Stay C.A.L.M.S.
A local company’s journey to DevOps

Sander van Zoest
@svanzoest
What is DevOps?

It is a human and management problem

- Development (Software Engineering)
- Quality Assurance (QA)
- Technology Operations

DevOps
DevOps is not a new name for System Administration

It is about collaboration between functional teams.
Enter C.A.L.M.S.

- Culture
- Automation
- Lean
- Measure
- Sharing
**Company Culture**

**Industrial**
- Functional Silos
- Blame runs downhill in public, and uphill at the water-cooler
- Avoid Risk and Failure
- Never revise policy on success (Retrospective)
- Heteronomy

**Agile**
- Cross-functional delivery teams
- Team Ownership (Trust)
- Fail Fast, Learning Opportunity
- Continuous Process Review
- Autonomy and Alignment
Culture: Breaking the Silos

- Seating Arrangements
  - Cubicles vs Open Spaces
  - Headphones
  - Sales versus Development tasks

- Unbiased Project Manager (reports to Product or Engineering?)

- Matrix Teams

- Spotify’s Squads
Culture: Spotify Squads

Henrik Kniberg & Anders Ivarsson
Oct 2012
Squads own a minimum viable product

How do split into squads?

Horizontal
  Based on Product Features

Vertical
  Based on Product Layers

Is Customer Service a Squad?

Is Account Management?

Is Operations?
Fear of Failure

Never ending 2 week sprints
  Intimidating teammates
  Constant context switching
Perceived failure in lack of progress
Lack of personal safety in admitting failure of a task
  Limited Long term vision
  Low Self-Esteem
  Perfectionism
Understand you can never know everything.

Account for unknown unknowns

Focus on unknowns first

Fail fast

Foster creativity and innovation

Learn

Create a safety net
Project Manager: "What is the status of project X? Did you ever complete that?"

Software Engineer: "Oh, that's done."

Project Manager: "Great, so it is live in production then?"

Software Engineer: "Ehhh...no, it still needs to be tested, integrated and deployed. I am just dev done”.

Rodeo Done
It isn't rodeo done, until it has passed acceptance testing in production.
Automation

- Infrastructure as Code
- Creates Consistency
- Avoids Repetitive (boring) Tasks
- Makes room for improvement and learning
- Removes Ambiguity (Blame Culture)
- Safety Net to try new things
Service Life Cycle
First determine what your product life cycle is.

We started with deploying the dev environment.
  Make it as close to production as possible
    Avoid separate ways of doing things.
    Create a single flow to production.

Goal
  Fully automated deployment to production
  Get it down to a single button press
Dev Environment Setup: Goal

- How long does it take for a new hire to setup their environment? How long until they commit to production?
  - In 2010 it took 2 months setup the env and update docs.
  - In 2014 they would push to production within 48 hours.

- What should be other goals? How would you measure them?
Dev Environment Setup: How

- Leverage Virtualization: We used Vagrant with Virtualbox
- It allowed us to DRY up use the same tools on dev as production.
- Don’t worry about what tools, focus on repeatability. Shell scripts? Doesn’t matter. As long as it is reliable and repeatable.
- Once it is, measure, review and iterate
Dev Environment Setup: Shell Scripts

- Now that it is repeatable, how would you apply it to production?
- Do any of the packages and/or configs change?
- Did you check them in version control?
- Next convert them into a more flexible tool.
- We chose Chef
Release Cycles
Release Frequently

Change

Time

Lots of lines of code

A long time
Goal: Reduce Risk of Release

feedback from users
reduce risk of release

Theory of Constraints

- Identify the system's constraint
  - Integration Testing and Release Management
  - Development completed. Needs to be tested (waterfall)

- Decide how to exploit the system's constraint
  - Build tests at the same time as developing the code (alignment/focus)
  - Minimize changes tested at once. Minimize backlog of releases.

- Subordinate everything else to the above decision
  - Align the whole system or organization to support the decision made above

- Elevate the system's constraint
  - Measure, rinse and repeat
<table>
<thead>
<tr>
<th>Release</th>
<th>Date</th>
<th>US</th>
<th>DE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6</td>
<td>20-Dec</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>5.7</td>
<td>4-Jan</td>
<td>8</td>
<td>49</td>
</tr>
<tr>
<td>5.8</td>
<td>9-Jan</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>5.9</td>
<td>17-Jan</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>5.1</td>
<td>25-Jan</td>
<td>3</td>
<td>26</td>
</tr>
<tr>
<td>5.11</td>
<td>7-Feb</td>
<td>4</td>
<td>21</td>
</tr>
<tr>
<td>5.12</td>
<td>14-Feb</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>5.13</td>
<td>18-Mar</td>
<td>5</td>
<td>29</td>
</tr>
<tr>
<td>6</td>
<td>15-Apr</td>
<td>19</td>
<td>169</td>
</tr>
</tbody>
</table>

**User Story/Defect**

- **20-Dec**
- **20-Jan**
- **20-Feb**
- **20-Mar**
What is continuous delivery?

