Using DevOps to Achieve Better Application Development and Security in Modern Cloud Environments

Sean Bird & Michael Corley
Quanterion Solutions, Inc.
NYS Cyber Security Conference
June 8, 2017
Agenda

• Background: What we do
• DevOps Foundational Ideas
• DevOps and the Cloud
• DevOps Overview
  • Principles
    • Collaboration, IaC, Automation, Monitoring
  • Security
  • Examples
  • Benefits
  • Risks/Challenges
• Tools
• Summary
• Demonstration / Environment Overview
• Questions
Quanterion Solution Incorporated

• Operate the CSIAC (Cyber Security Information Analysis Center)
  • CSIAC - STI gathering and dissemination in technical competency areas encompassing cyber security, software engineering, and modeling & sym.
    • Visit CSIAC booth for details

• We (as individuals) support R&D (DoD) contracts and product/tool evaluation initiatives relating to Information Management/Assurance, Cyber Security, and High Performance Computing (HPC)
  • Experiment with relevant new/evolving technologies
    1) Leverage new capabilities into our development/research culture, for internal organic growth and the growth of our customers.
    2) A resource for good counsel (our customers, peers and the community)
DevOps Foundational Ideas

• DevOps is not an entirely new idea...
  • Grew out of the disconnects that naturally arise between (people) disparate groups within an organization...specifically “Development” and “Operations” (DevOps)
    • Miscommunication, misaligned goals and objectives manifest as conflict and inefficiency.
      • Problems compound to become increasing complex as the size of an organization increases (as development teams increase in size)

• DevOps evolved out of approaches to overcome these problems by focusing on the communication, collaboration, and integration between software development and information technology personnel [1]
  • However, DevOps affects all people in the path in the product development lifecycle
DevOps and Cloud?

• What is the Cloud?
  • (Simply put), the provisioning of (and access to) computing resources through well defined service models (e.g. IaaS (CaaS), PaaS, SaaS, etc.)
  • Industry continues to move toward architectures that are service/cloud oriented.
    • Why?
      • Measurability, controlled/shared access, efficient/cost effective, highly flexible
        • Public, Private, and hybrid solutions
    • An interesting way to layer ((re)factor with) cutting edge technologies
      • A rich API/ecosystem
        • Virtual machines, micro services (containers), various orchestration models etc.
  • Cloud API/ecosystem (orchestration services), and natural fit for facilitation of DevOps processes
    • Public Cloud providers support native DevOps enabled services
DevOps Terms and Concepts Overview

- An extension of the Agile methodology...
  - An Iterative cycle (like Agile) but differs in that it seeks to highlight the breakdowns and bottlenecks (*the Silos*) that impede the development process [1]
  - Collaboration and Communication is a key focus
    - Example of a Silo: Developer depends on the Operations staff for compute resources (development/production environment etc.)
      - Operations staff may not respond punctually (limited staff/authorization process)
      - Developer might seek to circumvent policy in favor of maintaining productivity (“I need to get my job done!!”)
      - Might not have knowledge of all important details of the environment
        - i.e. policies that detail security constraints.
          - firewall rules, open ports, acceptable communication protocols, etc.
    - Consequences bring about a variety of problems: increased complexity, process inefficiencies associated with integration and deployment processes, and increased risk security vulnerabilities.
DevOps extends Agile

• Agile embraces the notion of *constant change* [1]
  • Regular customer interaction for requirements and domain expertise
    • ensuring value etc.

• DevOps embraces the notion of constant testing and delivery [1]
  • Regular participation/interaction from the full breadth of expertise (development, operational (security), legal, business (financial), etc.

• No single method fits all, but emphasize an “all hands” organizational movement
  • An iterative process but with intent to eliminate the solos (i.e. the bottle necks) that inhibit collaboration.
    • Get disparate groups to *iterate* together
Goal: Break down the silos!

- Silos inhibit collaboration and reinforce the waterfall methodology
  - A cultural movement in as much as it is a development cycle [1]
    - “…that’s not my Job!” “That’s is my Job, stay out!…” *Ownership ideology*
  
- Everyone becomes a first class citizen and works collaboratively across the disparate areas: Requirements->Design->Implementation->verification>maintenance
  - e.g. Operations (security staff) should be involved in requirements and design phases else domain expertise is lost!

