6 Steps to Implement Infrastructure as Code

Published 22 October 2020 - ID G00726774 - 15 min read
By Analysts Daniel Bowers, Manjunath Bhat

Initiatives: Agile and DevOps; Infrastructure, Operations and Cloud Management

Infrastructure as code can deliver transformational agility and efficiency via infrastructure automation. Infrastructure and operations leaders should follow these six steps to implement IaC as part of their agile, DevOps and cloud initiatives.

Additional Perspectives
- Summary Translation: 6 Steps to Implement Infrastructure as Code
  (13 November 2020)

Overview

Key Findings
- Organizations that deploy orchestration and configuration management tools without changing the processes and culture of infrastructure management routinely fail to achieve their desired efficiency gains.
- Gartner clients consistently identify programming skills gaps in I&O teams as the biggest obstacles to automating infrastructure.
- Although organizations are familiar with version control and continuous integration/continuous delivery practices for application delivery, they lack the process maturity to apply the same practices to infrastructure delivery.

Recommendations
I&O leaders responsible for implementing infrastructure as code as part of their agile, DevOps and cloud initiatives must:

- Snowball adoption by building a batteries-included IaC framework that includes reusable templates and other features with clear benefits that will inspire teams to embrace it.
- Build skills and prove efficiency improvements by starting IaC initiatives with provisioning and settings management of servers and networks.
- Achieve transformative agility benefits by making IaC the default approach to infrastructure automation, with exceptions for legacy infrastructure for which it doesn't deliver benefits.
Strategic Planning Assumptions

By 2023, 40% of on-premises server provisioning will be powered by infrastructure as code (IaC), which represents a significant increase from less than 5% in 2020.

By 2023, 60% of organizations will use infrastructure automation tools as part of their DevOps toolchains, improving application deployment efficiency by 25%.

Analysis

IaC is a foundational requirement to scale infrastructure automation. By leveraging IaC, infrastructure and operations (I&O) leaders can reduce technical debt, increase agility, and improve governance and security. However, as of mid-2020, fewer than 5% of organizations manage their infrastructure this way. A lack of skills, and uncertainty about where to start, are holding organizations back.

IaC is defined as:

- The creation, provisioning and configuration of software-defined compute (SDC), network and storage infrastructure as source code
- The application of the process and disciplines traditionally associated with software development to manage IT infrastructure

IaC is also commonly called programmable infrastructure (PI), although PI can include other aspects of cloud-native automation, including containers and immutable infrastructure.

Most IT leaders correlate IaC with automation. Automation is the second-most common technology in which I&O organizations plan investment in 2021. IaC is a foundational building-block for automation, but doesn't deliver automation on its own. Automation requires both technical and nontechnical changes. Although you must deploy orchestration and configuration management tools to implement IaC, automation should be custom built to match your own templates, processes and governance.

You cannot buy automation.

Although IaC powers the DevOps toolchain, it can also be used outside DevOps. Tools and adoption of IaC are more mature for public cloud infrastructure than for on-premises infrastructure; however, capable tools and skills to apply IaC to on-premises infrastructure are available. Ultimately, not all infrastructure should use IaC, but IaC should be the preferred methodology, unless the use case justifies traditional manual management.

This research examines six steps for implementing infrastructure as code (see Figure 1).
1. Run Pilot — Start with the impactful, “low-hanging fruit” of automated deployment of server and network infrastructure configuration by implementing minimal-required-feature IaC, using orchestration tools, such as Terraform and Ansible.

2. Establish Toolsets — Enable operations at scale by creating artifact management of infrastructure definition, such as repositories, and roles, such as automation engineer.

3. Set Governance — Define objectives and key results (OKRs) that foster credibility through a culture of continuous improvement.

4. Scale Up — Snowball adoption through the reuse of use cases, templates, create automation engineers, establish or transfer ownership to platform teams.

5. Mature Tooling — Expand maturity via immutable infrastructure, validation testing and embracing CI/CD workflows for IaC.

6. Expand Use Cases — Expand use cases to disaster recovery (DR), autoscaling, flexible self-service catalogs for users and supporting DevOps.

