Agile DevOps: Continuous software delivery in the cloud

Leverage an open Continuous Delivery platform

Paul Duvall

January 08, 2013

When developers and operations work together in a collaborative manner, they often need one place to manage the software delivery process and pipeline of changes. A Continuous Delivery (CD) platform addresses this need. In this Agile DevOps installment, DevOps expert Paul Duvall lays out how you can use OpenDelivery, an open CD platform.

View more content in this series

When implemented effectively, Continuous Delivery (CD) embraces the DevOps philosophy of collaboration between traditionally separate teams: development and operations. CD is the automated implementation of the build, deploy, test, and release process. With CD, software is always in a releasable state. With every change to any part of the software system — infrastructure, application code, configuration, or data — a release candidate goes through a delivery pipeline where the software is created, environments are built, databases are constructed, tests are run, and software is deployed. Many tools make up a platform capable of moving the software system along this pipeline.

This article introduces an open source platform for CD called OpenDelivery, developed by Stelligent. OpenDelivery combines open source tools, a cloud infrastructure, and scripts. It supports the concept that the whole software system is code. The scripts are all versioned in a GitHub version-control repository. OpenDelivery uses the numerous tools listed in Table 1 to create the platform, and it supports Rails, Grails, and Java™ development on Linux™. In this article, I'll describe the basic steps for running the platform, using the delivery pipeline, and running jobs to provision environments and deploy applications.

Table 1. Tools used by the OpenDelivery platform

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Web Services (AWS) CloudFormation</td>
<td>OpenDelivery uses CloudFormation — a template language for describing the provisioning of AWS resources — for the full infrastructure automation of Jenkins environments and the configuration of AWS resources for target environments.</td>
</tr>
<tr>
<td>Amazon Elastic Compute Cloud (EC2)</td>
<td>OpenDelivery uses EC2 for all its compute instances.</td>
</tr>
</tbody>
</table>
Amazon CloudWatch
OpenDelivery uses CloudWatch to monitor some basic performance metrics and sends Simple Notification Service (SNS) notifications when thresholds are exceeded.

Amazon Route 53
Route 53 is a highly available and scalable DNS that OpenDelivery uses to set domains to endpoints based on user preferences.

Amazon SimpleDB
SimpleDB is a NoSQL database that is highly available and scalable. OpenDelivery uses it to store pipeline configuration such as IP addresses and CloudFormation stack names.

Amazon Simple Notification Service (SNS)
OpenDelivery uses SNS to send messages when it creates environments.

Amazon Simple Storage Service (S3)
OpenDelivery uses S3 to store ephemeral files used by the platform.

Capistrano
Capistrano is a Ruby-based deployment-specific DSL that OpenDelivery uses to deploy applications to environments.

GitHub
GitHub hosts the OpenDelivery source code. Users of OpenDelivery must also use a GitHub account for their application code.

Jenkins
Jenkins is a Continuous Integration (CI) server. OpenDelivery uses Jenkins as its platform for running all builds and deployments of application code, configuration, data, and infrastructure changes.

Puppet
Puppet is an infrastructure automation tool with its own domain-specific language (DSL). All of the infrastructure automation for the target environments is written in Puppet and called by CloudFormation.

Ruby
Most of the scripts that automate the pipeline are written in Ruby.

Prerequisites
The three prerequisites for successfully installing the OpenDelivery platform are:

- **CloudFormation**: Sign up for AWS CloudFormation to gain access to all of the required services that OpenDelivery uses.
- **GitHub**: You must have a GitHub account. It’s free to sign up with GitHub.
- **Route 53**: You need to set up a hosted zone in Route 53 for a domain that you own. This domain will be used in your OpenDelivery scripts to address the resources that are created as a part of the platform.

See Related topics for a link to the OpenDelivery Quick Start Guide. Be aware that in using OpenDelivery, you will be charged AWS fees for resource usage.

