Editor in Chief: **Ipek Ozkaya**
Carnegie Mellon Software Engineering Institute
ipek.ozkaya@computer.org

Infrastructure as Code and Software Architecture Conformance Checking

Ipek Ozkaya

INFRASTRUCTURE AS CODE (IaC) is the practice to automatically configure system dependencies and create deployment specifications to enable the orchestration of any system mostly running in the cloud. IaC is an established approach to support DevOps, the set of practices to enable software developers and operations teams to accelerate the delivery of software through automation, collaboration, fast feedback, and iterative improvement. IaC focuses on automating the processes needed to manage the deployment and configuration infrastructure that otherwise would rest on the shoulders of system administrators and, by doing so, aims to create a single source of truth for the deployment structure of the system.

The infrastructure that needs to be managed by IaC techniques include physical servers as well as virtual machines, databases, and all the related configuration resources. Any tool using a programmatic approach, including continuous configuration automation tools, which assist with defining and executing infrastructure configurations and frameworks, often is considered IaC. Such tooling and scripting allow developers and operations roles to contribute to the goal of managing infrastructure elements with automation collaboratively. Developers define configurations, identifying dependencies among software elements, and addressing how to orchestrate runtime and development time software structure concerns. Operations teams get involved in the software development, deployment, and operations process earlier along with developers, bringing visibility to the state of the servers, their specifications, and enterprise-level requirements and constraints.

There has been a significant amount of attention paid to researching the

The infrastructure that needs to be managed by IaC techniques include physical servers as well as virtual machines, databases, and all the related configuration resources.

Digital Object Identifier 10.1109/MS.2022.3213880
Date of current version: 23 December 2022

automated tool support aspect of IaC. There are quite a number of existing frameworks and tools provided by the dominant cloud providers to support development and operations teams so that they can take advantage of the speed through automation vision of IaC. These IaC tools aim to make it easier to deploy the system in the corresponding vendor's cloud platform. Therefore, it is not surprising to see that the systematic mapping study of IaC research conducted by Rahman et al. identified the framework and tools for IaC as the top area with significant research attention. Other areas where IaC research has mostly focused on include adoption of IaC, empirical studies related to IaC, and testing in IaC, where many of the studies either propose a new framework or tool for IaC or extend a new one.[1]

While most research emphasis is on developing tools for IaC, these tools do not necessarily assure that the code developed to enable the automation of the infrastructure setup and configuration is not without issues itself. Jiang and Adams, in their 2015 study, demonstrated that projects they studied had a substantial number of IaC files, hinting at a need to study how they evolve along with source code, test code, and built scripts.[2] In addition to its large size, the study identified that code realizing IaC churns significantly more often than test and build files. Furthermore, the study found that infrastructure code files are coupled tightly with other files in a project, especially test files, indicating an increased likelihood of change propagation and bugs in IaC, similar to source code.

Existing research studies demonstrate that IaC occupies a significant and critical part of the overall software ecosystem. IaC represents a substantial amount of code that needs to align with the rest of the source, test, and build code to avoid unintended rework and technical debt, even within IaC.[2,3]

While importance of the alignment of IaC with other implementation elements is recognized, its potential in contributing to improving the structure and behavior of the software through realizing the deployment concerns of its architecture is less taken advantage of.[4]

The Allocation Structures

In defining the software architecture of a system, there are three dominant types of structures: the module structures, the component and connector structures, and the allocation structures. The module structure assists software engineering teams with reasoning about the implementation elements and their design and development time relationships. The component and connector structures focus on the runtime behavior of software elements and are used to reason about behavior, such as how resources are shared, whether they are available when needed, and how secure they are at runtime. Allocation structures describe the mapping of software elements defined through module and component and connector structures to the environment elements where the software executes. Naturally, the most dominant concern in allocation is the deployment, the servers, the databases, the virtual machines, and the infrastructure that provides runtime support and how the system is structured to align with the environment that it will execute within. Software architects, when designing the system, are responsible

