Low code Ops and the art of abstraction

Gustaf Lidfeldt, @glidfeldt, lidfeldt@kth.se
Isak Hassbring, @hassbring, isakha@kth.se

DD2482 DevOps and Automated Testing
KTH Royal Institute of Technology
April 2021

Please note *that this is a non final draft for the feedback-submission.*

1 Introduction

DevOps in general could be seen as a set of practices serving as abstraction layers that aims to let more people be more effective, by building an interface around difficulties - hence abstracting them away [10]. The same goes for many engineering fields, e.g. Computer Engineering is an abstraction on top of Electrical engineering and programming languages are abstractions (sometimes even on abstractions) on machine code. Among programmers, a "one-liner" is often considered superior compared to multi-line programs (given equal task and performance). Why? Because we love the art of simplicity and abstraction. It is as beautiful and satisfying as it is easy to understand [1] [2].

Between 1970 and 1990, the fourth-generation programming languages where emerging[6]. The second-generation of programming languages had been able to abstract machine code into assembler. The third generation took it a step further with the compiler. The fourth generation aimed to make coding more accessible by having the instructions be more similar to our natural language. The computer program Excel lets "the average Joe" compute calculations without programming skills. The fourth-generation of programming languages has to today emerged as low-code/no-code environments.

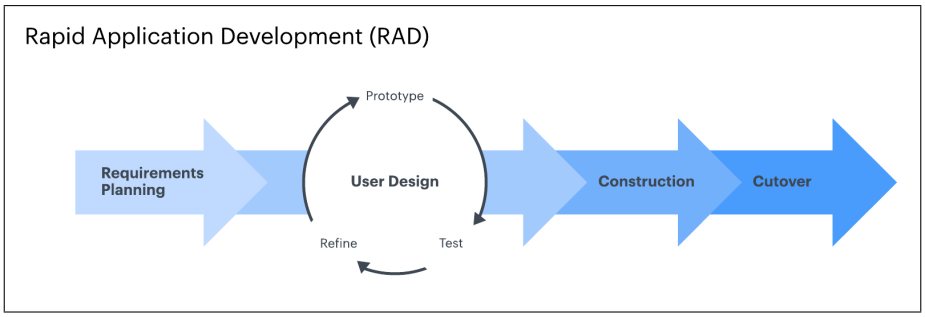
Now, a lot has happened since then. Today, platforms such as Peltarion.ai even go beyond the once beloved one-liners and let creators (people building applications) set up, train, test and deploy cutting edge deep learning techniques without writing code. Bubble.io lets you write interactive front ends, also as a no-code practice. Shuttleops.io offers a no-code continuous delivery platform that literally lets you draw your operations infrastructure / architecture and deploy.

As an example, we all know what the emergence of the no-code platform wordpress.org meant for web developers as it completely reshaped their industry. So, what does the broad no-code emergence mean for teams in both development and operations? Is this the beginning of the end of DevOps as we know it? We will explain why this matters for a DevOps engineer (or any engineer in general!) and discuss possible practical implications of low-code in the future.

2 Background

2.1 Low-code abstraction

Low-code and no-code development refers to the usage of a GUI (graphical user interface) for code generation instead of typing code by hand within an IDE (integrated development environment). Low-code development typically follows an agile methodology such as RAD (Rapid Application Development).



The term *low-code development* refers to development requiring little code knowledge, and no-code similarly refers to development without any code at all. In the scope of this text we refer to both within the term low-code.

While low-code platforms are not (and will perhaps never be) useful in all development scenarios, they are increasingly useful for a large and growing set of applications and practices. A UI/UX designer can for instance use *Figma* to automatically generate fully functional React-JS components from their design, for prototyping and user testing. This hugely speeds up and automate the front end application development. It lets the front end developer take a step back and focus on API:s, integration, functionality, putting the pieces together instead of actually building the views. This is a more efficient use of resources and increases productivity.

Low-code platforms are especially useful in smaller, agile organizations or for smaller projects with few development resources. Low-code platforms or tools fit very well in such application development life-cycles and "lets the builder build" and create custom-made apps for specific use-cases - faster and easier.

"The rise of low- and no-code is comparable to mobile devices being packed with computing power that once resided in mainframes. It is a level of abstraction that keeps going, making something that is tedious, hard and time consuming much easier to do."
- Mike Duensing, CTO at the low-code cloud platform Skuid.

