Technology Trends

Infrastructure as Code

Enterprise Architecture, Chief Technology Officer Branch

Version 0.1
Date 2020-Jan-28

Shared Services Canada | Services partagés Canada

Canada

Table of Contents

Business Brief

Infrastructure as Code (IaC) represents the modern practices of provision, configuring and managing an IT infrastructure through machine-readable configuration files rather than deploying physical hardware and systems. As such, data centers define traditional IT components such as servers, virtual machines, load balancers, databases, network topology and other related systems as their software equivalent.

IaC grew as a response to the difficulty many large tech companies were facing when it came to scale their IT infrastructure. Historically, software delivery was a manual process that required a system administrator to setup a physical server with the appropriate OS and all the necessary service packs to be installed and configured with the desired settings before deploying the applications onto it. As part of the software development process, the whole setup had to be deployed in multiple environment such as development, staging and production. This cause the environmental drift problem where, overtime, each environment becomes a snowflake with a unique configuration cannot be reproduced automatically resulting in inconsistencies or conflicts during deployments.

IaC evolved to solve this problem in the release pipeline where files are changed under source control. The IaC concept is like programming scripts, which are used to automate IT processes. However, scripts are primarily used to automate a series of static steps that must be repeated numerous times across multiple servers.[1] IaC uses higher-level or descriptive language to code more versatile and adaptive provisioning and deployment processes.[2]

The wide adoption of virtualization, self-service cloud infrastructure and, particularly, Infrastructure as a Service (IaaS), institutions had to automate their whole infrastructure deployment process in a repeatable and consistent manner to eliminate human error. This principle of Infrastructure as Code is called idempotence where a deployment using a specific configuration will always generate the same results regardless of the environment. The benefits of using Infrastructure as Code over the manual approach are :

- **Speed**: IaC makes the software development lifecycle more efficient by allowing you to quickly set up a complete infrastructure just by running a script.
- **Consistency**: Many mistakes occur when using the manual process because humans are fallible. IaC solves that problem by having the configuration files themselves be the source of truth and to avoid any discrepancies.
- **Cost**: By employing cloud computing along with IaC, the cost of infrastructure management is dramatically reduced. Money spent on hardware, operators, physical space to store the systems all add up but with IaC, automation strategies frees engineering from performing manual, slow and error-prone tasks to focus on what matters the most.

Technical Brief

Infrastructure as Code represents the point where both automation and virtualization come together. Based on the practices of software development, it emphasizes consistent, repeatable routines for provisioning and changing systems and their configuration. Below are some of the problems of traditional that Infrastructure as Code addresses.

- **Server Sprawl**: This situation occurs when IT resources including hardware and software applications in a data center remains under-utilized, leading to poor productivity and proficiency.
- **Configuration Drift:** When a primary hardware and software infrastructure is initially configured, differences or "drifts" with a supposedly identical secondary recovery configuration can creep overtime due to the sheer number of ongoing infrastructure changes. This accounts for most of the disaster recovery and high availability systems failure.
- **Snowflake Servers:** As mentioned before, a snowflake server is different from any other server on the network in such ways that it's too difficult to replicate. Should the hardware start having problems, setting up another server that supports the same configuration is improbable.
- **Fragile Infrastructure:** Expended on the idea of a snowflake server, a whole system that can be easily disrupted and not easily fixed is a major problem for data centers.
- **Automation Fear:** Many system admins are reluctant to adopt automation tools because of their servers inconsistency and that they might break the underlying systems.

Infrastructure as Code is meant to overcome these issues presented above cause by the manual approach to infrastructure management. Because IaC solutions is often wrapped up with the topic of automation, many of the best practices involves smarter deployment of scripts and automating the manual process. Below are some of the benefits IaC provides.

