Configurability for Cloud-Native Applications: Observability and Control

The billowing multi-cloud, with loosely coupled services, requires better observability of live configuration changes and management tools. Here's how to address these challenges.

Executive Summary

Over the last few years, becoming a cloud-native enterprise has become an obsession for many organizations. To facilitate this transition, some enterprises have adopted a hybrid cloud (i.e., public and private) approach for its operational flexibility and greater options of data deployment and use. Multi-cloud adoption is also on the rise to prevent vendor lock-in and to give IT the ability to pick the most suitable offering from the assemblage of public cloud providers.

These transitions are enabled by implementing cloud-native principles like containerized loosely coupled microservices, which leverage cloud infrastructure. Cloud-native support spans infrastructure, platform and services shared across many applications, also referred

Novmber 2020

to as tenants. Finally, agility is derived through automating certain infrastructure elements, primarily configuration and deployment.[1]

In order to reduce application or service downtime, architects choose to design systems where the configuration can be changed at runtime. Here is where IT organizations run the risk of injecting improper or erroneous configurations, which can lead to service outages or abnormal service behavior. To overcome this, IT must continuously monitor system states and corresponding configurations. Control over configuration improves operational efficiency and provides business scalability with speed and automation. It also protects organizations from reputational or financial losses owing to incorrect configurations.

Today, configurability and observability are available as a service, as well as frameworks that can be integrated with core run-the-business applications. However, in the proliferating digital world, configuring and observing tools and solutions abound, each suited for a particular platform and environment. Thus, monitoring becomes complex, particularly when operators switch between multiple monitoring and configuration applications – for example, by switching monitoring between AWS Wavelength, Google Anthos and Microsoft Azure Stack Edge.

The solution lies in providing a single mechanism to observe and manage dynamic configurations across cloud infrastructure providers. This approach can enable IT to apply the do not repeat yourself (DRY) paradigm to configuration, to improve the productivity of the IT staff.[2]

This white paper focuses on how organizations can adopt a controlled configuration, treating configuration as code, and improve the agility and quality of deployments. While going through the challenges, we provide recommendations and a framework that can accelerate cloud-native application deployments and address the aforementioned challenges. These recommendations are based on our experience in deploying hybrid cloud-native applications for our customers by using tools that provide dynamic configuration and control.

Cloud-nativity: A primer

Cloud-native applications are built on the principles of quick and automated deployment – including infrastructure and platform – as well as continuous integration and continuous deployment, the ability to scale up and down, persistent monitoring and automated recovery. Configurability requires an ability to quickly apply and modify configurations to automate the scaling requirements by activating new software instances.

Accurate configurations are essential, since incorrect ones not only affect applications availability and service configuration, but also have an impact on downstream applications and services. When systems are detected as having incorrect configurations, and are not functioning as required, then we need to roll back to the right working combination of configurations. This is difficult when multiple changes are done simultaneously to loosely coupled microservices. Good rollback is made possible by maintaining proper audit records. Again, the challenge here is that each cloud platform provider brings its own tools to configure the software, maintain the history of changes and provide rollback options.

Since organizations are adopting a multi-cloud[3] and/or hybrid strategy, the tools from each platform must be integrated or orchestrated using proprietary methods. However, they can be difficult to maintain and scale. Orchestration tools that provide a unified interface to manage and control this heterogeneous landscape are classified as *imperative* or *declarative* (see additional detail in the next section). A combination of these tools provides the capability to react and manage these configurations. With this single system, organizations are informed of a code change and can then visualize and understand the impact of the change.

The benefits to IT leaders include the ability to first foresee and then overcome a variety of challenges, which start with a means for managing heterogeneous, hybrid cloud environments. This enables them to:

- Quickly and efficiently configure services, through advanced automation, across the entire IT landscape: infrastructure, platform, application and services.
- Continuously monitor and reconcile differences.
- Control the changes and reconciliation process via a customizable workflow.
- Provide proper visualization of changes to understand the impact radius of the change.
- Maintain a history of changes, with the ability to cherry-pick changes and apply them or roll back to a defined working set.
- Migrate data between different environments.

