White paper
Cisco public

Data Center Automation with Infrastructure as Code

Contents

Overcoming the human and technical challenges to achieve an effective automation implementation

Executive summary

Today, automation is becoming a must to cope with the demands of rapid changes and updates to data center elements such as network, storage, compute, security, and more. This comes with various challenges that relate to technology and people and processes.

One of the most complex infrastructure domains to automate is the data center, as automation of the entire IT infrastructure includes approvals, requests, and other IT Infrastructure Library (ITIL®) related processes. It spans across multiple silos such as networking, compute and virtualization, cloud, security, etc. Automating these components requires a comprehensive understanding of their interdependencies and the ability to orchestrate and manage them effectively. In addition, many companies have these technology domains as separate organizations, each with its own management team, and sometimes they may even have different high-level goals and objectives. The network team will, of course, have different objectives than the compute team at a direct level, but the two teams' "north star" ideally should be the same. It may also involve discussions such as moving from the User Interface and Command-Line Interface (UI/CLI) to DevOps, but other aspects, such as platform engineering, could benefit from this white paper as well.

This white paper dives not only into the technical challenges but also into practical experiences and lessons learned from automation projects with multiple customers. It will go into the organizational challenges, such as siloed organizations and personnel not being comfortable with automation and digitization within the company. Very commonly automation also must navigate security considerations, which can be complex and challenging to satisfy.

We will address how these challenges can be overcome, as well as describe different ways that the automation can be approached, depending on where the company is on its automation journey.

Introduction

Data centers are complex environments made up of multiple technologies, often under the responsibility of different teams, which can very quickly lead to silos within an organization. Adding to the structural and internal complexity are software and firmware versions, change frequencies, and the use of multiple vendors. The data center can also suffer from organic growth as part of expansions and acquisitions, or just technology refresh cycles. The scale itself, and the number of changes to be made, are strong drivers for introducing automation to help process all the changes required to run the business, but other drivers involve changes at scale, agility, reliability, configuration consistency, reduction of human errors, maintenance of a history log of network changes—and the list continues.

The value of automation can be broken into the following measurable categories.

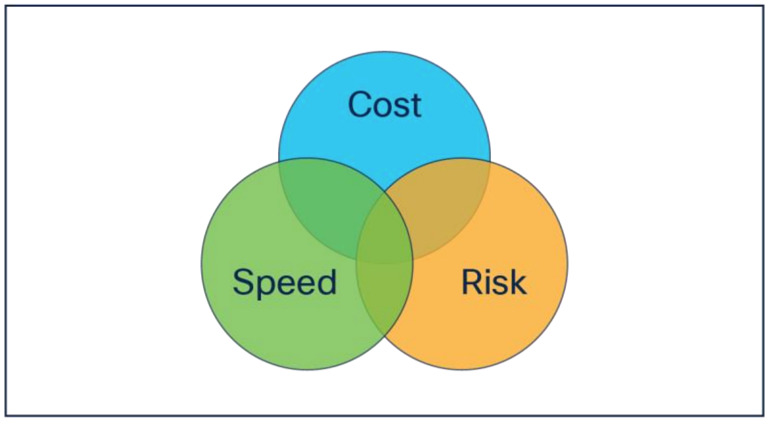

Figure 1.
Infrastructure as code value

Cost reduction is related not only to the financial cost of operating and managing the data center but also to the cost in terms of personnel and efforts required. Automating tasks previously performed manually enables people to refocus their efforts on other tasks that bring value to the company.

Speed improvements with the use of automation relate not only to faster configuration of the infrastructure, but also to the ability to provide visibility and predict how long a given task will take. Such visibility and predictability help other teams within the company to work more efficiently.

Risk relates to the removal of human errors such as misconfigurations, etc. This in turn decreases the risk of outages and increases the reliability of the infrastructure. These outcomes can help companies move toward a culture of DevOps and further enhance and expand the role of automation.

Often automation is thought of in the context of configuring things, but it is equally important to consider automation when deprovisioning things. Automation can help to implement full lifecycle management of the infrastructure configuration to avoid situations where a configuration is left behind due to questions like "What's that do?" that are answered with "We're not sure, but we're afraid to remove it because it may break something."

Similarly, another key benefit of automation is the ability to perform a rollback in the event of an unsuccessful change, as the automation system will be able to clean up the bad configuration faster than any engineer is able to—especially in the case of a complex change involving multiple devices.

There are also some challenges that could be highlighted, such as:

- The need to educate/upskill the staff
- The cost to maintain and update the automation framework, such as scripts, tools, etc.
- The time required to implement new automation tasks

However, in our experience, these challenges do not outweigh the benefits of automation.

Understanding the data center ecosystem

Introducing automation into a data center can be challenging, as there are many factors that can slow down the adoption. To better evaluate the problem of introducing automation, we introduce the following model, to provide an understanding of the various parts of a data center.

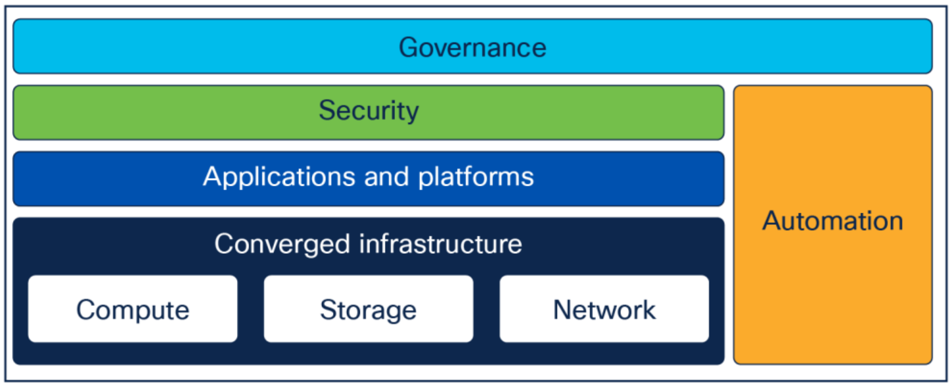

Figure 2.
Data center model

This figure identifies how the various areas of a data center interlock. The data center consists of the following areas, (top to bottom):

- Governance
- Security
- Applications and platforms
- Automation
- Converged infrastructure (compute, storage, network)

Using this model when automating a solution will help you understand how everything interacts and must work in unison to make the solution successful. The following example illustrates this with the use case of a simplified way of deploying a Kubernetes cluster using automation:

- Governance to approve the process
- Allocate a network/subnet
- Allocate compute resources
- Allocate persistent storage
- Deploy a Kubernetes controller and worker nodes
- Create a Kubernetes cluster
- Update security rules to allow communication to and from the cluster

It is not uncommon for a use case to involve multiple departments, highlighting technical and nontechnical challenges as previously stated.

