VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

+64 4 463 5341
office@ecs.vuw.ac.nz

Building A Scalable, Production Ready Development Pipeline For Machine Learning And AI Services Using Modern Continuous Integration Practices And Infrastructure As Code Solutions

Zoltan Debre
300360191

Supervisors:
Ian Welch
Bryan Ng

Submitted in partial fulfilment of the requirements for
Master of Computer Science Degree

COMP 501
Research Essay in Computer Science
2019

Abstract

Continuous integration and development pipeline are commonly used technics in traditional software development. Data management and data analysis has been revolutionized by big data and machine learning tools. Our vision is to merge these technologies and technics to bring together professionals like software engineers, data engineers and platform engineers. Companies and researchers are looking for solutions to use these new tools and skills to automatize data processing and continuously deploy big data driven reports, analysis, and results. This research essay lists tools, processes and help to determine a possible solution for building a data focus continuous integration pipeline.

Table of Contents

1. Introduction

Developing, testing, and running a scalable platform in production is a complex task. Either a research project or a commercial product, both are constructed from various components. These applications or service components help decouple functionality. The Twelve-Factor App methodology (Wiggins, 2017), Enterprise Architecture Patterns (Martin Fowler, 2003) showed breaking traditional monolith systems to microservices increases maintainability. Our running environment is more reliable, because building a redundant, scalable platform is easier.

Data processing and management systems have been changing. They move from a simple database, where we run SQL queries, to a wide functionality architecture. Components such as real-time data streaming, big data analyst tools, and machine learning frameworks are smaller connected components. On one side, managing microservices is simplifies development. In the other hand, connecting independent parts together can be complicated and it might involve new tools and practices.

Developing, testing, and releasing a multi-component service cannot work without a dedicated, automated testing and releasing pipeline.

In each phase of the pipeline lifecycle we have to replicate the running infrastructure.

I **hypothesise** that adapting Infrastructure of Code (IoC) and devops practices into Data Pipeline management can speed up development and release lifecycle, as well as improve reliability and maintainability.

In order to build a development and release pipeline, we have to learn the best practices and collect the tools used the most often by the industry. The goal of the research is to provide a guideline to setup a developer pipeline for big data applications.

For validating best practices, the information is collected from three different direction. One of the most important source is the industry expert interviews. I interviewed three experts who are leaders on data platform and devops area. They represent companies where data plays significant value. The size of the companies varies, from large organisation to a service provider start-up. The experts described the actual market needs, necessary skills, and tools. Interview summary is attached in the appendix. Their views and suggestions are explicitly stated in the research in different sections.

Furthermore there are academic and industry sources to help underline commonly used practices. Additionally, we also share our own experience of building developer pipeline.

End of this research, I propose a plan about building a demo Data Pipeline, that can be used for developing, testing, and releasing a machine learning focused application platform. The demo data pipeline will be evaluated with industry subjects by them following instructions on how to set up a demo experience. Then a short survey will be presented to them to report back on the demo environment.

Measuring the activity of development and release lifecycle can be validated by measuring the time of releases on the pipeline. Tracking the time between the new code merge and its deployment of production shows the speed of deployment process.

Improving maintainability and reliability of an application can be validated by measuring the automated test coverage. Unit and acceptance test frameworks improve maintainability of the application components with providing a test harness, a protection layer around the implemented logic. Maintaining the source code of an application improves the reliability because the connected tests protect the previously implemented features by alerting the maintainers about unexpected changes.

1.1 Motivation and questions

As active contributors in the IT industry, we have seen various trends, hypes, and interesting periods in the past three decades. The growing big data services and the importance of data science are undoubtedly one of the most important IT segments in the last few years. In a previous work I analysed the influence of data management, data valuation, and big data influenced business models. (Debre, 2017) I investigated, the data behaves the same way as oil: it is more valuable when the raw data is processed, analysed, categorized, and used for a higher level analysis. Machine Learning and AI require incredible amount of raw data which only can be managed within a controlled environment.

Why Devops is important to Data Science?

Data management services, data warehouses, as I already mentioned in the introduction, have been transforming into more complex systems. Historically, data scientist worked in isolation and ad hoc way. For instance, they received a snapshot from an active database. Then, they run analysis on dated information mass. We reached a level where the amount of data is incredibly huge. The value of the data is high when it is fresh and up-to-date. New data streaming solutions can help manage this important business requirement. However, the management of these tools involves skills that extend beyond traditional data science.

Merging technology and using proven techniques in data engineering can help elevate this important segment to the next level. We are looking for an answer how devops practices can help us to build and run data focused developer pipelines. For understanding this segment of computer science, I collect tools and practices. I also describe a possible implementation and process framework for managing a data science focused demo application.

2. Related methodologies and concepts

The main goal of this chapter to list and describe the most important terms and concepts; and to set a foundation for our further investigation.

2.1 About DevOps

Devops word comes from Development and Operations (DevOps). It covers a wider area, which is mainly a union of people, products, and processes. It also describes how they work together to achieve a continuous delivery of a product. This expression usually includes agile planning, Continuous Integration, and monitoring the working product. (Guckenheimer, What is DevOps?, 2018)

Why devops practices are important for Big Data services?

Adopting devops practices for Big Data projects help us to achieve as similar velocity of improvement as other area of the software architecture segments do. We can talk about three important aspects: reliability, scalability, and performance. Ash Munshi talked about these components in one of his interviews. As he states devops practices and continuous integration help big data experts keep their system reliable. The system has to work in a big scale, and it has to be performant. A big data and machine learning tool running on a huge cluster while managing hundreds of computers. Dataset is large and the data changes rapidly. Additionally, most of these services, big data and machine learning applications often used by lot of people simultaneously. The complexity of these architecture leads us to adapt the same practices that we use in classic software development projects. (Wells, 2017)

2.2 Continuous Integration

Modern software development is fast and agile. Nowadays, it is a common requirement to deliver solution, new features or changes as quickly as possible. Our software project is not only the code running in production on a server or as a downloadable application. It is also our automated test that checks whether our code behave as expected. When we change our code we also have to run our test. If the test fails, we need to check our application and fix it quickly. If this cycle of coding and testing is fast, than we can deliver at much higher pace. The most important point is to accelerate our code-test-deploy cycle. (Guckenheimer, What is Continuous Integration?, 2017)

This non-stop coding and testing activity can be continuous only, if we automate this cycle. Continuous Integration (CI) is simply the process that automates our application build and testing process. We change our code. Usually these code changes are committed in a version control system. The version control system triggers the build and run the testing environment for validate our latest changes. (Fowler & Foemmel, Continuous Integration, 2006)

Continuous Integration is a process which represents the following best practices:

- source code stored in a shared version control system,
- new changes committed in a feature branch,
- peer code review checks the quality of our changes,
- new code will be merged in the master (main or trunk) branch,
- avoiding from "merge conflicts" the new code has to be merged quickly and continuously,
- one "pull request" (PR) can cover only a tiny task, small feature, little improvement or manageable code refactoring.

