CHEF™

Compliance at Velocity

10/2015

Compliance at Velocity

Executive Summary

Regulatory compliance is a fact of life for every enterprise. At the same time, competitive pressures are increasing with the advent of game-changing new technologies and customer expectations for digital services. Is it possible for regulated industries to deliver new products and services at high velocity while still satisfying their obligations for regulatory compliance? While it is often thought that compliance always puts a drag on velocity, in this paper we demonstrate that compliance at velocity is not only a possibility but a reality.

The solution is to embed regulatory compliance into the *software production line* in the same way we embed other qualities, such as frame stiffness in automobiles or round-trip response time in banking applications. Compliance is no longer a detour. It's engineered into every step. Like a production line with robots and state-of-the-art sensors, compliance at velocity uses extensive automation to increase velocity and accuracy.

Compliance at velocity is based on the idea of *infrastructure as code,* which allows an enterprise to specify its compliance-related requirements in ways that can be automatically tested. Not only does automation increase velocity, it also makes it possible to consistently apply regulatory requirements in large-scale environments that may include many thousands or tens of thousands of servers.

Chef is an example of an automation platform that lets you manage compliance. The Chef Compliance server lets you write rules that express your requirements, and then uses those rules to test your infrastructure for noncompliant configurations and out-of-date software. Once problems are identified, you can use the Chef server to deploy corrections. For a completely automated workflow, you can use Chef Delivery to test and propagate your changes.

Implementing an automation cycle using modern development practices sets the stage for an enterprise to become a *coded business*, one that is nimble enough to compete in the age of digital everything. This is a truly surprising result: solving regulatory problems can also help address some of the most pressing competitive pressures you face.

The Conflict between Compliance and Velocity

In every large company, whether industrial or financial, software is playing an increasingly central role. Software-based services are often now the primary means of contact between a company and its customers; IT is no longer a back-office support function. From sophisticated banking services accessed entirely through mobile phones and browsers to automobiles differentiated in the market by how well they integrate with the consumer's technology ecosystem, companies are under pressure to deliver new digital services at unprecedented velocity. Software is eating the world. This is the new normal.

"We're not an airline. We're a software company with wings.[1]"

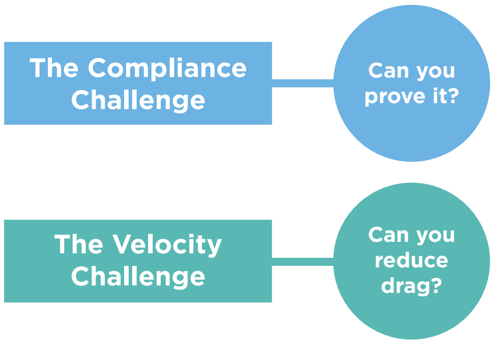
– CIO of a major U.S. air carrier

At the same time, regulators are placing increasing focus on detailed compliance. The sheer number of compliance frameworks is daunting, as is the proliferation of detailed requirements within each regulatory framework. Enterprises have more motivation than ever to reconcile the conflict between complying with regulatory requirements and competing in the fast-moving digital marketplace.

Here are just a few examples of compliance frameworks.

Office of Foreign Assets Control regulations (OFAC).[2] Enforces economic and trade sanctions.	**USA PATRIOT Act**.[3] Requires business cooperation in the U.S. for national security and anti-terrorism.	**Gramm-Leach-Bliley Act**.[4] Governs information sharing and safeguarding of customer data by financial services companies in the U.S.
Red Flags Rule.[5] Requires identity theft protections for consumers.	**Bank Secrecy Act**.[6] Requires U.S. financial institutions to assist U.S. government agencies to detect and prevent money laundering.	**Sarbanes-Oxley**.[7] Financial reporting standards for all U.S. public company boards, management and accounting firms.
Regulation E.[8] Regulations for electronic funds transfers.	**Dodd-Frank**.[9] A major overhaul of the U.S. financial regulatory framework.	**False Claims Act (FCA)**.[10] Requires accuracy when reporting information to the U.S. government.
Health Insurance Portability and Accountability Act of 1996 (HIPAA).[11] Comprehensive rules for health care providers in the area of patient privacy.	**European Central Bank**[12] regulations. The ECB took over supervisory responsibility for banks in the euro area in November 2014.	**Prudential Regulation Authority**.[13] Standards and regulations for banking and financial services, credit unions, insurers and investment firms in the United Kingdom.
Financial Conduct Authority[14] regulations. Standards and regulations for firms providing financial services to consumers in the United Kingdom.	**Health Information Technology for Economic and Clinical Health** (HITECH).[15] Requirements for information systems that contain medical records.	**PCI Data Security Standards**.[16] Standards for payment card data security.

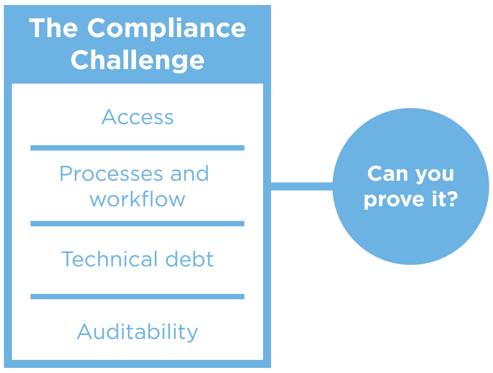
Of course, any snapshot of the current compliance landscape won't match next year's, or even next month's. Regulations evolve. For example, the Dodd-Frank Act includes provisions that are still in the process of being defined. Any approach to compliance must be agile enough to accommodate new and changing requirements.