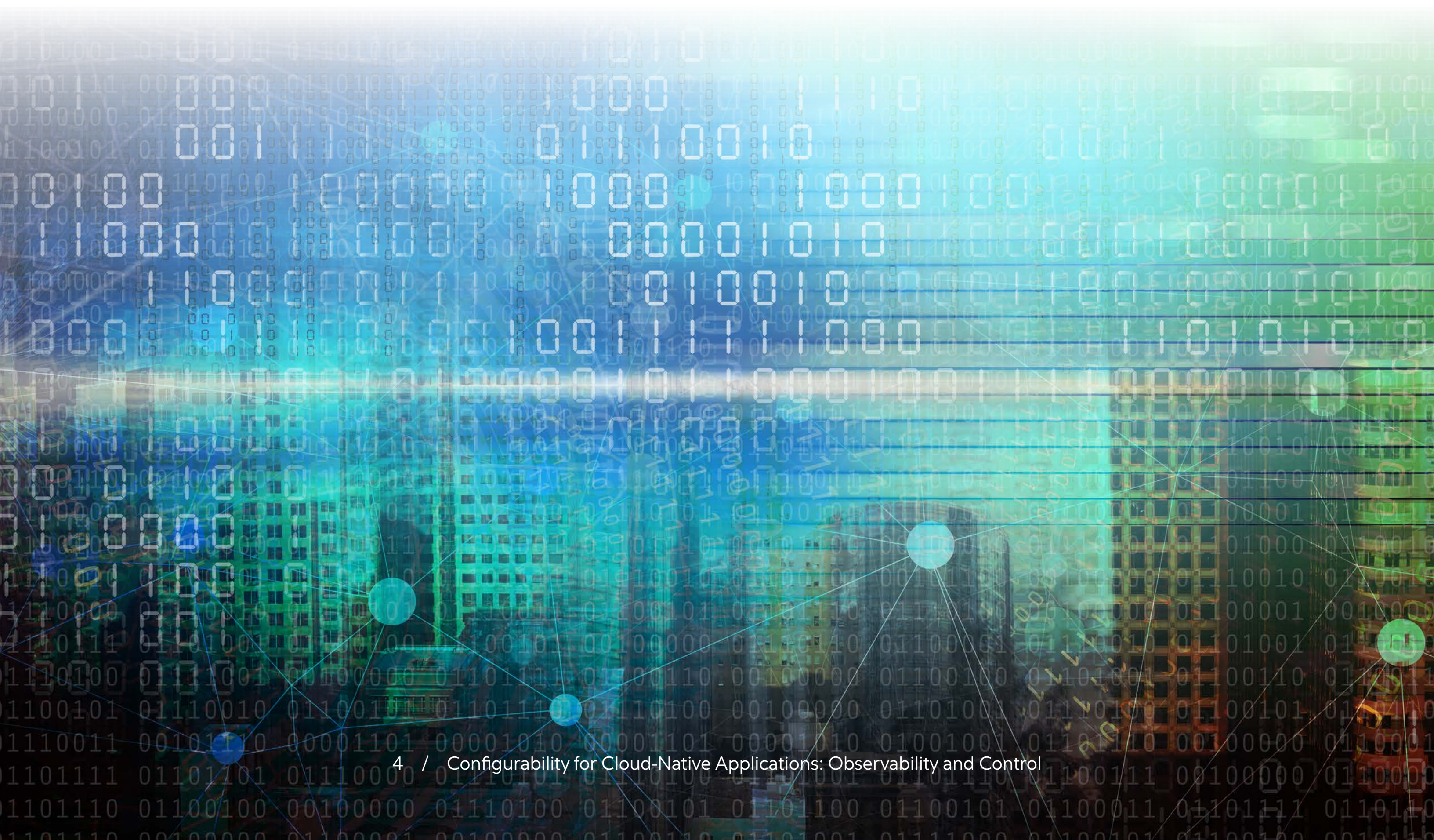
The declarative or imperative tool trade-off

Contemporary DevOps practices emphasize the accomplishment of tasks with code rather than graphical user interfaces. DevOps devotees have progressed from using custom scripts with remote command line interfaces (CLIs)[4] or using infrastructure providers' application program interfaces (APIs) to configure resources.