For automation to be successful, it must effectively bridge organizational silos and seamlessly integrate technology across different domains. First, we will explore the technical side, with DevOps and Continuous Integration/Continuous Delivery (CI/CD) and explain why this approach is becoming the most common for automation, one reason being that CI/CD pipelines enable shorter time to market for new features, creating quicker results and thus lowering the strain on development. Later in this white paper will we elaborate on the organizational challenges and constraints associated with the execution of automation projects.

Approaches to automation

One of the initial drivers that causes organizations to look at automation is the ability to reduce the time it takes to complete a given task. While there are multiple ways to automate a configuration task, not all of them necessarily give the same return when it comes to time savings.

The following illustration shows different approaches to operate a network and the time it takes to execute a given task across a large number of devices.

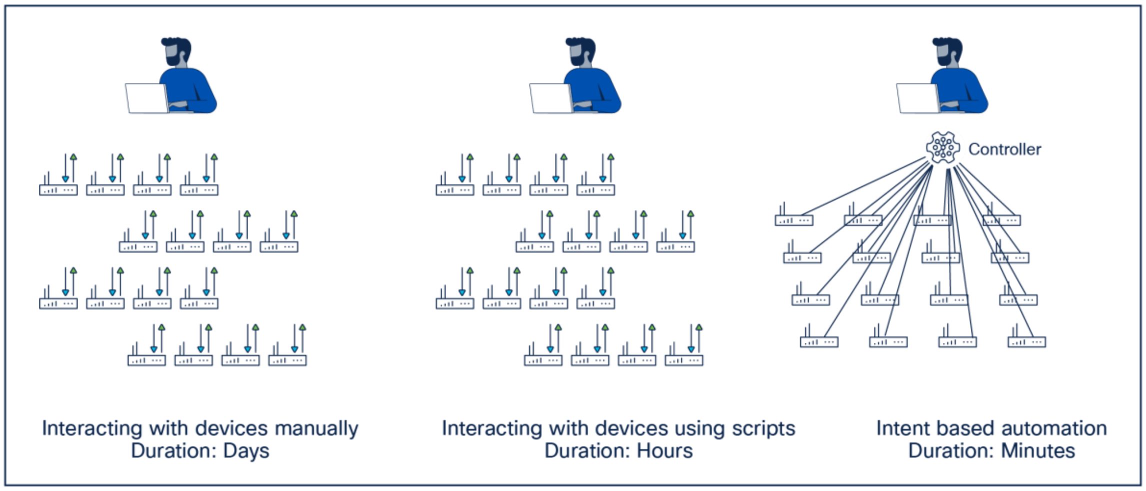

Figure 3.
Approaches to infrastructure automation

A common starting point for automation is to create standalone scripts that automate the execution of a given task by automating the interaction needed toward each network element. While this gives an immediate increase in speed for that task, it may not give the rest of the organization much improvement in speed, as such scripts are not often shared throughout the organization. Knowledge about how such scripts work is also typically confined to the author who created them and tends to lack documentation. This results in tribal knowledge and introduces risk if the author leaves the company, as the scripts are often stored by the author without proper controls or repositories.

An alternative approach to automation is the introduction of a controller-based architecture. In such architectures, the controller is responsible for translating the intended configuration provided by the engineer to a device-level configuration for each of the devices it manages. The controller in this context can be anything from an integral part of the infrastructure architecture, such as the Cisco® Application Policy Infrastructure Controller (APIC) [1], Cisco Intersight® [2], Nutanix Prism [3], or Red Hat Ansible Automation Platform [4] to an automation solution developed in-house that automates the infrastructure in question.

Even when the infrastructure architecture has a centralized controller, it often makes sense to build an automation solution using the controller API to eliminate manual configuration through the controller's user interface. Such a solution can provide an abstraction that simplifies the work required to specify the intended configuration.

It is, however, unlikely that you will find a single controller that spans the breadth of all technologies in the data center. Instead, it is often better to ensure that all technology domains follow the same principles, often referred to as a design pattern. This gives each technology domain enough flexibility to implement the optimal solution for its needs while ensuring some level of consistency in terms of a way of working for engineers across technology domains.

Infrastructure as code and DevOps

As described in the previous section, one can have multiple approaches and design patterns to choose from when it comes to data center automation. Infrastructure as code (IaC) is one of these approaches. It is centered around managing the infrastructure configuration through machine-readable definition files rather than directly through interactive configuration tools like a CLI, GUI, etc. provided by the underlying hardware and software solutions.

IaC is applicable in the scenario where the infrastructure devices are configured directly, as well as when the infrastructure itself provides a controller that acts as single point of entry for configuration tasks.

Operate configuration as code

A key concept with IaC is the use of machine-readable definition files used to store the configuration. While multiple file formats exist for such definition files (JSON, XML, YAML, etc.), it is important that these are also human readable.

