ACCELERATING DEVOPS THROUGH OPENSHIFT
BY RED HAT

Trevor Quinn
PaaS and DevOps Practice Lead, North America
Red Hat Consulting
Agenda

- Current IT Landscape
- Traditional IT organization
- State of DevOps
- Cloud automation
- Application lifecycle automation
- Demo
CURRENT IT LANDSCAPE
Customers

Ubiquitous, timely access to data

Enterprises

Smaller, innovative startups armed with cloud tech
Increased quality

Rapid delivery of product features and service

Better market fit

Doing more with less
TRADITIONAL IT ORGANIZATION
EXPECTATIONS INSIDE IT ORGANIZATIONS

LINES OF BUSINESS
Responsive Delivery

DEVELOPERS
Autonomy, Ability to Focus

QA
Testability

OPERATIONS
Stability
Software should never break.

Ops teams are not required in application design discussions.

Production environments are provisioned/through a mostly manual process.

Developers should not have any access to the production environment.

You have to give a lot of lead time for getting an application environment.

An application is deployed to production after all development is complete.

Deployments are a headache—software is deployed using a mostly manual process.

We cannot keep deploying code to production on a regular basis.
"THROW IT OVER THE WALL"

Walled off people, processes, and technology

Opportunities to improve at the **system** level are potentially lost
STATE OF DEVOPS
DEVOPS LINEAGE

- Agile Development
- Lean
- Continuous Delivery
- Cloud Automation
- Cultural Change
DEVOPS

People

Process

Technology

Information and Reporting
Build and Deploy
Test and Verification
Data Management
Culture and Organization
Release

INITIAL → IMPROVED → OPTIMIZING
CLOUD AUTOMATION
CLOUD COMPUTING

- on-demand self service
- broad network access
- resource pooling
- rapid elasticity
- measured service

(NIST Definition of Cloud Computing)
CLOUD SERVERS

- Ephemeral
- Anonymous
- Multitenant
Cloud automation is DevOps technology.
## Levels of Automation

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Life Cycle Automation</td>
<td>Application</td>
</tr>
<tr>
<td>Application Platform Automation</td>
<td>Containers</td>
</tr>
<tr>
<td>Infrastructure Automation</td>
<td>Container Host</td>
</tr>
</tbody>
</table>
# Levels of Automation

## Application Life Cycle Automation
- Application

## Application Platform Automation
- Containers | Web/app servers | Libraries

## Infrastructure Automation

<table>
<thead>
<tr>
<th>Provisioning resources operating system and down</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Operating systems</td>
</tr>
<tr>
<td>• Network</td>
</tr>
<tr>
<td>• Disk and storage</td>
</tr>
<tr>
<td>• CPU, RAM, and compute</td>
</tr>
</tbody>
</table>

Typically provided by IaaS capabilities such as OpenStack

Virtualization - Limitations

<table>
<thead>
<tr>
<th>Typical use cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Developers, testers, and ops teams requesting VMs</td>
</tr>
<tr>
<td>• Allocating compute power to your applications during peak load times</td>
</tr>
<tr>
<td>• Dynamically adding storage based on consumption</td>
</tr>
<tr>
<td>• Compute governance policies and automatic set up and tear down of resources</td>
</tr>
<tr>
<td>• Utility-based consumption models, pay what you use</td>
</tr>
<tr>
<td>• Does not include application platforms (only VM and down)</td>
</tr>
<tr>
<td>• Standard operating environment</td>
</tr>
</tbody>
</table>
LEVELS OF AUTOMATION

APPLICATION LIFE CYCLE AUTOMATION
Application

APPLICATION PLATFORM AUTOMATION
Provisioning middleware platforms
• Load balancers
• Application servers
• Java/JDK environments
• Stand-alone frameworks

Typically provided by container orchestration and PaaS capabilities such as OpenShift

Typical use cases
• Developers, testers, and ops teams requesting middleware platforms
• Auto-scaling
• Compute governance policies and automatic set up and tear down of resources
• Resource optimization
• Standard operating environment

INFRASTRUCTURE AUTOMATION
Container Host | Virtualization | OS | Bare metal
LEVELS OF AUTOMATION

APPLICATION LIFE CYCLE AUTOMATION
Application life cycle
- Software features, enhancements, versions
- Version control, builds, IDE integration, continuous integration, release management
- Common frames of references for monitoring

Typical use cases
- Continuous integration
- Continuous delivery
- Automated testing

APPLICATION PLATFORM AUTOMATION
Containers | Web/app servers | Libraries

INFRASTRUCTURE AUTOMATION
Container Host | Virtualization | OS | Bare metal
OPENSHIFT
by Red Hat®