With the advent of multi-cloud environments, third-party and cross-platform tools have become exceptionally popular. Tools in this category, which include Terraform,[5] Ansible, Chef and Puppet, can be classified as declarative or imperative.[6] An imperative tool has step-by-step instructions on the sequence of deployment and is better suited for the flexibility of deployments offered (e.g., Chef and Ansible).

Declarative tools, on the other hand, provide built-in support that can help IT organizations reach the desired state by just providing configuration values. Declarative tools allow operators to define a desired system state, and always allow current configurations to be compared against it. This approach has certain limitations; mainly, a concrete syntax or schema has to be defined and agreed upon.[7] Sometimes, these packages do not support the required custom configurations or interfaces.

Given this dichotomy, IT organizations need a means for pulling the two categories together into a single platform. Take the case of declarative tools: IT needs an ability to insert certain customizations or functions that are not supported by third-party vendor packages. Hence, declarative tools must be extended for custom configurations or unsupported interfaces. This is because the current declarations do not support our requirement. Once this happens, the entire system – infrastructure, platform, applications and services configuration – can be managed by a single system.



Integration with source control systems for a multiuser environment

In a microservices, multiuser, multiple-applications environment, concurrent changes are executed simultaneously. Observation of these changes requires an ability to differentiate and protect changes made in each of these sessions. In order to manage the configuration changes in a multiuser environment, it is necessary to maintain it as a version of configuration files (versioned artifact).

In order to maintain configuration versions, the various system and application configurations need to be stored in a version control system. Once configuration versions are checked, IT must maintain the configuration written in a code format so it can utilize all the features of a version control system (i.e., difference, patch, merge, etc.). There are two approaches when code is used to represent configurations – infrastructure as code (IaC) and configuration as code (CaC). IaC represents configurations for virtual machines, networks, storage, etc., whereas CaC represents the configurations for applications, servers, jobs, etc.

CaC is a set of prescriptions that allow configuration changes to be written once and then applied with DRY principles to avoid repetition and improve productivity. IT organizations must consider merging IaC and CaC in a single system. In order to ensure a common tool for management, the same language must be used to describe all configurations. Many languages are used to describe configurations, such as HCL, YAML and JSON. Each allows the creation of a controlled approval workflow similar to merge requests used for code reviews and to baseline their impact. This also allows IT to patch or roll back changes in case they are needed. Once a history of changes is established, audits can be conducted in the event the environment is integrated with control tools.

Visualizations and impact radius

In the age of loosely coupled cloud applications, it is necessary – and often challenging – to figure out the impact of change. Two categories of tools are available – generic and domain-specific visualizations. Generic tools include the text diff provided by version control tools; some provide reference to related systems that use the same configuration and hence might malfunction. Blast radius is one such tool; it highlights areas of impact using the dependencies within the configuration.[8]

Beyond this, organizations must create domain-specific visualizations. These are custom visualizations that can provide operators specific

There are two approaches when code is used to represent configurations – infrastructure as code (IaC) and configuration as code (CaC). IaC represents configurations for virtual machines, networks, storage, etc., whereas CaC represents the configurations for applications, servers, jobs, etc.

views in a monitoring dashboard. For multiple systems operators (delivering video over wired or wireless connections, for example), it is important to configure programs and to channel lineups by different regions or geographies. If one channel source is changed, it will be useful to visualize where various packages and programs are impacted.

Security and compliance

These tools are automated and require authorization tokens.[9] Tokens must comply with organization practices and standards. The security and management of these tokens must be a top priority. This can be achieved by integrating with secret management solutions such as Hashicorp VAULT. This handles the secret sprawl.[10]

Multi-cloud support

As organizations embrace a hybrid or multi-cloud strategy for cloud-native applications, infrastructure must be spread across multiple clouds to ensure multiple processing pipelines. Cloud platform providers often don't offer tools that allow interaction with other platforms. Doing so requires an abstraction of functionality offered by each of the clouds. Therefore, organizations need platform-agnostic tools to achieve this, such as Terraform, Chef and Puppet.