The YAML definition file shown in Figure 4 is defining the Kubernetes cluster, which describes not only the cluster itself but also the characteristics of the virtual machines on which it runs.

```
---
kubernetes_general:
  k8s_cluster_name: ”cluster01"
  guest_id: ubuntu64Guest
  guest_memory: 8192
  guest_vcpu: 4
  guest_template: ”ubuntu-2004-hx”

kubernetes_controller:
  hosts:
    cluster01-controller-1:
      name: "cluster01-controller-1"
      network_interfaces:
        - name: ”cluster01"
          ip: "172.20.213.3"
          netmask: "255.255.255.224"
      disk_layout:
        - size_gb: "150"

kubernetes_worker:
  hosts:
    pod1-worker-1:
      name: "pod1-worker-1"
      network_interfaces:
        - name: "pod1"
          ip: "172.20.213.6"
          netmask: "255.255.255.224"
      disk_layout:
        - size_gb: "150”
```

Figure 4.
Sample YAML code for IaC

As illustrated in the figure, YAML is human readable and therefore easy to read and update. This is one of the reasons why YAML is used in many IaC implementations.

Describing the desired infrastructure configuration in human and machine-readable definition files is, however, only one piece of the puzzle in an IaC implementation. It is also necessary to define and implement the engine that will consume these definitions and translate them into device-level configurations that can be pushed to the infrastructure.

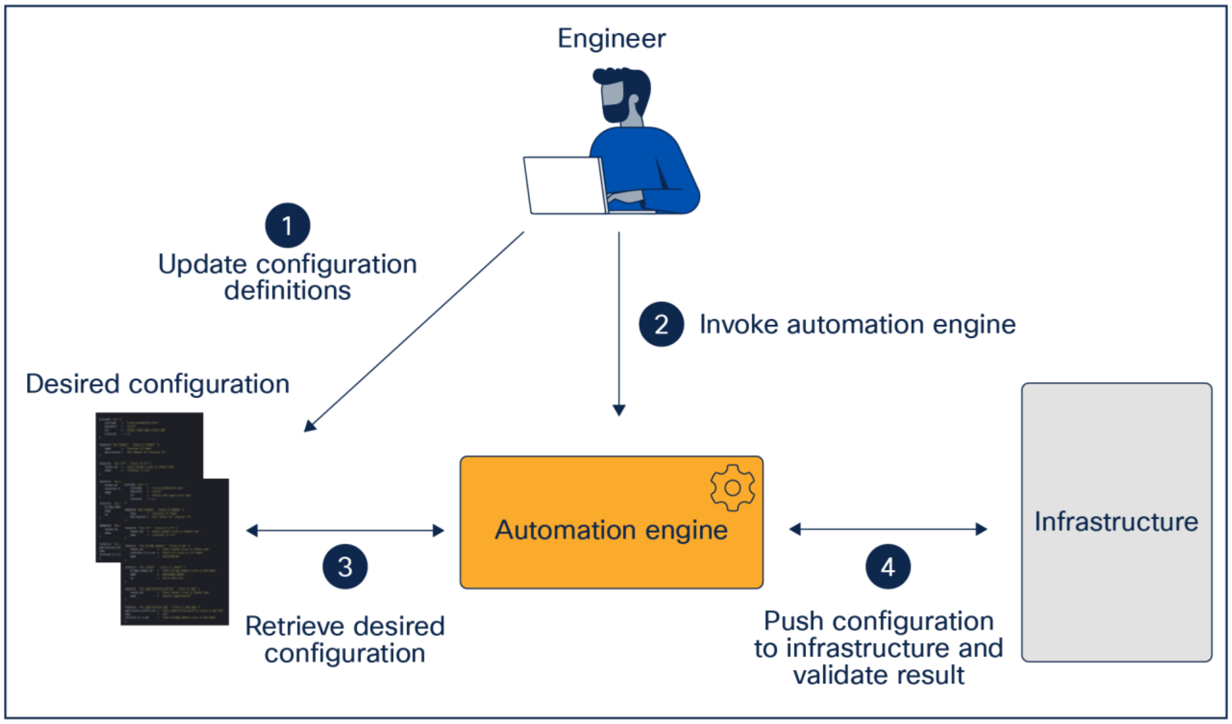
Engineer
1
Update configuration definitions
2
Invoke automation engine
Desired configuration
Automation engine
Infrastructure
3
Retrieve desired configuration
4
Push configuration to infrastructure and validate result

Figure 5.
IaC automation engine

The automation engine is responsible for the following tasks:

- Ensuring that the automation inputs are valid, from both a syntax and semantics point of view, with the goals of catching errors prior to pushing the configuration to the network. This is often referred to as pre-change validation, which may, depending on the implementation, be leveraging the target's API.
- Translating the desired configuration into device-level configurations that can be pushed to the infrastructure.
- Verifying that the desired configuration has been successfully applied to the infrastructure without error. This is often referred to as post-change validation, which will be leveraging the target's API.

When it comes to the implementation of the automation engine, there are several fundamentally different approaches that can be chosen:

- Imperative/procedural vs. declarative
- Stateful vs. stateless

The **imperative/procedural** approach specifies the sequence of steps required to achieve a certain outcome—i.e., do task A, do task B, verify task C, and do task D, in that exact order. Such an approach is not ideal for configuration tasks, as the described procedure often assumes a given state of the infrastructure when starting the workflow. As an example, how would the workflow detect that task A has already been completed and therefore can be skipped—especially if the operation in task A is not idempotent, meaning that if it is run again for the same target it will not create the same results. The imperative/procedural approach to automation is often required for use cases like software upgrades, as these typically need a specific sequence of tasks to be executed in a predetermined sequence of steps and validations.

The **declarative** approach to automation instead focuses on describing the desired state for the infrastructure and leaves it up to the automation system to derive the changes and sequence of actions that need to occur in order for the infrastructure to have the desired state. A declarative implementation is ideally suited for managing infrastructure configuration.

With a **stateless** approach to automation, the automation solution does not have any knowledge about what it has done previously. Good candidates for automation tasks using a stateless approach are software upgrades, etc., where only an indication of success or failure after the execution is needed.

A **stateful** approach to automation, on the other hand, stores the state of the infrastructure between execution runs, allowing for detection of configuration changes performed outside of the automation framework—also known as configuration drift. Stateful implementations of automation are typically seen when the automation solution is used to manage the configuration of the infrastructure.

