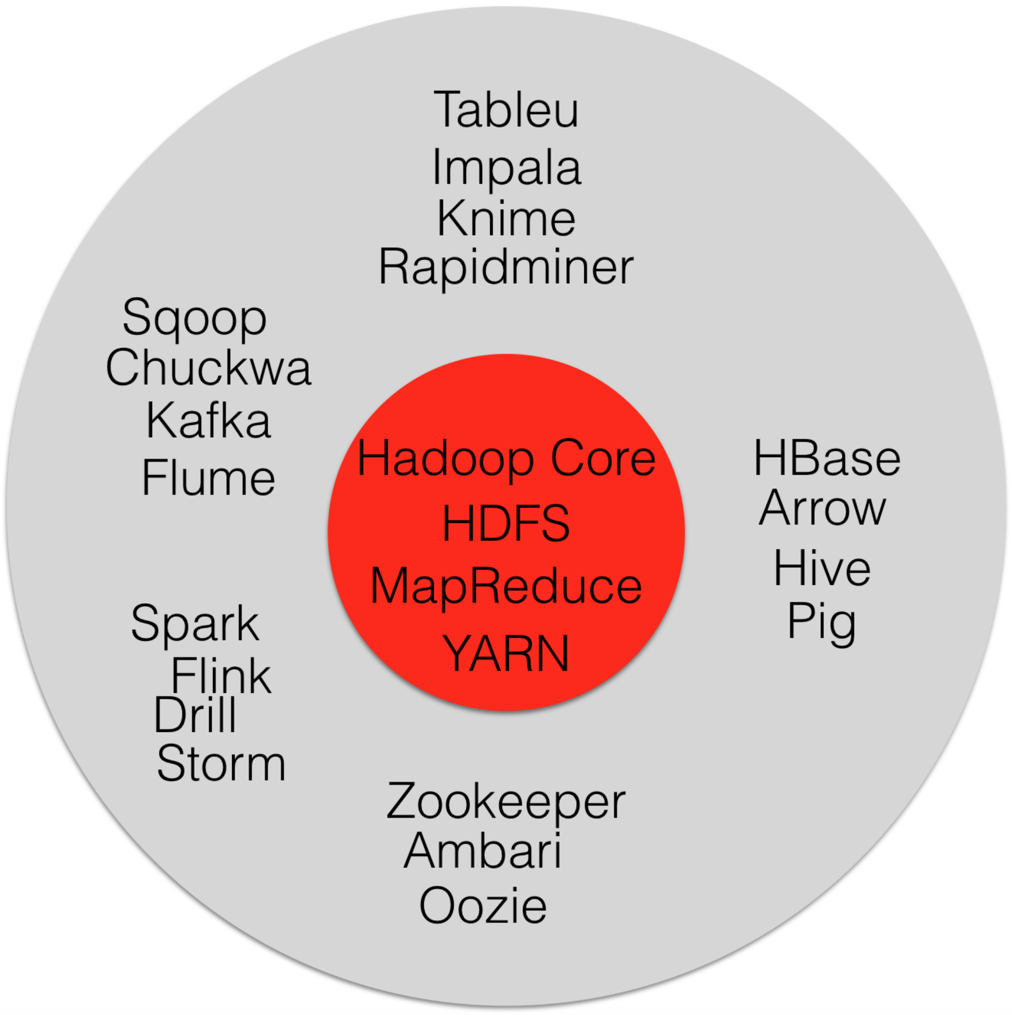

Figure 5: Hadoop Ecosystem Components

HOW THE ECOSYSTEM'S COMPONENTS WORK TOGETHER Remember my big data platform blueprint? The blueprint has four stages: Ingest, store, analyse and display.

Because of the Hadoop ecosystem" the different tools in these stages can work together perfectly.

