Report from Dagstuhl Seminar 23082

Resilient Software Configuration and Infrastructure Code Analysis

Jürgen Cito*[1], Ruzica Piskac*[2], Mark Santolucito*[3], Andy Zaidman*[4], and Daniel Sokolowski†[5]

1 **TU Wien, AT.** `juergen.cito@tuwien.ac.at`
2 **Yale University – New Haven, US.** `ruzica.piskac@yale.edu`
3 **Barnard College – New York, US.** `msantolu@barnard.edu`
4 **TU Delft, NL.** `a.e.zaidman@tudelft.nl`
5 **Universität St. Gallen, CH.** `daniel.sokolowski@unisg.ch`

Abstract

Errors originating from infrastructure and their configurations are one of the major causes of system failures and system degradation, resulting in security vulnerabilities, application outages, and incorrect program executions. Investigating the root causes of such issues and remedies for them requires insight from different research perspectives, including systems, programming languages, software engineering, and verification. To facilitate progress in this field, this Dagstuhl Seminar brought together experts from academia and industry, enabling synergies between different software systems subareas. The seminar was a forum for cross-disciplinary discussions, bridged communities, and forged new conversations on new approaches. Emerging themes that were revealed during the seminar included a focus on Infrastructure as Code, the similarities and differences between configuration engineering and software engineering, the portability (or lack thereof) of program analysis techniques to configuration analysis, the design space of expressibility of configuration languages, and future challenges of analysis for safety, security, and auditing. The seminar led to new short-term and long-term collaborations and connections, including organizing additional workshops and a joint vision paper.

Seminar February 19–24, 2023 – https://www.dagstuhl.de/23082
2012 ACM Subject Classification Security and privacy → Operating systems security; Software and its engineering → Parallel programming languages; Software and its engineering → Compilers; Software and its engineering
Keywords and phrases Computing infrastructure, Configuration, Program analysis
Digital Object Identifier 10.4230/DagRep.13.2.163

1 Executive Summary

Jürgen Cito (TU Wien, AT)
Ruzica Piskac (Yale University – New Haven, US)
Mark Santolucito (Barnard College – New York, US)
Andy Zaidman (TU Delft, NL)

Errors originating from infrastructure and their configurations are one of the major causes of system failures and system degradation, resulting in security vulnerabilities, application outages, and incorrect program executions. Investigating the root causes of such issues and

* Editor / Organizer
† Editorial Assistant / Collector

remedies for them requires insight from different research perspectives, including systems, programming languages, software engineering, and verification. From these areas, approaches are emerging to manage the complexity of infrastructure and configuration, covering a breadth of forms, such as domain-specific languages, standalone verification tools, automated learning techniques, specification-based synthesis, security annotation extensions, and configuration optimizers.

The Dagstuhl Seminar on Resilient Software Configuration and Infrastructure Code Analysis brought together experts from different fields to explore new cross-disciplinary approaches to configuration management. The seminar facilitated collaboration between academia and industry and enabled synergies between different subareas of software systems. The seminar was a forum for cross-disciplinary discussions, bridged communities, and forged new conversations between academic and industrial perspectives. The shared knowledge built during the seminar is captured in this report, which we hope can act as a body of knowledge for researchers joining this newly forming community.

Overall, the seminar consisted of 3 tutorial talks, 16 presentations, and 5 group discussions. Emerging themes that were revealed during the seminar included a focus on Infrastructure as Code, the similarities and differences between configuration engineering and software engineering, the portability (or lack thereof) of program analysis techniques to configuration analysis, the design space of expressibility of configuration languages, and future challenges of analysis for safety, security, and auditing. In addition, we had a joint evening session with the parallel seminar "Agents on the Web" (Dagstuhl Seminar 23081), where each organizer presented an overview of their seminar. As a result, we started joint discussions where we investigated the use of formal methods, in particular synthesis, for establishing semantic relations between the data.

Key outcomes of this seminar were evident both in new short-term collaboration and connections, as well as the initiation of longer-term projects. For example, a collection of the participants have connected to host a workshop on configuration languages and analysis called CONFLANG 2023, which will be hosted at SPLASH 2023. Additionally, a vision paper outlining key future directions of the field is being drafted by participants of the event.

After two postponements due to COVID-19, this seminar was a pleasure to hold in person and a great success from both a community and a research perspective. We would like to thank the team of Schloss Dagstuhl for their hospitality and support as well as all the participants for their valuable contributions.

2 Table of Contents

3 Overview of Talks

3.1 Improving Infrastructure Security by Analyzing Pre-Deployment Artifacts

Claudia Cauli (Amazon Web Services – London, GB)

Joint work of Claudia Cauli, Meng Li, Nir Piterman, Oksana Tkachuk
Main reference Claudia Cauli, Meng Li, Nir Piterman, Oksana Tkachuk: Pre-deployment Security Assessment for Cloud Services Through Semantic Reasoning. CAV (1) 2021: 767-780
URL https://link.springer.com/chapter/10.1007/978-3-030-81685-8_36

Over the past ten years, the adoption of cloud services has grown rapidly, leading to the introduction of automated deployment tools to address the scale and complexity of the infrastructure companies and users deploy. Without the aid of automation, ensuring the security of an ever-increasing number of deployments becomes more and more challenging. To the best of our knowledge, no formal automated technique currently exists to verify cloud deployments during the design phase. In this case study, we show that Description Logic modeling and inference capabilities can be used to improve the safety of cloud configurations. We focus on the Amazon Web Services (AWS) proprietary declarative language, CloudFormation, and develop a tool to encode template files into logic. We query the resulting models with properties related to security posture and report on our findings. By extending the models with dataflow-specific knowledge, we use more comprehensive semantic reasoning to further support security reviews. When applying the developed toolchain to publicly available deployment files, we find numerous violations of widely-recognized security best practices, which suggests that streamlining the methodologies developed for this case study would be beneficial.

