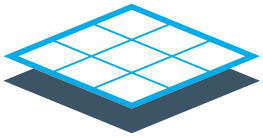

PLATFORM9

The Seven-Step Recipe for Continuous Integration Using OpenStack

An exploration of how to use the Platform9 SaaS product to successfully implement a CI/CD development-to-deployment workflow, including the benefits of OpenStack, automation requirements, detailed test and deployment procedure, and useful OpenStack features.

PLATFORM9

Table of Contents

Executive Summary

This white paper discusses the seven steps required for successfully implementing a fully automated Continuous Integration/Continuous Delivery (CI/CD) development-to-deployment workflow from a SaaS (Software as a Service) perspective using the Platform9 product and deployment process as the example. The specific topics covered include:

- Why a completely automated CI/CD workflow can benefit your product development and deployment.
- How using OpenStack can benefit CI/CD workflow automation.
- Requirements for automating your CI/CD workflow.
- The detailed seven-step recipe used by Platform9 to test and deploy our product.
- Some OpenStack features that we have found very useful at Platform9.

Why Completely Automate Your CI/CD Workflow?

A SaaS company must be able to rapidly ship code without sacrificing quality in order to survive and thrive. Providing a 100% automated CI/CD build infrastructure accelerates development by allowing the developers to check in code and immediately move on to other tasks while the system automatically performs the appropriate unit, integration, and other tests. If the tests pass, then the system automatically checks the new code in to the main source code branch. If there is a problem, then the system notifies the developer. Automating tests has the added benefit of encouraging developers to write more tests, thereby driving a culture of test-driven development.

Increasing developer efficiency allows your organization to ship code more rapidly. Pairing this efficiency with automated testing ensures that faster development does not mean sacrificing high quality.

Benefits of Using OpenStack for CI/CD Automation

OpenStack is an open source project that offers a number of key benefits for running a private cloud as efficiently as a public cloud. It is becoming the standard for current and future private clouds, thanks to a thriving community and support from key organizations in the enterprise infrastructure space such as Cisco, VMware, Rackspace, EMC, and IBM.

Some of the key benefits of using OpenStack to automate a CI/CD workflow include:

- Efficient resource usage
- End-to-end automation
- Better visibility and sharing

Let's look at each of these benefits in more detail.

EFFICIENT RESOURCE USAGE

OpenStack includes several features for getting the most from your available infrastructure resources. For example, the auto-placement engine matches *instances* (virtual machines) with the appropriate resources while honoring administrator-defined policies. You can also set over-subscription levels for CPUs, RAM, and storage.

END-TO-END AUTOMATION

OpenStack also contains a number of features that can help create an automated CI/CD workflow, such as:

- Including a large catalog of standard virtual machine images.
- Allowing a developer to upload SSH keys that can be injected into virtual machine images during deployment.
- Tagging virtual machines with simple key value pair tags in a manner similar to that used by Amazon EC2. This can be a very powerful way to annotate virtual machines with appropriate information that is consumed at run time.
- Supporting cloud-init based customization allows scripts (Chef, Puppet, Ansible, etc.) to be injected into a virtual machine once it has powered on and acquired an IP address. This high level of customization support is key for automating CI/CD workflows.
- Taking a snapshot of a virtual machine to create a new image at any time.

All of these features are available via simple and intuitive RESTful APIs, script wrappers, and the Command Line Interface (CLI).

BETTER VISIBILITY AND SHARING

OpenStack includes a number of dashboards and reports that give your Developer Operations (DevOps) team real

time visibility into your infrastructure, builds, and current workload. An administrator who spots an error can quickly pause and snapshot the affected resource(s) and then share those snapshots and associated logs with developers for faster troubleshooting and debugging.

Note: Platform9 typically recommends running your automated CI/CD workflow in a dedicated tenant to facilitate resource allocation and enhance security while improving visibility.

Requirements

Your infrastructure must meet the following requirements in order to support a fully automated CI/CD workflow with OpenStack:

- OpenStack private cloud
- Vanilla virtual machine images
- CI/CD workflow automation tool
- Version control system

Let's look at each of these requirements in more detail.

OPENSTACK PRIVATE CLOUD

Your organization must have an OpenStack private cloud with sufficient CPU, RAM, and storage resources to support your automated CI/CD workflow. You can do this in one of three ways:

- Download and deploy OpenStack to your own environment on your own.
- Pay a provider to deploy both the hardware and the OpenStack environment.
- Partner with an OpenStack vendor such as Platform9, thereby making it easy to go from bare metal to a fully operational OpenStack environment.