For the implementation of the automation system, multiple tools and solutions are typically used. Popular choices for the configuration management elements include:

- RedHat Ansible
- HashiCorp Terraform

Ansible [5] is an automation tool from RedHat that simplifies the management and configuration of a wide range of systems. it uses YAML to define the order in which the tasks should be executed on the target devices to achieve the desired configuration. Ansible does not require any agents to be installed on the target systems. It is based on an imperative/procedural approach to automation in the sense that you define a sequence of tasks that are executed in the specified sequence, but it is possible to achieve declarative behavior with Ansible. This does, however, required special care and considerations when writing the Ansible Playbooks and comes with some additional complexity—especially if state is required between executions to achieve the declarative behavior, as Ansible is stateless by nature.

Terraform [6] is an infrastructure automation tool from HashiCorp that enables a wide range of infrastructure resources to be configured based on a declarative intent definition. This definition is specified using a human-readable format called HashiCorp Configuration Language (HCL). Once the desired state is specified, Terraform will automatically determine the configuration additions, updates, and removals required in the infrastructure to ensure that it matches the defined intent. Terraform keeps state between executions, and this state information is used when determining the actions required to achieve the defined intent.

Ansible and Terraform each have their strengths and weaknesses, which means there often is not one tool that fits all automation use cases. Ansible is often used to implement use cases like software upgrades of network devices or to manage the lifecycle of software packages on server workloads, while Terraform is used to manage the lifecycle of infrastructure configurations like networks/VLANs, deployment of VMs, etc.

DevOps and CI/CD

IaC provides the technical implementation of the automation solution using machine-readable definition files. This ability to treat configuration as code enables organizations to evolve the way they are working using DevOps principles.

DevOps introduces a paradigm shift from infrequent large changes to frequent small changes that allow for real-time course corrections based on business and customer feedback.

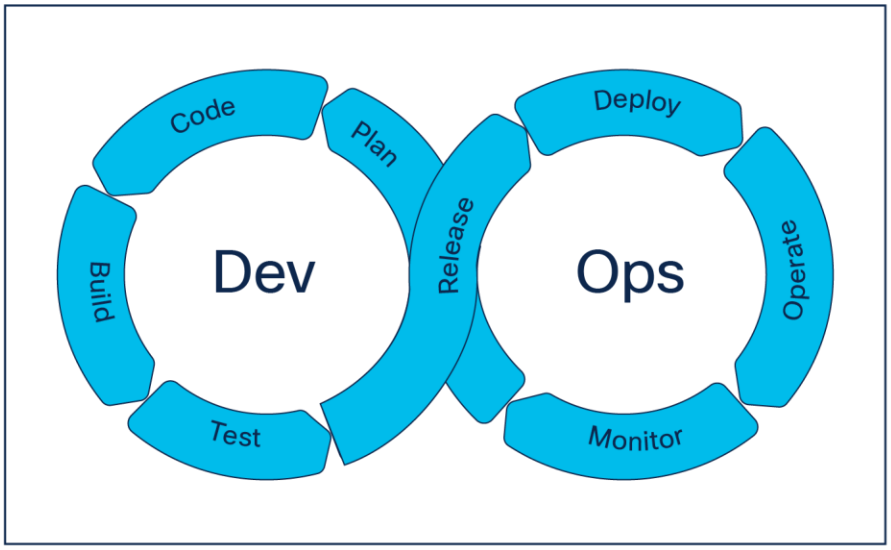

Figure 6.
DevOps model

Another key element in the DevOps paradigm shift is the use of practices known from the software development world for infrastructure management. With this, we mean the adoption of version control, automated testing to ensure the success of a planned change both before and after the change, as well as automatic deployment of the change using automation.

Standardization plays a vital role in the effectiveness of automation. Complex customizations are not feasible in automated tasks. An obstacle often encountered in automation projects is the extensive range of customization options provided for each IT service offering. Standardizing the service offerings into a limited number of flavors significantly increases the probability of successful automation.

A concrete example is to provision a virtual machine. If we were to list all the possible customization options, the list would be endless and impossible to realize into code. Instead, we could agree on a set of configuration flavors such as the ones listed in Table 1.

Table 1. Sample sizes for virtual machines

Size	Memory	vCPUs	Storage
XL	128 GB	20	20 TB
L	64 GB	16	10 TB
M	32 GB	16	1 TB
S	16 GB	8	10 GB
XS	8 GB	4	2 GB

If we were to map all existing workloads into these five configuration flavors, automation would be much easier, without the need for custom approaches.

Source version control system

When operating an environment that is defined by IaC, a source version control system is a must. The de facto standard for source version control in today's world is Git [7]. Git is an open-source tool for version control that operates in a distributed manner, where the change history is stored both in the central repository and on the local copy of the repository on each engineer's workstation.

When it comes to implementing Git, there are multiple options, such as GitHub, GitLab, Bitbucket, etc. In an IaC implementation, Git is often used to store the source of truth for the intended configuration. This configuration is then what is pushed to the infrastructure using tools like Ansible or Terraform.

Continuous Integration/Continuous Delivery (CI/CD)

In the context of IaC and DevOps, CI/CD can be used as the process orchestrator that connects the different elements together using one or more pipelines. Several pipeline tools are available; some commonly used ones include Jenkins, GitHub Actions, and GitLab CI.

In an infrastructure automation solution, the pipeline is responsible for executing the different automation tasks in the correct sequence. The following figure shows an example of a network infrastructure pipeline. The example pipeline is taken from the Nexus as Code [8] solution available on Cisco DevNet.

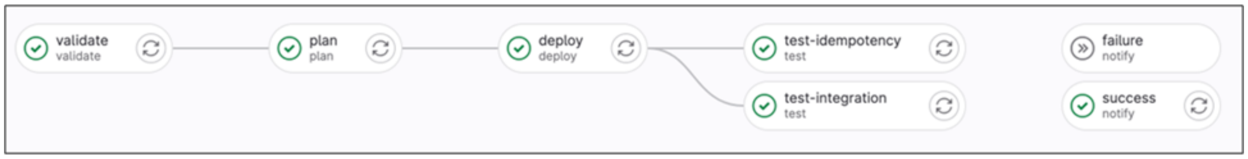

Figure 7.
Network infrastructure CI/CD pipeline example

If this had been a sample pipeline for other data center technologies such as virtual machine deployment, firewall configuration, storage provisioning, etc. it would look very similar.

