CitihubDigital

November 2019

Implementing Chaos Engineering & Continuous Compliance for Financial Services

Recoding the Digital DNA of Financial Services

citihub.com

Contents

Authors

Ian Tivey
ian.tivey@citihub.com

Brett Aukburg
brett.aukburg@citihub.com

Paul Jones
paul.jones@citihub.com

Jim Oulton
jim.oulton@citihub.com

Ming Zheng
ming.zheng@citihub.com

Introduction

The Simian Army was originally developed by Netflix as a set of tools to ensure that its video streaming service was always available to global customers without any service degradation (while ensuring compliance with all policies relating to security, conformity and cost). That goal may seem relatively straight forward. However, given the scale of its operations, it is anything but. To frame the challenge, consider some of the following statistics:

- Netflix services approximately 150 million paying accounts[1] across 190 countries[2]
- It streams more than a billion hours of video content every week[3]
- It is said to consume approximately 15% of the world's internet bandwidth[4]

Netflix clearly faces requirements (in terms of scale and service quality) that would dwarf those posed by most financial applications. But financial institutions have other unique factors to consider - specifically much more stringent policies and regulations governing access controls, information security and availability (for example MAS 644 specifies a maximum total downtime of 4-hours in a 12-month period for all critical systems operated by banks).

This paper explains why the concepts introduced by the Simian Army are important to any financial institution adopting cloud services. It provides an overview of those concepts – specifically chaos engineering and continuous compliance – along with a more detailed explanation of relevant tools and guidance on how to implement them, both from a practical perspective and in terms of a suggested organisational model.

[1] https://s22.q4cdn.com/959853165/files/doc_financials/quarterly_reports/2019/q2/Q2-19-Shareholder-Letter-FINAL.pdf

[2] https://www.netflixinvestor.com/ir-overview/profile/default.aspx

[3] https://news.softpedia.com/news/netflix-users-spend-1-billion-hours-per-week-watching-movies-514989.shtml

[4] https://fortune.com/2018/10/02/netflix-consumes-15-percent-of-global-internet-bandwidth/

Why is the Simian Army Important to Financial Institutions?

Public cloud adoption by the financial services industry has lagged behind other sectors. Financial services firms are heavily regulated and subject to more stringent requirements relating to data privacy and security.

Applicable regulations, to name a few, include Dodd-Frank, FFIEC, PCI DSS, GLBA, SOX, USA Patriot Act, MAS TRM, MAS 644, HKMA TM-G and GDPR. Additionally, high profile data leaks have tempered some of the appetite for hosting critical workloads and sensitive data in the cloud, emphasising the importance of controls and continuous compliance.

Operating in a cloud paradigm has some fundamental differences to traditional modes of managing IT infrastructure and software. Cloud supports the creation, modification, and destruction of resources with orders of magnitude greater speed than traditional systems. Cloud environments generally expect relatively high rates of component failures because they are built on large quantities of inexpensive, commodity components.

The growing use of public cloud services in the financial services industry therefore requires a rethink of some key aspects of application development, service management and support:

1) Availability

Public cloud services can experience a higher rate of component failure than traditional on-premise dedicated infrastructure. It is therefore vital that applications developed for the cloud are built to fail. Resilience needs to be architected into software. This core requirement has triggered several corresponding trends in software design, including adoption of microservices, a move from stateful to stateless architectures and a tendency to decouple data from applications.

Similarly, when it comes to service management, the ease with which cloud services can be provisioned enables applications to be re-built more easily and at regular intervals - ensuring system entropy (another potential cause of availability issues) can be re-set.

2) Security

When it comes to information security, as well as identity and access management, the financial industry is subject to much more exacting standards than most other verticals. Although many financial institutions have grown comfortable with the use of Infrastructure as a Service (IaaS) by implementing an Infrastructure as Code (IAC) approach to define and enforce minimum security standards, the adoption of Platform as a Service (PaaS) has introduced greater complexity and new challenges.