- Making sure your software is always production ready throughout its entire lifecycle
- reduce the cost, time, and risk of delivering incremental changes to users
- Everybody is responsible for delivery
Continuous Delivery Process Flow

- Delivery team
- Version control
- Build & unit tests
- Automated acceptance tests
- User acceptance tests
- Release
Our Implementation
Findings & Learnings

- Quicker turn around from Concept to Deployment
- Easier to quickly fix issues in production
- Focus on Roll forward, Not backwards
- More focus on automated tests
- More focus on process improvement
- Quality focus rather than individual throughput focus
### The Continuous Delivery Maturity Model

<table>
<thead>
<tr>
<th><strong>Base</strong></th>
<th><strong>Beginner</strong></th>
<th><strong>Intermediate</strong></th>
<th><strong>Advanced</strong></th>
<th><strong>Expert</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Culture &amp; Organization</strong></td>
<td><strong>Design &amp; Architecture</strong></td>
<td><strong>Build &amp; Deploy</strong></td>
<td><strong>Test &amp; Verification</strong></td>
<td><strong>Information &amp; Reporting</strong></td>
</tr>
<tr>
<td>- Prioritized work</td>
<td>- Organize system into modules</td>
<td>- Vue aimed code base</td>
<td>- Automatic unit tests</td>
<td>- Baseline process metrics</td>
</tr>
<tr>
<td>- Defined and documented process</td>
<td>- API management</td>
<td>- Scripted builds</td>
<td>- Separate test environment</td>
<td>- Manual reporting</td>
</tr>
<tr>
<td>- Frequent commits</td>
<td>- Library management</td>
<td>- Basic scheduled builds (CI)</td>
<td>- Automatic integration tests</td>
<td>- Measure the process</td>
</tr>
<tr>
<td>- One backlog per team</td>
<td>- Version control DB changes</td>
<td>- Dedicated build server</td>
<td>- Automatic component tests (isolated)</td>
<td>- Static code analysis</td>
</tr>
<tr>
<td>- Share the pain</td>
<td>- No (or minimal) branching</td>
<td>- Documented manual deploy</td>
<td>- Some automatic acceptance tests</td>
<td>- Scheduled quality reports</td>
</tr>
<tr>
<td>- Stable teams</td>
<td>- Branch by abstraction</td>
<td>- First step towards standardized deploys</td>
<td>- Common information model</td>
<td>- Measure the process</td>
</tr>
<tr>
<td>- Adopt basic Agile methods</td>
<td>- Configuration as code</td>
<td>- Auto triggered build (commit hooks)</td>
<td>- Traceability built into pipeline</td>
<td>- Static code analysis</td>
</tr>
<tr>
<td>- Remove boundary dev &amp; test</td>
<td>- Feature hiding</td>
<td>- Automated tag &amp; versioning</td>
<td>- Report history is available</td>
<td>- Scheduled quality reports</td>
</tr>
<tr>
<td>- Extended team collaboration</td>
<td>- Making components out of modules</td>
<td>- Build once deploy anywhere</td>
<td>- Graph as a service</td>
<td>- Dynamic graphing and dashboards</td>
</tr>
<tr>
<td>- Component ownership</td>
<td></td>
<td>- Automated build of DB changes</td>
<td>- Dynamic test coverage analysis</td>
<td>- Cross silo analysis</td>
</tr>
<tr>
<td>- Act on metrics</td>
<td></td>
<td>- Basic pipeline with deploy to prod</td>
<td>- Automatic performance tests</td>
<td>- Report trend analysis</td>
</tr>
<tr>
<td>- Remove boundary dev &amp; ops</td>
<td></td>
<td>- Scripted config changes (e.g. app server)</td>
<td>- Automatic security tests</td>
<td>- Cross functional teams</td>
</tr>
<tr>
<td>- Common process for all changes</td>
<td></td>
<td>- Standard process for all environments</td>
<td>- Risk based manual testing</td>
<td>- No rollbacks (always roll forward)</td>
</tr>
<tr>
<td>- Decentralize decisions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Dedicated tools team</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Team responsible all the way to prod</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Deploy disconnected from Release</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Continuous improvement (Kaizen)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Cross functional teams</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- No rollbacks (always roll forward)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Legend:**
- **Base:** Basic understanding and implementation of Continuous Delivery practices.
- **Beginner:** Practices are applied in a controlled manner, focusing on automation and efficiency.
- **Intermediate:** Practices are widely implemented, with a focus on integration and collaboration.
- **Advanced:** Practices are fully integrated into the organization, with a focus on innovation and continuous improvement.
- **Expert:** Practices are beyond implementation and focus on continuous learning and improvement.
Continuous Delivery Patterns

