A DevOps Approach for the Systematic Development and Evolution of Built Assets Digital Twins

Sara Aissat, sara.aissat.1@ens.etsmtl.ca
Department of Software Engineering and IT, École de Technologie Supérieure (ÉTS), Canada

Jonathan Beaulieu, jonathan.beaulieu.2@ens.etsmtl.ca
Department of Software Engineering and IT, École de Technologie Supérieure (ÉTS), Canada

Francis Bordeleau, francis.bordeleau@etsmtl.ca
Department of Software Engineering and IT, École de Technologie Supérieure (ÉTS), Canada

Ali Motamedi, ali.motamedi@etsmtl.ca
Department of Construction Engineering, École de Technologie Supérieure (ÉTS), Canada

Érik Poirier, erik.poirier@etsmtl.ca
Department of Construction Engineering, École de Technologie Supérieure (ÉTS), Canada

Abstract
Digital twins (DT) have emerged over the last decade as a potential solution to monitor and improve different aspects of built assets. However, little emphasis has been placed so far on the iterative development and evolution of DTs. In software engineering, DevOps has established itself over the last decade as the leading paradigm to improve software processes. This paper proposes a DevOps approach for the development and evolution of DTs for built assets. It focuses on three main aspects: (1) Software methodology to support the systematic and iterative development and evolution of DTs; (2) Microservice architecture to enable the independent development and composition of different aspects of DTs; and (3) DevOps infrastructure to support the development, integration, and deployment of DTs. The articulation of the proposed approach is based on experience gained in the development of various DTs in the context of built assets and other application domains.

Keywords: DevOps, digital twin (DT), built assets, BIM, microservices, software methodology

1 Introduction

Digital twins (DT) have emerged over the last decade as a potential solution to enable the monitoring and improvement of several aspects of built assets across the different stages of their lifecycle (Jones et al., 2020). In this paper, we focus on the Operation and Maintenance (O&M) phase of build assets. The services offered by existing DTs in the built environment include the monitoring and improvement of occupant comfort, energy consumption (Halhoul Merabet et al 2021), air quality (Chenari et al 2016), and security (Camposano et al 2021), among many others. Their development has required addressing significant challenges related to different aspects of the digitalization of built assets, including integration of heterogenous data from different sources and device types, data security and secure access to guard against unauthorized access and cyber-attacks, as well as deployment of software components on heterogeneous execution platform/environments composed of IoT devices and cloud. However, little emphasis has been placed so far on how to concurrently develop and maintain DTs while increasing their range of functionalities in an iterative and evolutionary manner to adapt to ever changing environments, and evolving user requirements and objectives.

DevOps has established itself over the last decade as the leading paradigm to improve software processes in software engineering (Forsgren et al 2018). This approach has allowed software organizations to increase their agility to adapt to environments that are constantly evolving in order to deliver solutions faster, with higher quality, and that are adaptable to user needs. DevOps aims to integrate the development (Dev) and operations (Ops) phases into a seamless, end-to-end, continuous process flow and emphasizes continuous integration and delivery, and quick feedback loops. It enables continuous improvement through automation and monitoring at every stage, from design to deployment, including planning, development, testing, integration, release preparation (building/packaging), and monitoring. Its adoption by industry leaders such as Amazon, Netflix, Google, and Facebook has led to spectacular progress (Kim et al 2021).

So far, DTs in the built environment have been mainly developed in the context of research projects or for prototypes/Proof of Concepts (PoC). Their development has been typically done in an ad hoc manner, following a known and recognized solutions architecture yet not in a structured nor through a defined and validated process. This leads to issues with scalability and replicability of development approaches and more importantly consistency and predictability of resource planning and management of outcomes (Shahzad et al 2022). The problem addressed by our research is the lack of systematic approach to support the scalable engineering and evolution of DTs.

In this paper, we propose an approach based on the best practices of DevOps to support the systematic development, deployment, and operation of industrial DTs in the context of built assets. Three key areas are presented and discussed in the paper: (1) Software methodology to support the systematic and iterative development and evolution of DTs; (2) Microservice architecture to enable the independent development and composition of different aspects of DTs; and (3) DevOps infrastructure to support the development, integration, and deployment of DTs. Regarding the DevOps infrastructure, we focused on three main aspects: the role of Git as a version control system to provide a single source of truth and support team collaboration, Continuous Integration (CI) to support the development phases, and Continuous Deployment CD to support the release and deployment phases. The proposed approach is based on experience gained in the development of various DTs in the context of built assets and other application domains.

2 Background

2.1 Digital Twin of the GRIDD Lab

A DT is a digital representation of an actual system, also called the Actual Twin (AT), that is dynamically and constantly updated with system data, and provides a set of services related to the system. Examples of DT services include the monitoring of specific properties, detection of issues, simulation, analysis of what-if scenarios, and data analytics to identify correlations between different system properties. One of the main advantages of using a DT is that it can provide advanced services that could not be provided without the use of a digital representation, e.g. simulation, advanced analytics, and AI/ML.

