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

Implementing a software development pipeline and adopting DevOps practices for helping data scientists to solve data analysis problems

Zoltan Debre
300360191

Supervisors:
Ian Welch
Bryan Ng

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

COMP 589
Project Report in Computer Science
2020

Abstract

Data scientists continuously bootstrap new projects to solve data analysis and big data related problems. Traditionally they mainly focus on developing data models to solve unique questions, and to use ad-hoc software development environments.

At the same time, a traditional application and software development project are using techniques such as continuous integration, continuous delivery, and software development pipelines to accelerate iterations while promoting the reuse of previously extended development environment.

There is an opportunity for data scientists to adopt these new tools and skills to automate data processing in order to continuously deploy big data driven reports, analysis, and results.

This project provides a reusable software development pipeline for data scientists. This report is a detailed description and evaluation of building this development pipeline. The project brings together software, data and platform engineers to develop a cloneable and reusable pipeline template which can be used in any Python based data analysis, machine learning or artificial intelligence software development.

The pipeline reusability has been evaluated by data science practitioners in multiple production targeted data analysis projects. The pipeline is an out of the box solution which uses GitLab source control management, GitLab CI/CD pipeline management, Kubernetes manifests, and Google Cloud implementation.

Table of Contents

1. Introduction

It has been investigated in my previous research how big data and data science projects are managed, which tools are used and what are the best practices. As part of the research interviews with industry experts provided information about real world problems and solutions. Our observation highlighted that data science projects are often individually developed using an ad-hoc architecture, procedural programming, furthermore the lack of test and manual deployment was also a problem.

1.1 Motivation

The motivation was to provide a more opinionated environment, with predefined tools and practices, for standard data science project to ease bootstrapping. Additionally with the help of a prebuilt pipeline process, we assume the development cycle could be accelerated and deployment tasks can be fully automated.

This is a follow-up project which is based on the preliminary COMP 501 research essay of 'Building a scalable, production ready development pipeline'. This part of the research focuses on the implementation of the pipeline and its evaluation which is based on user/usability testing.

1.2 Problem and questions

The original problem was that data analysis projects use ad-hoc development environments, usually script based implementation which can be used only once. The lack of testing, the limited usage of software design practices (e.g. separation of concerns, package management) decrease maintainability and portability.

Our hypothesis with providing a template project for data scientists with a working DevOps pipeline and helping adapt modern development practices could speed up the development process, and increase reliability and maintainability.

1.3 Objectives

The purpose of this project and documentation is to provide guidance for bootstrapping and developing data science focused applications and evaluate our template in production environment. During this project we experiment with the following topics:

- Implementing a continuous development pipeline to accelerate software development cycle.
- Creating a real world application based on sample project for data scientist to introduce software development practices.
- Promoting test driven development methodologies.
- Using "12 factor app" principles in data science projects.
- Following Agile practices and scrum rituals.
- Working with data scientists and data engineers to evaluate our development pipeline implementation in practice.
- Using a private GitLab project to validate the pipeline.
- Determining key areas where knowledge sharing is crucial.
- Experimenting with DevOps related technologies.
- Finding a simple approach, and document it as a tutorial for non-developer audiences.
- Creating a reusable framework, and present it with a "hello world" project.

Milestones:

- Phase 1.
    - Experimenting with DevOps related technologies.
    - Researching docker, source control and pipeline management software.

- Phase 2.
    - Building the foundation of the pipeline.
    - Building the foundation of the project and run the pipeline with basic tests.

- Phase 3.
    - Interviewing data scientist to evaluate our pipeline with extending our pipeline with data analysis feature.

    - Using private GitLab project to work with data scientists and evaluate our pipeline implementation in practice.

- Phase 4.
    - Documenting and improving our framework.

1.4 Overview of this project report

This project report is structured as follows.

- Chapter 1. Introduction and motivation.
- Chapter 2. Description of the conclusion of the previous research and how the original proposal changed in this project.
- Chapter 3. Design and a description about the project implementation.
- Chapter 4. Tutorial and details about how to use the pipeline in a new project
- Chapter 5. Evaluation of the template pipeline. We summarize our evaluation which is based on test users' feedback.

2. Background and related work

2.1 Context

In our previous research (Debre 2019) DevOps practices and suggested solutions were discussed. Interviews with experts helped to determine gaps in data focused projects. The research showed that data scientists focus mainly on solving an ad-hoc problem instead of building a software solution which can be reused, maintained and improved further. Additionally we learned what kind of tools and programming languages data scientists preferred. Python and Python based tools are their main environments.

We reviewed modern DevOps pipelines best practices that were already adopted in standard software development projects. Our suggestion was to increase cooperation between Data Engineers, Software Engineers and Platform Engineers. However setting up a DevOps pipeline can be done in the early stages of a project. Software Engineers can force suggested practices with rules in the pipeline, including mandatory testing, code coverage, and code review. For this reason we proposed a pipeline template that can be used by data scientist in data focused projects.

During the review a few important building blocks of a modern pipeline were listed. It was determined that the usage of containers, especially Docker containers are inevitable. Containers help wrap an application in a controlled, stateless environment that can be relaunched and destroyed at any time (Lewis & Fowler, 2014).

2.1.1 Product / Data Pipeline Layer

The following concept was described in the previous research. The research suggested a plan that the pipeline architecture manages two layers, as it is visualised in *Figure 1.* 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 deploy the solution in different cloud providers. At the end of the pipeline, the pipeline manager code deploys the product in an independent production ready Kubernetes cluster (Debre, 2019).

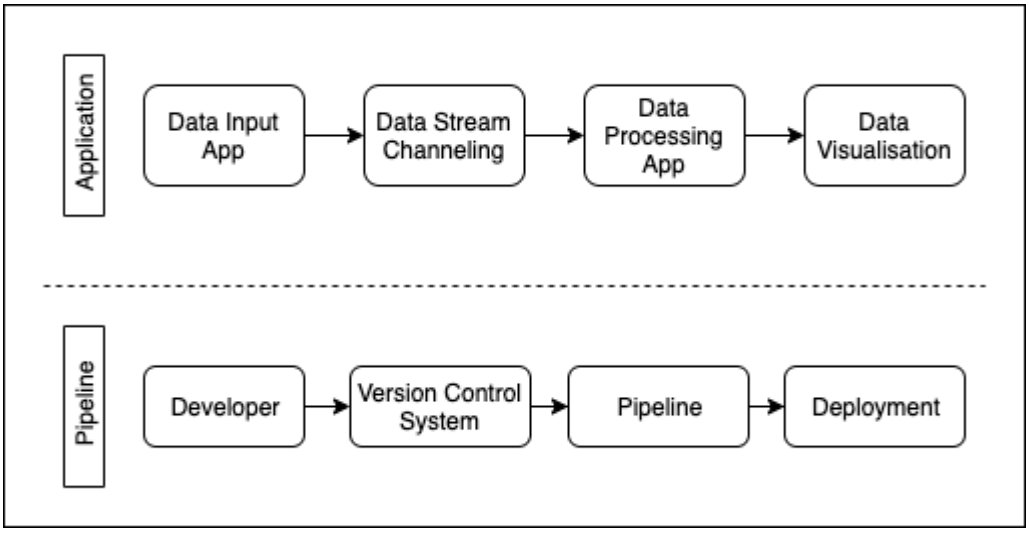


Figure 1. The architecture of the two layers

Figure 2 shows the four stages of a simple business case. The raw data enter in the data processing pipeline. It can be a manual entry on a website, a CSV[1] file upload, or a JSON[2] data stream provided via a standard web protocol.

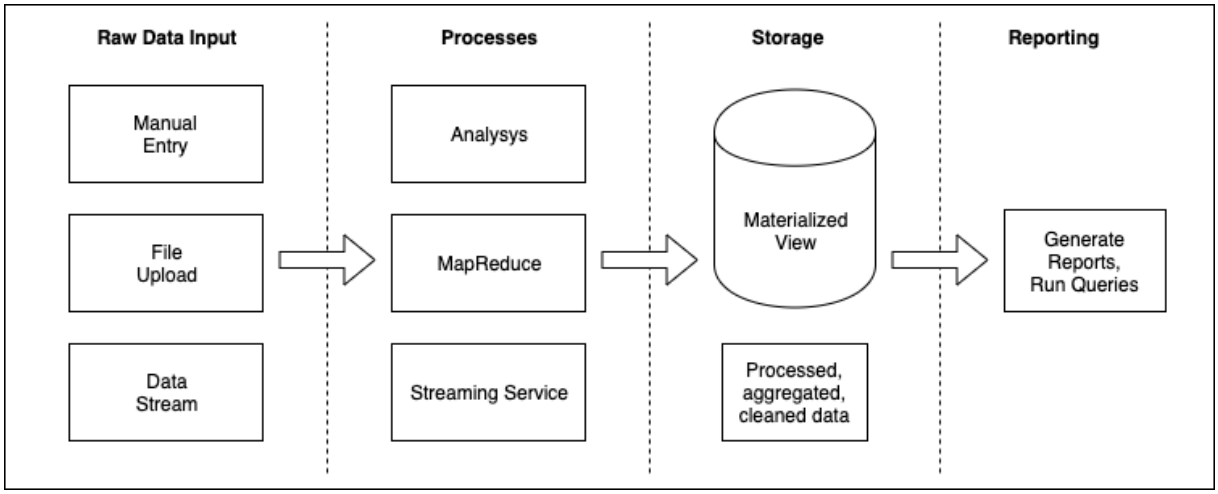


Figure 2. A standard data streaming pipeline

In the original proposal a more complex system was described using independent components and microservices. In which the first microservice can accept the data and stream, and other services can subscribe and use event streams. It was also discussed

[1] CSV: Comma Separated Values, the most common import and export format for spreadsheets and databases. Source: https://docs.python.org/3/library/csv.html
[2] JSON: JavaScript Object Notation is a lightweight data-interchange format. Source: https://www.json.org/json-en.html

that using a separated analysis platform for complex algorithm and an independent data visualization application. The following list is an example of a complex architecture:

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

2.1.2 DevOps Pipeline Layer

In our proposal we suggested a separate pipeline layer which orchestrates the testing and deployment process including version control system and pipeline runner.

It was discussed the benefits of DevOps pipelines and our motivation in our preliminary research. This idea and the importance of this research area was also supported by a recent publication from C. Nelson et. al. They expressed a personal experience in their "Building Machine Learning Pipelines" book. They wrote: *"most data science projects do not have the luxury of a large team including multiple data scientists and machine learning engineers to deploy models. This makes it difficult to build an entire pipeline in-house from scratch. It may mean that machine learning projects turn into one-off efforts where performance degrades after time, the data scientist spends much of their time fixing errors when the underlying data changes, or the model is not used widely."* They underline in the business case section that building pipeline helps data science teams develop new models faster and update existing models in a more efficient way. (Chapter 1., C. Nelson et. al. 2020).

The importance of DevOps practices is in a growing phase. "The Accelerate State of DevOps Report from DevOps Research and Assessment" (DORA) is considered as a comprehensive DevOps research in the industry. Companies from across the globe send their response to the researchers and the result is freely available. 31 thousands[3] responses were processed in 2019. Key findings are analysed and published by market leaders. As Google Cloud researchers summarized, one of the most important results is that 20% of responders categorized themselves as "High Performer" this year, which means that they invest significantly in DevOps practices. Previously this rate was only 7%. This independent analysis supports that DevOps is in high demand and it can be

[3] Source: Google Cloud - What is DevOps? Research and Solutions. https://cloud.google.com/devops/

useful in other parts of an organisation, including data science and data analysis (Forsgren, 2019; Wieldt, 2019).

Our goal is to simplify and accelerate data science projects with providing a pre-built project template. However finding the ideal combination of tools required more experiments. The second sub section (*Section 2.2*) sums up which tools we were experimenting and what our findings were.

2.2 Experimenting with alternative solutions

Driven by our motivation to build a project template and accelerate data science projects, I started to follow the footsteps of a traditional software developer's implementations, because building DevOps pipeline for data science project is similar. There are a number of DevOps pipeline tools that can support our "Data Science DevOps Pipeline" goal, however selecting the exact solution is based on our evaluation and experiments.

2.2.1 Finding the ideal source control management tool

Source control management is the first most important player in a modern development pipeline. It is the source of truth and the actual state of a software (Brockschmidt, 2016)

Any change in the source code triggers the pipeline process. The most popular source control management applications compete with each other with implementing more features to keep developers close to their platform.

