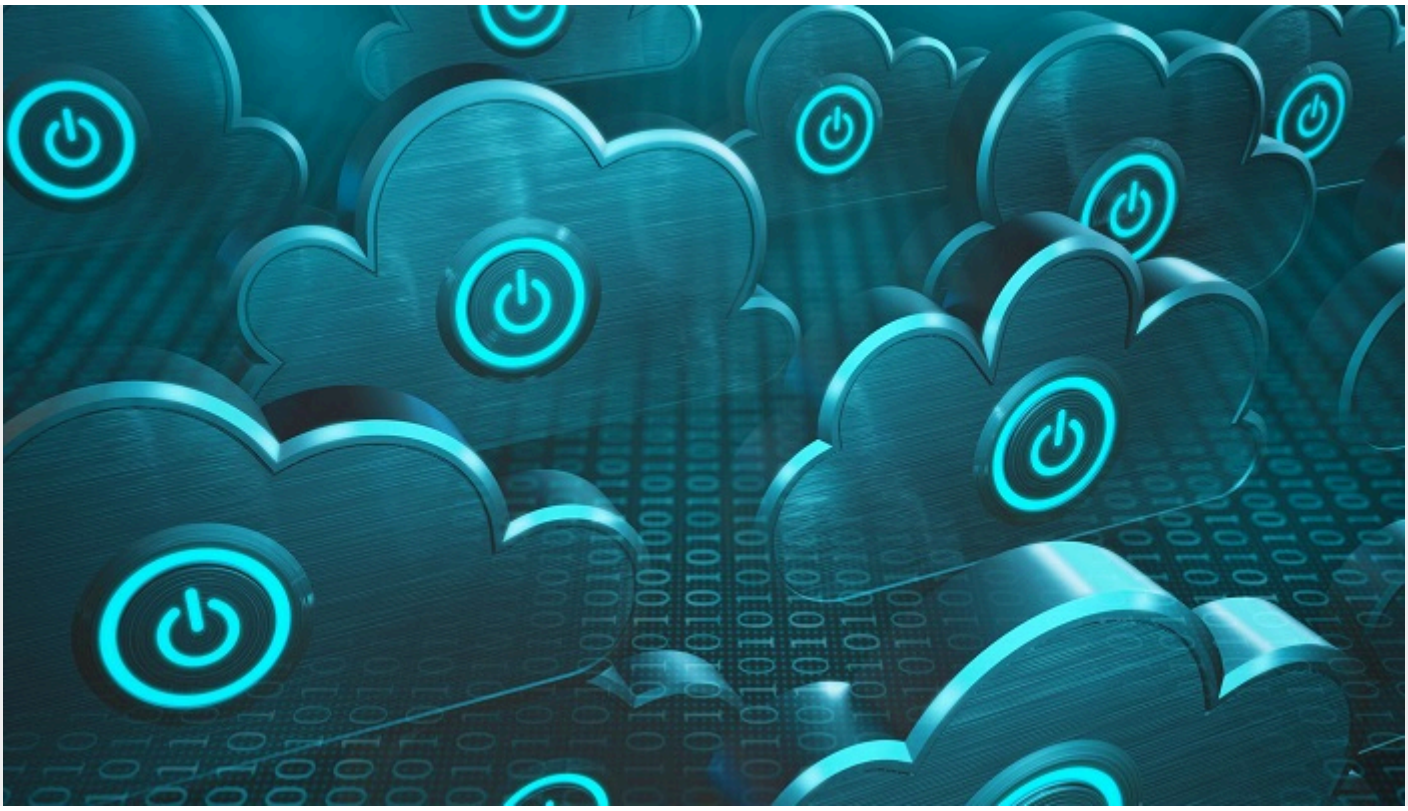
INFRASTRUCTURE AND HOW “EVERYTHING AS CODE” CHANGES EVERYTHING



This is the third blog in our mini-series that illustrates how BMC was able to use agile development, cloud services, an Infrastructure as Code approach, and new deployment technology to deliver a new cloud native product.

Be sure to also read: Getting Started with Cloud Native Applications, 9 Steps for Building Pipelines for Continuous Delivery and Deployment, and Five Best Practices for Building Security as Code into a Continuous Delivery Pipeline

This is the third blog in our mini-series that illustrates how BMC was able to use agile development, cloud services, an Infrastructure as Code approach, and new deployment technology to deliver a new cloud native product.

Remember how long it used to take to release software when infrastructure, security, compliance, and operations processes were done by independent teams separate from application development? Operations worked mostly in isolation, occasionally interacting with application development teams. The operations and development teams used their own, often different, tools and complex slow-moving processes, such as change boards, approvals, checklists and 100-page policy documents, along with specialized tribal knowledge. That is quickly changing as today's DevOps teams embrace best practices, such as the concept of ***Everything as Code*****,** to ensure agility while maintaining governance.