The GRIDD[1], a research laboratory at ETS, focuses on advancing digital transformation, industrialization, collaboration, and sustainability. Committed to driving sustainable change, particularly in the construction industry, the GRIDD research group has increasingly focused on innovative projects that explore the potential of DTs. The GRIDD Lab is the multi-functional room at ETS that is used for both research and teaching activities. It has been the source of multiple DT projects that aimed at enhancing various aspects of its functionality and management. These projects have allowed experimenting with various technologies and contributed to the definition of requirements and challenges for the development of a systematic approach and framework to support the engineering of DTs.

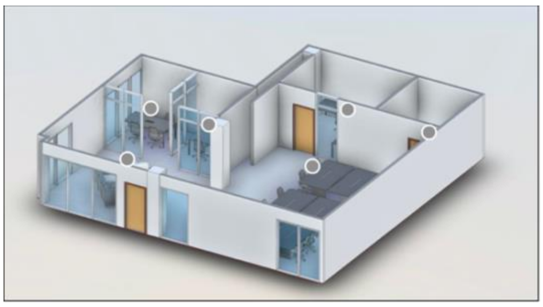
To illustrate the approach presented in this paper, we use the example of a DT that has been developed for the purpose of improving the comfort of the ETS GRIDD Lab. Figure 1 shows three

[1] Groupe de Recherche en Intégration et Développement Durable en environnement bâti. The english translation would be Research Group in Integration and Sustainable Development in the Built Environment.

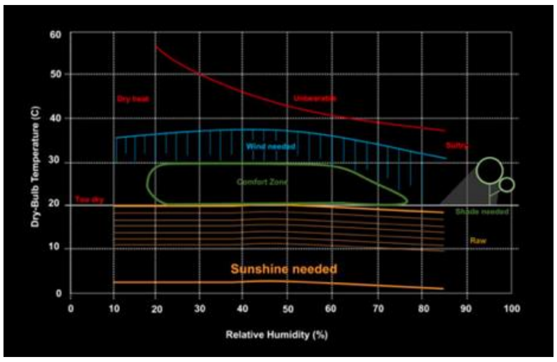
aspects of this DT: a) a 3D virtual view [2] of the GRIDD Lab build based on Building Information Model (BIM) data; b) a PMV (Predicted Mean Vote) (Fanger 1977) thermal comfort chart; and c) a diagram outlining the high-level software architecture of the DT in relation to the GRIDD Lab (AT) and the User Interface that provides access to DT information (or insights).

In the current implementation, the GRIDD Lab is equipped with a network of sensors that collects data related to different aspects of the lab (CO^2, noise level, temperature, and humidity) at regular intervals and stores them in the Sensor Data timeseries database of the DT[3]. Also, a BIM model of the lab is available.

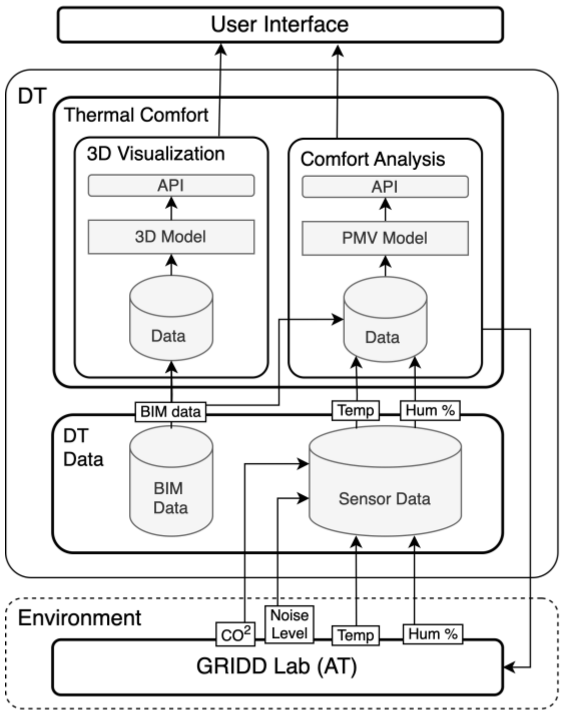
From a software architecture perspective, the DT is composed of a DT Data component and a set of components responsible for delivering the DT services. The DT Data component, which can also be viewed as a data lake, is responsible for storing the AT data and making them available to the different service components. In the example of Figure 1.c, it contains the BIM data of the GRIDD Lab and the data provided by the sensors (Sensor Data) deployed in the lab. From a service perspective, the DT provides two main services related to the improvement of comfort: a 3D Visualization of the lab and its properties related to Thermal Comfort; and a Comfort Analysis service. Each service component is composed of a database, one or more models used to produce the required service based on the available data, and an API that provides external access to the services. The data used by each service is a subset of the data available from the DT Data component, e.g. while the DT Data component contains raw sensor data regarding CO^2, noise level, temperature, and humidity, the Comfort Analysis service only uses temperature, and humidity. Similarly, the BIM data used by each of the service are subsets of the complete BIM Data.



a) 3D virtual view of the GRIDD Lab



b) PMV thermal comfort chart



c) DT software architecture

Figure 1. DT to improve occupant comfort of the GRIDD Lab

2.2 Challenges in the development of DT for built assets

The development of digital twins for built assets involves several layers, each presenting a variety of challenges that must be systematically addressed to ensure successful systems that are not only scalable but also able to keep pace with changing environments and user requirements,

[2] The 3D virtual model was implemented using the Autodesk Revit tool and Autodesk Construction Cloud.
[3] In our current implementation of the DT, this database is deployed in Azure cloud.

while retaining their reliability and efficiency. In this paper, we focus on the software engineering aspects of the development of DTs for build assets . These challenges are indicative of the inherent complexities associated with DTs, highlighting the need for expertise beyond traditional construction knowledge, in favor of a multidisciplinary approach to software engineering. These challenges can be grouped by focusing on data, model and service.

