Decentralizing Infrastructure as Code

Daniel Sokolowski
University of St. Gallen

Pascal Weisenburger
University of St. Gallen

Guido Salvaneschi
University of St. Gallen

***Abstract*—Infrastructure as Code (IaC) automates deployments for single teams, falling short of decentralized deployments across groups. We need mature IaC solutions that embrace and consolidate software engineering principles to enable testing and automation advances for decentralized organizations.**

This preprint is accepted for publication in the IEEE Software magazine as: Daniel Sokolowski, Pascal Weisenburger, and Guido Salvaneschi, "Decentralizing Infrastructure as Code," in IEEE Software, vol 40, no. 01, 2023, doi: https://doi.org/10.1109/MS.2022.3192968.

Index Terms: DevOps, Infrastructure as Code, Cloud

Sidebar 1: Actionable Insights.

- For software deployment and undeployment, dependencies across teams and organizations should be taken into account because they often constrain the order of operations.
- Deployment coordination across teams, which is often manual, should be automated for better software delivery and operational performance.
- Future decentralized infrastructure-as-code technology should leverage explicit inter-deployment interfaces for better testing and automation.

SOFTWARE MUST ADAPT quickly to changing business requirements while ensuring stability and robustness. The past two decades have seen various IT approaches aiming at these goals, e.g., Unified Process, Scrum, or Extreme Programming, converging into DevOps [1]. The key focus of DevOps is to enable the frequent development of software updates and ensure reliable software operations. The objectives of DevOps are commonly measured through the software delivery and operational (SDO) performance metrics (cf. Sidebar 2). In contrast to outdated concepts, studies on the state of DevOps show that, in practice, throughput metrics (e.g., higher deployment frequency) correlate positively with service stability metrics (e.g., lower change failure rate) [2]. As the base philosophy of modern IT organizations, DevOps inspired a range of practices with additional focus and insights, e.g., GitOps, MLOps, and DevSecOps.

The premise for good SDO performance is a high degree of automation along the whole software pipeline [1]. In practice, Infrastructure as Code (IaC) [3] automates application deployments and plays a key role in DevOps organizations. Modern IaC solutions compare the present infrastructure with the desired state and automatically derive the required deployment actions to move the infrastructure into that state. In last-generation IaC solutions, i.e., Pulumi, Amazon Web Services Cloud Development Kit (CDK), and Terraform CDK, the desired state is defined in a general-purpose programming language, e.g., TypeScript, Python, C#, or Go.

Such IaC scripts are amenable to well-known software engineering techniques, including versioning and testing, ensuring robust and repeatable deployments. Adopting these methods for infrastructure provisioning and application deployment has become more and more relevant because system complexity is being moved from inside software components into their composition. Traditional monolithic applications only have a few separately deployed components, while modern, serverless equivalents consist of tens or hundreds of smaller components. For instance, a monolithic webshop could be a single web service and a database. In contrast, a serverless webshop typically is split up into various services, including authentication, order, and shipment, each including a dedicated database and multiple serverless functions.

State of the DevOps Vision

DevOps encourages cross-functional teams, which contrasts the previous silo-based approach with separate teams for, e.g., development, operations, and testing [1]. Such teams combine competence, interest, and responsibility for a single application, demolishing the so-called wall of confusion between teams and preventing unclear, shared responsibilities. Within a cross-functional team, an application is jointly designed, developed, tested, and operated, reducing friction and enabling fast feedback between activities.

Building teams around applications—not roles—aims at team independence. Applications, however, are usually not fully isolated but interact with other applications they depend on. In our webshop example, the order service requires the authentication service to verify the user's permission to view or place an order. Do such dependencies constrain the order of the applications' (un)deployments? For instance, must the authentication service be deployed before the order service because of the dependency? If this is the case, how do the responsible teams maintain their independence as prescribed by DevOps? Today's IaC solutions either require centralizing deployments, hindering team independence, or force out-of-band coordination. An external tool is needed or manual coordination via phone, chat, and mail is required, coupling the deployment times synchronously. Is the lack of automated coordination of deployments across teams a real problem in DevOps organizations?

Sidebar 2: Software Delivery and Operational Performance.

The *(Accelerate) State of DevOps* reports[1] are an annual study, begun in 2014, of the application and trends of DevOps. Their authors developed a widely accepted instrument to measure the maturity of DevOps organizations, consisting of the following correlating key metrics:

Throughput

The *deployment frequency* measures how often code is deployed to production.