Here's an example:

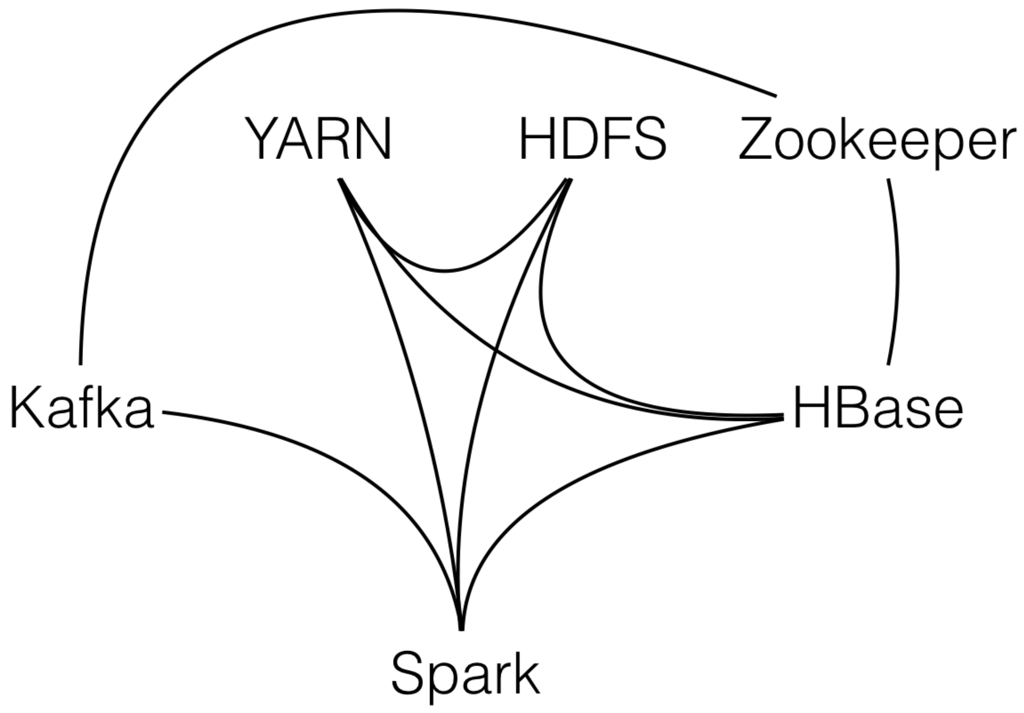

Figure 6: Connections between tools

You use Apache Kafka to ingest data, and store the it in HDFS. You do the analytics with Apache Spark and as a backend for the display you store data in Apache HBase.

To have a working system you also need YARN for resource management. You also need Zookeeper, a configuration management service to use Kafka and HBase

As you can see in the picture below each project is closely connected to the other.

Spark for instance, can directly access Kafka to consume messages. It is able to access HDFS for storing or processing stored data.

It also can write into HBase to push analytics results to the front end.

The cool thing of such ecosystem is that it is easy to build in new functions.

Want to store data from Kafka directly into HDFS without using Spark?

No problem, there is a project for that. Apache Flume has interfaces for Kafka and HDFS.

It can act as an agent to consume messages from Kafka and store them into HDFS. You even do not have to worry about Flume resource management.

Flume can use Hadoop's YARN resource manager out of the box.

