ACCELERATE DEVOPS USING OPENSSHIFT PaaS
AGENDA

• World we live in today
• IT organization: Charter, goals, and challenges
• DevOps: Problem statement, what, and why
• How to enable DevOps
• Application lifecycle automation
• Demo
• How can you start?
• Q&A
THE WORLD WE LIVE IN TODAY

Customers and consumers
• Ubiquitous access to data and services
• Impatient, want everything NOW
• Increased QoS expectations

Businesses
• New opportunities and markets
• Threat of being disrupted, intense competition
• Small time frames to get products and services out
THE WORLD WE LIVE IN TODAY

- Increased quality
- Rapid delivery of product features and service
- Doing more with less
BOTH TEAMS ARE THERE TO ENABLE THE BUSINESS
DEVELOPERS
Rapid development

OPERATIONS
Stability

BOTH TEAMS ARE THERE TO ENABLE THE BUSINESS
A methodology to deliver software more efficiently by emphasizing collaboration, communication, and integration across different teams (Dev, QA, Ops) in an IT organization.
TRADITIONAL SOFTWARE DELIVERY ENVIRONMENT
TYPICAL ASSUMPTIONS AND EXPECTATIONS

- 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.
Developers should focus on writing code.
Quality engineers should focus on testing.
Ops engineers should focus on providing reliable and stable environments.
REALIZING EFFICIENCIES

STANDARDIZATION

AUTOMATION

CONTINUOUS IMPROVEMENT
STANDARDIZATION
STANDARDIZATION

STANDARDIZE TECHNOLOGY

- Operating systems (with patch levels)
- Application servers
- Java/JDK/JRE
- Common libraries
- Build and packaging technologies
- COTS

STANDARDIZE PROCESSES

- SDLC
- Release management
- Monitoring
- Escalation management
AUTOMATION
THREE LEVELS OF AUTOMATION

APPLICATION LIFE CYCLE AUTOMATION
Application

MIDDLEWARE PLATFORM AUTOMATION
Web/app servers | Libraries

INFRASTRUCTURE AUTOMATION
Virtualization | OS | Bare metal
THREE LEVELS OF AUTOMATION

APPLICATION LIFE CYCLE AUTOMATION

Application life cycle
• Software features, enhancements, versions
• Release management version control, build, release management, IDE, continuous
• Integration frameworks, common frames of references for monitoring, configuration management

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

MIDDLEWARE PLATFORM AUTOMATION
Web/app servers | Libraries

INFRASTRUCTURE AUTOMATION
Virtualization | OS | Bare metal
# THREE LEVELS OF AUTOMATION

## APPLICATION LIFE CYCLE AUTOMATION

**Application**

## MIDDLEWARE PLATFORM AUTOMATION

**Provisioning middleware platforms**
- Load balancers
- Application servers
- Java/JDK environments
- Stand-alone frameworks

**Typically provided by 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

**Virtualization | OS | Bare metal**
THREE LEVELS OF AUTOMATION

APPLICATION LIFE CYCLE AUTOMATION
Application

MIDDLEWARE PLATFORM AUTOMATION
Web/app servers | Libraries

INFRASTRUCTURE AUTOMATION
Provisioning resources operating system and down
• Operating systems
• Network
• Disk and storage
• CPU, RAM, and compute

Typically provided by IaaS capabilities such as OpenStack

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

AGILE

- PLAN
- CODE
- BUILD
- TEST
- DEPLOY
- MONITOR
- MEASURE ROI–METRICS
- IDENTIFY AREAS OF IMPROVEMENT
CONTINUOUS IMPROVEMENT

THE FIRST WAY:
Systems thinking

(BUSINESS)  (CUSTOMER)

Dev  Ops

*The Three Ways: The Principles Underpinning DevOps by Gene Kim*
CONTINUOUS IMPROVEMENT