3.2 GLITCH: Automated Polyglot Code Smell Detection in Infrastructure as Code

João F. Ferreira (INESC-ID – Lisboa, PT)

Joint work of João F. Ferreira, Nuno Saavedra
Main reference Nuno Saavedra, João F. Ferreira: "GLITCH: Automated Polyglot Security Smell Detection in Infrastructure as Code", in Proc. of the 37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022, pp. 47:1–47:12, ACM, 2022.
URL https://doi.org//10.1145/3551349.3556945

Infrastructure as Code (IaC) is the process of managing IT infrastructure via programmable configuration files (also called IaC scripts). Like other software artifacts, IaC scripts may contain code smells, which are coding patterns that can result in weaknesses. Automated analysis tools to detect code smells in IaC scripts exist, but they focus on specific technologies such as Puppet, Ansible, or Chef. This means that when the detection of a new smell is implemented in one of the tools, it is not immediately available for the technologies supported by the other tools – the only option is to duplicate the effort.

We present GLITCH, a technology-agnostic framework that enables the automated detection of code smells in IaC scripts. GLITCH allows polyglot smell detection by transforming IaC scripts into an intermediate representation on which different smell detectors can be defined. GLITCH currently supports the detection of nine security smells and nine design & implementation smells. We compare GLITCH with state-of-the-art smell detectors. For security smells, the results show that GLITCH can reduce the effort of writing security smell analyses for multiple IaC technologies and that it obtains higher precision and recall than the current state-of-the-art tools. For the design & implementation smells, we show that GLITCH has enough information in its intermediate representation to detect technology-agnostic smells supported by state-of-the-art tools.

3.3 Correctness and Fault Tolerance of Kubernetes Operators

Tianyin Xu (University of Illinois – Urbana-Champaign, US)

Main reference Xudong Sun, Wenqing Luo, Jiawei Tyler Gu, Aishwarya Ganesan, Ramnatthan Alagappan, Michael Gasch, Lalith Suresh, Tianyin Xu: "Automatic Reliability Testing For Cluster Management Controllers", in Proc. of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), pp. 143–159, USENIX Association, 2022.
URL https://www.usenix.org/conference/osdi22/presentation/sun

Modern cluster managers like Borg, Omega, and Kubernetes rely on the state-reconciliation principle to be highly resilient and extensible. In these systems, all cluster-management logic is embedded in a loosely coupled collection of microservices called controllers. Each controller independently observes the current cluster state and issues corrective actions to converge the cluster to a desired state. However, the complex distributed nature of the overall system makes it hard to build reliable and correct controllers – we find that controllers face myriad reliability issues that lead to severe consequences like data loss, security vulnerabilities, and resource leaks.

In this talk, I present Sieve, the first automatic reliability-testing tool for cluster-management controllers. Sieve drives controllers to their potentially buggy corners by systematically and extensively perturbing the controller's view of the current cluster state in ways it is expected to tolerate. It then compares the cluster state's evolution with and without perturbations to detect safety and liveness issues. Sieve's design is powered by a fundamental opportunity in state-reconciliation systems – these systems are based on state-centric interfaces between the controllers and the cluster state; such interfaces are highly transparent and enable fully-automated reliability testing. To date, Sieve has efficiently found 46 serious safety and liveness bugs (35 confirmed and 22 fixed) in ten popular controllers with a low false-positive rate of 3.5%.

3.4 Configuration Validation and Testing for Cloud Systems: Research and Practice

Tianyin Xu (University of Illinois – Urbana-Champaign, US)

Main reference Shuai Wang, Xinyu Lian, Darko Marinov, and Tianyin Xu: "Test Selection for Unified Regression Testing", In Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE'23), Melbourne, Australia, May 2023.
URL https://tianyin.github.io/pub/ctest.pdf

Configuration management is an integral part of modern DevOps-based cloud system management. Many critical operations are done by updating configurations to change system behavior in production dynamically. Today, large-scale cloud and Internet services evolve rapidly, with hundreds to thousands of configuration changes deployed daily. For example, at Facebook, thousands of configuration changes are committed every day, outpacing the frequency of code changes. Other cloud services from Google and Azure also frequently deploy configuration changes. It is not surprising to hear that the "cloud feels more about configuration management than software engineering."

With the high velocity of changes, faulty configurations inevitably have become major causes of system failures and service outages. For example, faulty configurations are reported as the second largest cause of service disruptions in a main Google production service. At Facebook, 16% of service-level incidents are induced by configuration changes. Many configuration-induced failures led to catastrophic impacts. For instance, in March 2019, a misconfiguration led to Facebook's largest outage in terms of duration (14 hours); in June 2021, a seemingly-valid configuration change at Fastly triggered an undiscovered software bug and broke the Internet for an hour.