Quick Take

Case in Point: Config as Code Accelerator

Our reference architecture and an accelerator (see Figure 1 , page 10) can solve many of the aforementioned challenges. It contains all required integrations and has been deployed in customer locations to manage platform and infrastructure configurations. Here is one such instance where the accelerator was modified to solve one of our customer's problem areas.

A global leader in media and entertainment products that power consumer entertainment experiences needed a better way of configuring shared services while onboarding new partners and video service providers. As part of its expanded product portfolio, it offers a suite of component technologies that can be integrated with customer platforms or deployed as an integrated solution for video service providers. These customers handle partner site data, applications and platform configuration data. Most of these configurations are handled by their own operators or are integrated with their partner systems.

As this product portfolio strategy unfolded, it encountered:

- Large cycles while onboarding new operators, which could take weeks to resolve. These need to be shortened to a day. This caused a delay in its go-to-market strategy for partners/operators.
- The need among IT staff for in-depth knowledge of the domain to understand the proper configurations and apply the right configurations.
- Failures of existing configuration processes of its partners to align with the principles of cloud-native adoption of our client.

With our client, we observed the following:

- A setup was required to separate and store configurations (configuration code, an artifact) of each operator in each of their environments and for each feature they offered.
- The store needs to maintain versioned artifacts.
- The configurations need to be modelled as domain objects.

Quick Take

- The relationships need to be visualized in web pages to understand the impact of changes.
- The changes in configurations need to be observed, reviewed, and approved or rejected.
- Based on the above, a change can continue to stay or be rolled back.

All of these events will enable automation, controlled approval and history preservation of changes.

We integrated the customer's platform services with its config as code accelerator to meet these requirements.

Once config as code was deployed, the following benefits were realized:

- New operators could be onboarded within a day in various environments.
- A configuration prepared for one environment for an operator can be used in multiple environments.
- Configurations change impact can be visualized using domain-specific visualization templates. So the impact of change is known without requiring deep know-how of the application.
- Following cloud-native principles resulted in dynamic configurability that allows applications to scale.
- New configurations are easy to propagate between various environments.

This has improved the stability of the deployment environments, leading to shorter verification cycles and reduced operations effort.

Code as config: Features

The accelerator described above has been extended and can be integrated with any public cloud or custom deployment management tool for managing multi-cloud/hybrid/multi-tenant applications or services. The framework is built along DRY principles and provides the following features:

