Infrastructure Security in a Cloud DevOps World

How to Automate the Launch and Maintenance of Consistent, Reproducible, Secure Cloud Environments

The days when IT departments managed a monolithic, infrequently modified stack are long gone. Applications change frequently, often daily. There are no “completion dates” on many cloud projects. Product teams need new cloud environments launched quickly, which means operations and security teams can no longer spend weeks manually building and delivering infrastructure.

Automation allows cloud operations teams to efficiently manage the fact that cloud infrastructure changes all the time. When common tasks such as updates, scaling, launching new environments, and patching are partially or fully automated, IT teams can control systems in a consistent, repeatable, and transparent manner. This has a significant positive impact on your overall security posture.

However, the reality is that focus is usually "higher up the stack" or "to the right" -- on automating application testing and deployment -- and not on the build-out and maintenance of cloud infrastructure. But as your cloud environment grows more complex, infrastructure automation is crucial. A self-regulating, self-correcting system significantly reduces both the risk and cost of security operations on Amazon Web Services (AWS).

Security on AWS is Your Responsibility

Many companies falsely assume that by migrating to AWS, AWS is responsible for the security of data and applications running on their platform.

By migrating to AWS, you have a shared security responsibility.[1] This shared model means that AWS manages the infrastructure components from the host operating system (virtualization layer) down to the physical security of AWS’ datacenters. It is your responsibility to configure and secure AWS-provided services. In other words, AWS controls physical components; you own and controls everything else. As AWS states repeatedly, “AWS manages security of the cloud, security on the cloud is the customer’s responsibility.”

The same line of demarcation applies to IT controls. Customers on AWS shift the management of some IT controls to AWS, which results in a shared control environment. AWS manages controls associated with the physical and architectural infrastructure deployed in the AWS environment; the customer is responsible for network controls (Security Group configurations), access controls, encryption, and any control not directly managed by AWS.

In short, running on AWS is very similar to running your applications in a rented datacenter. You are still responsible for common security operations tasks and for using AWS services in a secure manner.

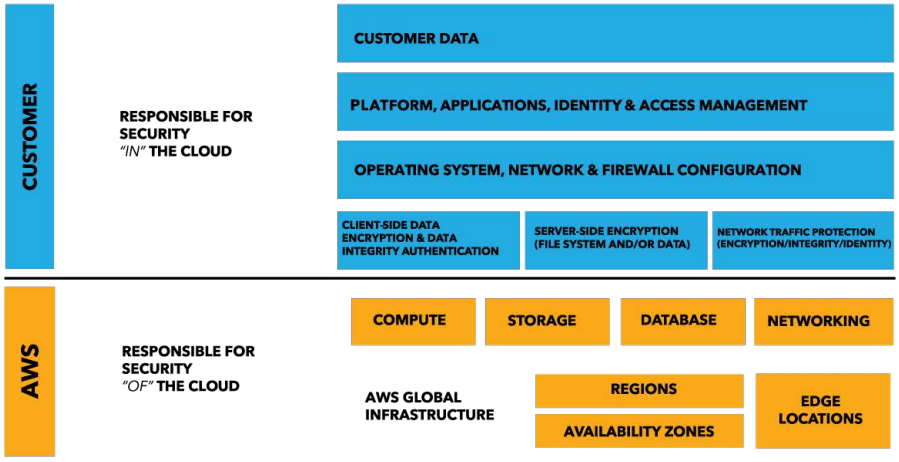
Physical Security and Environmental Controls

If you are concerned about the security of AWS' datacenters, any customer can access a copy of AWS' SOC 2 Type II report, which provides significant detail about physical security and environment controls. This report and ISO 27001 are available for review by audit and compliance teams. This means that if an auditor requests specifics regarding the physical controls of your system, they can reference the AWS SOC 2 Type II report. AWS does not allow datacenter tours, as independent reviews of datacenter security are also part of the SOC, ISO 27001, and other audits.

Data Privacy

AWS customers retain control and ownership of their data, and customers can move data on and off of AWS storage as required. AWS does not leverage any third-party providers to deliver services to customers and therefore does not provide any customer information or access to data to any other provider. Customers must control access to applications and data through the AWS Identity and Access Management service.

Client environments on AWS infrastructure are by default logically segregated from each other and have been designed to prevent customers from accessing instances not assigned to them. AWS has both instances that are dedicated to a single customer (Dedicated Instances) and instances hosted on shared infrastructure. AWS is responsible for patching the hypervisor and networking services, while customers patch their own guest operating systems, software, and applications.



