Automation and Analysis of DevOps Steps

Thomas Knickerbocker

2 Dec. 2023

Introduction

The aim of the research project was to conduct a detailed analysis of the "operate" step within the DevOps workflow cycle, as well as a lighter analysis of the “deploy” step. Initially, a comprehensive explanation of the position that the step occupies within the broader DevOps cycle was presented to elucidate its connections with both preceding and succeeding stages. Subsequently, to underscore the significance of the analysis, the benefits inherent in automating this step were discussed. Following this, criteria for evaluating the effectiveness of automation tools in this context were formulated and justified. The paper then delved into a few approaches for automating both the “operate” and “deploy” steps, so as to illuminate the reader regarding their innerworkings. The objective of this is to offer a holistic perspective on how automation could be strategically implemented. This approach enabled an assessment of three widely-used contemporary automation tools, circa December 2023, through the lens of the defined criteria.

Finally, findings are summarized, and a recommendation based on the preceding investigative work is provided to guide DevOps programmers in the selection of suitable tools tailored to their specific operational contexts. The conclusion is complemented by references to validate this research.

DevOps Steps: Operate & Deploy

The DevOps cycle is an ordered process for the continuous integration and delivery of software. Its ordered steps are: plan, code, build, test, release, deploy, operate, and monitor (Canlas). The goal is to provide an extremely adaptable infrastructure for successful product and service availability. With this in mind, the "operate" step is nestled within the deploy and monitor stages, both of which are post-release practices explicitly included in the model. The goal of the "deploy" stage is to move code to a production environment so that it can be available to users ("Importance of the DevOps Pipeline (and How to Build It) - DevOps at Scale."). Meanwhile, the "monitor" step aims to both track and measure the performance and health in systems and applications (Sharif). Thus, a successful operate stage must bridge the gap between deployment and monitoring such that upon deployment, systems are healthy and monitorable. This consists of ensuring high application availability to end users via ensuring the deployment of releases to servers and the availability of servers so that they can support the number of application users at any given time (Pennington).

Automating any step within the DevOps cycle ultimately expedites the process of delivering changes to end-users, in addition to slashing long-run operating costs for businesses. There are multiple ways in which this occurs. Firstly, it has been suggested that an automated development process reduces employee turnover rates (Masse). Second, automation naturally leads to less time consumption on development, meaning faster development cycles, and thus, a higher deployment frequency. The 2019 State of DevOps Report details that the most "elite" DevOps performers deploy on-demand, multiple times per day, and had a lead time from commit to deploy 106 times faster than that of "low" performers (Accelerate). Finally, unmanning the DevOps process leads to reduced compute costs via the leveraging of both serverless and pay-as-you-go compute instances from cloud providers such as AWS (Masse).

Within the context of the "operate" step, automation plays a pivotal role in enhancing the overall availability of the application and facilitates seamless scaling of deployments in response to user demand, as well as software configuration. By dynamically automating these tasks, application owners can achieve cost reduction, ensure continuous availability for users, and enhance deployment frequency. The frequency of adjustments should be as close to "constant" as possible; ideally, scaling occurs instantaneously so that applications can immediately meet an increase in user need, or scale down to save cost if usage drops. This is perhaps the most valuable segment of operations to automate, as it directly translates to pricing, and manually altering these things requires employees to be paid by the hour, who will certainly lag in performance when compared to a computer orchestrating such adjustments. With that being stated, automating software configuration has the exact same benefits and should absolutely be automated along with server adaption in any high-functioning DevOps environment. A more general answer to the question of how often the operate step should occur is as follows: whenever developers want an application to be available for use by others in a production environment.

Evaluation Criteria

Now that the scope of the "operate" step has been defined, we can feel confident in defining a set of criteria used to evaluate the performance and value of tools for its automation. The criteria which I have named includes the reusability of the automation, its setup time, price, how easy it is to learn, its scalability, versatility, and the platforms it both supports and is supported by. I believe all of these to be extremely important metrics, and shall justify each inclusion subsequently.

The reusability of an automation sometimes makes and can always break its value. Recyclable processes minimize setup redundancy. Even reusable components are useful, for they mean less code and/or setup should be required in future cycles. In fact, an automation which is not reusable is not an automation at all; it is manual.

The setup time of an automation, that is, the amount of time required to initialize it (both on a per-use basis and in the case of a first-time setup), is an important factor. Employee time costs money. A considerable subcomponent of setup time is how easy to learn a tool is. If the tool is highly abstracted and has a UI workers are already somewhat familiar with, then it can be learned and implemented more quickly by existing employees. This becomes even more significant if non-technical employees are able to grasp and implement the automation, as they are invariably less expensive to hire.

The scalability of automation applications is critical to their usefulness. To provide full value, an automation must be able to scale to fully meet the current organizational needs. If the DevOps applications the organization employs are expected to grow beyond their current utilization in the future, then scalability beyond what is currently used also becomes an important factor. At the core of scalability is the ability to quickly upsize and downsize operations, and as is the case with reusability, application breadth is value.