CONTACT US

AUTHORS
For detailed information on submitting articles, visit the "Write for Us" section at www.computer.org/software

LETTERS TO THE EDITOR
Send letters to software@computer.org

ON THE WEB
www.computer.org/software

SUBSCRIBE
www.computer.org/subscribe

SUBSCRIPTION CHANGE OF ADDRESS
address.change@ieee.org
(please specify *IEEE Software.*)

MEMBERSHIP CHANGE OF ADDRESS
member.services@ieee.org

MISSING OR DAMAGED COPIES
contactcenter@ieee.org

REPRINT PERMISSION
IEEE utilizes Rightslink for permissions requests. For more information, visit www.ieee.org/publications/rights/rights-link.html

Digital Object Identifier 10.1109/MS.2022.3223719

for ensuring that the alignment of development, build, staging, and production environments is consistent and playing from the same sheet of music.[5] IaC tools and frameworks provide an opportunity to provide that enforcement.

Provisioning and infrastructure consistency as a desired outcome of DevOps starts by understanding the expected response measures and service-level agreements of system functionality. The software architecture of the system describes and prescribes the development time, runtime, deployment time, and operational concerns. The responsibility of IaC tools is to provide automation that takes the best advantage of the infrastructure available in meeting some of these concerns, especially when unexpected changes occur.

IaC assists with development time expectations; for example, the ability to run security and user acceptance testing without roadblocks at scale and to identify build dependencies that may create inconsistencies during deployment is achieved more consistently with IaC. Making the deployment structure explicit—what you get from the scripts for deployment tools—can make enforcement and conformance much easier. IaC enables conformance to a well-defined runtime and deployment architecture by managing the ad hoc configuration changes, installation and configuration of operating system software, and connection to middleware, networks, storage, servers, and the like.

The software architecture of a system has the goal of defining the fundamental structures of a system through the software elements, relationships among them, and properties of both elements and relations. IaC, as a corollary, has the goal of automating the process of supplying the resources where these elements will run. Since an increasing number of systems run in the cloud, concerns, such as on-demand resource access, including network and storage, resource pooling, elasticity to scale up and down, and being aware of the resources and optimizing for them, are both design time as well as deployment and operation time concerns. The allocation structures provide the keystone, where IaC tooling not only serves the goal of infrastructure scaling in the cloud but can also enable conformance to a well-defined system.

Architecture Conformance Is Not Up-Front Design

Creating a reliable software architecture conformance checking tool has been difficult to implement because the architecture is typically buried in the code. To some, architecture conformance implies making the most of design decisions up front and exposing these design concerns. Conformance suggests compliance and governance, the scrutiny of the design, code, and other system artifacts against established architectural criteria and business objectives and, at the enterprise level, adherence to rules and guidance to ensure that the available resources are utilized appropriately.[6] The spirit of both activities is to ensure high-quality systems that meet their business and user goals, where resources are spent effectively and systems are designed to be easily adaptable to change. Without effective tool support, architecture conformance becomes a difficult goal to achieve, especially as technologies evolve and code evolves along with them.

INTRODUCING THE "DEVELOPER PRODUCTIVITY FOR HUMANS" COLUMN

With this first issue of 2023, we are launching a column dedicated to featuring the research and practices that support developer well-being and productivity. We are calling this the "Developer Productivity for Humans" column. The title of the column has the goal of enforcing that we are opposed to treating developers as "interchangeable cogs in a machine," as expressed by Ciera Jaspan, one of the coeditors of the column. Ciera and her coeditor, Collin Green, with this column, will reinforce that software engineers and developers are human and that productivity tools should support making their jobs easier as opposed to turning practitioners into productivity machines. This is the primary lens they use in their day job at Google when understanding, measuring, and improving developer productivity.

