Harsh Artwani
1001953350

Assignment 2
Testing Infrastructure as Code (IaC)

Preamble:

A common definition of IaC is that it replaces manual effort associated with resource management and provisioning by simply writing a line of code. There are two types of IaC methods: declarative and imperative. We specify what the desired end state should be in the declarative approach, and the system ensures that we obtain the desired result. To achieve the intended end state, the imperative approach requires us to define each and every step in the process explicitly.
IaC is crucial in software testing because it improves speed and consistency. The purpose of IaC is to speed up the process by reducing manual processes and slack. A code-based approach allows us to accomplish more in less time. There's no need to wait for the IT Admin to finish the current task before moving on to the next; IaC puts the power in the hands of the developer, allowing for a more efficient software development lifecycle. Developers can begin to focus more on application development as infrastructure provisioning becomes more dependable and consistent. Also, IaC reduces the requirement for storage, networking, computation, and other hardware and middleware layers to be governed and managed. Admins can now identify the next exciting technology they wish to utilize.

Project Author View:

In his blog, Carlos Nunez explains how it eliminates the decades-old "works well on my computer" problem, in which code that works in testing does not work in production. IaC provides consistency by automatically provisioning and configuring all environments, leaving no space for human error, substantially speeding up and simplifying software development and infrastructure management.
With human error being one of the leading causes of IT system downtime, where even mistyping a DNS record can completely destroy an online business, reducing the possibility of these errors through automation is of paramount importance to every firm.
Carlos takes us through a series of examples that show how various tasks may be integrated into our overall DevOps pipeline flow such that continuous infrastructure testing is a key element of our overall continuous delivery life-cycle.
As a result, we'll be able to build a solid basis for high confidence in the release cycle. Because every sandbox environment will eventually become production, every sandbox environment created by an integration test will be an exact clone of production. This is a crucial step toward infrastructure immutability, as any modifications to production will be included in a hotfix or future feature release, and no changes to production will be authorized or required.

Potential Generalized View:

Sean Porter, another author, agrees with Carlos that integrating IaC gives operators and devs a single way to learn about their infrastructure and gives them a sense of how things are deployed and configured. We also give the security team more leverage because they can audit that

single source of truth to ensure we're deploying in a secure manner. If they have issues, they can express them in the context of infrastructure as code – because everything is codified and documented in that code, we can have cross-team discussions with the same context and knowledge.
IaC has the benefit of being a single source of truth: knowledge about how our services work and the dependencies they require can be shared throughout an entire application delivery team, rather than just a small group of highly technical operations personnel.
Because our infrastructure is built from code, we can version it and discuss modifications with our team.
We can (and should!) test our setup code to make sure it doesn't break unexpectedly. If an upstream package changes an interface and breaks its dependencies, for example, we can notice the change as soon as it occurs in our testing environment and postpone any package upgrades until the problem is resolved.

Personal Viewpoint:

To summarize, the IaC described in the studies could have the following potential future implications. What if, in order to fully operationalize the approach, everyone in the engineering organization needs to comprehend IaC (language, concepts, etc.)?
This may result in a mostly unresolved gap between Ops, who may be attempting to optimize their setup as much as possible, and developers, who may be wary of modifying IaC scripts for fear of breaking something.
What if there was a platform that could bridge this gap and provide an extra layer between developers and IaC scripts in the future? Platform could allow developers to easily self-serve infrastructure via a user interface. Developers may only need to worry about what resources (DB, DNS, storage) they'll need to deploy and run their apps, while the platform handles IaC script calls.

References:

- Nilesh Deo. 3 Advantages and Challenges of Infrastructure as Code. Link:https://www.cloudbolt.io/blog/3-advantages-and-challenges-of-infrastructure-as-code-iac/
- 2i Testing (2021, February 11). The Role of Software Testing in 'Infrastructure as Code'.Link:https://2itesting.com/media/blog/the-role-of-software-testing-in-infrastructure-as-code/
- Carlos Nunez (2017, July 3). Strategies for Ultra-Reliable Infrastructure-as-Code. Link:https://www.contino.io/insights/top-3-terraform-testing-strategies-for-ultra-reliable-infrastructure-as-code
- Sean Porter (2019, April 10). Infrastructure-as-Code: evolution and practice. Link:https://sensu.io/blog/infrastructure-as-code-evolution-and-practice
- Luca Galante (2021, June 30). Infrastructure as Code: The Good, the Bad and the Future. Link:https://humanitec.com/blog/infrastructure-as-code-the-good-the-bad-and-the-future