DevOps in Regulated Environments

Achieving Continuous Compliance at Speed
Balancing Risk, Speed, Feedback and Control
Jim Bird
Disclaimers

• I am not a lawyer, and I’m not offering legal advice

• I’m not representing my employer
Then why listen to me?

• 20+ years experience in global financial services and fintech
• Worked with stock exchanges, investment banks, and central banks in 30 countries
• CTO of a major institutional trading platform
• Experience with many different regulators
In my spare time...

- SANS Analyst and co-author of SANS DEV-534 Secure DevOps
- OWASP (Cheat Sheet series, Proactive Controls co-lead)
- Write about AppSec, Agile and DevOps: books and blogs
Finding the right balance between speed, cost, feedback and control, when meeting your regulatory obligations

ACHIEVING COMPLIANCE
Drivers and Conflicts

- **Speed**: Agile, MVPs, hypothesis-driven design, Continuous Deployment and Continuous Flow. Cycle time to delivery and velocity – compliance can’t block delivery
- **Cost**: minimize cost of change, eliminate waste and bottlenecks and delays and overhead, Lean optimization, working software over documentation. Keeping compliance and auditing costs to a minimum. Scaling (up and down)
- **Feedback**: DevOps teams iterate and run experiments (A/B) in production with real users, experiment-driven design... lots of throw away changes... requires quick/measurable feedback. Compliance can’t get in the way of feedback
- **Control**: risk management and governance, management accountability, assurance, approvals, and auditing. ITIL and COBIT and NIST/ISO standards. Conflicts with continuously improving, self-managing, self-service DevOps teams
Compliance Approaches

**Prescriptive**
- Rules-Based: regulations tell you what you have to do, how often, what evidence is required
- You know what auditors are looking for
- You – and auditors – can build checklists
- Sets minimum standards – but discourages organizations from doing more than the minimum
- Too much in some areas, not enough in others
- Examples: PCI-DSS, safety regs, FISMA, FedRAMP

**Descriptive**
- Outcome-based: regulations tell you what you can/cannot do, but not how to do it
- Directional, appeals to “recognized best practices”, “adequate and reasonable”
- Compliance is in the opinion of the auditor
- Allows for innovation, but…
  - ...creates uncertainty: are we doing “the right things”, are we doing enough?
- You will need to defend your program (and your assessment of risk) to auditors and others
- Pushes organizations (and auditors) towards recognized frameworks (ISO 27001, NIST 800-53, COBIT, SOC 2...)
- Examples: HIPAA, Reg SCI, SOX 404
What Auditors Look for…

- **Scope**: everyone clearly understands what data/systems/networks and what activities are in/out of scope
- **Privacy**: classifying, tracking and protecting private/confidential/sensitive data
- **Risk management**: top down (management accountable), continuous review and implementation, policy-driven and procedural, active (respond to new/changing threats)
- **Approvals**: management authorization, protection against malicious insiders and fraud
- **Separation of Duties**: no one person has control end-to-end, “need to know” access to data
- **Awareness**: regular training on compliance requirements, policies are reviewed/published, NDAs…
- **Assurance**: regular testing and evaluation of controls
- **Auditing**: evidence and traceability
- **Consistency**: walk the talk, internal and external reviews to identify exceptions (red flags)
- **Corrective Actions**: failures will happen (at least minor failures), but you must show that problems were understood and remediated
- **Sufficient and Appropriate**: structured and comprehensive controls, standards, certification
What you want from compliance

• **Just enough**: you don’t get an award for being “more compliant” – minimize costs and overhead
• **Free**: where possible, take advantage of work that people are already doing
• **Clear**: everyone should know when they are doing things right/wrong
• **Measurable**: you can tell if you are doing things right/wrong
• **Practical**: rules that people can and will follow
• **Shareable**: work can be reused across systems/teams
• **Consistent**: checks/tests that work every time
• **Non-blocking**: don’t stop people from getting useful work done
Compliance is about understanding, managing and minimizing risk – and proving that you are doing this