As organizations transform to deliver new digital services faster, the disciplines of infrastructure, security, compliance and operations must also evolve to meet the requirements for speed, agility, and governance. **The idea behind the *Everything as Code* concept is that infrastructure, security, compliance and operations are all described and treated like application code such that they follow the same software development lifecycle practices.**

We have been employing these principles in our development of a new SaaS product that runs on Amazon AWS. The following represents some of the lessons we have learned from others and then modified by our experiences in developing, delivering and implementing a new cloud native application.

Infrastructure as Code and the Impact on DevOps
Let's look at some Infrastructure as Code best practices we've learned after operating several production cloud applications on Amazon Web Services (AWS) cloud. Infrastructure includes anything that is needed to run your application: servers, operating systems, installed software packages, application servers, firewalls, firewall rules, network paths, routers, configurations for these resources, and so on.

1. *Define and codify infrastructure*
   Infrastructure is codified in a declarative specification, such as Cloud Formation templates for AWS cloud, Azure resource templates for Azure cloud, Docker compose and Dockerfiles, Chef cookbooks and BMC Cloud Lifecycle Management (CLM) blueprints for both public cloud and on-premises datacenters. These templates describe the cloud resources, their relationships and configurations. They are used to easily provision infrastructure and applications since these templates represent the single source of truth. They are also used for version control; to track and make changes to infrastructure and applications in a predictable, governed manner, often integrated with development tools. These benefits are the key reasons that infrastructure as code is being widely adopted.
2. *Source repository, peer review and test*
   Next, Infrastructure as Code is kept in a version control system, such as Git, where it is versioned, under change control, tracked, peer reviewed and tested just like application software. This will increase traceability and visibility into changes, as well as provide collaborative means to manage the infrastructure with peer reviews.

Example: If operations wants to roll out a change to the production infrastructure, operations does not need to do it through a console directly in production, as traditionally done in IT datacenters. Instead, operations can create a pull request on "infrastructure as code" Git artifacts with peer reviews conducted on the changes and then the code is deployed to production. This review process ensures higher quality of infrastructure changes as multiple team members have visibility into the changes and can assess the impact of the changes. It also enables "testing" of the changes early in the cycle. Version controlled infrastructure changes also allow easy rollback to a prior infrastructure version.

3. *Follow the DevOps pipeline process*
   Infrastructure as Code templates go through the DevOps pipeline just like application code and gets deployed to production. The DevOps pipeline allows the infrastructure change delivery and governance to ensure that changes are tested and deployed in a controlled manner *before* moving to production environments. At each stage of the DevOps pipeline, these templates are used to provision or update "environments" such as Dev, QA and Production, rapidly creating and de-provisioning dynamic infrastructure.Example: In AWS clouds, operations will do pull requests on CloudFormation templates to make changes to configuration parameters or AWS resources. These changes flow through lower environments such as Dev and Test, and are fully exercised. This ensures a higher confidence that changes will not adversely impact the application when these changes are promoted to the production environment.
4. *Support Immutability*
   Finally, Infrastructure as Code also supports server and container immutability. Prior to immutable infrastructure paradigms, operations teams would manage infrastructure manually, by updating or patching software, adding software package dependencies, changing configurations and so on. This resulted in inconsistent infrastructure not only across development, test and production environments, but also within each of these environments. Inconsistent infrastructure makes troubleshooting difficult. It also means that the infrastructure is not easily scaled, updated or automated to achieve efficiencies in operations. With immutable infrastructure, operations engineers treat infrastructure as disposable. They don't make changes to infrastructure such as servers or containers directly in production. Instead, they go through a full DevOps pipeline to create new server or container images through the DevOps pipeline, and then deploy into production to replace running servers or containers.

This allows consistency of infrastructure in environments that facilitates automation in DevOps, auto-scaling and remediation.

Why DevOps Should Embrace Infrastructure as Code Principles

By following these best practices, Infrastructure as Code can be successfully used for both cloud-native and on-premises application delivery. Developers can specify infrastructure as a part of their application code and manage it all in a single repository. This keeps the full application stack code, definition, testing and delivery logically connected, resulting in better agility, consistency, automation and autonomy for developers for full-stack provisioning and operations.

For the operations team, best practices for software development are also applied to infrastructure, which helps to drive improved automation and governance, stability and quality without negatively impacting agility. Improvements in stability and quality can be attributed to following a DevOps pipeline with versioning, early testing of infrastructure, peer reviews and collaborative process — just like code. Finally, there is traceability and the ability to easily answer questions, such as:

- What is my current infrastructure?
- Who made infrastructure changes in the past few days?
- Can I roll back the latest configuration change made to my infrastructure?

We strongly believe that using Infrastructure as Code principles in managing application delivery can result in compelling advantages to both developers and operations by increasing agility while maintaining governance.