Next comes a point which no organization can ignore: price. Some automation tools cost money to use, and others do not. Virtually all commercial softwares base their pricing on some combination of utilization and time, so organizations must be mindful of their expected usage patterns when selecting tools.

Versatility is also an important factor. I define this as how many services a tool can and will provide an organization. Tools offering multiple services simplify workflows, which is a preferable trait; Less stress over software versioning, components, potential failure points, and business coordination are involved by tools offering multiple services in one.

Finally, organizations should consider the applications and infrastructure which both support and are supported by the tool. A tool which integrates into existing infrastructure is more valuable than one that does not. Tools also gain support value if they are supported by technologies which an organization may use in the future as well. For example, if an organization anticipates a cloud migration to Azure, and the tool in question easily integrates with Azure, then the support value attributed to that tool by the organization may be quite high. Applied more generally, more integrations with other softwares and platforms are better.

Automation Approaches and Challenges

Automation of the operate step can be done in a many ways. One way is managing automatic production environment server scaling (where servers automatically allocate computing resources when certain parameters are met, generally along the guidelines of current and expected demand). This helps ensure availability, and in the case of cloud apps, automatically scaling these resources down during periods of low activity can save users money. Services which are able to automatically communicate with softwares such as Kubernetes are able to do so.

Additional automation approaches may involve configuration, encompassing the setup of storage resources and server configurations, such as defining permissible IPs. For example, one might create or use a tool which blocks certain IPs from visiting a domain, or sets a limit to the number of requests they're allowed to send to it. This could be automatically set and enforced. Another example would be configuring databases to automatically move data that has not been accessed in over 90 days from a database to a data warehouse, or even delete it. One might also automatically configure the database to expand or restructure itself once certain parameters are met, such as some number of hours passing or some threshold for storage utilization being reached.

Challenges in automation (affecting both the operate and deploy steps, in this instance) manifest themselves in proper configuration; Connecting to a horde of servers to scale with an application at will requires great distributed coordination and bidirectional information sharing between systems coordinating resource usage and those providing services to end-users. Selecting automation tools which fit into existing infrastructure is an additional challenge which organizations face; Compatibility of software either makes or breaks its practicality, and large organizations may find themselves constrained by previously existing infrastructure and practices.

Tool Analysis

AWS CDK is a robust infrastructure as code (IaC) tool developed by Amazon Web Services, allowing users to define cloud infrastructure using familiar programming languages like TypeScript, Python, and Java. It excels in promoting reusability via constructs for automatic build deployment, and allows for the creation of modular components. CDK has a quick setup and learning process, especially for users familiar with these programming languages and AWS itself. CDK meets scalability needs by enabling the definition of scalable infrastructure components, and AWS. It is also incredibly versatile, as a user could elect to use AWS to meet virtually any organizational needs on a single platform, making it incredibly convenient ("AWS Cloud Development Kit."). Additionally, server scaling can be automatically configured through AWS EC2, which enables automatic scaling and is highly configurable to allow users to balance server availability with pricing to suit their needs (Engdahl).

A limitation lies in support, primarily focusing on AWS services, which may impact its value for organizations in multi-cloud environments. It does, however, integrate with Docker images, and AWS provides interfaces for leveraging with kubernetes clusters. While CDK itself is free for small amounts of usage, frequent usage requires payment from the customer. Additionally, its parent platform, AWS, is more expensive than competitors such as Azure and

GCS for storage (Gil). In essence, CDK is easy to learn with a low setup time, reusability, scalability, and versatility but has weaknesses in both support and cost.

Terraform is an IaC too created by HashiCorp in 2014. Terraform had an open-source license until August of 2023, when HashiCorp announced that it was transferring the license used for Terraform to a BSL1.1, meaning that code access is available to all, with the specific caveat that competitors of HashiCorp within the cloud space are denied access from its latest versions (Horovits). Terraform allows for policy as code, is highly reusable (utilizing a plug-and-play model), and scalable, is free to use, and is only slightly more challenging to learn than CDK. Its versatility is also considerable: users can automate usage limits, rollbacks, event notifications, and offers private datacenter connectivity ("Automate infrastructure on any cloud with Terraform"). Pricing limits on server usage can be set via Terraform's integration with policy-as-code tools like Sentinel ("Enforce Policy as Code."). In terms of support, Terraform integrates with Azure, AWS, Terraform Cloud, Google Cloud, Oracle Cloud, Docker, Kubernetes, and on-prem services, making it impressively non-partisanly well-supported ("Automate infrastructure on any cloud with Terraform.").