- Technical and project risks are handled implicitly in Agile (Scrum cycle + XP practices)
- Compliance requires explicit risk management (especially for security, privacy, legal risks) – add to backlog
- Top-down risk management: policies, control objectives, reviewed and communicated
- Upfront reviews: understand/assess technical, operational and compliance risks – Netflix’s “Paved Road”
- Operational risks: change/release control (small changes done often so change is routine, dark launching, canaries), automation, aligning test and production, metrics/monitoring to detect exceptions and failures
- Build and delivery pipelines provide control plane: build risk management into workflows around the pipeline (including reviews, testing, approvals, notifications)
- OWASP Top 10 for application security risks: awareness, training, scanning and reviews
- Retrospectives and postmortems: leverage to assess and improve, think of reviews and feedback loops as risk management controls, involve security/compliance, record outcomes
- Transparency and collaboration: reduce misunderstandings and missed requirements, solves problems better/faster, improves feedback – use as a risk management tool
Data Privacy for DevOps

- Understand privacy obligations upfront – and when rules change – legal requirements and your organization’s privacy notices
- Document your obligations and make sure that everyone understands them – reviews, training
- Legal requirements – translate into concrete requirements for teams – compliance stories and compliance in stories (review them with legal)
- Map out data and sensitivity levels – where data is created, updated, stored, accessed (everywhere – watch out for spreadsheets, tempfiles, caches, build artifacts…)
  - Minimize use of private data – only what is absolutely necessary
  - Crypto (detailed requirements), masking and tokenization (pseudo-anonymization…)
  - Access control
  - Auditing
  - Retention (and right to use, right to erasure)
- Be extremely careful when using production data for testing
- Ties directly to security… vulnerabilities/weaknesses compromise privacy, preventing and handling data breaches
Getting compliance out of policy statements, contracts and checklists and into code and tests

COMPLIANCE AS CODE
Compliance as Code

• Like “Infrastructure as Code”…. Get policies and procedures into code and automated tests
  – Implement policies in configuration recipes: provisioning and hardening (servers, databases, applications, firewalls....), pipeline workflows, scanning rules, check-in filters (secrets, banned functions), component/dependency analysis (OWASP Dependency-Check)
  – See dev-sec.io for hardening recipes (Chef, Puppet, Ansible) and SIMP Project (Puppet modules, open sourced in collaboration with NSA)
  – Automated tests to assert policies: SAST, DAST, compliance and vulnerability scanning, TDD/BDD (test first = compliance first), Gauntlt, InSpec/Serverspec
  – Runbooks become automated build, deployment and release orchestration instructions
• All code (including configuration and tests and test data) is checked in (revision history, approvals), reviewed, scanned, and tested (yes, you need to test your tests!)
• Code in repo becomes the single source of truth for your technical controls – protect it
• Compliance rules often detailed and fussy, so review/test carefully
• Automatic execution on every change: assurance, evidence, pipelines block if checks fail
• “DevOps Kata” – walk through deploying a change to a single line of code, trace every step
Writing code for auditors

- Auditors don’t (usually) understand code, engineers don’t understand why not
- Unit tests are not that useful, but unit test coverage is
- Declarative code is easier to audit – look at the end state, not all of the steps in between
- Engineers need to write clean, clear, well-structured code
- Annotate configuration recipes and playbooks (tie back to rules/policies)
Writing code for Auditors

InSpec: automated compliance checks

- Open source framework (like Serverspec) to write declarative compliance tests on Linux and Windows
- Also available for AWS, Azure and VMWare
- Annotations to define priority (based on risk/impact), and tie back to documented compliance controls
- Language for Chef Compliance product but you don’t need Chef to use it

```ruby
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."
  tag security: "level-1"
  tag "openssh-server"
  ref "Server Security Guide v.1.0" url: "http://..."

  describe sshd_config do
    its('Protocol') { should cmp 2 }
  end
end
```
Engineers and auditors