AWS Shared Responsibility Model

Source: AWS

Ops Challenges in a DevOps World

DevOps describes a wide and varied set of cultural shifts associated with moving from a vertically structured, poorly integrated IT team – where developers and IT work in isolation and often at cross-purposes – to a team that collaborates more effectively and delivers software more efficiently. DevOps is not a tool or a platform, but there are many new technologies that have the potential to significantly accelerate this cultural shift.

In practice, the goal of DevOps for systems professionals is to tightly integrate applications with the infrastructure on which they run. Traditional infrastructure management tasks often presented roadblocks to rapid software development and delivery; today's ideal infrastructure "gets out of the way" of developers and enables them to securely and reliably carry out their tasks with minimal concern for the underlying infrastructure. Therefore, DevOps teams are often composed of system engineers who have developed integrated, automated, repeatable deployment and provisioning processes.

The majority of IT teams are transitioning to a DevOps cultural mindset.[2] But despite all the hype, mature DevOps programs are rare. Even companies with a relatively mature continuous integration / continuous delivery pipeline (CI/CD) often rely on static, manually built, manually operated infrastructure.

Manual infrastructure maintenance is not a problem when infrastructure changes are infrequent. But with 95 percent of operations professionals noting that they must create, modify, or retire server workloads anywhere from two to 100 times more frequently on cloud infrastructure environments than in their traditional data centers, automation is increasingly critical.[3]

The result is an overburdened operations team that is struggling to keep up, all while business professionals (who buy into the idea that the cloud is "easy" and "manages itself") do not increase IT operations resources. IT professionals report that 80% of IT managers underestimate the time and effort required to manage cloud resources.[4] With limited resources, IT operations professionals don't have time to automate infrastructure, and as a result, DevOps teams often must wait weeks or even months for new cloud infrastructure.

According to a recent survey of IT professionals[5]:

- 54% of respondents indicated they had no access to self-service infrastructure.
- 33% of respondents said it takes up to a month to deliver infrastructure.
- 26% of respondents said it takes one month or more to deliver infrastructure.

Security professionals report similar lack of resources and automation tooling. 85 percent of IT security professionals say security team hiring has not kept pace with the rate at which new server workloads are created, changed or retired in the cloud. And yet less than a third are leveraging a full suite of tools that enable them to secure and audit cloud server workloads automatically when configuring and deploying them.[6]

What is Security by Design?

It is no longer scalable to task systems engineers to monitor every change for every environment, particularly as the business creates more digital applications and must update these applications more frequently. But it is also not appropriate to abandon controls in favor of business agility. New processes and new technologies must be deployed in order to meet these new requirements.

In order to maintain a secure system, systems engineers and/or security professionals must overcome two principle challenges:

> Challenge #1: Ensure that IT controls are maintained even as applications and cloud environments change.
>
> Challenge #2: Reduce the amount of manual effort required to implement and maintain such controls in order to keep up with the pace of change.

Security by Design (SbD) is a approach to security that allows you to formalize infrastructure design and automate security controls so that you can build security into every part of the IT management process. In practical terms, this means that your engineers spend time developing software that controls the security of your system in a consistent way 24x7, rather than spending time manually building, configuring, and patching individual servers. Others promote similar or related concepts, often called Secure DevOps or Security Automation or Security-as-Code or SecOps.

This approach to system design is not new, but the rise of public cloud has made SbD far simpler to execute. Amazon Web Services has recently been actively promoting the approach and formalizing it for the cloud audience[7]:

> *"Security by Design (SbD) is a security assurance approach that formalizes AWS account design, automates security controls, and streamlines auditing. Instead of relying on auditing security retroactively, SbD provides security control built in throughout the AWS IT management process."*

In many ways, the principles of SbD fit naturally into DevOps principles:

- Manual work is risk; reduce human intervention in your cloud environment
- Expect systems to change – and welcome those changes
- Spend time fighting new threats, not re-fixing or governing old ones
- Test everything often (automatically)
- Simplify ("the art of maximizing the work not done")

In practice, AWS recommends building security and compliance into your system with a four phase approach.