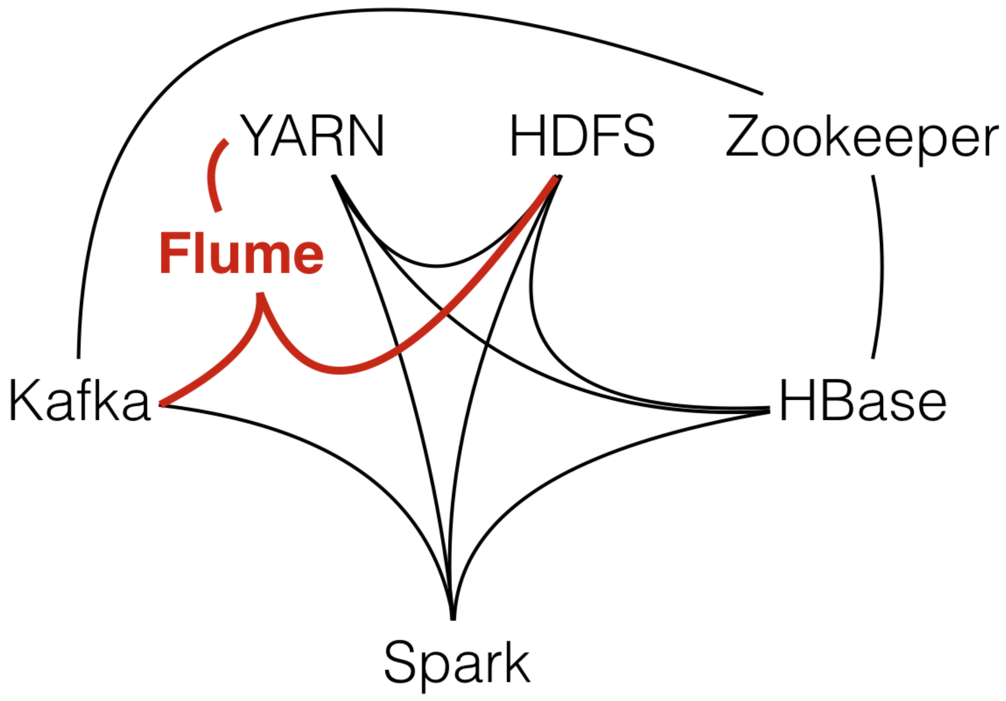

Figure 7: Flume Integration

IS HADOOP WORKING EVERYWHERE? Although Hadoop is so popular it is not the silver bullet. It isn't the tool that you should use for everything.

Often times it does not make sense to deploy a Hadoop cluster, because it can be overkill. Hadoop does not run on a single server.

You basically need at least five servers, better six to run a small cluster. Because of that. the initial platform costs are quite high.

One option you have is to use a specialized systems like Cassandra, MongoDB or other NoSQL DB's for storage. Or you move to Amazon and use Amazon's Simple Storage Service, or S3.

Guess what the tech behind S3 is. Yes, HDFS. That's why AWS also has the equivalent to MapReduce named Elastic MapReduce.

The great thing about S3 is that you can start very small. When your system grows you don't have to worry about s3's server scaling.

SHOULD YOU LEARN HADOOP? Yes, I definitely recommend you to get to now how Hadoop works and how to use it. As I have shown you in this article, the ecosystem is quite large.

Many big data projects use Hadoop or can interface with it. Thats why it is generally a good idea to know as many big data technologies as possible.

Not in depth, but to the point that you know how they work and how you can use them. Your main goal should be to be able to hit the ground running when you join a big data project.

Plus, most of the technologies are open source. You can try them out for free.

6.5.2 How does a Hadoop System architecture look like

6.5.3 What tools are usually in a with Hadoop Cluster

Yarn Zookeeper HDFS Oozie Flume Hive

6.5.4 How to select Hadoop Cluster Hardware

6.6 Is ETL still relevant for Analytics?

I talked about this in this podcast: https://anchor.fm/andreaskayy/embed/episodes/Is-ETL-Dead-For-Data-Science–Big-Data—PoDS-039-e2b604

6.7 The Cloud

6.7.1 AWS,Azure, IBM, Google Cloud basics

6.7.2 cloud vs on premise

6.7.3 up & downsides

6.7.4 Security

Listen to a few thoughts about the cloud in this podcast: https://anchor.fm/andreaskayy/embed/episod
Be-Arrogant-The-Cloud-is-Safer-Then-Your-On-Premise-e16k9s

6.8 Docker

6.8.1 What is docker and what do you use it for

Have you played around with Docker yet? If you're a data science learner or a data scientist you need to check it out!

It's awesome because it simplifies the way you can set up develpoment environments for data science. If you want to set up a dev environment you usually have to install a lot of packages and tools.