PR has to be reviewed, and it can be merged into a master branch but only if the build and test are "green", so the CI system also approves these changes. CI keeps our main branch deployable and clean. It means our codebase is always in a deployable state. However, in order to achieve this criteria, we have to keep the more complex changes behind a "feature flag". (But it is more likely a development practice and continuous delivery topic than only CI related.)

2.3 Continuous Delivery

Continuous Delivery process provides a constant stream of customer values with building, testing, configuring and deploying our application into production. The main addition to the previously mentioned CI process is that we can have different stages between the actual code change (development) and the final stage (production). The actual code change "commit" travels through a Release Pipeline which contains different testing or staging environment. (Guckenheimer, What is Continuous Delivery?, 2017)

Transitioning between these stages can happen only when an authorized person, typically someone from business area or a decision maker, officially approves.

Additionally, we could have more "deployment ring" close to the final production deployment. It means we can have a release for a limited user base, so only a special group of people accesses the production environment. It is called "canary" deployment ring. Its main purpose is the final testing before a wider public release.

We already mentioned the importance of "feature flags". CD helps us keep our certain features and experiments hidden. We can use feature flags to release changes only to a limited pilot group.

There is one more important expression related to CD. It is the "blue/green deployment". It means we deploy a production ready version on the "green" ring, however we use load balancer techniques to direct our traffic to the live "blue" servers. We can transition a portion of our visitors with load balancer traffic manager to the new "green" environment. When we do not see any problem with the new version, then we can finish the new deployment and the "green" server became the new "blue". If any problem would arise, we can switch back immediately to the old version.

2.4 Continuous Deployment

When we use Continuous Delivery practice without Continuous Deployment, it means we still have a certain day or period of time when the software can be released. It is usually called the “release day”. Continuous deployment means we do not block our production deployment with human interaction and approval. The only block can occur during the automated pipeline process for example, one of our test fails. Otherwise the committed code automatically deploys to production. (Pittet, 2019)

2.5 Infrastructure as Code

A modern web application is built of different components, such as database, queue management system, static web artefacts, and dynamic backend API. These components are running on an infrastructure which connects them together. The infrastructure contains networks, virtual machines, load balancers, and connection topology. The main purpose of the Infrastructure as Code (IaC) is to manage the connected infrastructure. (Fowler, InfrastrctureAsCode, 2016)

An important attributes of IaC is being descriptive and declarative. It describes the infrastructure with configuration code, that can be stored in source code versioning system. It has version numbers which helps to follow the history of changes. A given version of the code always generates the same infrastructure and environment. (Guckenheimer, What is Infrastructure as Code?, 2017)

2.5.1 Usefulness of Infrastructure as Code

There is an interesting problem with infrastructure environments. It is called “environment drift”. It means we use a slightly different environment for development, testing, staging and production deployment. It can be modified by different teams. Sometimes these changes can surprise others who contribute to the development lifecycle. When different team maintains their environment and they invoke changes on servers manually, they create a unique “snowflake”. (Chatterjee, 2017)

Martin Fowler wrote “snowflake – good for a ski resort, bad for a data center”, because a snowflake server is difficult to reproduce. (Fowler, SnowflakeServer, 2012) We lose the scalability, because a manually patched server can be in sync with other servers if we manually run same commands on them also. When we talk about cluster of servers, keeping machines “independent” is a tedious and an almost impossible process.

A practical solution to this problem would be if each environment would use identical infrastructure with the same configuration. So that, when a team deploys the application for development, testing or production, the same practices and patterns would be followed. This configuration and settings are essentially the code which determines the infrastructure.

To avoid environment drift IaC can help us to make our deployment process automatic while keeping the changes always in sync, and helping with the adaptation of best practices.

2.5.2 Idempotence as key principle

Important principle of IaC is idempotence. The key feature that we always build our special target environment from the same starting state. It means we can delete and recreate the whole infrastructure any time while getting the same result. Changes can be made with altering the configuration code (for example a JSON or YAML configuration file). However, the changed code will have different version number. This change goes through on the same approval process as the application source code, so that it is documented and trackable.

IaC let us recreate production environment before it goes live. Testing our infrastructure is crucial The test is more accurate if it replicates the real production environment. Because there is not manual configuration, everything is done by scripts and automation. Using IaC speeds up deployment and make it more reliable. (Guckenheimer, What is Infrastructure as Code?, 2017)

Vienna University of Technology, Austria and IBM T.J. Watson Research Center published a paper about a testing approach of IaC. They state that deployment automation should be idempotent. Idempotent code can guarantee convergence and repeatability and it has to be tested. They come up with a model-based testing framework, which mainly target Infrastructure as Code tools. (Hummer W., 2013)

The main idea is, we setup a state transition graph, so we know from which state to which state the infrastructure moves. The graph helps to generate test cases which could run in separate docker containers.

Modern IaC tools such as Chef or Puppet, provide interface (resources), that guarantee idempotent environment. For example, if our script checks on the existence of a directory, and if that directory does not exist on the actual server, then the frameworks create a new directory.

In my opinion, the key take-away from this paper is that, we should determine the starting state and the desired state of an element of our pipeline. A testing tool checks the starting state, it runs the code which modifies the infrastructure, and it checks again the infrastructure element. The testing framework can validate if the code has changed the environment to the desired state.