- Compliance as code creates “opportunities” for engineers and auditors to work together
- Engineers and auditors are methodical, stubborn, and certain that they are right
- Some auditors are sharp, engaged, practical, creative, good to work with; others... not so much
- Engineers need to understand the auditor’s goals, rules, priorities, and why the audit is necessary: make sure that engineers understand and buy in upfront
- Auditors are not friends: never try to hide mistakes or oversights, but don’t offer information that they don’t ask for, if in doubt refer questions to management
- Auditors are not enemies: don’t attack the process or audit standard, don’t try to intimidate them, don’t patronize them
- Auditor questions and challenges are not attacks on the engineer’s competency – don’t be defensive
- Clarify requirements, make sure that everyone is satisfied with the evidence – auditors will ask for evidence that worked for them in the past, but there may be a better/simpler way for everyone
- Patience (on both sides) is required
Integrating compliance into development/engineering workflows

CONTINUOUS COMPLIANCE
Compliance as Stories

• Provide the team training up front: especially product owners/managers so they understand compliance/privacy obligations/constraints
• Compliance and security stories
  – Translate compliance and legal requirements/constraints into concrete work for engineers to do, and problems for them to solve
  – Include compliance steps in “conditions of satisfaction” and “definition of done” for user stories – and write tests to check them (negative tests!)
  – Write security stories for security controls
  – Write compliance stories for required reports, other evidence, audits
  – Add to backlog, update as needed
• Make compliance requirements explicit, clear and visible to everyone
Continuous Compliance

- **Infrastructure provisioning/configuration through code**
  - Secure Baseline: manage configuration and patches, apply hardening rules automatically
  - Production and Testing: prove that environments are in sync
  - No snowflakes: reset automatically if config drifts or tear down/rebuild on every change (immutable)

- **CI/CD pipeline for all changes: repeatable and consistent, automatic tests and checks**
  - Repos to support everything as code (source and binary)
  - Runbooks into deployer rules and run-time checks
  - Move checklists into automated tests that pass/fail
  - Code reviews: leverage SAST and tools like Gerrit, review for risk not style (understandability, correctness – tie back to stories and conditions of acceptance, and safety/security – defensive coding)
  - Logs as evidence: log all steps, protect the logs and repos and build/delivery chain

- **Scan everything and scan often**
  - Run-time, code and dependencies, apps, containers, network
  - Feed findings into backlog to be remediated, and to GRC system for tracking (through APIs)
DevOps Audit Defense Toolkit

- Free, community built process framework based on a fictional organization (as described in “The Phoenix Project” book)
- Not recognized “best practice” but can be used as a template for continuous compliance
- Identify risks and controls, then map them to Continuous Deployment workflow
  - Track stories, bugs and vulnerabilities electronically for traceability (Jira tickets)
  - Check into repo using comment tag linking back to ticket
  - All code is scanned using approved tools and rules
  - Code reviews using tools (Gerrit, Review Board…) to enforce reviews and provide audit trail
  - CI/CD workflows – traceability and control from check in to deployment
  - TDD with test coverage thresholds – ensures that responsible testing is done
  - Artifacts are protected (signed and stored in repos)
  - Programmable configuration management (Chef, Puppet, Ansible) using same pipeline approach (check in, CI, CD, with scanning, testing, automated deployment)
  - All changes are done through CD pipeline: detective change control to alert on other changes
  - Metrics and “telemetry” part of reviews/requirements to provide feedback from production
Metrics and Feedback

• Use feedback to drive compliance and risk management – like other DevOps feedback
• DevOps teams are “metrics obsessed” (“measure everything”). Take advantage of this:
  – Automated test frequency and coverage
  – Scanning frequency and coverage
  – Vulnerability data from scanning/testing: #, severity, how long left open (in code, dependencies and runtime environment)
  – Defect density (security and logical/functional bugs)
  – Deployment success/failure – correlate with frequency/size of change, type of change
  – Mean time to detect/recover from failures and cycle time to deploy – window of vulnerability
• Measure cycle time to delivery and impact of compliance on cycle time
• Trend analysis and dashboards – use metrics to identify risks in teams/systems
• Dark spots (where you don’t have measurement) are risk areas
Understanding and overcoming objections from auditors, and creating a program that will make everyone (engineers, auditors, managers...) happy