- **Systems can be easily reproduced:** IaC provides the ability to effortlessly and reliably build and rebuild any part of the infrastructure without any risks or fears. Making changes and encountering failures can be handled quickly and can be resolved with confidence.
- **Systems are disposable:** Because of the dynamic nature of the infrastructure, resources can be easily created, destroyed, replaced, resized, and moved. Systems are designed based on the assumptions that the infrastructure is always changing, and it should continue running while changes are made. This enables the ability to constantly make improvements to a running infrastructure.
- **Systems are consistent:** While deploying two infrastructure providing similar services, these servers should be mostly identical. Having the ability to reproduce multiple identical infrastructure elements eliminates the configuration drift problem.

- **Processes are repeatable:** Building of the idea of reproducibility, any actions made on an infrastructure should repeatable. The benefits of using scripts and configuration managements tools rather than manual changes
- **Design is always changing:** It is impossible to predict how a system will be used in practice seeing how requirements changes over time. IaC ensures a system can be changed safely, quickly and frequently to meet the change and release schedule.

From a technical stand point, Infrastructure as Code is the process of configuring an infrastructure by coding it instead of doing a manual execution. Software that implements IaC are known as Configuration Management tools. There are several IaC tools available in the market that offers different approaches to Infrastructure as Code. Generally, every tool follows one of three approaches presented below. The main differences between declarative, imperative and intelligent approach can be summarize by "what" versus "how" versus "why" respectively.

- **Declarative:** In declarative programming, the desired and intended outcome of the infrastructure is defined in a configuration file. There is no need to outline the sequence necessary to reach this end result. AWS CloudFormation follow the declarative style of IaC.
- **Imperative:** The imperative approach focuses on how the infrastructure needs to be change in order to reach the desired outcome. This procedural approach defines a sequence of commands or instructions. A tool such as Chef can be used in the declarative manner, but imperatively as needed.
- **Intelligent:** The intelligent approach considers the reasoning behind the configuration by determining all the co-relationships and co-dependencies of multiple applications running on the same infrastructure. IaC tools of this nature determines the correct desired state before the system executes what needs to happen to achieve a desired state that does not impact co-dependent applications.

Infrastructure as Code is part of the larger practice known as "DevOps" whose goals is to shorten the systems development life cycle thought the use of different sets of tools called "toolchains". The different categories of DevOps can be automated by repackaging platforms, systems and applications into reusable building blocks through by using technologies such as virtualization and containerization. Here are the different Implementations of DevOps automation and how Infrastructure as Code interacts with each of them:

1. **Coding**: The software engineering practice wants to keep track and provide control overs changes to source code. This is accomplished using source code management tools that performs version, revision control or source control. Because IaC configuration files are essentially scripts, a layer of versioning can be added to Infrastructure as Code to see the history of incremental changes to the environment. Value is generated by this traceability and reusability of the code.

2. **Building**: In software engineering, continuous integration and delivery (CI/CD) is the practice of merging all developers' working copies to a shared mainline. These automation tools are particularly necessary when working in an agile software development methodology where changes are made daily. Value comes from the quality of the software and the fact that it is always in a state that can be deployed to users.
3. **Testing**: Continuous Testing tools is the process of executing automated tests as part of the software delivery pipeline. The scope of these tests aims to validate the system requirements associated with the overarching business goals.
4. **Packaging**: The software development lifecycle aims to group systems and application into reusable packages by using virtualization and containerization technologies. Instead of reinventing the wheel for each new software, package managers give access to previously developed software to new applications. The value from these tools comes from the fast and easy distribution of their dependencies. The latest trend in architectural style is microservices where every IaC component is modular to give better control and efficiency.
5. **Releasing**: This Application-release automation (ARA) is the process of packaging and deploying an application. It combines the workload automation and release-management tools as they release the packages that allows for an increase in visibility for the whole team. Because IaC serves as a Documentation in itself, deploying code is more clear-cut and consistent.
6. **Configuration**: Configuration management tools utilize Infrastructure as Code to define and maintain configuration of their system. The goal is maintaining the highest level of serviceability for the lowest cost.
7. **Monitoring**: Applications performance monitoring (APM) strive to detect and diagnose complex application performance problems through performance metrics. They monitor the end user experience, the application runtime, the user-defined transaction profiling, component monitoring, etc.