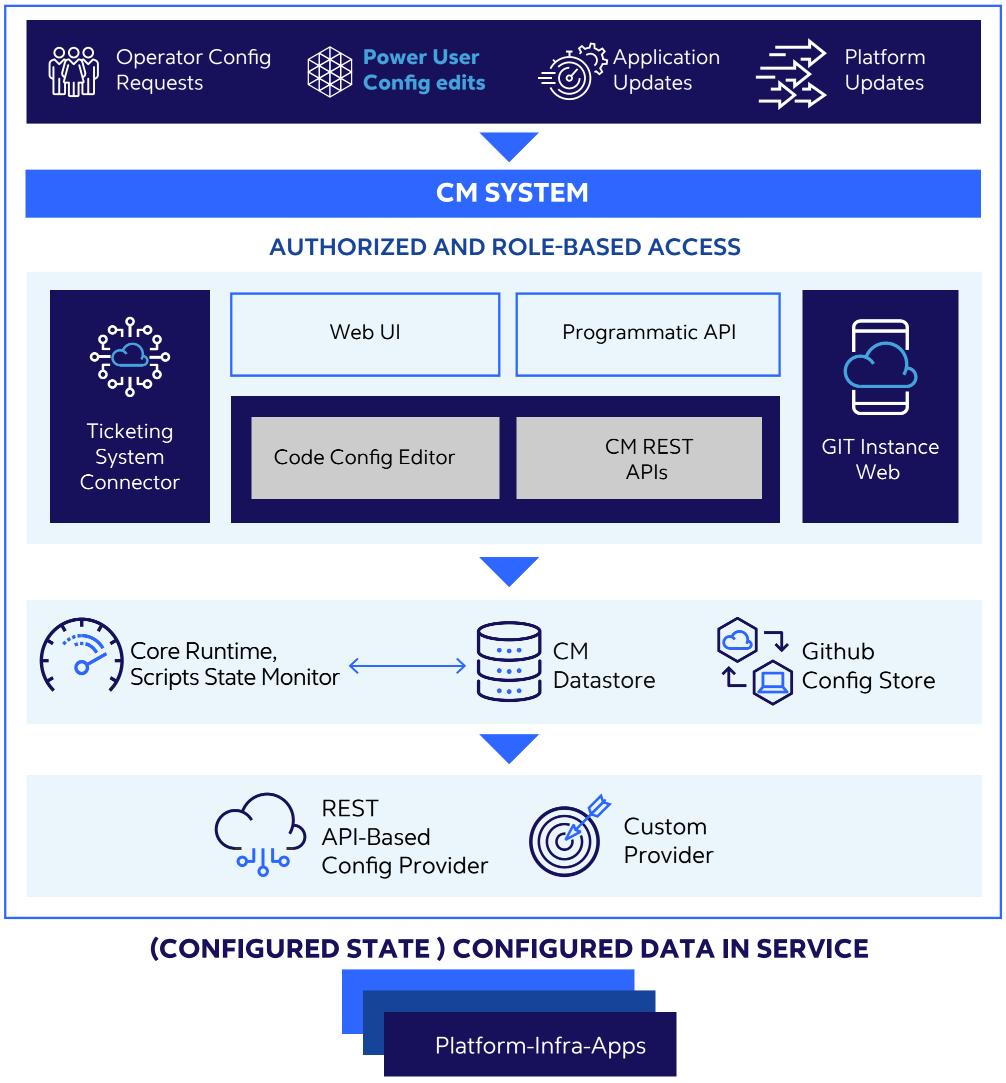
- CaC capabilities.
- A configuration management database that captures the desired and currently configured values for various service environments.
- Methods to observe and approve new configurations on an environment and/or update existing ones after initial system dependencies are set.
- Tracking, comparing to a previously configured state version, and updating of desired and deployed configurations in various environments.
- Secured secret storage and role-based access controls.
- An audit trail/changelog of all approved, rejected, implemented changes with success, failure or rollback states.
- Ways to visualize and compare the configuration across environments.
- Integration with known identity and access management solutions for security assertion markup language (SAML)-based authentication, e.g., Okta for role-based access control.
- Secure storage with a vault.
- Open source software deployed in a Kubernetes environment.

Our approach includes an accelerator that supports and can be integrated with the following capabilities:

- Any platform: AWS, GCP, Azure and Openstack.
- A variety of version control systems for workflow management, such as Github and Gitlabs.
- Many ticketing systems, such as ServiceNow and JIRA.
- Numerous virtual network functions, such as Palo Alto, etc.
- Software such as Splunk and ELK for monitoring, notification and alerts.

This solution generalizes the services offered by configuration management across platforms, thereby abstracting the details and specifics of each cloud platform.

The anatomy of a CaC tool



ADVANTAGES OF THE PROPOSED ARCHITECTURE

- Extensible for integration with infrastructure and host configuration.
- Rich visualizations with deeper insights on impact.
- Configuration as code, approval and versioning.
- Vault-based secrets store.
- Scale with Kubernetes.

Figure 1

Looking forward

Our analysis of the framework's deployment provides insights into the best practices for configuration control in a hybrid/multi-cloud environment.

Rich visualizations of configuration changes based on individual domains help IT realize the impact of change, without understanding the application or service in depth and avoiding dependency on the development teams for support. This has helped in reducing time to productionize applications or services.

A single monitor for observing and controlling the configurations helps operators to react to changing configurations in an informed way. This has led to reduction in errors (which have not been quantified here).