VANILLA VM IMAGES

Your environment must include a catalog with an assortment of vanilla virtual machine images that completely encompasses all of the operating system versions used by your automated CI/CD workflow. Platform9 recommends starting with minimal operating system images that include cloud-init and then customizing those images dynamically as part of your automated CI/CD workflow.

Note: Platform9 includes these images out of the box along with our managed OpenStack deployment.

CI/CD WORKFLOW AUTOMATION TOOL

Your environment must include a CI/CD workflow automation tool. Jenkins and TeamCity two of the most widely used tools in this category.

- Jenkins has been one of the most popular tools for years because it is both free and open source. The primary drawback is that it requires a number of plug-ins for every function that you want to run with it—and the burden is often on you to find and install the correct plug-ins.
- TeamCity includes a number of useful functions out of the box, which allow you to get up and running more quickly. The primary drawback is that it is a paid, proprietary product. Platform9 currently uses TeamCity.

VERSION CONTROL SYSTEM

Your organization must have a good version control system such as Perforce, Subversion, or Git. Platform9 prefers Git because it is very efficient, creates branches quickly, uses a distributed peer-to-peer model, and includes a full history tree that is available even when you are offline. The primary drawback is that it tends to be a complex system with a steep initial learning curve.

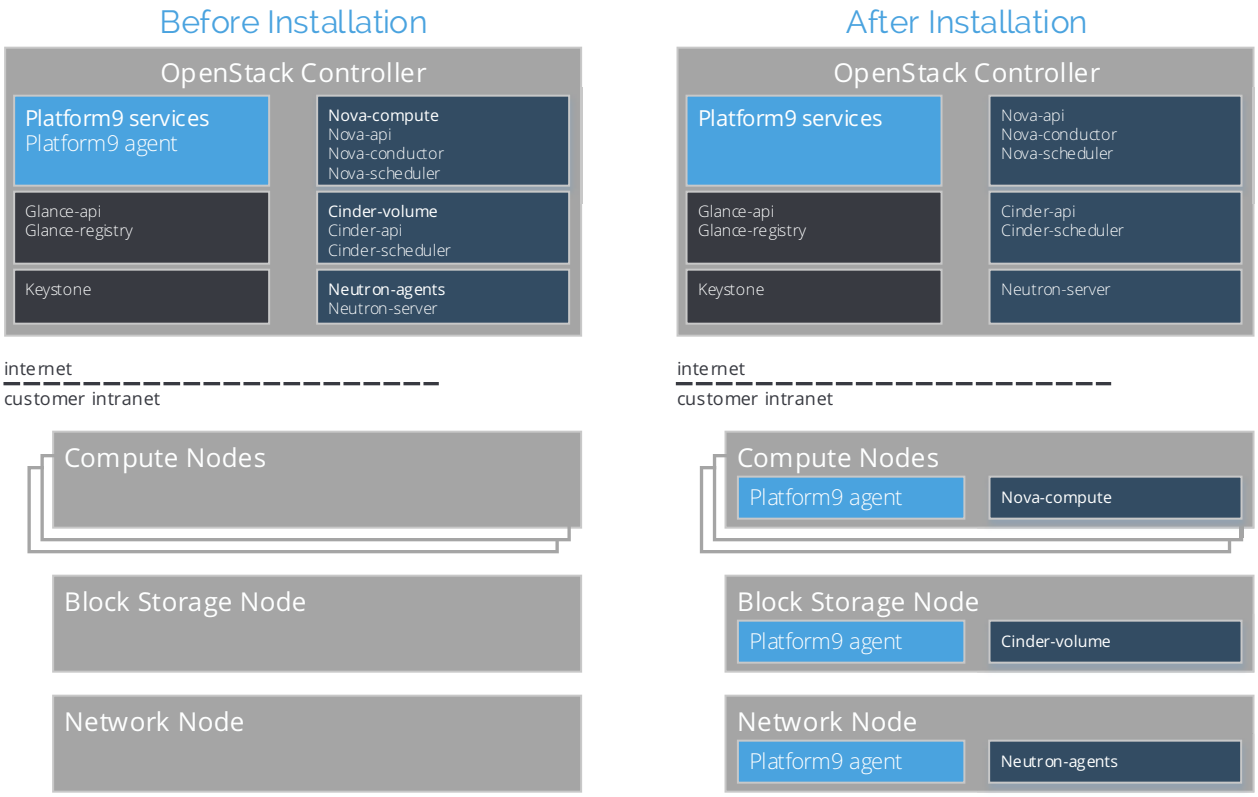

Figure 1: Platform9 automatically installs the appropriate agents and packages on network nodes.