The need to lock down all potential attack surfaces in an environment that has primarily been architected to be internet-based, open and multi-tenant requires continuous monitoring to ensure all security policies are properly implemented and do not change.

3) Cost

Cloud economics can be compelling when using appropriate software architectures but it requires good hygiene. Making resources easier to procure can lead to sprawl, so organisations will need to continuously monitor services to ensure they are making use of everything they procure. Equally, cloud resources are most cost effective when software is architected appropriately, with modern architectures helping to reduce reliance on dedicated resources and ensure firms only pay for the CPU cycles necessary to support application processes.

4) Conformity

The move towards Agile & DevOps development methodologies has evolved in tandem with the adoption of cloud. These approaches encapsulate a crucial benefit that financial services are trying to unlock - enabling software development teams to innovate faster. However, as resources become easier to provision and application teams take on more responsibility for their own destiny, new risks need to be managed.

As more responsibility shifts to the application teams, it is vital that those teams are continuously monitored to ensure they conform with all relevant IT policies.

Introducing the Simian Army

Tools like the Simian Army were designed to help organisations adapt to the cloud and minimise risks associated with software defined environments. They do this in the following ways:

1) Teaching people

Simian Army helps to keep IT professionals on their toes. Whether that means regularly injecting some element of chaos (pseudo-random abnormal events and failures) into the system or actively handling compliance with policies and security controls - it prepares an organisation to perform at its best when working in software defined environments. Key roles that need to be involved include:

- Developers are trained by ensuring their code is designed to failover in any given circumstance, whether that involves the failure of an individual instance, a data centre or an entire region.
- Operators are trained by having to deal with frequently injected nuisance-level chaos and being better able to set appropriate thresholds for what constitutes an incident.
- BCP/DR teams are trained by being able to simulate and respond to more serious incidents.
- Compliance and security personnel are trained by having to “codify” their rules and controls to enable pre-emptive actions at the speed of software.
- Procurement professionals and budget holders are trained by having to set, monitor and enforce rules for resource consumption.

2) Training systems

The key aim of chaos engineering is to ensure applications are ready to handle the higher frequency of component failures in the cloud. By injecting chaos at every stage in the application lifecycle – from development and testing through to production – systems should be accustomed to deal with such failures as a matter of routine. Failure can, and should, be simulated at various levels. While the original Chaos monkey simulated the loss of one or more instance, chaos engineering evolved to simulate broader disaster-level events such as the loss of an availability zone or region. Equally, the evolution of microservices architectures has required chaos engineering to simulate failure at a more granular level. This can help identify combinations of small individual faults that lead to cascading failures with critical implications on complex systems made up of many inter-dependent microservices.

3) Monitoring and overlaying controls

The immediacy of service provision offered by the cloud requires more automated controls to ensure policies are adhered to (in terms of security, cost and conformity). Continuous compliance is a discipline that ensures cloud environments are always monitored and controlled to ensure they are compliant with relevant policies, remain secure and operate efficiently in terms of resource consumption. This includes overlaying both active and passive measures of enforcement - ensuring non-compliant conditions are prevented when possible, immediately disabled if not preventable, or immediately alerted upon whenever appropriate.

Chaos Engineering

The Simian Army was developed as a set of individual monkeys, each of which serve a distinct function. We would typically group those monkeys into two broad categories – those designed to support chaos engineering principles, and those charged with ensuring continuous compliance with relevant IT policies (addressing security, cost and conformity).

Chaos Monkey is a tool that randomly disables or disrupts resources to make sure an application can survive common types of failure without customer impact. When resource-level actions are allowed, this could include running processes with memory leaks, erratic CPU consumption, and intermittent network disruption.