Prework

Prior to launching an IaC pilot, you must understand several key paradigms about infrastructure management:

- **Orchestration versus configuration management** — Orchestration tools (e.g., Terraform and AWS CloudFormation) perform initial deployments or configurations of infrastructure. Configuration management tools (e.g., Ansible and Chef) automate the control of deployed resources to support operations. Orchestration and configuration management toolsets are merging.
Mutable versus immutable infrastructure — With mutable infrastructure, changes (e.g., OS update patches) are made on live production systems. With immutable infrastructure, systems are never updated in place; instead, the infrastructure is completely reconfigured whenever changes are required.

Procedural versus declarative definitions — Procedural definitions, sometimes called scripts, are an ordered sequence of instructions to perform that accomplish a specific task. Declarative definitions specify a desired state of a system without explicitly specifying the steps required to get there.

Step 1: Use Orchestration Tools to Deploy and Maintain Server and Network Configuration

The goal of Step 1 is to train some members of the team through lessons from small projects that demonstrate the measurable advantages of IaC. Quick wins not only prove the concept, they help build team confidence.

For the initial quick win, start with server provisioning, as well as server and network configuration management. The goal is building a minimum viable product to showcase its viability, not sophisticated wizardry that delivers on the full IaC promise in one step. For this first foray into IaC, the public cloud may be easier, because API-driven configuration is the norm. However, early IaC efforts can also be successful on-premises, because API-driven management is widely available in servers and network infrastructure systems.

Unless you standardize on a single toolset, automation efforts will lead to tool sprawl. Reuse investments by developer teams in automation tools, such as Ansible, Chef, Puppet and SaltStack. Paid (enterprise) variants of open-source-based IaC tooling are preferred to ensure access to ongoing bug fixes and feature enhancements, without breaking changes. Also choose tools with large ecosystems that include integrations with other infrastructure management systems you’re using, such as IT infrastructure monitoring (ITIM) and application performance management (APM) tools.

I&O must adapt to development processes and tools that require some software engineering skills. However, many I&O teams lack those skills, with a lack of staff expertise cited as one of the top-three greatest inhibitors of PI adoption (see Market Guide for Infrastructure Automation Tools). In addition, I&O teams often lack the sociotechnical organizational structures that allow developer teams to collaboratively maintain code and configuration files. Training through coaches is a prerequisite to build these skills (see How to Fix the Software Engineering Resource Gap in I&O). The use of “low code” IaC platforms, such as VMware’s vRealize Blueprint Designer, can be an onramp or crutch for I&O teams with no programming or scripting experience at all.

A seed crystal approach of taking the people involved in an initial project and using them to “seed” additional teams has had good results with Gartner clients adopting PI. However, the teams you are “seeding” must be prepared, with expectations set. In addition to technical skills, the seeds must also have experience with teaching or leading as a peer.
Step 2: Enable Operations at Scale Through Artifact Management of Infrastructure Definitions

The goal of Step 2 is to make IaC consistent and repeatable, enabling use across multiple infrastructure types. Tools will be opened for use by multiple teams, and the groundwork is laid for sharing best practices. As this happens, you will also expand your initial projects to adjacent use cases — for example, expanding from server deployment to user account management.

Now that infrastructure is defined declaratively in Step 1, Step 2 involves setting up artifact management for those infrastructure definitions. Code that describes infrastructure configuration and updating should be stored in a repository and managed using version control. Secrets, such as passwords required to connect to remote systems, should be stored in a vault solution, rather than in the code itself. The consistency of single repositories establishes a single-source-of-truth that enforces quality and reliability. Reasons for changes should be included in code, to make the system self-documenting. Community- and vendor-provided code, in the form of scripts, playbooks, and cookbooks, is leveraged to expand your capabilities, while avoiding the need to have staff reinvent common tasks.

IaC involves adopting a new language that describes the state of your infrastructure. For example, with tools Terraform and Puppet, the new language is actually a new domain-specific set of commands and grammar, similar to programming languages Java or Python. For other PI tools, including AWS CloudFormation and Ansible, the languages (e.g., JSON and YAML) are like configuration files. Although it is not a true development language (such as C and Python), repositories and version control for IaC are handled the same way as development code. Therefore, choose repositories and version control tools already being used by your organization’s developers. This will avoid the friction of adopting new software, while allowing you to leverage organizational expertise in these tools.