THE SECOND WAY:
Amplify feedback loops

_The Three Ways: The Principles Underpinning DevOps by Gene Kim_
CONTINUOUS IMPROVEMENT

THE THIRD WAY:
Culture of continual experimentation and learning

The Three Ways: The Principles Underpinning DevOps by Gene Kim
SO, HOW DO WE DO ALL THAT?
ACCELERATE DEVOPS USING OPENSSHIFT PAAS
PEOPLE

- Cultural paradigm shift
- Cross-training of skills
- Collaboration and involvement of teams across all aspects from designing through monitoring of application
- The question everyone should ask is “Is my application driving business value based on the state it is in now?”
- Short-lived and interim DevOps Enablement Team can be created in organizations
PROCESS

- Agile methodologies
- Governance and continuous feedback loops to reduce and eliminate technical debt
- Define metrics for measure
- Project is not done until the application is driving value for the customers and business
- Automate everything
- If something breaks, don’t hack. Fix the automation script and start over.
- Common frames of reference (for dev, qa and ops) for application monitoring in production
- Open access
- Developers on call
TECHNOLOGY

• Automation is key. OpenShift provides lots of required automation capabilities out-of-the-box.
• Standardize software versions, patch levels, and provisioning mechanisms
• Faster application environment provisioning, root cause analysis
• Notifications and pro-active monitoring
HOW OPENSIFT ACCELERATES DEVOPS

Automation is a cornerstone of DevOps practices.

APPLICATION LIFE CYCLE AUTOMATION
Application

DevOps platform automation leads to efficient, repeatable DevOps application life cycle management. **OpenShift is an accelerator to application DevOps.**

MIDDLEWARE PLATFORM AUTOMATION
Web/app servers | Libraries

**OpenShift** is advanced platform automation. Does not replace DevOps collaboration, but provides a framework for it, so you don’t have to roll your own.

INFRASTRUCTURE AUTOMATION
Virtualization | OS | Bare metal
CONTINUOUS DELIVERY THROUGH OPENSSHIFT

OPENSSHIFT

SELF PROVISIONING
ENVIRONMENT STANDARDIZATION
AUTO SCALING
CENTRALIZED OPS MANAGEMENT

DEV
QA
UAT
PROD

CONTINUOUS DELIVERY

IDEA

IDE
CONFIGURATION MANAGEMENT
CONTINUOUS INTEGRATION
MONITORING

PRODUCT FEATURE

PRODUCT MANAGER
DEVELOPERS
TEST ENGINEERS
OPERATIONS PERSONNEL

ACCELERATE DEVOPS USING OPENSIFT PAAS
HOW OPENSSHIFT ACCELERATES DEVOPS APPLICATION LIFE CYCLE MANAGEMENT

Solves platform automation...
• Standardized operating environments
• Environment configuration as code
• Self-provisioning
...so the DevOps focus can be on application delivery

Continuous integration/delivery
• Integration with major DevOps tools
• Just-in-time delivery (and teardown) of single-purpose platforms for resource efficiency

Ready-to-go framework for application configuration as code
• Action hooks
• Cartridges
• Environment variables

Auto-scaling
CENTURYLINK SAVES TIME AND MONEY, ADDS FLEXIBILITY

CHALLENGE
• Decrease operational cycle time for deploying applications
• Save time and money spent on environment setup and operations
• Migrate from proprietary application servers to open source solutions for easier system administration

SOLUTION
• Migrated from Oracle WebLogic Server to Red Hat® JBoss® Middleware for deploying applications
• Added OpenShift Enterprise by Red Hat as the deployment platform to host internal applications
• Engaged with Red Hat consultants for adopting open source solutions and Red Hat Training for in-depth knowledge on using JBoss Fuse

BENEFITS
• Increased solution flexibility at a lower cost with an open source subscription
• Allowed better separation of operational considerations and application deployment
• Decreased number of application server versions
• Established relationship with Red Hat for further support with products
• Adopted newer technologies with better access to product evaluation and information