Don't Mess Up Your System What this does is you basically mess up your operating system. If you're a starter you don't know which packages you need to install. You don't know which tools you need to install.

If you want to for instance start with Jupyter notebooks you need to install that on your PC somehow. Or you need to start installing tools like PyCharmor Anaconda.

All that gets added to your system and so you mess up your system more and more and more. What Docker brings you, especially if you're on a Mac or a Linux system is simplicity.

Preconfigured Images Because it is so easy to install on those systems. Another cool thing about docker images is you can just serach them in the Docker store, download them and install them on your system.

Runing them in a completely pre-configured environment. You don't need to think about stuff you go to the Docker library you search for deep learning GPU and Python.

You get a list of images you can download. You download one, start it up, you go to the browser hit up the URL and just start coding.

Start doing the work. The only other thing you need to do is bind some drives to that instance so you can exchange files. And then that's it!

There is no way that you can crash or mess up your system. It's all encapsulated into Docker.Why this works is because Docker has natively access to your hardware.

Take It With You It's not a completely virtualized environment like a VirtualBox. An image has the upside that you can take it wherever you want. So if your on your PC at home use that there.

Make a quick build, take the image and go somewhere else. Install the image which is usually quite fast and just use it like you're at home.

It's that awesome!

Kubernetes Container Deployment I am getting into Docker a lot more myself. For a bit different reasons.

What I'm looking for is using Docker with Kubernetes. Kubernets you can automate the whole container deployment process.

The idea with is that you have a cluster of machines. Lets say you have 10 server cluster and you run Kubernetes on them.

Kubernetes lets you spin up Docker containers on-demand to execute tasks. You can set up how much resources like CPU, RAM, Network, Docker container can use.

You can basically spin up containers, on the cluster on demand. When ever you need to do a analytics task.

Perfect for Data Science.

6.8.2 How to create, start,stop a Container

6.8.3 Docker micro services?

6.8.4 Kubernetes

6.8.5 Why and how to do Docker container orchestration

Podcast about how data science learners use Docker (for data scientists): https://anchor.fm/andreaskayy
Data-Science-Go-Docker-e10n7u

7 How to Ingest Data

7.1 Application programming interfaces

Check out my podcast about how APIs rule the world: https://anchor.fm/andreaskayy/embed/episodes/
APIs-Rule-The-World–PoDS-033-e24ttq

7.1.1 REST APIs

7.1.2 HTTP Post/Get

7.1.3 API Design

7.1.4 Implementation

7.1.5 OAuth security

7.2 JSON

7.2.1 Super important for REST APIs

7.2.2 How is it used for logging and processing logs

8 Distributed Processing

I talked about why distributed processing is so super important in this Podcast:

8.1 MapReduce

8.1.1 Why was MapReduce Invented

8.1.2 How does that work

8.1.3 What is the limitation of MapReduce?

8.2 Apache Spark

I talked about the three methods of data streaming in this podcast: https://anchor.fm/andreaskayy/emb
Methods-of-Streaming-Data-e15r6o

8.2.1 Spark Basics

8.2.2 What is the difference to MapReduce?

8.2.3 How to do stream processing

8.2.4 How to do batch processing

8.2.5 How does Spark use data from Hadoop

8.2.6 What is a RDD and what is a DataFrame?

8.2.7 Spark coding with Scala

8.2.8 Spark coding with Python

8.2.9 How and why to use SparkSQL?

8.2.10 Machine Learning on Spark? (Tensor Flow)

8.3 Message queues with Apache Kafka

8.3.1 Why a message queue tool?

8.3.2 Kakfa architecture

8.3.3 What are topics

8.3.4 What does Zookeeper have to do with Kafka

8.3.5 How to produce and consume messages

My YouTube video how to set up Kafka at home: https://youtu.be/7F9tBwTUSeY

My YouTube video how to write to Kafka: https://youtu.be/RboQBZvZCh0

8.4 Machine Learning

My podcast about how to do machine learning in production: https://anchor.fm/andreaskayy/embed/ep
Learning-In-Production-e11bbk

8.4.1 Training and Applying models

8.4.2 What is deep learning

8.4.3 How to do Machine Learning in production

Machine learning in production is using stream and batch processing. In the batch processing layer you are creating the models, because you have all the data available for training.