Provisioning the continuous-integration platform
The first step in setting up the OpenDelivery CD platform for your projects is to provision the Jenkins CI server in AWS. To begin, run the CloudFormation setup template. You will be prompted to enter the following parameters:

- **ApplicationName**: The CNAME prefix for the Jenkins CI environment you’re creating. For example http://ApplicationName.domainname.com.
• **GithubUsername**: Your GitHub username.
• **ProjectName**: The name of your GitHub project. For example, if my project's GitHub URL is https://github.com/stelligent/continuous_delivery_open_platform, then my ProjectName is continuous_delivery_open_platform.
• **Email**: The email address where you’d like messages sent.
• **HostedZone**: The domain name you own and have created a hosted zone for in Route 53.
• **GithubOrganization**: The name of your GitHub organization. For example, if my project's GitHub URL is https://github.com/stelligent/continuous_delivery_open_platform, then my GithubOrganization is stelligent.
• **GithubPassword**: Your GitHub password for the GithubOrganization and ProjectName.
• **KeyName**: The name of your EC2 key pair. Do not include any file suffixes.
• **InstanceType**: The EC2 instance type. Use the default if you are unsure.

After you complete the wizard, the platform will take about 20 minutes to build. When it’s done building, go to the CloudFormation Console and select the check box next to the stack with jenkinsstack in its name. Then, click the **Outputs** tab. Copy the value from the Domain key row and enter this value in your web browser to launch the newly created Jenkins CI server running in AWS. You'll see the initial Jenkins configuration, shown in Figure 1:

**Figure 1. Initial Jenkins configuration after it's provisioned using CloudFormation**

<table>
<thead>
<tr>
<th>All</th>
<th>DeliveryPipeline</th>
<th>Email Group Administration</th>
<th>+</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>W</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>🌞</td>
<td>🍃</td>
<td>AddUserToEmailGroup</td>
<td></td>
</tr>
<tr>
<td>🌞</td>
<td>🍃</td>
<td>Build</td>
<td></td>
</tr>
<tr>
<td>🌞</td>
<td>🍃</td>
<td>BuildAMT</td>
<td></td>
</tr>
<tr>
<td>🌞</td>
<td>🍃</td>
<td>CapistranoModuleBuild</td>
<td></td>
</tr>
<tr>
<td>🌞</td>
<td>🍃</td>
<td>CommonStepDefinitionGemBuild</td>
<td></td>
</tr>
<tr>
<td>🌞</td>
<td>🍃</td>
<td>CreateTargetEnvironment</td>
<td></td>
</tr>
<tr>
<td>🌞</td>
<td>🍃</td>
<td>DeployApplication</td>
<td></td>
</tr>
</tbody>
</table>

**A CD pipeline**

With OpenDelivery, a standard delivery pipeline is provided out of the box. It runs the StartDeliveryPipeline, Build, CreateTargetEnvironment, and DeployApplication Jenkins jobs in sequence. If any of these jobs fails, the pipeline fails and software is not deployed.

An example of a delivery pipeline — as represented in Jenkins — is shown in Figure 2:
Figure 2. Delivery pipeline as represented in the Jenkins CI server

In Figure 2, the software goes through three stages. First, after a build is run through a CI process, Jenkins runs any required setup using the StartDeliveryPipeline job. Then it creates a target environment with the CreateTargetEnvironment job. Finally, it deploys the application into the target environment as represented in the DeployApplication job. These three stages run as a unit of work in which if any of the jobs fails, the pipeline fails.

By establishing a delivery pipeline such as this, you assess a release candidate for risk — as more and more automated processes occur — with each step in the pipeline the software goes through. This means that once it successfully completes the pipeline, the release candidate could potentially be released to production if the business chooses to do so. You can also configure the pipeline to deploy to production.

Provisioning target environments

In the context of the OpenDelivery platform, environments are defined as the operating system on EC2 instances along with other provisioned resources — such as security groups, domain configuration, and storage. Once an environment is provisioned, software can be deployed to it. The Jenkins job that is primarily responsible for running the scripts that create these environments is the CreateTargetEnvironment job.

The delivery pipeline in OpenDelivery creates environments from scratch by calling a CloudFormation template. The template executes Puppet scripts (calling puppet apply). (For more information on Puppet, see "Agile DevOps: Infrastructure automation.")

Listing 1 shows the portion of the CloudFormation template that calls Puppet:

Listing 1. CloudFormation snippet from production.template that calls Puppet

```
"# Install Ruby 1.9.3\n",
"rpm -Uvh /tmp/ruby-1.9.3p0-2.amzn1.x86_64.rpm\n",

"# Install Puppet 3.0.1 from Rubygem\n",
"gem install puppet --no-rdoc --no-ri\n",
"groupadd puppet\n",

"# Run Puppet\n",
"puppet apply --modulepath=/home/ec2-user/modules /home/ec2-user/manifests/site.pp\n",
```

OpenDelivery provides these scripts in its GitHub repository. You can alter the standard scripts for the servers and configuration in use in your infrastructure.