---

DevOps Core Principles [1]

• Collaboration
  • Efficiency (communication and processes) between disparate roles, both within teams between teams in an organizations

• Infrastructure as Code (IaC)
  • Script and version control as much of infrastructure as possible, principally for capturing the disparate environment details (e.g. development, testing, production, etc.)
    • Goal: (repeatable, recoverable, reusable, sharable)

• Automation
  • Automate human (error processes) wherever possible
    • Goal: Increase frequency of testing -> continuous (consistent/uniform) integration, delivery, and deployment processes

• Monitoring
  • Data to needed to inform development/operational decisions
    • policies and priorities are domain dependent
      • Is the System working? Performance acceptable? Are users consuming the system as expected? Are we being attacked?
Principles as means to inform Security

• DevOps process promotes Security to first class citizen
  • *Collaboration, IaC, Automation, and Monitoring*: all utilize and inform traditional principles relating to Security
  • Collaboration
    • All expertise is involved/informed in design, decision and development processes
    • Environment/ network configuration, code reviews, deployment planning etc.

Infrastructure as Code
• Endeavors to ensure to the environment is correct (repeatability, reproducibility, reusability)

• Automation
  • Remove error prone processes where possible (eliminate drudgery)

• Monitoring
  • Data collection and analysis aimed at verification, and reasoning need to make informed decisions
More about Security

• The rugged manifesto (a culture for secure building software)
  • proactive versus reactive reasoning (*DevSecOps, RuggedDevOps, etc.*)
• Platform Security – the DevOps pipeline (code, infrastructure, automation CI/CD)
• Application Security – SLDC: -> Inception, code/test, commit, (review), CI, QA/CD, transition.
  • Every transition point is security concern
    • Security first focus (shift security concerns leftward in the pipeline)
    • SDLC -> Design | Build | Deploy | Operation
      • Opportunities to address Security at every phase
        • Design: How to secure? (threat modeling etc.?)
        • Build: Components? (vetted repository)
        • Deploy: Secrets: passwords private keys
        • Operation: Attacked?
          • Need appropriate monitoring
More About Security (cont.)

• Not everything can or should automated (CI/CD)
  • Regimented source code management
    • Code pushes trigger security checks
      • Take every opportunity (goal is process parity)
      • Manual, automated analysis
    • Vetted 3rd parity components, versions, CVE’s etc. (80-20 rule)
      • OpenSSL -> Heartbleed bug etc.

What is Infrastructure as Code (IaC) ?

• First, what’s Infrastructure?
  • Everything the code touches: production servers, networks, developer workstations, repositories, test servers, runtime environments, etc.

• .. and IaC?
  • Program/(formally scripted) process that offloads the creation of all of infrastructure characteristics: (server machines, networks, databases,
    • Lives in version control and maintained throughout the development cycle expected to reliable, consistent, and repeatable
    • Formally defined with contributions by all individuals in the deployment path (development, IT (security), management, and stake holders
      • Everybody has visibility
    • Applicable to all of the environments: development, testing, staging, production
      • e.g. copy of the production database in staging/development (but with sanitized data)
Example (Scenario)

- Most systems in deployments are distributed architectures composed of multiple disparate servers networked together by variety of infrastructure include network protocol(s), middleware, etc.
  - Often implemented manually (perhaps via scripts) using virtual machines in oriented/cloud-centric model
    - Slow, time consuming, tedious (error prone)
    - Multiple environments: Development, Test, Staging (QA), production
      - e.g. code doesn’t behave the same in production environment as does in development/test environments: (tests fail in testing but pass on developer workstation etc.)
      - QA discovers a bug, but can’t reproduce in developer environment, and can’t roll back production environment:
        - Changes: patches, software versions, platform and network differences etc.
Benefits of IaC

- **Operational**
  - Promotes collaboration
    - Expertise from all teams necessary to establish scripted definition
    - Formally captures the infrastructure, configuration, and policy
    - Visible to everyone, not limited to person ownership
    - Using a standard tool set promotes consistency in terminology, and therefore productivity to increases efficiency.
    - Different incentives, but shared resource allows expertise to converge.

- **Security**
  - Patch coverage and propagation
  - Stable roll back (version controlled)
  - Fast onboarding (error prone / manual provisioning)
  - Parity across various environments
Risks/Concerns?