Required criteria:

- manage git sources, branches, pull requests
- support code review
- support project management
- team and permission support
- built in CI/CD support

	GitHub	GitLab	Bitbucket
Website	https://github.com/features	https://about.gitlab.com/features/	https://bitbucket.org/product/pricing
Highlighted features	code review support project management tool integration with 3rd party tools team management documentation	code review support project management tool 3rd party tool integration (plugins) team management - deploy documentation	unlimited private repositories Jira software integration Trello integration CI/CD support: Bitbucket Pipeline (Only 50 minutes)

Table 1. Source Control Systems

In our use case, as we listed the required criteria above, the following features are important. *Table 2* shows their availability.

Features	GitHub	GitLab	Bitbucket
code review support	yes	yes	yes
project management tool	yes	yes	limited (Jira)
3rd party integration	yes	no, but Google Cloud with Kubernetes is supported	limited (Jira and Trello)
team management	yes	yes	
CI/CD pipeline	no	yes	yes (very limited)

Table 2. Comparison of Source Control System features

Our most important selection criteria was that how many elements of a modern DevOps pipeline is covered by the same tool. These elements should include source control management, pipeline orchestration, and cloud platform integration.

We selected GitLab, because the out of box CI/CD feature that could heavily simplify the implementation process and helps focus on building the pipeline instead of spending too much time to setup a custom CI/CD solution.

Additionally, GitLab provides the following required tools:

- traditional source control management,

- project management with opening issues,
- code review support,
- easy to use pipeline management integration directly from the source code
- integration with Google Cloud and automatic deployment with Kubernetes Engine.

2.2.2 Pipeline orchestrators

The pipeline orchestration tool is the heart of the CI/CD pipeline. This tool is responsible for executing the steps in the pipeline using shell scripts[4] or YAML[5] file, which sequentially are executed. Available commercial options are significant, however if we are looking for an open source and free solution the following three are popular: Jenkins, Concourse, and GitLab *(Table 3)*.

Jenkins is a well-known and widely used orchestrator. It is easy to extend with plugins, which helped Jenkins to be one of the most obvious choices for many DevOps teams, because its flexibility makes the integration smooth in complicated environments. It uses full featured programming language, Groovy, to configure and manage pipeline scripts. From one side, the programmable interface increases flexibility, but on the other side it needs a deeper learning curve and more practice. Jenkins is free to use, however it has to be installed and maintained by the user. There is no free to use cloud based solution to date.[6]

Concourse CI emphasized feature is simplicity. It focuses mainly on the pipeline and task management. Concourse can be managed by a YAML script. Because of its simplicity, learning Concourse CI is not as complex as Jenkins, but Concourse CI needs to be installed and self-hosted.[7]

GitLab became one of the full featured DevOps applications in the last three years. Their goal is to integrate and provide most of the features out of the box. GitLab CI/CD could manage containers, supports Kubernetes, and can be integrated with the source control

[4] Shell script is a file containing a series of commands. Source: http://linuxcommand.org/lc3_writing_shell_scripts.php
[5] YAML is a human friendly data serialization standard for all programming languages. Official website: https://yaml.org/
[6] Jenkins User Documentation: https://jenkins.io/doc/
[7] Concourse CI official website: https://concourse-ci.org/

management system. The documentation is detailed. The out of the box experience decreases the adaptation time. The maintainability cost is minimal. It provides free hosted, cloud based service.[8]

	Jenkins CI	Concourse CI	Gitlab CI/CD
Website	https://jenkins.io	https://concourse-ci.org	https://gitlab.com
Free	yes	yes	yes
Cloud Based	no	no	yes
Support Docker	yes	yes	yes
Learning curve	steep	medium	simple
Scripting	Groovy	YAML	YAML

Table 3. Comparison of Pipeline Orchestrators

This project uses GitLab CI/CD. It is free, cloud based and easy to use, and as it was mentioned earlier the source control management is also integrated.

2.2.3 Containers and clusters

Modern DevOps workflow uses containers. In our preliminary research, we have discussed why containers are critical and why they are the main building blocks of modern pipelines. They are immutable, re-creatable, and easy to deploy. The container unifies the running environment in each stage of the development. It is the same on the developer machine and in the production cluster (Wenzel et. al, 2019; Parizo, 2019).

For our project, we had to find ideal and easy to use solution for the following stages:

- running application on developer machine
- running test on developer machine
- running test on the pipeline
- building the application on the pipeline
- building containers in different stages
- hosting the built image

[8] GitLab Continuous Integration and Delivery documentation: https://about.gitlab.com/product/continuous-integration/

- deploying the built image in staging and production environment

CI/CD world is consolidating. We see in the IT industry, there is a consensus and most of the projects uses Docker containers and Kubernetes clusters. For this reason during our project we invested more time to learn how Kubernetes and Docker based containerization works. Understanding these technologies is important to be able to build a pipeline.

2.2.4 Docker

Building a development pipeline starts with setting up a fully featured development machine. It sounds obvious but in reality the installation of a smooth, easy to use development environment is not trivial.

Interviewing with data scientists and big data experts, we realized that they prefer macOS or Linux rather than Windows. Most of the solutions can be replicated on Windows machine, however this implementation focuses on macOS and Linux.

Setting up Docker on Mac starts on the official website. Docker.com provides a user interface based, easy-to-follow installer for users. After the installation, the following applications can be used

- Docker
- Docker Machines
- Kubernetes
- Kubectl

There are differences between Docker and Docker Machine. As explained in the official website Docker means Docker Engine, which run in the background as a daemon and we can communicate with it using “docker” command in command line.

Docker Machine works a little bit differently than the native Docker Engine. It uses VirtualBox to launch a virtual machine. In this virtual machine a simple Linux runs, which executes the Docker Engine. Docker Machine is considered as a deprecated option,

however it is still used in certain environments. For instance, when corporate proxy and corporate security rules force developers to use a virtual machine based Docker solution.[9]

The latest Docker for Mac and Docker for Windows add Kubernetes cluster support to the development machine, which makes development much easier. We do not need an expensive cloud based cluster to be able to develop a Kubernetes ready solution. So what is Kubernetes?

2.2.5 Learning Kubernetes

Our industry experts emphasized in our preliminary research that Kubernetes helps standardize Docker-based deployment processes. The main goal of using Kubernetes is to describe the combination of application containers with infrastructure of code, so that the state of our cloud based application is always the same until we change the state description (Debre, 2019).

The description of an actual state of our cloud based application is written in YAML manifest files. Building Kubernetes cluster means that we write manifest files.

The learning curve of Kubernetes is steep. We have to understand these manifest files and we learn how we write and change them. We have a repository with sample codes which was used for learning Kubernetes: https://github.com/zoltan-nz/playing-with-kubernetes

It can be concluded that learning Kubernetes is more and more important for software developers and DevOps expert. It is basically inevitable. Our learning journey started with the official documentation. The official tutorials helps understand the main concept and component behind Kubernetes.

For our project the following solutions are important:

- Namespace management
- Docker container deployment
- Launching services and exposing ports and endpoints
- Managing networking and subdomains with Ingress Controllers

[9] Source: Docker Machine Overview – What's the difference between Docker Engine and Docker Machine?

It was also important to find a solution to dynamically generate Kubernetes configuration. Kubernetes uses static manifest files. However we would like to manage multiple environment and reuse manifest files instead of maintaining more static files. The difference between these files are mainly the name of a namespace, the name of the application container, and the subdomain URL. Usually these names are programmatically determined in one of the steps of the pipeline, so we need a solution to keep content dynamically injectable.

There are Kubernetes template solutions to provide solution for dynamic Kubernetes environment. One of the most popular is the Helm Kubernetes template management framework. However it adds an extra layer of complexity to our work. I preferred to use a simple Linux bash based string interpolation. Our actual implementation uses "envsubs" command line tool to replace template tags with environment variable values.

Finding the optimal development environment for Kubernetes helped understand this new ecosystem of a cloud based deployment.

We touched the following tools as well. They are good alternatives or extensions, however we found that the main Kubernetes solution is sufficient.

Alternative Kubernetes solutions or extensions that I considered:

- OpenShift and MiniShift. A Kubernetes solution from RedHat.
- Kubernetes Platform using Minikube. An alternative option to install Kubernetes on local development machine.
- Kubernetes Platform using Docker For Mac. We use this tool on our local machine, however this is only for development and supports only a single machine cluster. The official Kubernetes implementation on Google Cloud supports multiple machine based cluster.
- Worth to mention: Nomad, Consul, Pulumi and JenkinsX. These tools uses Kubernetes and they add extra functionality.

2.2.6 Python

In the last couple of years the Python programming language is more popular in data science projects. As we can read in JetBrains Python developers survey, 58% of the participants mentioned that they use it for Data Analysis.[10]

In interviews in our previous research our industry experts and industry partners also clarified that the most of their data analysis projects used Python. We can also see this trend in "The State of Developer Ecosystem 2019" survey from JetBrains. "As Python is reported to be one of the best tools for data science, it's not surprising that the number of Python developers involved in Data analysis and Machine learning is so high." as it was concluded in their report (JetBrains, 2019).

For this reason our pipeline targets Python based applications and we built our solution using Python.

2.3 Summary

In this chapter we reviewed the previous research and determined the main building blocks for our design and implementation. As source control management tool and CI/CD pipeline orchestrator GitLab is the preferred tool. Docker and Kubernetes were chosen as main deployment target and container management tool. While Python was picked as the main supported programming language.

The next chapter describes the design and implementation of the framework and the template pipeline.

[10] Source: JetBrains Python Developers Survey 2018

3. Design and Implementation

This chapter focuses on our pipeline framework implementation. Furthermore we discuss why certain solutions were selected, and how we use them to improve developer experience and to accelerate data science project execution.

3.1 Design

In the final implementation the application layer is much simpler than what we proposed in our preliminary research which we have been referred to in *Section 2.1*. Our template and pipeline bootstrap support one python application out of the box.

Source control management and CI/CD pipeline are managed by the same tool. We use GitLab because it provides excellent source control management and pipeline orchestration support.

The final implementation is not cloud platform independent. We focused only on Google Cloud Kubernetes Engine migration at this stage, however thanks to Kubernetes our solution can be easily migrated to other cloud platforms, such as Amazon AWS or Microsoft Azure, that supports Kubernetes clusters.

The final implementation is not language agnostic. At the moment it supports Python projects, but it can be easily migrated to other frameworks and languages with changing the package management tools. The pipeline and Kubernetes related scripts are reusable.

All components are listed in *Figure 3*, which summarizes the main building blocks in three layers. These layers and implementation steps are described in *Section 3.2*.

