DevOpsing Your Greenfield: Cultivating New Growth

Richard Mills
DevOps Solution Lead, Coveros Inc.
rich.mills@coveros.com
@armillz
Who is this guy?

• Me: Mad-Software-Developer turned Mad-Software-Engineer turned DevOps-Solution-Lead. Particular focus on tools and automation. CI, CD, DevOps ... what’s next?
  • PS: Thanks for inventing the term “DevOps” to describe what I like to do.

• Pays my bills: Coveros helps organizations accelerate the delivery of secure, reliable software using agile methods.
  • Agile transformations, development, and testing
  • DevOps implementations
  • Training courses in Agile, DevOps, Application Security

• Keeps me intrigued: SecureCI
  • Open-source DevOps product
  • Integrated CI stack with security flavor
Why is he here?

• Open your mind about most important aspects of "new" DevOps
• Share some of my experiences (and failures)
• Give you a reference to walk away with

• NOT: Explain fundamentals of DevOps (or Agile)
• NOT: Sell you on DevOps (or Agile)
Somewhere in our DevOps dreams...

I want to do some DevOps and Agile on our important new project!

Sigh

Great. How about this? Let’s BE Agile and adopt a DevOps approach to structuring our teams, designing our architecture, and leveraging automation to rapidly deliver value to our customers?

Right. That’s what I said.
Typical DevOps Transformation

• Frequently a fire-fighting effort
• Find the most important thing, improve it
• You have something concrete to start with, so you just start fixing it.
Greenfield DevOps

- Wide open options
- No obvious path
- Nothing to grab onto yet, so nothing to "fix" and no clear direction to follow
You have ONE job

• Don't blow it
• You have a clean slate
• Don’t create your own dumpster fire
General Agenda

- DevOps baseline – picking the seeds
- Greenfield approach – planting the seeds
- Important aspects for cultivating growth
  - Organization structure and mission
  - Infrastructure and tools
  - Pipeline design with delivery and branching model
  - Integrating with your architecture
  - Testing and quality analysis
DevOps – picking the seeds
The Many Things of DevOps

Notice that not many of these are PURELY the responsibility of one person or even team
Industry Views on DevOps

“DevOps (a portmanteau of "development" and "operations") is a software development method that stresses communication, collaboration and integration between software developers and Information Technology (IT) professionals.” – Wikipedia

The Three Ways - made popular by Gene Kim, et. al. in *The Phoenix Project*

- Rapid flow to deliver value (left to right)
- Rapid feedback (right to left)
- Continuous experimentation and learning

C.A.L.M.S. - developed by John Willis and Damon Edwards and added to by Jez Humble to describe DevOps (Culture, Automation, Lean, Metrics, Sharing)

*Point: DevOps is more than just automation and 20 deploys per day*
Ok, But What Does That Actually Mean?

- DevOps is the natural evolution of Agile: how to get working software into the hands of the people who need it rapidly and reliably
  - Who? Developers, Product Owners, ultimately users
  - When? Now!
    - Seriously … within weeks we can measure delivery with "minutes"
  - How? Team structure, processes, automation, tools

- Extension of Agile
  - Focus on value delivery
  - Common sense
  - Enable developers to be creative and do great work

- Work with developers to integrate system architecture with deployment architecture
  - Absolutely makes things easier and work better
  - Examples: metrics, testing, health, scaling
Greenfield approach – planting the seeds
So ... Where do we actually start?

It’s actually pretty easy:
1. Create a team
2. Set up some infrastructure and tools
3. Build a pipeline
4. Establish quality gates
5. Iterate

Ok, It’s a little harder than that...
Most Important: Iterate!

• Don't solve everything at once (breadth first)
• Don't even solve one thing all the way (depth first)
• Get the most important things functional, then improve over time
• Start left, build toward the right. Most important first.
• Don't paint yourself into a corner, but don't over-engineer
  • Wait until you know what you need for things that are easy to rework
  • Caveat: beware things that are immensely hard to re-work.