In the stream in processing layer you are using the created models, you are applying them to new data.

The idea that you need to incorporate is this is a constant a constant cycle. Training, applying, re-training, pushing into production and applying.

What you don't want to do is, you don't want to do this manually. You need to figure out a process of automatic retraining and automatic pushing to into production of models.

In the retraining phase the system automatically evaluates the training. If the model no longer fits it works as long as it needs to create a good model.

After the evaluation of the model is complete and it's good, the model gets pushed into production. Into the stream processing.

8.4.4 Why machine learning in production is harder then you think

How to automate machine learning is something that drives me day in and day out.

What you do in development or education is, that you create a model and fit it to the data. Then that model is basically done forever.

Where I'm coming from, the IoT world, the problem is that machines are very different. They behave very different and experience wear.

Models Do Not Work Forever Machines have certain processes that decrease the actual health of the machine. Machine wear is a huge issue. Models that that are built on top of a good machine don't work forever.

When the Machine wears out, the models need to be adjusted. They need to be maintained, retrained.

Where The Platforms That Support This? Automatic re-training and re-deploying is a very big issue, a very big problem for a lot of companies. Because most existing platforms don't have this capability (I actually haven't seen one until now).

Look at AWS machine learning for instance. The process is: build, train, tune deploy. Where's the loop of retraining?

You can create models and then use them in production. But this loop is almost nowhere to be seen.

It is a very big issue that needs to be solved. If you want to do machine learning in production you can start with manual interaction of the training, but at some point you need to automate everything.

Training Parameter Management To train a model you are manipulating input parameters of the models.

Take deep learning for instance. To train you are manipulating for instance:

How many layers do you use. The depth of the layers, which means how many neurons you have in a layer. What activation function you use, how long are you training and so on.

You also need to keep track of what data you used to train which model.

All those parameters need to be manipulated automatically, models trained and tested.

To do all that, you basically need a database that keeps track of those variables.

How to automate this, for me, is like the big secret. I am still working on figuring it out.

What's Your Solution? Did you already have the problem of automatic re-training and deploying of models as well?

Were you able to use a cloud platform like Google, AWS or Azure?

It would be really awesome if you share your experience :)

8.4.5 How to convince people machine learning works

Many people still are not convinced that machine learning works reliably. But they want analytics insight and most of the time machine learning is the way to go.

This means, when you are working with customers you need to do a lot of convincing. Especially if they are not into machine learning themselves.

But it's actually quite easy.

No Rules, No Physical Models Many people are still under the impression that analytics only works when it's based on physics. When there are strict mathematical rules to a problem.

Especially in engineering heavy countries like Germany this is the norm:

"Sere has to be a Rule for Everysing!" (imagine a German accent) When you're engineering you are calculating stuff based on physics and not based on data. If you are constructing an airplaine wing, you better make sure to use calculations so it doesn't fall off.

And that's totally fine.

Keep doing that!

Machine learning has been around for decades. It didn't quite work as good as people hoped. We have to admit that. But there is this preconception that it still doesn't work.

Which is not true: Machine learning works.

Somehow you need to convince people that it is a viable approach. That learning from data to make predictions is working perfectly.

You Have The Data. USE IT! As a data scientist you have one ace up your sleeve, it's the obvious one:

It's the data and it's statistics.

You can use that data and those statistics to counter peoples preconceptions. It's very powerful if someone says: "This doesn't work"

You bring the data. You show the statistics and you show that it works reliably.

A lot of discussions end there.

Data doesn't lie. You can't fight data. The data is always right.