When using IaC, the trigger for taking action is not a request or event directly, but instead a push of code into the repository. A “push” means that an infrastructure engineer has made changes within the code that manages the infrastructure, and then has saved it to the repository. Requiring infrastructure changes be made only through commitments to the repository guarantees that no “in-place” changes to systems will fail to be documented. However, it also means that staff is not done once the server is configured, but instead owns the server definition indefinitely.

Step 3: Select OKRs That Foster a Culture of Continuous Improvement

Step 3 and Step 4 represent a qualitative shift from proving the minimum viable IaC to accelerating adoption across the organization with a scalable, value-rich and sticky platform. With these two steps, you cross a threshold of trust in your PI tooling and staff skills. Step 3 and Step 4 involve chartering
automation expert roles who own the infrastructure-as-code process, and motivating them to continuously improve capabilities.

Establish customer-centric objectives and key results (OKRs) to push teams to seek breakthrough performance. OKRs should allow continuous monitoring to measure progress, and to indicate when effort needs to be adjusted. Objectives should be ambitious, and make everyone a bit uncomfortable.

OKRs should be visible to all, so every team must know the key results they are responsible to deliver. OKRs should be at the product level, not at the individual employee/role level. These aren't measurements for employee evaluation.

Select three objectives, with three matching key results, aligned with strategic goals for the organization, and that foster a culture of continuous improvement. Table 1 gives three example OKRs for PI.

<table>
<thead>
<tr>
<th>Objective</th>
<th>Key Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Automate common operations tasks</td>
<td>Ratio of automated versus manual changes to infrastructure configuration</td>
</tr>
<tr>
<td>2  Keep software up-to-date</td>
<td>Percentage of systems at the most-recent, security-patch level</td>
</tr>
<tr>
<td>3  Grow IaC skills</td>
<td>Percentage of FTEs able to demonstrate minimum viable IaC skills</td>
</tr>
<tr>
<td>4  Expand user self-service</td>
<td>Number of services offered in a self-service catalog</td>
</tr>
<tr>
<td>5  Speed delivery of services</td>
<td>Average time to provision or update common services</td>
</tr>
</tbody>
</table>

Source: Gartner (October 2020)

**Step 4: Snowball Adoption Through Templates and Automation Architects**

Now that tools, standards, and goals are established, the goal of Step 4 is to foster wide expansion of IaC across teams and platforms. Scale initial small wins to a larger scale. Enable reuse of templates by other teams and projects. Shift ownership of the tools and templates to chartered automation specialists, such as automation engineers and automation architects.

IaC toolsets have a learning curve, and it is tempting for developers and administrators to revert to known scripting methods to complete specific tasks. IT leaders who want to maximize the value of PI
investments must replace these insular automation implementations with enterprisewide, consistent tools and processes. However, adoption can't be forced through executive dictate. Instead, product teams should be motivated to voluntarily adopt PI by offering a batteries-included platform that enables immediate benefits, along with ease of use, through things like reusable templates. Equally important to tool adoption is ensuring that organizations' cultures can embrace these tools strategically.

Dedicated automation specialists develop standards, guidelines, processes and useful tools. To succeed, automation specialists must combine real domain expertise with credibility and power in the organization. Automation architects use their understanding of IaC toolsets, plus knowledge about organizational policies, to achieve the promises of IaC. Automation engineers should evolve to pick up skills in multiple domains — for example, knowledge of both storage and identity management processes. (For more information on automation engineers, see Essential Skills for Automation Engineers.)

In more mature organizations, ownership of the IaC toolsets will transition to a responsible platform team. The platform teams perform all operational activities, including platform update and optimization, as well as the deployment of security, audit and compliance capabilities. In addition, the platform team continuously delivers and improves the processes and capabilities (see How to Scale DevOps by Building Platform Teams).

**Step 5: Expand Maturity via Code Validation Testing, CI/CD Tooling and Immutable Infrastructure**

Moving to Step 5 and Step 6 represents a qualitative step-change in PI maturity. PI has been established, but needs to mature in both process (how things are done) and scope (what things are done). Step 5 and Step 6 should be done in tandem.

In Step 5, expand your IaC maturity by using a CI/CD pipeline for infrastructure management. This incorporates validation and testing into the automated build and deploy process. For much (but not all) infrastructure, this also involves embracing the philosophy of immutable infrastructure to manage infrastructure configuration. In some shared services organizations, or organizations where users need self-service access to variable hardware configurations, Step 5 can also involve deploying composable infrastructure.