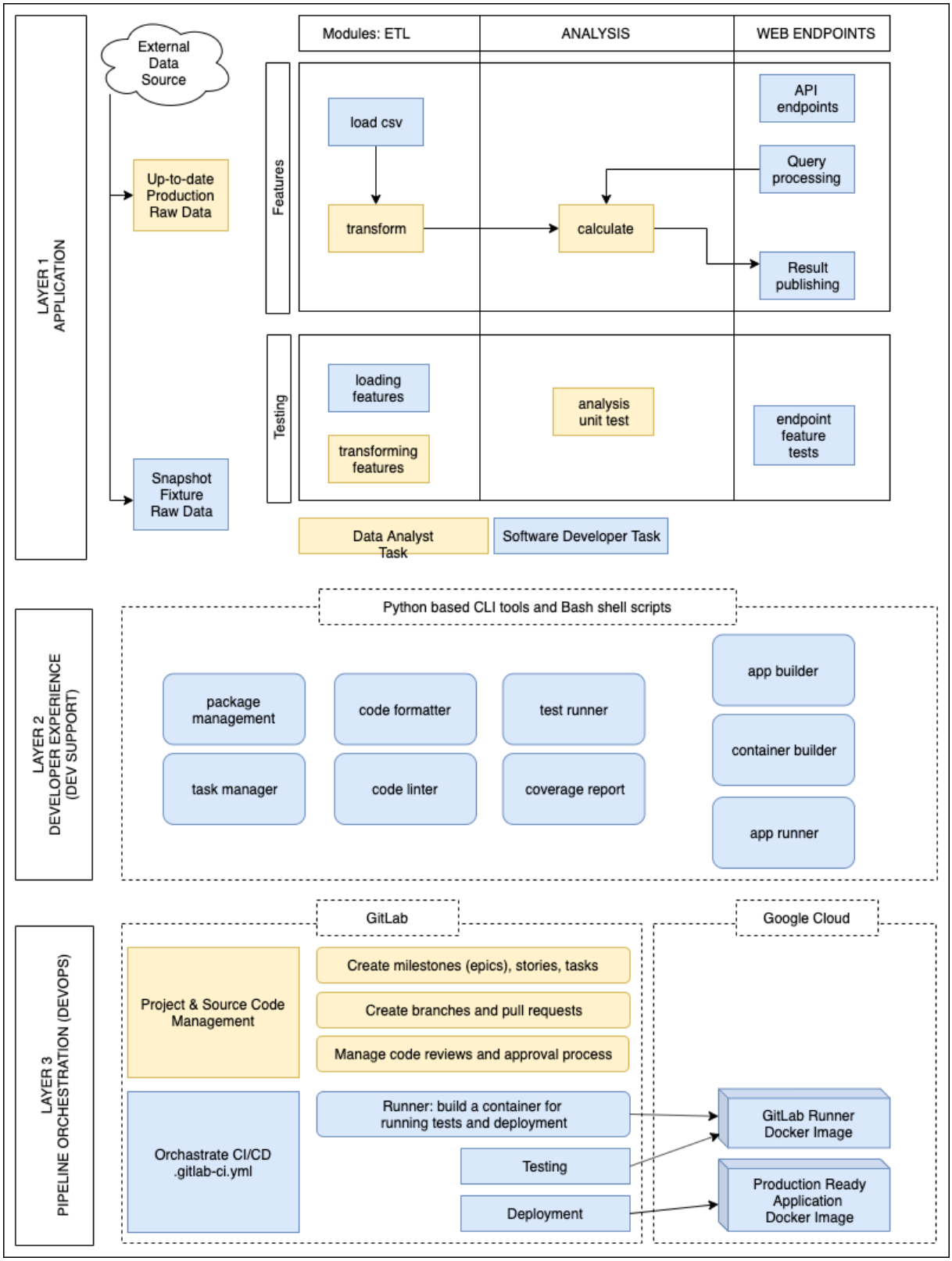


Figure 3. Overview of the final pipeline implementation

3.2 The framework implementation

The final product of our project is a GitHub / GitLab repository. It is defined as a "template" repository (on GitHub), so it can be used to bootstrap a new data science project. It is open source and released under MIT license.

We can find the project under the following GitHub or GitLab URLs *(Table 4)*.

Website	Repository
GitHub	https://github.com/zoltan-nz/ci-cd-pipeline-template-for-data-projects
GitLab	https://gitlab.com/zoltan-nz/ci-cd-pipeline-template-for-data-projects

Table 4. Public repositories of the pipeline template

We used GitLab for development, however we clone and share the project also on GitHub, because it has a bigger community, so it helps to reach more data scientist in the future.

3.2.1 Creating a project on GitLab

GitLab is an open source product. It can be used as a Service as a Platform (SaaP). Their free option is sufficient for smaller projects. Another opportunity is to install their software on our own hardware or cloud based hosting service. These two options are almost identical, and they provide the same service to users. In this project the official, cloud based GitLab was used, because it helps keep focus on the main goal, building the pipeline. we do not have to worry about maintaining the source control management service. Furthermore the official GitLab is accessible to the user testing candidates, so they can easily connect to our test environment.

It is also a good practice to start a project by creating an empty repository or cloning a template repository. For this reason we selected the template option on GitHub. (GitLab supports this feature in Premium subscription only.) Users can generate a new project using the template.

3.2.2 Package and task management

As it was mentioned earlier data science projects are mainly built in Python so it was an obvious choice that this project will utilize Python as well.

Modern Python supports package and dependency management. There are two popular tools:

- Poetry: https://poetry.eustace.io/
- Pipenv: https://pipenv-fork.readthedocs.io/en/latest/

Based on GitHub star popularity, we started using pipenv. However poetry and pipenv are almost identical. The main difference that pipenv uses Pipfile and poetry uses pyproject.toml as configuration file.

With a dependency and package management tool we can clearly setup what type of packages we would like to use in our project. The development and production packages are separated.

Package management tool installs all the necessary package and can help update dependencies.

Other important feature is to keep special task scripts. This preconfigured scripts can simplify the development and maintenance processes. Task management feature exists in other programming language ecosystem as well. For instance, in Node.js we use package.json to manage packages. Additionally in package.json has a "scripts" section, in which the own commands can be configured. This feature is commonly used in Node.js development. As we can see in our project and in Pipenv[11] and Poetry[12] documentation, the Python ecosystem is adopted it, so we can setup scripts in Pipfile or in pyproject.toml.

As it can be seen in *Figure 3* our framework's Layer 2 supports developer's tasks and increases the developer experience (DX). *Figure 4* shows Layer 2 in an excerpt from our main architecture diagram.

[11] Pipenv custom script shortcuts: https://pipenv.kennethreitz.org/en/latest/advanced/#custom-script-shortcuts
[12] Poetry documentation: https://poetry.eustace.io/docs/cli/#run

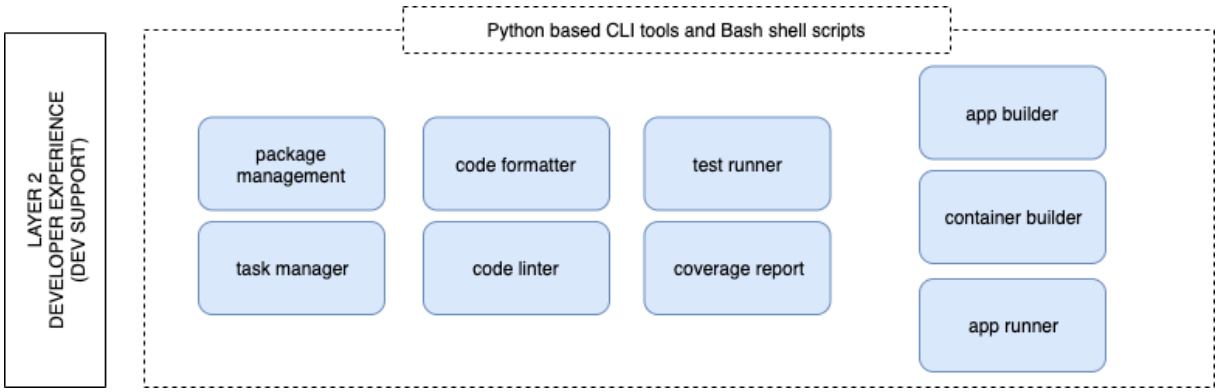


Figure 4. Supported developer tasks

The following code snippet represents how we implemented the above listed Developer Experience tasks in Pipenv format.[13] More detailed tutorial is in *Chapter 4*, so we can see in *Section 4.2* how these commands can be utilised in development. Detailed description of Pipenv tasks is listed in *Table 5.* in *Section 4.2* as well.

```
[scripts]
setup = “pipenv install --dev”
clean = “rm -rf .pytest_cache .tmontmp .coverage .testmondata htmlcov build dist
flaskr.egg-info”
lock-req = “sh -c ‘pipenv lock -r > requirements.txt’”
server-prod = “sh -c ‘FLASK_APP=my_hello_world_app FLASK_ENV=production
waitress-serve --call my_hello_world_app:start’”
server-watch = “sh -c ‘FLASK_APP=my_hello_world_app:start FLASK_ENV=development
pipenv run flask run’”
lint = “pylint --load-plugins pylint_flask my_hello_world_app tests”
lint-types = “mypy my_hello_world_app tests --strict”
format = “black .”
test = “pytest”
test-watch = “ptw -- --testmon”
cov = “coverage run -m pytest”
cov-report = “sh -c ‘pipenv run cov && coverage report’”
cov-html = “sh -c ‘pipenv run cov && coverage html && open htmlcov/index.html’”
build = ‘pip wheel -w dist .’
build-docker = “sh -c ‘pipenv run build && IMAGE_TAG=${IMAGE_TAG:-latest};
docker build --build-arg docker_image_tag=$IMAGE_TAG --tag my-hello-world-
app:$IMAGE_TAG .’”
deploy-kubernetes-local = “sh ./scripts/local-kubernetes-deployment.sh”
```

[13] Source: https://gitlab.com/zoltan-nz/ci-cd-pipeline-template-for-data-projects/blob/master/Pipfile#L31

3.2.3 Version management

One of the main reasons for using a package manager is to keep packages up-to-date. However not only packages have version in this project. The following dependencies can change and can be upgraded:

1. Python related versions:

    - Python language version
    - Pip, pipenv, poetry version
    - Application pip packages

2. App production Docker container related versions:

    - App's Docker container version

3. Pipeline runner related versions:

    - Docker engine version
    - Gitlab Runner version
    - Docker container components which is used inside Gitlab Runner
    - Kubernetes version
    - Google cloud management tool versions (kubectl, gcloud)

At the beginning of this project, our strategy was to use the latest version from most of the component. The risk of doing so was that some incompatibility and connections might not be tested properly. However, the latest versions have modern features and less security issues.

During the project we experienced a few times that keeping the version numbers flexible (less strict) caused incompatibility issues. For example when Python released the new version 3.8 my Docker configuration started to download the latest image and a few pip packages were not compatible with the new Python.

However, this incompatibility does not break the production, because we cannot deploy until we fix our issue. So the pipeline worked well and helped catch the issue in time.

The suggested way to manage versions to keep them fixed and upgrade them occasionally, especially when a necessary feature is released and we would like to use it, or when a security issue is fixed.

3.2.4 Why strict typing and type checking are important?

Python traditionally is a dynamic language. We have not had to declare types in our code. Dynamic languages are popular for scripting when the described problems are mainly procedurals and declarative. Learning a dynamic language is easier, because we do not have to adapt an extra layer of computer science, such as type theory (Harper R. 2016). Other popular dynamic language is Ruby or JavaScript.

However type system and typed languages have benefits. Advance interactive development environment (IDE), for example PyCharm from IntelliJ, provides more feature when the languages is typed. Type definition helps directly connect methods and variables to each other. IDE can refactor your code accurately. The IntelliSense auto completion provides exact method name matching. Writing code is faster and more accurate and we do not have to look up documentation, because our developer tool shows the available options.

As a result, the code quality increases. We declare clear contract with types. A method, a class or a data structure is described with type definition, so if we break this contract, that was not implemented correctly, the type checker will show the issues.

For this reason our project suggests the usage of Python strict typed options. Typing can be a new concept for data scientists. There are a few tools pre-installed in the framework to support strict typing.

The pypy package is a commonly used library for type checking.

3.2.5 Environment variables

Our project uses operation system's built in environment variables feature to manage custom values and data to keep our solution reusable and portable.

In Linux, macOS and Windows and also inside a Docker container which runs Linux as well, an environment variable can store a value which is accessible inside an application.

Environment variables give users the flexibility to dynamically inject values and customize behaviour. The environment variables can be changed via shell scripts or using configuration tools. For instance in our case GitLab uses this feature to pass sensitive data or customized changed values to the CI/CD pipeline.

3.3 Summary

In this Design and Implementation chapter we described the main building blocks of the pipeline template. The architecture diagram introduced all the three layers of the pipeline implementation, in which the top layer is the data science application itself, the second layer contains scripts and tools for supporting software development and testing, and the third layer is the pipeline, the cloud based environment, and deployment.

The next chapter provides a practical approach how to use the pipeline in a real data science project.

4. Using the pipeline template to bootstrap new project

The following chapter works as a tutorial. It describes how our framework can be used in any data analysis project.

4.1 How do you start a project?

This section is a step by step description about the template. It helps to understand how can we setup development environment and connect with cloud based services.

4.1.1 Environment setup

Prerequisites on developer machines:

- Python v3.8 on our developer machine
- Docker on our developer machine

Cloud based prerequisites:

- GitLab account and owner access to our project. (More details is provided in *Section 4.3.1*)
- Google Cloud access with Kubernetes support. (More details is provided in *Section 4.3.2*)

About installation of Python v3.8. One of the suggested ways to install Python on a developer machine is using the tool called "pyenv". On macOS, users can simply run the following commands if they already have "brew" installed.

```
$ brew install pyenv
$ pyenv install 3.8.0
$ pyenv global 3.8.0
```

- Brew website: https://brew.sh/
- Pyenv repository: https://github.com/pyenv/pyenv

The next step is to install "pipenv".

```
$ pip install pipenv
```

- Pipenv website: https://pipenv.kennethreitz.org/en/latest/

As detailed in the previous section, "pipenv" can alias custom scripts. I created one for environment setup.

```
$ pipenv run setup
```

4.1.2 Running the application

We can run the application in development mode and in production mode. We have two handy alias scripts for these tasks.

```
$ pipenv run server-watch
$ pipenv run server-prod
```

If the first script is run, the application will run in development mode on http://localhost:5000. In development mode we can see more debugging messages. In case of a code change, the modification will appear immediately, and the application will be restarted automatically.