Start with the simplest thing that could work
Team structure and mission
Avoid DevOps Team Anti-Patterns

- Separate silos
- Dev, Ops, DevOps silos
- No Ops

http://web.devopstopologies.com/
Successful DevOps Organizational Structures

• Ideal: Cross functional teams with Dev, Ops & QA
  • Shared goals and values
  • Collaborative
  • Unified processes and tooling

• Sometimes: Embedded Ops
  • Suitable for single web-based product

• Maybe: DevOps as a service
  • Smaller teams/orgs
  • Transitional situations
Horizontal DevOps Guild

- Group of DevOps professionals working together to solve cross-team DevOps problems
- Guild members in-team are focused on team-specific problems
- Dedicated guild members support cross-team needs
- Guild establishes cross-team standards and shared success
- Important: share knowledge across team members

Cross-team function (vs. cross-functional team)
Example DevOps Guild and Team Structure

Our team organization on a 50-ish person development project

DevOps and Platform Core Horizontal Team
Finding the Right DevOps Members

• A good DevOps person ...
  • ... Is a Developer
  • ... Is a Tester
  • ... Knows about system administration
  • ... Has strong attention to detail
  • ... Has high standards for quality
  • ... Knows how to solve problems
  • ... Good at making things work

• Has experience and skills with build tools (maven, gradle, npm), test tools (junit, testNG, Selenium), database programming (sql, no-sql), CI build servers (Jenkins, Travis), operating systems (Linux, Windows), software installation/configuration (nginx, tomcat, databases), CM automation tools (chef, puppet, ansible), scripting (python, groovy, ruby), cloud systems (AWS, Google, Azure), virtualization and containerization (virtualbox, vmware, docker), and many, many other buzzwords (REST, HTTP, SSL, API, UI, e2e)
Infrastructure and Tools
### Tools, tools, and more tools

<table>
<thead>
<tr>
<th>Collaborate</th>
<th>Build</th>
<th>Test</th>
<th>Deploy</th>
<th>Run</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Lifecycle Mgmt.</strong></td>
<td><strong>SCM/VCS</strong></td>
<td><strong>Testing</strong></td>
<td><strong>Deployment</strong></td>
<td><strong>Cloud/IAAS/PaaS</strong></td>
</tr>
<tr>
<td>Atlassian JIRA</td>
<td>GitHub</td>
<td>Selenium Testing, JUnit</td>
<td>Jenkins</td>
<td>Amazon Web Services, Docker</td>
</tr>
<tr>
<td><strong>Communication &amp; ChatOps</strong></td>
<td><strong>Continuous Integration</strong></td>
<td><strong>Configuration Mgmt.</strong></td>
<td><strong>Orchestration &amp; Scheduling</strong></td>
<td><strong>Monitoring &amp; Logging</strong></td>
</tr>
<tr>
<td>Atlassian HipChat, Slack</td>
<td>Jenkins</td>
<td>Ansible, OpenShift</td>
<td>OpenShift</td>
<td>New Relic, Splunk, Sysdig, Grafana, Prometheus, Nessus</td>
</tr>
<tr>
<td><strong>Knowledge Sharing</strong></td>
<td><strong>Build</strong></td>
<td><strong>Analysis</strong></td>
<td><strong>Artifact Management</strong></td>
<td><strong>Tools, tools, and more tools</strong></td>
</tr>
<tr>
<td>Confluence, Swagger</td>
<td>Maven, gradle, docker, Node.js</td>
<td>SonarQube, jFrog, Fortify</td>
<td>jFrog Artifactory</td>
<td>New Relic, Splunk, Sysdig, Grafana, Prometheus, Nessus</td>
</tr>
<tr>
<td><strong>Database Management</strong></td>
<td><strong>Artifactory</strong></td>
<td><strong>Twistlock</strong></td>
<td><strong>OPENSHIFT</strong></td>
<td><strong>OPENSHIFT</strong></td>
</tr>
<tr>
<td>PostgreSQL, Liquibase, Cassandra</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© COPYRIGHT 2018 COVEROS, INC. ALL RIGHTS RESERVED.
Let’s start with LESS tools
Infrastructure Platforms

• Infrastructure: Operating Platform
  • Server platform - physical, VM, container
  • OS configuration
  • Middleware installation and configuration (java, nginx, tomcat, database, message queue, etc.)