We argue that continuous testing is a key missing piece of today's configuration management practice. Despite the "configuration-as-code" movement, there is no widely-used, systematic configuration testing technique, and thus configuration changes are not unit-tested—imagining a world where code changes only go through manual review and static analysis, without regression testing.

We will introduce the idea of configuration testing, a new testing technique that enables configuration changes to be unit-tested in DevOps-based continuous integration/deployment. The basic idea of configuration testing is connecting system configurations to software tests so that configuration changes can be tested in the context of code affected by the changes. We will introduce a new type of tests, termed Ctests, to fill the critical need for configuration testing. Ctests complement static validation (the de facto protection), analogous to how testing complements static analysis:

- Ctests can detect failure-inducing configuration changes where the failure root causes are in the code, e.g., valid configuration value changes that trigger dormant bugs.
- Ctests can detect sophisticated misconfigurations (e.g., those that violate undocumented, hidden constraints) by capturing the resulting unexpected system behavior.

We will demonstrate that ctests can be generated by transforming existing software tests that are abundant in mature software projects. The generated ctests selectively inherit test logic and assertions from the original tests. The inherited assertions hold for all correct configuration values. We have successfully generated 7,974 ctests for five widely-used open-source cloud systems (Hadoop Common, HDFS, HBase, ZooKeeper, and Alluxio).

We will show that the generated ctests are effective and outperform state-of-the-art static configuration validation techniques based on extensive evaluations with real-world failure cases, synthesized misconfigurations, and deployed configuration files in public Docker images.

References

1 Runxiang Cheng, Lingming Zhang, Darko Marinov, and Tianyin Xu, Test-Case Prioritization for Configuration Testing, In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'21), Virtual Event, July 2021.

2 Yuanliang Zhang, Haochen He, Owolabi Legunsen, Shanshan Li, Wei Dong, and Tianyin Xu, An Evolutionary Study of Configuration Design and Implementation in Cloud Systeums, In Proceedings of the 43rd International Conference on Software Engineering (ICSE'21), Virtual Event, May 2021.

3 Xudong Sun, Runxiang Cheng, Jianyan Chen, Elaine Ang, Owolabi Legunsen, and Tianyin Xu, Testing Configuration Changes in Context to Prevent Production Failures, In Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20), Virtual Event, Nov. 2020.

4 Qingrong Chen, Teng Wang, Owolabi Legunsen, Shanshan Li, and Tianyin Xu, Understanding and Discovering Software Configuration Dependencies in Cloud and Datacenter Systems, In Proceedings of the 2020 ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'20), Virtual Event, Nov. 2020.

5 Tianyin Xu and Darko Marinov, Mining Container Image Repositories for Software Configurations and Beyond, In Proceedings of the 40th International Conference on Software Engineering, New Ideas and Emerging Results (ICSE'18, NIER), Gothenburg, Sweden, May 2018.

6 Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy, Early Detection of Configuration Errors to Reduce Failure Damage, In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16), Savannah, GA, Nov. 2016.

7 Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker, Hey, You Have Given Me Too Many Knobs! Understanding and Dealing with Over-Designed Configuration in System Software, In Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'15), Bergamo, Italy, Aug. 2015.

8 Tianyin Xu and Yuanyuan Zhou, Systems Approaches to Tackling Configuration Errors: A Survey, ACM Computing Surveys (CSUR), Vol. 47, No. 4, Article 70, Jul. 2015.

9 Tianyin Xu, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, and Shankar Pasupathy, Do Not Blame Users for Misconfigrations, In Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP'13), Farmington, PA, Nov. 2013.

3.5 Configurations Here and There, Configurations Everywhere

Myra B. Cohen (Iowa State University – Ames, US)

Joint work of Paul Gazzillo, Myra B. Cohen
Main reference Paul Gazzillo, Myra B. Cohen: "Bringing Together Configuration Research: Towards a Common Ground", in Proc. of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Onward! 2022, Auckland, New Zealand, December 8-10, 2022, pp. 259–269, ACM, 2022.
URL https://doi.org//10.1145/3563835.3568737

Configurable software makes up most of the software in use today. Configurability, i.e., the ability of software to be customized without additional programming, is pervasive, and due to the criticality of problems caused by misconfiguration, it has been an active topic researched by investigators in multiple, diverse areas. This broad reach of configurability means that much of the literature and latest results are dispersed, and researchers may not be collaborating or be aware of similar problems and solutions in other domains. We argue that this lack of a common ground leads to a missed opportunity for synergy between research domains and the synthesis of efforts to tackle configurability problems. To provide a foundation for addressing these concerns, we suggest how to bring the communities together and propose a common model of configurability and a platform, ACCORD, to facilitate collaboration among researchers and practitioners.

3.6 Correct and Modular Configuration with Nickel

Yann Hamdaoui (Tweag I/O – Paris, FR)

From a distance, Infrastructure as Code should really be called Infrastructure as Configuration. DevOps, SRE and other engineers dealing with infrastructure are mostly managing pure configuration data by writing, editing and auditing JSON, YAML or similar serialization formats.