Phase 1: Understand your requirements

The first step is to outline applicable regulations and enterprise governance standards as a single set of configuration requirements. In smaller organizations, this important step is often overlooked or left to systems engineers during the build process, and can result in inconsistent policies based on who built each system at the time. This is particularly the case with a set of regulations like HIPAA, where regulations are not technically specific and can be satisfied with a wide variety of tools.

Once those standards are established, enterprises often conduct a Security Gap Analysis, where current systems configurations are mapped to those guidelines. Steps for remediation are then stipulated. This can be performed by a third party consulting agency or in-house.

Logicworks Gap Analysis

The gap analysis examines the existing environment against a list of industry standards, Logicworks best practices, and if required, compliance standards including, SOC 2, HIPAA, and PCI. For each discrepancy found, Logicworks will provide suggestions for remediation.

ID	Priority	Category	Standard	Console Page	State	Notes
		Auditability	Does the account utilize CloudTrail in all Regions with resources present?			
		Auditability	Does the account utilize Config in all regions with resources present?			
		Auditability	Are resources tagged appropriately?			
		Auditability, Operations, Availability	Are root cause analysis criteria defined?			
		Availability	EBS volumes are separated for OS and Application/DB data			
		Availability	EC2 instances are backed up regularly			
		Availability	Critical application components are deployed in a redundant manner			
		Availability	Failure/recovery testing is regularly scheduled			
		Availability	Failover process is documented and understood			
		Availability	CNAME records are used to map DNS names to ELB or S3 buckets, rather than using A records			

Excerpt from a Gap Analysis by Logicworks.

Phase 2: Build a "secure environment" that fits your requirements

The next step is to build a standard, baseline template that can be used to build environments with ideal configurations. This template or series of templates should include the "must-have" security controls that together form a central, universal configuration. You should include things like installing NTP, MFA, IDS, and other essentials, installing log shipping and monitoring, requiring Multi-Factor Authentication on your bastion host or root account, binding instances to central authentication, etc. These features together form what NIST calls the Standard Operating Environment (SOE), also called a Canonical Environment.

This is simple but also revolutionary: it means that rather than spending months testing and reviewing each new cloud environment, your security team spends time upfront collaborating with systems teams to build a common standard, and then only needs to be involved when that standard changes and at other key points. Rather than asking your security team to review your code and system at the end of a project, they should be involved in building a SOE that gets "baked into" a templating tool like AWS CloudFormation and a configuration management script in Puppet or Chef. They only need to be notified when that SOE is changed.

AWS CloudFormation

In order to build a system that embodies the above principles, you need to treat your infrastructure like a piece of software; in other words, your networks, storage, etc. are manipulated with code, versioned in a code repository, and continually updated and improved. Builds are repeatable and reliable, and require minimal human effort.

In Amazon Web Services, this usually involves the use of AWS CloudFormation. AWS CloudFormation allows you to write a template in JSON and output a foundational AWS environment. This means that a systems engineer could write a set of templates for AWS environments that includes the company's baseline configurations and universal attributes, and then use those templates to launch future environments rapidly.

Here are a few tasks that AWS CloudFormation can perform:

- Build network foundation
- Configure gateways and access points
- Install management services, like Puppet
- Allocate Amazon S3 buckets
- Attach encrypted volumes
- Control and manage access though IAM
- Register DNS names with Amazon Route 53
- Configure log shipping and retention

The value of using AWS CloudFormation in compliant environments is tremendous. Entire environments can be quickly duplicated, modified and deployed in quick time. Templates can be checked into a source code management system, linted and validated using tools and IDEs. CloudFormation can also kick off bootstrapping through UserData, creating a seamless bridge into the operating systems where machine images and configuration management can take over. In addition, any infrastructure change can have an audit trail and proper process built around it, automated to reduce human error and disparate configurations.