The *lead time for changes* measures how long it takes until committed code is deployed to production.

Stability

The *time to restore service* measures the required time to restore service after an incident happens.

The *change failure rate* measures the ratio of failed changes to production.

In recent reports, these key metrics are accompanied by a fifth one measuring availability and, since 2021, reliability.

[1] https://www.devops-research.com/research.html#reports

The Dependencies in DevOps Survey

Our survey of 134 IT professionals sheds light on these questions. We now provide detailed insight into the results, extending the presentation of preliminary results in our previous work [4] with better precision and reduced error margin. The full report is publicly available [5]. Figure 1 includes the demographics and broad spectrum of participants from different companies that we acquired through snowball sampling in our networks and social media.

Figure 2a shows that for 89 % of the respondents, the primary application requires another application to provide full functionality; 87 % of the participants state, to different extents, that such dependencies may constrain the order of the applications' deployments (Figure 2b). For undeployment, more than two-thirds confirm that dependencies may constrain the order of undeployment. Accordingly, dependencies among applications are ubiquitous and often constrain the order in which applications can be safely (un)deployed.

We asked the participants how they coordinate the order of (un)deployment operations across independent teams in their organization. Figure 2c illustrates that only one-fourth of the respondents coordinate in a fully automated way; 76 % rely on manual coordination, e.g., via phone, chat, and mail. We further asked whether they expect worse, similar, or better SDO performance for solely manual and fully automated coordination compared to the case where no coordination is needed. We expected worse or similar performance for both cases because any coordination introduces overhead. However, 51 % of the respondents expect better SDO performance for automated coordination. From our perspective, this is an unjustified belief, but it shows how much practitioners value automation. In Figure 2d, we compare, for each SDO performance metric, whether automated coordination is expected to yield better performance than manual coordination. For each metric, the majority of respondents expect better performance. In summary, IT professionals expect better SDO performance from automated coordination, but manual coordination is still the norm.

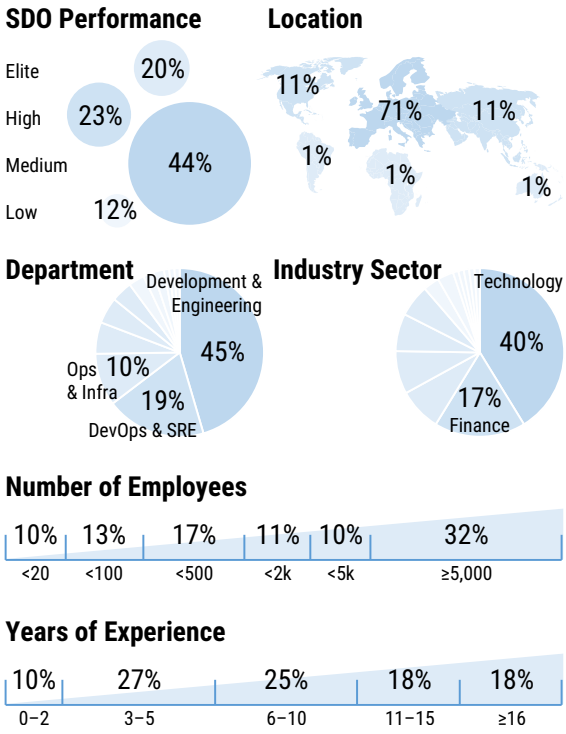


Figure 1: The demographics of the survey respondents and the organization they work in.

This mismatch is remarkable. Apparently, there is no simple, widely adopted solution to coordinate deployments automatically. Hence, we asked which tools the respondents would use for coordination. Their answers confirm our impression that current IaC solutions cannot automate the coordination: the respondents named various chat and CI/CD platforms as well as custom scripts, but no one mentioned an IaC solution, even though IaC is the designated tool for deployment automation. This circumstance raises the question: What is missing in today's IaC solutions for decentralized teams?