More commonly for finance industry use cases, actions are limited to CSP-level. APIs, either directly from the CSP or from an abstraction layer, are leveraged to disable resources/instances and test the resiliency of the application's architecture. For example, leverage Azure's APIs, we can test an application's network resilience by detaching/attaching virtual networks (vNet). Cloud-ready applications are usually deployed across several distributed groups (aka clusters) rather than a single machine. When we choose which resources/instances to disable, we need to be careful. When we target those machines, we need to query their meta-data to make sure we don't disable all nodes in the same cluster or else the application will be disabled without achieving the goal of the test.

Chaos Gorilla and Chaos Kong

When Netflix developed the Simian Army, one of the company's underlying design philosophies was to have every software service replicated and able to operate in any one of three availability zones, with the same philosophy applied to its Cassandra clusters for content storage.

Chaos monkey was designed to take down individual instances to test failover capabilities within a zone. However, to simulate broader failures, Netflix created Chaos Gorilla to take down an entire availability zone and Chaos Kong to bring down a region.

These tools are designed to simulate failures on such a scale that they lie more in the sphere of disaster recovery (DR) and business continuity planning (BCP). That means they are more likely to be triggered in a controlled manner, more akin to fire drills than the regular cadence of chaos monkey activities. As such, the personnel and organisational model of this type of chaos is likely to require more foresight into when drills ought to be scheduled to ensure minimal disruption to clients.

Failure Injection Testing (FIT)

While Chaos Gorilla and Kong ramped up the level of chaos to simulate more widescale failures, Failure Injection Testing (FIT) was developed to trigger more granular failures – breaking things in realistic ways, while limiting their impact. This includes application and data level anomalies, such as injecting malformed messages or slowing the response time of systems or examining how a system functions from the perspective of an individual user under different scenarios. In one example, Netflix was able to use FIT to confine failure experiments to specific test accounts, ensuring production systems could be tested (end-to-end) but without any potential fallout on paying subscribers.[5]

Chaos Automation Platform (ChAP)

Whilst FIT was designed to enable more granular failure scenarios, in order to understand the resilience of an overall system it may be necessary to scale out those granular failure scenarios, given that the loss or degradation of certain service instances may have repercussions on the broader system. To do this, Netflix developed its Chaos Automation Platform (ChAP). ChAP enables multiple FIT experiments to run concurrently – helping to analyse the impact of many smaller failures at the same time. It also allows FIT experiments to run at a regular cadence to ensure test results are regularly updated.

Lineage Driven Fault Injection

Lineage-Driven Fault Injection (LDFI) is a technique that looks to identify combinations of smaller injected faults that can escalate and trigger cascading failures in broader systems. LDFI works by first looking at what constitutes success – identifying all paths and processes required for a system to behave as intended. This provides a list of candidates into which to inject faults. The technique is particularly relevant to large, distributed systems, where dependencies between services are not easy to model. By injecting different combinations of faults, operators can seek to identify potential triggers, which in themselves may seem innocuous but have the potential to seem innocuous but have the potential to result in broader system issues.

[5] https://medium.com/netflix-techblog/fit-failure-injection-testing-35d8e2a9bb2

Continuous Compliance

While chaos engineering principally focuses on ensuring cloud systems are highly available, continuous compliance tools have evolved to help address other key aspects of application design and service management that require a rethink in the cloud – namely, security, cost and conformity. Original members of the Simian Army in this category include Security Monkey, Conformity Monkey, Doctor Monkey and Janitor Monkey.

These monkeys all follow a similar general pattern. They pull resource information from the cloud and evaluate it with a predefined set of rules. The monkey then takes an action based on the results of their evaluation. Actions generally fall into one of the following types: notify, sequester, shutdown, or destroy. The monkeys are typically not used to correct the configuration of resources which violate one or more rules – the intention is to enforce good behaviour by system developers, rather than correct issues on their behalf. Prioritisation rules are leveraged to prevent conflicts between monkeys.