Industry Use

The industry has been leveraging IaC to quickly and effortlessly build cloud-based IT infrastructure through software and data definitions. Those businesses are able to use software development tools such as version control systems, automated testing libraries, and deployment orchestration to manage and configure their infrastructure in the cloud. The IaC methodology also provides new opportunity to use other software development tactics like, test-driven development, continuous integration, and continuous delivery. Environments capable of testing software are now be created easily and with little manual work put in. With this methodology, IT infrastructure becomes less of a constraint on development but rather supports it.

With the emergence of the DevOps model, IaC is being used in virtual infrastructure maintenance and design. Its premise grants the user more flexibility with the creation of

their environments and how they can test their software. For companies like Amazon, Netflix, Google, Facebook, and Etsy, IT systems are not just business critical; they are the business. There is no tolerance for downtime.

Infrastructure as Code is made more accessible thought a comprehensive range of tools available for automating the entire provision process thanks to the fast-paced evolution of the practice. IaC tools are divided in two categories. The first is configuration orchestration tools which includes Terraform and AWS CloudFormation, which are designed to automate the deployment of servers and other infrastructure. The second of which is configuration management tools like Chef and Puppet that help configure the software and systems already provisioned on this infrastructure. The most popular IaC tools are listed below:

- **Terraform**: This Open Source infrastructure provisioning tool created by Hashicorp allows teams to build and manage wide scale infrastructure estates with software delivery principles. Being multi cloud compatible, Terraform supports most common cloud and DevOps tooling by using its own domain-specific language (DSL) known as HCL to create JSON-compatible configuration files.
- **AWS CloudFormation**: Similar to Terraform, this configuration orchestration tools allows you to code your infrastructure to automate deployments. As its name suggests, it is deeply integrated and can only be used with AWS but it uses YAML in addition JSON to define the configuration files.
- **Chef:** This popular configuration management tools help organisation in their continuous integration and delivery processes. Chef uses a procedural approach by creating "recipe" and "cookbooks" in a Ruby-based DSL to specify the exact steps needed to achieve the desired configuration of your applications and utilities on existing servers. Chef is compatible with cloud service providers such as AWS, Microsoft Azure, Google Cloud Platform, and more.
- **Puppet**: Similar to Chef, this tool is another popular configuration management tools to help engineers continuously deliver software. It differentiates itself from other solution by using a declarative approach where a user declares what the configuration should look like and Puppet figures out how to get there.
- **Docker**: This tool provides an easy way to package your code and dependencies into containers that can run in any environment. Docker's configuration files use YAML and they are called Dockerfiles. They serve as blueprints to build container images that includes everything from code, runtime, system tools and libraries needed to run a piece of software.

Canadian Government Use

The GC has begun consuming public cloud services. These services are offered as part of the Government of Canada's (GC) Cloud Adoption Strategy. IaC is a strategic methodology when embarking in a cloud-based endeavours since it facilitates the deployment and maintenance of virtual infrastructure. One such example is the Government of Canada Financial and Material (GCFM) solution aiming at providing central infrastructure to support financial planning and analytics in the GC. With a public cloud virtual infrastructure the need for IaC is paramount to keep cost down and maximise the use of cloud based resources.

The GC will also be adopting SAP Business Panning and Consolidation (SAP BPC), a business application which provides the end user with a target environment for planning, consolidating, and reporting of financial processes. With this application being used, IaC will prove to be useful in two ways, in automating maintenance activities and accelerating repeatable testing. IaC allows the user to retain multiple versions of the same environment and only changing small configurable data at a time.