The conflict between compliance and velocity reduces to two questions. In essence, the question of compliance is "Can you prove it?" and for velocity it's "Can you reduce drag?" Reducing drag increases velocity and translates into practical advantages, such as reducing time to market or increasing the number of services delivered within a certain time frame.

The Challenge of Compliance

Regulatory compliance is a large subject. In overview, we can say that it includes providing access, ensuring the appropriateness of processes and workflow, managing technical debt and ensuring auditability.



Providing access

Access means letting people use information or systems for which they are authorized and protecting information from unauthorized use. It includes establishing user identity through credentials. It includes access control at both the personal and system levels. Additionally, it includes requirements for making those systems available to users when needed. The key issue is, "In order to do task *x*, can I access the systems *when I need to*?"

Providing access is not a simple requirement to meet. It can involve significant effort and complexity. For example, recent regulations require named control over third-party access to enterprise systems. In other words, when a company outsources IT functions, it is no longer allowed for the outsourcing vendor to have access credentials that are shared by its employees collectively. Instead, each employee of the vendor company must have individual access credentials, which must be created and maintained.

This requirement can make it a challenge for an enterprise with any level of outsourcing to be able to manage the network perimeter at the required level of detail. Typical solutions include identity proxy servers at the edges of their networks, and these require new processes and information flows.

Creating appropriate processes and workflows

Providing access is just the start. Processes and workflows must also ensure that information and systems are used in acceptable and appropriate ways. The key question is "Are tasks done in the correct sequence?"

For example, banks are required to prove four-eyes accountability (the requirement that two individuals approve an action before it can be taken) in their processes. Their processes and workflows must ensure that individuals cannot misdirect transactions for personal gain.

Service management is another example of a workflow with compliance implications. The traceability of service tickets in some ITIL-based service management workflows satisfy the regulatory requirement for documenting system changes. Such systems can be effective but they are often slow moving and poorly adapted to high-velocity development of large-scale systems.

Managing technical debt

Maintenance and upgrades are additional challenges for compliant systems.

Technical debt, as it applies to IT infrastructure, is the need to maintain systems and upgrade them over time. Technical debt is deferred maintenance, and it starts accruing the day a system is launched. It includes both the ongoing costs of maintaining infrastructure by purchasing licenses and using suitable versions of software. For example, even though there may not be a licensing issue associated with using an outdated version of Apache Web Server, there can be security concerns that can only be addressed through an upgrade.

Technical debt is often a problem for established information systems that are no longer undergoing significant maintenance and upgrades. Such systems often use older or sometimes obsolete versions of operating systems and other software. Because they are less frequently maintained, these older systems tend to accrue significant technical debt. They are often inadequately documented and difficult to maintain and change. They will eventually be replaced, but until that time, there must be a strategy in place for keeping them in compliance.

One of the problems of technical debt is that it is magnified by scale. When an enterprise has a large number of servers to maintain, using service management tickets to keep them up to date is no longer feasible. When there are a large number of systems to maintain, manual update processes always allow technical debt to accrue. In addition, extended support contracts are often expensive and may require specialized internal or external resources.

Ensuring auditability

It is not enough to meet regulatory requirements; you must be able to prove that they are being met. You must ensure auditability.

For access, you must be able to prove that access is controlled and that X people accessed Y systems at some time for a particular, appropriate reason that had certain expected consequences.

For processes and workflows, you must be able to show that acceptable tasks occurred in the right order.

For systems and infrastructure, you must be able to show that updates and security patches were consistently and fully applied to address known vulnerabilities and to adhere to policy.

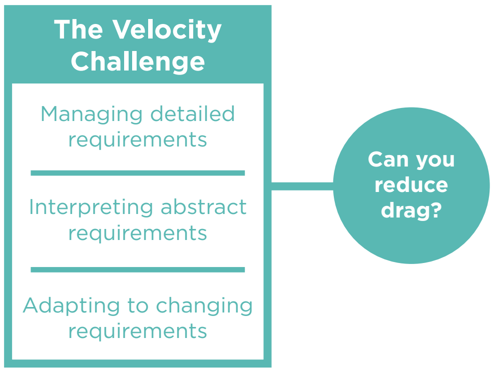
It is very common that infrastructure changes are not fully documented. Undocumented changes can occur when informal change requests are manually implemented. This causes a phenomenon called "configuration drift," where the state of production systems becomes unknown or inconsistent with policy. Configuration drift occurs even when regulatory frameworks require all access and changes to production systems to be logged and auditable—who made the change, when the change was made and what change occurred.

Confidence

A more difficult question than "Can you prove it?" is "How do you know that you know?" This second question is a measure of your confidence in the processes and procedures that produce the proof. For instance, you may be able to prove the effectiveness of a number of methods for preventing data leaks, but your confidence that you are actually covering all or most of the possible cases is likely significantly lower.

The Challenges of Velocity

The challenges of velocity are coping with large numbers of detailed regulatory requirements (the "granularity" problem), being empowered to interpret abstract requirements into specific policies and adapting to changing requirements.


The Velocity Challenge
Managing detailed requirements
Interpreting abstract requirements
Adapting to changing requirements
Can you reduce drag?

What is Velocity?

Speed is a scalar. That means that the distance you cover in any direction is an adequate measure of progress. An example in large enterprises is the fairly common lament that, "The team made great progress until the securocrats got involved and decided to add requirements, so we were two months late to market." Great speed initially but sadly in the wrong direction. Velocity is a vector—in other words, speed in a specified direction. When traveling the great coast road outside Melbourne, Australia you may keep your speed constant, but your velocity changes every 10 meters! For the enterprise, the goal is rapid, forward velocity.