**Saving a script to a Git repository does not constitute IaC.**

At this point in your IaC journey, infrastructure configuration should be mostly declarative (see Prework section for a description of declarative versus procedural). Incorporate automated validation and testing of configuration definitions. Leverage tools used in DevOps, such as Spinnaker or Jenkins, maintaining a
commitment to quality and reliability. Your goal is to evolve toward a fully orchestrated workflow that includes ticket creation, security compliance and policy governance.

Immutable infrastructure is the management style in which the infrastructure, once created or configured, is never updated in place. Instead, when changes are required, that configuration is simply replaced. Immutable infrastructure reduces operational errors, improves security and simplifies troubleshooting. Instead of in-situ changes that lead to configuration drift, consistency is enforced through source-code control of the entire infrastructure configuration. Being able to treat infrastructure immutably is a good test of the maturity of your PI framework. In most cases, immutable infrastructure is only possible with appropriate workloads, and collaboration with development teams is required. (For more on immutable infrastructure, see the Hype Cycle for Infrastructure Strategies, 2020.)

At this stage of IaC, composable infrastructure can be exploited to provide rapid configuration of customized configurations, with highly efficient use of on-premises hardware. Composable infrastructure is an emerging architecture for server, storage and network hardware that creates physical systems from shared pools of disaggregated resources using an API. With PI in place, hardware that enables composable infrastructure can be used to deliver unpredictable or changing needs for things such as graphics processing units (GPUs) or nonvolatile memory express (NVMe) storage. (For more on composable infrastructure, see Understand the Hype, Hope and Reality of Composable Infrastructure.)

Step 6: Expand Use Cases to Self-Service Catalogs and Disaster Recovery

As you mature, expand the reach of IaC to automate other use cases:

- **Flexible Self-Service Catalogs** — Catalogs with static definitions for each service won’t meet user needs, and can’t be sustained at scale. Use IaC to create dynamic service definitions that enable users to change infrastructure parameters or resource requirements. Such flexibility accelerates user-driven adoption across an enterprise.

- **Disaster Recovery** — Because repositories hold all the necessary configuration information, and toolsets can automate infrastructure creation, use automate failover and failback of disaster recovery. You can also use IaC-powered automated disaster recovery to measure your actual recovery time, so you can specify more accurate and supportable RTOs.

- **Autoscaling** — By collecting runtime telemetry from applications, you can use IaC to automate the creation of additional resources as needed, using arbitrarily complex policies that match business requirements.

- **Provisioning of Test Environments** — Automate the creation of test, development, staging and other pre production environments using IaC, to ensure rapid, consistent deployment and tear-down.

- **Automated Rollback and Remediation** — IaC makes rollback easier. Leverage insights from IaC to prevent misconfigurations, resolve problems and provide more-advanced deployment and optimization capabilities.
Although IaC adoption is often driven by DevOps initiatives, I&O leaders can use IaC for infrastructure-led innovation as a spark for DevOps adoption (see Top DevOps Questions Answered).

Judiciously apply IaC practices to manage heritage infrastructure. Using IaC with highly customized, “snowflake” systems that require infrequent configuration changes might appeal as modernization, but they add little value. The mainframe running a monolithic core-banking application and the isolated cluster running a near-obsolete RDBMS often lack the configurability APIs and IaC toolset support. Licensing issues and risks of changing long-proven processes might also make IaC inappropriate. Instead, mesh the mature with the modern by applying IaC practices on the fringe of heritage systems. For example:

- Use a gateway to encapsulate the monitoring telemetry of heritage infrastructure into a RESTful API that can be consumed by modern toolsets.
- Store mainframe JCL code in the common artifact library used for other IaC.

**Evidence**

1. 76% of I&O organizations plan investments in YE21 in infrastructure automation tools (Source: Gartner Annual I&O Leaders Survey 2020, Question 4a; n = varies). “Automation” is the No. 2 I&O investment area, after cloud (Question No. 8).


**Recommended by the Authors**

- Market Guide for Infrastructure Automation Tools
- Hype Cycle for Infrastructure Strategies, 2020
- How to Automate Your Network Using DevOps Practices and Infrastructure as Code
- How to Automate Server Provisioning and Configuration Management
- Essential Skills for Automation Architects
- Solution Path for Infrastructure Automation