Bridging the Gap: The Future of Decentralized IaC

Current IaC solutions are effective for centralized deployments. However, automating deployment coordination across decentralized teams requires coordinating configuration and the points in time when teams perform their (un)deployments. Teams should not have to interact synchronously but operate independently and on-demand. Ultimately, new features or bug

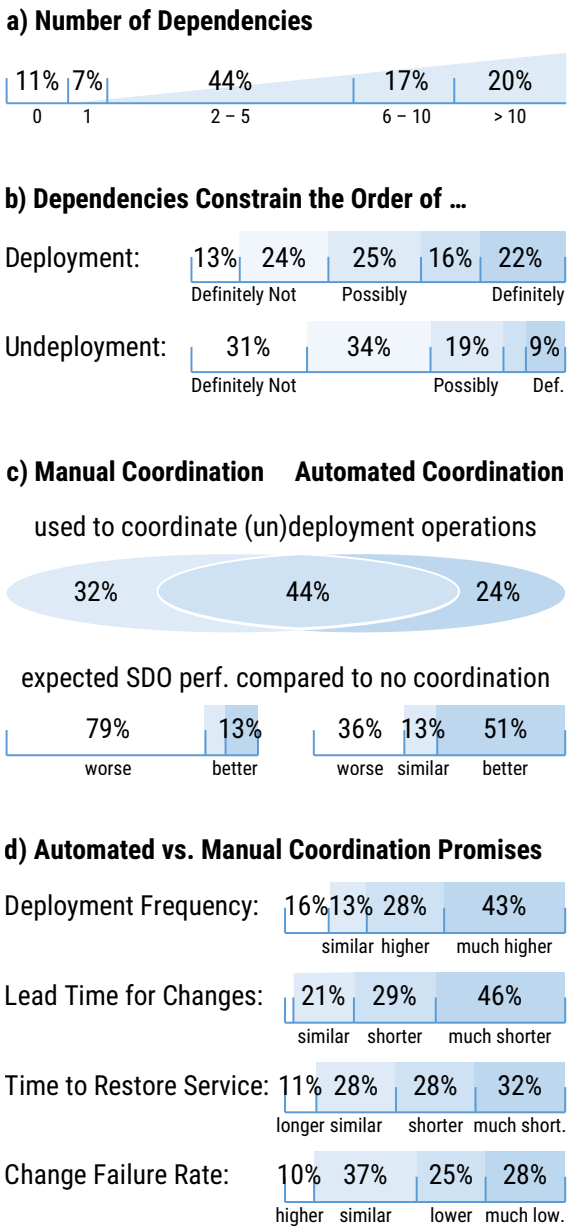


Figure 2: Applications usually depend on one another, and such dependencies constrain the order of (un)deployment operations. Manual deployment coordination is common, but automated coordination promises better SDO performance.

fixes may be developed anytime, requiring continuous development *and* deployment. We argue that strong interfaces and operational decoupling solve this issue. Their absence truly limits current IaC solutions in decentralized organizations.

Strong Interfaces

In current IaC solutions, direct interaction between deployments is only supported via (cross-)stack references. Such references are dynamically typed and enable access to the exported configuration of other deployments. In the webshop example, the authentication service may export its IP address and port as an object `{ "address": "10.1.2.43", "port": 83 }`. The order service deployment, which depends on the authentication service, reads the exported object from a stack reference and uses its values. Nevertheless, the order deployment fails if the authentication service is not deployed or the format is changed, e.g., `"address"` is renamed `"host"`. Hence, deploying a service can fail due to another service's deployment that is controlled by another team. Stack references do not enable detecting that deployment will fail before starting it. Thus, they are ultimately unsafe and do not help coordinate deployments.

Instead, interfaces between deployments should be explicit and, ideally, cover all aspects relevant to other deployments. For example, the order deployment should explicitly state that the authentication service must be deployed before and that it needs its IP address and port. Symmetrically, the authentication deployment should offer its service with the IP address and port. Describing the interfaces in both the consuming and the providing deployments—not only on one side—enables leveraging methods from (consumer-driven) contract design and testing. Teams can use mocking and simulation techniques to test their deployments against connected deployments before integration. Further, safe evolution of interfaces between teams can be facilitated once all relevant information is explicitly encoded.

Operational Decoupling

Another aspect that hinders decentralized IaC is that current IaC solutions treat deployments as one-off tasks that start, execute once, and

terminate. This behavior carries the design-time dependencies between the teams' deployments to deployment time. Each (un)deployment requires highly coupled, synchronously coordinated operations between teams. For instance, before each service deployment, the order team has to ensure through synchronous communication that the authentication team has already deployed its service.

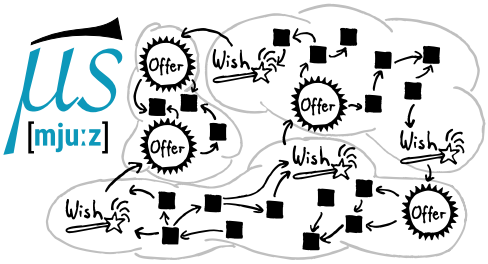
To overcome this limitation, IaC deployments should run continuously and deploy resources reactively. Deployments should be long-running processes that communicate with each other. The order team, for example, could start its deployment without requiring synchronization with the authentication team. However, the order service remains undeployed as long as its dependency on the authentication service is unsatisfied. Once the authentication team starts its deployment, the order team's deployment is notified and automatically deploys the order service. Such continuous, reactive deployment behavior can further be leveraged to automate safe undeployment and information exchange across deployments.