Conformity Monkey (since incorporated into Spinnaker)

Conformity Monkey finds resources that do not adhere to best-practices and takes action. This component focuses on enforcing a set of compliance rules set by a compliance team. For example:

Virtual machines need to belong to an auto-scaling group

- At least two machines in each group
- All resources/instances need to be tagged properly
- Certain data cannot be in the public cloud for more than N days

The cloud provider's policies can enforce some rules for us. So, we verify that those policies are in place. We can define all those rules in a semi-structured data format, such as JSON or YAML. We query resources/instances in the public cloud frequently by using a scheduler and comparing retrieved information with the appropriate rules.

Although originally developed as a standalone set of tools, Conformity Monkey functionality has since been rolled into open source continuous delivery platform Spinnaker (Chaos Monkey is also integrated with Spinnaker, although continues to be available as a standalone service). Netflix open sourced Spinnaker in 2015 and since then other leading technology firms, including Google and Microsoft, have joined the community and contributed to the initiative.

The goal of Spinnaker is to automate software deployment to support a much higher cadence of code releases. That means automating parts of the release pipeline with tools to bake, deploy and test new code releases, detect bugs and provide an opportunity to fix them, roll back changes and minimise the impact on the broader system.

Security Monkey

Security Monkey is an extension of Conformity Monkey. It finds security violations or vulnerabilities. For the finance industry, security plays perhaps the most important role when building an application. Small security flaws can be expensive to fix, and the reputational damage may be irreparable. It is critical to continually check that public cloud security features are in place. Example rules could include:

- All virtual machines need to only open ports approved by the security team
- All virtual networks should be connected to the company's on-premises network using a VPN gateway
- All instances/resources should not have public IP assigned by the cloud provider; all data at rest in storages need to be encrypted by the company's key
- Use key-vault for key encryption management

Security Monkey rules should be incorporated early into the software development life cycle (SDLC) to ensure security requirements are well thought through, defined and implemented at an early stage.

Some cloud providers, such as Microsoft Azure, also provide built-in threat protection tools, which can detect suspicious user activities, such as abnormal login locations, brute force attacks, suspicious authentication failures, etc. Security Monkey can be used to retrieve and act on those alerts and send to stakeholders.

Janitor Monkey (now known as Swabbie)

Janitor Monkey ensures that cloud environments run free of clutter and waste. It searches for unused or expired resources and disposes of them. For this component, we can define expiration or lifespan rules and apply them to resources. For example, if Janitor Monkey finds a virtual machine that is eight days old, and the maximum lifespan defined for virtual machines running that application is seven days, we delete it. Exceeding lifespan is easy to detect. While identifying idle resources is more challenging, it is doable. This will vary depending on the CSP and the service in question. question. Not all resources will have a ready API that can be queried to detect whether it is in use.

Recently, Janitor Monkey functionality has been rolled into a replacement set of tools known as Swabbie. Swabbie (version 0.1) was first released in February 2018 and follows the same principles as Janitor Monkey. It applies a set of user-defined rules to mark resources for deletion. However, it also offers some additional steps to ensure needed resources are not deleted. For example, once Swabbie has marked and scheduled a resource for deletion, the owner of that resource is notified. The resource is then checked one final time to ensure it still meets the criteria before finally being deleted. deleted. Swabbie also offers the ability to opt certain resources out from the rules to ensure they are never deleted.

Doctor Monkey

Doctor Monkey taps into health checks that run by default on many resources as well as monitoring other signs of health (e.g. CPU load, number of requests, queue depth) to detect unhealthy instances. We establish thresholds, upper limits for CPU, hard drive, memory, and bandwidth usage in Doctor Monkey's rules. Any instance that exceeds the limit for a specified duration is flagged for more in-depth diagnosis. Key metrics are obtained by querying the monitoring logs. Constantly checking resource health is key to improving the availability and reliability of an application.