Purely data-oriented languages like JSON might be fine for managing small and simple infrastructure, but when the size and complexity of a configuration grows, data languages don't seem to be the right tool anymore.

There is no way to reuse data and have a single source of truth, with all the pain and inconsistencies that duplication inevitably brings with time. We can't express data dependencies either. For example, the open ports of a firewall may depend on which services are enabled on a server. In JSON, everything must be hardcoded.

Infrastructure tools have thus incorporated programming features in their languages (Terraform, Ansible, Puppet, etc.). But those are often unplanned and unprincipled, resulting in a complex system.

In the end, Configuration Management looks like the poor sibling of Software Engineering. What about tests, types, LSP integration for real-time feedback, completion, and documentation? What about modularity, code reuse, and abstraction?

In this talk, I will present Nickel, a configuration programming language I am currently working on at Tweag, to help finally enter the era of Configuration as Code. I'll discuss more specifically the approach of Nickel to modularity, which is how to write small and reusable configuration snippets that can be combined into a complex configuration based on a merging operation. I'll talk about type-checking and built-in schema validation as well.

3.7 The Theory of Real-life Small and Large Configurations

Marcel Van Lohuizen (CUE – Zug, CH)

Configuration is in more places than people imagine. Every part of your tech stack – databases, apps, schemas, services, workflows, policy, models, and networking must be configured. Not only that, there are dependencies between these configurations. With cloud, multi-cloud, IoT, and edge computing, the number of things to configure within growing systems has exploded. The consequences of getting the configuration wrong have only worsened over time.

We are in the middle of a major shift where configuration is becoming a first-class citizen across your stack. New engineering roles dedicated to solving configuration problems have emerged: Platform, Site Reliability, Resilience, Observability, Data, "DevOps" and "YAML" engineers all deal with easing configuration toil. There are many tools and approaches that aim to help developers deal with growing configuration complexity: Configuration Management, Infrastructure as Code, GitOps, Policy as Code, and finally, Infrastructure as Data. The industry needs something to unify all these roles, approaches, and challenges and break down configuration silos. CUE (`cuelang.org`) does exactly that.

In this presentation, you will hear:

- hard-won insights and experiences of configuration at scale, culminating in the design of CUE,
- how configuration can go wrong,
- the need for testing and validation,
- how CUE is paving the way for a holistic approach to configuration via a language, tooling, and APIs that support a vibrant configuration ecosystem.

3.8 Using CUE to Model Configuration

Marcel Van Lohuizen (CUE – Zug, CH)

Configuration is inherently cross-cutting. Combing configuration, therefore, requires commutative and associative composition. In this tutorial, we show how to use CUE to model all configuration aspects of a distributed system into a single space. Configuration aspects can be data, schema, policy, validation, and templates, or any combination thereof.

3.9 The Do's and Don'ts of Infrastructure Code: A Systematic Gray Literature Review

Dario Di Nucci (University of Salerno, IT)

Joint work of Indika Kumara, Martin Garriga, Angel Urbano Romeu, Dario Di Nucci, Fabio Palomba, Damian Andrew Tamburri, Willem-Jan van den Heuvel
Main reference Indika Kumara, Martin Garriga, Angel Urbano Romeu, Dario Di Nucci, Fabio Palomba, Damian Andrew Tamburri, Willem-Jan van den Heuvel: "The do's and don'ts of infrastructure code: A systematic gray literature review", Inf. Softw. Technol., Vol. 137, p. 106593, 2021.
URL https://doi.org//10.1016/j.infsof.2021.106593

This talk provided an overview of the qualitative analysis we conducted to summarize the industrial gray literature (e.g., blog posts, tutorials, white papers) on IaC.

In particular, it provided a general definition of IaC and a broad catalog outlined in taxonomy consisting of ten and four primary categories for best practices and bad practices, respectively, both language-agnostic and language-specific ones, for three IaC languages of best and bad practices for widely used IaC languages (i.e., Ansible, Puppet, and Chef).

Our findings highlighted that the IaC development and maintenance field is in its infancy and deserves further attention.

3.10 Automotive (and Some Other) Configuration Problems

Wolfgang Küchlin (Universität Tübingen, DE)

Joint work of Wolfgang Küchlin, Carsten Sinz
Main reference Wolfgang Küchlin, Carsten Sinz: "Proving Consistency Assertions for Automotive Product Data Management", J. Autom. Reason., Vol. 24(1/2), pp. 145–163, 2000.
URL https://doi.org//10.1023/A:1006370506164

We report on industrial configuration problems outside the field of infrastructure as code. We focus on our work on automotive configuration which goes back to the 1990s and continues until today by providing software to the industry in the context of our Steinbeis Transfer Centre STZ OIT.

Automotive configuration is managed on two levels using (a flavor of) Boolean Algebra. Every equipment or sales option is represented by a Boolean variable x, where x=true represents the presence of the option in a car, and x=false represents the absence, such that a valid car order is given by a complete valuation of all variables. The upper level, model description, comprises the configuration rules for an entire model line of cars (e.g. Mercedes C-Class). The lower level, the Bill-of-Materials, contains the list of all materials needed for the model line. Each material also has a selection formula, and the material is needed to produce a car order if the selection formula evaluates to true under the order.