As demonstrated, to drive the agility of deployment of cloud-native applications, services, platforms and infrastructure, with loosely-coupled microservices, and a multi-cloud strategy, IT organizations need tools that behave like code. Tools that provide IaC and CoC aim to cover the full landscape.

Coupled with deep visualizations, version control and secrets management, IT can address all requirements without the need to switch between multiple tools. Passive control of configuration changes – by observing and reacting – ensures that operational agility is not compromised.

Endnotes

1 Tom Grey, "5 principles for cloud-native architecture – what it is and how to master it," June 19, 2019, https://cloud.google.com/blog/products/application-development/5-principles-for-cloud-native-architecture-what-it-is-and-how-to-master-it.

2 https://www.oreilly.com/library/view/97-things-every/9780596809515/ch30.html.

3 https://www.gartner.com/en/conferences/apac/infrastructure-operations-cloud-india/gartner-insights/swg-why-organizations-choose-a-multicloud-strategy.

4 https://www.w3schools.com/whatis/whatis_cli.asp.

5 https://www.terraform.io/docs/modules/composition.html.

6 Tytus Kurek, "Declarative vs Imperative: DevOps done right," Ubuntu, blog, Aug. 6, 2019, https://ubuntu.com/blog/declarative-vs-imperative-devops-done-right.

7 "The architecture of declarative configuration management," Made of Bugs blog, Nov. 12, 2019, https://blog.nelhage.com/post/declarative-configuration-management/.

8 https://github.com/28mm/blast-radius.

9 "Token Based Authentication Made Easy" Auth0, https://auth0.com/learn/token-based-authentication-made-easy/.

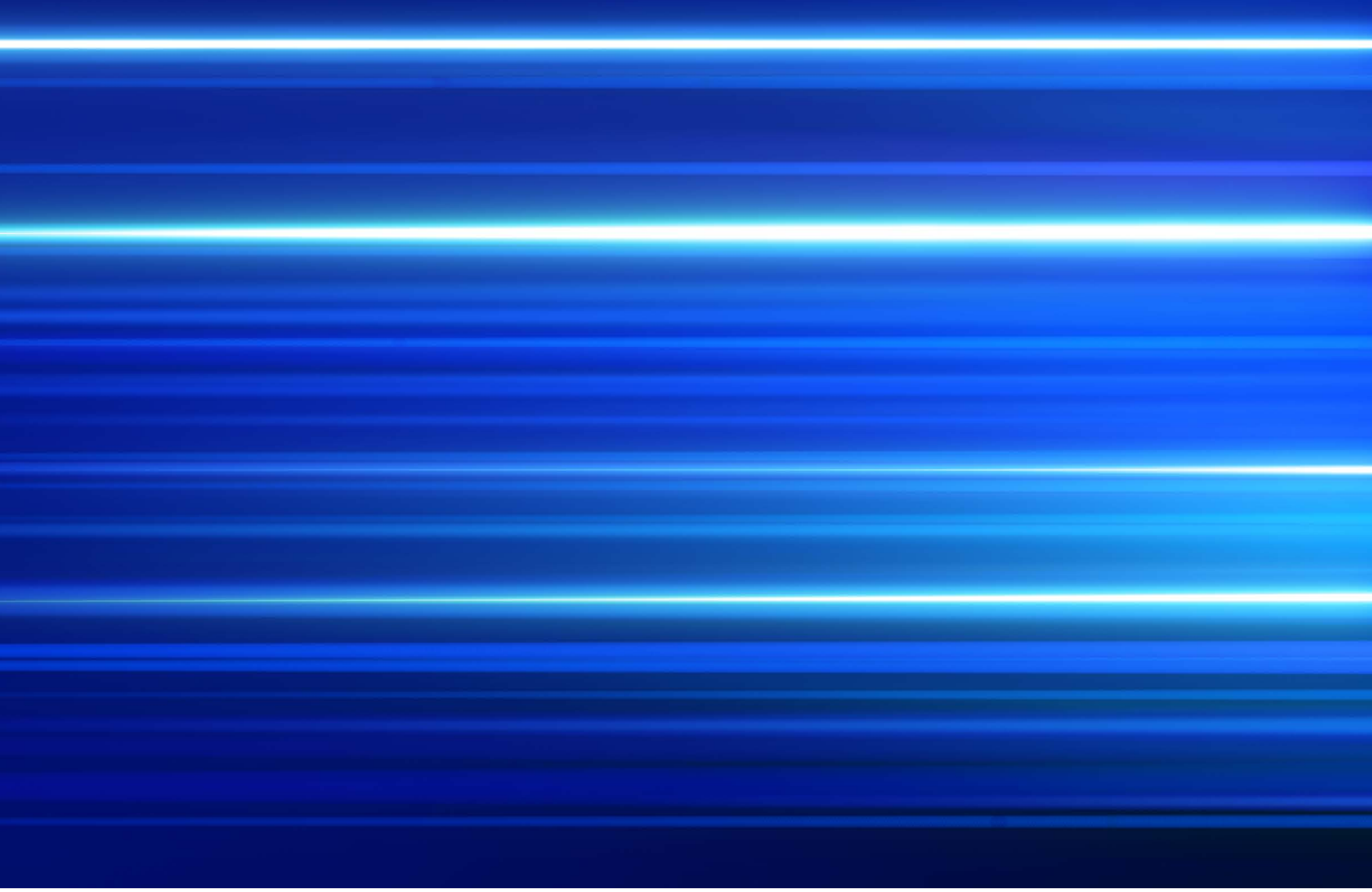
10 Armon Dadgar, "What is 'secret sprawl' and why is it harmful?" https://www.hashicorp.com/resources/what-is-secret-sprawl-why-is-it-harmful/.