The **data challenges** are related to four main dimensions: acquisition, integration, transmission and security. During the acquisition, the initial focus is on the accurate and efficient collection of data from built assets. The integration of these sensors into existing structures requires careful planning in terms of positioning, powering devices, access for maintenance and vandalism prevention amongst other considerations. Those devices must establish clear and uninterrupted connections with a communication network, complicating integration at the onset, and potentially O&M later on. (Dave et al 2018) Those connections must support potentially large data throughputs, which can put a strain on existing network infrastructures. A lack of expertise, namely related to choosing the right technologies for subsequent phases and meet built asset service life expectations can also constitute a challenge. Once the data is properly transmitted, it must be stored. Towards seamless integration, data interoperability and data fusion must be achieved to ensure that data from a variety of sources (e.g. sensors, systems, external) particularly in a heterogeneous systems environment, are successfully used. (Khajavi et al 2019) It also is critical that all data related matters must be dealt with security and privacy in mind (Shahzad et at 2022).

The **model challenges** concern the creation of a dynamic model that can evolve according to changes in the physical asset and new user requirements. It involves sophisticated architectural considerations to ensure that the model faithfully reflects the built asset. Interoperability issues may need to be addressed again here, alongside the development of standardized classifications and structures. (Shahzad et al 2022)

Closer to users, **service challenges** involve defining services and their requirements. Common challenges revolve around poorly defined user requirements, leading to wasted efforts and delays. Services often suffer from a lack of foresight relating to scalability, maintenance and resource management, resulting in inefficient and inflexible systems.

Finally, beyond those known challenges in the built asset community, many obstacles typically met in software engineering have yet to surface.

2.3 DevOps

Over the last decade, DevOps has emerged as the prominent approach to increase agility, productivity, and system quality in the software industry. It has resulted in spectacular progress regarding key aspects of software development and operations (Forsgren et al 2018). (Kim et al 2021) defines the three ways of DevOps: 1- Flow, 2- Feedback and 3- Continual Learning and Experimentation. DevOps is based on the principle of continuous improvement and focuses on optimizing the flow of activities involved in the creation of end user value, from idea to deployed functionality, and on providing fast and continuous feedback throughout the entire process. It extends the Lean and software agile methodologies by integrating aspects like quality assurance, continuous integration and deployment, and system operations together with development to enable more frequent and reliable releases. It aims to use automation as much as possible to improve productivity, predictability and quality.

Figure 2 illustrates the infinite loop that describes the DevOps workflow that is composed of the following phases: Plan, Code, Build, Test, Release, Deploy, Operate, and Monitor. These phases can be grouped in four main process parts: Planning, which relates to the Plan phase; Dev, which includes the Code, Build, and Test phases; Release & Deployment, which includes the Release and Deploy phases; and Operation & Monitoring, which includes the Operate and Monitor phases.

The last part of the DevOps workflow, Operation & Monitoring, aims at ensuring the proper operation of the DT and the real-time monitoring of different aspects of its execution to enable early detection of issues (before they become a problem) and the identification of aspects that need to be improved. This part of the process rely on DT telemetry to collect the required data. It is important to mention that the Monitor phase provides a main input into the Plan phase and allows closing the DevOps (infinite) loop that supports continuous improvement. Because of the space limitation of the paper, the technical aspects related to the Operate and Monitor phases, including their automation and tool support, will not be further discussed.

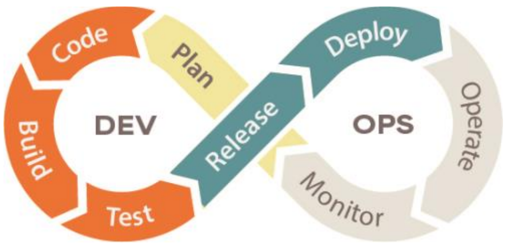


Figure 2. DevOps workflow

2.4 Related work

The development of digital twins has reached an important milestone, evolving from simple 3D models to integrated systems that harness real-time data, predictive analytics and machine learning (Li et al 2024). However, studies in built asset are still emerging and the body of work remains sparse. BIM plays a key role in this progression, improving the accuracy and efficiency of DT applications, marking a shift from traditional practices to advanced, data-driven methodologies. This underscores the importance of the growing development of long-term DTs in this field. Also, with data often at the heart of DT development challenges in the built environment, the lack of structure/framework around it is readily apparent. We have gathered here the most relevant studies to this paper.

The study on the development of a DT at the University of Cambridge (UK) describes a dynamic DT at the building level that integrates various heterogeneous data sources, such as a multi-layered BIM and IoT-based sensors. The system is built on a comprehensive system architecture aimed at "[…] supporting intelligent asset management, providing effective O&M management, and further bridge the gap between human relationships with buildings/cities via more intelligent, visual and sustainable channels." (Qiuchen Lu et al 2019). Despite the system approach, a framework designed to allow continuous updating and improvement to support long-term development is required.