2.6 Pipelines

As Martin Fowler et al. mentioned in a white paper (Fowler & Foemmel, Continuous Integration, 2006), that is the most cited source in continuous integration based on Google Scholar[1], if we would like to get the benefit of Continuous Delivery, we have to setup a Deployment Pipeline. It helps us to run our application build and test in sequence. This pipeline starts the process with a commit initiated by a software developer. A simple but efficient version is the two stage pipeline, which first run the fast, isolated unit tests; and later it runs the slower but wider cross platform integration tests.

A pipeline contains a few steps. Moving between these steps can be automatic. If it is fully automatic, it is the previously discussed Continuous Deployment. However the deployment approval process also can be part of the pipeline. In this case, as we

[1] Cited by 604 -
https://scholar.google.co.nz/scholar?cites=695927602274289959&as_sdt=2005&sciodt=0,5&hl=en

mentioned earlier, a delivery or business manager can manually approve or reject the next step. (Homer, 2018)

Standard steps of a pipeline:

1. Developer machine. Developer changes the code. Run tests locally and commit to the version control system.
2. Code review system. Other developers and testers review the code change and approve or reject it. Meanwhile the app can be built and tested in the background. The code can be moved to the next step, if the code approved, and it is merged to the "master" branch.
3. Unit testing. Run unit tests on the "merged" code.
4. Integration testing. Run cross platform integration tests.
5. Canary testing.

(Mukherjee, 2019)

2.7 Metrics of devops

One of the goals of the implementation of devops practices is to speed up development, and deploy features and bug fixes faster. Different metrics are introduced for measuring devops related improvements. For instance, in one of the surveys about devops by IDC is collected information from 15 devops related categories (Elliot, 2014). I selected the following four metrics what could be relevant in my research project.

- Average Time to Repair an Infrastructure Failure
- Average Time to Repair an Application Failure
- Average Time to Restore a Production Failure
- Timeline for Code Change Impact on Customers

My suggestion is to use in the demo project the time of code change deployment metrics, because it can be measured in demo environment. Other metrics can provide relevant information only if the project runs in real and live production environment for an extended period.

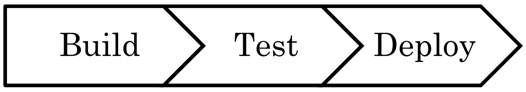
Measuring the recovery time from a failure is also a common metrics in computer system management. The acronym of this metrics is MTTR. A paper from 1981 uses the phrase of “Mean Time of System Restoral” for MTTR (Kullstam, 1981). However, in modern articles prefer “Mean Time To Recover”. (Riley, 2015) Measuring MTTR or the above mentioned Average Time to Repair/Restore metrics show similar values. The time of recovery should decrease as the team learn more about pipeline deployment.

3. Devops tools and techniques

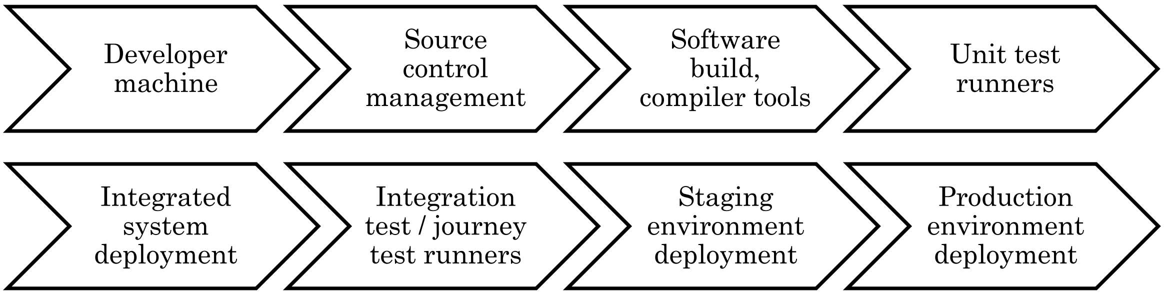
The previous section listed the most important components of a standard development and deployment pipeline. The support of automated processes is in the spotlight. Various tools, commercial and open source services are released in the last few years. The goal of this section is to show an overview and a list of the commonly used tools. Comparing similar services helps us to find an ideal solution for demo pipeline.

3.1 Differences between traditional pipeline, Data Pipeline and ETL Pipeline

The standard software developer pipeline has three stages:



Our experience suggests that in reality there are more stages and components:


Developer machine
Source control management
Software build, compiler tools
Unit test runners
Integrated system deployment
Integration test / journey test runners
Staging environment deployment
Production environment deployment

The pipeline above mainly describes the process of the development and deployment. However, the phrase of "data pipeline" is often used for the "product" which processes some data. Traditional data processing pipeline is usually called "ETL Pipeline". The more modern analytical focus pipeline is called "Data Pipeline".

3.1.1 ETL Pipeline

ETL is the acronym from extracting, transforming and loading. We use ETL Pipeline for systems that we extract information from a source (for example from other application, log file, or other database). These systems process data in a certain time of a day and extract a huge batch. Chunk of record is transformed to different format or structure, and finally they loaded in another common database. The legacy ETL pipelines were mainly isolated, separated, and static. The modern ETL pipelines run in the cloud. They use elastic solutions to be more scalable, which also helps to stay up to date, almost real-time. (Alley, What is ETL?, 2018)

3.1.2 Data Pipeline

Data Pipeline is try to be more than a modern ETL and mainly used in generic way. One of the main attributes is processing data in real time by using streaming tools. However, the main point of a data pipeline is to move data from one system or location to another. The information moved may be processed or transformed. (Alley, What is Data Pipeline?, 2018)

3.2 Important building blocks of a pipeline

The following five areas play important role in a software development pipeline:

1. Version control system
2. Pipeline management tool
3. Container management tool
4. Docker registry and docker repository services
5. Infrastructure of Code programming frameworks

In this section, I focus on tools can be used to manage the deployment process of a data pipeline.

We prefer services which are open source, free to use, widely supported by the community, documentation detailed, maintained, and easy to understand. The developer experience (DX), which means how easy or difficult to use a tool could be one of the consideration factors. However, DX is subjective, provide an easy to use command line interface and/or a graphical user interface help during the onboarding process.

Tables below list the investigated tools, as well as the corresponding documentation websites. There are notes about commercial state, popularity, our subjective experience about usability and possible disadvantage if we would use it in our project.