Configuration Management

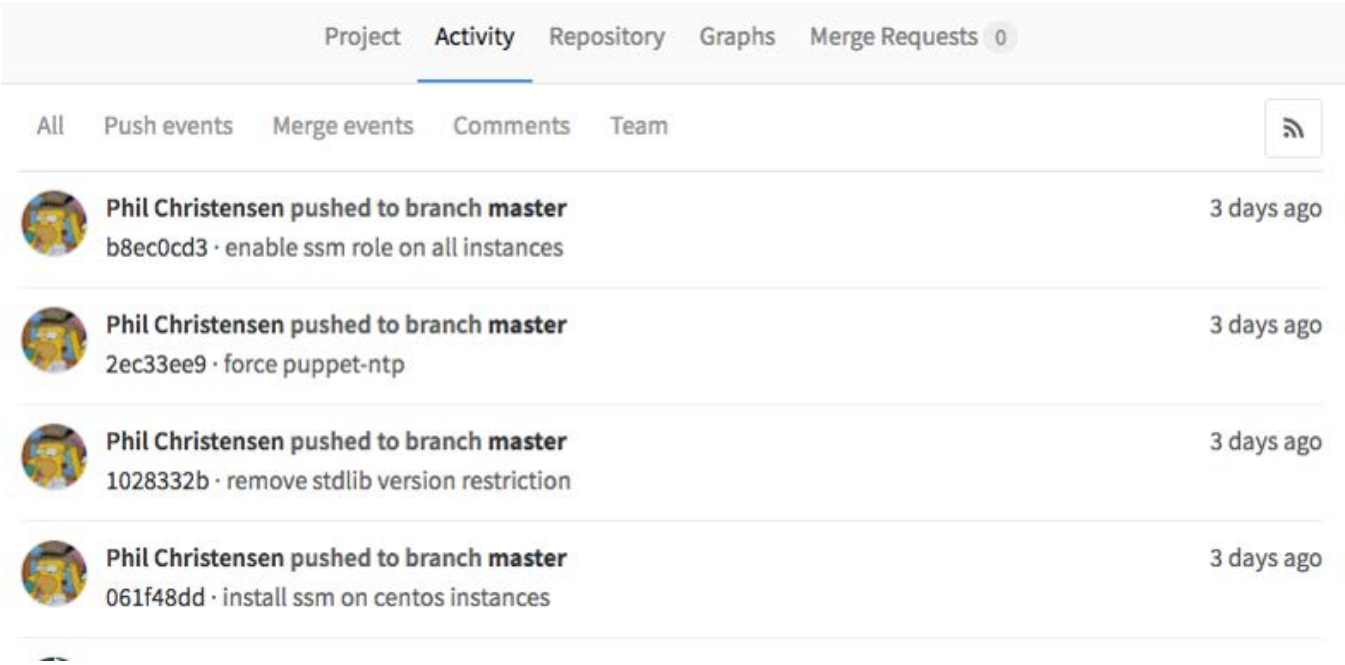
A configuration management tool is the second component of building a SOE. Configuration management is a 15+ year old technology that has come to prominence in the age of cloud due to its ability to programmatically “dictate” the ideal configuration of an operating system, and then maintain that configuration over time.

The fundamental purpose of configuration management is to deploy environments prescriptively. By defining everything on the servers in a single location, there is a single source of truth about the state of the entire infrastructure. The value of configuration management is that when you want to change the operating system (OS), you can change code on the servers without making major changes to the servers themselves. Configuration management is also used to install various security features on an instance, like Identity Detection System agents, log shipping, monitoring software, etc. as well as requiring MFA and binding the instance to central authentication. That means changes can be rolled out to many instances at once — and be rolled back. When you are done, each server is in a known, ideal state. When you build and maintain your environment manually, change is slow, cannot be rolled back, and has a higher risk of human error.

With AWS, there are many possible tools to control configurations. You can choose to use an AWS native tool like OpsWorks, which uses Chef. You can deploy your own server with Puppet, Chef, or another configuration management tool of your choice. Obviously, the latter provides greater control, but requires that you provision and maintain your own Puppet/Chef server. You can also manage services in a more restricted (but more fully-managed) way through Amazon EC2 Systems Manager, which is essentially a collection of a la carte services to create system images, apply OS patches, track system configurations, etc.

Version control is well-supported by all configuration management tools. It is possible to use a mature Software Development Lifecycle to manage the development and maintenance of your recipes / manifests, tightly integrated with a branch-based workflow that truly realizes the ideals of “infrastructure-as-code.” For example, in the latest versions of Puppet, environments are simply different directories of modules on your Puppetmaster. Each environment will only see the version of a module that is in its directory, so you can simply check out the corresponding branch of your custom module in each directory.

Combined with an automatic update of the Puppetmaster's modules in response to Git hooks, engineers can try out the latest version of a manifest in a testing environment first. Once it is ready to push to staging or prod, the manifest in the staging folder is merged directly over to your production or master branch.



Example of a customer Git repository showing updates to Puppet.

