Infrastructure as Code: A Devops First Approach

Shivam[1], Poornima Kulkarni[2]

[1]Shivam, Student, Dept. of ISE, RVCE
[2]Poornima Kulkarni, Asst. Professor, Dept. of ISE, RVCE

---***---

Abstract - *Infrastructure as code (IaC) is a set of methods which make use of "code (in lieu of than manual operations) for getting setup with (virtual) machines as well as networks, installing dependencies, and configuring the development and production environment for the tool or software at hand. The infrastructure controlled by this code includes both the physical machines ("bare metal") and virtualized machines, docker containers, software-defined virtual networks. This code should be developed and managed using the same version control system as any other repository, for illustration, it should be constructed, tested, and warehoused in a version-controlled repository.*

Even though IT operators have long used automation by the use of ad hoc scripting for tasks, IaC technology and practices emerged with the introduction of cloud computing, and particularly infrastructure-as-a-service (with big names like Google and Microsoft offering their own cloud infrastructure services). While cloud-based service providers enable the administrative console that abstract an interactive application on top of REST APIs, it is not feasible to use a management console to create an automated system with more complex nodes. For example, creating a new virtualized resource using Microsoft Azure requires an IT operator having to step through 4 web-forms and filling some 20 or more fields. VMs are created and torn down many times during the day with deployment and tests running continuously, so performing these tasks physically is not advisable

Key Words: IaC, Docker, REST APIs, Terraform

1. INTRODUCTION

IaC code can be used common throughout development, integration, and production environments. This improved environment parity and can remove scenarios where software works in one developer's environment but not for another developer, or scenarios where software works in development but not in the integration or production environment. The infrastructure code used for IaC ought to be stored in a version-controlled repository. This enables vigorous versioning of a deployed infrastructure: Any adaptation of the infrastructure can be produced using the IaC code corresponding to the desired edition. Together, automation and versioning deliver the potential to recreate a composition efficiently and consistently. This can be used to roll back a switch made during development, integration, or just as production and to support trouble-ticket regeneration and debugging. IaC be able to empower an IT operation put into put into practice called immutable infrastructure. In a traditional operations approach, infrastructure and application software is connected on individual nodes. Overtime, each node is individually patched, software is updated, and network and other configuration parameters are increased as needed. Configuration drift may develop, for example, as the patch up level varies across nodes. In a few advanced cases, nodes can be recreated just from a backup, with no means to reconstruct the structure from scratch. In an immutable infrastructure, patches, revises, and configuration changes are never put into the deployed nodes. Instead, a new version of the IaC code is built with the alterations that reflect the needed changes to the deployed infrastructure and applications. Environment parity allows the new version to be assessed in development and integration environments preceding to production, and environment versioning make available the new changes to be rolled back if there is an unanticipated issue after deploying to production. Each of the highlights about IaC become our purpose for this project.

2. MOTIVATION

IaC is closely related to DevOps. By computerizing the creation of execution/test environments, IaC practices promote agile values. Reduce the time that is between committing a shift to a system and the shift being placed into production, while ensuring elevated excellence greatly enhances software development.

The aim of the study is to develop code repository for provisioning of virtualized resource instances for all machines involved in the product deployment and integrating them to the existing developer pipelines. Added effort of resource provisioning and setup of those resources can cause hinderance to development effort. This could likely cause developers to skip integration tests altogether. Enforcement of such checks can only be made feasible if the entire operation is automated and well-integrated into the testing pipeline. This should help in enforcing more quality checks and discovery of bugs and issues will be quicker and reliable.

3. LITERATURE SURVEY

A systematic learning about the mapping of existing IaC pillar technologies which had already helped rapidly deliver software and service to the consumer, [1] talks about the IT giants such as GitHub, Facebook, Mozilla Firefox, , Google and Netflix premier who have implemented IaC and collection a

set of complex artifact information about DevOps and its insights.
[2] talks about various frameworks/tools for infrastructure as code, adoption of IaC, empirical investigations which are related to infrastructure as code. According to [3]s analysis, 50% of the studies publication reports propose to choose a framework or tool which implements the practice of IaC or to some extend provide the capabilities of some previously existing IaC tool.
[4] findings lead us to believe that framework or tools is a well-premeditated topic and research about the faults and security flaws can have major significances for the production positioning and development environments involved in DevOps, it is well observed that the need for studies and research in this area is important and much needed.
[5] talks about how IaC is recent approach that and intend to improve collaboration between the development team and IT operation teams and create a channel for efficient communication and understanding. The studies involved are for the current frameworks which support such dimensions of operations. The approaches are compared and the tools like Chef and Puppet, are evaluation for coherency with existing CI/CD pipelines.
The lessons learned in [6] are those which allows use to create a set of concrete practices that would assist in transitioning from existing traditional approaches to an automation process of continuous software delivery.
Works in [7] are primarily focused in proposing abstract frameworks, which are designed to create a consensus to the ownership of DevOps characterization and their features. Some components such a collaboration as a philosophy and monitoring have surfaced as part of the study and are well discussed to and event approach.

4. PROPOSED METHOD

To develop the system required to automate the task at hand, the setup of virtual infrastructure with services pre-installed and in a ready to deploy stage, use of open source tool Terraform was chosen.

The workflow around terraform was divided into three parts where the first part is Code. During this phase, a declarative language is used to define out end state, this includes all the attributes and properties of the desired infrastructure. The second phase is the plan phase, it used a Terraform command and it is essentially to validate our declarative files and run static as well as dynamic checks on the code repositories. It also calculates the delta between the current resources available and the resources desired, thus only provisioning those added or modified.