The production mode is identical to the final production build, so it is suggested when we would like to check how our application will behave when we deploy. This mode uses different port, so the app can be open up in our browser on http://localhost:8080.

This practice uses an automatic linter and formatter during development, so the development team will deliver consistent code quality. The source code will be formatted in the same way.

The recent Python versions supports strict typing as well. Using types in our project improves code quality, the code editors can suggest accurate IntelliSense autocomplete. Additionally types can help when we would like to refactor our code, because code editors can find the connected segments in our application more accurately.

The alias script commands for linting, formatting and type linting are the followings.

```
$ pipenv run lint
$ pipenv run format
$ pipenv run lint-types
```

4.1.3 Replace the default application name with your real application name

The template implementation uses a default "hello world" name for the project. After the project was cloned, it is suggested to replace it with the real project application name.

Python naming convention expects snake case names (e.g. snake_case_name), however labels and names in Kubernetes has to be kebab case (e.g. kebab-case). Therefore, both string should be replaced.

In this way, we replace the default strings using code editor's "Find and Replace" feature. The following two strings have to be replaced in the project: "my_hello_world_app" and "my-hello-world-app". For example, if our real application name would be "analytics app", we should replace the first string to "analytics_app", the second string to "analytics-app".

We have to rename the "my_hello_world_app" folder name as well. Following the example above, the folder should be renamed to "analytics_app".

Our search will find references in our Kubernetes configuration files, they also have to be renamed.

4.2 Unit testing

Testing is an important part of any software development project. Our framework supports it out of the box. We adopted the "pytest" library and implemented in our codebase. Users can run test only once or in watch mode, which will rerun the test when files change detected. Our implementation supports code coverage report as well. The following commands help keep the code reliable and bug free.

```
$ pipenv run test
$ pipenv run test-watch
$ pipenv run cov-html
```

The summary of available "pipenv run" scripts *(Table 5).*

Pipenv task	Description
pipenv run setup	Setup local pip environment. Run it first.
pipenv run clean	Clean up temp files.
pipenv run lock-req	Update requirements.txt. Run it if you added or upgraded some packages in Pipfile.
pipenv run server-prod	Run your Flask app in production.
pipenv run server-watch	Run your Flask app in dev mode with watch task.
pipenv run lint	Run pylint.
pipenv run lint-types	Run mypy type checking linter.
pipenv run format	Run the black code formatter.
pipenv run test	Run pytest.
pipenv run test-watch	Run pytest in watch mode.
pipenv run cov	Run code coverage report.
pipenv run cov-html	Generate a code coverage report in html format.
pipenv run build	Build python binary version.
pipenv run build-docker	Run a local Docker image.
pipenv run deploy-kubernetes-local	Run ./scripts/local-kubernetes-deployment.sh script to deploy in your local Kubernetes cluster.

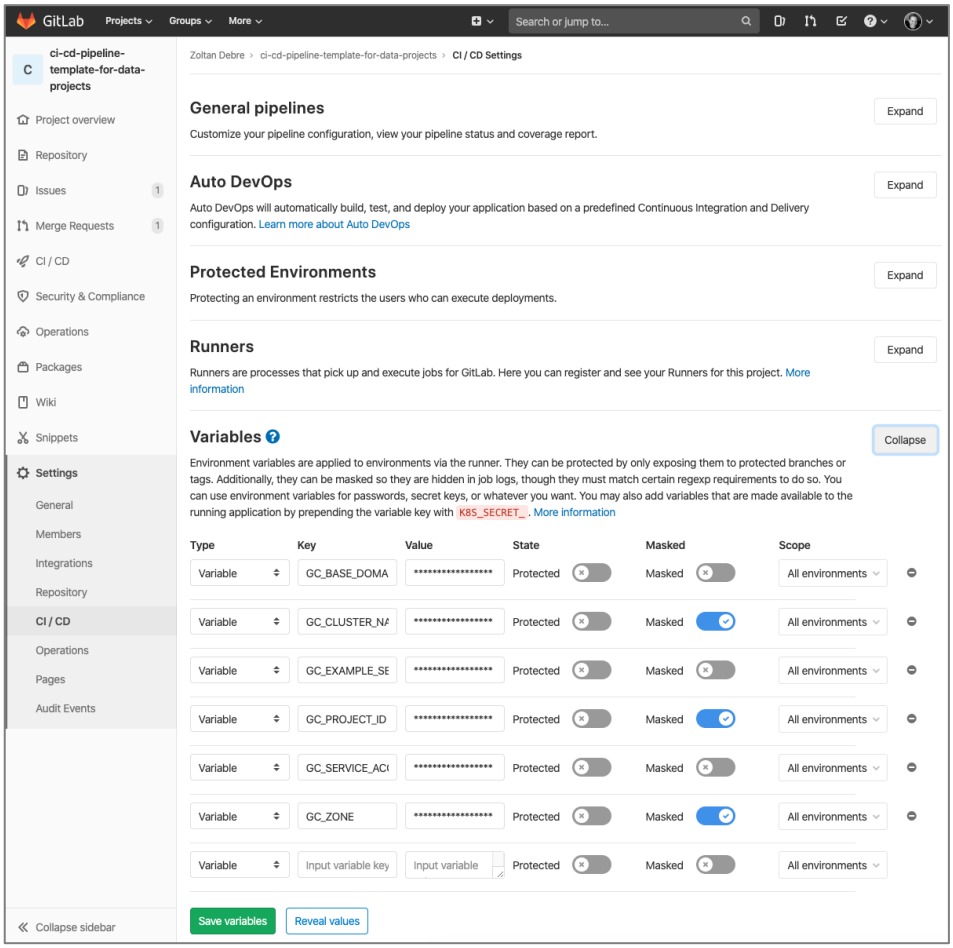
Table 5. Pipenv run scripts

4.3 Using the pipeline

This section shares more details about the pipeline and how can it be used in any project.

4.3.1 Creating a GitLab project

A project cannot work without a source control management system. As it was previously suggested, users host the source code and run the CI/CD on GitLab.



Screenshot 1. Environment variable setup on GitLab

GitLab account is a prerequisite. However we have to setup a few details in GitLab platform, especially our environment variables for CI/CD which connects GitLab to the Kubernetes cluster.

You can find this "Variables" menu point under the Settings → CI/CD menu *(Screenshot 1)*. The environment variable values can be listed there and they will be injected into the CI/CD runner, so the build tool can access customized or sensitive information.

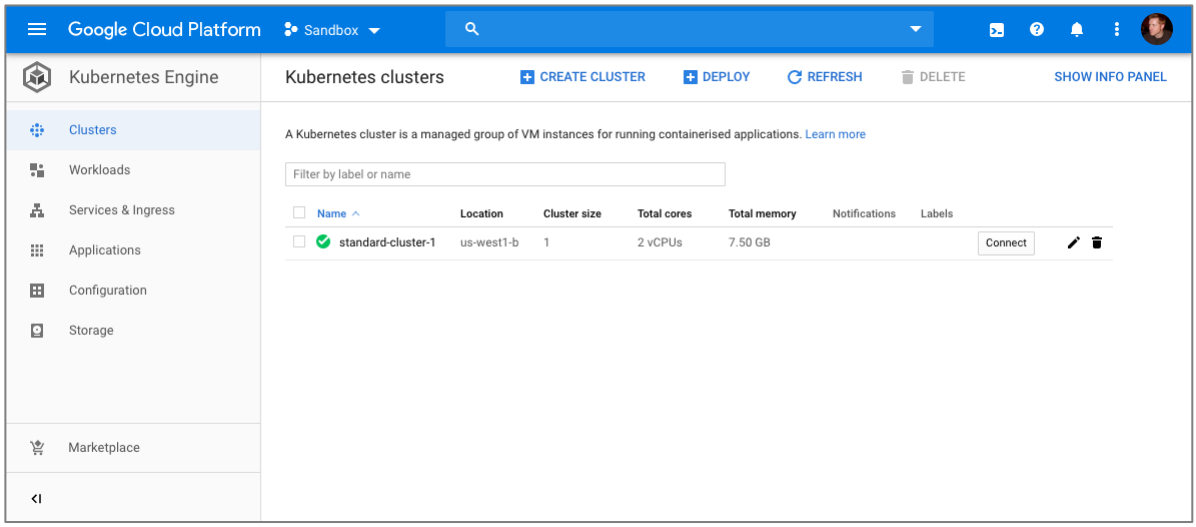
Most of the settings are used to configure Google Cloud Kubernetes Engine. A rule of thumb is that sensitive information must not be committed in the source code. However, for example if we need an access key or a password in our codebase and in our pipeline runner, they should be passed forward using environment variables. We discuss this in detail in the next section.

To be able to setup Google Cloud and Google Cloud Kubernetes Engine variables, users need to have Google Cloud access, and at least one Google Cloud Kubernetes cluster needs to be created.

4.3.2 Google Cloud and Kubernetes Engine

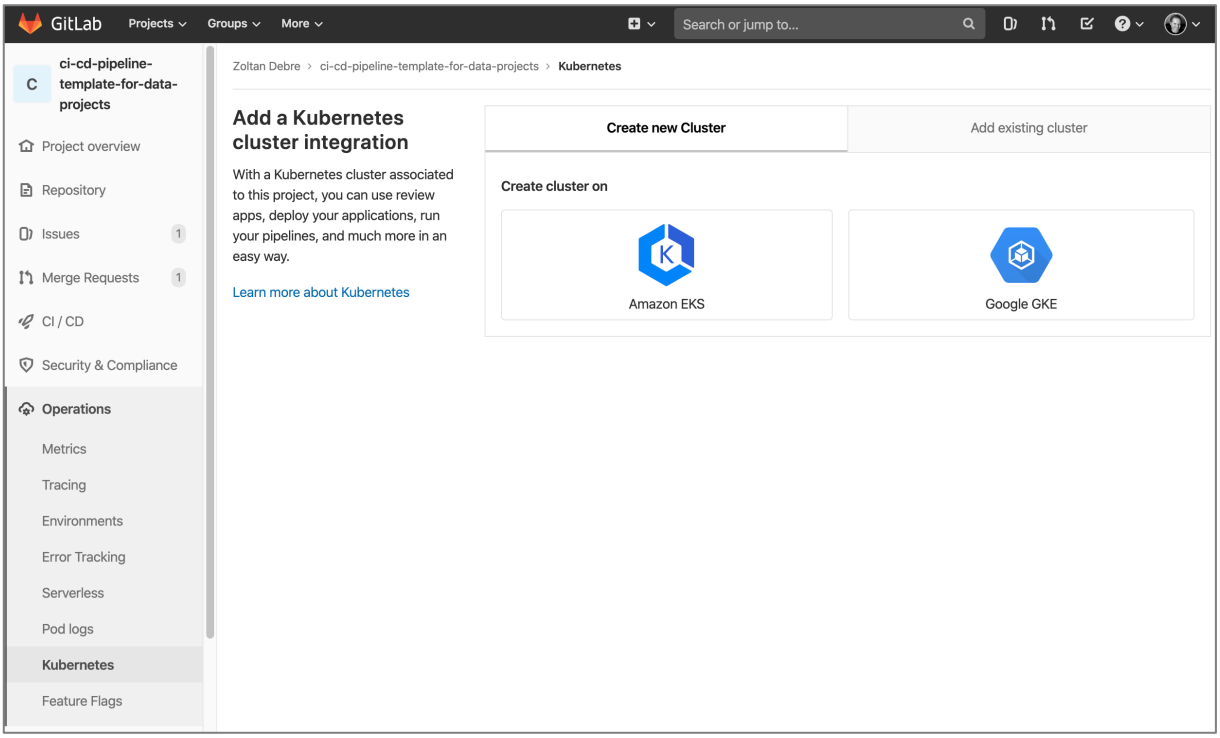
This project supports Google Cloud (GC) out of the box. It means, we have a few lines of code which helps to connect Google Cloud and Google Cloud Kubernetes Engine (GKE). However, this part of the infrastructure of code can be replaced with other commands to support alternative cloud based providers. Furthermore these templates can support Amazon Web Services if it is desired.

We use Kubernetes Engine feature from Google Cloud platform. A new Kubernetes cluster can be created with one click on the Kubernetes Engine page *(Screenshot 2)*.



Screenshot 2. Kubernetes Engine home page

When the cluster is up and running we can connect it to our GitLab project using the dedicated GitLab setup page. When this project was started GitLab supported only Google Cloud. Since the start of the project, GitLab added Amazon AWS Kubernetes (EKS) support as well *(Screenshot 3)*.