Puppet also helps maintain the correct configuration even if it means reverting back to an earlier version. You describe what the environment should look like – this directory has to have these permissions, these versions, this Apache virtual host for a website, this alias, etc. – and if the configurations are wrong it will change them back to the correct permissions. For example, one could write a custom fact to report whether or not each instance Puppet knows about is running the correct version of a database or in the correct security group. You can programmatically discover all of the systems that are not registered with a certain security group and automatically register the instance if it is not.

In complex environments with compliance requirements, this can significantly lower the risk of configuration drift. These authoritative versioning controls alone make Puppet worth the effort of mastering.

Finally, the effort put into building an SOE is very similar if not identical to creating "DevOps" infrastructure practices. Once you create an SOE, you can then automate the process by which an alert (such as an Auto Scaling event, a code push, or instance failure) kicks off the build process. In very highly automated environments, such as those built in alignment with the principles of Immutable Infrastructure, an entire testing environment can be destroyed and rebuilt with AWS CloudFormation, configuration management, and a deployment automation tool like AWS CodeDeploy in minutes rather than updating the code.

The security team is usually not involved directly in the launch of the product, but is deeply involved in making sure the deployment process is secure.

The key areas of focus for security teams in building a secure deployment pipeline is access. Here are just a few practices security teams can automate, using configuration management in conjunction with a simple deployment tool like AWS CodeDeploy:

- Do not allow SSH/RDP access to test/dev/staging
- Restrict who can deploy by environment; a very limited number of engineers should have permissions to push to production
- Run automated tests either as part of the deploy process or when changes are committed to your SOE
- For compliant or sensitive workloads, no one can make direct changes to test/dev/staging. It must be done by modifying SOE. Depending on risk tolerance, this may not be necessary.

The key here is to prevent engineers from making destructive changes to the production environment. In a worst case scenario, building a system in this way allows you to blow away an entire environment and rebuild an SOE through existing automation tools.

Phase 3: Enforce the use of templates

Over time, a company develops a collection of CloudFormation templates and configuration management scripts for various use cases. A central IT office can even organize AWS CloudFormation templates and make them accessible to engineers in a controlled, self-service fashion through a tool like AWS Service Catalog. Through Service Catalog, project teams can access centrally-approved templates that build out critical components, significantly accelerating build-out while simultaneously putting up guardrails to control core architectural standards. The company can also simply keep both CloudFormation templates and configuration management manifests in a code repository like Git.

This means you can effectively outlaw one-off changes and engineers cannot just go in and change the cloud instance in the console or command-line interface. You have to go into GitHub, check out a branch of your Puppet manifest, make the change, submit it to either a senior DevOps or security team member, and have it get approved. This means that engineers will very rarely or never need to get access to the actual environment – all the work is done with temporary, assumed roles. This also means you have a record of every change ever made on your system and a way to roll back changes that fail.

Keeping templates in a central repository helps companies enforce their use. One can even restrict an engineer's ability to manually create resources. Usually, there is some resistance to the idea of replacing "quick" one-off changes with this more controlled process. As the team matures, they will decide where controls are appropriate and where they hinder agile processes.

Phase 4: Perform validation activities

The final step is to continuously monitor the current state of your environment and self-correct systems in which configurations do not match the “secure environment” template.

As described above, because each environment is linked to configuration manifests, it proactively monitors the configuration of each environment and corrects instances to maintain proper configurations. For example, say that a client engineer mistakenly reverted their OS back to a previous version, which was updated due to security vulnerabilities. Puppet / Chef continually checks on instances and rolls back non-authorized changes like this one. This means that engineers can ensure that no (permanent) changes are made on the instance level that compromise security. This central hub of configuration management scripts also allows the enterprise to push out “emergency” fixes rapidly; for example, in April 2014 when the zero day attack Heartbleed occurred, we were able to require the most recent version of SSL in the central hub, update every spoke account within a short time, and ensure that all instances maintained that new version in perpetuity.

At Logicworks, we also have a central Jenkins server for deploying global changes to all AWS clients. This is a little more complicated because it leverages a central management hub that has the knowledge and access rights to iterate through all environments looking for particular flaws, mistakes, or common errors. We call these “scanners”; basically, Python scripts that run across every environment performing housekeeping jobs, alerting our Network Operations Center of non-compliance with Best Practices, taking snapshots, and maintaining consistent configurations.

Essentially, it allows us to find the “error” in the haystack; with thousands of security groups, finding one with a misconfigured security group that has, say, Port 22 open is almost impossible. These scanners can either alert our Network Operations Center or actively configure the security group to close the port. Of course, in order for these tools to work properly in the first place, you need to have a standard build process.