Being able to return to the previous versions of the same environment will prove to be useful in maintenance, as the exact version of an environment can be redeployed and configured in the same manner without manual work.

Shared Services Canada (SSC) has created a Github workspace for Microsoft AZURE Tools and templates to accelerate GC service delivery, deployments, and to be reused and improved upon via a whole of Government approach.[3] The objective is to accelerate service delivery and compliance through the use of automation and tools that will enable departments to deploy secure cloud-based environments aligned with GC policies and standards.[4]

Implications for Shared Services Canada (SSC)

Value Proposition

IaC can provide business value for IT service delivery, cloud based application development, DevOps, and general IT maintenance through automation, versioning, and documenting configurations. However, in general, IaC provides value in Speed and Simplicity, Configuration Consistency, Visibility and Auditability, Increased Efficiency in Software Development, Cost Savings, and Security.

Speed and Simplicity

One thing that professionals like about IaC is the portability. If hardware systems are provisioned as code, it is easier to move that code or deploy it in different environments.[5] Just run the IaC script and the infrastructure environment is up and running, ready to test environment and configuration changes. IaC can spin up an entire infrastructure architecture by running a script. Not only can virtual servers be deployed, but pre-configured databases, network infrastructure, storage systems, load balancers, and any other cloud service can be launched via scripts. This can be done quickly and easily for development, staging, and production environments, which can make the software development process much more efficient. Additionally, standard infrastructure environments in other regions where the cloud provider operates can be used for backup and Disaster Recovery, all by writing and running code.

Configuration Consistency

Through IaC an organization can minimize human error through the standardized IaC code. Standard Operating Procedures (SOPs) can help maintain some consistency in the infrastructure deployment process, but human error still exists and still results in subtle differences in configurations that may be difficult to debug. IaC completely standardizes the setup of infrastructure so there is a reduced possibility of any errors or deviations in redeployments. This will decrease the chances of any incompatibility issues with an organizations infrastructure and help applications run more consistently across the IT environments of the organization.

Visibility and Auditability

An IaC template serves as a very clear reference of what the organization's resources are, including their settings. IT personnel do not have to navigate to the web console to check the parameters. In this way, if the lead infrastructure analyst leaves, the knowledge won't leave with them. IaC not only automates the process, but it also serves as a form of documentation of the proper way to deploy infrastructure and also acts as a type of organizational knowledge insurance in the case of employee departure.

Configurations will change to accommodate new features, additional integrations, and other edits to the application's source code. If edits are made to the deployment protocol, it can be difficult to know what adjustments were made and who was responsible. Since code can be version-controlled, IaC allows every configuration change to be documented, logged, and tracked. These changes in configurations can also be rigorously tested, just like code. If there is an issue with the new setup configuration, it can be pinpointed and corrected with much more ease, minimizing risk of issues or failure. IaC supports and enables change, rather than being an obstacle or a constraint. Changes are made in small increments instead of batches.

The version control system (VCS), usually Git via Github, is a core part of IaC. The VCS is the source of truth for the desired state of infrastructure. Changes to infrastructure are driven by changes committed to the VCS. VCS is essential for infrastructure management in that is provides Traceability (history of changes), Rollback (ability to restore to previous version), Correlation (tracing problems across environments), Visibility (changes are public to the team), and Actionability (automatically trigger actions when a change is committed).[6]

Increased Efficiency in Software Development

IaC allows an organization to use Continuous Integration and Continuous Deployment (CI/CD) techniques while minimizing the introduction of human errors after the development stage. IaC scripts can also spin down environments when they're not in use. This will shut down all the resources that the script created, reducing the likelihood of orphan cloud components that IT personnel don not know if they should delete or not. This simplifies the working area the productivity of IT staff should increase through having a clean and organized account to work with.

Standardization and Change Management: When the creation of new infrastructure is coded there is the assurance of a consistent set of instructions and standardization. Manual configurations are prone to errors and minor changes which can create ever so slight differences that over time represent major nonconformities with the standard (and technical debt). Standardization assurance enables safer changes to take place, with lower deviation rates.