Managing detailed requirements

Regulatory bodies are demanding compliance that has ever finer-grained requirements. This is a problem of increasing granularity. There was a time when banks only had to worry about employees leaving customer printouts on the train. Now, any data that is traceable to a customer must be inaccessible without the customer's approval. Some banks even require finger-vein or card-based authorization from the customer before allowing a cashier to access the records.

Over the last ten years the regulations controlling the use of personal customer information have matured and become much more detailed. Depending on the geographical region, there may be restrictions about which elements of personal customer data may be shared. These restrictions can limit how data can be shared with other systems within the enterprise, can be shared with partners, can be transmitted outside of the country of origin and can be transmitted in unencrypted form. Data elements like name, account number and password always have to be encrypted.

The detailed requirements for handling personally identifiable information are especially challenging in complex systems. For example, a customer record shown to a bank teller can contain up to 200 types of data in order for the teller to interact with a customer meaningfully. The data can include information about where the customer resides, what the customer's house is like, how many bedrooms it has, and so on. Businesses have had to start classifying data at field level, with some fields marked as personal customer information and others not. This kind of detailed data protection is required by PCI DSS and is relatively recent.

As the number of requirements increases, the process of checking them manually becomes overwhelming and error prone.

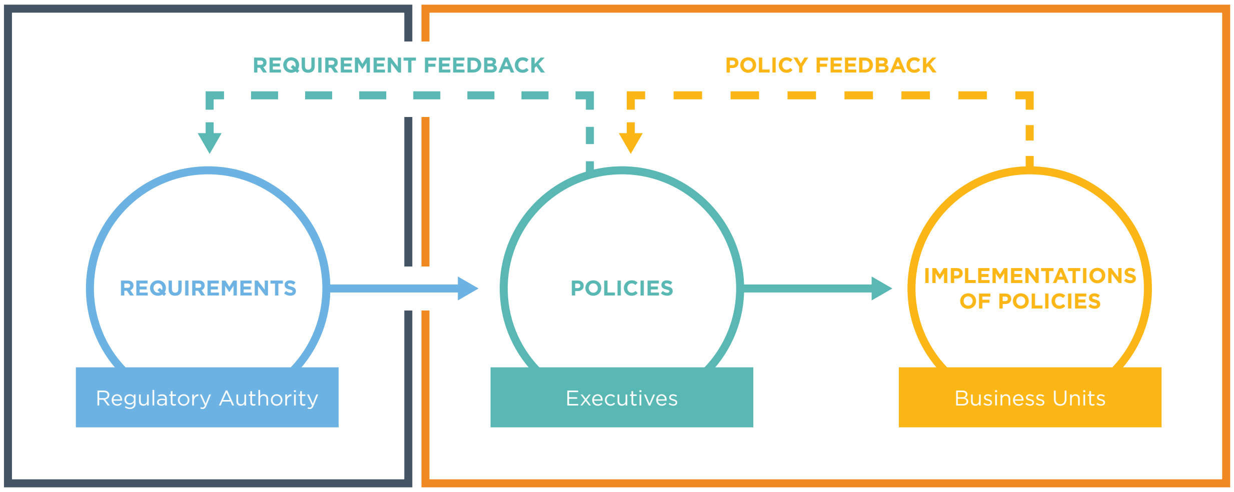
Interpreting abstract requirements

Abstract requirements pose the challenge of interpretation. To put them into effect, enterprises must translate these requirements into concrete policies. The policies themselves undergo a process of interpretation when the company's business units implement them.

The gap between abstract requirements and what it takes to implement them is too large for an approach that does not include the intermediate step of interpretation into policy. Making sure that policies and implementation choices are unambiguously communicated is also a challenge.

For example, a requirement might state that “only users and automated processes that need access to a server should be allowed that access.” This requirement sounds concrete, but in a large system there are many resources to access and many applications that need access. It’s actually an abstract requirement. There are a number of ways a company might develop policies and implementations that meet the top-level requirement.

Enterprises must recognize when they are facing abstract requirements and take control of the situation. Trade-offs must sometimes be made; you cannot refer every question to a regulator. Clarifying the ownership of these decisions is critical in order to enable the enterprise to take action when abstract requirements are encountered. Further, organizations need a framework for prioritizing and planning the trade-offs they will make. Economic methods, such as Cost of Delay, need to underpin the decision-making process.



The requirements of regulatory authorities and the policies created by executives are not one-way communications. Feedback occurs at each level. Enterprises lobby for pragmatism both before and after regulations are known, and regulations evolve as regulators learn from experience in the wild. Policies may change as executives get feedback from regulators and from the experiences of policy implementers in their organizations.

The requirements of regulatory authorities and the policies created by executives are not one-way communications. Feedback occurs at each level. Enterprises lobby for pragmatism both before and after regulations are known, and regulations evolve as regulators learn from experience in the wild. Policies may change as executives get feedback from regulators and from the experiences of policy implementers in their organizations.

The regulatory conversation

The US FFIEC position paper on banking in the cloud (Outsourced Cloud Computing Statement) is a good example of the conversation that occurs when enterprises lobby for pragmatism. Although there have been many subsequent statements on cyber-security, clarity on what cloud-based banking may mean is still being worked out and is going to be largely a function of a bank’s ability to convince regulators of its confidence in non-colocated infrastructure. This does not mean that the FFIEC and other regulators are standing still. Discussions are occurring at every level, but how lessons at each of these levels will ultimately cohere into firm regulation is unknown.