- Continuous Integration
  aka Develop on Mainline
- Feature Toggles
- Branch by Abstraction
- Dark Launching
- Database Versioning
- Database Backwards Compatibility
- Production Immune System
- Blue-Green Deployments
- Canary Releasing
- Expand/Contract
- A/B Testing
Pattern: Feature Flags / Dark Launching

- Allows code to be released but not visible to everyone
- Works in parallel with A/B Testing
- Allows for feature iteration by product owner in while using it in production
- Allows Customer Service and Account Management to get familiar with feature and communicate to customers at their own pace
Pattern: Develop on Mainline

- We explored many branching models
- Git Flow
- Pull Request
- Etc.

We currently standardize on Pull Requests to allow for peer code review.
Pattern: System Composition

Location (Data Center)
- for gateway, NTP, APT Mirror, etc.

Cluster
- for the smallest logical horizontal scalable node
- E.g. Front End, Data Store

Environment
- Dev, QA, CI, Staging, Sandbox, Production

Realm
- Data type or source: Identity, Content, BI
Pattern: System Composition

Translates to monitoring, graphing, reporting, alerting
Pattern: Vendor during Development

- Starts by creating a vendor directory in your project
- You ultimately only need to vendor your dependencies in your build artifacts
- We use:
  - Composer/Satis for PHP
  - Bundler for Ruby
  - Berkshelf for Chef Cookbooks
  - NPM for Node/JavaScript
Pattern: Build Artifacts for Deployment

Version Control -> Build Process -> Artifacts
Pattern: Control your Environment

- Setup your own OS Package Repository
- Setup controlled mirrors of all software you depend upon in production
- We used *apropos and freight* for our debian packages
Remember team ownership

Collaboratively determine the true problem (root cause)

Pick tools useful by everyone

Communicate Reasoning

Eliminate waste

Amplify learning

Decide as late as possible

Deliver as fast as possible

Empower the team

Build integrity in

See the whole
Measure

- Build Confidence
- Provides Context
- Incidents
  - Frequency
  - Severity
  - Root Cause
  - Time-To-Detect (TTD)
  - Time-To-Resolve (TTR)
  - Coordinate Responses

- Measure Progress
- Track Change
- Value Stream Mapping
- Graph It
- Alert on it
Take a chapter from the "Infrastructure as Code" movement and appreciate the benefits that come from interoperable pieces that are inexpensive, scalable and easily automated with a little code and a stable API. -- Jason Dixon (obfuscurity)
Monitoring: Tracking

- Application Data:
  - Statsd
  - Application Logs and Events

- Deployment Data
  - Jenkins
  - Chef Report Handler

- System Data
  - Sensu

- etc
Monitoring: Graphing

- Graphite
- ELK Stack
  - ElasticSearch
  - Logstash
  - Kibana

The key thing is to build systems that can be easily maintained and accept new arbitrary data, so you can focus on the data and presentation.

Not the creation of dashboards.
Does two things:
- Store numeric time-series data
- Render graphics of this data on demand

Consists of three things:
- Receiver Daemon - Carbon
- Disk Storage Library - Whisper
- Web Application – Graphite / Graphana

Used for Application and System metrics
Monitoring: What is an ELK Stack?

- ElasticSearch
- Logstash
- Kibana

We feed it via rsyslog and allows querying of the logs and alerting on log data.
Monitoring: Alerting

- Tried Nagios, Opsview, etc.
- Landed at Sensu
  - Event Messaging Bus
  - Maps monitoring to Chef cookbooks
  - Automatic Client detection and decommission
  - Easy integration with other software
    - Graphite
    - Email
    - Slack (ChatOps)
Good News and Bad News

Creates Trust and Confidence

Helps collaboration

Within your team

Between departments

Between companies
Share: Real-Time Communication

- Dashboards
- Hallway Monitors
- Via Email
- Over the Cube
- ChatOps

- Focus on information sharing
- System Integrations
- Continuously tune to improve signal to noise ratio.
Share: Community

- Open Source
  - Bug Reports
  - Code Patches
  - Documentation

- Contribute back to the community
- Open Source re-usable generic tools and methods
- Write a blog
- Join mailing lists, etc.
- Join San Diego DevOps
  - SDDevOps.org
San Diego DevOps Speakers & Members
Every Journey is different
You need to determine what is right for you
Learn, Implement, Measure, Iterate and Share