- Self-Service
- Multi-Language
- Automation
- Collaboration
- Standards Based
- Web Scale
- Open Source
- Enterprise Grade
APPLICATION LIFECYCLE MANAGEMENT

- Configuration and Change Management
- Automated Testing
- Continuous Integration
- Management/Monitoring
- Deployment Pipelines
CONFIGURATION MANAGEMENT

Definition

All artifacts relevant to the project, and the relationships between them, are stored, retrieved, uniquely identified, and modified. (Humble and Farley, 2011)

Benefits

- Allows you to exactly reproduce an entire environment (OS, system configuration, application server, server configuration, application, etc.)
- Trace changes
- Rollback an environment to earlier working state

Tools

Version control and library repositories
CONFIGURATION MANAGEMENT WITH OPENSSHIFT

Images as Managed Artifacts
CONFIGURATION MANAGEMENT WITH OPENSHIFT

Application Topology as Managed Artifact
**Definition**

Automate tests beyond unit, including integration, system, functional, and even some non-functional acceptance tests (performance, security, etc.)

**Benefits**

- Supports **rapid development** by providing quick feedback (through CI process) on functional breaks, performance degradation
- Safeguards against regression when refactoring

**Tools**

Automated functional and behavior-driven development test suites
AUTOMATED TESTING WITH OPENSHIFT

Traditional

APPLICATION TEST BUILD A
APPLICATION TEST BUILD B
APPLICATION TEST BUILD C

TRADITIONAL STATIC SERVER ENVIRONMENT

OpenShift

APPLICATION TEST BUILD A
APPLICATION TEST BUILD B
APPLICATION TEST BUILD C

EPHEMERAL CONTAINERS DEPLOYED ON PRIVATE CLOUD SERVERS
AUTOMATED TESTING WITH OPENSSHIFT

SELF-CONTAINED UNIT

TESTS RUN ON ALL COMPONENTS:
- OS resources
- Application server
- Configuration
- Application
- Libraries
- And the interactions of all of these
CONTINUOUS INTEGRATION

Definition
Every time somebody commits a change, the entire application is built and a comprehensive set of automated tests are run against it. (Humble and Farley, 2011)

Requires frequent code check ins, good test coverage, preferably a CI server

Benefits
- Normal state of the application is working, functional
- If the application is broken, it is treated as abnormal and requiring immediate attention

Tools
- Version control
- CI server
CONTINUOUS INTEGRATION WITH OPENSHIFT

Developer

1. pushes

Enterprise SCM

1. pushes

2. notifies

Master

3. pulls

Enterprise CI/CD Server

4. compiles

5. saves app binary (if build succeeds)

6. builds image pushes image

Note: Step 5 is redundant in a purely container-driven CI/CD workflow

Enterprise Application Artifact Repository

Enterprise Image Registry

2. notifies

4. unit tests packages application binary

7. reports results
MANAGEMENT AND MONITORING

Definition

Having tools that provide fine grained detail on all aspects of the application lifecycle:

- Design-time API governance
- Build-time quality metrics
- Feature traceability
- Run-time application and platform behavior

Benefits

- Ability to gauge software quality and infrastructure performance
- Ability to gauge DevOps program improvement

Tools

- Application monitoring suites
- Cloud management tools
Standardized container system and orchestration leads to standardized management and monitoring, driving down MTTR.
DEPLOYMENT PIPELINES

Definition
Well-described, automated, measured, and continually optimized process for moving an application through the life cycle from idea to production

Benefits
- Process control over releases: Releases cannot go to production without passing through all prior stages of validation
- Optimization of the entire delivery process: Understanding where bottlenecks are and means to reduce them

Tools
- Self-service requirements of deployment pipelines require mature automation of builds and deployments (including environment provisioning)
- Version control, binary management (e.g. Maven), CI/CD server
DEPLOYMENT PIPELINE EXAMPLE

- **Commit Stage**
  - Compilation
  - Unit Tests
  - Code Quality Tests

- **Acceptance Stage**
  - Environment Configuration
  - Deployment
  - Automated (Functional) Acceptance Tests
  - Integration Tests

- **UAT Stage**
  - Environment Configuration
  - Deployment
  - Manual User Testing
  - Usability Testing

- **Capacity Stage**
  - Environment Configuration
  - Deployment
  - Load Testing
  - Stress Testing
  - Soak Testing
  - Spike Testing

- **Production Stage**
  - Environment Configuration
  - Deployment
  - Rollback Capability

**Flow**

- Developer commits, triggering automated build.
- QA authorizes push-button deploy.
- Operations authorizes push-button deploy.
DEPLOYMENT PIPELINE WITH OPENSHIFT
UAT STAGE EXAMPLE
DEMO (OSE 2)
OPENSSHIFT AS DEVOPS ACCELERATOR
QUESTIONS