The Direction of the Army

Since it is no longer actively maintained by Netflix (the last version, v2.5.3, was released on the 4th of January 2017), the Simian Army has largely been integrated into parallel Netflix-founded open source project/s Spinnaker and Swabbie. Integrating key components of the Simian Army into Spinnaker/Swabbie is emblematic of the need to embed chaos engineering and continuous compliance principles into the software development lifecycle (SDLC).

However, it is important to note that the Simian Army can be deployed independently for organisations that are not currently looking to use Spinnaker, with the Simian Army code base still available to download via GitHub.

A Roadmap for Implementation

Because of its importance in preparing developers and ITSM personnel alike, tools like the Simian Army ought to be fully embedded into an organisation's software development and release pipeline.

To ensure that applications are "cloud ready" before they are deployed for production use in the cloud, it is essential to integrate Simian Army into every environment (e.g. dev, QA, prod) across your application delivery pipeline. Surviving and complying with the Simian Army are part of the criteria for promotion to the next environment. This full pipeline integration aligns with the "shift-left" approach recommended for agile development and allows developers to iterate and quickly correct during their earliest phases rather than discovering architectural design and coding weaknesses just before or just after production release.

As an extension of this philosophy, Simian Army should also be implemented early in an organisation's cloud program. Deploying the Simian Army enforces a strong level of discipline from the start of every project – helping to catch and resolve potential flaws with regards to availability, security, conformity or cost. It is useful for all technical resources to get used to both the chaos and the enforcement before the cloud becomes an area for hosting production workloads. That is because it is much harder to retrofit and gets harder with each production workload deployed without Simian Army in effect.

Setting Requirements

Financial applications have a much higher bar when it comes to security and controls than the original use case envisioned for the Simian Army. To achieve continuous compliance with those requirements, it is crucial that institutions can implement Simian Army tools in a way that captures all regulatory obligations and maps them to corresponding policies, controls and tests.

To support that process, Citihub recommends that organisations first map all relevant regulatory obligations onto a cloud controls framework. This ensures a comprehensive set of controls are in place to satisfy the superset of rules and regulations applicable to the institution.

To ensure that framework is then implemented correctly, each control will have to be translated into relevant scenarios. We recommend using a behavioural driven development (BDD) language to express those scenarios. Defining certain behaviours and corresponding actions accurately, using logical constructs but using plain English rather than code should help ensure more active participation from non-technical functions in the requirement setting process.

Finally, once all scenarios are mapped out, these will need to be translated into corresponding Simian Army rules to continuously monitor and test that those controls have been implemented correctly and implemented for each cloud being integrated.

Adapting Simian Army for Hybrid Multi-Cloud

Most large financial institutions will typically have evolved a mix of on-premise private cloud infrastructure along with integrated services from multiple CSPs to support a wide variety of applications. Although the Simian Army was originally implemented in a single CSP environment, it should be relatively straight forward to adapt to a hybrid multi-cloud world.

The benefit of using behavioural driven development and deploying a common set of tools like the Simian Army across all cloud environments is that a core set of behaviours and rules can serve universally, helping to define a baseline set of standards that applies to all environments. It is only the literal execution of each rule that will be bespoke for each cloud.

Perhaps the single biggest question mark will be in the way Simian Army tools are deployed on private clouds, given that many on-premise private clouds do not operate in the same way as public cloud services.

Unleashing your Army

Simian Army monkeys can execute actions in the context of the CSP's control plane; the context of the cloud resources themselves; or the context of the applications. "CSP control plane" activities are those that an administrator could perform from a CSP's web console or via a CSP's API and include activities such as like resource creation, destruction and modifying configuration. "Resource level" actions are those that an administrator could perform within the context of a specific resource type, for example, commands within a secure shell session to a virtual machine, the "kubectl" interface of a Kubernetes cluster, or the CLI of a network device. Actions in the application context could be simulations of user activity, injection of malformed messages into the network, artificially slowing down system responses, or any number of other application level issues.