The validate stage ensures that the desired state is defined using the correct syntax and, where possible, also validates the semantics of the defined state. In other words, this stage aims at catching errors prior to deploying the configuration to the production network.

The plan stage determines the actions (add, modify, delete) that are required in the infrastructure to match the desired state. The output of this stage can be used as input into the change management process required to approve the configuration change (if applicable).

The deploy stage modifies the configuration of the infrastructure to ensure that it matches the desired state. Depending on the nature of the change and the maturity of automation within the organization, this stage may be executed after an automatic change approval, or it may be triggered manually after an approval of the change.

The two test stages (test-idempotency and test-integration) are executed after the deploy stage to ensure that the required changes have been successfully applied to the infrastructure. These stages are typically executed using automated testing tools that inspect things like interface status, error messages, etc.

The final stage in an infrastructure pipeline is the notification stage, where the success or failure of the pipeline is communicated. Such notification can be done in multiple ways, depending on the maturity and desire of the organization, but commonly used methods include instant messaging, email, etc.

People and process challenges when adopting automation

Technical and human challenges

This section explores in more details the various types of organizational challenges that we have experienced.

The complexity of a data center

As described earlier, the data center is complex due to its wide range of technologies, platforms, and vendors. In addition, it is an area with a high frequency of changes such as add/modify VM, add/modify ACL, attach switch port to network segment, etc. This can pose constraints to automation initiatives, making it challenging to ensure compatibility, standardize processes, and adapt to evolving technology.

The human factor

People in general do not feel comfortable embracing change. Change is often seen as scary, or there is a lack of understanding of why it is needed. The resistance to change when introducing automation is a very key factor to address to achieve success. One way to help motivate the adoption of automation is to address the benefits, such as:

- Fewer human errors
- Consistency of deployments
- Easy rollback
- Risk reduction
- Automated testing and validation
- Self-documenting infrastructure
- Complete audit trail of deployments

Stopping the need to perform boring and repetitive tasks will free up time for more personal development, as the automation is supporting the changes required, or configuration updates. The main driver contributing to resistance to change is the fear of losing one's job or being made obsolete by technology.

The other aspect of IaC in particular is that the whole configuration of devices is kept outside of the device itself in a version control code repository. If you are or have been a developer, this would appear very normal, but a traditional systems/network administrator may be uncomfortable with this approach. However, the repository can be beneficial during an audit, where everything is verified; having the source of truth outside of the devices will greatly support the audit process.

When IaC gets introduced, it is often done in an all-or-nothing approach, since also performing direct configuration on a device would offset the configuration stored in the code repository and lead to configuration drift. Moving the configuration outside of the actual device is by far the biggest change to accept and really trust. However, it is important to make sure that it does not become an all-or-nothing approach. Try to begin by adopting even small automation tasks that feed into the bigger automation strategy.

Organizational challenges

Little kingdoms

To better understand the organizational challenges mentioned previously, we first take a look at the classic hierarchy.

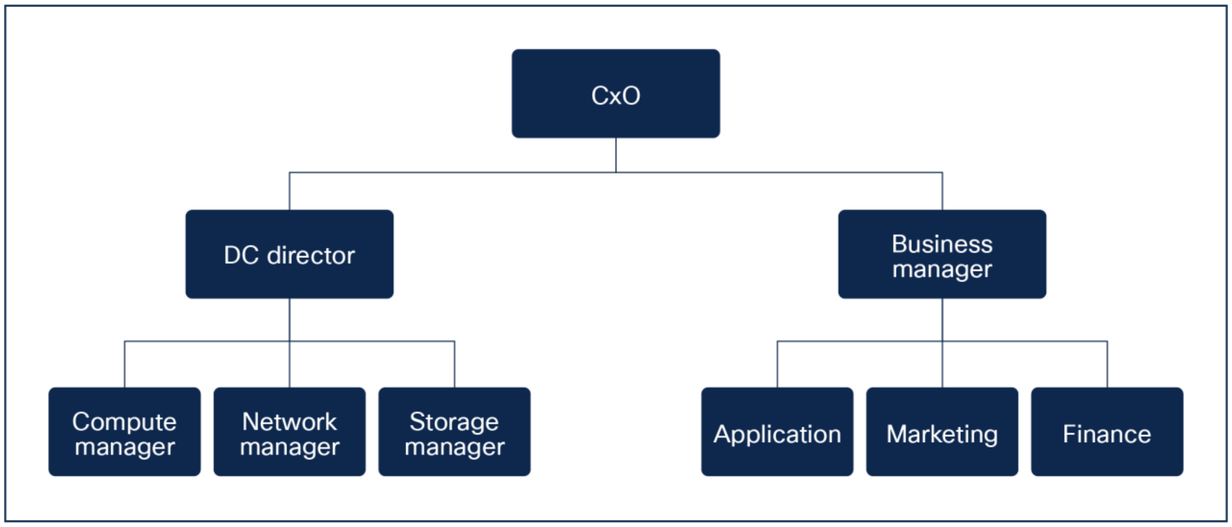

Figure 8.
Classic organizational hierarchy

In the classic hierarchy, the business is separated from the infrastructure services. The infrastructure is then separated into network, compute, and storage, thus making automation a challenge, since it touches multiple areas. If we collapsed this into a more functional organization, it could work better, but it is only one solution of many. Another possible way is to have clear design patterns during an automation implementation, so they can merge ideas and toolsets, resulting in immediate synergies. It is also quite common for security to form its own department, which creates another challenge on top of automation.

What if we took this organization, collapsed the infrastructure, and make a dotted line between the silos? It would then turn into something like the following.