Data is Stronger Than Opinions This is also why I believe that autonomous driving will come quicker than many of us think. Because a lot of people say, they are not safe.

That you cannot rely on those cars.

The thing is: When you have the data you can do the statistics.

You can show people that autonomous driving really works reliably. You will see, the question of: Is this is this allowed or is this not allowed? Will be gone quicker than you think.

Because goverenment agencies can start testing the algorithms based on predefined scenarios. They can run benchmarks and score the cars performance.

All those opinions, if it works, or if it doesn't work, they will be gone.

The motor agency has the statistics. The stats show people how good cars work.

Companies like Tesla, they have it very easy. Because the data is already there.

They just need to show us that the algorithms work. The end.

9 How to Store Data

Check out my podcast how to decide between SQL and NoSQL: https://anchor.fm/andreaskayy/embed/
Vs-SQL-How-To-Choose-e12f1o

9.1 Data Modeling

9.1.1 How to find out how you need to store data for the business case

9.1.2 How to decide what kind of storage you need to use

9.2 SQL

9.2.1 Database Design

9.2.2 SQL Queries

9.2.3 Stored Procedures

9.2.4 ODBC/JDBC Server Connections

9.3 NoSQL

9.3.1 KeyValue Stores (HBase)

9.3.2 Document Stores (HDFS, MongoDB)

HDFS The Hadoop distributed file system, or HDFS, allows you to store files in Hadoop. The difference between HDFS and other file systems like NTFS or EXT is that it is a distributed one.

What does that mean exactly?

A typical file system stores your data on the actual hard drive. It is hardware dependent.

If you have two disks then you need to format every disk with its own file system. They are completely separate.

You then decide on which disk you physically store your data.

HDFS works different to a typical file system. HDFS is hardware independent.

Not only does it span over many disks in a server. It also spans over many servers.

HDFS will automatically place your files somewhere in the Hadoop server collective.

It will not only store your file, Hadoop will also replicate it two or three times (you can define that). Replication means replicas of the file will be distributed to different servers.