Stability: If you accidentally change the wrong setting or delete the wrong resource in the web console you can break things. Infrastructure as code helps solve this, especially when it is combined with version control, such as Git.

Scalability and Immutability: With IaC source code can be written once and then reused many times again. One well written template can therefore be used as the basis for multiple services, in multiple regions, and around the world. This makes it much easier to horizontally scale. IaC provides the ability for additional resources to be provisioned during burst periods allowing horizontal scaling and the ability to replace resources in

the event of failure. Since the code has already been written, high request periods can be better planned as infrastructure is spun up and down as needed.

Testing: Measuring recovery time can be accomplished much more easily with IaC. For modern cloud environments, an isolated environment can be created much faster with IaC where an exact copy of the production environment is created. This new environment can be completely isolated from the actual production or live environment, making simulating disasters and measuring recovery time a much simpler task. Additionally, by having a copy or multiple copies of production QA, Security and User Acceptance can all be thoroughly tested at the same time in separate staging environments.

Reduced Shadow IT: Much of Shadow IT, meaning the hardware/software not supported by the organization's central IT department, is due to the inability of IT departments to provide satisfactory and timely answers to operational areas concerning IT infrastructure and systems enhancements. Shadow IT is usually the result of an employee whose desire for immediate access to hardware or software leads them to turn to obtaining that hardware/software without going through slower corporate IT processes. Shadow IT poses significant security risks as well as potential unforeseen costs for the organization. IaC mitigates this risk as it enables a fast response to new IT requirements through IaC assisted deployments and speedier testing. This not only assures higher security and compliance with corporate IT standards, but is also helpful with budgeting and cost allocation.

Cost Savings

Manual provisioning consumes a large part of human capital. It demands a string of analysts, network engineers, and storage engineers acting within an extensive process. The more the people involved, the higher the costs. Through IaC, fewer IT personnel spend less time doing low-value manual work and more time working on high-value tasks. Automating the infrastructure deployment process allows It personnel to spend less time performing manual work, and more time executing higher-value tasks. Because of this increased productivity, your company can save money on hiring costs and engineers' salaries. As mentioned earlier, your IaC script can automatically spin down environments when they're not in use, which will further save on cloud computing costs. An IaC model generates the same environment every time it is applied.

Security

IaC provides an organization with a unified template for how to deploy the architecture. A well scripted infrastructure code with secured architecture built in can be reused multiple times and every subsequent deployed version can follow the security parameters and security compliance measures consistently imposed by the IaC.

Challenges

While IaC can provide many benefits, there are challenges associated with implementing this method, including: Initial Set-Up Costs, Server Sprawl, Configuration Drift, Automation Fear, Error Duplications, Infrastructure Erosion,

Initial Set-Up Costs and IaC Planning

People have to be hired to perform the tedious setup work. You'll need network engineers to set up physical network infrastructure, storage engineers to maintain physical drives, and many others to maintain all of this hardware. That leads to more overhead, management, and costs. Real estate has to be acquired to build data centers to house all of this hardware. On top of that, you'll have to maintain these data centers, which means paying maintenance and security employees, HVAC and electricity expenses, and many other costs. Since different people are manually setting-up these servers, setups are bound to be inconsistent. This can lead to unwanted variance in configurations, which can be detrimental to how your applications run.

Additionally, a main challenge is the governance and proper planning for onboarding IaC as a process. Once a company decides to move towards having an IaC capable IT landscape in place, there is the mandatory need to define the infrastructure that will allow the implementation, configuration, and operation of IaC tools. Resolving this in large organizations can prove very challenging before any IaC work begins.

Getting Environments In Sync and Server Sprawl

For IaC to work properly, the test environment and the production environment need to be synced up, and the documentation kept organized.[7] Without the test and production environments being in sync, changes in configuration could have wildly different consequences and impacts when implemented on the production environment.