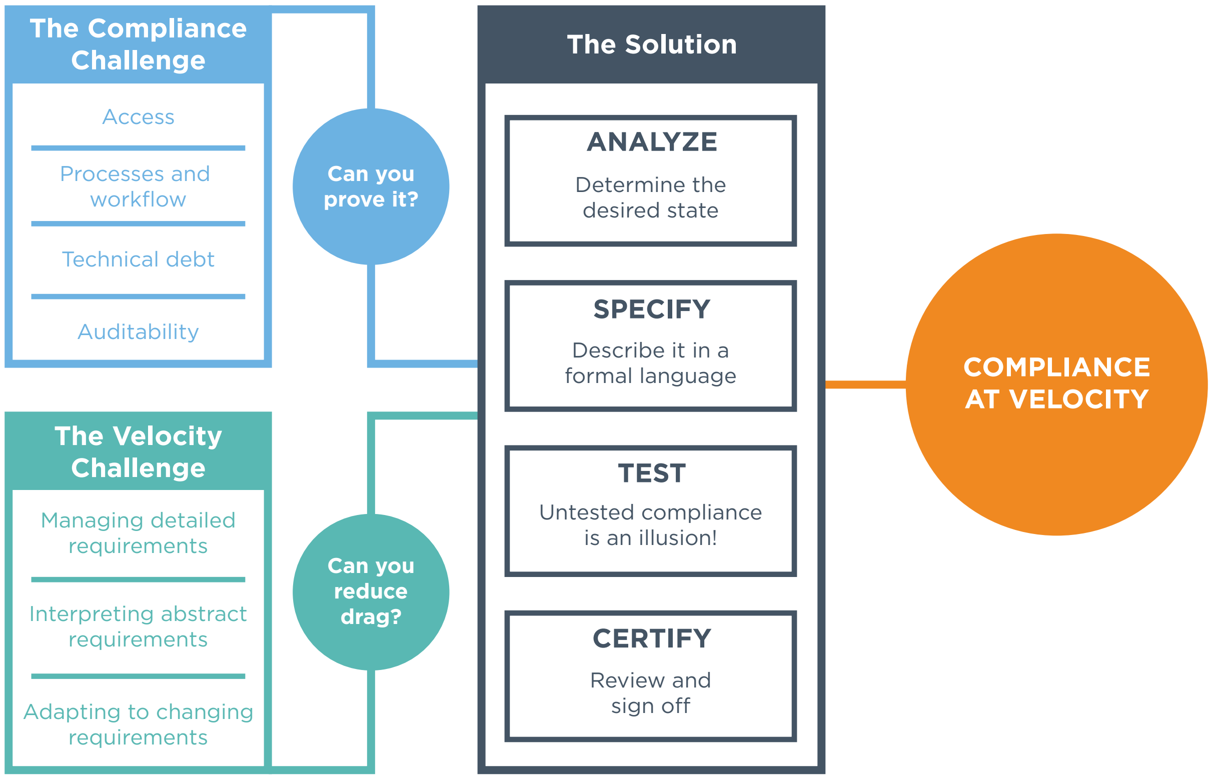
Adapting to changing requirements

In some cases, such as Dodd-Frank in the United States, regulatory provisions are still evolving. By some estimates, only 40% of the eventual requirements have been finalized to date. Any approach to compliance must be able to accommodate new and changing requirements.

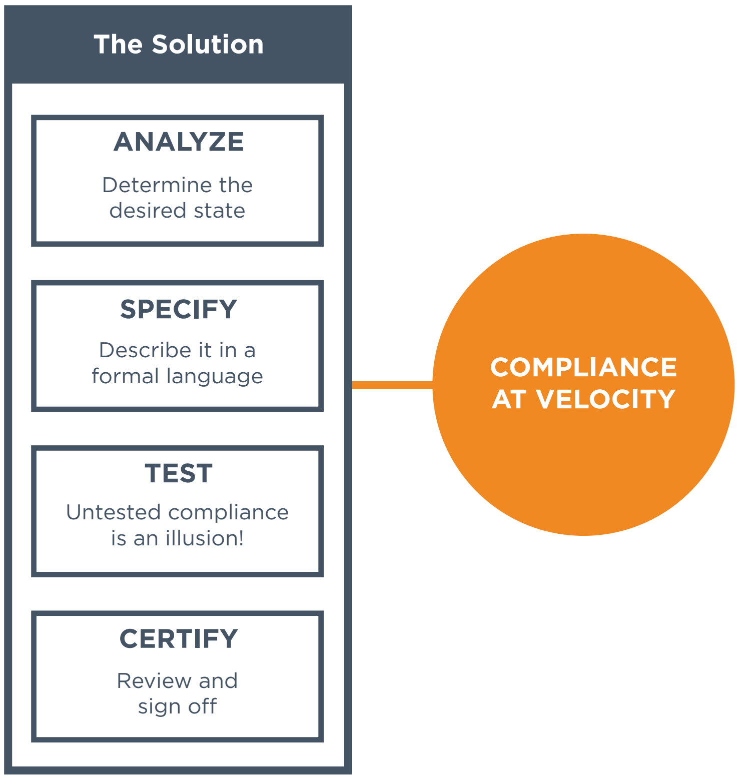
In addition, emerging regulations generally include a time frame. Organizations must remediate the gaps between the requirements and their current systems within a fixed time. In some cases, a credible plan to remediate over a longer term is acceptable. Even when there may be some flexibility in the schedule for remediation, compliance is eventually required.

Reconciling Compliance and Velocity

Compliance and velocity can be reconciled by embedding compliance into the software production line in the same way we embed other qualities, such as frame stiffness in cars or round-trip response time in banking applications. Compliance should not be a detour.



For example, it is helpful to think of compliance as a framework that allows us to refine requirements over time, as we better understand the demands placed on the enterprise. This allows us to put measures in place that evolve as the intentions of the regulators become clearer or change. It also allows an enterprise to perform compliance *at velocity*.