(López-Peña et al 2020) worked on "fast and continuous monitoring feedback of system availability" activity (F&CF availability). This leverages the DevOps contribution to a component of DTs in IoT system from an availability standpoint. This approach materializes through the deployment of distributed components monitoring, which provides the DevOps team with the flexibility to configure specific metrics and indicators at runtime, thereby enabling customized, on-demand monitoring.

(Chamari et al 2023) proposes a service-oriented architecture (SOA) for data-driven applications. It focuses on component modularity and reusability as well as real-time integration of heterogeneous data sources. While this study does not explicitly speak to the evolution of DT over time, the SOA lays the foundation for a systematic evolution of DT.

Regarding DevOps, some papers have discussed the use of its principles for the development, deployment, and operations of DTs (Hugues et al 2020, Ugarte Querejeta et al 2020, and Dobaj et al 2022) and Cyber-Physical Systems (CPS) (Combemale & Wimmer 2020, Mertens & Denil 2022), but, to the best of our knowledge, there exist no DevOps approach or framework to support the systematic development and evolution of DTs.

In summary, the lack of structure in DT development for built assets appears in the various studies on the subject. This would be consistent with the experimental nature of the studies. Furthermore, the studies mostly concentrate on the improvements on single aspects of a DT. A broader and more systematic integration of DevOps aims at addressing the identified challenges and providing support for the systematic and scalable development and evolution DTs.

3 DevOps approach for DT

From an engineering perspective, DTs are generally considered as complex distributed software systems. As such, they need to be designed, developed, tested, deployed, and operated as any complex software system. For this purpose, the development of DTs needs to be well-planned and conducted in an iterative manner to support its systematic evolution and continuous improvement. From a functional perspective, the number of aspects addressed by a DT and the set services it provides need to be incrementally developed.

In this section, we describe a DevOps approach that can be used to support the systematic development and evolution of DTs in the context of build assets. We focus on three aspects: software methodology to support the iterative development of DTs; 2- Microservice architecture to enable the independent development and composition of different aspects of DTs; and 3-DevOps infrastructure to automate the continuous integration and deployment (CI/CD) of DTs.

3.1 Software Methodology for DT

From a software methodology perspective, the overall development process needs to be structured to support the systematic and iterative development and evolution of DTs so that they can be adapted to continuously changing environments, requirements, and needs. Throughout their lifecycle, besides the changes made to correct defects or refactor part of the software to reduce the technical debt, a DT may need to be modified for three main reasons:

- Modifications made to the AT or its environment that need to be reflected in the DT, e.g. structural properties of the AT have been modified, new sensors have been added or a new source of external data has become available.
- Modifications made to the DT to improve the precision of one of its existing services, e.g. a DT can be modified to take advantage of a new source of data or to introduce of a new type of model to improve its current management of a specific aspect (e.g. Thermal Comfort)
- Increase the set of services provided by the DT, e.g. the scope of a DT that was initially developed to improve comfort in a building can be modified to include a new aspect and services like the reduction of energy consumption, the improvement of air quality, or the management of assets.

For this purpose, each iteration needs to be carefully planned in the Plan phase of the DevOps workflow. To support the management of the software development, DevOps promotes the use of agile principles like the use of Kanban boards (or other types of sprint planning boards). Among other things, the planning of an iteration requires the determination of the percentage of resources that will be allocated to the development of new features/services, the improvement of existing ones, the correction of software defects, and the reduction of the technical debt (Kersten 2018). In this phase, the information collected by the DT telemetry in the Monitor phase plays a key role.

In the case of the GRIDD Lab, we started by focusing on the Thermal Comfort aspect, as illustrated in Figure 1. We first developed the 3D Visualization service and then added a Comfort Analysis service using on a PMV model. Each service was independently developed and then integrated in the DT. In a second phase, we increased the scope of the DT by adding an Asset Management aspect, which includes two main services: Equipment Inventory and Room Configuration. Figure 3 provides a high-level architecture diagram of the GRIDD Lab DT software architecture resulting from this new aspect. As the DT continues to evolve, additional aspects such as energy consumption management and air quality improvement will be iteratively incorporated.

3.2 DT Architecture - Microservices

DevOps promotes the use of a microservice architecture to manage the software complexity of DTs. Some existing DTs, while complex, are structured with most of the code running on one machine, gathering and processing data, akin to monolithic style architecture. This is easier to understand but challenging to scale when adding new features. Some others employ distributed systems that use cloud services with some degree of decoupling. While the latter are closer to the proposed architecture, this section examines what they are and how they could positively impact DTs in built assets using GRIDD's Lab example.