Problems on the upper level include the computation of forced options, which must be contained in any car, and impossible options, which cannot be contained in any car. Problems on the lower level include the detection of car orders which will select alternative materials or will fail to select a necessary material. These problems are efficiently solved by modern CDCL SAT-Solvers, although the typical variance of the model description is practically infinite (more than a trillion).

Beyond the detection of configuration errors, we proceed to configuration optimization. As an example, we describe the problem of computing the configuration of optimal (minimal) sets of test vehicles on which a given set of testing demands can be carried out.

In addition, we report on some other industrial configuration verification problems whose solutions we published in the past. This includes the detection of misconfigurations of Storage Area Networks (SAN) and of Apache Webservers, the analysis of LINUX Kernel configurations, and the analysis of the completeness of the online help system of a line of computer tomographs.

3.11 Your Shell Reasoning Toolkit

Michael Greenberg (Stevens Institute of Technology – Hoboken, US)

Main reference Michael Greenberg, Austin J. Blatt: "Executable Formal Semantics for the POSIX Shell", Proc. ACM Program. Lang., Vol. 4(POPL), Association for Computing Machinery, 2019.
URL https://doi.org//10.1145/3371111
Main reference Konstantinos Kallas, Tammam Mustafa, Jan Bielak, Dimitris Karnikis, Thurston H.Y. Dang, Michael Greenberg, Nikos Vasilakis: "Practically Correct, Just-in-Time Shell Script Parallelization", in Proc. of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), pp. 769–785, USENIX Association, 2022.
URL https://www.usenix.org/conference/osdi22/presentation/kallas

The shell is a critical part of modern software operations. I have three tools for helping you work with the shell:

- libdash, bindings to the dash parser in OCaml and Python
- smoosh, an executable small-step operational semantics tested against the POSIX spec
- pash-annotations, specifications of common commands

3.12 IaC for Architectural Reconstruction

Davide Taibi (University of Oulu, FI)

Main reference Tomás Cerný, Amr S. Abdelfattah, Vincent Bushong, Abdullah Al Maruf, Davide Taibi: "Microservice Architecture Reconstruction and Visualization Techniques: A Review", in Proc. of the IEEE International Conference on Service-Oriented System Engineering, SOSE 2022, Newark, CA, USA, August 15-18, 2022, pp. 39–48, IEEE, 2022.
URL https://doi.org//10.1109/SOSE55356.2022.00011

The continuous development of new services and the time pressure imposed by the development of new features commonly result in the introduction of non-optimal and temporary solutions. As a result, the software architecture is commonly not compliant with the originally designed one, and nobody has a complete view of the whole system. Static and dynamic analysis methods can help to reconstruct the architecture of the system.

In this talk, we showed some possible techniques for reconstructing the architecture from different perspectives: the infrastructural, the structure of the system analyzed statically or dynamically, and the structure of the organization.

Last we presented a roadmap of possible shared research work towards the development or the extension of tools for IaC architectural reconstruction.

3.13 Decentralizing Infrastructure as Code

Daniel Sokolowski (Universität St. Gallen, CH)

Joint work of Daniel Sokolowski, Pascal Weisenburger, Guido Salvaneschi
Main reference Daniel Sokolowski, Pascal Weisenburger, Guido Salvaneschi: "Decentralizing Infrastructure as Code", IEEE Softw., Vol. 40(1), pp. 50–55, 2023.
URL https://doi.org//10.1109/MS.2022.3192968
Main reference Daniel Sokolowski, Pascal Weisenburger, Guido Salvaneschi: "Automating serverless deployments for DevOps organizations", in Proc. of the ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021, pp. 57–69, ACM, 2021.
URL https://doi.org//10.1145/3468264.3468575

DevOps unifies software development and operations in cross-functional teams to improve software delivery and operations (SDO) performance. Ideally, cross-functional DevOps teams independently deploy their services, but the correct operation of a service often demands other services, requiring coordination to ensure the correct deployment order. This issue is currently solved either with a central deployment or manual out-of-band communication across teams, e.g., via phone, chat, or email. Unfortunately, both contradict the independence of teams, hindering SDO performance – the reason why DevOps is adopted in the first place.

We conducted a study on 134 IT professionals, showing that, in practice, they resort to manual coordination for correct deployments even if they expect better SDO performance with fully automated approaches. We find that Infrastructure as Code (IaC) automates deployments for single teams, falling short of decentralized deployments across groups. To enable testing and automation advances for decentralized organizations, we need mature IaC solutions that embrace and consolidate software engineering principles.

To address this issue, we proposed μs ([mju:z] "muse"), a novel IaC system automating deployment coordination in a fully decentralized fashion, still retaining compatibility with DevOps practice – in contrast to today's solutions. We implement µs, demonstrate that it effectively enables automated coordination, introduces negligible definition overhead, has no performance overhead, and is broadly applicable, as shown by the migration of 64 third-party IaC projects.

3.14 Answer Set Programming: The Powerhouse Technology You've Never Heard Of

Michael Greenberg (Stevens Institute of Technology – Hoboken, US)

Main reference Aaron Bembenek, Michael Greenberg, Stephen Chong: "From SMT to ASP: Solver-Based Approaches to Solving Datalog Synthesis-as-Rule-Selection Problems", Proc. ACM Program. Lang., Vol. 7(POPL), pp. 185–217, 2023.
URL https://doi.org//10.1145/3571200

Answer set programming (ASP) is a powerful tool that folks in PL/FM/SE do not know about. I give two case studies applying ASP to synthesis (once for Datalog, once for RBAC policies); I also explain how ASP relates to SAT (its parent) and SMT (its sibling).