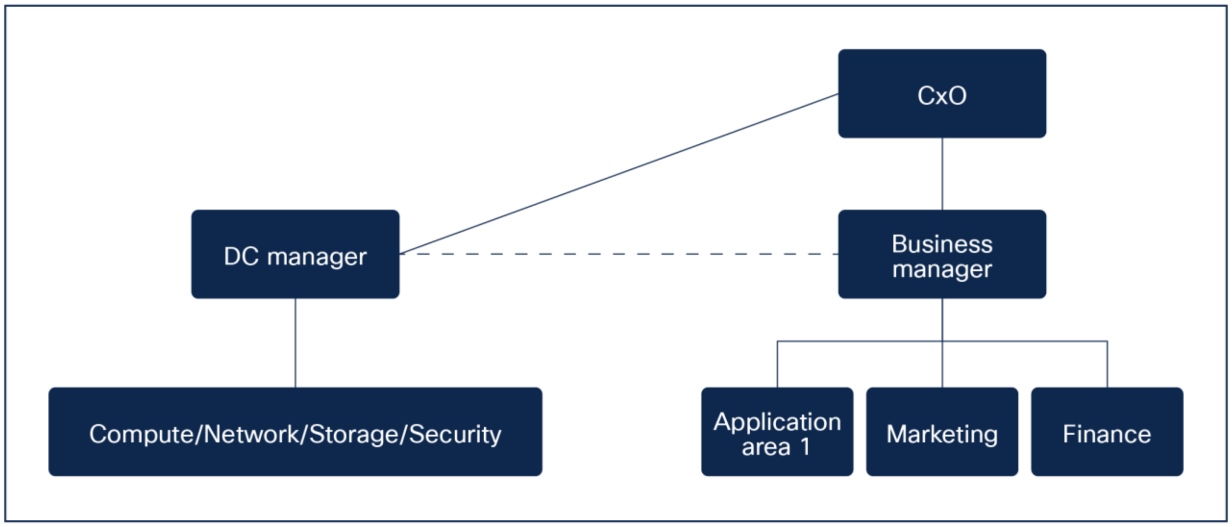

Figure 9.
Updated organization

What could make this scenario more likely to succeed? Notice that we now have a single team responsible for infrastructure services. This could be seen as IT as a service, with a common goal of having the best working private/hybrid cloud. Automation is now easier to implement, as things are now seen as services provided to the business manager. There could then be multiple application areas.

There are other types of siloed organizations that would also challenge automation. This scenario is aimed more at people who are facing the change with fear, which automatically leads to pushback when something is unknown or new. Here the key would be education, and helping people embrace the future by starting with automation of small and noncomplex tasks. Also very important would be to drive smaller automation projects to get some small successes that can later be used to scale up the model. This would also build confidence with the team.

IT as a service provider

Another scenario would be to organize based on complete service transparency. This would mean that the service provided could almost be seen as an external company.

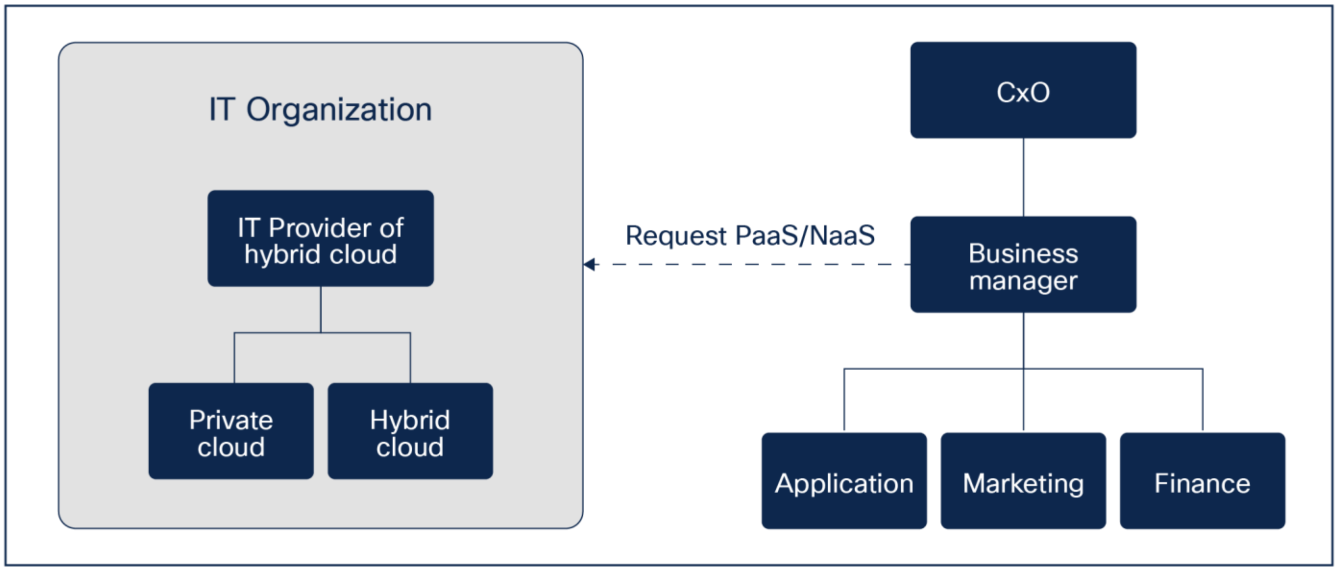

Figure 10.
IT as a service provider

One of the challenges here would be local cost vs. profit. Another important aspect to address in the absence of shared management is ownership and budget allocation. When multiple organizations are involved, determining who should invest in licenses and other resources can be a challenge, particularly in larger enterprises. Resolving this issue is not always easy, but it can be facilitated by engaging an external consultant or vendor who could provide support. By collaborating with the various organizations within the customer, a consultant can foster an environment where there is an incentive to work together toward a unified approach, rather than having separate independent solutions. This allows for a more strategic and cohesive approach to be implemented across the entire organization.