TELECOMMUNICATIONS
SOFTWARE AND SERVICES
Red Hat Consulting
Red Hat Enterprise Linux
Red Hat JBoss Web Server
Red Hat JBoss EAP
Red Hat JBoss Operations Network
OpenShift Enterprise by Red Hat
Red Hat JBoss Fuse
Red Hat Training

Location: Monroe, LA
APPLICATION LIFE CYCLE AUTOMATION
APPLICATION LIFE CYCLE MANAGEMENT AUTOMATION TOOLS AND PROCESSES

CONFIGURATION AND CHANGE MANAGEMENT

AUTOMATED TESTING

CONTINUOUS INTEGRATION

MANAGEMENT AND 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: VERSION CONTROL FOR CONTINUOUS DELIVERY

- Most development on trunk
- Short-lived branches for feature development if absolutely necessary
- Long-lived branches in support of releases
CONFIGURATION MANAGEMENT WITH OPENSSHIFT

Configuration management for the application’s operating environment is provided out-of-box:

- OpenShift containers (gears) provide the entire operating environment for the application:
  - OS resources
  - OS secure containerization
  - Application server
  - Application server configuration
- These containers are standardized in OpenShift
- Containers are customized through the OpenShift cartridges, which can be version controlled
### Configuration Management with OpenShift

**OpenShift Cartridges**

<table>
<thead>
<tr>
<th>cartridge name</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ bin</td>
</tr>
<tr>
<td>+ setup</td>
</tr>
<tr>
<td>+ install</td>
</tr>
<tr>
<td>+ post_install</td>
</tr>
<tr>
<td>+ teardown</td>
</tr>
<tr>
<td>+ control</td>
</tr>
<tr>
<td>- hooks</td>
</tr>
<tr>
<td>+ set-db-connection-info</td>
</tr>
<tr>
<td>+ versions</td>
</tr>
<tr>
<td>+ $software_version'</td>
</tr>
<tr>
<td>+ bin</td>
</tr>
<tr>
<td>+ ...</td>
</tr>
<tr>
<td>+ data</td>
</tr>
<tr>
<td>+ .openshift</td>
</tr>
<tr>
<td>+ ...</td>
</tr>
<tr>
<td>(directory/file tree)</td>
</tr>
<tr>
<td>+ template.git</td>
</tr>
<tr>
<td>(discretionary)</td>
</tr>
<tr>
<td>+ ...</td>
</tr>
<tr>
<td>(git bare repo)</td>
</tr>
<tr>
<td>+ ...</td>
</tr>
<tr>
<td>+ env</td>
</tr>
<tr>
<td>+ *.erb</td>
</tr>
<tr>
<td>+ template</td>
</tr>
<tr>
<td>+ ...</td>
</tr>
<tr>
<td>(directory/file tree)</td>
</tr>
<tr>
<td>+ template.git</td>
</tr>
<tr>
<td>(discretionary)</td>
</tr>
<tr>
<td>+ ...</td>
</tr>
<tr>
<td>(bare git repository)</td>
</tr>
<tr>
<td>+ usr</td>
</tr>
<tr>
<td>+ ...</td>
</tr>
<tr>
<td>+ metadata</td>
</tr>
<tr>
<td>+ manifest.yml</td>
</tr>
<tr>
<td>+ managed_files.yml</td>
</tr>
<tr>
<td>+ conf.d</td>
</tr>
<tr>
<td>+ openshift.conf.erb</td>
</tr>
<tr>
<td>+ conf</td>
</tr>
<tr>
<td>+ magic</td>
</tr>
</tbody>
</table>

1. **required** items must exist for minimal OpenShift support of your cartridge
2. **optional** exist to support additional functionality
3. **discretionary** should be considered best practices for your cartridge and work. E.g., `conf.d` is the usual name for where a web framework would install its **httpd** configuration.