3.15 Transposing Static Analyses from Application to Infrastructure Code: the Curious Case of Ansible

Ruben Opdebeeck (VU – Brussels, BE) and Coen De Roover (VU – Brussels, BE)

Joint work of Ruben Opdebeeck, Jonas De Bleser, Coen De Roover, Dario Di Nucci, Camilo Velázquez-Rodríguez, Ahmed Zerouali
Main reference Ruben Opdebeeck, Ahmed Zerouali, Coen De Roover: "Smelly Variables in Ansible Infrastructure Code: Detection, Prevalence, and Lifetime", in Proc. of the 19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022, pp. 61–72, ACM, 2022.
URL https://doi.org//10.1145/3524842.3527964

This talk presents our journey and experiences in transposing static analyses from application to infrastructure code.

We realized the need to analyze infrastructure code while designing the Chaokka automated tester. Chaokka injects cloud-specific faults during the execution of a test suite to assess the resilience of a cloud-native application against these faults. The approach uses delta debugging to speed up the exploration of the corresponding fault space, which we expect to benefit further from architectural insights extracted from the supporting infrastructure code (e.g., how many replicas the application maintains of each service, ...).

The second part of the talk focuses on analyses for infrastructure code in general and Ansible in particular. To this end, we present two static representations of Ansible code. First, we describe a structural model akin to an AST and its applications in empirical research on Semantic Versioning in the Ansible Galaxy ecosystem. Second, we present a program dependence graph representation that captures an Ansible script's control and data flow. We describe the challenges faced when building such graphs caused by unconventional and undocumented Ansible semantics. Finally, we describe two applications of Ansible program dependence graphs, namely the detection of variable-related smells and a graph pattern mining approach to defect detection.

3.16 Application, Orchestation, and Infrastructure, Oh My! Cross Layer Static Analysis Strategies

Mark Santolucito (Barnard College – New York, US)

With Infrastructure as Code (Iac), the process of configuring complex infrastructure has been simplified and made more accessible to developers. However, with the new expressive power of IaC and the consequentially more complex cloud infrastructure deployments, understanding this complexity has emerged as an unsolved issue. In particular, estimating the cost of an Infrastructure as Code deployment requires understanding the pricing models of every cloud resource being used, as well as an understanding of the interactions between the resources. Existing work either relies on historical usage metrics to predict cost (which has limited utility for new deployments), or on coarse-grain static cost analysis that ignores interactions between resources. We pose as a challenge to the community the need for fine-grained static cost analysis techniques for IaC that incorporate reasoning about the interactions between resources. We walk through a motivating example of this problem that demonstrates the need for such analysis as well as the complexity of the problem.

3.17 TOSCA Explained!

Damian Andrew Tamburri (TU Eindhoven, NL)

Joint work of Paul Lipton, Derek Palma, Matt Rutkowski, Damian A. Tamburri
Main reference Paul Lipton, Derek Palma, Matt Rutkowski, Damian A. Tamburri: "TOSCA Solves Big Problems in the Cloud and Beyond!", IEEE Cloud Computing, Vol. 5(2), pp. 37–47, 2018.
URL https://doi.org//10.1109/MCC.2018.022171666

The anatomy of infrastructure code, as well as all elements and abstractions necessary in writing and maintaining that blueprint, are addressed, among other IaC formats, in the industrial standard for IaC, namely, the OASIS "Topology and Orchestration Specification for Cloud Applications" (TOSCA) industrial standard adopted by as many as 60+ big industrial players worldwide.

Paraphrasing from the standard specification itself, "TOSCA [...] uses [...] service templates to describe cloud apps as a topology template, [...]. TOSCA provides a type system of node types to describe the possible building blocks for constructing a service template and relationship type to describe possible relations". TOSCA-ready orchestrators such as Cloudify or Apache Brooklyn normally come with a considerable number of reusable TOSCA nodes (e.g., a MySQL DB or a WordPress host), while more are proliferating in both research and practice.

Although several alternatives to TOSCA exist (e.g., HashiCorp Terraform, Ubuntu Juju), for the sake of simplicity, we focus on outlining the anatomy of infrastructure code using the technology-agnostic TOSCA standard notation and its intended levels of abstraction.

3.18 Static Detection of Silent Misconfigurations with Deep Interaction Analysis

Ruzica Piskac (Yale University – New Haven, US)

Joint work of Jialu Zhang, Ruzica Piskac, Ennan Zhai, Tianyin Xu
Main reference Jialu Zhang, Ruzica Piskac, Ennan Zhai, Tianyin Xu: "Static detection of silent misconfigurations with deep interaction analysis", Proc. ACM Program. Lang., Vol. 5(OOPSLA), pp. 1–30, 2021.
URL https://doi.org//10.1145/3485517

The behavior of large systems is guided by their configurations: users set parameters in the configuration file to dictate which corresponding part of the system code is executed. However, it is often the case that, although some parameters are set in the configuration file, they do not influence the system runtime behavior, thus failing to meet the user's intent. Moreover, such misconfigurations rarely lead to an error message or raising an exception. We introduce the notion of silent misconfiguration, which are prohibitively hard to identify due to (1) lack of feedback and (2) complex interactions between configurations and code.

