INFRASTRUCTURE AS CODE (IAC): AN INTRODUCTION

Infrastructure as Code, or IaC, is a method of writing and deploying machine-readable definition files that generate service components, thereby supporting the delivery of business systems and IT-enabled processes. IaC helps IT operations teams manage and provision IT infrastructure automatically through code without relying on manual processes. IaC is often described as "programmable infrastructure".

Infrastructure as Code can be applied to the entire IT landscape but it is especially critical for cloud computing, Infrastructure as a Services (IaaS), and DevOps. DevOps requires agile work processes and automated workflows which can only be achieved through the assurance of readily available IT Infrastructure – which is needed to run and test the developed code. This can only happen within an automated workflow.

IaC for DevOps

Within the context of software development, a fundamental constraint is the need for the environment where recently developed software code is tested to exactly mirror the live environment where such code will be deployed to. This is the only way of assuring that the new code will not collide with existing code definitions – generating errors or conflicts that may compromise the entire system.

In the past, software delivery would follow this sort of pattern:

- A System Administrator would setup up a physical server and install the operating system will all necessary service packs and tuning to mirror the status of the main operating live machine that supports the production environment.
- Then a Database Administrator would undergo the same process regarding the support Database, and the machine would be handed off to a Test Team.
- The developer would deliver the code/program by copying it to the test machine, and the Test Team would run several operational and compliance tests.
- Once the new code had gone through the entire process it would be deployed to the live Operational environment. In many cases, the new code would not work correctly and additional troubleshooting and rework would be necessary.

Manual recreation of a live environment leaves doors open to a multitude of most likely minor but potentially quite important human errors e.g. OS version, patch level, time zone etc. A live environment clone created using the exact same IaC as the live environment has the absolute guarantee that that if it it works in the cloned environment it will work in live. Imagine a Software Delivery process involving DEV, UAT and Production environments - there seems little value in having a DEV and UAT environment that isn't an exact mirror of the Production environment given that those early environments are critical to measuring the quality and production readiness of a software build version.

The introduction of virtualization enabled the process to be expedited with regards to the phase of creating and bringing up to date a test server that would mirror the live environment, yet the process was manual, meaning a human would have to create and update the machine accordingly and in a timely fashion.

With the introduction of DevOps, the process became even more "agile" due to the addition of automation in the Server Virtualization and Testing phases, replacing human intervention.

To summarize, in the past several man-hours and human resources were required to complete the software deployment cycle (Developers, Systems Administrators, Database Administrators, Operation Testers), while now it is possible to have the developer complete all tasks:

- The developer writes down the application code plus configuration management related instructions that will trigger actions from the virtualization environment, and other environments such as the database, appliances, testing tools, delivery tools, and more.
- Upon new code delivery, the configuration management set of instructions will automatically create a new virtual test environment with an application server plus database instance that exactly mirror the live operational environment structure, both in terms of service packs and versioning as well as live data that is transferred to such virtual test environment. This is the Infrastructure as Code part of the process.
- Then a set of tools will perform necessary compliance tests and error identification & resolution. The new code is then ready for deployment to the live IT environment.

Current State of Infrastructure as Code

The general IaC concept, as well as available tools, has reached a very mature state with a lot of organizations having defined their roadmaps for adopting it.

There are now a number of tools available to adopt Infrastructure as Code with and the right tool will differ for every Infrastructure or DevOps team. The available tools differ widely in usage and

functionality so below is a high level overview of some of the most popular tools available for building and managing your IaC. Software forums and GitHub issue pages are packed with information, documentation and people willing to assist and therefore adopting IaC is much easier than it historically was.

Puppet/Chef

Chef was developed under the perspective of enabling fast collaboration amongst team members and is, therefore, a DevOps context focused support asset while Puppet has evolved targeting sheer processes automation making it useful for the fast creation of new infrastructure as per client requirements.

Ansible

Although not specifically an IaC tool, Ansible, an Open Source and popular configuration management tool, does have the required modules to build Infrastructure in a number of cloud and on premises providers. Whilst not cloud agnostic, Ansible does support multiple cloud providers.