3.2.1 Version Control Systems

Github	https://github.com	- free option available - wide support across the industry - larger group support in private repositories available only in paid version - pipeline services via third party partners
Bitbucket	https://bitbucket.com	- free option available - wide support across the industry - in house installation available for corporate clients - strong integration with commonly used project management tool (Jira) - pipeline services via third party partners
Gitlab	https://gitlab.com	- free option available - open source - in house hosted installation possible from source code - built in continuous integration service

3.2.2 Pipeline Management Tools

Drone CI	https://drone.io	- limited free option available - easy to install - command line tool - modern interface - easy to learn
Concourse CI	https://concourse-ci.org/ https://concoursetutorial.com	- easy to install - command line tool - modern interface - easy to learn - great support from Pivotal
Jenkins	https://jenkins.io	- easy to install - wide industry support - classic and modern interfaces - deeper learning curve

		- wide range of plugins

3.2.3 Container Management Tools

Kubernetes	https://kubernetes.io	- free, open source - growing popularity, chance to become the standard container management tools - deep learning curve - containerized cluster environment, independent from cloud based services
Docker-compose	https://docs.docker.com/compose/	- free - simple to use, for this reason it is commonly used in development - production ready, however need extra tools (Swarm) for using it in a cluster
Open Shift	https://openshift.com	- commercial from Red Hat (IBM) - container application platform
Nomad	https://www.nomadproject.io/	- a lightweight task scheduler across a cluster of machines - from HashiCorp - open source - simple binary

3.2.4 Docker Registry, Repository

Docker Hub	https://hub.docker.com/	- free for public containers - paid for private containers - easy to use
Quay	https://quay.io/	- paid (only 30 days free trial) - easy to use - part of CoreOS, which is part of Red Hat (IBM)
Google Container Registry	https://cloud.google.com/container-registry/	- paid, but limited free tier is available - extra focus on security and vulnerability scanning

		- can be obvious choice for GCR users
AWS Container Registry		- paid service - detailed documentation on AWS - can be obvious choice for AWS users

3.2.5 IOC Frameworks, Tools

Ansible, Chef, Puppet	https://www.ansible.com/ https://www.chef.io/chef/ https://puppet.com/	- help to configure individual computers or individual instances in a cluster - these tools mainly procedural, so an instance can change independently, there is no guarantee of immutability
Terraform	https://www.terraform.io/	- declarative, the dictated environment will be forced to the cluster - immutable, simple, fast - platform agnostic (it can manage Google Cloud Platform, Amazon AWS, etc.) - wider adoption, more articles and documentation
Ksonnet	https://ksonnet.io	- open source Kubernetes configuration tool - quite new, documentation, best practices are not so detailed

3.3 Other interesting projects

We can find other tools, although few of them are in fairly early stage. However, the trend shows, more development teams and companies would like to solve the problem of cluster management, deployment, and automatic scalability. We are entering an era where distributed cloud computing will be easier to manage and more maintainable.

I heard of interesting projects during the interviews. The experts suggested the following tools for further investigation:

- Helm: https://helm.sh/ - A popular package manager for Kubernetes
- Kapitan: https://kapitan.dev/ - Generic templated configuration management for Kubernetes, Terraform
- Kustomize: https://github.com/kubernetes-sigs/kustomize - Kubernetes customization
- Jsonnet: https://jsonnet.org/ - Data templating language, extension of JSON
- Packer: https://www.packer.io/ - Creating machine images automatically
- Prometheus: https://prometheus.io - Monitoring Solution
- FlyWay: https://flywaydb.org/ - Version control for database

3.4 Appropriate tools and how they help solving our data-devops pipeline problem

When I listed the tools above, I put emphasis on, the open source and free solutions. But it does not mean that they are not appropriate for commercial usage. Nowadays, most of the tools provide free access of other open source or educational project. However if an application is commercial, there are more appropriate licencing and support options available.

Any combination of tools from the list above would be appropriate, because most of them provide similar solution in their field. However, based on my personal preference, I would suggest an architecture that can provide an efficient and modern environment for a standard data-devops pipeline.

1. Source control: GitHub if our project is public. GitLab if we would like to maintain and store the source code in our environment, because GitLab can be installed locally.
2. Pipeline orchestration: Jenkins can be a safe choice, because it is the most popular and functional rich. However, Drone and Concurse CI are more simple. They also provide a more focused solution, so they can be a good choice in case we do not have special or custom requirement, and if we would like to spend less time on configuration.

3. Container management: If we use Kubernetes then the whole architecture and cluster will be packed in a standardized format, which can be deployed in any cloud service. In this case, we can use the same architecture on our development machine, on our local cloud or we can move it into Amazon AWS or Google Cloud.
4. Docker registry: If our project is fully open source Docker Hub is a great choice. If our project is deployed on one of the popular cloud providers (AWS, GCP), then a more obvious option would be the provider's registry service.
5. IOC framework: Terraform help us to maintain the infrastructure in an abstracted way without locking our system into any cloud service. It allows the configuration to be more generic. Experimenting with Ksonnet is also a good idea because it can help speed up Kubernetes management.

3.5 New way to work, new types of roles

New tools and new concepts help us to manage the challenges big data revolution brought in our field. But not only the technology changes, people in this field as well.

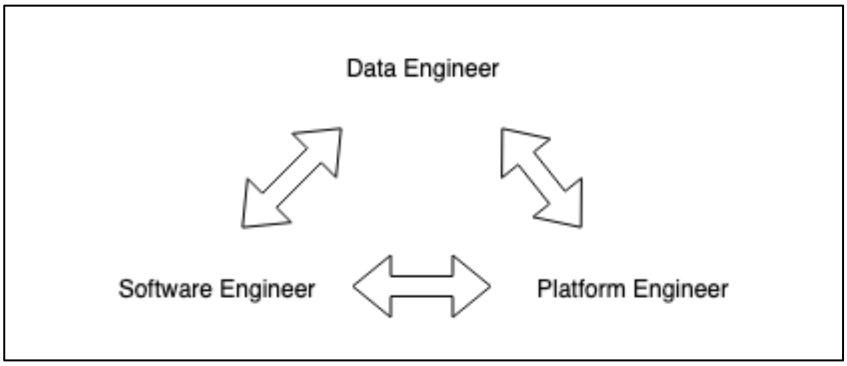


Figure 1: Building a big data CI pipeline brings together engineers from different area

As the experts highlighted, data science and business intelligence were usually independent departments in the structure of a commercial organisation. For example data analysts were part of a business related segment such as marketing or financial department. The importance, the complexity, and the dependency to computer science and technology brought data science closer to the information technology world. Also new type of roles are formed. For instance, there are data engineers.