2.2 DevOps perspective

What serves as a good area to provide with a low-code platform framework? Standardized development practices could be a perfect scenario. Why? Well, if something is standardized, it typically involves repetitive tasks. Hence you could build reusable, modular, object-oriented code around it. And if so - you'd be able to provide a GUI for it as well. And in the scenario of a standardized end-to-end independent task in development or operations, the entire task development could be made through a low-code GUI.

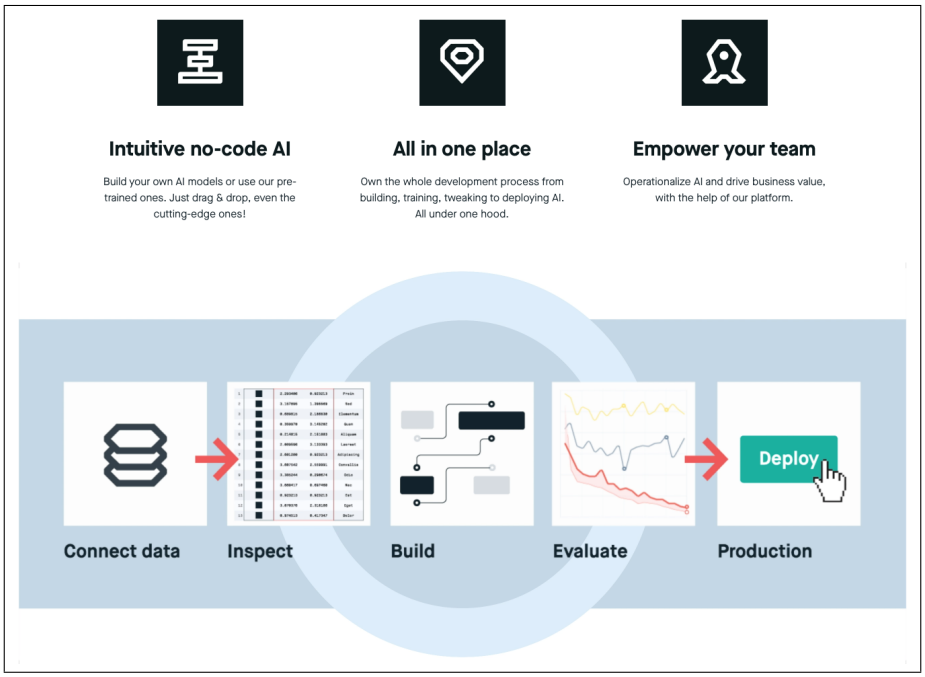
Now, what's interesting here is that standardized dev/ops practices is often times *exactly* what DevOps provides development and operation teams with, and hence a subset of these practices can be provided with end-to-end GUI:s. This is in fact what several companies have realized, and a lot of big players are entering the field one way or another. In fact, *all* the devops cloud tools provide solutions that *do* fit into the low-code definition. But they still have a somewhat long way to go - and for us to sneak peak into the future, we will look at the ones that are one step ahead - such as Peltarion and Figma, in the next section.

The difference between DevOps and Low-code Ops is that DevOps streamlines everything like an hour-glass, through the use of *code*. What if there is no code involved in parts of the workflow? Suddenly it would be a struggle to put git be the core of the work processes. This is a struggle for DevOps today that must be addressed to keep up with the pace of low code - and the reason why this Ops-buzzword might be worth using.

3 Practical examples

In this section we will cover some concrete practical examples.

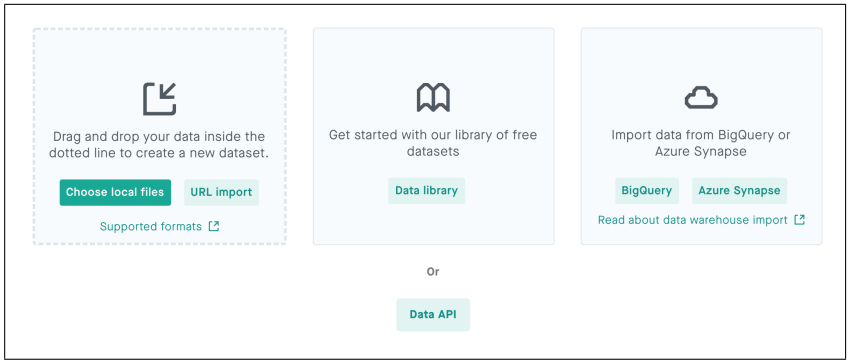
3.1 Peltarion - a low-code platform perfect for ML Ops