Screenshot 3. GitLab "Add Kubernetes" home page

4.3.3 Pipeline orchestration

Pipeline orchestration is represents the collection of automated tasks which are executed during CI/CD process.

CI/CD tasks are visualized in Layer 3 in the main architecture graph *(Figure 5)*.

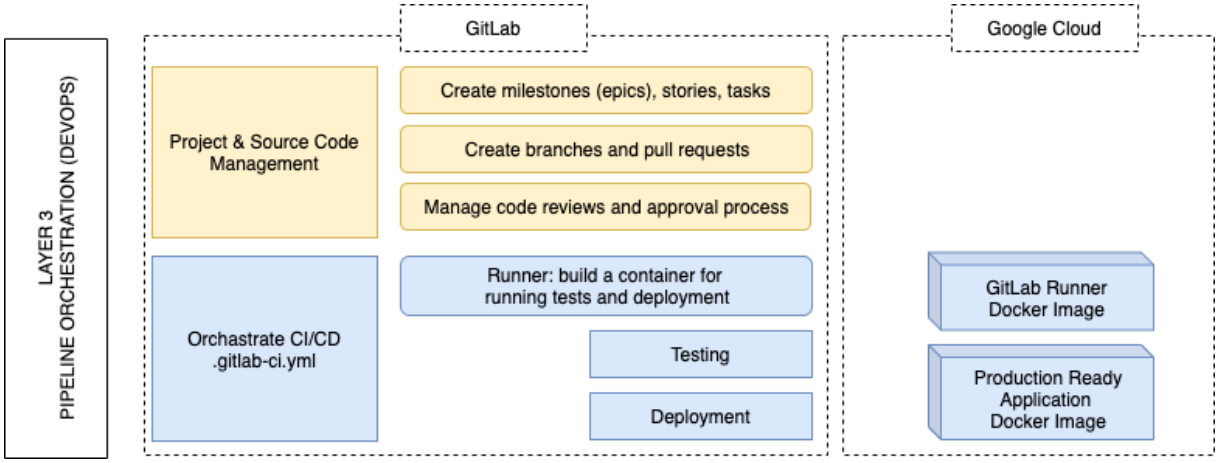


Figure 5. DevOps Layer

GitLab help manage the source code and the overall project with milestones, stories, and tasks. Pull requests and code reviews are connected to tasks, so activities and code improvements can be documented, and it keeps track record about feature implementation.

GitLab pipeline orchestration code is managed in a special file. It is called ".gitlab-ci.yml".

This YAML file controls each step of the pipeline from running tests, building Docker images and deploying to Kubernetes.

Main pipeline stages *(Figure 6):*

- **Test stage** is used to run unit tests, linters, and format checking.
- **Build stage** is responsible for creating production ready Docker image and upload to a Docker Registry.
- **Review stage** is a powerful feature. It is used only in development branch, and it deploys a preview version of the application under a temporary subdomain on Kubernetes Cluster. Our framework uses the branch name as a subdomain.
- **Staging phase** runs only in master branch, and it is the same as in production, but uses a special subdomain. In this stage the pipeline deploys the production ready Docker container. As an end user we can check that our application works as intended.
- **Production stage** can be triggered manually and it elevates and deploys the image from staging state to the final production domain.

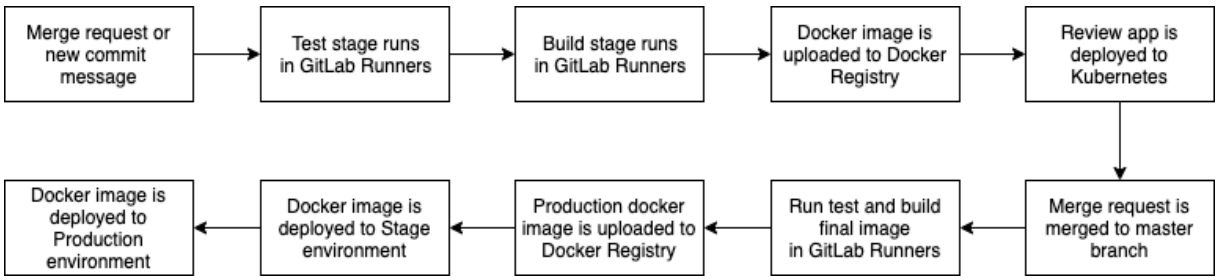


Figure 6: GitLab CI stages

4.3.4 Developing and debugging the pipeline

The challenge is how we could experiment with our Kubernetes scripts without using cloud based platform and replicate the cloud based environment in our local machine.

We need a local Kubernetes cluster. The easiest way is to use the Docker for Mac's Kubernetes option, so the local machine can run a single node Kubernetes Cluster by ticking a checkbox. We already discussed this in our experimental section.

The pipeline runs a sequence of commands in each step. These commands can be added to a shell script which can be part of our repository in test or utility folder. Good practice to keep all of these unique tasks in one place. In the Python based demo project, it was the local package and task manager tool, the pipenv configuration file, Pipfile., as we discussed earlier.

GitLab Runner is the core of the GitLab CI/CD implementation. It is triggered when the source code is changed and the repository contains a ".gitlab-ci.yml" file.

Experimenting with a pipeline and its steps could be slow when developers commit each changes and wait for the cloud based service to execute each step. GitLab provides a local version from the GitLab Runner. It can be installed on the developer machine.

With the local version users can run the pipeline code and debug without waiting for cloud based services.

There is a sample bash script in our project under "scripts" folder. It can be used to run GitLab Runner locally and test the whole pipeline with our local Kubernetes environment.

For instance the following script runs the local GitLab Runner with our unique setup. Environment variables helps to keep our solution dynamic and customisable. We can use our local values for deployment. The cloud based live version will use environment variables from GitLab settings as described above.[14]

```
#!/usr/bin/env bash

gitlab-runner exec docker\
  --docker-volumes /var/run/docker.sock:/var/run/docker.sock\
  --docker-privileged\
  --env GC_PROJECT_ID="$(<./secrets/gc-project-id.txt)"\
  --env GC_EXAMPLE_SECRET="$(<./secrets/example-secret.json)"\
  --env GC_SERVICE_ACCOUNT_KEY="$(<./secrets/gc-service-account-key.json)"\
  --env GC_CLUSTER_NAME="standard-cluster-1"\
  --env GC_ZONE="us-west1-b"\
  --env GC_BASE_DOMAIN="$(<./secrets/gc-base-domain.txt)"\
  test
```

4.3.5 Dynamic Kubernetes manifest files

Key element of the pipeline is the Kubernetes manifest file management. These manifest files represent an immutable state of the Kubernetes cluster, mainly which Docker containers should run and which subdomain should point to that container.

Because Kubernetes manifest files describe a state, for this reason they originally static files. However, we have to use them with different values in each step of our pipeline. These values are for example the namespaces inside the cluster, which isolate a deployment, or the dynamic URL what we use for reviewing the app.

We use the popular Linux shell command, "envsubs" to replace placeholders with environment variable values. This way the manifest files are reusable and maintainable.

The following variables are used in our Kubernetes configuration *(Table 6)*.

[14] Source: https://gitlab.com/zoltan-nz/ci-cd-pipeline-template-for-data-projects/blob/master/scripts/gitlab-runner-test.sh

Environment Variable	Description
KUBE_NAMESPACE	Kubernetes namespace. It changes based on the deployment stage. E.g. my-app-review, my-app-staging, my-app-production.
KUBE_DEPLOYMENT_NAME	Pod deployment name. E.g. my-app-deployment
KUBE_APP_NAME	The app name used as label mainly. E.g. my-app
KUBE_IMAGE_NAME	Important! The Docker image name with the registry URL: gcr.io/my-project-id/my-app:hash
KUBE_CONTAINER_PORT	Depend on your application. Check Dockerfile's EXPOSE value. E.g. 8080
KUBE_SERVICE_NAME	Load balancer service name. E.g. my-app-preview-load-balancer-service.
KUBE_SERVICE_EXTERNAL_PORT	Service exposes the app container on this port. It can be used to connect directly to the app or in an Ingress controller. E.g. 9090
KUBE_INGRESS_NAME	Ingress controller creates subdomains. This is just the name of the Controller. E.g. my-app-router.
KUBE_PUBLIC_APP_DOMAIN	The public domain address. It can be the production domain or a subdomain. E.g. example.com, staging.example.com, some-review-branch.1.2.3.4.nip.io
KUBE_IMAGE_PULL_POLICY	Use "Always" in production, Docker for Mac Kubernetes needs "Never" value for locally built images.
KUBE_GC_BUCKET_KEY	A base64 encoded json string which will be written using gc-bucket-key.json filename in a Secret Volume and attached to /home/app/secrets/

Table 6. Environment variables for Kubernetes customisation

4.3.6 Manifest templates

Our Kubernetes manifest templates are placed in "kubernetes" folder in our project *(Table 7)*. These YAML files describe the desired deployment as infrastructure of code. GitLab CI process use these files to apply states to Kubernetes cluster using "kubectl" command.

Template file	Role
namespace.yaml	Setup the namespace of a stage. E.g. appname-preview, appname-production, appname-staging. Namespaces create a virtual cluster inside the same physical cluster, so it is ideal to separate deployments from one another.

deployment.yaml	This manifest is responsible to deploy the application's Docker container in a pod which will run the application. It exposes a port number, and a load balancer can direct traffic to that pod and port.
service.yaml	Services are responsible for managing load balancers and direct traffic to a pod.
ingress.yaml	Ingress controllers help configure unique subdomains for production, staging, and review environments. Stage and production configuration will be deployed once and they usually do not change. Subdomains of the review applications will change more frequently, because they dynamically generated from a new pull request based on the source control commit branch name.
secret.yaml	Kubernetes provides a secret management layer. Secrets can be stored in a key value tuple or in a separate volume. Our project uses this manifest to inject sensitive data (access keys) to the production ready python application.

Table 7. Kubernetes manifest template files and their roles

For demonstrating how our project uses the above manifest template files and combine previously listed environment variables, the following code snippet is extracted below. It is from ".gitlab-ci.yml" and responsible for the review application deployment.[15]

```
review_deploy:
  <<: *auth
  variables:
    KUBE_NAMESPACE: my-hello-world-app-preview
  stage: review
  script:
    - envsubst < ./kubernetes/namespace.yaml | kubectl apply -f -
    - envsubst < ./kubernetes/secret.yaml | kubectl apply -f -
    - envsubst < ./kubernetes/deployment.yaml | kubectl apply -f -
    - envsubst < ./kubernetes/service.yaml | kubectl apply -f -
    - envsubst < ./kubernetes/ingress.yaml | kubectl apply -f -
    - echo "Review url:" $KUBE_PUBLIC_APP_DOMAIN
  allow_failure: true
  environment:
    name: review/$CI_BUILD_REF_SLUG
    url: http://$CI_BUILD_REF_SLUG.$GC_BASE_DOMAIN
    on_stop: review_clean
  only:
    - branches
  except:
    - master
```

Kubernetes configurations are deployed by calling "envsubst" and "kubectl apply" in the "script" section.

[15] Source: https://gitlab.com/zoltan-nz/ci-cd-pipeline-template-for-data-projects/blob/master/.gitlab-ci.yml#L128

4.3.7 Inject Docker image tag in the app

It is a useful practice for debugging purposes to inject the actual application version inside the running python application. We use the latest git commit short hash string as a version number.

The Docker image tag is injected in the Flask application. We use an ARG value in Dockerfile. It is mapped to an environment variable, so the Flask app can access it using "os.getenv".

There is a custom "/version" route is implemented in the python (Flask) application to print out the injected tag, so if we would like to know which version is running, we can just navigate to the version route and the actual version number (commit hash) will be printed out.

4.3.8 Managing secrets using Kubernetes

Accessing resources on Kubernetes need service access key, which is usually stored in a simple JSON file. Additionally python applications can use Google Cloud SDK library to connect to Google Cloud services. The default behaviour of this library is to read a text file from a local storage area. The path of the file is stored in an environment variable. We needed to inject access key inside our python (Flask) application, because it was used to connect other Google Cloud bucket service to download raw data files.

We had two challenges in this case. Firstly, the application should be able to access this environment variable and read the string of the path. This task managed automatically by Google provided SDK. Secondly, the file should be injected in the application filesystem, inside the container. This can be solved by using Docker Volume. Kubernetes also support a special Secret management for sensitive volumes.[16]

Our final framework and template contains a sample implementation, so this approach can be used in other projects.