Ansible is an agent less tool that operates over SSH (linux) or WinRM (windows) with the Infrastructure configuration code being written in YAML. Ansible is not stateful and is therefore it's main purpose is for creation of the infrastructure and not management of.

Great for Configuration Management, not so great for building and managing cloud and on-premise infrastructure.

https://www.ansible.com

Terraform

Terraform is an Open Source, stateful, multi vendor Infrastructure as Code tool that codifies API's into Terraform configuration files to allow teams to build and manage wide scale infrastructure estates with software delivery principles. Whilst not cloud agnostic, Terraform is multi cloud compatible whilst additionally supporting vendors who provide SaaS services or Container Orchestration tools e.g. StatusCake and Kubernetes.

Terraform is widely supported by most common cloud and DevOps tooling and the Terraform Module Repository has a great deal of pre-written terraform modules which you can simply populate with your input variables to build and manage your infrastructure. When building and maintaining infrastructure Terraform requires approval before applying destructive changes, adding a safety net incase of "bad" IaC causing undesired outcomes.

When moving from a previously non IaC environment, it is possible to import your existing resources into Terraform for continue management however this is a very manual and time consuming task - but possible nonetheless.

https://www.terraform.io

CloudFormation

Amazon Web Services specific Infrastructure as Code tool, written in JSON and run via the AWS Console or AWS CLI allows you to build and manage your infrastructure as code in AWS. Obviously,

not multi cloud however does offer the best AWS support whilst still being a stateful IaC tool. AWS provides a lot of templates that can be used to get started with CloudFormation so it's pretty easy to get up and running.

https://aws.amazon.com/cloudformation/

ARM Templates

ARM templates are Microsoft Azure's implementation of IaC and allow you to provision Microsoft Azure resources using a Declarative template. Not stateful and not Multi Cloud - simply good for building infrastructure and not managing it moving forwards.

https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-overview

Benefits of Infrastructure as Code

Some of the major benefits of Infrastructure as Code are:

- **Reducing Shadow IT** – Much of the shadow IT within organizations 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 poses significant security risks as well as potential unforeseen costs for the organization. Enabling a fast response to new IT requirements through IaC assisted deployment not only assures higher security and compliance with corporate IT standards, but is also helpful with budgeting and cost allocation.
- **Improving Customer Satisfaction** – Being able to deliver a quality service component within a short period of time contributes to customer satisfaction and improved perception of IT within an organization (as measured by Net Promoter Score or other method).
- **OPEX reduction** – If a company can configure and deploy a fully tested and compliant new IT Infrastructure asset within a matter of minutes either with minimal or even no human intervention, this represents a colossal saving in work time and security-related financial risk potential.
- **CAPEX reduction** – Being able to have a developer accomplish on her own the tasks of several team members, particularly in the context of DevOps, will significantly improve the project CAPEX.
- **Standardization** – 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).
- **Safer Change Management** – Standardization assurance enables safer changes to take place, with lower deviation rates.
- **Application of Software Delivery Principles** - Ability to promote and use Software Delivery principles, such as version control, peer programming, and code reviews to Infrastructure results in fewer un-planned outages and better change history tracking.
- **Scalable and Immutable Infrastructure** - 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.

Risks Involved with Infrastructure as Code

There is some risk that needs to be accounted for:

- **Missing proper planning** – Once a company decides to move towards having an IaC capable IT Landscape in place, there is the mandatory need to define Infrastructure that will allow the implementation, configuration, and operation of IaC tools.A simple example is that you can only create and operate virtual machines if you have in place a physical server infrastructure that can run a tool like VMware and is powerful and scalable enough (CPU, RAM, HDD) to support several heavy demanding simultaneous virtual machines running in parallel without any performance impact, plus redundancy that allows all to continue working within normal operational standards if problems occur.
- **IaC requires new skills** – Most existing IaC tools require expertise to be handled, and reaching such levels requires significant time in learning and training. Some companies are likely to start by resorting to outsourcing services until the tools become more user-friendly, staff is trained on the new tools, or new experts are brought on to the team.
- **Error replication** – Since the initial code is developed by humans, there is always the chance that it contains minor errors that will only produce impact after some time. The problem here is that meanwhile, 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.
- **Configuration Drift** – Once a machine is created via an IaC workflow, it should not suffer 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.
- **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.