To facilitate this process, it would be advantageous to implement a service catalog. A service catalog serves as a framework that drives the automation of services and clearly outlines the available offerings. By having a well-defined service catalog in place, organizations can proactively prevent issues such as shadow IT (the use of technology or software within an organization without the knowledge, approval, or oversight of the IT department). Additionally, to drive an automation initiative forward, it is beneficial to have a global goal that encompasses IT. When there is a shared objective that includes IT, it becomes easier to implement changes, even if they are taken in small steps. This approach allows for a gradual and systematic implementation of a unified strategy across the organization. It would also show a measured way of success, whether services are repeated or just one-offs.

Squad-based organization

Another scenario we often encounter is the formation of functional squads, consisting of representatives from various areas such as security, network, applications, and more. Squad teams offer several positive aspects that can contribute to their success, including diverse experience, improved collaboration, and efficient decision-making.

While this approach allows for cross-functional collaboration and expertise, the challenge lies in establishing a common goal among the squads. However, with some strategic thinking and setting aside personal emotions, this challenge can be effectively addressed.

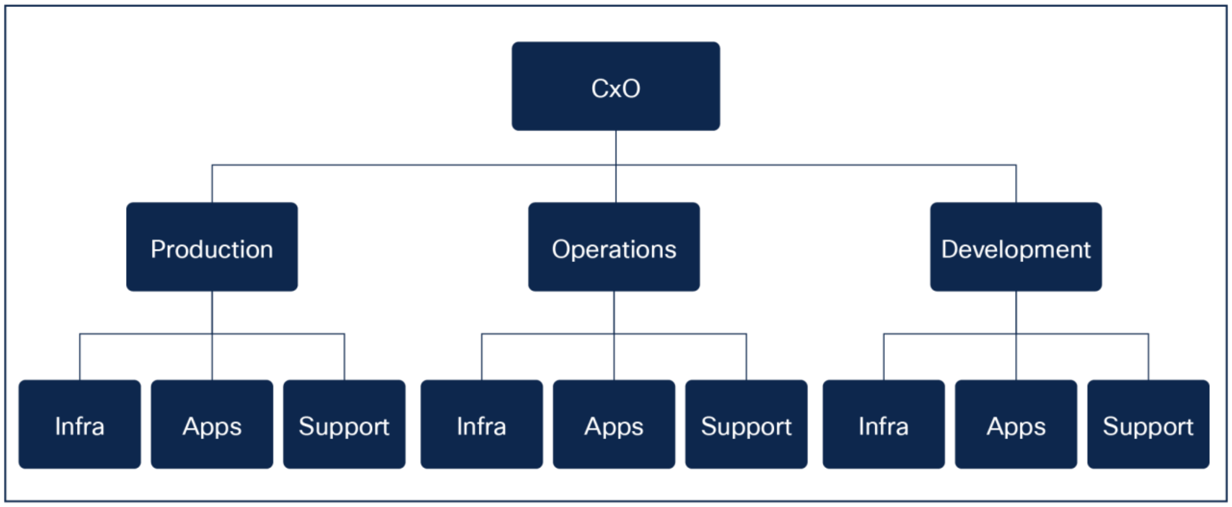

Figure 11.
Squad-based organization

Another challenge associated with squad-based organizations is cost and priorities. If the development squad wants to buy a license but the production squad may also need it, where should the cost go? There are, of course, benefits to keeping the functional areas together, but it also creates these types of challenges.

One benefit of this model could be the competitive aspect between squads, which could lead to increased innovation but can also create friction between them related to budget and resource allocation.

Private agendas

An important aspect to consider is the potential for individuals to focus on personal goals and establish their own "kingdoms," rather than prioritizing the collective greater good. It is important to note that this behavior is not always the fault of the employee, as it can stem from management not providing a clear vision and motivation. However, by fostering a positive work environment, organizations can address these challenges. Encouraging open communication, providing a clear direction, and promoting collaboration can inspire employees to align their efforts toward the organization's success.

Job protection

People often perceive automation as a potential threat to their job security, which makes it a common concern when considering whether to adopt automation. Of course, protecting oneself is a natural reaction. However, based on our experience, individuals who embrace new technologies and adapt to new ways of working actually enhance their personal value on the job market.

Trust or lack thereof

Implementing automation can sometimes require a leap of faith. Many individuals may have experienced the initial hesitation of running an automated process and then waiting anxiously for it to complete, meticulously checking logs to ensure that it was successful. Even when running the automation repeatedly, there may still be lingering doubt as to whether it will perform as expected. To establish confidence in automation, incorporating testing is key. Integrating thorough testing into the automation workflow can ensure that the system promptly identifies any issues or failures. This approach enables real-time feedback, notifying you if something didn't work as intended.

Summary of organizations

Let us now compare the various types of organizations. The following table outlines some pros and cons of the various styles we have discussed.

Table 2. Comparison of organizations

Type of organization	Pros	Cons
Little kingdoms	• Commonly used and understood	• Less visibility into strategy • IT separated from the business • Siloed organizations, even inside IT
IT as a service provider	• Transparency of IT cost for the provided services • Clear visibility into provided IT services	• Requires a clear definition of all the provided IT services • Higher risk of shadow IT due to the speed of services, costs, and processes to request a new service
Squad-based organization	• Each squad could be set up to be competitive • Faster changes	• Isolated knowledge • Defined silos • No cost sharing such as Enterprise Agreements on licenses

Conclusion

There is no "magic bullet" to completely solve every challenge that arises, but we have tried to share our experiences and give our view of what a successful automation implementation could look like. We base our conclusion on the following three topics:

Structured approach

One of the most important things to learn from this white paper is the need for a structured approach to automation, and this includes people and processes. When a process has structure, it is much easier to follow, measure, and see the progress made. Seeing progress will help build trust in the organization. It is also vital for the structure to be reviewed on a regular basis, since a long-term plan is valid only when it comes to the details for the short term. This is mainly because everything is evolving, and we must adapt the plan accordingly.

One of the key things we have seen is that technology cannot be the only driver. Working with people and the corresponding processes has to occur side by side.

So do we have a golden recipe for making automation a success?