They are more technical than data analysts and not isolated anymore. One of their next steps in this journey is to adopt more software development practices. In the field of software development, a decade of experience for using continuous integration and devops practices already exist. These practices are facilitated the software development, deployment processes, and have significantly changed the pace of the software development lifecycle.

4. Data pipelines and devops pipelines

When we talk about pipelines we basically talk about two applications. One of the applications is our real system, which runs in production and serves the business logic. It can be a streaming data pipeline with a web app or an API. The product can also be a traditional downloadable application. We can call it "The Product". The other application is the devops pipeline which moves "The Product" across the development and deployment phases. For this reason, two layers need to be considered all the time: the devops pipeline and the product (data pipeline).

Our focus in this essay is to build a big data related "product" with data pipeline which is supported by a devops pipeline. In the previous section we already listed our devops pipeline components.

4.1 Architecture related concepts

We cannot skip to clarify the following patterns and methodologies, that can help us to understand and build a more ideal architecture.

4.1.1 Importance of streaming

Martin Kleppmann expresses a view about data intensive applications in his popular presentation titled as of "Turning The Database Inside Out". He suggests that instead of changing the state of a database record directly, we should write an event stream, an event log instead. Based on this stream, we can create "materialized views". These views are for reading optimized database tables, an always up-to-date cache. Using that approach, the writing and reading would be decoupled and separated. He emphasizes Apache Kafka as a great tool for managing event streams; and Apache Samza as a framework for transforming event streams to a "materialized view" table. (Kleppmann, 2015) (Kleppman, 2017)

4.1.2 KAPPA Architecture

Kappa Architecture is a data-processing architecture design concept based on Lambda Architecture. Nathan Marz's Lambda Architecture suggests that a stream should flow in two directions. One direction processes the data in batch. The other direction keeps the flow real-time. For example, the batch path could use Hadoop and Elephant DB, the real-time path could utilise Storm and Cassandra. The idea is that using a combination of batch processing and a parallel real-time flow provides a consistent, available, and/or partition tolerant solution. We can almost "beat the CAP theorem" (Marz, 2011). Jay Kreps went further with this concept by suggesting we do not need the batch process path anymore, we should use real-time stream processing instead. Apache Kafka is capable and fault tolerant to achieve the same result as Lambda Architecture, but with a much simple way. (Kreps, 2014)

4.1.3 OLTP vs OLAP

In terms of databases we also need to talk about the two different types of implementations of database systems.

In one hand, there are the Online Transaction Processing (OLTP) applications focusing on capturing and creating data real time. They are part of the core product and business, they process large volumes of transactions. (Ed., online transaction processing, 2019)

In the other hand, we could have an Online Analytical Processing (OLAP) system. That main purpose is to analyse data, build data warehouses, and collect data from different source systems. (Ed., online analytical processing, 2016)

4.2 Stateful and stateless applications

Modern software architecture promotes the usage of microservices. (Lewis & Fowler, 2014) Smaller application scope helps to develop and maintain the little services efficiently. These applications can be stateful or stateless. Stateful application preserves state of active objects. For instance, it keeps track of user sessions or some business logic. In contrast, stateless application does not store any data or information about previous processes or requests, so it works as a pure function, so that the request with the same input will generate the same response.

Microservices can be deployed in containers. We expect a container can be launched and destroyed any time. It means, the app cannot store any state. For this reason, a containerized architecture prefers stateless application components.

However, we still have to maintain states. For this reason we use databases or key value stores. These services can be part of our cluster, although they are permanent solutions, so we cannot simply delete and recreate them.

It is important to emphasize that any logic can be stateless, and they should be built in their own application and container. If we would like to store some result or state, it has to be sent to the database or the storage service. We can simplify this rule with saying that program code should live in microservice and data should be stored in database.

4.3 Event Sourcing

Other important pattern in modern software architecture is event sourcing. In this pattern, when any business logic would change the state of an object in our application, than the “change event” will be sent as an “event package” to a storage. This message will be appended to an event list. If we need the state of a certain business entity, we can just replay the event log and get the final state. In this pattern, the event store can work as a message broker also, so if other services subscribe for certain type of messages, they can get a copy about the new event. (Richardson, Pattern: Event sourcing, 2018)

There is a connected pattern. It is called CQRS (Command Query Responsibility Segregation). Meanwhile event sourcing applications sending events to an event store, we continuously update a materialized view database. This database behaves as a subscriber of our message broker. So when one of our other application is requesting the state of a certain entity, instead of re-playing from events, the materialized view will be queried. One of the drawbacks of this solution is the materialized view will be eventually consistent, because the updates come from the event log, so that there could be a tiny latency. (Richardson, Pattern: Command Query Responsibility Segregation (CQRS), 2018)

4.4 List of tools for modern data services

I asked the industry experts what kind of tools they use or plan to implement in their data services. They listed a few commonly used products that are usually part of a modern data cluster. We can consider to use them in our demo project.

List of important data science and data analysts products:

- Apache Spark – Unified Analytics Engine for Big Data (https://spark.apache.org)
- Apache Kafka – A Distributed Streaming Platform (https://kafka.apache.org/)
- Apache Flink – Stateful Computations over Data Streams (https://flink.apache.org/)
- Apache Samza – A Distributed Stream Processing Framework (https://samza.apache.org) (Kleppmann, 2015)
- TerraData – Database (https://www.teradata.com)
- Elastic Search – Open Source Search (https://www.elastic.co)
- Apache Cassandra – Database (https://cassandra.apache.org)
- Kong – The Cloud-Native API Gateway & Service Mesh (https://konghq.com/)

We would like to keep our demo application as simple as possible. In order to do so, I consider using only a few components. I also would like to focus on achieving a more stateless architecture by using an event sourced implementation. Therefore, Apache Kafka as a message broker and Apache Spark as data processor can be an interesting combination; and adding Kong as the load balancer.