While cloud and virtualization can make it easy and trivial to provision new servers from a pool of resources. This can lead to the number of servers growing faster than the ability of the team to manage them well. When this happens, teams struggle to keep servers patched and up to date, leaving systems vulnerable to known exploits. When problems are discovered, fixes may not be rolled out to all of the systems that could be affected by them. Differences in versions and configurations across servers mean that software and scripts that work on some machines don't work on others. This leads to inconsistency across the servers, called Configuration Drift.

Configuration Drift

Configuration Drift is when servers are initially created and configured consistently, but where differences creep in over time. Unmanaged variation between servers leads to

snowflake servers and automation fear. Drifts in configuration can happen over time and a variety of things can cause this. If administrators change server configurations outside of the set IaC template, there is potential for Configuration Drift. It's important to fully integrate IaC into systems administration, IT operations, and DevOps practices with well-documented policies and procedures.[8] Once adherence to an IaC workflow is achieved to create something, any foreign interference will change the server environment. Once a machine is created via an IaC workflow, it should not experience intervention outside of an automated, aligned, and compliant maintenance workflow. Manual or external updates (even if just security patching) may result in Configuration Drifting which in time has the potential of producing massive non-compliance or even service failure.

Automation Fear

Many IT personnel have a fear of letting the automation tools run on their own. IT personnel often use automation selectively based on their confidence in the tools and/or the environment, for example to help build new servers, or to make a specific configuration change. However, because the IT personnel do not fully trust the automation tools and/or process and will often tweak the configuration each time they run it to suit the particular task at the time.

IT personnel are afraid to use automation properly, because of a lack in confidence in what they would do because servers are not consistent, or there is a lack of understanding of automation. Servers tend not to be consistent because automation is not run frequently and/or consistently. Automation fear is a risk that can plague many teams. Although automation saves time, placing trust in the system can be a difficult task especially when the code is populating a cloud full of server instances and supporting infrastructure configuration.

Accidental Destruction – Some IaC tools that maintain state have the ability to automatically destroy resources should the code reflect that action. IaC in an automation pipeline can sometimes have undesired outcomes.

Error Duplications

Although the subsequent creation of machines would be through automation, the development of the initial parent code will be done manually. More often than not, whenever there is a human process involved, there is the possibility of minor errors that creep into the overall process. The problem here is that several machines may have been automatically created where such errors exist. So there is the need for applying a solid auditing process to the creation of IaC generating code.[9]This can happen despite regular QA checks. These minor issues could prove to be crucial, as such errors might also be in multiple machines created by means of automation.[10]

Infrastructure Erosion

Erosion is the idea that problems will creep into a running system over time, even without. In an ideal world, once an automated infrastructure is in place, it should not require manual changes other than to support something new or fix things that break. Sadly, the forces of entropy mean that even without a new requirement, infrastructure decays over time.

Demand for New Skills

There is a need for a high level of technical expertise to work with IaC tools. From a management perspective, this means investing in current employees and hiring new ones. Some even to resort to outsourcing services during the initial phases. Outsourcing the onset will give the organization's staff an opportunity to familiarize themselves with the tools and time for the tools to become user-friendly. Regardless, there an investment in the knowledge and skills of the resources will be a challenge when implementing IaC. Most existing IaC tools require expertise to be handled, and reaching such levels requires significant time in learning and training.

Considerations

When implementing IaC across an organization, certain considerations should be made to avoid service failure. Items to consider include: Consider the Organization's Workflow, Deploy Code As Much as Possible, Version Everything, Consult the DevOps Teams, Requires Strong In-Depth Knowledge, and Pilot Small and Scale Success.