The Solution
ANALYZE
Determine the desired state
SPECIFY
Describe it in a formal language
TEST
Untested compliance is an illusion!
CERTIFY
Review and sign off
COMPLIANCE AT VELOCITY

The solution to the compliance at velocity problem is multifaceted. A fundamental component is an *automation cycle* that is part of a deployment pipeline for infrastructure. The cycle includes four steps. They are analyze, specify, test and certify.

Analyze: *Choose your desired state*

ANALYZE
Determine the desired state

The first stage of compliance at velocity is being clear about what the desired state actually is. Desired state is an implementation choice. Regulatory requirements and enterprise polices influence the decisions of implementers within business units as they design and build systems.

The idea of desired state configuration can be contrasted with what is known as checked state configuration. Checked state configuration uses some method (manual or automated) to check that what was built is in the expected state. Deviations from policy can be flagged or reported but not automatically repaired.

With desired state, the actions of the automation framework take the system as a whole closer to a specified goal. Elements that may have drifted from the desired state are corrected when the discrepancy is detected. In other words, builds and status checks are part of the same process, which is what we want for compliance at velocity.

Choosing the desired state and expressing it at an appropriate level of detail are more challenging problems than writing the automation code itself.

Specify: *State requirements in a formal language*

SPECIFY
Describe it in a formal language

Closing the gap between specifying and implementing regulations requires an unambiguous expression of the requirement in human- and machine-readable form. A formal language can achieve this level of clarity and precision.

A concrete example of a formal language is the one used by the Chef Compliance server. The server is a part of the Chef automation platform. It provides a language for creating rules that express your requirements. It then uses those rules to test the nodes in your network for problems.

DESCRIBING PCI DSS REQUIREMENTS WITH CHEF COMPLIANCE

The Payment Card Industry Data Security Standard (PCI DSS) is an information security standard for organizations that handle major credit cards. Its goal is to reduce credit card fraud by safeguarding cardholder data. The Chef Compliance language lets you express PCI DSS requirements as *rules*.

Here is a Chef Compliance rule that ensures that insecure services and protocols, such as `telnet`, are not used.

```
describe package('telnetd') do
 it { should_not be_installed }
end

describe inetd_conf do
 its("telnet") { should eq nil }
end
```

PCI DSS requires that cardholder data that is sent across open, public networks be encrypted. Here is a Chef Compliance rule that ensures that the web server is only listening on well-secured ports.

```
describe port(80) do
 it { should_not be_listening }
end

describe port(443) do
 it { should be_listening }
 its(‘protocol’) {should eq 'tcp'}
end
```

Here is a Chef Compliance rule that controls the available users for a server.

```
describe user('root') do
 it { should exist }
 it { should belong_to_group 'root' }
 its('uid') { should eq 0 }
 its('groups') { should eq ["root"] }
end

describe user('mysql') do
 it { should_not exist }
end
```

Compliance at velocity requires that members of different teams, such as development, operations, compliance and security, all have access to compliance rules. You can add metadata to those rules to ensure that everyone can understand the requirements. For example, here is a rule (or control) to specify that only SSH version 2 is acceptable.

```
control "sshd-11" do
 impact 1.0
 title "Server: Set protocol version to SSHv2"
 desc "
   Set the SSH protocol version to 2. Don't use legacy
   insecure SSHv1 connections anymore.
 "
 describe sshd_conf do
   its('Protocol') { should eq('2') }
 end
end
```

Here is a rule that ensures that only enterprise-compliant ciphers are used for SSH servers.

```
describe sshd_config do
   its('Ciphers') { should eq('chacha20-poly1305@openssh.com,aes256-
ctr,aes192-ctr,aes128-ctr') }
end
```

Test: *Untested compliance is an illusion*

Test-driven development (TDD) is a proven approach to designing software. Formally defined, automated tests are written first, and the software is developed until all the tests pass. The advantage of this approach is that what you are testing for and what success looks like are explicitly defined. Feature-level components of compliance (for example, the ability for a website to be translated for vision-impaired users in Europe) should be tested this way. TDD is a great way to be sure that 'you know that you' know you are compliant.

TEST

Untested compliance
is an illusion!

Infrastructure that is described as code is also testable by means of an automated process. Every requirement, especially those related to compliance, can be expressed as a test. Apart from the usual advantages, the test results act as a record of your interpretation of a particular requirement. If the only way that a system is configured is through an automation server, auditors then have a reliable record of exactly how you are satisfying compliance. This is taking control of the situation.

By defining compliance requirements as testable code, compliance professionals, developers and system administrators have a clear set of standards that must be met for compliant systems.

Certify: *Review and sign off*

Incorporating the tests for compliance in the production pipeline means that at the

CERTIFY

Review and sign off

time of deployment, a separate certification step is not always required. The automated tests give confidence that the requirements have been met. However, in some cases, regulatory requirements or organizational processes do require a final human sign off before promotion to production. The timeliness of the sign-off indicates how well the compliance officers have specified the desired state. The clearer the specification, the faster the software is released to production. The automation cycle changes the role of compliance officers from being interested observers of the development process to being invested contributors to the velocity of deployment.

Separate certification from testing

One surefire way of slowing down production velocity is to confuse the process of certification, required in some industries, with the process of automated testing of applications and infrastructure. Such testing is based on the system's desired state expressed in a human- and machine-understandable form.

Those usually responsible for certification, which might include groups such as Security, Internal Audit, and IT Audit, should be responsible for agreeing to the scope of testable compliance criteria for each system release cycle. Software developers and system administrators are then responsible for implementing features that pass these tests, just like any other feature required by a business user or product owner.