• DevOps Infrastructure: DevOps Platform
  • Build server (Jenkins)
  • Quality analysis server (SonarQube)
  • Artifact server (Nexus)
  • Source control server (Git)
  • Test server(s)? (Selenium)
  • Target environment for CI/CD builds (see: Operating Platform)

• Developer Infrastructure: local development
Operating Platform

• VMs – mainstream and the current “normal”
  • Choose your OS: Linux, Windows
  • Orchestration and CM tools: Chef, Ansible, Puppet, ...
  • Pipeline produces installable software
  • CM tool install/configure software

• Containers (e.g., Docker) are the new hip
  • Choose a smaller OS (RHEL, Alpine, Busybox)
  • Kubernetes – “which version” (e.g., Kube vs. OpenShift vs. ...)
  • Pipeline produces container images
  • Orchestration tool (Kube) deploys and configures

• To the cloud!
  • Amazon, Google, Azure ...
  • Infrastructure as a Service vs. Platform as a Service vs. completely serverless
  • Networking, Virtual Machines, Docker containers
  • This is likely to form the basis of your VM’s or containers
Setting up your Platform

• Start small/easy – get something running for developers to work in
• Automate from the bottom up ← most frequent == most valuable
• Move towards dynamic environments – easily launch new ones
DevOps Platform

- DevOps Platform - the set of tools and environments you use
- Start with your operating platform, build from there

- Jenkins - build and pipeline orchestration tool
- SonarQube - quality analysis dashboard
- OpenShift & Docker - deployment and configuration

- Pick some assessment tools:
  - JUnit, Jest
  - OWASP Dependency Check
  - Selenium

- Again, start manually then move to automation
Example Platform Architecture

- Ideally, your “operating platform” and “DevOps platform” share a lot of underlying infrastructure.
Align the Developers

- Developer Environments - align with operational platform

- Should be able to fetch code build with very little effort; well documented, highly automated

- Local deploy/test cycle should be similar to operational (e.g., VM, docker, Operating system, etc.)

- Caveat: support rapid change/build/test cycles supported by modern IDEs
Test Environments

• Why do you have them? What purpose do they serve?
• Static vs. dynamic environments
• A/B deployment patterns relate to this
Constructing the Delivery Pipeline
Stages of Delivery

Incremental quality gates as code makes its way through the delivery pipeline
- Continuous Integration produces deployable software that is scanned and tested in isolation
- Continuous Delivery produces running software that is validated for functionality, security, performance to enable “promotion” to higher environments
- Continuous Monitoring ensures continuous secure/reliable behavior and adapts/reCOVERs to anomalous behavior
Pipeline and Tools

Use automation and tools to continuously assess quality, performance, and security in rapid feedback loops.
Configuration Management and Version Control

“Version control everything”

• Application code
  • No brainer, but need branching and build pattern

• DevOps automation code
  • This is software. How will you build/test/release it? Branching?
  • Maturity model: start simple, eventually need a "pipeline for your pipeline"

• Infrastructure as code
  • Provisioning – scripted creation of infrastructure
  • Configuration – servers, network, storage

Your CM process will define your delivery process
Branching Strategy

- Strive for "main line" development
  - Use short, small feature branches for isolated changes
  - Consider Github Flow (very simple), Git Flow (complex)
  - Avoid "parallel release development" at all costs
- What problems does branching cause vs. solve?
- Align with your delivery pipeline
Integrating with Software Architecture