Consider the Organization's Workflow

For IaC to work properly it must be conducted in an enclosed process system using the appropriate automation. Consider the workflow of the organization before implementing IaC widespread. Manual additions and/or changes often break the entire system, one would have to ask whether the current IT ecosystem state is ready for such a restriction to be applied. Careful consideration must be made as to what parts of the IT ecosystem can be implemented and managed with IaC before deploying it wholesale. Piloting and small case studies are highly suggested to help an organization achieve a level of understanding and maturity on the subject. If the organization has a largely manual process, identify areas that can experiment with IaC, do not look to IaC for critical business applications without first acquiring some experience.

Deploy Code As Much as Possible

A good practice to consider for IaC is to, wherever possible, deploy code to describe the infrastructure. Often, it is possible to codify traditional and cloud infrastructure, even legacy systems. For instance, the physical/virtual server management can be codified by Terraform, CloudFormation, YAML, and Python scripts. From there Puppet/Chef modules can be utilized for network management, Dockerfile for container

management, and so forth, which can establish these configuration files as a single source of truth when it comes to the organization's exact infrastructure specifications.[11]

Version Everything

Versioning is heavily dependent on deploying code as much as possible. Configuration files will need to be version controlled. Currently this is done in the industry using Git and Github. Since the files are coded, it becomes possible for to track, manage, and restore the changes made, should the need arise. It'll also assist in diagnosing problems. There are many source code management tools available that you can make use of for versioning and change tracking. Consider the need for auditability for each application. IaC code serves as documentation in and of itself. So, instead of humans manually executing based on the guidelines in documents, deploying code is more clear-cut and consistent.

Consult the DevOps Teams

IaC will be crucial if when implementing DevOps in an organization. It can be the key component needed to enable the DevOps best practices and to get the most out of DevOps. The principles of IaC and DevOps intertwine when it comes to collaboration and automation. Also, the DevOps toolchain often encompasses infrastructure automation tools. When infrastructure is coded, it paves the way for the platform to achieve superior quality control through better testing, reduced recovery times, and more predictable—as well as more effective—deployments.[12]

Requires Strong In-Depth Knowledge

IaC requires a significant investment in knowledge and skills on the IT personnel. SSC should consider the investment required to retrain current employees or hire new talent. Hiring new talent would need to be a major consideration but it should be coupled with other experience in DevOps and programming for best candidate recruitment. Another aspect of training personnel is to reduce their fear of automation. The fear of using automation, or employee trust in the automation tools will be something the organization will need to seriously consider in it's training requirements.

Pilot Small and Scale Success

SSC should consider evaluating the current Service Catalogue in order to determine where IaC can be leveraged first to improve efficiencies, reduce costs, and reduce manual administrative burdens of existing services. Additionally, determining how IaC will integrate into existing services on a consistent basis. Any new procurements of devices or platforms should have high market value and can be on-boarded easily onto the GC network such as Terraform or AWS. SSC should avoid applying IaC for production mission-critical apps at the onset. SSC should pilot and establish IaC test

clusters and scale the success. With all new cloud-based technologies, piloting is preferred. Focus should first be on a narrow set of objectives and a single application scenario to stand up a test IaC process cluster. The DevOps teams are the prime location for testing IaC in the IT environment.

References

Cardinal, G. (2017, January 24). *GCFM Planning, budgeting and forecasting Prototype*. Retrieved from ppx.ca/: http://ppx.ca/wp-content/uploads/2016/09/GCFM-Solution-BPC-PPX-Jan-2017.pdf

Chan, M. (2018, April 3). *15 Infrastructure as Code tools you can use to automate your deployments*. Retrieved from thorntech.com: https://www.thorntech.com/2018/04/15-infrastructure-as-code-tools/

Continuous testing. (2019, December 12). Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/Continuous_testing

Cowles, L. (2019, July 1). *How to use infrastructure as code*. Retrieved from opensource.com: https://opensource.com/article/19/7/infrastructure-code