5. Future work: proposed architecture for a case study project

In the previous sections we described a few components for the demo architecture. In this section I summarize my proposal.

The pipeline architecture manages two layers. The first layer is the big data demo product itself. The second layer is the pipeline management architecture. A possible solution is to wrap both layers in one Kubernetes cluster. It helps us to deploy the solution in different cloud provider. Further improvement could be that at end of the pipeline, the pipeline manager code deploys the product in an independent production ready Kubernetes cluster.

5.1.1 Product / Data Pipeline Layer

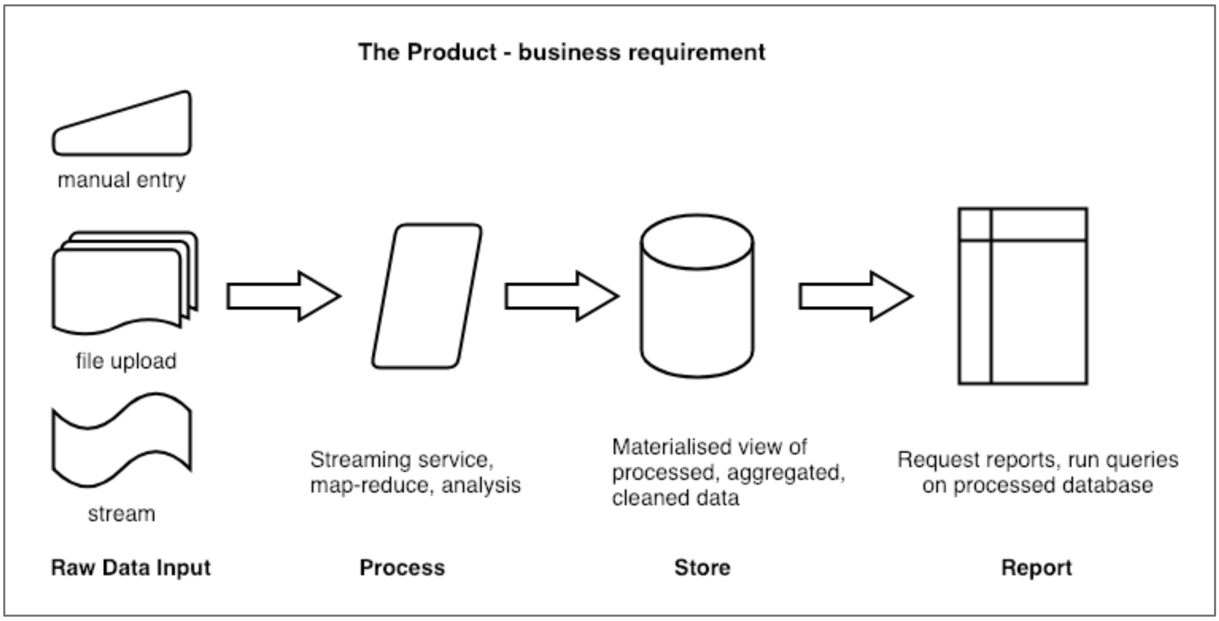

The Product - business requirement
manual entry
file upload
stream
Streaming service, map-reduce, analysis
Materialised view of processed, aggregated, cleaned data
Request reports, run queries on processed database
Raw Data Input
Process
Store
Report

Figure 2: A standard data streaming pipeline

Above 4 stages visualize a simple business requirement. The raw data enters in the data processing pipeline. It can be a manual entry on a website, or a CSV file upload, or a JSON data stream via a standard web protocol.

The first microservice will accept the data and stream to an Apache Kafka instance. Other services can subscribe to this event channel. One of the services can be an Apache Spark implementation that runs a special algorithm. (This segment can be replaced with a simple Python based algorithm runner app.) In the end of the data pipeline, the processed information will be visualized.

In the simple demo we create the following microservices:

- Container #1: Data input service
- Container #2: Apache Kafka
- Container #3: Apache Spark / Python App
- Container #4: Data Visualisation App

5.1.2 Devops Pipeline Layer

For running our development pipeline we have to implement the following services.

- Version Control System (Cloud based GitHub preferred)
- Container #5: Devops Pipeline runner - Concurse CI