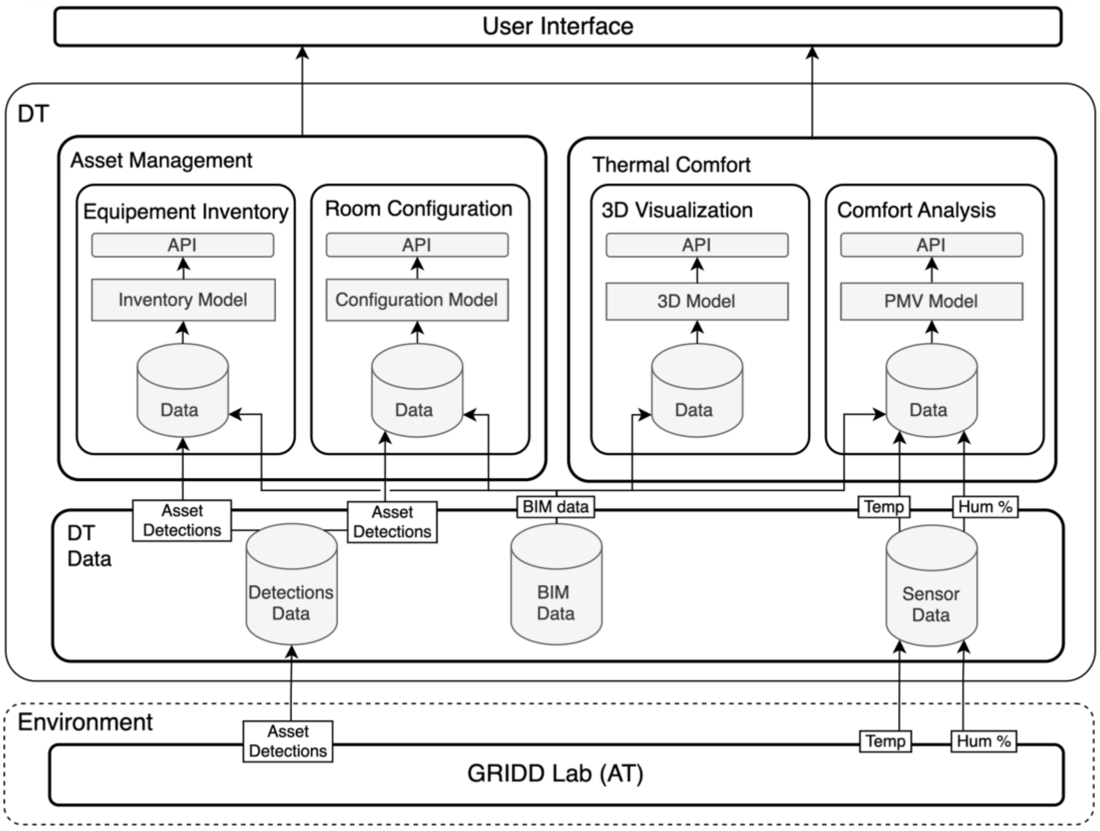


Figure 3. Software architecture of the GRIDD Lab DT after the addition of Asset Management

First, independent deployability is considered. This principle asserts that a microservice can be changed and deployed without affecting other services. Achieving this requires ensuring that microservices are loosely coupled yet maintain stable interfaces, enabling flexibility, faster release cycles, and simplified deployments across diverse platforms and environments. For instance, in Figure 3, Thermal Comfort aspect could be updated to improve performances without impacting Asset Management services. The User Interface would remain unimpacted if the API stays the same. This helps to address model and service challenges.

Second, models are developed around business domains to decompose the DT system into a set of components based on real-world domains they operate in. Each DT is associated with a domain of expertise, one team and one microservice. Splitting a larger DT into smaller components aligns with the Conway Law as domain-specific teams, such as those working on equipment inventory, are unlikely to collaborate with unrelated teams, such as those focused on PMV. Those are two separate domains, heterogenous both in model and data, thus likely to communicate very differently. In our example, it is likely that any aspect added to the GRIDD Lab DT will be by a different research team, following its own objectives and methodologies. Drawing clear boundaries helps organize around specific purposes and goals of a DT by keeping internal cohesion high and external coupling low. This helps to address some data and service challenges.

Third, state ownership is enforced. Each microservice must be able to independently manage its data, maintain its state and control what information is shared, exposing only the information that needs being seen. For instance, a Room Configuration component would store the current state of configuration, allowing it to check against a planned configuration and create work orders to change them. Another component or service could need this information, but would have to request it through the API. As depicted in Figure 3, this means having its own data to facilitate transformation of data while ensuring persistence. This abstraction typically spans several containers (e.g. Docker), each encapsulating everything needed to run a microservice in any environment, regardless of the programming language, thereby standardizing deployment and simplifying the management of various service instances. This helps to address some data challenges.

Finally, by fragmenting a DT into manageable and autonomous services, microservices architecture offers a modularity and flexibility that promotes simpler maintenance, greater scalability and continuous innovation, while responding more effectively to the changing needs of complex environments. DevOps processes and their associated tools thrive in decoupled architectures. Beyond the benefits brought by independent deployability, business domain decomposition, and state ownership, DevOps provides a natural solution to address challenges of DT development in build assets.

3.3 DevOps infrastructure to support the Dev and Release & Deployment phases of DT

A main contribution of the DevOps approach comes from the automation of different aspects of the DT software process. The goal is to streamline and improve the efficiency of the overall software process by automating manual repetitive tasks using different types of software technologies and tools[4]. The elimination of these repetitive tasks, which from a Lean perspective constitutes a waste, allows development teams to focus on value-added activities such as developing new features, improving existing ones, or correcting software defects. The automation of these manual tasks also has the advantage of reducing human error and increasing the consistency of tasks performed, resulting in a significant improvement in the quality of the final product. This not only facilitates the ongoing evolution of the DT, but also enhances its maintainability.