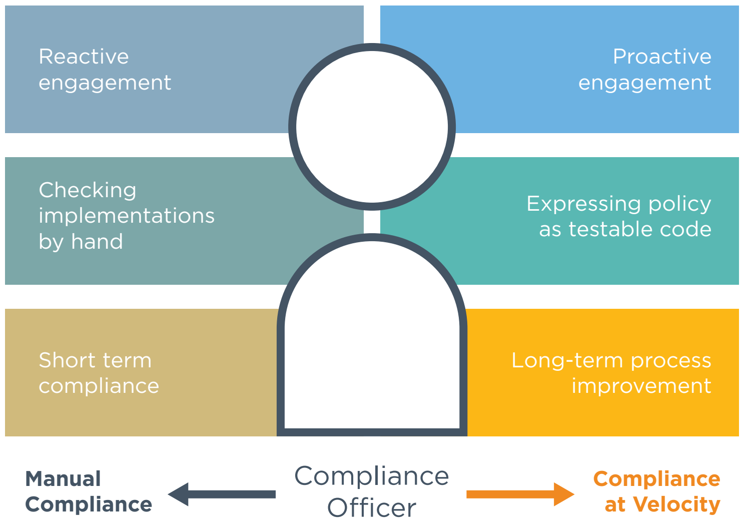
Separating compliance testing from sign off, and automating such testing, is a major contributor to fast and scalable compliance.

The Role of the Compliance Officer

The compliance officer's role changes with automation. Without automated testing, the effort of compliance is spent on checking for what breaks rules rather than on formulating the clearest and most effective compliance rules for the enterprise. When rules are imprecisely specified, the compliance officer's talents are wasted on many individual conversations about whether a particular implementation meets the requirements.

Another factor is that, without a high-velocity compliance process, the compliance officer has a relatively long time frame within which to establish compliance. Over some period, data is collected and scans run, or analysis performed and the effectiveness of the organization at driving compliance is evaluated. The compliance officer spends his/her time between longer-term estimates of what can realistically get done and short-term panic at how little has stuck.

The diagram below shows how the compliance officer's role changes and becomes increasingly significant when enterprises aim for compliance at velocity.

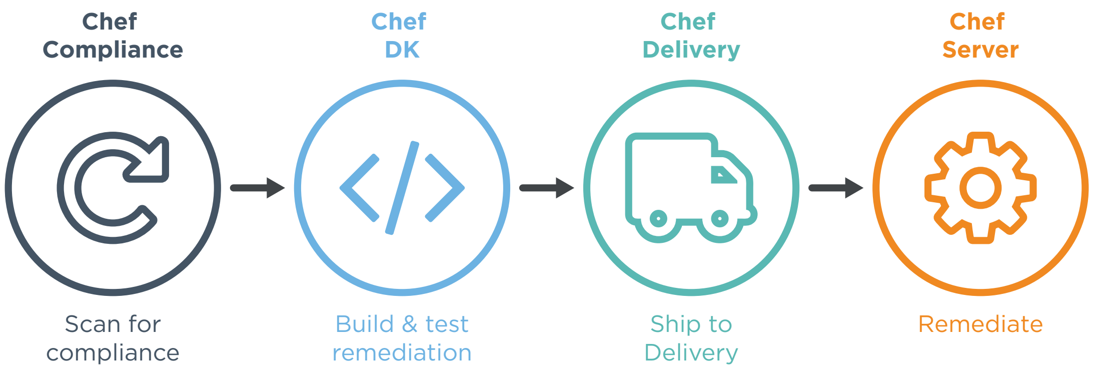


When compliance is supported by automated testing and integrated in the development process itself, compliance officers no longer need to focus on stopping people from breaking rules. Instead, they make very enterprise-specific rules and embed them in the development process. Instead of being seen as drags on velocity, compliance officers become a critical part of the enterprise's transformation into a high velocity production line.

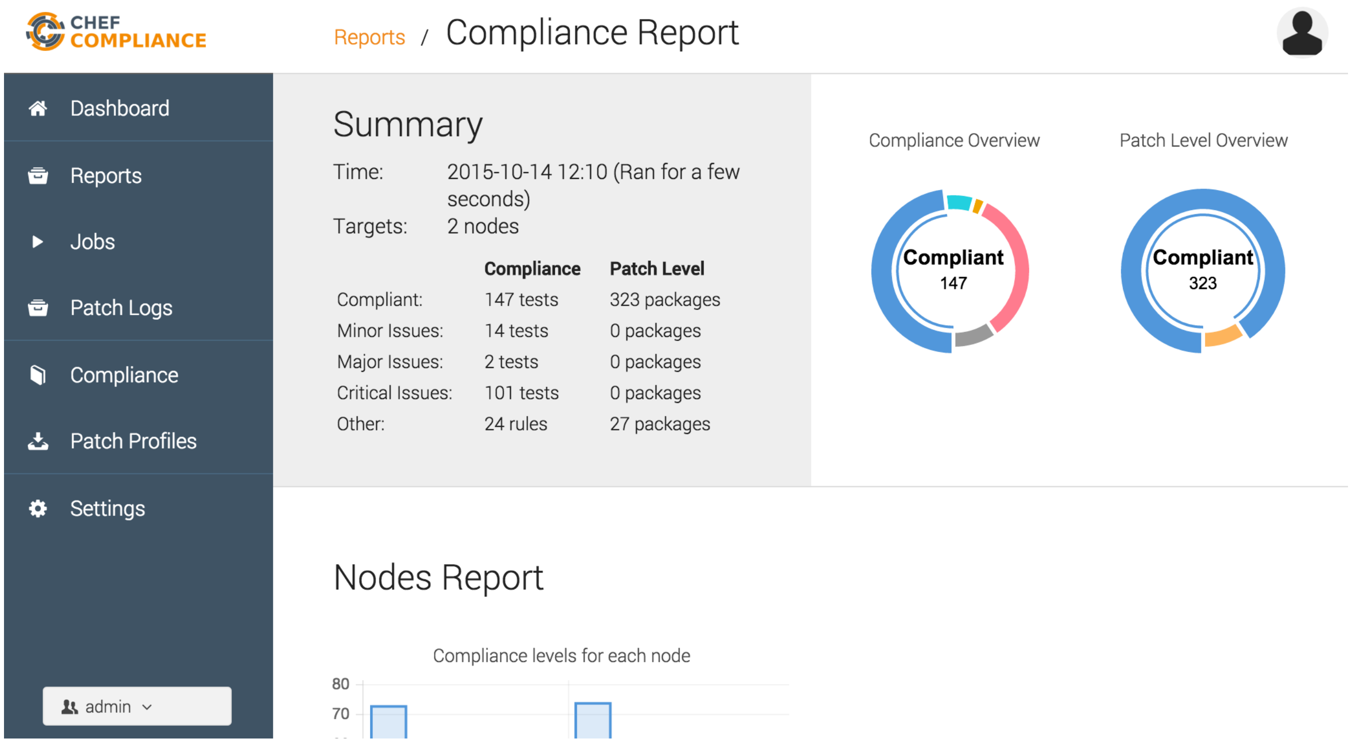
Bringing together the compliance officer and the development and operations teams improves the quality and throughput of all. This interaction has many benefits. [A good overview is chapter 12 of *Lean Enterprise: How High Performance Organizations Innovate at Scale* by Jez Humble, JoAnne Molesky and Barry O'Reilly.]