Here are a few of Logicworks’ current “scanners”:

- Ensure AWS Config and CloudTrail are enabled
- Check for upcoming EC2 restart events
- Check that MFA is required to login to the root account
- Check for client resources deployed to a new region
- Check for legacy admin rules
- Snapshots of EBS volumes based on tagging with customizable retention
- Auto-update AMI when upstream changes
- Scan for soon to expire SSL certs on ELBs
- Scan for dedicated tenancy or lack thereof

Scanners ensure that your environment is as close as possible to an ideal, standard state.

The Value of Security by Design

Enterprise security leaders are worried about their ability to keep up with the rate of system change in the cloud. Security by Design relies on set of tools that both DevOps teams and security teams can love.

As cloud security practices mature, we need to take the best traditional security methodologies and empower security professionals to work at the speed of DevOps. Configuration management and related software development-oriented technologies can make this a reality. Throughout this ebook, we have emphasized the tremendous technical and operational benefits of Security by Design. Just to summarize, we see the most important benefits as:

Improved Transparency: You know exactly how every system is configured for security at any point in time.

Increased Efficiency: You reduce the time and cost of deploying future systems; you do not have to rebuild security configurations or get them approved by security teams.

Enforced Policies: Your configuration management tool regularly "checks in" to your system to make sure your baseline configurations are maintained, meaning your system never suffers from "configuration drift". By converting traditional manual controls to automated controls, the customer is assured that controls are operating 24x7x365, rather than relying on point-in-time reviews.

Reduced Manual Work: By centrally managing configuration, you discourage ad hoc work; any change made directly to the instance and not to the script will be overwritten when your configuration management tool runs anyway. Systems engineers can work on tasks that differentiate the business, not on maintaining spreadsheets or frantically preparing documentation at audit time.

Simplified Patching: Patches can be distributed across every system rapidly and with a complete audit trail of what was patched where.

Auditors Love It: You can tell them exactly how your system is configured, critical compliance features like logs and MFA can never be "forgotten", and every change is centrally documented.

The main question is whether or not enterprises will have the time to build SOEs and set up these processes as they migrate to the public cloud. It requires training, a team of advanced DevOps engineers and Puppet/Chef experts, and months of work. Outsourcing Security by Design set-up and maintenance to a 3rd party provider can protect cloud projects.

About Logicworks

Logicworks is a cloud automation and managed services provider. As an AWS Premier Partner and a member of the AWS Managed Services Program, Logicworks has proven expertise in managing complex cloud infrastructures for enterprise clients in the healthcare, financial services, and commerce industries. We design, build, and maintain cloud environments following Security by Design principles and have developed a library of proprietary security automation software.

Contact Logicworks at 212-625-5300 or visit www.logicworks.com.

References

1. Amazon Web Services. *Overview of Security Processes*. May 2017. https://d0.awsstatic.com/whitepapers/Security/AWS_Security_Whitepaper.pdf.
2. RightScale. *State of the Cloud: DevOps Trends Report*. January 2016. http://www.rightscale.com/lp/devops-trends-report?campaign=70170000001DOW8.
3. CloudPassage. *Survey: Exponential Server Growth, Dynamics of Cloud Increase Attackable Surface Area and Risk.* August 19, 2016. https://www.cloudpassage.com/company/press-releases/cloudpassage-survey-exponential-server-growth-dynamics-cloud-increase-attackable-surface-area-risk/.
4. Logicworks. *Survey: Roadblocks to Cloud Success*. August 2016. http://go.logicworks.net/report-roadblocks-to-cloud-success.
5. Quali. *2016 Cloud and DevOps Survey*. March 15, 2017. http://www.prnewswire.com/news-releases/qualis-survey-offers-insights-about-it-challenges-in-cloud-and-devops-300423438.htm.
6. CloudPassage. Survey: Exponential Server Growth, Dynamics of Cloud Increase Attackable Surface Area and Risk. August 19, 2016. https://www.cloudpassage.com/company/press-releases/cloudpassage-survey-exponential-server-growth-dynamics-cloud-increase-attackable-surface-area-risk/.
7. Amazon Web Services. *Introduction to AWS Security by Design*. November 2015. https://d0.awsstatic.com/whitepapers/compliance/Intro_to_Security_by_Design.pdf.