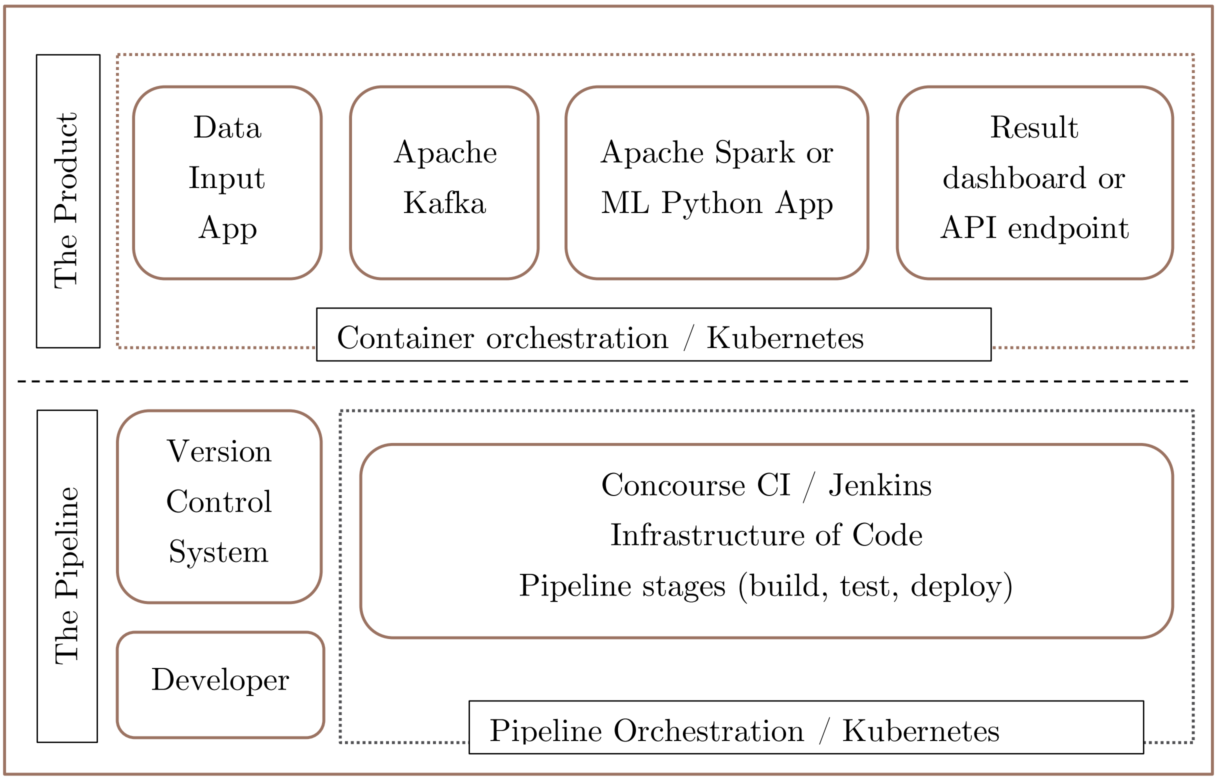

The Product
Data Input App
Apache Kafka
Apache Spark or ML Python App
Result dashboard or API endpoint
Container orchestration / Kubernetes
The Pipeline
Version Control System
Developer
Concourse CI / Jenkins
Infrastructure of Code
Pipeline stages (build, test, deploy)
Pipeline Orchestration / Kubernetes

Figure 3: The architecture of the two layers

Further notes:

- Each component will be wrapped in a docker container.
- All container will be stored on Docker Hub.
- Containers will be wrapped in a Kubernetes Cluster and managed by Terraform and/or Ksonnet.

6. Conclusion

The beginning of our research the question was how we could use software development related devops practices with big data focused applications. I collected traditional devops tools, from which I selected a few than can help further my research. Additionally, I gathered the most commonly used big data components that can play a significant role in the demo application.

The industry experts instructed me to find best practices and to identify special challenges in terms of big data applications and devops pipelines.

I discussed about how the industry has changed, and how new professions and roles have evolved withing organisations.

Finally, I proposed an architecture for further research, where I will build a demo devops pipeline for a big data application.

My **hypothesis** stated that adapting Infrastructure of Code (IoC) and devops practices into Data Pipeline management can speed up development and release lifecycle, as well as improve reliability and maintainability.

I learned by interviewing industry experts and reviewing existing literatures, and using my experience with IoC and devops practices, that it organizing a reliable and maintainable data pipeline is important and speed up development and release lifecycle.

This qualitative evidence lends support to the acceptance of the hypothesis. I intend to investigate this further by transferring this learning into an industry motivated development project. I will validate the improvements to reliability and maintainability through a mix of quantitative and qualitative methods.

7. Appendix

7.1 Interviews summary

I invited 3 industry experts to share their view about big data management pipelines. Personal interviews approved by the Victoria University of Wellington Human Ethics Committee. (ResearchMaster application reference number: #0000027081)

Participation was voluntary, our interviewees and the represented company cannot be identified.

Our participants represented different type of organisations, including large commercial institutes with huge databases and smaller service provider also.

In the following list is a draft summary from their answers:

- It is important to separate stateless and stateful applications and architecture.
- Event stream based architecture is an important concept.
- Important tools: Apache Kafka, Terraform, Helm, Kubernetes, Apache Flink
- Important software developer languages: Scala, Go because they are strong in functional programming which force developers to write more stateless solution.
- Security: Zero Trust Security Model. Useful tool is Istio.io.
- Database schema versioning with FlyWay.
- Everything is in a container.
- Experts should work together: software engineers, data engineers, platform engineers.
- Improve your software in tiny steps. The development lifecycle: monitor your application, use the feedback to improve your software further.
- Challenges in data management: GDPR, Software as a Service providers
- Performance testing is important.
- Architecture patterns, like KAPPA.
- Tools for observability is key. Prometheus, Honeycomb.io
- Tools: Kubernetes, Docker, Cassandra, Elastic Search, Hashicorp Packer, Codeship, Concourse from Pivotal, Drone, Jsonnet.org, Ksonnet from Bitnami.

- Important languages: Elixir, Scala, Go
- Tools: Terradata, Oracle BI
- Architecture: ETL Pipeline

7.2 Questionnaire

As part of the research I reached out to a few industry experts to share their view and opinion about Data Pipelines anonymously. Victoria University of Wellington Human Ethics Committee approved our research (reference number: #0000027081). The following interview guide helped me during the interview.

Please describe your role at your company.

- The purpose of this question is to learn about the interviewee, and to get context about his/her responsibilities.
- I am looking for an answer to the question of what type of roles and what type of skills you need to work with devops team in a commercial organisation.

How do you process and manage your big data flow and data stream in your organisation. Could you please describe how your data management architecture looks like?

- The purpose of this question is to get an overview about the AI and Machine Learning infrastructure.
- I would like to know what type of services and applications are being used to manage big data streams, and how they connect together.

Please talk about your development pipeline and devops practices especially if you have dedicated support for big data, ML and AI pipeline.

- The purpose of this question is to learn about the practical aspect of the big data and ML pipelines. How developers improve, commit, and add new features. Also how the code is tested and how it is released to production.
- I would like to know the tools, applications, and services being used for development, testing, and production release.

What kind of tools do you use to support devops, development and production releases.

- It is a follow up question to the previous one, so I can aquire more details about on the tools.

The following tools are commonly used in development and release pipelines. Please scale the importance of the following tools/practices/services in your environment:

- *Docker*
- *Kubernetes*
- *Infrastructure As A Code*
- *AWS*
- *Google Cloud*
- *VMware, virtual machines*
- *Distributed computing*
- *Distributed / cloud databases*
- *Microservices*
- *Jenkins*
- *Terraform*
- *Streaming technologies*
- *Big Data tools (Hadoop, Spark)*
- *Queue Management tools (RabbitMQ, Apache Kafka)*

- The purpose of this question to clearly determine the most important tools, and to get a scale of the importance of the tools.
- I would like to know how important these tools are, so I can prioritize them, then use them in future planning. Especially when I would like to build a practical tutorial for learning devops.