Dadgar, A. (2018, August 20). *Infrastructure as Code: What Is It? Why Is It Important?* Retrieved from hashicorp.com/: https://www.hashicorp.com/resources/what-is-infrastructure-as-code

Google. (2020, January 28). *Infrastructure as code*. Retrieved from cloud.google.com: https://cloud.google.com/solutions/infrastructure-as-code/

Guckenheimer, S. (2017, March 4). *What is Infrastructure as Code?* Retrieved from docs.microsoft.com/: https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code

Lewis, T. (2017, June 29). *Devops Benefits of Infrastructure as Code*. Retrieved from stelligent.com: https://stelligent.com/2017/06/29/devops-benefits-of-infrastructure-as-code/

Mamnani, D. (2017, September 5). *From testing code to testing infrastructure as code—the new must-have testing skill*. Retrieved from capgemini.com: https://www.capgemini.com/2017/09/from-testing-code-to-testing-infrastructure-as-code-the-new-must-have-testing-skill/

Merron, D. (2018, December 17). *What is Infrastructure as Code? IaC Explained*. Retrieved from bmc.com: https://www.bmc.com/blogs/infrastructure-as-code/

Morris, K. (2020, January 28). *Infrastructure as Code*. Retrieved from oreilly.com: https://www.oreilly.com/library/view/infrastructure-as-code/9781491924334/ch01.html

Nallamala, N. (2019, April 19). *The Top 7 Infrastructure-As-Code Tools For Automation*. Retrieved from dzone.com: https://dzone.com/articles/the-top-7-infrastructure-as-code-tools-for-automat

Nallamala, V. (2018, November 11). *What Is Infrastructure as Code?* Retrieved from dzone.com/: https://dzone.com/articles/what-is-infrastructure-as-code-2

NetApp. (2020, January 28). *What Is Infrastructure as Code (IaC)?* Retrieved from netapp.com: https://www.netapp.com/us/info/what-is-infrastructure-as-code-iac.aspx

Null, C. (2020, January 28). *Infrastructure as code: The engine at the heart of DevOps*. Retrieved from techbeacon.com/: https://techbeacon.com/enterprise-it/infrastructure-code-engine-heart-devops

Plutora. (2019, July 29). *Infrastructure as Code: What Is It, and Why Should My Engineers Care?* Retrieved from plutora.com/: https://www.plutora.com/blog/infrastructure-as-code

Robinson, M. (2018, November 20). *Infrastructure as Code: Everything You Need to Know*. Retrieved from rollout.io: https://rollout.io/blog/infrastructure-as-code/

Rouse, M. (2015, October 30). *infrastructure as code*. Retrieved from searchitoperations.techtarget.com/: https://searchitoperations.techtarget.com/definition/Infrastructure-as-Code-IAC

Schults, C. (2019, September 5). *What Is Infrastructure as Code? How It Works, Best Practices, Tutorials*. Retrieved from stackify.com: https://stackify.com/what-is-infrastructure-as-code-how-it-works-best-practices-tutorials/

Sitakange, J. (2016, March 14). *Infrastructure as Code: A Reason to Smile*. Retrieved from thoughtworks.com/: https://www.thoughtworks.com/insights/blog/infrastructure-code-reason-smile

Wikipedia. (2019, December 7). *Application performance management*. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/Application_performance_management

Wikipedia. (2019, October 13). *Application-release automation*. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/Application-release_automation

Wikipedia. (2019, December 31). *Continuous integration*. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/Continuous_integration#CI/CD

Wikipedia. (202, January 23). *Infrastructure as code*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Infrastructure_as_code

Wikipedia. (2020, January 25). *DevOps*. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/DevOps

Wikipedia. (2020, January 13). *Software repository*. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/Software_repository

Wikipedia. (2020, January 17). *Version control*. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/Version_control

Woods, E. (2018, October 4). *Infrastructure as Code, Part One*. Retrieved from crate.io: https://crate.io/a/infrastructure-as-code-part-one/