Ciera Jaspan is the tech lead manager of the Engineering Productivity Research team within Core Developer at Google, where she uses a data-driven mixed-methods approach to drive tool, process, and culture decisions made by Google leadership. The team's infrastructure, metrics, and research results are used to motivate changes to Google's developer tools that will increase productivity and then to measure the impact of these changes to developer productivity across Google. She previously worked on Tricorder, Google's static analysis platform. She received her B.S. in software engineering from California Polytechnic State University and her Ph.D. from Carnegie Mellon University, where she worked with Jonathan Aldrich on cost-effective static analysis and software framework design.

Collin Green is a user experience researcher and manager of the Engineering Productivity Research team within Core Developer at Google. His research focuses on applying combined quantitative and qualitative behavioral research methods to understand developer experience and engineering productivity. In prior roles, he has studied the design and usability of software tools for technical users in medicine and aerospace and the impacts those tools have on productivity. Green received his Ph.D. in psychology from the University of California, Los Angeles.

Ciera and Collin bring an interdisciplinary perspective to this important subject. They will feature their experiences through this column as well as those of others who have successfully empowered the improved productivity and well-being of developers in their organizations. I am very excited to be introducing this column. Ciera, Collin, and myself as well as the rest of the board at *IEEE Software* all believe there is much work to be done in this area. We would like to contribute to progress by featuring what has been accomplished so far while shining a light on the remaining challenges for the software engineering community to work on. We are looking forward to your feedback as well as contributions in this very important topic area.

IaC assists with infrastructure resource management. IaC can at least supply one set of inputs by automating the change management of systems from the server, network, virtual machine, database, and any other operational infrastructure perspective of the system. These resources are critical both during the development and operation of systems. The nimble way the high-level descriptive languages and scripts of IaC are designed allows room for introducing conformance to quality, resource utilization, and change management goals of the system. If we envision architecture conformance not as a single tool but as a set of information that needs to be continuously collected and checked for, IaC tools and frameworks readily provide part of the software system design allocation structure information needed for conformance. Improved collective ownership of the structure and behavior of systems; improved alignment of system artifacts, including the design of the system; and improved management of runtime and deployment time resources and architecturally significant requirements of the system are essential for both architecture conformance and configuration and infrastructure flexibility that IaC aspires to. Responsibility sits on the shoulders of both roles. Software architecture and engineering teams need to take advantage of IaC as an approach to enforce the allocation structures of the systems. DevOps teams need to think about the system holistically, including the alignment of IaC code with not only source code, test code, and build code but also with the architecturally significant requirements of the system and the design that realized them.

References

1. A. Rahman, R. Mahdavi-Hezaveh, and L. Williams, "A systematic mapping study of infrastructure as code research," *Inf. Softw. Technol.*, vol. 108, pp. 65–77, Apr. 2019, doi: 10.1016/j.infsof.2018.12.004.
2. Y. Jiang and B. Adams, "Co-evolution of infrastructure and source code–An empirical study," in *Proc. 2015 IEEE/ACM 12th Working Conf. Mining Softw. Repositories*, pp. 45–55, doi: 10.1109/MSR.2015.12.
3. P. Kruchten, R. Nord, and I. Ozkaya, *Managing Technical Debt: Reducing Friction in Software Development.* Reading, MA, USA: Addison -Wesley, 2019.
4. L. Bass, I. Weber, and Z. Liming, *DevOps: A Software Architect's Perspective*. Reading, MA, USA: Addison-Wesley, 2015.
5. L. Bass, "The software architect and DevOps," *IEEE Softw.*, vol. 35, no. 1, pp. 8–10, Jan./Feb. 2018, doi: 10.1109/MS.2017.4541051.
6. "The Open Group Architecture Framework," TOGAF™, version 8.1.1., 2006. Accessed: Oct. 2022. [Online]. Available: https://pubs.opengroup.org/architecture/togaf8-doc/arch/toc-pt4.html

Digital Object Identifier 10.1109/MS.2022.3223188