What skills you need if you hire new developers in your team, what they should know about developer pipelines and infrastructure as a code solutions?

- The purpose of this question is to get information about required skills.
- A developer needs much broader skill-set nowadays. I am looking for an answer to what type of extra skills a developer needs.

What type of other applications and solutions do you use for data analysis and machine learning, which was not mentioned in your previous answers because it is *not part of the development pipeline or because it is new or only in experimental phase. Maybe it is used for ad hoc analysis?*

- A follow up question to get more information about independent data analysis tools.
- It would be great to see which tools can work independently, and which tools are good candidates to add to an integrated pipeline.

How your organisation's culture changed when the company adopted modern development pipelines? Benefits and difficulties? If you have development pipelines for big data and ML services, how the automatically available information changed your company culture?

- The purpose of this question is to get more information about the human side of implementing a development pipeline.
- Generally when a new development practice is introduced to a team, the human factor is important to adapt these changes. We would like to know more about difficulties and challenges, and how it can be addressed next time.
- It is important that no sensitive information should be shared or provided. If sensitive information would come up during the interview process, it will not be recorded in the interview notes. This question would focus on general suggestions in terms of human factor consideration.

8. Bibliography

Alley, G. (2018). *What is Data Pipeline?* Retrieved from alooma.com: https://www.alooma.com/blog/what-is-a-data-pipeline

Alley, G. (2018). *What is ETL?* Retrieved from alooma.com: https://www.alooma.com/blog/what-is-etl

Chatterjee, M. (2017). *How to stop configuration drift during application deployment lifecycle.* Retrieved from IBM: https://www.ibm.com/blogs/cloud-computing/2017/04/20/stop-configuration-drift-deployment/

Debre, Z. (2017). *Big data and the market distortion factors and regulation of data network effect (Original title: Big data és az adat hálózati hatás piactorzító tényezői és annak szabályozása.).* Retrieved from MA/MSc Thesis: http://szd.lib.uni-corvinus.hu/10508/

Ed. (2016). *online analytical processing.* Retrieved from Oxford Reference: http://www.oxfordreference.com/view/10.1093/acref/9780198743514.001.0001/acref-9780198743514-e-5158.

Ed. (2019). *online transaction processing.* Retrieved from Oxford Reference: http://www.oxfordreference.com/view/10.1093/oi/authority.20110803100250478.

Elliot, S. (2014). *DevOps and the cost of downtime: Fortune 1000 best practice metrics quantified.* Retrieved from International Data Corporation (IDC): http://www.smallake.kr/wp-content/uploads/2013/07/DevOps-metrics-Fortune1K.pdf

Fowler, M. (2012). *SnowflakeServer.* Retrieved from martinfowler.com: https://martinfowler.com/bliki/SnowflakeServer.html

Fowler, M. (2016). *InfrastrctureAsCode.* Retrieved from MartinFowler.com: https://martinfowler.com/bliki/InfrastructureAsCode.html

Fowler, M., & Foemmel, M. (2006). *Continuous Integration.* Retrieved from Thought-Works: http://www. thoughtworks. com/Continuous Integration. pdf

Guckenheimer, S. (2017). *What is Continuous Delivery?* Retrieved from Microsoft: https://docs.microsoft.com/en-us/azure/devops/learn/what-is-continuous-delivery

Guckenheimer, S. (2017). *What is Continuous Integration?* Retrieved from Microsoft: https://docs.microsoft.com/en-us/azure/devops/learn/what-is-continuous-integration

Guckenheimer, S. (2017). *What is Infrastructure as Code?* Retrieved from Microsoft: https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code

Guckenheimer, S. (2018). *What is DevOps?* Retrieved from Microsoft: https://docs.microsoft.com/en-us/azure/devops/learn/what-is-devops

Homer, A. (2018). *What are release pipelines?* Retrieved from Microsoft: https://docs.microsoft.com/en-us/azure/devops/pipelines/release/what-is-release-management?view=azure-devops

Hummer W., R. F. (2013). *Testing Idempotence for Infrastructure as Code.* Retrieved from Middleware 2013. Lecture Notes in Computer Science, vol 8275. Springer, Berlin, Heidelberg: https://link.springer.com/chapter/10.1007%2F978-3-642-45065-5_19

Kleppman, M. (2017). *Designing Data-intensive Applications.* O'Reilly Media.

Kleppmann, M. (2015). *Turning the database inside-out with Apache Samza.* Retrieved from Confluent.io: https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

Kreps, J. (2014). *Questioning the Lambda Architecture*. Retrieved from O'Reilly: https://www.oreilly.com/ideas/questioning-the-lambda-architecture

Kullstam, P. A. (1981). Availability, MTBF and MTTR for Repairable M out of N System. *IEEE Transactions on Reliability, R-30*(no. 4), 393-394.

Lewis, J., & Fowler, M. (2014). *Microservices*. Retrieved from martinflowler.com: https://martinfowler.com/articles/microservices.html

Martin Fowler, D. R. (2003). *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional.

Marz, N. (2011). *How to beat the CAP theorem*. Retrieved from nathanmarz.com: http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html

Mukherjee, J. (2019). *What is a continuous delivery pipeline?* Retrieved from Atlassian: https://www.atlassian.com/continuous-delivery/pipeline

Pittet, S. (2019). *Continuous integration vs. continuous delivery vs. continuous deployment*. Retrieved from Atlassian: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

Richardson, C. (2018). *Pattern: Command Query Responsibility Segregation (CQRS).* Retrieved from microservices.io: https://microservices.io/patterns/data/cqrs.html

Richardson, C. (2018). *Pattern: Event sourcing.* Retrieved from microservices.io: https://microservices.io/patterns/data/event-sourcing.html

Riley, C. (2015). *Metrics for DevOps.* Retrieved from devops.com: https://devops.com/metrics-devops/

Wells, J. (2017). DevOps for Big Data: Q&A With Pepperdata's Ash Munshi. *Big Data Quarterly, Vol 3, Iss 2*, 31-32.

Wiggins, A. (2017). *The Twelve-Factor App.* Retrieved from 12factor.net: https://12factor.net/