Monkey Rules

As outlined in the section of requirement setting, it is important that monkey rules form part of a broader control framework. This helps to ensure that all rules align to specific policy or regulatory requirements, ensuring there is limited conflict between rules in different engines and that the rules are designed to ensure specific outcomes.

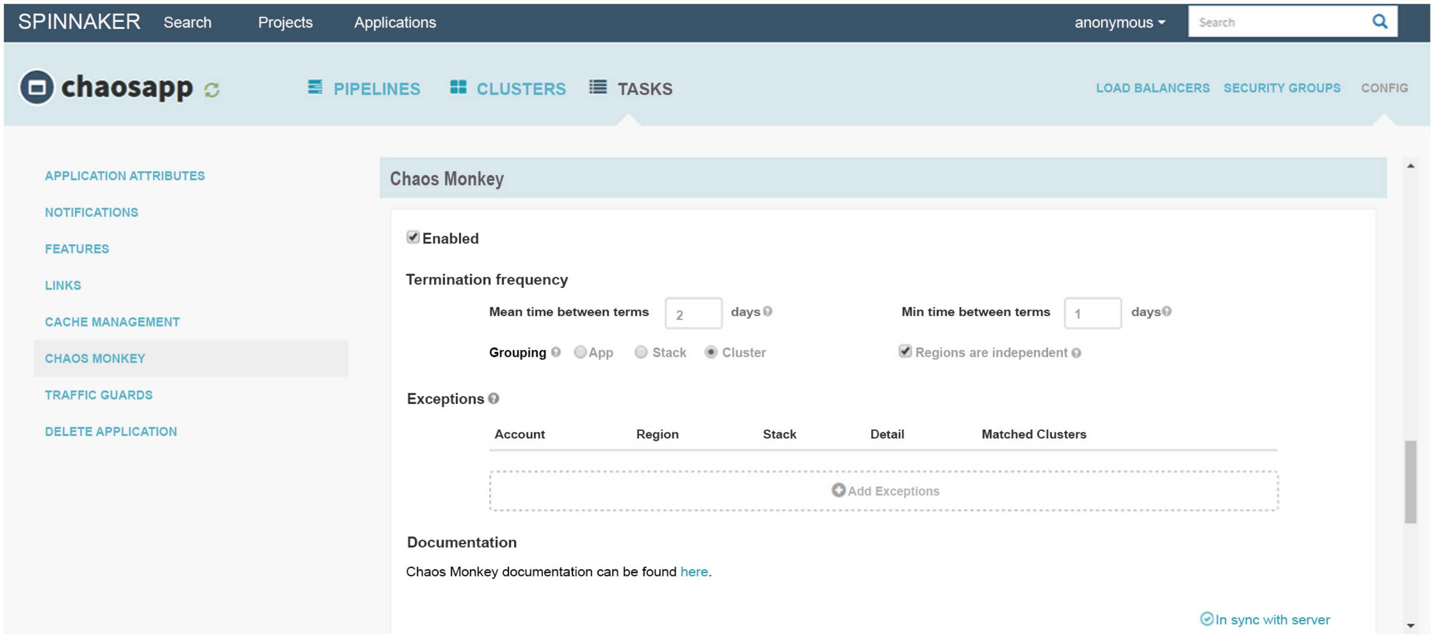
When it comes to writing those rules, organisations can opt for a number of choices. As previously detailed, using BDD to define rule behaviour is an important step as it helps to ensure that teams think in logical terms and remain focused on outcomes, in view of different scenarios. At the very least, we would recommend using a human readable language (such as JSON or YAML) to ensure that all members of a cross-functional team can review and contribute to the process.