An End-to-End Example

Patch management is one of the most critical aspects of IT security. It is important that you be able to identify out-of-date systems and upgrade them. Most regulatory frameworks, such as PCI DSS, require it. You can use Chef to manage patches, throughout the workflow, which is shown here.



Beginning with the Chef Compliance server, you can scan your nodes to see if they are compliant and their software is up to date. You'll receive a report telling you the status of your infrastructure. Here is an example of a report from the Chef Compliance dashboard.



Summary

Time: 2015-10-14 12:10 (Ran for a few seconds)

Targets: 2 nodes

	Compliance	Patch Level
Compliant:	147 tests	323 packages
Minor Issues:	14 tests	0 packages
Major Issues:	2 tests	0 packages
Critical Issues:	101 tests	0 packages
Other:	24 rules	27 packages

Nodes Report

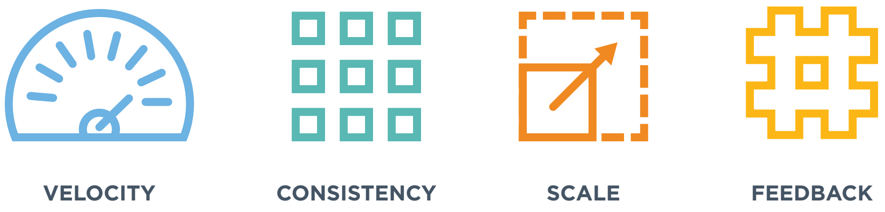
Once you have the report, you can use Chef DK to begin to build and test the remediation. Chef DK contains all the tools you need to create and test your code on your workstation.

You can then send your changes to Chef Delivery (Delivery). Delivery provides a pipeline for deploying changes. The pipeline contains stages for testing your changes and making sure they work. Within the pipeline are two manual gates. One of them is for code review, and the other sends the code to the release environments. You can involve compliance and security officers at either or both of these points, to make sure they are actively engaged in the release process.

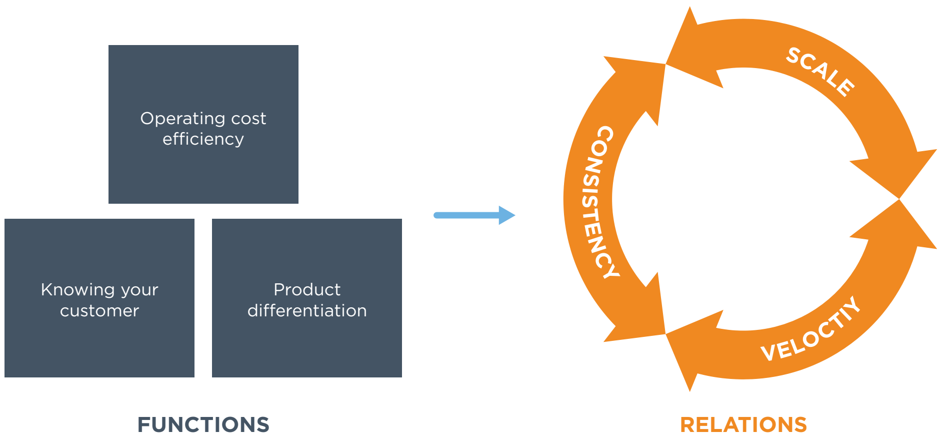
Once the changes have passed all the stages in the Delivery pipeline, you can send them to the Chef server. The Chef server can then begin to bring the nodes up to date.

The Promise of the Coded Business

Operating at velocity has benefits in addition to agile, scalable compliance. The processes, culture and technology used for compliance at velocity also give an enterprise a level of velocity, scale and consistency that lets it compete in the digital age. We refer to such an enterprise as a "coded business".



The coded business collapses the traditionally distinct strategic focus areas of customer intimacy, operational efficiency and product improvement into a single accelerated cycle. We refer to this as the automation cycle, because the automation of the process throughout the product delivery lifecycle is core to its effectiveness.