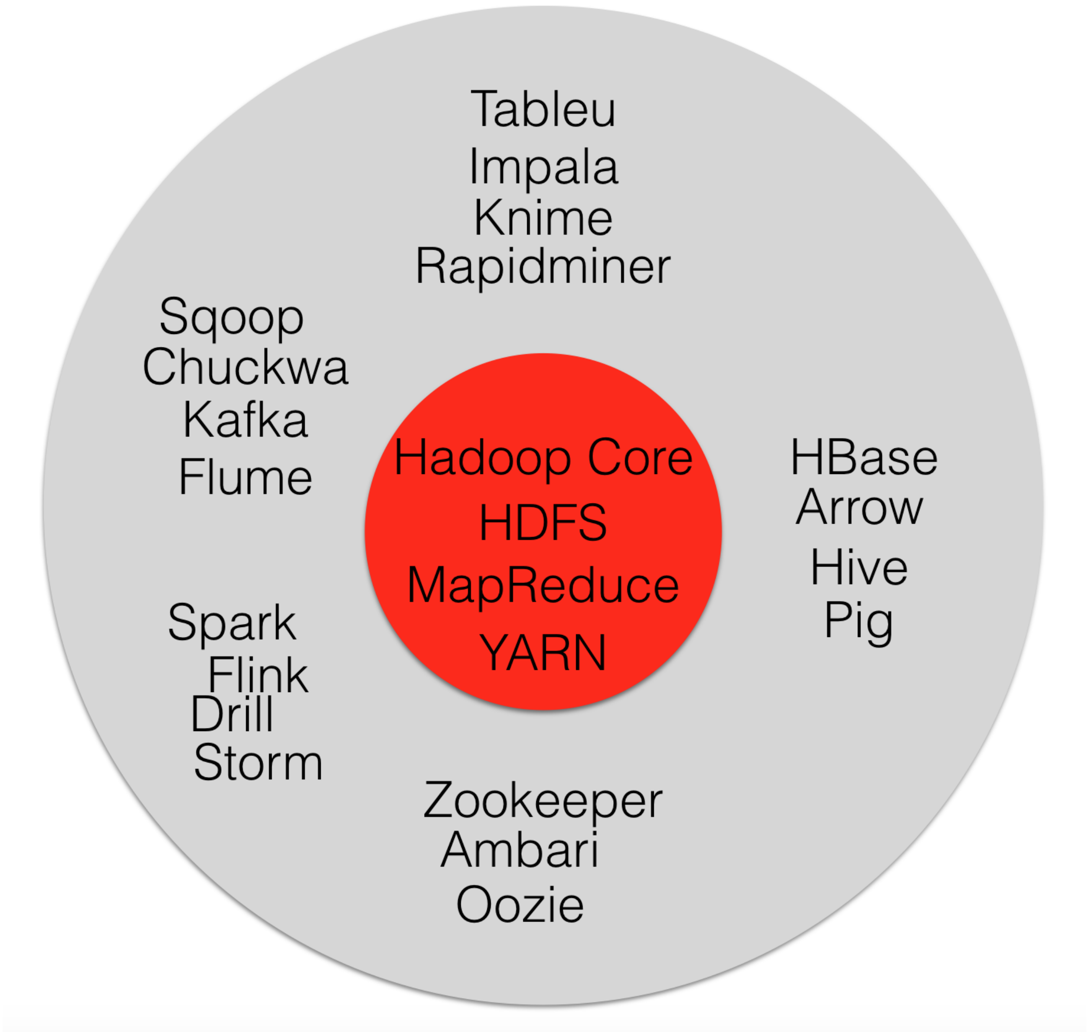

Figure 8: HDFS configuration *fehlen*

This gives you superior fault tolerance. If one server goes down, then your data stays available on a different server.

Another great thing about HDFS is, that there is no limit how big the files can be. You can have server log files that are terabytes big.

How can files get so big? HDFS allows you to append data to files. Therefore, you can continuously dump data into a single file without worries.

HDFS physically stores files different then a normal file system. It splits the file into blocks.

These blocks are then distributed and replicated on the Hadoop cluster. The splitting happens automatically.

In the configuration you can define how big the blocks should be. 128 megabyte or 1 gigabyte?

No problem at all.

9.3.3 Hive Warehouse

9.3.4 Impala

9.3.5 Time Series Databases (?)

DalmatinerDB InfluxDB Prometheus Riak TS OpenTSDB KairosDB Elasticsearch Druid

9.3.6 MPP Databases (Greenplum)

10 How to Visualize Data

10.1 Mobile Apps

10.1.1 Android & IOS basics

10.1.2 How to design APIs for mobile apps

10.2 How to use Webservers to display content

This section does not contain any text that's why the page is messed up

10.2.1 Tomcat

10.2.2 Jetty

10.2.3 NodeRED

10.2.4 React

10.3 Business Intelligence Tools

10.3.1 Tableau

10.3.2 PowerBI

10.3.3 Quliksense

10.4 Identity & Device Management

10.4.1 What is a digital twin?

10.4.2 Active Directory

11 Case Studies

11.1 Data Science @Airbnb

11.2 Data Science @Netflix

11.3 Data Science @Uber

11.4 Data Sciecne @Zalando

12 DEV/OPS

12.1 Hadoop

Listen to an introduction to Hadoop for Data Scientists: [1]

12.1.1 Hadoop Cluster setup and management with Cloudera Manager (for example)

12.1.2 Spark code from coding to production

12.1.3 How to monitor and manage data processing pipelines

12.1.4 Oozie

12.1.5 Airflow Application management

12.1.6 Creating Statistics with Spark and Kafka

References

[1] J. Ely and I. Stavrov, *Analyzing chalk dust and writing speeds: computational and geometric approaches*, BoDine Journal of Mathematics **3** (2001), 14-159.

List of Figures

List of Tables