• Lots happening without human intervention
  • What happens when things go wrong?
    • Do you have the visibility to reason to solve problems?
    • Can you manage/administer the system effectively?
    • Can you make the guarantees required by the domain?
      • Gets back to Monitoring

• e.g. implementing a system micro services (i.e. container/unikernel)
  • Orchestration is the basis for management...
  • 1 level – the cloud service provider (.e.g. openstack - Magnum, AWS)
  • 2 level – container management provider (.e.g. Kubernetes, Docker Swarm)
  • 3 level – configuration management (platform, users, services, runtime environments)
    • When things go wrong how do we “reason” effectively across the boundaries”
Tools that enable IaC

• Shell scripts
  • Platform-specific commands

• Vagrant
  • Creating and managing virtual machines (VirtualBox, VMware, Openstack, etc)
  • Works seamlessly with Chef and Puppet in the local environment

• Chef
  • “Recipes” based approach to describe sources that comprise server/application configuration
  • Integrates well with Cloud platforms including Openstack (Heat), Amazon EC2, Microsoft Azure, Google Cloud, and tools such as Vagrant

• Puppet
  • describes system resources and their state, either using Puppet's declarative language or a Ruby DSL. Managed as Pupoeet manifests.
Tools that enable IaC (cont.)

- Docker
  - Deployable Linux containers (whole environments easily shared)
- Ansible
  - Automates software provisioning, configuration management, and application deployment.
- Fabric
  - Python library used for SSH based remote systems administration tasks, and application deployment
- Heat
  - Openstack orchestration services: uses text file templates (YAML, JSON) to describe the infrastructure for Cloud applications
    - seeks compatibility with Amazon AWS CloudFormation
    - Integrates well with (configuration management tools) including chef, puppet
Automation

• Integration: merging disparate system artifacts is challenging
  • Often done manually – (slow and error prone) == complex and expensive
  • Can’t automate everything, but ... big changes have security implications that require review etc.)
    • Continuous Integration (automate what makes sense...)
      • Automate/standardize submission process (get artifacts) into the various environments (testing, staging...)
        • Automate builds, testing, deployment (to QA), message changes
      • Greater visibility: everybody gets notified of issue(s) sooner: developers, testers (QA), Op (Security), Management, etc.
        • fail fast and fix fast == increase efficiency.
        • goal is partitioning complexity
• Continuous Delivery
  • Changes pushed often to customer/mock production environment (perhaps staging is configured as such)
  • Give everybody (customer/stake holders) visibility
• Continuous Deployment
  • Changes are pushed continuously (often) into production
    • Domain specific
      • doesn’t mean this makes sense to do
      • does mean the process for deployment is automated, consistent, reliable, and repeatabe.
• Goal is Process Parity to achieve frequent/regular testing Make small, incremental changes!
  • is on achieving regular testing
Techniques to support Automation

• Feature flag pattern: keep features in the production, but dormant
  • Helps continually push into production
  • “dark launch”

• Canary Release
  • Deploy to subset of users (regionally etc.)

• Definitely risky, but cloud have business value
Tools that enable Automation

• **Jenkins**
  • Open source (Java based) automation server. Plugin based approach to support building and deploying code (CI/CD): [https://jenkins.io/](https://jenkins.io/)

• **TeamCity**
  • Commercial (Java based) CI automation server

• **AWS tools**
  • Code Build – automated build server (compilation, run tests, packaging)
  • Code Commit – secure repository service
  • Code Deploy – automated code deployment
  • Code Pipeline – CI/CD service
  • [https://aws.amazon.com/](https://aws.amazon.com/)
  • [https://aws.amazon.com/free/](https://aws.amazon.com/free/)

• **Visual Studio Team Services**
Tools that Support Monitoring

- **Nagios Core** is open source monitoring software for systems, networks and infrastructure.
- **Zabbix** is enterprise open source monitoring software for networks and applications.
- **IDS** (SNORT, etc.)
- **ELK** stack, Splunc (log analysis/metrics/analytics)

Tools – The Legos of the Cloud

<table>
<thead>
<tr>
<th>Strata of the Container Ecosystem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Layer 7:</strong> Workflow</td>
</tr>
<tr>
<td>OPENSHIFT</td>
</tr>
<tr>
<td><strong>Layer 6:</strong> Orchestration</td>
</tr>
<tr>
<td>kubernetes</td>
</tr>
<tr>
<td><strong>Layer 5:</strong> Scheduling</td>
</tr>
<tr>
<td>MESOS</td>
</tr>
<tr>
<td><strong>Layer 4:</strong> Container Engine</td>
</tr>
<tr>
<td>docker</td>
</tr>
<tr>
<td><strong>Layer 3:</strong> Operating System</td>
</tr>
<tr>
<td>ubuntu</td>
</tr>
<tr>
<td><strong>Layer 2:</strong> Virtual Infrastructure</td>
</tr>
<tr>
<td>VMware</td>
</tr>
<tr>
<td><strong>Layer 1:</strong> Physical Infrastructure</td>
</tr>
<tr>
<td>Raw compute</td>
</tr>
</tbody>
</table>