In this paper, we concentrate on three main aspects of DevOps process that are used to reap the full benefits using existing technologies:

- Version control using Git to support the overall DevOps process
- Continuous Integration (CI) to automate the Dev phases
- Continuous Deployment (CD) to automate the Release & Deployment phases

Figure 4 illustrates CI and CD in the context of the overall DevOps process lifecycle.

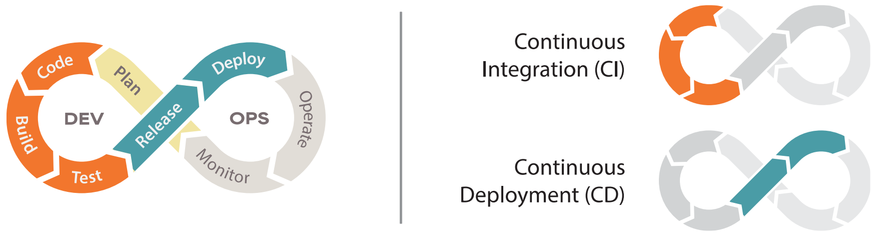


Figure 4. DevOps workflow – CI/CD phases

Version Control

In DevOps, Git[5] is used as a version control system to support efficient team collaboration and provide a single source of truth for all project artifacts, including code, test cases, configuration files, and documentation. It provides the foundation to support the overall DevOps process. It enables precise tracking of every modification made to a file, whether additions or deletions, and offers the flexibility to revert to previous states and compare file changes over time. It also provides support for file diff/merge, which allows teams to collaborate more efficiently on development projects.

Git is particularly relevant for offering robust branching models such as Gitflow, which organizes projects into two primary branches: Development and Master. This model streamlines workflows by allowing developers to work in dedicated development branches, often referred to as feature branches. Such a structured approach not only clarifies the development process, but

[4] A broad range of technologies and tools (open source and proprietary commercial) have been developed over the last two decades to support the different aspects of all phases of the DevOps process.
[5] Git: https://git-scm.com/

also integrates seamlessly into continuous integration and deployment (CI/CD) pipelines, supporting the iterative nature of modern software development.

Furthermore, Git platforms provide a set of basic functionalities to support different aspects of the DevOps process, including mechanisms for peer-review[6] and for the development of CI/CD pipelines[7]. These platforms also facilitate the creation and management of tasks/issues, incorporating different project management tools like Kanban boards to ensure a smooth, organized workflow across all stages.

In our project, the GitHub[8] platform is used to support our DevOps approach and GitHub Actions are used to implement all CI/CD pipelines. This solid infrastructure enables a smooth and secure transition of code from development to production. By properly exploiting Git, teams can maximize the benefits of CI/CD pipelines, taking full advantage of available tools and technologies.

Continuous Integration (CI)

In the Dev phases, different tools and technologies can be used to support and automate various aspects of the software development process. In particular, the CI pipeline automates the execution of the different steps involved in the Code, Build, and Test phases. This automation facilitates the seamless integration of new code into a shared Git repository (on Master on Main branch), where it is continuously tested to identify errors as early as possible to ensure that the latest software version is always in a state that is ready to be released to production.
Figure 5 outlines how each phase of the CI pipeline—Code, Build, and Test—is meticulously thought out to improve the efficiency across various activities, to support and automate various aspects of the software development process.

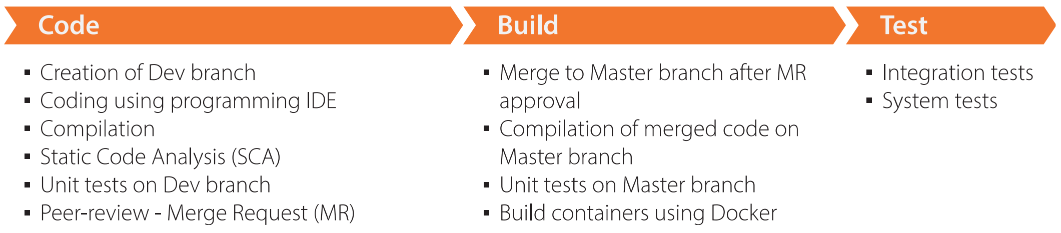

Code
Creation of Dev branch
Coding using programming IDE
Compilation
Static Code Analysis (SCA)
Unit tests on Dev branch
Peer-review - Merge Request (MR)
Build
Merge to Master branch after MR approval
Compilation of merged code on Master branch
Unit tests on Master branch
Build containers using Docker
Test
Integration tests
System tests

Figure 5. Typical CI pipeline tasks

Code: In this phase, software developers perform various tasks associated with the development of new functionalities, improvement of existing ones, or fixing of issues. Different tools[9] are used to support the coding tasks, including code editing (using programming Integrated Development Environments (IDE)), compilation, static code analysis, and unit tests. In this phase, the work is done on a local Git development branch, which allows developers to work independently from each other. The phase concludes with a code peer-review.