Peltarion [8] is a Wallenberg and EQT backed Swedish company that provides an end-to-end machine learing (ML) platform that makes the fuzz unfuzzy, when it comes to ML usage and ML Ops. Deeply technical companies such as Tesla, Spotify, King etc are using the Peltarion platform, so it is definitely a big deal. We've tried it - here's how it works:

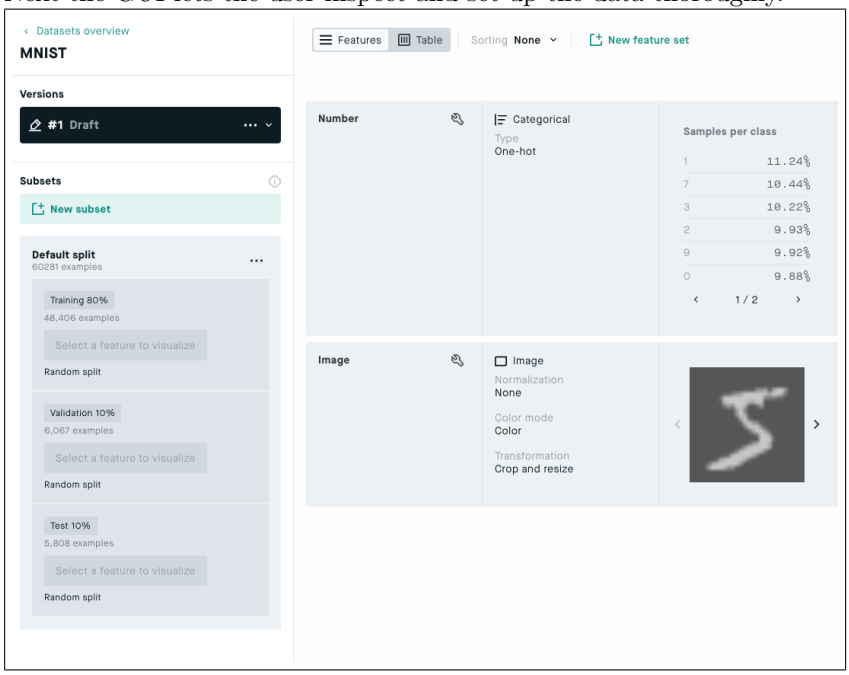
1. First, you create a project. A project holds the pre-processing of datasets, model building, evaluation and deployment. A project is a shared resource, it can be used with others.

2. Then, you're asked how you would like to provide your model with data.

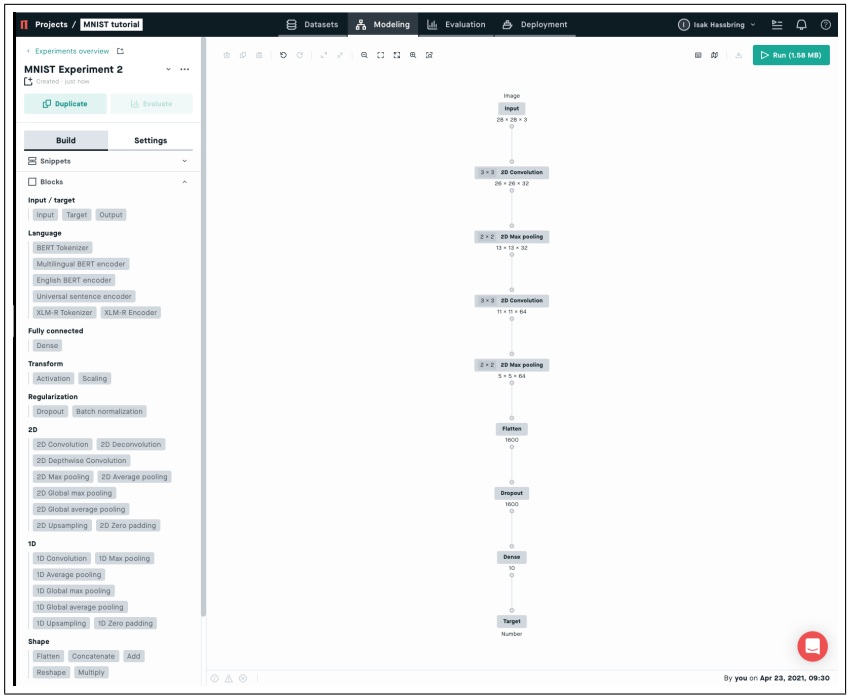


There are 4 main ways to connect to data, see image above.