The recipe we have seen to be successful is a repeatable three-stage approach:

- Automate a task or IT service by starting with low-complexity, high-impact tasks before tackling more complex tasks.
- Build on the benefit realized from the automated tasks or IT services to implement full automation of additional tasks or IT services.
- Use the benefits (time saved and ease of deployment) to build trust in automation, both from a technical perspective and a people and process perspective.

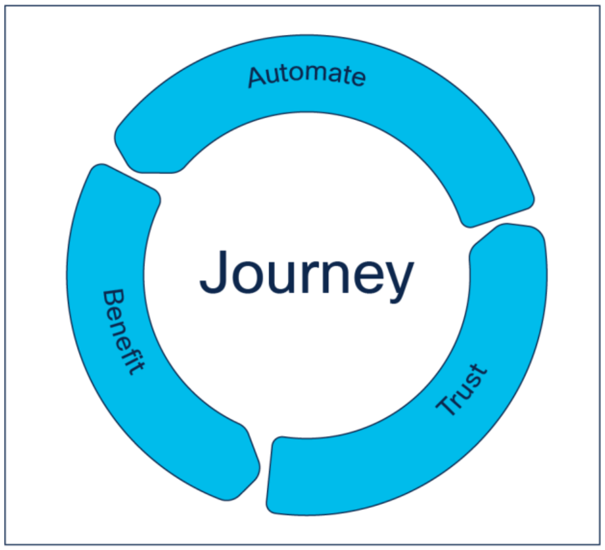

Figure 12.
Automation journey

As part of this journey, make sure to advertise the success of automation to make it "popular," which then will inspire others to adopt automation in their technical domain.

Automation naturally requires a clearly documented plan to be in place, but it must be revisited frequently and adapted to the ongoing change, digitization of the organization, and changes to the way of working. If a plan is created and defined to encompass 3 years, the plan should be revalidated at least every 6 months to provide guidance (quarterly, semiannually, in alignment with the audit cycle but 2 months before the actual audit), and then should be amended based on progress and any other developments. A company in general will always continue to grow, so the goals that were set in the beginning may not be the same over time. In addition, technology advances with tremendous speed.

Another good flipside of having a working automation framework could be the reporting function, which can show how many times a device or system has been updated, changed, added, or deleted, thus enabling the organization to track time spent, implementation costs, potential cost savings, and similar metrics.

People and processes

In our experience, this is the most overlooked aspect to address to make automation successful. Technology can solve a lot of problems, but we must consider the overall company as well. If we stop now and then, rethink, and adapt, that is a good start. It is very important to reiterate the long-term goals and translate them into short-term, tangible, actionable items. This allows the people and processes to grow with the technology, so they don’t end up fighting it.

Quickly going back to the data center model, we feel it is important to keep dissolving those silos and to clearly define common goals that everyone can not only benefit from but also see the reasons behind. It is very easy to state that automation is a must, but very easy to forget the true complexity of the relationship between humans and technology.

Lessons learned and experience

We have listed numerous challenges and issues, and recommendations for overcoming them. Here we will try to structure this into an approach that we have seen working in practice based on years of experience with implementing automation for our customers:

- Break through the organizational silos and define common Key Performance Indicators (KPIs).
- Provide training and information so people use the same terminology and have a full understanding of both the technology and the automation goals.
- See information as something to be shared, and encourage information sharing on all levels: executives, directors, managers, architects, engineers. Encourage more transparency between roles and positions.
- Understand the complexity and risks associated with “doing it ourselves” vs. purchasing off-the-shelf products or solutions.
- Start small and grow, so the organization can evolve at the same pace as the software.
- All opinions and inputs count, but emotions must be put aside for critical decisions. Once a decision has been reached, everyone must agree to follow that path.

Summary

We trust you have gained some new insights from this white paper, and that it is fostering some discussions. To help you with these discussions, we’ve distilled a short list of things to consider. One of the biggest challenges is the link between technologies, organizations, and human processes. When introducing a new technology, organizations frequently forget to let the people adapt and evolve as well.

It does not have to be hard to implement automation for a data center; it just requires some thinking, planning, and agreement. It also requires an open mind. The following list summarizes some key tasks:

- Define and validate why automation would help.
- Create common KPIs across organizations.
- Decide how to automate: Classic automation vs. infrastructure as code.
- Choose between home-grown vs. off-the-shelf solutions, as both should be considered.
- Train people and update organizational structure.
- Define a clear roadmap spanning 2 to 3 years, and review it constantly.
- Implement one use case to prove the framework.

- Use sprint-based development to add new automation use cases.
- Provide ongoing training and documentation.
- Provide cross-organizational updates and sprint demos.
- Keep track of ongoing lessons learned.

There is no order to these points, but they are all important to reflect on and define.

In conclusion, here are the truths as we see them:

- There is no magic bullet.
- There is no single technical solution that fits all.
- Investment from the business is vital.
- Evolution and building trust take time, even in a modern age.
- Involve multiple areas of the business to ensure a higher chance of success.
- Promote standardization of IT services.

Authors

Peter Charpentier, Principal Architect, Cisco Customer Experience

Morten Skriver, Principal Architect, Cisco Customer Experience

Reviewers

Asier Arlegui Lacunza, Principal Architect, Cisco Customer Experience

Michael Tievenow, Senior Solution Architect, Cisco Customer Experience

Nicole Wajer, Technical Solutions Architect, Cisco Sales

Steve Shaw, Solutions Development Architect, Cisco Customer Experience

Greg Simmons, Customer Success Enablement Leader, Cisco Customer Experience

Hank Preston, Principal Engineer, Learning and Certifications

Roger Dickinson, WW Solutions Engineer, CISG

Citations and bibliography

[1] Cisco Application Centric Infrastructure (ACI)

[2] Cisco Intersight

[3] Nutanix Prism

[4] RedHat Ansible Automation Platform

[5] RedHat, Ansible

[6] HashiCorp, Terraform

[7] Git, Distributed Version Control System

[8] Cisco, Nexus as Code