Build: The Build phase is triggered once the new code has been accepted by the reviewer. It starts with the integration (merge) of the new code on the Git Master branch. During this step, Git ensures that the code can be integrated and that no conflict arise with the existing code. Once the new code is integrated, the unit tests are executed on the Master branch and a new version of the microservices (affected by the code change) is build (e.g. as a Docker container). Finally, the new candidate version of the software is packaged and made available for testing.

Test: As soon as a new candidate version is made available (by the Build phase), the different integration and system test phases are automatically executed. The testing is conducted in a pre-production environment. The goal is to identify issues before the new version of the software is deployed in the pro-duction environment. In the context of DT for build assets, this typically

[6] For example, Pull Requests in GitHub and Bitbucket, and Merge Requests in GitLab.
[7] For example, GitHub Actions and GitLab CI.
[8] GitHub: https://github.com/
[9] Different programming languages are used for different components, including C++, Java, and Python. ESlint (eslint.org) and SonaQube (www.sonarsource.com) are the main tools used for static code analysis.

includes testing on specific devices and environments (e.g. IoT device or cloud environment) depending on the component (microservice).

In our projects, we use both Azure and AWS for different cloud services (e.g. Azure for hosting the main database that stores the data collected from the different sensors deployed in the room and AWS for the deployment of different microservices), and a combination of Arduino, Raspberry Pi, and Jetson devices for the deployment of different data acquisition components and edge-microservices that perform local treatment of data. Although the automation of deployment on cloud environments is nowadays common, the deployment on IoT devices is still often performed manually. As a result, a deployment pipeline needs to be developed for each of the environments on which components/containers are deployed, this includes Azure, AWS, Arduino, Raspberry Pi, and Jetson devices.

Continuous Deployment (CD)
In the Release & Deployment phases, CD pipelines automate the deployment of verified code in the production environment, while maintaining the stability of the DT system, thus ensuring rapid and secure updates. In a fully automated DevOps process, CD pipelines can be triggered whenever the product manager is ready to release a new version of the DT software in production. By effectively implementing CI pipelines and leveraging a microservices architecture, the transition to CD occurs naturally. The CI ensures a reliable source of code changes, setting the stages for these changes to be automatically deployed anytime through a streamlined process. This process uses Docker containers to provide an isolated and consistent environment for each component, making it easier to deploy, scale, and maintain the application across different operational environments—from development to production. CD pipelines enable a seamless transition from development to production, crucial in heterogeneous environments where both cloud infrastructure and edge devices, such as IoT devices, are used. This deployment flexibility is essential for maintaining the real-time operational integrity and scalability of DT systems, particularly in complex built asset management scenarios.

4 Conclusion

This paper proposes a DevOps approach to support the systematic development and evolution of DTs for built assets. The proposed approach is described in terms of (1) a software methodology to support the systematic and iterative development of DTs, (2) microservice architecture to enable the independent development and composition of different aspects of DTs, (3) DevOps infrastructure to support the development, integration, and deployment of DTs. Regarding the DevOps infrastructure, we focused on three main aspects: version control using Git to provide a single source of truth and support team collaboration, Continuous Integration (CI) to support the Dev phases, and Continuous Deployment CD to support the Release & Deployment phases.

The paper illustrates the approach using the example of the GRIDD Lab, effectively showcasing how DevOps can support iterative development, and enhance the capability of DTs to provide valuable insights and services such as comfort management and energy efficiency.

Besides providing a solid foundation for the development and operations of scalable DTs, the use of DevOps also allows addressing several of the main challenges associated with the development of DTs for build assets as described in section 2.2.

In conclusion, the proposed DevOps approach marks a significant shift towards more adaptive DT systems in the built environment. It not only fosters a culture of continuous improvement and collaboration but also sets a foundation for future research and development in this area. The development of a DevOps framework for DT to support the approach proposed in this paper is currently underway and will be published and released as open source in the coming months. The proposed approach continues to evolve as we work on the ongoing evolution of the GRIDD Lab DT and the development of new DTs.

Acknowledgements

This research was funded in part by the Natural Sciences and Engineering Research Council of Canada.

References

Bolton, A., Butler, L., Dabson, I., Enzer, M., Evans, M., Fenemore, T., Harradence, F., Keaney, E., Kemp, A., Luck, A. and Pawsey, N. (2018). Gemini principles. https://doi.org/10.17863/CAM.32260.

Combemale, B. and Wimmer, M. (2020). Towards a model-based devops for cyber-physical systems. In Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment: Second International Workshop, DEVOPS 2019, Château de Villebrumier, France, May 6–8, 2019, Revised Selected Papers 2 (pp. 84-94). Springer International Publishing.

Camposano, J. C., Smolander, K., & Ruippo, T. (2021). Seven metaphors to understand digital twins of built assets. IEEE Access, 9, 27167-27181. https://doi.org/10.1109/ACCESS.2021.3058009

Chamari, L., Petrova, E. and Pauwels, P. (2023). An end-to-end implementation of a service-oriented architecture for data-driven smart buildings. IEEE Access.