---

### Key Features

- **Language runtimes and versions**
- **Application servers**
- **Networking configuration**
- **Scripts to run at various points in the provisioning process**
- **Environment variables**

A standard, consistent way to provide automated provisioning instructions to the PaaS
AUTOMATED TESTING

DEFINITION:
Automate tests beyond unit, including integration, system, functional, and even some non-functional acceptance tests (performance, security, etc.)
• Integrated with the continuous integration process

BENEFITS:
• Supports rapid development by providing quick feedback (through CI process) on functional breaks, performance degradation
• Provide insurance against regression when refactoring

TOOLS:
• Automated functional and behavior-driven development test suites
<table>
<thead>
<tr>
<th>AUTOMATED</th>
<th>MANUAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional acceptance tests</td>
<td>Showcases</td>
</tr>
<tr>
<td>Unit tests</td>
<td>Usability testing</td>
</tr>
<tr>
<td>Integration tests</td>
<td>Exploratory testing</td>
</tr>
<tr>
<td>System tests</td>
<td></td>
</tr>
<tr>
<td>Nonfunctional acceptance tests</td>
<td>(capacity, security, etc.)</td>
</tr>
</tbody>
</table>

Source: Humble and Farley, *Continuous Delivery*, 2011
AUTOMATED TESTING WITH OPENSHIFT

- Integration tests validate a standardized environment, rather a hand-crafted or custom-scripted one

- One form of testing (performance, for example) need not create a bottleneck for the other
  - OpenShift environments are consumed on demand—created for a particular test run and destroyed after run completion
  - Constant cycling of environments means no more queuing waiting for a single test environment to become available
CONTINUOUS INTEGRATION

DEFINITION:
Everytime 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

OpenShift continuous integration encompasses the operating environment of the application ("the platform"), not just the application itself

• Normal state of both the application and the platform is working, functional

• If the application or the platform is broken, it is treated as abnormal and requiring immediate attention

• Frequent, continuous platform builds transform environment provisioning bottlenecks into automated solutions, making it easier to deploy to, scale, and migrate environments
CONTINUOUS INTEGRATION WITH OPENSHEFT

1. PUSHES
2. NOTIFIES
3. PULLS BRANCH
4. PROVISIONS PLATFORM PUSHES BRANCH TESTS TEARS DOWN PLATFORM
5. REPORT RESULTS
CONTINUOUS INTEGRATION WITH OPENSHEET

CI continually deploys platform and application combinations, validating a complete operating environment.
MANAGEMENT AND MONITORING

• Red Hat Satellite 6 (including Foreman and Puppet integration) can be used as the management layer for OpenShift infrastructure.
• Monitoring information provides feedback information to support continuous improvement.
• Open source monitoring tools like Nagios and Zabbix can be integrated into OpenShift solution.
DEPLOYMENT PIPELINES

DEFINITION:
Well-described, 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

DEVELOPER COMMITS, TRIGGERING AUTOMATED BUILD

COMMIT STAGE
- Compilation
- Unit Tests
- Code Quality Tests

ACCEPTANCE STAGE
- Environment Configuration
- Deployment
- Automated (Functional) Acceptance Tests
- Integration Tests

QA AUTHORIZES PUSH-BUTTON DEPLOY

UAT STAGE
- Environment Configuration
- Deployment
- Manual User Testing
- Usability Testing

QA AUTHORIZES PUSH-BUTTON DEPLOY

PRODUCTION STAGE
- Environment Configuration
- Deployment
- Rollback Capability

CAPACITY STAGE
- Environment Configuration
- Deployment
- Load Testing
- Stress Testing
- Soak Testing
- Spike Testing