In this talk we present ConfigX, the first tool for the detection of silent misconfigurations. The main challenge is understanding the complex interactions between configurations and the code they affected. Our goal is to derive a specification describing non-trivial interactions between the configuration parameters that lead to silent misconfigurations. To this end, ConfigX uses static analysis to determine which parts of the system code are associated with configuration parameters. ConfigX then infers the connections between configuration parameters by analyzing their associated code blocks. We design customized control- and data-flow analysis to derive a specification of configurations. Additionally, we conduct reachability analysis to eliminate spurious rules to reduce false positives. Upon evaluation on five real-world datasets across three widely-used systems, Apache, vsftpd, and PostgreSQL, ConfigX detected more than 2200 silent misconfigurations. We additionally conducted a user study where we ran ConfigX on misconfigurations reported on user forums by real-world users. ConfigX easily detected issues and suggested repairs for those misconfigurations. Our solutions were accepted and confirmed in the interaction with the users who originally posted the problems.

3.19 Generating Infrastructure Code from System Interactions

Jürgen Cito (TU Wien, AT)

Setting up complex, large-scale infrastructure is an iterative process that includes installing dependencies and adjusting parameters within a vast space of configuration options and eventual infrastructure testing in a trial-and-error fashion. The result of this process serves as the underlying base for the deployment and execution of computation defined in the program code. This ad-hoc style of infrastructure setup cannot scale to operations with scalable compute instances (i.e., cloud computing). Infrastructure as code allows to express all infrastructure concerns in the form of code, enabling automation to achieve scalability, transparency, and reproducibility. However, creating code for infrastructure inherently differs from writing program code. The fast feedback cycles required for the iterative setup process outlined above are not feasible when infrastructure code is written in the same way as program code. Thus, infrastructure code is commonly written retrospectively in a cumbersome back-and-forth process querying configuration parameters that are prone to human error. My proposed research aims to intercept lower-level system interactions by experts setting up systems interactively and infer higher-level actions. We generate action sequences guided by the existing infrastructure serving as an oracle goal state.

4 Working groups

4.1 Intersections of Infrastructure as Code (IaC) and Configurable Software

Myra B. Cohen (Iowa State University – Ames, US), Claudia Cauli (Amazon Web Services – London, GB), Coen De Roover (VU – Brussels, BE), Dario Di Nucci (University of Salerno, IT), Thomas Durieux (TU Delft, NL), João F. Ferreira (INESC-ID – Lisboa, PT), Wolfgang Küchlin (Universität Tübingen, DE), Shane McIntosh (University of Waterloo, CA), Ruben Opdebeeck (VU – Brussels, BE), Akond Rahman (Auburn University, US), Martin Schäf (Amazon Web Services – New York City, US), Davide Taibi (University of Oulu, FI), and Marcel Van Lohuizen (CUE – Zug, CH)

In this breakout, we discussed the intersections between infrastructure as code and traditional configurable software, where configurability models the variability in system behavior rather than defining how the system is composed at a single point in time. Our goal was to find ways we can share techniques, tools, and benchmarks across these two domains. We discussed the fact that configurability in software is usually declarative, while IaC is often imperative. We also discussed the idea that the IaC view can be considered a unit level (or bottom-up view) of the system, while those in traditional configurable systems are using a systems approach (or top-down view). However, we agreed that many challenges are shared, so it would be important to look for places we can merge approaches. Some roadblocks to using a systems approach on IaC are that the systems are often unbounded; hence the complete configuration space is unknown. There is also no uniform modeling approach for IaC. We further discussed the problem that many of the dynamic techniques used in traditional configurable software, such as testing across configurations, may not work. We also noted that it is often too expensive to perform dynamic analyses because many IaC systems are often charged by usage micro-services. We agreed we should start with some example benchmarks, such as train-ticket to further explore the differences (and commonalities).

4.2 Architectural Reconstruction and IaC

Davide Taibi (University of Oulu, FI), Wolfgang Küchlin (Universität Tübingen, DE), Ruben Opdebeeck (VU – Brussels, BE), Ruzica Piskac (Yale University – New Haven, US), and Damian Andrew Tamburri (TU Eindhoven, NL)

During this breakout session, we discussed on pros and cons of techniques for architectural reconstruction. We identified mainly two analysis techniques: static and dynamic analysis techniques. Static analysis techniques scan different software artifacts, including IaC, source code, but also repository activity from git logs. We agreed that static analysis of source code might not be a viable solution, mainly because of the large number of technologies existing on the market. However, analysis of IaC might still be performed, because of the limited number of languages adopted.

Dynamic analysis instead models the system's behavior at runtime and allows one to understand how the system is performing or structured at runtime.

We highlighted the importance of merging the different visualizations, including the runtime analysis and the visualization of the infrastructure, which might become very important when considering continuum edge-to-cloud systems, where different types of devices and their resources can make the difference.

Security analysis might also be an important research avenue that could benefit from static, dynamic analysis, and their composition.