[16] Source: https://kubernetes.io/docs/concepts/configuration/secret/

We use the following approach to inject secretes in the app:

1. Add secrets in JSON format to "./secrets" folder.
2. Use gitignore configuration to avoid committing these files in your source code.
3. Update "Dockerfile" and add a volume where Kubernetes can attach a cluster based secret volume.
4. Use "kubernetes/deployment.yaml" and "kubernetes/secret.yaml" to configure volume injection.

We realised there are differences how base64 encoding was managed between Linux and macOS. "base64" is a commonly used command line tool to convert file content to a string. However this tool inserts a new line after each 76 characters which brakes an encoded JSON file on Linux meanwhile on macOS there is not any new line character. For this reason it is important to use "base64 -w 0" option to disable line wrapping and get the same result in each system.

The "Dockerfile" is listed below. It creates a "/home/app/secrets" folder in the application container and open it as a Volume, so it allows us to attach external storage to the container.[17]

```
FROM python:3.8.0-slim

RUN pip install --upgrade pip

RUN mkdir /home/app
RUN addgroup --gid 1024 app-group
RUN useradd --uid 1024 --gid 1024 --non-unique --comment "" --home-dir /home/app
app
RUN chown 1024:1024 /home/app
RUN chmod 775 /home/app
RUN chmod g+s /home/app
WORKDIR /home/app
USER app

RUN mkdir /home/app/secrets
RUN chown 1024:1024 /home/app/secrets
VOLUME /home/app/secrets


ARG docker_image_tag='pass this value when building the image with docker build
--build-arg docker_image_tag=demo'

ENV FLASK_APP=my_hello_world_app
```

[17] Source: https://gitlab.com/zoltan-nz/ci-cd-pipeline-template-for-data-projects/blob/master/Dockerfile

```
ENV FLASK_ENV=production
ENV DOCKER_IMAGE_TAG=$docker_image_tag

ENV PATH=/home/app/.local/bin:${PATH}

COPY requirements.txt ./requirements.txt
RUN pip install --user -r requirements.txt

COPY ./dist ./dist
RUN ln -s /home/app/secret /home/app/.local/lib/python3.8/site-packages/secrets
RUN pip install --user ./dist/*

EXPOSE 8080

CMD [“waitress-serve”, “--call”, “my_hello_world_app:start”]
```

Using the “kubernetes/secret.yml” manifest file, we can create a secret object inside the Kubernetes cluster from our environment variable, which is injected during the build process.[18]

```
apiVersion: v1
kind: Secret
metadata:
  name: gc-bucket-key
  namespace: ${KUBE_NAMESPACE}
type: Opaque
data:
  gc-bucket-key.json: ${KUBE_GC_BUCKET_KEY}
```

Here is a snippet from our “kubernetes/deployment.yml”, in which we mount our previously created secret to a path of our Docker container, so our application can read it inside the container.[19]

```
        volumeMounts:
           - name: gc-bucket-key
             mountPath: “/home/app/secrets/”
             readOnly: true
```

[18] Source: https://gitlab.com/zoltan-nz/ci-cd-pipeline-template-for-data-projects/blob/master/kubernetes/secret.yaml
[19] Source: https://gitlab.com/zoltan-nz/ci-cd-pipeline-template-for-data-projects/blob/master/kubernetes/deployment.yaml#L28

4.4 Summary

This chapter provides a step-by-step tutorial introducing the pipeline, while describing in technical way how a user can utilize the pipeline template.

The next chapter will focus on evaluation of the usability of the pipeline and how our test users helped continuously evolve the product.

5. Validating our framework

This chapter focuses on the evaluation of the pipeline. Firstly we can read about our methodology, followed by details about case studies, and finally the empirical evaluation.

5.1 Methodology

5.1.1 Overview

The most important aspect of the validation was to use our framework in industrial environment and support production targeted data science application deployment.

Our preliminary research helped determine real business requirements. The focus was to create a pipeline which can be used in production. Additionally, a paper from Per Runeson and Maring Höst with the title of "Guidelines for conducting and reporting case study research in software engineering" gave guidance to evaluate the product (Runeson et al, 2008). They referred to Lethbridge et al. (2005) where data collection techniques are listed in three levels. The first degree data collection is when the researcher is in contact with users, and collects data directly, for instance interviews or focus groups. The second degree data collection is used for indirect methods where the researcher is not in direct contact with the user, for example using a questionnaire. The third degree data represents data analysis, where we use already collected data.

We learned about the industry and DevOps practices by using first degree data collection methods such as interviews in our preliminary research. We used questionnaires and user tests in our actual evaluation process, which can be categorized as second degree data collection. We also used third degree data collection in the form of user test. However our user test was a type of observation. In our case study we observed the activities of the users. The usage of our framework is recorded by GitLab source control system, which can be considered as a telemetric or activity log. In the evaluation, I used a mix of second and third degree data collection. As Runeson et al. (2008) mentioned in their guide, observation is a useful practice to investigate software engineers, how they use a tool. In our case this observation, user test monitored by source control system. We also use this data to calculate code related metrics.

We cooperated with an industry partner and practitioner data scientists who used our pipeline in real-world application. These case studies helped us to evaluate and observe how data scientists interact with modern software development pipelines. The observation of their activities and their constant feedback looped back to our pipeline implementation, so it helped evolve our solution further.

5.1.2 Ethics approval

Ethics committee application and approval supported us to cooperate with our industry partners and individuals.

Following documents are provided and approved by the committee:

- Individual User Test Information Sheet
- Information Sheet for Organisations
- Questionnaire
- Consent Form For Individuals
- Consent Form For Organisations

The information sheet helped participants to learn about the goal and the context of this research and the steps of the user test.

The important part of the evaluation process was to use our pipeline for real world application. For this reason participants granted access to our pipeline implementation and the source control system. They were encouraged to use the source control system, create commits, write comments, open pull requests, and use the pipeline. Their activity, commit messages, their code shared with the researcher and with the supervisors and the activity is reported anonymously and summed up in summary form below.

We also invited users to a follow-up interview session where we asked them about their overall experience.

The Participant Consent Form was also provided for participants. They agreed that we can observe them and they gave consent to use the collected data.

An interview guide have also been attached to the ethics approval process. This guide listed the following questions and instructions.

1. Please describe your role at your company.
   - The purpose of this question to know more about the interviewee and getting some context about his/her responsibilities.
   - I am looking for to know more about what type of role and what type of skills you need to work as a data scientist and data engineer at a commercial organisation.

2. Please talk about what type of software development practices do you follow when you build a data analysis application.
   - The purpose of this question to learn about how a data engineer or data scientists build a web application.
   - I would like to know more about what type of programming languages they use, how they use source control systems, what is they preferred developer IDE.

3. Do you write any test when you implement a feature in your application?
   - The main reason for asking this question to learn more about how they test their code, do they test at all, have they heard about software developer test practices?

The following questions will be raised after showing them the developer pipeline. They have the opportunity to use the pipeline in practice. Participants will be invited to a private GitLab project. They can connect and use the source control system, also contribute to a sample project with adding data analysis features.

1. Please talk about your experience using the pipeline. What was difficult and what was easy to use? How would you improve it?
   - The purpose of this question to get feedback about our approach and find areas where we can improve our framework.
   - Answers can help us to determine which code and solutions should be part of a project template and which can be an empty, updatable section where data scientists can fill the gaps with their implementation.

2. Please show us how have you used the source control system, created pull requests, added tests and commented or reviewed code changes? Which part was easy and where would you like to learn more about best practices?
   - With this question we can learn more about how they actually used the pipeline and which area where we should add more information.

3. After using a developer focused pipeline, would you use in your future project? What kind of features should be part of a pipeline to provide more value for your work?
   - This is a more open question to learn more about their real needs and help us to be more focused in our research.

5.2 Case studies

Our pipeline was evaluated by our industry partner using for production targeted real-world application. The use case was a standard data engineering task, such as they received a daily-updated raw data in CSV files which was processed and analysed. As a result they wanted a microservice with an http endpoint, which listed the processed and summarized data.

Previously they often developed a simple python script which were tested manually and solved a certain problem instead of using a full featured solution which can evolve and is reusable.

As Jakob Nielsen explained in his research that a single user test provides a third of insights, the second user adds some amount of insights and the third user adds a small amount of new data. He stated adding more user adds less and less to the usability test (Nielsen, 2000).

We observed three users who directly used our pipeline, we also contacted and interviewed two other data scientists who were also interested, although they would like to use our pipeline in a later project, however their feedback was also valuable.

5.2.1 User feedback

The following list highlights the most important responses and suggestions from our participants.

They mainly used the following categories to describe their profession:

- database developer,
- data engineer,
- data analyst,
- data scientist.

In terms of software developer practices, they talked about the following topics:

- They know about agile methodologies and they usually use them in their daily work.
- Pair programming was also mentioned as a practice what they use for learning new skills and solutions.
- They usually build a functional implementation, then add more functionality later.
- They occasionally add tests to their code, but not as common practice than in general software development.

When we asked them about pipeline implementations and techniques, we got the following responses:

- They stated that dealing with git, repository, source control is less problematic.
- Difficult and more challenging with the following technologies: Docker, CI/CD orchestration, using Kubernetes, using Google Cloud, using secure data and managing secrets in an application.
- Our users found the project management support in our CI/CD implementation useful, including creating, documenting issues and tracking changes in a dedicated branches.
- They expressed that it would be definitely useful to provide a generic project setup, where Python, Docker, CI/CD, Google Cloud and Kubernetes are

already pre-configured and ready to run, so they can just focus on their data science implementation.

After using our pipeline we got the following responses and suggestions:

- Positive experience, that our solutions supports local development environment. So they can improve their code on their personal notebook, run most of the checking tool (testing, linter, formatting) and later they can push changes to trigger remote pipeline.
- Predefined and opinionated template helped them to bootstrap a new project. They did not have to invest time finding compatible combination of tools, because our solution worked out of the box.
- The automatic deployment helped them to see their changes in real production environment. It was especially useful, because they saved loads of time between iterations and they did not worry about production deployment.
- They also suggested that more visual aid, graphs and diagrams how the pipeline works could help them to understand about processes. Visual presentation help to see the connected components more clearly.
- Our users found useful that we suggested them to implement small changes and push and review only simple features in a pull request. It was much easier to review and forced them to simplify and to break into smaller parts of their implementation.
- In terms of source control management, they had issue sometimes with merge conflicts. It would be great to learn more about how they can avoid it, or minimize this issue.
- We got positive responses about Dockerfile and Kubernetes configuration, because they do not really would like to change it, being a template based solution is reusable and flexible to adopt it in other projects as well.

5.2.2 Reported issues and suggestions from user tests

During our user tests the following issues reported or questions raised. Fixing them helped us to improve further our solution and provide a more stable final product.

5.2.2.1 Rolling Policy

There was an issue when a user deployed a review application with the pipeline, the temporarily generated review website randomly switched between a previous and a latest implementation.

After our investigation, we realized that Kubernetes automatically keeps previous Docker images, and the load balancer service just automatically switch between the deployed containers.

Kubernetes has a deployment strategy option which can be configured in Kubernetes manifest file. After changing the Rolling Policy in our configuration, this issue never came back again.

The following fix was implemented in Deployment script.[20]

```
spec:
  replicas: 1
  strategy:
    type: RollingUpdate
```

5.2.2.2 Administrator access

We got an important question about how can our users' production deployment administrator access the pipeline built, production ready Docker image. Their use case was to pull this artefact in their protected in-house environment, where they can run in their production cluster.

Our implementation can work with any standard container registry, however we used Google Container Registry (gcr.io) in our project.

One of the outputs of our pipeline is the production ready web application. It can be used directly, however these artefacts can be imported to other clusters or environments. In this case, the artefact was the production ready Docker image.

[20] Source: https://gitlab.com/zoltan-nz/ci-cd-pipeline-template-for-data-projects/blob/master/kubernetes/deployment.yaml#L9

Using a Google Access Key with proper entitlements give access to authorised users or system to download these images.

In public academic project it is encouraged to deploy these images in a public repository, for example Docker Hub.

5.2.2.3 Managing secrets

A use case requirement was to be able to manage secret access tokens inside the production application. The question was how secrets can be passed to a running container in Kubernetes.

The real requirement was how the client python application can download data from a different server and access a separate bucket. To access this bucket we have to pass authentication details to the python application.

One option could be injecting tokens when we build the image which runs the application. However, in this case, anybody who can download the image, get access to the remote storage.

More safer option to pass secret with an API call to the application is using POST request via https. However in this case we need an external call from and to our application, which would involve contributions from other systems. This solution can work only if we have an independent authentication service in our cluster.