Chenari, B., Dias Carrilho, J., & Gameiro da Silva, M. (2016). Towards sustainable, energy-efficient and healthy ventilation strategies in buildings: A review. Renewable and Sustainable Energy Reviews, 59, 1426-1447. https://doi.org/10.1016/j.rser.2016.01.074

Dobaj, J., Riel, A., Krug, T., Seidl, M., Macher, G., & Egretzberger, M. (2022). Towards digital twin-enabled DevOps for CPS providing architecture-based service adaptation & verification at runtime. In Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems (pp. 132-143).

Dobaj, J., Riel, A., Macher, G. & Egretzberger, M. (2023). Towards DevOps for Cyber-Physical Systems (CPSs): Resilient Self-Adaptive Software for Sustainable Human-Centric Smart CPS Facilitated by Digital Twins. Machines, 11(10), p.973.

Dave, B., Buda A., Nurminen, A., and Främling, K., (2018). "A framework for integrating BIM and IoT through open standards," Automation in Construction, vol. 95, pp. 35–45, doi: 10.1016/j.autcon.2018.07.022.

Eramo R., Bordeleau F., Combemale B., et al. (2021). Conceptualizing digital twins. IEEE Software 39(2):39–46.

Fanger, P. O., Breum, N. O., & Jerking, E. (1977). Can colour and noise influence man's thermal comfort Ergonomics?, 20(1), 11-18.

Forsgren, N., Humble, J. and Kim, G. (2018). Accelerate: The science of lean software and devops: Building and scaling high performing technology organizations. IT Revolution.

Grieves, M., & Vickers, J. (2017). Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In Springer International Publishing (pp. 85-113). https://doi.org/10.1007/978-3-319-38756-7_4

Halhoul Merabet, G., et al. (2021). Intelligent building control systems for thermal comfort and energy-efficiency: A systematic review of artificial intelligence-assisted techniques. Renewable and Sustainable Energy Reviews, 144, 110969. https://doi.org/10.1016/j.rser.2021.110969

Hugues, J., Hristosov, A., Hudak, J.J. & Yankel, J. (2020). TwinOps-DevOps meets model-based engineering and digital twins for the engineering of CPS. In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (pp. 1-5).

Jones, D., Snider, C., Nassehi, A., Yon, J., & Hicks, B. (2020). Characterising the Digital Twin: A systematic literature review. CIRP Journal of Manufacturing Science and Technology, 29, 36–52. https://doi.org/10.1016/j.cirpj.2020.02.002

Kim G., Humble J., Debois P., et al. (2021). The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations. IT Revolution.

Kersten, M. (2018). Project to product: How to survive and thrive in the age of digital disruption with the flow framework. IT Revolution.

Kritzinger W, Karner M, Traar G, et al (2018). Digital Twin in manufacturing: A categorical literature review and classification. Ifac-PapersOnline 51(11):1016–1022.

Khajavi, S. H. Motlagh, N. H. , Jaribion, A., Werner, L. C. , and Holmström, J. (2019). "Digital Twin: Vision, Benefits, Boundaries, and Creation for Buildings," IEEE Access, vol. 7, pp. 147406–147419, doi: 10.1109/ACCESS.2019.2946515.

Li W., Zhou H., Lu Z., & Kamarthi S. (2024). Navigating the Evolution of Digital Twins Research through Keyword Co-Occurence Network Analysis. Sensors, 24(4), p. 1202. Available at: https://doi.org/10.3390/s24041202.

Lo, C. K., Chen, C. H., & Zhong, R. Y. (2021). A review of digital twin in product design and development. Advanced Engineering Informatics, 48, 101297. https://doi.org/10.1016/j.aei.2021.101297

López-Peña, M. A., Díaz, J., Pérez, J. E., & Humanes, H. (2020). DevOps for IoT systems: Fast and continuous monitoring feedback of system availability. IEEE Internet of Things Journal, 7(10), 10695-10707.

Lu, Q., Vivi, et al. (2019). Developing a dynamic digital twin at a building level: Using Cambridge campus as case study. International Conference on Smart Infrastructure and Construction 2019 (ICSIC) Driving data-informed decision-making. ICE Publishing, 2019.

Mertens, J., & Denil, J. (2020). Digital twins for continuous deployment in model-based systems engineering of cyber-physical systems. In CEUR workshop proceedings (Vol. 2822, pp. 32-39).

Mertens, J., & Denil, J. (2021). The digital twin as a common knowledge base in devops to support continuous system evolution. In International Conference on Computer Safety, Reliability, and Security (pp. 158-170). Cham: Springer International Publishing.

Sharafdin, P. (2024). Building occupant comfort monitoring using IoT-based sensor data integrated into Digital Twin. M.Sc. Thesis, Ecole de technologie superieure (ETS), Montreal (Canada).

Shahzad, M., Shafiq, M.T., Douglas, D. and Kassem, M. (2022). Digital twins in built environments: an investigation of the characteristics, applications, and challenges. Buildings, 12(2), p.120.

Tao, F., Zhang, H., Liu, A., & Nee, A. Y. (2018). Digital twin in industry: State-of-the-art. IEEE Transactions on industrial informatics, 15(4), 2405-2415.

Ugarte Querejeta, M., Etxeberria, L. and Sagardui, G. (2020). September. Towards a devops approach in cyber physical production systems using digital twins. In International Conference on Computer Safety, Reliability, and Security (pp. 205-216). Cham: Springer International Publishing.