4.3 Reasoning about Code and Infrastructure

Martin Schäf (Amazon Web Services – New York City, US), Claudia Cauli (Amazon Web Services – London, GB), Myra B. Cohen (Iowa State University – Ames, US), Coen De Roover (VU – Brussels, BE), Dario Di Nucci (University of Salerno, IT), Thomas Durieux (TU Delft, NL), João F. Ferreira (INESC-ID – Lisboa, PT), Michael Greenberg (Stevens Institute of Technology – Hoboken, US), Shane McIntosh (University of Waterloo, CA), Akond Rahman (Auburn University, US), Daniel Sokolowski (Universität St. Gallen, CH), Marcel Van Lohuizen (CUE – Zug, CH), and Tianyin Xu (University of Illinois – Urbana-Champaign, US)

Many of the most prevalent CWE security issues, such as cross-site scripting (xss) or command injection, can be detected statically via data-flow or taint analysis. We simply define what parameters are considered untrusted (e.g., the arguments to a Servelet request in a Java application) and what sinks are considered sensitive (e.g., the members of a Servelet response in Java). This method works great if the application is known to the analysis. However, in a cloud setting, we usually have separate code bases and repositories for each compute component of the system. Consider an AWS application that receives some job identifier via a message queue, process that information via a serverless function, and then has the serverless function pass that job identifier on to another queue once processing is done.

While this is a typical design for processing information with micro-services, from a static, this serverless function has a cross-site scripting vulnerability since it parrots its input without modification. This cross-site scripting vulnerability will be a false alarm unless the input to the function can be controlled by an adversary and the output is sent back to a browser. To decide this, we will need to analyze the code and its infrastructure.

- We see multiple challenges in this space. The static analysis of the code will need to produce a summary that can be re-used by the static analysis of the infrastructure code.
- The mitigation for the code vulnerability may not happen in the code (e.g., xss could be mitigated by a firewall or gateway), so tracking issues, producing counter-examples, or generating patches will have to span over infrastructure code and application code.
- The infrastructure code itself my not be complete and not show the entire system, so analysis of infrastructure and application code will need to be able to report conditional findings (e.g., if message queue X receives untrusted data, then lambda Y will forward it to queue Z).

4.4 Emerging Trends in Infrastructure as Code (IaC) Research

Akond Rahman (Auburn University, US), Jürgen Cito (TU Wien, AT), Myra B. Cohen (Iowa State University – Ames, US), Thomas Durieux (TU Delft, NL), João F. Ferreira (INESC-ID – Lisboa, PT), Michael Greenberg (Stevens Institute of Technology – Hoboken, US), Wolfgang Küchlin (Universität Tübingen, DE), Daniel Sokolowski (Universität St. Gallen, CH), Davide Taibi (University of Oulu, FI), Marcel Van Lohuizen (CUE – Zug, CH), and Tianyin Xu (University of Illinois – Urbana-Champaign, US)

The objective of this working group is to identify emerging areas in the domain of Infrastructure as Code (IaC). As part of this breakout group, participants from academia and industry both agreed that the domain of IaC research has a lot of potential and till date remains an under-explored area. Through rigorous discussion, the participants suggested the following areas as emerging and therefore need to be addressed: (i) discovering and evaluating representations and reasoning for IaC, (ii) deriving metrics to measure complexity of configurations, (iii) derivation of single source truth for IaC-based infrastructure, (iv) investigate quality assurance for IaC compilers, (v) understand how to combine static and dynamic analysis procedures for IaC, (vi) quantify coverage for IaC-specific tests, such as idempotency detection coverage, and (vii) architectural inference from IaC.

While discussing these emerging areas participants shared real-world examples as well as described their personal experiences when working with industry partners. Participants also mentioned research papers in other CS-related research areas, such as security and software product lines can benefit IaC research. Participants also remarked that the ongoing discussions of the seminar helped them realize the scope and novelty of IaC research.

Participants

- Claudia Cauli
Amazon Web Services – London, GB
- Jürgen Cito
TU Wien, AT
- Myra B. Cohen
Iowa State University – Ames, US
- Coen De Roover
VU – Brussels, BE
- Dario Di Nucci
University of Salerno, IT
- Thomas Durieux
TU Delft, NL
- João F. Ferreira
INESC-ID – Lisboa, PT
- Michael Greenberg
Stevens Institute of Technology – Hoboken, US
- Yann Hamdaoui
Tweag I/O – Paris, FR
- Wolfgang Küchlin
Universität Tübingen, DE
- Anthony W. Lin
RPTU – Kaiserslautern, DE
- Shane McIntosh
University of Waterloo, CA
- Ruben Opdebeeck
VU – Brussels, BE
- Ruzica Piskac
Yale University – New Haven, US
- Akond Rahman
Auburn University, US
- Guido Salvaneschi
Universität St. Gallen, CH
- Mark Santolucito
Barnard College – New York, US
- Martin Schäf
Amazon Web Services – New York City, US
- Daniel Sokolowski
Universität St. Gallen, CH
- Davide Taibi
University of Oulu, FI
- Damian Andrew Tamburri
TU Eindhoven, NL
- Marcel Van Lohuizen
CUE – Zug, CH
- Tianyin Xu
University of Illinois – Urbana-Champaign, US

Remote Participants

- Andy Zaidman
TU Delft, NL