The third and final stage for our task becomes the apply phase where our resource configuration and API tokens are used to communicate with real world cloud provider API end-points and work with those end points to deliver us the resources we defined. This phase entails certain output variable and is essentially another module in the Terraform binary.

5. RESULTS AND ANALYSIS

This section deals with the results obtained by putting the developed system into a production like environment and monitoring its performance. An analysis of the results thus obtained is examined and the impact of the new in placed system is checked against the existing solution. Thus, the automated provisioning and setup is put to test against manual operation and configuration, and the pre-established objections are verified in parallel.

5.1 RESULTS

The outcome of the project as already defined in the initial chapters would be a pipeline whose task is to provision and setup virtual machines for the integration tests to run. The given pipeline had various specifications which were required to be met. The outcome of the project could only be achieved by following the planned strategy and methodology, achieving the objectives is described in the given order.

The major outcomes of this project were:

1. Pipeline in place and coherent with existing CI/CD pipelines.
2. Pipeline definition defined and jobs of the pipeline limited only to the action of context required.
3. Mindful resource management by the pipeline, this means that resource be destroyed as soon as their scope of use is closed.

The consequences and results of the objectives established:

1. A completed successful build for the pipeline artifacts. The artifacts in concern are mainly the terraform resource files, the python modules created to install dependencies and test scripts written in various languages.
2. Importing of successful build artifact from the build pipeline to the release pipeline. This object is straight forward but involved manual configuration as of now.
3. Displaying output visualization in the form of debugged json files, the output JSON can be scanned for the credentials and log file paths.
4. Successful start-up event messages from each of the services which are launched on the virtual machines.

5. Each of the microservices setup should be alive when checked through the health status API. The microservice should be containerized in its environment and should be able to communicate on the desired API port.
6. Understanding the flow of data through the pipeline and understating the resource life cycle for the virtualized resources, the resources are not limited by hardware but the network as well as data resources.
7. The resolution of identified issues during the knowledge kit transfer is the seventh objective.
8. Integration of existing pipeline of build and release with the newly created testbed pipeline. Testbed pipeline here refers to the CI runner which executes the MakeFile targets.
9. Report collection at a central place and visualization through a common dashboard.

Though not explicitly mentioned the results also showcase the benefit of time and effort that was the direct result of this study.

5.2 COMPARATIVE ANALYSIS

The current method of manual provision and setup which was being use since the release of the first beta version has been overcome in its limitations and barriers. The major highlights that are overcome by the new system in place are:

1. Manual effort reduced by 90% during the operation.
2. Time required for the effort reduced by 50%.
3. Amount of developer time saved for each testing phase 2-man days.

Benefiting from these advantages would again allow the consumer of this system to:

1. Have better enforcement of coding quality and bug discovery.
2. Issues faced and barricades to VM provisioning reduced by greater margin than expected.

6. CONCLUSION

The base motivation for the project was the limitation and enhancements identify by the existing testing strategy, which involved manual effort and crucial developer time. This limitation was then translated to the requirement for the project "automation provisioning of virtualized resources", to overcome the limitation faced by the existing developer team. For teams which aim at using Terraform as a key player to their changing management and deployment pipeline, it becomes crucial to identify a orchestration terraform module which can also deal with some category of automation in order to guarantee consistency flanked by runs, and deliver interesting insights about such deployments. The project was successful in bring about this automation and should be considered while implementing any future endeavours of the same nature.

REFERENCES

[1] Terraform by HashiCorp stack https://terraform.io/document/api/v2/refrence

[2] Waldemar Hummer, Florian Rosenberg, Fabio Oliveria. "Testing Idempotence for Infrastructe as code" Springerlink . LNCS volume 8275, (2003)

[3] O. Lavriv, M. Klymash, G. Grynkevych, O. Tkachenko and V. Vasylenko, &quot;Method of cloud system disaster recovery based on &quot;Infrastructure as a code&quot; concept,&quot; 2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET), Slavske, 2018, pp. 1139-1142.

[4] Juve G, Deelman E. Automating Application Deployment in Infrastructure Clouds. Cloud Computing Technology and Science (CloudCom). IEEE Third International Conference on. Athens: IEEE; 2011. p. 658-65.

[5] 2. Zhang R, Shang Y, Zhang S. An Automatic Deployment Mechanism on Cloud Computing Platform. Cloud Computing Technology and Science (CloudCom). IEEE 6th International Conference on. Singapore: IEEE; 2014. p. 511-8.

[6] Jan Stanek, and Lukas Kencl, "Enhanced Secure Thresholded Data Deduplication Scheme for Cloud Storage", IEEE Transactions on Dependable and Secure Computing, Vol.15, Issue.4, pp.694-707, 2017

[7] Nguyen Cong Luong, Ping Wang, Dusit Niyato, Wen Yonggang and Zhu Han, "Resource Management in Cloud Networking", IEEE Communications Surveys &amp; Tutorials Vol.19, Issue.2, pp.954-1001, 2017

[8] A. Rahman, &quot;Anti-Patterns in Infrastructure as Code,&quot; 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), Vasteras, 2018, pp. 434-435.

[9] M. Guerriero, M. Garriga, D. A. Tamburri and F. Palomba, &quot;Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry,&quot; 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), Cleveland, OH, USA, 2019, pp. 580-589.

[10] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero and D. A. Tamburri, &quot;DevOps: Introducing Infrastructure-as-Code,&quot; 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, 2017, pp. 497-498.

[11] V. Shvetcova, O. Borisenko and M. Polischuk, &quot;Domain-Specific Language for Infrastructure as Code,&quot; 2019 Ivannikov Memorial Workshop (IVMEM), Velikiy Novgorod, Russia, 2019, pp. 39-45.