• Not all software lends itself to easy build, test, and deployment
• Work directly with software architecture and development teams
• Software must support
  • Rapid build
  • Automated test (controllability, observability)
  • Data initialization
  • Installation/configuration
  • Monitoring/metrics
• Standards and "Definition of Done" should reflect this
• Development stories aren’t complete until all these things work
Testing and quality analysis
Your efforts in DevOps will fail without proper automated testing and assessment that is fully integrated into the pipeline.
Goals for Testing in DevOps

• Keep software in continuous working state
• Establish confidence in change
• Force teams to build quality in
• Avoid creating too much technical debt too early

Important: everywhere I say “Quality” I mean “Quality, Security, Performance, and all the other –ilities” you can think of for your software.
Put Some Effort into a Test Strategy

- Determine what you want to do, who's going to do it, how it's going to tie into the pipeline
  - Developers
  - Engineers in Test
  - Business Analysts
- Establish standards for quality, security, performance, etc.
- Define the skills you will need on the team to succeed
- Heavy focus on automation
  - CAN'T succeed without automation
  - You cannot automate EVERYTHING

- Developer tests vs. "other" person tests
  - Who? Consider roles and who can do it.
  - Agile Test Engineer - automation, frameworks, close alignment with DevOps
  - BA/Scrum Master - test definition, creation, exploratory testing
  - Developers - technical tests
Quality Gates are Critical

*Quality gates ensure that bad code cannot make it through the pipeline*

- **Static analysis** - set gates early; avoid tech debt
  - Standards: style, bugs, vulnerabilities (standards not as important as following them)
  - 3rd party dependency analysis - vulnerabilities, licensing
- **Testing phases**
  - Pre-deployment testing - unit, component, mocking
  - Post-deployment functional testing - services, integration, databases, etc.
  - Non-functional testing - security, performance, reliability, availability
- **Code Review**
  - Tie to branching process (Github pull requests)
  - Small feature branches help this
    - *Start left, move right (quickly)*

---

© COPYRIGHT 2018 COVEROS, INC. ALL RIGHTS RESERVED.
Getting Started with DevOps Testing

Create ONE of each test and make sure it gets executed by the DevOps pipeline continuously

- Cover each test level
  - Unit
  - UI
  - API
  - Security
- Once the test framework is in place, it removes the barrier and enables test writing
- Pitfall: initially, your test results will be all over the place. You will realize the need for a “quality dashboard”
Greenfield DevOps: Grow!
Start Small, Make It Work

- Launch basic DevOps tools (Jenkins, Sonar, Nexus, Git)
- Setup basic CI builds
  - Developers use same gradle/npm build scripts as the pipeline
- Create some quality gates
  - SonarQube static analysis
  - Unit testing
- Create a test environment (by hand, if need be)
- Initialize deployment automation
  - Docker deploy, Ansible installation
  - Developers use same (similar) local deployment

*All this can be done in days or weeks with the right people*
Iterate to the next level

- Establish a branching process for the developers with code-review on merge
- Set up multi-branch builds in the pipeline
- Measure code coverage with your unit tests (establish metrics and standards)
- Automate installation of your operating platform (dynamic environments)
- Deploy for after every build
- Execute UI or API tests in the pipeline
Don’t forget to build a solid team!

- Build a strong integrated team of Dev, Ops, and QA
- Strong problem solvers
- ”Can do” attitude
- Don’t put barriers in front of them
Generating Value

Remember, the main goal of DevOps is to deliver working code to the people who need it
• Prioritize what the developers need first
• Get software in front of stakeholders ASAP
Keep it real
• Don’t get hung up on making things perfect at first
• Establish quality gates to **avoid technical debt**
Questions?

Thank you!
rich.mills@coveros.com
https://www.coveros.com/services/devops/