The following is an example of a Chaos Monkey rule that has been designed using YAML.[6]

```
name:  “shutdown VM rule”
type:  “chaosMonkey”
target:  “VM, ScaleSet”
description:  “if shut down 5% of instances in same group, application should survive”
appId:  “12345678”
appEnv:  “prod”
appStatusUrl: “https://someuri.com/status”
parametres:
  •   “minimumNumberOfInstance=1”
  •   “maximumPercentNumberOfInstance=0.05”
conditions:
  •   “isAppAlive == false”
actions:
  •   “chaosMonkey.sentFailNotice”
```

FIGURE 1: EXAMPLE OF CHAOS MONKEY RULE WRITTEN IN YAML[7]

Now that Chaos Monkey has been integrated with Spinnaker, it is also worth noting that monkey rules can be configured via a user interface as shown below in Figure 2.



Chaos Monkey

☑ Enabled

Termination frequency

Mean time between terms: 2 days | Min time between terms: 1 days

Grouping: ○ App ○ Stack ● Cluster | ☑ Regions are independent

Exceptions

Account	Region	Stack	Detail	Matched Clusters

Documentation

Chaos Monkey documentation can be found here.

FIGURE 2: SCREENSHOT OF SPINNAKER GUI TO CONFIGURE CHAOS MONKEY RULES[8]

[6] YAML is a human-readable data-serialization language. It is commonly used for configuration files and in apps where data is being stored or transmitted

[7] https://netflix.github.io/chaosmonkey/Configuring-behavior-via-Spinnaker/

[8] https://netflix.github.io/chaosmonkey/Configuring-behavior-via-Spinnaker/

Organisational Model - Who Should be Involved?

Tools like the Simian Army need to be implemented by cross-functional teams given that they span multiple roles and responsibilities within an organisation - including application developers, testing and service management through to compliance, audit, security and finance.

We recommend implementing minimum standards for the rules associated with each monkey. While application teams should be permitted to change rules for their application prior to cloud development, those values cannot fall below the minimums. Equally, applications without specific rules inherit a standard set of rules and values.

To illustrate this approach, consider the following examples:

- Central cloud teams can be responsible for setting minimum standards for Chaos Monkey and Doctor monkey. Application teams can then choose to increase those standards if they need to ensure more stringent levels of availability.
- Compliance and information security teams set baseline rules for Conformity and Security Monkeys, again, with individual application teams able to set more stringent rules if dealing with sensitive data sets.
- Budget holders can set baseline requirements for Janitor Monkey. Individual application teams can configure stricter rules if working to tighter budgets.

For more information about setting up and running projects to implement this successfully, please contact Citihub at enquiries@citihub.com

Build versus Buy?

The Simian Army offers a powerful set of tools to support any organisation adopting cloud services. However, any decision to implement such tools should be benchmarked against comparable commercial solutions in the market and/or other open source toolsets.

Some key factors to consider when determining whether to buy, build or integrate an open source toolset, include:

Breadth in Functionality. The Simian Army can service a broad range of requirements covering availability, security, compliance and cost. Commercial solutions may have more well-developed functionality in any one of these areas but are unlikely to offer comparable breadth. A decision to 'buy' is therefore likely to require more than one product, which could result in complications with integration and additional cost.

Level of Integration Required. Implementing the Simian Army will inevitably require some integration with existing systems management tools, particularly for a regulated organisation. Commercial solutions are likely to be more difficult to integrate given that access to the code base will be limited. Alternatives can include other open source projects such as the Cloud Custodian initiative or Spinnaker/Swabbie that are either targeted at financial services use cases or pre-integrated with continuous delivery tools, respectively.

Support Model. As with any vendor evaluation, organisations need to consider the level of support on offer, including the speed with which partners respond to change requests, and balance this against the responsiveness of their own in-house IT organisations.

Cost. Perhaps the most obvious advantage of opting for an open source set of tools is that it negates the need to license commercial software.