Our work on μS (cf. Sidebar 3) demonstrates that strong interfaces and operational decoupling enable automated decentralized deployment coordination. Still, there is huge untapped potential for such IaC techniques for decentralized organizations to truly improve team independence, even if the deployments depend on each other. The gained automation and safety will boost the teams' productivity and the SDO performance of entire organizations.

Acknowledgments

This work is supported by the Swiss National Science Foundation (under grant 200429), the Hessian Initiative for the Development of Scientific Economic Excellence (through the Software-Factory 4.0 project), and the University of St. Gallen (through the International Postdoctoral Fellowship 1031569). This work involved human subjects or animals in its research. The authors confirm that all human/animal subject research procedures and protocols are exempt from review board approval.

Sidebar 3: Automated Decentralized Deployment Coordination with μS.



There is a declarative infrastructure-as-code solution, μS ([mju:z] *"muse"*) [4], in which independent teams explicitly define their interfaces to other teams' deployments. Teams define an offer resource to provide values and resources to a specific remote deployment. In the consuming deployments, teams define a wish resource to explicitly define the types of the values and resources they expect from a remote deployment. Together, offers and wishes define a contract expressing the assumptions about the deployments' connection on the providing and consuming sides.
In μS, deployments are continuously running processes, not one-off tasks like, e.g., in Pulumi. Thanks to the information encoded in wishes and offers, it is possible to automate the coordination across deployments of independent teams. In particular, μS ensures the following:

1) A wish and dependent resources are deployed after the corresponding offer.
2) A wish and dependent resources are automatically undeployed before the corresponding offer.
3) Changes in an offer are reactively propagated to the corresponding wishes.

Further, offers and wishes enable better testing prior to deployment. Before a deployment update, teams can check whether the new version is compatible with the wishes and offers of the connected deployments. A prototype of μS based on Pulumi TypeScript is available at https://mjuz.rocks.

REFERENCES

1. C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “Devops,” *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016. [Online]. Available: https://doi.org/10.1109/MS.2016.68
2. N. Forsgren, D. Smith, J. Humble, and J. Frazelle, “2019 accelerate state of DevOps report,” https://services.google.com/fh/files/misc/state-of-devops-2019.pdf, last accessed on 2022-04-07.
3. K. Morris, *Infrastructure as Code: Dynamic Systems for the Cloud Age*, 2nd ed. O’Reilly Media, Inc., 2021.
4. D. Sokolowski, P. Weisenburger, and G. Salvaneschi, “Automating serverless deployments for DevOps organizations,” in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. ACM, 2021, pp. 57–69. [Online]. Available: https://doi.org/10.1145/3468264.3468575
5. ——, “Dependencies in DevOps survey 2021: Version 2.0 (until april 15, 2021).” [Online]. Available: https://doi.org/10.5281/zenodo.6372120

Daniel Sokolowski is a Ph.D. candidate in the Programming Group, University of St. Gallen, 9000 St. Gallen, Switzerland. His research interests include software engineering and programming languages for modern distributed systems, i.e., modern infrastructure as code for cloud-native applications. Sokolowski received his M.Sc. from the Technical University of Darmstadt, Germany, and is a member of the ACM. Contact him at https://dsoko.de or daniel.sokolowski@unisg.ch.

Pascal Weisenburger is a postdoctoral researcher in the Programming Group, University of St. Gallen, 9000 St. Gallen, Switzerland. His research interests include the design of languages for distributed systems with sound programming models, i.e., for safe interaction, composition, privacy, and placement. Weisenburger received his Ph.D. from the Technical University of Darmstadt, Germany, and is a member of the ACM. Contact him at pascal.weisenburger@unisg.ch.

Guido Salvaneschi is an associate professor of programming at the University of St. Gallen, 9000 St. Gallen, Switzerland. His research interests include programming languages and software engineering, including languages and architectures for distributed systems, reactive programming, and secure software systems. Salvaneschi received his Ph.D. from Politecnico di Milano, Italy, and is a member of the ACM. Contact him at guido.salvaneschi@unisg.ch.