About the author

Varadarajan

Domain Architect, Communications, Media & Technology, Cognizant

Varadarajan is a Domain Architect within Cognizant's Communications, Media and Technology business unit. He has over 24 years of development and design experience in IT and related systems. A technology enthusiast, Varadarajan focuses on applying the latest trends across industries in the communications and media domains. He has a B.Tech degree from IIT Kharagpur in India. Varadarajan can be reached at Varadarajan.A@cognizant.com| www.linkedin.com/in/varadarajan/.



About Cognizant Communications, Media & Technology

Cognizant's Communications, Media & Technology (CMT) business unit helps clients transform into people-centric enterprises — enabling organizations to create new business models that deliver more personal and relevant customer experiences. We combine human insights with advanced technology to translate customer needs into differentiated content, products and services that power our clients' futures. We apply domain expertise and digital know-how to help CMT companies optimize performance for today and accelerate digital transformation for tomorrow. Our technology innovations, proven solutions, product and software engineering expertise, creative interactive prowess and global delivery excellence enable businesses to scale to meet the needs of the market. Visit us at www.cognizant.com/cmt-solutions.

About Cognizant

Cognizant (Nasdaq-100: CTSH) is one of the world's leading professional services companies, transforming clients' business, operating and technology models for the digital era. Our unique industry-based, consultative approach helps clients envision, build and run more innovative and efficient businesses. Headquartered in the U.S., Cognizant is ranked 194 on the Fortune 500 and is consistently listed among the most admired companies in the world. Learn how Cognizant helps clients lead with digital at www.cognizant.com or follow us @Cognizant.

Cognizant

World Headquarters	European Headquarters	India Operations Headquarters	APAC Headquarters
500 Frank W. Burr Blvd.	1 Kingdom Street	#5/535 Old Mahabalipuram Road	1 Changi Business Park Crescent,
Teaneck, NJ 07666 USA	Paddington Central	Okkiyam Pettai, Thoraipakkam	Plaza 8@CBP # 07-04/05/06,
Phone: +1 201 801 0233	London W2 6BD England	Chennai, 600 096 India	Tower A, Singapore 486025
Fax: +1 201 801 0243	Phone: +44 (0) 20 7297 7600	Phone: +91 (0) 44 4209 6000	Phone: + 65 6812 4051
Toll Free: +1 888 937 3277	Fax: +44 (0) 20 7121 0102	Fax: +91 (0) 44 4209 6060	Fax: + 65 6324 4051

Codex 5971