OPERATIONS AUTHORIZES PUSH-BUTTON DEPLOY

AUTOMATIC
DEPLOYMENT PIPELINES WITH OPENSSHIFT

1. COMMIT STAGE
   - DEVELOPERS
   - ENTERPRISE SCM

2. ACCEPTANCE STAGE
   - DEVELOPERS
   - DEV PaaS

3. UAT STAGE
   - TESTERS
   - TEST PaaS
   - Test Team Approves

4. PRODUCTION STAGE
   - OPERATIONS
   - PROD PaaS
   - Operations Team Approves

ENTERPRISE CI/CD SERVER
ENTERPRISE ARTIFACT REPOSITORY

ACCELERATE DEVOPS USING OPENSIFT PAAAS
DEPLOYMENT PIPELINE PRACTICES

- Only build your binaries once
- Deploy the same way to every environment
- Smoke test deployments
- Deploy into a copy of production
- Every change should propagate through the pipeline instantly
- If any part of the pipeline fails, stop the line

Source: Humble and Farley, *Continuous Delivery*, 2011
PAAS-ENABLED DEPLOYMENT PIPELINE PRACTICES

• Only build your binaries once

• Deploy the same way to every environment

• Smoke test deployments

• Deploy into a copy of production

• Every change should propagate through the pipeline instantly

• If any part of the pipeline fails, stop the line

PAAS IMPLICATIONS

Use PaaS API to create standardized deployment process without having to create your own platform automation.

Use PaaS cartridge specification to create standardized deployment template without having to create your own platform automation.

Use high-density, high-turnover PaaS containers to reduce pipeline queuing.
AN FSI CUSTOMER REDUCED DEPLOYMENT TIMES FROM WEEKS TO DAYS, BECOMES MORE EFFICIENT

FSI

Location: NORTH AMERICA

FINANCIAL SERVICES

SOFTWARE AND SERVICES
Red Hat Consulting
OpenStack
OpenShift
JBoss Enterprise Middleware
Jenkins
GitHub
Zabbix

CHALLENGE

- Decrease time to market for delivering new features and functionality
- Create a homogeneous application platform
- Remove barriers to rapid, iterative development

SOLUTION

- Built a push-button developer stack based on a single PaaS architecture for rapid deployment
- Fully integrated the stack to maximize continuous delivery
- Established common templates for application and middleware creation
- Mentored teams to establish DevOps capability and streamline workflow through governance, process, and operationalization

BENEFITS

- Reduced deployment times from weeks to days
- Improved developer efficiency through the ability to spin up environments without having to wait for infrastructure team
- Facilitated more robust production deployments
DEPLOYMENT PIPELINES
WITH OPENSHEET
RED HAT DEVOPS AND PAAS SERVICE OFFERINGS
## RED HAT CONSULTING CAN HELP

<table>
<thead>
<tr>
<th>Service Offering</th>
<th>Description</th>
<th>LOE</th>
<th>Deliverables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery Workshop</td>
<td>Requirements and use case gathering session</td>
<td>2 days</td>
<td>Journal describing business and technical priorities and our proposed solution</td>
</tr>
<tr>
<td>Architecture Review &amp; Mentoring Service</td>
<td>Cloud Discovery Workshop plus basic software installation use case</td>
<td>2 weeks</td>
<td>Current state assessment, mentoring assessment, software installation and basic mentoring</td>
</tr>
<tr>
<td>Pilot Implementation</td>
<td>Architecture Review &amp; mentoring Service plus design and implementation of a pilot use case for an application</td>
<td>~ 6 weeks</td>
<td>All of the above plus working pilot implementation</td>
</tr>
<tr>
<td>Enterprise Strategy Rollout</td>
<td>Phased approach for legacy migrations and greenfield implementations</td>
<td>~ 6 months and over</td>
<td>Enterprise Strategy, Project Plans, Design and Code artifacts and entire working solution</td>
</tr>
</tbody>
</table>

Skills: Iaas, Paas, Middleware, Enterprise Architecture, SDLC, Agile, DevOps Process Architect
QUESTIONS?