In our use case we preferred to implement secret management in Kubernetes level, so we can pass secret tokens from using safe environment variables to the Kubernetes cluster which inject these secrets inside the running container. This solution is described in *Section 4.3.8.*

5.2.2.4 Running scheduled tasks

Other use case requirement was to run scheduled task. Data analysis applications often download raw data, log files or reports from an external source, and they provide an updated result. The up-to-date source data should be downloaded regularly, usually in a strictly scheduled way. The question was how the application can update data source using Kubernetes or modern pipeline.

If you run your application in a standard Linux environment this task is mainly solved with cron jobs. Cron jobs are system level schedulers and they can run a script or application in given time or after a certain period of time. However cron jobs can be used only if the application runs directly on a server constantly or a cron job installed on the server by a server administrator.

Using Docker containers and Kubernetes, our Linux environment lifetime is limited, a container can be restarted any time and a scheduler is not as reliable. Especially when the application would like to run a task periodically, for example every 30 minutes. If the application have been rebooted the scheduler counter would be reset as well.

Our user implemented a scheduler in their python application which partly solved their requirement. They also added an http API endpoint to be able to trigger source data update manually, using a URL.

Kubernetes provides a CronJob Controller which can be used to run scheduled applications. Ideally the scheduled task should be implemented in a separate application and Docker container, so it can run in an independent Kubernetes Pod. This pod can be scheduled with Kubernetes CronJob Controller. This solution has not been implemented in our pipeline yet. It could be a useful improvement in future work.[21]

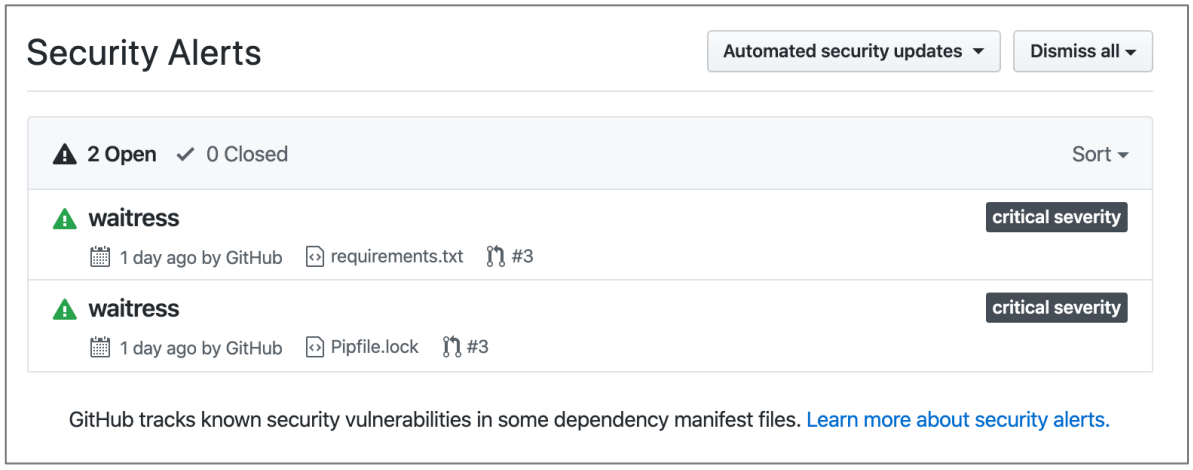
5.2.2.5 Managing security

A question about managing security and update vulnerable packages surfaced during user tests.

The actual implementation is supported with two security check automatically. Firstly the source control systems (GitHub and GitLab) provides package security check automatically *(Screenshot 4)*. These systems send emails and open pull requests to alert maintainers about vulnerable packages. Additionally Google Cloud Registry scans all of our Docker images regularly. Google scan application packages and raise issues in terms of the wrapped Linux environment as well.

[21] More details about CronJob Controller in Kubernetes:
https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

It is a common step in corporate development pipeline to implement an external audit and security check, which automatically runs before deployment. Most of these services are commercial, and open source support is limited. However SonarQube, a code quality and security check cloud service provider, has recently announced[22] a more integrated support for GitLab users, so this feature can be added to our pipeline in the future.



Screenshot 4. GitHub alerts about security vulnerabilities found in our project

5.2.2.6 Importing third party libraries via GitHub

One of the use cases of our industry partner was to import a third party library via GitHub repository.

Originally they tried to add packages directly to requirements.txt using the following format:

```
-e git+https://github.com/some-name/some-repository#egg=some-package-name
```

However the pipeline stopped showing the following alert: "ModuleNotFoundError: No module named some-package-name"

The template pipeline uses Pipfile to manage dependencies.[23]

```
<vcs_type>+<scheme>://<location>/<user_or_organization>/<repository>@<branch_or
_tag>#egg=<package_name>
```

[22] Source: SonarQube 8.1 - https://www.sonarqube.org/sonarqube-8-1/
[23] Source: https://pipenv-fork.readthedocs.io/en/latest/basics.html#a-note-about-vcs-dependencies

A git based package can be added using the following format.

```
$ pipenv install -e
git+https://github.com/requests/requests.git@v2.20.1#egg=requests
```

Our custom implemented Pipenv script can generate the final requirements.txt.

```
$ pipenv run lock-req
```

It generates a new requirements.txt file, so we can use it as follows:

```
$ pip install -r requirements.txt
```

Using the above practice keep our package management consistent and open the opportunity to add packages from GitHub or other git repository management service.

5.2.2.7 Customisation

Feedback from users helped to highlight areas where a customisation would improve further the pipeline reusability by providing custom or dynamic values, such as project name, cloud provider, domain names, etc. The opportunity to change more options in the pipeline and the running environment, the framework became more reusable and flexible and users are able to reinstall and reuse to different tasks.

The customisation allows to use the pipeline with different cloud credentials, be more portable and use different names and deployment targets. The custom value is stored in environment variables. For example the project name, cluster details, secret value stores, and Docker configuration.

We use custom values between pipeline cycles also. The values on developer machines can be different than in test or in the cloud based environment. The ability to recreate the same environment on our development machine and in the cloud help us to try out ideas locally, so we can experiment and debug faster, because we have direct access to the components, such as Docker engine and local Kubernetes.

However, this customisation freedom cannot be limitless. A certain level of control and restriction keep the balance of reusability and efficiency. An opinionated (strictly defined) and hardcoded solution can help in development because the developer can focus on different parts of the application meanwhile the opinionated solution works out of the box.

5.2.2.8 Logging support

Twelve factor application principal lists logs as one of the principals (Wiggins, 2017). It is important to implement a logging feature for transparency, auditability and accelerate debugging. One of the test users also mentioned that our framework should support logging out of the box.

It can be argued who should be responsible for implementing logging in an application for data scientists. Developers can implement abstracted log management. The app should provide logging methods. DevOps engineers also can add more detailed reports about environment and pipeline steps. However we cannot avoid asking data scientists to extend their implementation with logging calls, because it is also part of the software development process.

5.3 Pipeline activity as empirical evaluation

Source control system (GitLab repositories) recorded activities automatically. This information helps see how data scientists interacted with our pipeline, such as commit frequency, successful builds, error reports. We can evaluate their practices and observe how the pipeline helped them.

The following graph *(Figure 7)* provides a screenshot of the production targeted real world application repository.

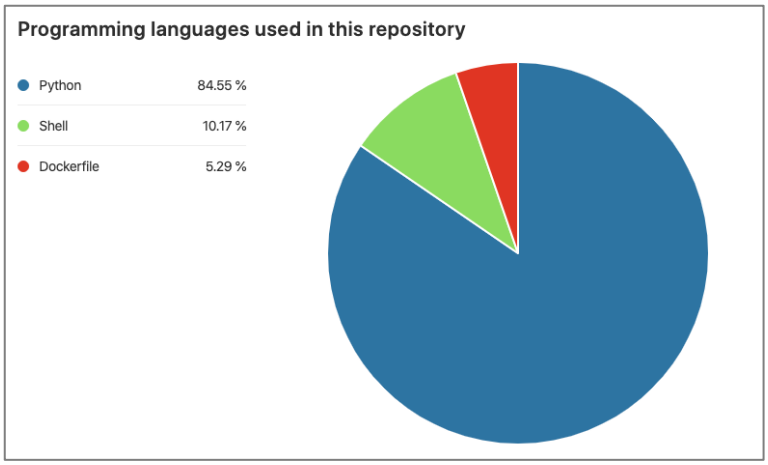


Figure 7: Programming languages used in user test repository (Source: GitLab project)

More than four-fifths part of the codebase is written in Python. For this reason supporting Python projects and improving Python skills are important.

The following charts *(Figure 8)* shows commit activities in the test repository. There were commits for 5 months. The number of total commits is 213.

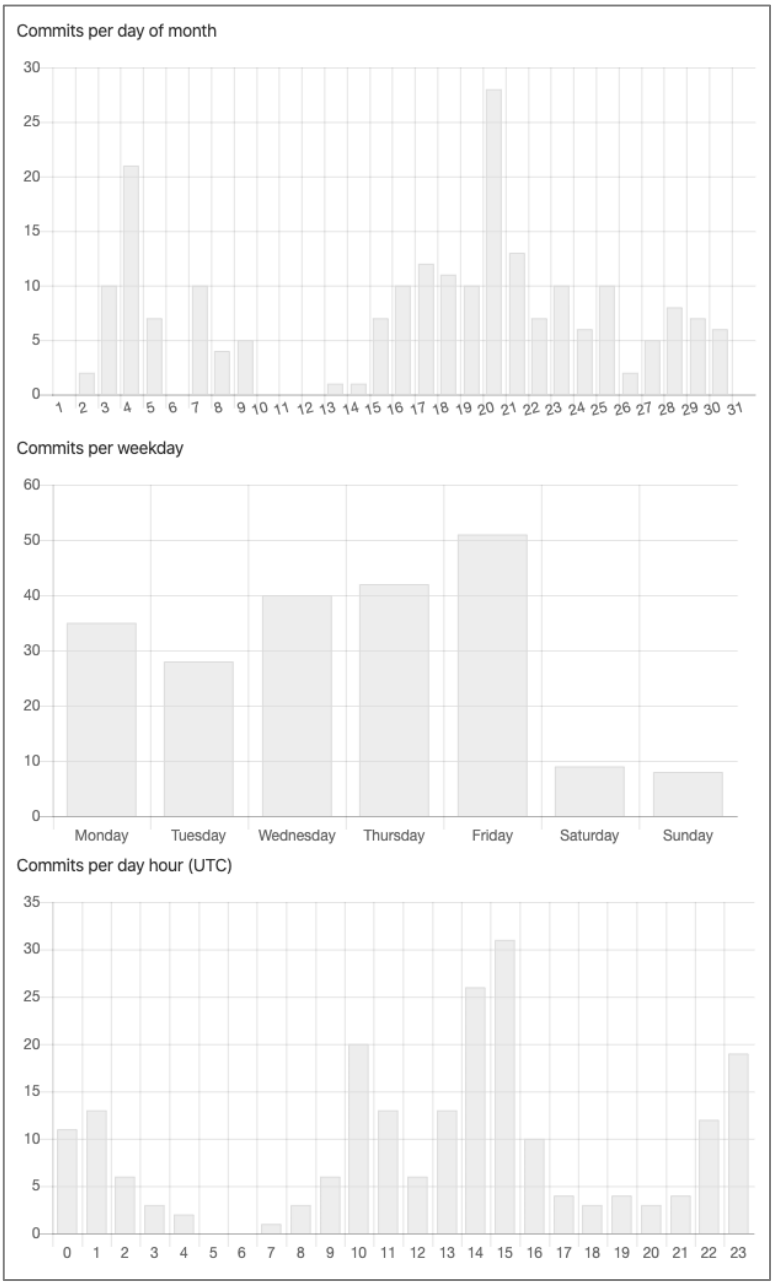


Figure 8. Commits statistics (Source: GitLab repository)

Overall statistics of the CI/CD pipeline *(Table 8)*:

Description	Result
Number of opened and closed issues	37
Number of created Docker containers in Google Docker Registry	87
Total number of pipeline run	266 run

Total number of successful pipeline run	173 run
Total number of failed pipeline run	85 run
Success ratio of pipeline run	67%

Table 8. GitLab repository activities (Source: GitLab and Google Docker Registry)

The next figure shows how the pipeline activity has changed. In the first wave our users used an experimental model to practice and adopt new DevOps rules. It was a setup and learning phase. In the following period the pipeline was not actively engaged, because our users determined their new production targeted requirements. It was the architecture phase. As we can see clearly in the last phase in *Figure 9*, the pipeline activity is increased. Data scientists started to implement their real world application and their progress accelerated as they utilised the pipeline. It was the implementation phase.