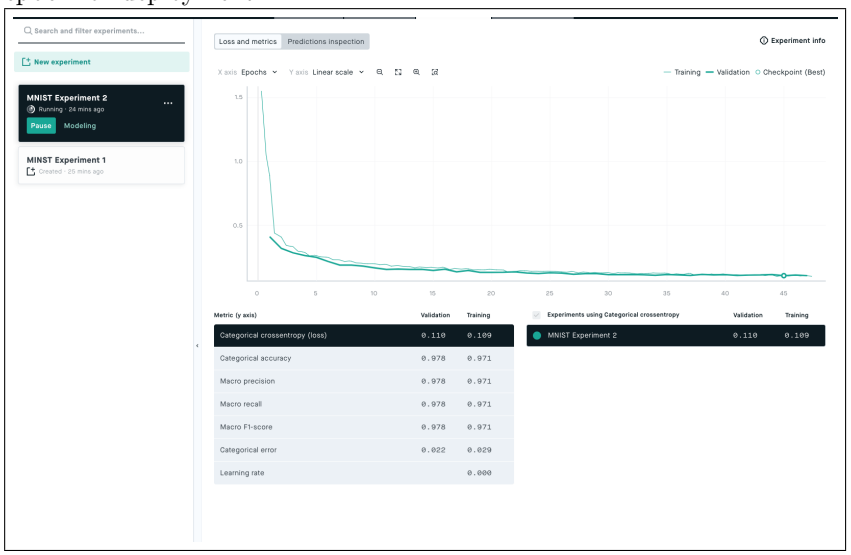
3. Next the GUI lets the user inspect and set up the data thoroughly.



4. Now it is time define how to use the pre-trained model. When ready, one simply presses "Run" - and the model starts running.



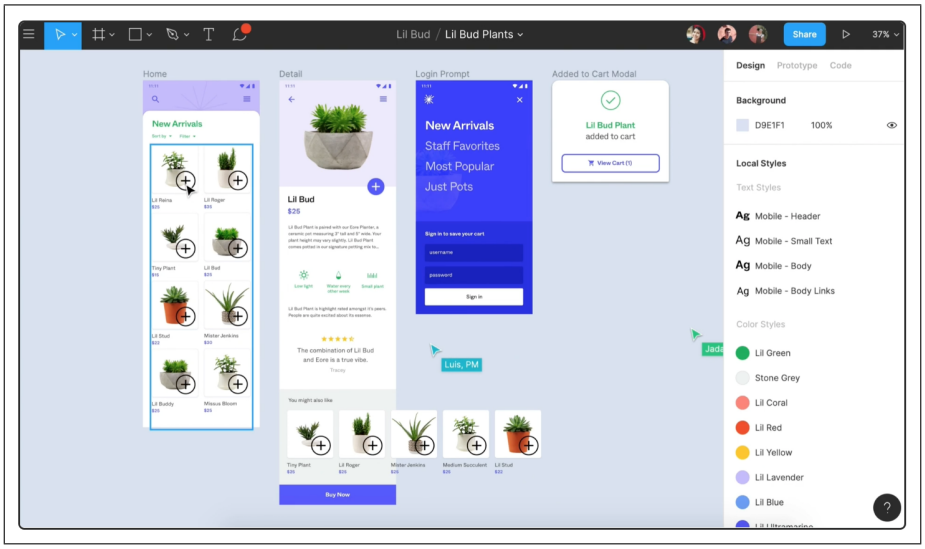
5. By pressing "Run" in step 4, we move over to the Evaluation section. Here, we're able to visually follow along during the model run and evaluate its performance. The platform automatically prepares the best performing epoch for deployment.



6. Finally, literarly by pressing "Deploy", the model is deployed to production, directly accessible though an auto-generated API end point.

For anyone familiar with more "tradional" ways of handling ML development processes and operations, this is a huge step indeed. As far as abstraction and simplicity goes, the Peltarion platform makes is much easier to work with ML Ops, and it is doing so without exhaustive requirements on coding skills.

3.2 Figma



Figma is a popular low-code tool used within front-end development and design teams to rapidly speed up development phases including GUI parts. Figma lets a designer or front end engineer design an element or a component just as you would do it in a traditional design tool - but instead of generating an image, you result is fully reusable JavaScript components (e.g. React-components), which can be used directly in prototyping. The use of Figma speeds up the actual code generation and enables for people who typically are not considered programmers, to participate in front end development or code generation. As compared to traditional hand-coding, this is a big change [9].