OVERCOMING OBJECTIONS
# Concerns/Sticking Points

<table>
<thead>
<tr>
<th>Auditor Concerns</th>
<th>Addressing Them</th>
</tr>
</thead>
</table>
| Separation of Duties – unauthorized or untested changes, malicious insiders, fraud | Pipeline – not people – makes changes  
Auditing and detective change control  
Testing – coverage and “conditions of satisfaction”  
Optional approval step                                                                 |
| Continuous Deployment (self service deploys)                                     |                                                                                                                                                   |
| Change Control/Authorization                                                     | Product Owner represents management  
They approve all stories and acceptance  
Peer reviews/pairing for all code changes  
All changes are standardized, transparent                                                                 |
| Access Control                                                                  | Pipeline – not people – makes changes  
Read-Only access for developers  
Encryptions tokens, pseudonymization                                                                 |
| “You Build it, You Run it” means developers need production access              |                                                                                                                                                   |
## Concerns/Sticking Points

<table>
<thead>
<tr>
<th>Auditor Concerns</th>
<th>Addressing Them</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation and evidence</td>
<td>Upfront policies – defined, communicated</td>
</tr>
<tr>
<td>“Working Software over Documentation”</td>
<td>Tickets and tests – requirements, verification</td>
</tr>
<tr>
<td>Stories on sticky notes and card walls, models on white boards – just enough to</td>
<td>Configuration policies in code (and tests)</td>
</tr>
<tr>
<td>understand</td>
<td>Code repos and version control</td>
</tr>
<tr>
<td></td>
<td>Log everything – write once, archived</td>
</tr>
<tr>
<td>Standards and Best Practices</td>
<td>Teams need flexibility to choose tools, tests</td>
</tr>
<tr>
<td>Reference frameworks for auditors</td>
<td>But core controls/policies need to be consistent</td>
</tr>
<tr>
<td></td>
<td>Changes to rules, workflows approved by CAB</td>
</tr>
<tr>
<td></td>
<td>Tools that reference back to OWASP Top 10, CIS…</td>
</tr>
</tbody>
</table>
Keeping Auditors, Engineers AND Managers Happy

• Provide engineers with problems, give them some freedom (and accountability) to come up with solutions and to iterate
• Fit into how teams work (tools, steps) based on environment and risk – don’t dictate, adapt – no one right way, tools may vary between projects, as long as policy goals are met
• Involve auditors upfront and get their buy in – walk through examples/code (DevOps Kata)
• Focus on controls/checks that are simple, repeatable, easy to explain, automated
• Always capture evidence: tickets, logs, version history, chat, email read receipts...
• Continuous Delivery: changes are deployed through tools/process – not by a person (SoD)
• Audit/spot check regularly to make sure that controls are actually working
• Rely on detective and compensating controls (“Trust, but Verify”)
• Manual reviews/approvals are expensive: make them lightweight and make them count
• Some “compliance theater” may be required… but keep it to a minimum
Conclusion

- Translate boring/vague legalese into stories, “conditions of satisfaction” and “definition of done” – make them visible to everyone, problems that engineers can/must understand and solve
- Getting everything into code is a good thing! (different, but much more powerful)
- Build on CI/CD, build pipelines and test automation – not just about speed…. this puts focus and visibility where you want it: on your main path to production
- Do things that you should be doing anyway – and create evidence to show to your auditors
- Tracking work in tickets instead of post-its is a drag, but can be done in a Lean way
- Try to make compliance and risk management transparent to engineers
- Iterate, review and continuously improve – build on retrospection and feedback loops in Agile/Lean/DevOps for compliance and risk management
- People talking to each other and solving problems together and learning together is a good thing!
- DevOps teams must work in a consistent, disciplined and thoughtful way – but that’s the point of compliance in the first place
- It will take time to replace all paper and spreadsheets, and get people to understand and accept changes, but it’s worth it
For more information

- Dev-Sec.io hardening framework: http://dev-sec.io/
- SIMP Hardening Project: https://simp-project.com/
- Risk Management Theater/re: https://continuousdelivery.com/2013/08/risk-management-theatre/
- The Phoenix Project: https://itrevolution.com/book/the-phoenix-project/