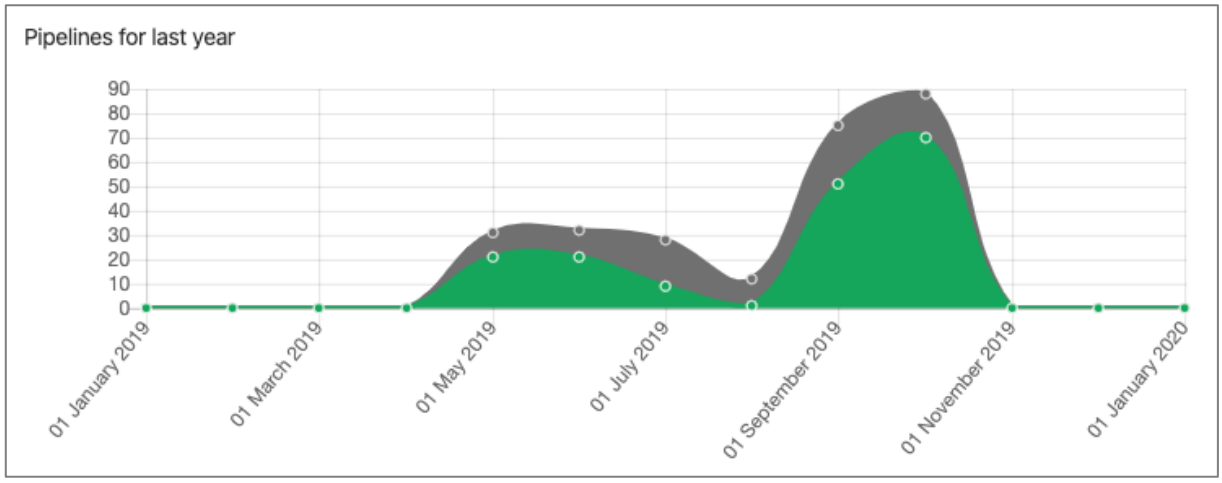


Figure 9: Pipeline activity, where the grey area shows all pipeline run and the green area represents successful run (Source: GitLab)

GitLab can show us the duration of the pipeline *(Figure 10)*. Each time when a new commit was pushed, it triggers the pipeline. Shorter run help us to iterate faster. As we can see in the last 30 commits our average time has decreased. Our pipeline runner container, namely "Gitlab Runner", deployed to Google Kubernetes Engine. Adding more resources (memory, processor power) to the runner pod was reduced this time.

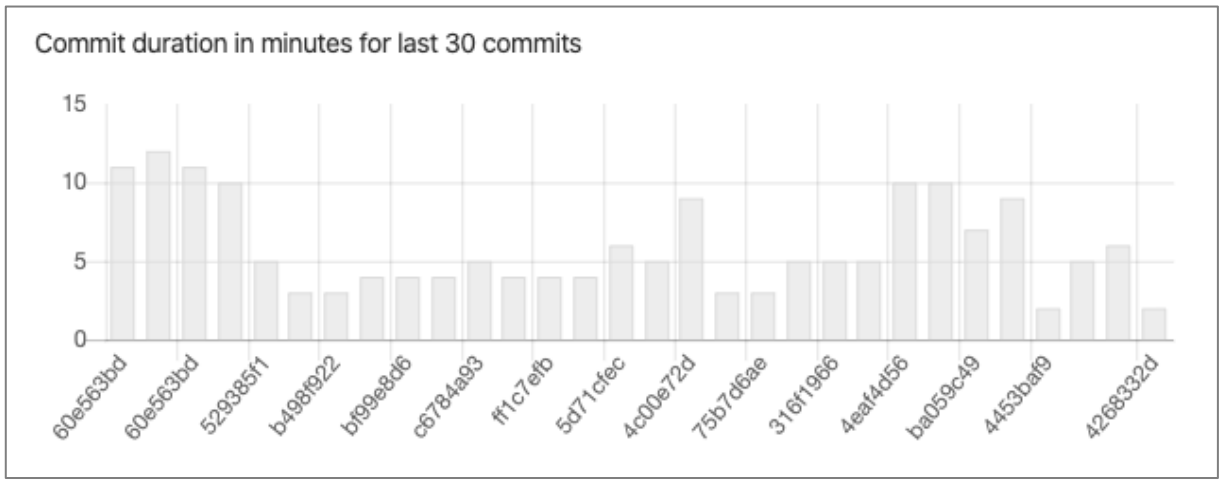


Figure 10. Commit duration in minutes (Source: GitLab)

Description	Result
Number of all merge requests	51
Merged merge requests	41
Closed without merge	10

Table 9. Merge request statistics

Merge request (aliases: pull request, PR) statistics *(Table 9)* help us to understand how long a merge request lives.

Most of the merge request finally merged however, there were a few PR which were closed.

A few examples why a merge request have not been merged:

- changes finally implemented in a different PR,
- experimenting with a solution but the final implementation raised in a different merge request,
- PR was empty.

We can individually check merge requests in GitLab. When our users started to use the pipeline they added more commits to a PR and the lifetime of a PR was long.

Here are two examples *(Table 10)*, one from the earlier phase of the experiment and one from later:

Merge Request Number #26	**Merge Request Number #42**
Number of commits: 39	Number of commits: 3
Pipeline run: 19	Pipeline run: 3
Changed files in the repository: 42	Changed files in the repository: 7
PR opened: 15th of July	PR opened: 4th of October
PR closed: 16th of Sep	PR closed: 4th of October
PR was live for 2 months	**PR was lived for less than one day**

Table 10. Merge request comparison (Source: GitLab repository)

We observed that our users started to embrace better practices. As they shared with us in interviews, they realized reviewing big Merge Requests and managing long lived PRs are difficult, so they started to follow our suggested practices: commit often, open pull request often, and close merge request quickly.

5.4 Evaluation summary

As a conclusion of user tests and case studies we can determine areas which were improved thanks to user feedback. We also can create a list of desired knowledge sharing sessions for data scientist.

Improved features based on user feedback:

- Secret token management.
- Improved customisation options with environment variables to increase template reusability.

Future work:

- Implement extended security check and code quality check.
- Implement automatic logging feature.
- Add Amazon AWS support.
- Using CronJob Controller in Kubernetes for scheduled tasks.

Positive feedback about the pipeline:

- Test runner increased code stability. Data scientists are forced to invest more in code quality and adding test, so they picked up better practices thanks to the pipeline.
- Automatic deployment helped to review changes in minutes. Without the pipeline data scientists had to wait for system administrators to deploy their application.

Need more improvement:

- More documentation, especially more diagrams to explain in the template, how different tasks and components are working together.
- Supporting Windows development environment.
- Supporting Amazon AWS.

More knowledge sharing for data scientist about the following topics:

- Advanced Python programming. Modular programming. Package management. Using types in Python.
- Testing practices. Unit testing and test coverage.
- Adding logging features to custom implemented logics and activities for monitoring the application behaviour.
- Docker containers.
- Basics of Kubernetes.
- Google Cloud and Amazon AWS.

5.5 Summary

We built a pipeline which is not only for experiment. It is dedicated for real world applications. It is evolved and improved through the evaluation process and it is ready to use in production environment. Once it was ready, our industry partner started and has kept using this pipeline to deploy data analytics applications in production environments, which was a great feedback.

Our project is open source and our goal is to improve it further to be ready for more use cases, including supporting other cloud systems, such as Amazon AWS and supporting different programming languages.

6. Conclusion and Future work

6.1 Conclusion

Our main motivation was to provide an opinionated (predefined, strict, template based) development environment for data science projects to accelerate and improve product delivery.

Our hypothesis was that providing a template project for data scientists with a working DevOps pipeline, and helping adopt modern development practices, could speed up the development process while increasing reliability and maintainability.

The strict, dictated, and opinionated template shortened bootstrapping time frame of a new project. The pipeline rules when combined with automated linting, formatting, and testing tools forced developers to add tests, and it essentially improved code quality. Reliability increased because deployment of out of the predefined standard implementation was stopped by the pipeline. Customisation features and fast feedback loop improved maintainability, because developers were able to check new implementations in review and staging environment before deploying into production.

We reviewed our preliminary research and our original findings in *Chapter 2*, which led to experimenting with alternative solutions, then finding the ideal combination of building blocks of the proposed DevOps pipeline.

Design and Implementation in *Chapter 3* described the main building blocks of the product and introduced the three layers of the software development pipeline: the application layer, the developer experience layer, and the pipeline orchestration layer.

The following chapter, in *Chapter 4*, a detailed technical tutorial explained the product and provided instructions about implementation, including technical prerequisites using GitLab, Google Cloud Kubernetes Engine and special Python features.

In *Chapter 5* we discussed our evaluation, our user testing methodology, and the evaluation of the use cases.

During our evaluation test users and industry partners developed real world applications which were deployed in production. Their feedback supplied evidence that our provided DevOps template and practices accelerated and improved their application development and delivery, which supported our hypothesis.

6.2 Future work

Further research on this topic could examine the usage of the framework to support different cloud based services, operations systems and programming languages.

As we previously highlighted a further research area could be to implement extended security check and code quality check in the pipeline, evaluating and including tools like SonarQube, Codacy or similar services.

Test user feedback suggested that an out of the box logging support can be beneficial. Adding customisable logging to the pipeline is a great candidate for future work.

Our preferred tools are also evolved as we mentioned earlier GitLab implemented Amazon AWS Kubernetes support. The pipeline template can be improved further with a customisable extension which automatically switches between cloud providers.

Kubernetes provides a dedicated support to run scheduled “cron” jobs. It is an important use case in data science projects to update the raw data set regularly. The pipeline template can be extended with an example to add Kubernetes CronJob Controller to run scheduled tasks automatically.

7. Bibliography

Brockschmidt, K (2016) *The Source of Truth: The Role of Repositories in DevOps*, Microsoft, MSDN Magazine, Volume 31 Number 9. Retrieved from: https://docs.microsoft.com/en-us/archive/msdn-magazine/2016/september/mobile-devops-the-source-of-truth-the-role-of-repositories-in-devops

Debre, Z. (2019) *Building A Scalable, Production Ready Development Pipeline For Machine Learning And AI Services Using Modern Continuous Integration Practices And Infrastructure As Code Solutions.* Retrieved from: https://github.com/zoltan-nz/comp-501-research/blob/master/research-essay/COMP-501-research-essay-zoltan-debre-300360191.pdf

Forsgren, N. (2019) *The 2019 Accelerate State of DevOps: Elite performance, productivity, and scaling, Google Cloud Blog.* Retrieved from: https://cloud.google.com/blog/products/devops-sre/the-2019-accelerate-state-of-devops-elite-performance-productivity-and-scaling

Harper, R. (2016) *Practical Foundation for Programming Languages, Section 23.4 Static Versus Dynamic Typing, Page 210,* Retrieved from: http://www.cs.cmu.edu/~rwh/pfpl/2nded.pdf

JetBrains (2019) *The State of Developer Ecosystem in 2019*, Retrieved from: https://www.jetbrains.com/lp/devecosystem-2019/python/

Lethbridge TC, Sim SE, Singer J (2005) *Studying software engineers: data collection techniques for software field studies, Empir Softw Eng 10(3):311-341*

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

Nelson, C.; Napke, H. (2020). *Building Machine Learning Pipelines, O'Reilly Media, Inc August 2020, ISBN: 9781492053194,* http://oreilly.com/catalog/0636920260912/errata

Nielsen, J. (2000) *Why You Only Need to Test with 5 Users.* Retrieved from: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Parizo, C. (2019) *Why containers are critical to successful DevOps projects. TechRepublic.* Retrieved from: https://www.techrepublic.com/article/why-containers-are-critical-to-successful-devops-projects/

Runeson, P.; Höst M. (2008) *Guidelines for conducting and reporting case study research in software engineering* DOI 10.1007/s10664-008-9102-8

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

Wenzel, M.; Parante, J; Schonning, N (2019) *Containers as the foundation for DevOps collaboration, .NET microservices - DevOps e-book, Microsoft.* Retrieved from: https://docs.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/docker-application-lifecycle/containers-foundation-for-devops-collaboration

Wieldt, T (2019) *DORA's 2019 Report: 'DevOps Has Crossed the Chasm', New Relic Blog.* Retrieved from: https://blog.newrelic.com/technology/dora-accelerate-state-of-devops-2019/

8. Appendix

8.1 Preliminary COMP 501 Research Essay

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

Retrievable from: https://github.com/zoltan-nz/comp-501-research/blob/master/research-essay/COMP-501-research-essay-zoltan-debre-300360191.pdf