3.3 ShuttleOps

ShuttleOps is a "no-code continuous delivery platform that makes it quick and simple to build, deploy and manage complex distributed applications"[3]. Using a simple drag and drop interface, ShuttleOps allows teams and organizations to quickly adopt and scale their DevOps practices.

4 Limitations

4.1 Abstraction comes at a price

Testing is difficult as awareness, visibility and control decreases. A graphical user interface that allows a developer to drag and drop different functionality is indisputable a time saver. Without full control of what is abstracted, testing and performance control could be an issue. As an example, imagine that you use low-code software to bind data from a poorly written database query to a user interface. After checking that everything works you realize that you application is extremely slow. What is the reason for the slow speed? The UI? The database? The network connection? A regular developer would likely be able to debug and find the faulty database query, but would a low-code developer who was not hired to know be able to figure it out? After all, the low-code developer was hired to drag and drop features. Organizations need to realize that there are many good reasons for why it is costly to develop software. Good experienced developers are costly, but if an organization takes shortcuts they might spend more on fixing "bad code" when they could have just writing good code from the beginning.

4.2 Source code ownership

The biggest no-code platforms all state that the data you generate is your own. However, for no-code platforms you generally do not own the source code of your creation. This results in you having to rebuild your entire project in the case you wish to not use the platform anymore. No-code platforms can be great if you quickly and cheaply want to develop a minimal viable product to help raise funds for your startup. But further development might be limited by number of plugins the platform have. Furthermore, if the value of your company is based on an algorithm and you do not own the ownership of said source code potential investors would be reluctant to invest in you.

Bubble addresses some of these concern by saying that you own your design and in the case that you wish to export your project they can provide a "raw dump of your application logic and visual design as a JSON file"[7].

5 Conclusion

While DevOps enables organizations to speed up the delivery of code it fails to impact the speed of production of code. Continuous integration and deployment is great, but a system is only as fast its' slowest part - actual code commits. An organization might need to build software quickly for internal or external use, but often times it is too expensive to hire software engineers. Here low-code development could play a crucial part.

Is this the end of DevOps? No, it is not. But, a growing set of what is typically considered DevOps practices will probably not require the DevOps engineer expertise in the future, due to low-code platforms providing abstraction. And for the DevOps practice itself, it *could be* a good idea to try to emerge from the dependency on services such as git being the core foundation of DevOps, and try to find a way to better include or collaborate with low-code development practices. For instance, such a solution could be to come up with a tool that can let organizations manage and work with version control not only for pure code files, but also for progress and development made in low-code tools.

Nevertheless, the need for traditional coding languages is still and will still be critical in restricted areas for a while. All the existing no-code platforms comes with their respective limitations and developers will still need to fill those gaps. However, low- and no-code platforms provide smaller companies with limited resources to quickly throw together an application that fulfills certain requirements.

References

[1] How low-code/no-code platforms may reinvent DevOps
https://medium.com/enterprise-nxt/how-low-code-no-code-platforms-may-reinvent-devops-a9ecb549cbbd

[2] How Low-Code and No-Code Fit into a DevOps Strategy
https://www.informationweek.com/devops/how-low-code-and-no-code-fit-into-a-devops-strategy/a/d-id/1334017

[3] Shuttleops.io
https://www.shuttleops.io/

[4] Forrester, *Low-Code Platforms Deliver Customer-Facing Apps Fast, But Will They Scale Up?*
https://web.archive.org/web/20170202125753/https://www.forrester.com/report/LowCode+Platforms+Deliver+CustomerFacing+Apps+Fast+But+Will+They+Scale+Up/-/E-RES122546

[5] SearchSoftwareQuality, *Why the promise of low-code software platforms is deceiving*
https://web.archive.org/web/20190501184114/https://devopsagenda.techtarget.com/opinion/Why-the-promise-of-low-code-software-platforms-is-deceiving

[6] "Minitrack Introduction," in 2014 47th Hawaii International Conference on System Sciences, Big Island, Hawaii, 2002 pp. 279. doi: 10.1109/HICSS.2002.10113
thttps://doi.ieeecomputersociety.org/10.1109/HICSS.2002.10113

[7] FAQ — Bubble
https://bubble.io/faq

[8] Peltarion AB
https://www.peltarion.com

[9] Figma
https://www.figma.com/

[10] Cap-Gemini, How to source your DevOps teams
https://www.capgemini.com/se-en/2019/04/how-to-source-your-devops-teams/