OpenTofu is an open-source IaC and a part of The Linux Foundation Projects. As of October 30th of 2023, it is functionally identical to Terraform in all ways but two: its licensing and owners. It was released following Terraform's licensing change announcement with the goal of ensuring the software remains open-source. Its development is community-driven and their website suggests that anyone is welcome to fork off of their repository and commit ("What Is OpenTofu?"). With that being said, a weakness presents itself, albeit a minor one: a lack of versatility. Hashicorp has many products, making it a convenient place for organizations to source many of their softwares from, and even try to strike special package deals with the company. Since OpenTofu is a standalone open-source software, it does not offer the same all-in-one convenience potential of HashiCorp’s Terraform. Additionally, the tool’s less professional oversight may indicate a higher propensity for bugs in the future, which large organizations should be weary of.

Conclusion

In conclusion, the analysis of AWS CDK, Terraform, and OpenTofu reveals distinct strengths and weaknesses, guiding the choice of a tool to automate DevOps operations tasks. AWS CDK stands out with its ease of learning, low setup time, reusability, scalability, and versatility. However, limitations in multi-cloud support and potential cost considerations need attention. Terraform, an open-source tool with policy as code, extensive support, and strong versatility, presents a compelling option. Its recent licensing change to BSL1.1 has raised considerations, but its free usage and robust features make it a formidable choice. OpenTofu, while functionally identical to Terraform, stands out for its commitment to open-source principles. However, it lacks the versatility of HashiCorp's suite of products, limiting its convenience for organizations, and is more likely to contain bugs in future releases than HashiCorp’s products.

Comparing the tools based on the defined evaluation criteria, Terraform emerges as a well-rounded choice, excelling in reusability, setup time, scalability, versatility, and broad platform support. While AWS CDK offers strengths in specific areas, its limitations in support and potential cost factors make Terraform a more comprehensive and cost-effective solution. OpenTofu, while commendable in its commitment to open-source, lacks the versatility and convenience potential offered by Terraform's broader ecosystem.

Therefore, my general recommendation is to adopt Terraform for automation of the "operate" step within the DevOps cycle, while advising users to remain apprehensive to future licensing changes coming from HashiCorp and remembering the existence of OpenTofu. Terraform's strengths in reusability, scalability, and extensive platform support make it a robust choice for organizations seeking a versatile and well-supported solution.

Works Cited

Accelerate, 2019, *State of DevOps 2019*, https://services.google.com/fh/files/misc/state-of-devops-2019.pdf. Accessed 5 Dec. 2023.

"AWS Cloud Development Kit." *Cloud Development Framework - Cloud Development Kit - AWS*, amazon, aws.amazon.com/cdk/. Accessed 5 Dec. 2023.

Canlas, Julian. "DevOps Loops: Everything You Need to Know." *Instatus Blog*, instatus.com/blog/devops-loops. Accessed 5 Dec. 2023.

"Enforce Policy as Code." *Terraform by HashiCorp*, www.terraform.io/use-cases/enforce-policy-as-code. Accessed 5 Dec. 2023.

Engdahl, Sylvia, et al. "AWS." *Amazon*, Greenhaven Press/Gale, 2 Nov. 2023, aws.amazon.com/blogs/mt/automating-amazon-ec2-auto-scaling-with-amazon-cloudwatch-custom-metric-and-aws-cdk/.

Gil, Laurent. "Cloud Pricing Comparison: AWS vs. Azure vs. Google Cloud Platform in 2023." *CAST AI – Kubernetes Automation Platform*, 16 Mar. 2023, cast.ai/blog/cloud-pricing-comparison-aws-vs-azure-vs-google-cloud-platform/.

Horovits, Dotan. "Terraform Is No Longer Open Source. Is OpenTofu (Ex Opentf) the

Successor?" *Logz.Io*, logz.io/blog/terraform-is-no-longer-open-source-is-opentofu-opentf-the-successor/#:~:text=HashiCorp%2C%20the%20company%20behind%20Terraform,Source%20License%20(BSL)%201.1. Accessed 5 Dec. 2023.

"Importance of the DevOps Pipeline (and How to Build It) - Devops at Scale." *Plutora*, Plutora, 14 July 2021, www.plutora.com/devops-at-scale/pipeline#:~:text=developer%20for%20resolution.-,Deploy,users%20in%20the%20final%20stage](https://www.plutora.com/devops-at-scale/pipeline#:~:text=developer%20for%20resolution.-,Deploy,users%20in%20the%20final%20stage.

Masse, Justin. "6 Ways DevOps & Automation Save Money." *1904labs*, 25 Feb. 2021, insights.1904labs.com/blog/2021-02-25-6-ways-devops-automation-save-money.

Pennington, Jakob. "The Eight Phases of a DevOps Pipeline." *Medium*, Taptu, 27 July 2020, medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba.

Sharif, Arlan. "What Is Devops Monitoring?" *Crowdstrike.Com*, crowdstrike, 6 Feb. 2023, www.crowdstrike.com/cybersecurity-101/observability/devops-monitoring/. Accessed 5 December 2023.

"Automate infrastructure on any cloud with Terraform." *Terraform by HashiCorp*, www.terraform.io/. Accessed 5 Dec. 2023.

"What Is OpenTofu?" *OpenTofu*, opentofu.org/docs/intro/. Accessed 5 Dec. 2023.