Mike Metral https://getcarina.com/docs/best-practices/container-technologies-orchestration-clusters/
**Micro services (container) Orchestration**

**Main goal: provide automated container management as well as guarantees for multi-container services and container engines.**

- Marathon with Apache Mesosphere(Scheduler) and DC/OS
  - Multiple physical node scaling (treated as one machine)
- Kubernetes (Google Inc.)
  - Large scale service oriented design
  - “Self-Healing” services and Load balancing
- Swarm (Docker)
  - Standard basic clustering tool
  - Native Docker API for third party tool integration

Google Docs: An Example

Bob

Alice

Kubernetes Master/Proxy

Kubernetes Agent

Kubernetes Node

Kubernetes Node

Persistent Storage Virtual Machine

Shared Hardware

= Container

= Service Pod
Unikernels versus Containers

- Specialized, single-address-space (single purpose) machine images constructed by using library operating systems
  - Compile high-level language code directly into a specialized machine image
    - Advantages:
      - No general purpose OS, specialized code/image needed to run the application
      - Reduced code size and resource footprint
      - Performance optimization, fast (millisecond) boot times
      - Shrink the attack surface, and better security
        - Transient micro-services
      - Run directly on a hypervisor, container, or bare-metal hardware
    - Open source work on Unikernels:
      - Rumprun, Clive, HaLVM, MirageOS, IncludeOS

http://unikernel.org/
Basic environment: Openstack

- **What is Openstack?**
  - (Private) Cloud (IaaS) solution
    - Keystone (identity) management service (controller)
    - Horizon (Dashboard) service
    - Nova (Compute – hypervisor) service
    - Neuron (Networking) (virtual networking) service
    - Cinder (Block storage – LVM partition management)
    - Glance (Image) service (1 node)
    - Swift (Object storage) service
    - Heat (Application orchestration)
    - Magnum (Container Orchestration – enables Docker swarm and Kubernetes)
- **Why?**
  - Open source, and rich user community
    - More that 500 companies as 2016 [7]
Openstack Deployment on VMware

Hardware: Dell Precision m6500 (Intel i7-CPU), 32g memory

Windows 10 Professional (Host Operating System)

VMWARE Workstation pro (v12)

Your Applications

OpenStack Dashboard

Compute

Networking

Storage

OpenStack Shared Services

APIs
Openstack on IBM Blade Center
Demonstration: HEAT (IaC): Wordpress example

```yaml
wordpress_server:
  type: OS::Nova::Server
  properties:
    image: { get_param: image_id }
    flavor: { get_param: instance_type }
    key_name: { get_param: key_name }
  networks:
    - network: External
  security_groups:
    - default
      - { get_resource: web_securitygroup }
  user_data:
    str_replace:
      template: |
      #!/bin/bash
      # INSTALL
      yum -y install mariadb mariadb-server httpd wordpress
      touch /var/log/mariadb/mariadb.log
      chown mysql:mysql /var/log/mariadb/mariadb.log
      systemctl start mariadb.service

      # Setup MySQL root password and create a user
      mysqladmin -u root password db_rootpassword
      cat >> EOF
      mysql -u root --password=db_rootpassword
      CREATE DATABASE db_name;
      GRANT ALL PRIVILEGES ON db_name.* TO "db_user"@"localhost"
      IDENTIFIED BY "db_password";
      FLUSH PRIVILEGES;
      EXIT
      EOF
      "http://my wordpress config config/
      see -i "Require local/Require all granted" /etc/httpd/conf.d/wordpress.conf
      see -i "database_name/db_name/" /etc/wordpress/wp-config.php
      see -i "database_name/db_name/" /etc/wordpress/wp-config.php
      see -i "database_name/db_name/" /etc/wordpress/wp-config.php
      systemctl start httpd.service

    params:
      db_rootpassword: { get_param: db_rootpassword }
      db_name: { get_param: db_name }
      db_user: { get_param: db_username }
      db_password: { get_param: db_password }

  outputs:
    WebsiteURL:
      description: URL for Wordpress wiki
      value:
        str_replace:
          template: http://host/wordpress
          host: { get_attr: [wordpress_server, first_address] }
```

Example IaC
Demonstration: HEAT (IaC): Wordpress Example
Sources