Out-of-the-Box Compliance. Many of the commercial solutions addressing continuous compliance claim to offer out-of-the-box compliance with certain standards. Most large financial institutions will need to comply with multiple standards and regulatory requirements, therefore most have evolved their own control framework designed to meet (and in some cases exceed) the superset of those requirements. Commercial solutions often therefore require significant customisation to configure the desired set of controls. Equally, from the perspective of conformity, cloud service providers offer a number of their own tools to help consumers apply restrict behaviours that contravene policy such as Azure Policy & Blueprints, AWS Service Control Policy, Config & RBAC and GCP Organization Policy.

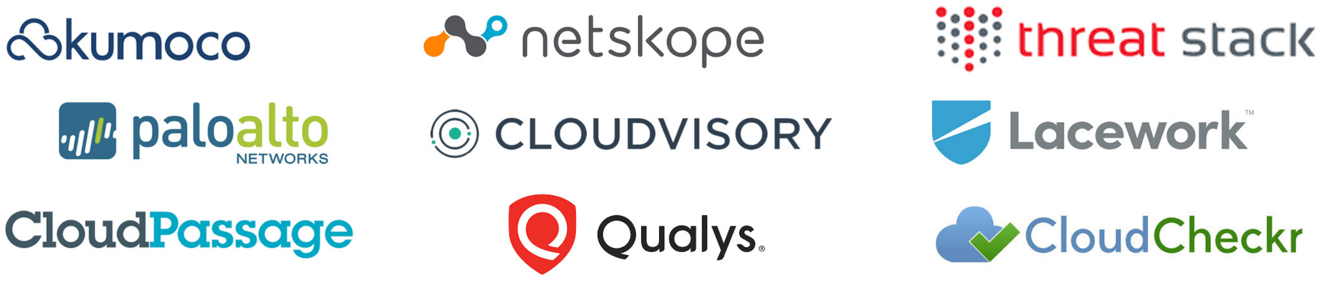


FIGURE 3: SAMPLE OF VENDORS OFFERING CONTINUOUS COMPLIANCE SOLUTIONS

Conclusion

As part of the DevOps ‘shift left’ philosophy, time needs to be invested early in the SDLC to ensure the availability of applications, as well as controls over security, conformity and cost are all thought through, implemented accurately and maintained. In order to manage shifts in responsibility, organisations need the right set of tools at their disposal.

Irrespective of whether one decides to license commercial software, build in-house or integrate open source tools, there is a clear need to implement many of the concepts introduced by the Simian Army.

By introducing chaos engineering principles early in the SDLC, organisations will be better able to enforce requirements to ensure the ‘built-to-fail’ mantra is properly implemented.

Equally, continuous compliance is a reaction to the greater responsibilities taken on by application teams and the immediacy of service provision offered in the cloud. Just as automation developed by CSPs has made resources faster to provision, automation needs to be introduced to ensure those resources are provisioned properly, in a way that conforms with all relevant policies, upholds security and is not wasteful.

Citihub is an industry leader in cloud control initiatives and has hands-on experience specifically implementing Simian Army tools, as well as broader chaos engineering and continuous compliance principles using diverse tool sets. For more information please email us at enquiries@citihub.com

About
Citihub Digital

Recoding the Digital DNA
of Financial Services

Citihub specializes in digital transformation for financial services. We are passionate about building more agile, dynamic companies by harnessing the full power of cloud technologies. Together with our clients, we solve some of the industry's toughest challenges.

Digital Natives, Born in Financial Services

We thrive in the complex, highly-regulated environment of financial services. We're intimate with the industry's application ecosystems, from high frequency trading to retail payments services, and have deep experience resolving security, risk and compliance challenges. We bring domain-specific methods and IP to every project.

Going Beyond Technology

The work we do reaches far beyond technology; it requires fundamental changes to the operating model, processes and culture of an organization. Our blended teams of technologists and industry specialists span diverse functions and disciplines, acting as catalysts to bridge silos and maximize the value of your own team.

Building Lasting Relationships

We work with 9/10 of the biggest banks in North America and Europe. Our clients trust us to deliver and they stay with us because we do. We've had year-on-year relationships with our top 15 clients for an average of 8 years.