Running deployments

Automated deployments run in Capistrano. You can use Capistrano to deploy to multiple types of development and OS platforms. These deployments are run from Jenkins jobs onto environments created from the CloudFormation and Puppet scripts described in the preceding section. The Jenkins job that is primarily responsible for running the scripts that run the deployment is the DeployApplication job.

The snippet in Listing 2 shows a Capistrano script that runs preconfigure and deploy tasks:

Listing 2. Capistrano snippet from deploy.rb that resets an environment and runs a deployment

```ruby
Listing 2. Capistrano snippet from deploy.rb that resets an environment and runs a deployment

    task :preconfigure do
        run "cd #{deploy_to} && sudo rm -rf #{deploy_to}/#{artifact_name}"
        config_content = from_template("config/templates/s3_download.erb")
        put config_content, "/home/ec2-user/s3_download.rb"
        run "sudo chmod 655 /home/ec2-user/s3_download.rb"
    end

    task :deploy do
        run "sudo ruby /home/ec2-user/s3_download.rb --outputdirectory #{deploy_to}/ \
            --bucket #{s3_bucket} --key #{artifact}"
        case
            when language == "rails"
                run "cd #{deploy_to} && sudo tar -zxf #{artifact}"
                run "sudo chown -R ec2-user:ec2-user #{deploy_to}/#{artifact_name}"
        end
    end

You'll find additional Capistrano recipes in a subdirectory within the directory named deployment in the OpenDelivery GitHub repository. In Listing 2, examples of these recipes are stripe (stripe.rb) and virtualhost (virtualhost.rb). You can add, modify, or delete recipes based on your deployment needs.

Continuous testing

The OpenDelivery platform includes essential Cucumber acceptance tests that run against the infrastructure (the CI and target environments) and deployments. These tests run every time that changes are applied to scripts that provision environments or run deployments from the CreateTargetEnvironment and DeployApplication jobs, respectively. The example in Listing 3 is from a Cucumber script in the OpenDelivery platform that verifies that certain server components are installed and operational:
Listing 3. Snippet from a Cucumber script in OpenDelivery (production.feature) that runs automated tests against an environment

<table>
<thead>
<tr>
<th>Feature: Scripted Provisioning of Target Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>As a Developer</td>
</tr>
<tr>
<td>I would like my target environment provisioned correctly so I can deploy applications to it</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Background:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given I am sshed into the environment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario: Is the proper version of Postgresql installed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>When I run &quot;/usr/bin/postgres --version&quot;</td>
</tr>
<tr>
<td>Then I should see &quot;8.4&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario: Is the proper version of Apache installed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>When I run &quot;/usr/sbin/httpd -v&quot;</td>
</tr>
<tr>
<td>Then I should see &quot;2.2&quot;</td>
</tr>
</tbody>
</table>

... You can modify or extend the supplied tests to test more environment or deployment configuration. 

**Build, deploy, test, and release with every change**

In this article, you learned how OpenDelivery leverages open source tools and configuration to create a platform for continuous software delivery in the cloud, and how you can take advantage of the platform. Continuous delivery works best for development and operations teams that work together. The next installment talks about how to break down departmental silos and create cross-functional DevOps teams.
Related topics

- "Continuous Delivery in the Cloud" (Brian Jakovich, Stelligent, October 2012): A six-part series on creating a CD platform.
- OpenDelivery Quick Start Guide: Instructions for setting up the OpenDelivery CD platform.
- DevOps in the Cloud LiveLessons: Create a Continuous Delivery Platform Using AWS and Jenkins: Find out about Paul Duvall's video-training series on continuous delivery.
- Collaborative development and operations: Find out how IBM SmartCloud Continuous Delivery helps you deploy application changes from IBM Rational Team Concert into a cloud environment.
- Java technical podcasts: The cloud files: A series of interviews with the companies, people, and technologies impacting cloud computing today.
- OpenDelivery: Download the source code and templates for running the OpenDelivery platform.
- IBM Tivoli Provisioning Manager: Tivoli® Provisioning Manager enables a dynamic infrastructure by automating the management of physical servers, virtual servers, software, storage, and networks.
- IBM Tivoli System Automation for Multiplatforms: Tivoli System Automation for Multiplatforms provides high availability and automation for enterprise-wide applications and IT services.
- Evaluate IBM products in the way that suits you best.

© Copyright IBM Corporation 2013
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)