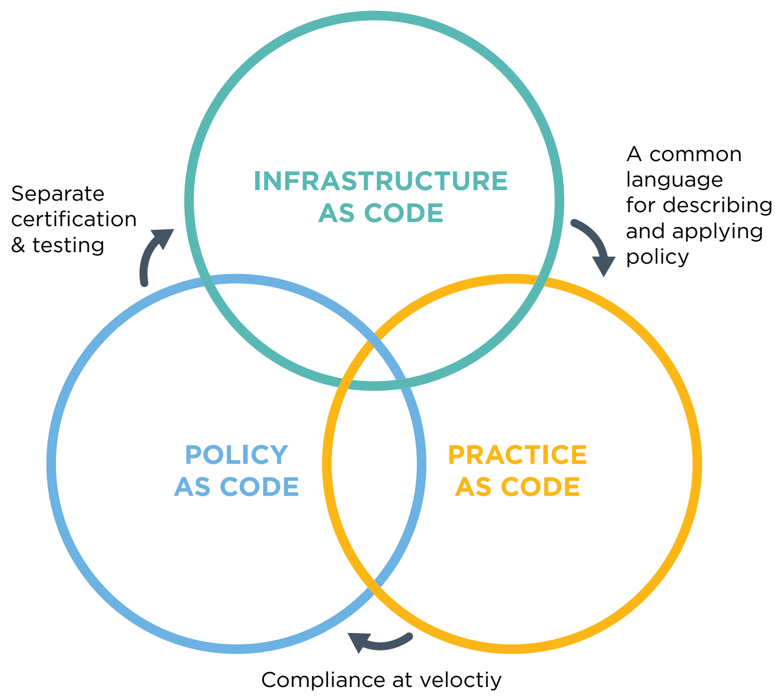
For example, automation lets you patch a collection of servers consistently. Every server node that performs a given role has exactly the same configuration and maintains that configuration. Consistency enables an organization to make assumptions about the state of the server at any future time. This means, in turn, that initiatives to apply future patches do not have to account for a period of rediscovery (and the cost of the software used to do this).

With automation, developers can be assured of the exact status of the environments to which they will be deploying, removing the frequent cycle of feedback and tuning that occurs in the opposite case. All of this enables the business to get to production faster and see the performance and use of the system 'in the wild.'

In production, we learn about the patterns of use of the system and discover how some elements that may be built into the consistent infrastructure definition may not be ideal. Given that we use automation and have a fast, repeatable process for improving the standard consistent infrastructure design, we can quickly apply the required changes to it. In other words, with automation we achieve scale where scale refers not to size but to the appropriateness of resources allocated.

Like the large Internet innovators that increasingly have the attention of today's consumers, a coded business is at its core an agile software business, regardless of whatever other goods or services it might provide. A coded business is capable of operating at massive scale without losing its ability to innovate and respond quickly to changes in the marketplace.

A coded business is also able to operate at high velocity within a highly regulated environment. Counter-intuitively, the practices that it puts in place to enable the specification and testing of compliance are the same practices that enable it to operate at high velocity. These practices are shown in this diagram.


INFRASTRUCTURE AS CODE
POLICY AS CODE
PRACTICE AS CODE
Separate certification & testing
A common language for describing and applying policy
Compliance at veloctiy

Core among these practices is the concept of a common language for expressing compliance requirements and the automation of compliance checking. The coded business has relentless focus on automation of practices (processes and workflow), policy (compliance) and computing infrastructure.

Key Points

- Demands for compliance are growing, even as companies are under pressure to deliver new products and services at velocity.
- To be compliant means maintaining control over access to resources, processes and technical debt.
- You must ensure auditability to prove that you are compliant.
- Compliance frameworks are constantly becoming more detailed and are open to interpretation.
- To be compliant and to move at velocity seems impossible.
- The solution is to embed compliance into the production pipeline.
- There are four steps towards compliance at velocity: analyze, specify, test, and certify.
- *Analyze* to decide what you want, *specify* to express what you want as code, automatically *test* that your policies are being followed and then sign off.
- Automation platforms, such as Chef, are the basis for compliance at velocity.
- With automation the compliance officer's role becomes proactive rather than reactive.
- The coded business uses automation to express infrastructure, policy and practice.
- The coded business delivers velocity, scale and consistency while maintaining compliance.

NOTES

1. Personal communication.
2. http://www.treasury.gov/about/organizational-structure/offices/Pages/Office-of-Foreign-Assets-Control.aspx
3. http://www.fincen.gov/statutes_regs/patriot/
4. http://www.business.ftc.gov/privacy-and-security/gramm-leach-bliley-act
5. http://www.business.ftc.gov/privacy-and-security/red-flags-rule
6. http://www.fincen.gov/statutes_regs/bsa/
7. http://en.wikipedia.org/wiki/Sarbanes%E2%80%93Oxley_Act
8. http://www.federalreserve.gov/bankinforeg/regecg.htm
9. http://www.sec.gov/spotlight/dodd-frank.shtml
10. http://www.justice.gov/sites/default/files/civil/legacy/2011/04/22/C-FRAUDS_FCA_Primer.pdf
11. http://www.hhs.gov/ocr/privacy/
12. https://www.ecb.europa.eu/ecb/legal/ssm/framework/html/index.en.html
13. http://www.bankofengland.co.uk/pra/Pages/default.aspx
14. http://www.fca.org.uk/firms/being-regulated/meeting-your-obligations
15. http://www.hhs.gov/ocr/privacy/hipaa/administrative/enforcementrule/hitechenforcementifr.html
16. https://www.pcisecuritystandards.org/security_standards/index.php
17. https://www.ffiec.gov/press/pr071012.htm