The Recipe

We will now explore the seven-step recipe in more depth using the Platform9 product itself as our example. Platform9 is an OpenStack private cloud that we offer to our customers through a SaaS model. During implementation:

1. Platform9 deploys OpenStack Controllers that contain a number of software components.
2. The customer selects her or his desired hardware nodes and then deploys the Platform9 agent on each selected node by downloading it from her or his Controllers.
3. The customer specifies a role for each node (such as Compute, Block Storage, or Network).
4. The Platform9 OpenStack Controllers deploy the correct OpenStack agent to each node (such as Nova-compute, Cinder-volume, or Neutron agents) and then establishes communications between those agents and their management components on the Controllers.

All software required for a Platform9 deployment comes from the Controller nodes where they are packaged as an assortment of self-contained, single-purpose artifacts that are built from individual components. So... How do we use the seven-step recipe to build our product from raw components to the final image?

STEP ONE: PUSH COMPONENT CHANGES

The seven-step process starts with the individual components that make up our product. Platform9 uses a dedicated Git repository for each component, and each

repository contains its own build scripts and unit tests. Our developers commit all changes to the `automerge` branch in the repository.

STEP TWO: BUILD THE COMPONENT

The automated CI/CD workflow takes over as soon as the developer commits the new code. TeamCity monitors the `automerge` branch and then performs the following tasks when it detects changes:

1. Checks out the code.
2. Builds the component.
3. Runs unit tests.

If the unit tests pass, then TeamCity moves on by:

1. Archiving the prior artifacts.
2. Marking the build as successful.
3. Merging the changes to the master branch and proceeding to build the Platform9 product.

STEP THREE: BUILD THE IMAGE

TeamCity checks for new component builds every hour, which allows rapid code deployment without wasting system resources. A change to one or more component(s) triggers TeamCity to create a new build of the overall product that will ultimately produce a cloud image by:

1. Checking out both the integration repository that represents the entire product and all dependent artifacts from all dependent components. TeamCity includes robust support for dependency trees.
2. Running the integration scripts stored in the integration repository. These scripts interact with our OpenStack private cloud using a variety of methods such as:
    - Using the robust CLI tools that offer "quick and dirty" OpenStack automation, such as checking build status and then making a decision based on that status.
    - Invoking HTTP requests.
    - Using a wrapper such as a Python or client library. We use Python for advanced tasks or when working with code that will be edited by multiple developers.
3. Creating a bare temporary instance using OpenStack images. Platform9 leverages the OpenStack image catalog to store some standard operating system images, such as Ubuntu or CentOS. The Platform9 Controllers are currently based on CentOS.
4. Using Ansible to install an artifact of the Platform9 product onto the instance and update the operating system with all applicable security patches.

Note: You can use other configuration management tools such as Chef, Puppet, or custom Bash scripts.

5. If the installation is successful, TeamCity shuts down the instance and uses the OpenStack snapshot feature to create a cloud Deployment Unit (DU) image from that instance.

STEP FOUR: DEPLOY INSTANCES

With any luck, the DU image created in Step Three will ultimately become a useful "golden" image by passing the integration tests that exercise the software contained within that image. Each integration test requires a dedicated isolated test environment that consists of a *pod* (collection) of virtual machines that replicates a real-world customer deployment of the Platform9 product. Each pod includes dedicated OpenStack Controllers and additional nodes, and the integration tests simulate installing the Platform9 agents and services and then establishing

communications, as described at the beginning of this section.

STEP FIVE: PERFORM INTEGRATION TESTING

Platform9 performs an ever-increasing number of integration tests to ensure the reliability of our product. These tests exercise product features such as:

- Uploading images to the private cloud.
- Spawning instances.
- Connecting instances over a virtual network.

Here again, the fact that our product consists of an OpenStack deployment allows us to leverage built-in OpenStack automation, such as using RESTful APIs to integrate with the cloud controller and the Fabric Python tool to automate running SSH commands inside instances.

STEP SIX: TAG THE IMAGE

Completion of testing triggers a status determination for the overall build. If all tests have passed, then the DU image is tagged with a PASS flag. If one or more test(s) fail, then the DU image is tagged with a FAIL flag.

STEP SEVEN: CLEANUP RESOURCES

The final step of the seven-step recipe is to clean up the resources used by the build and integration tests. If all tests have passed, then the instances are deleted. Any failure preserves those instances and allows developers to log in for troubleshooting and debugging.

The images themselves are retained for possible reuse, but all images are subject to eventual *pruning* (deletion after any applicable grace period). Using a private cloud as the primary tool for automating your CI/CD workflow creates a large number of objects such as instances and temporary repository branches, and pruning keeps the cloud from filling up with useless clutter. Platform9 uses a scheduled TeamCity job to look for objects older than the appropriate grace period, which is four hours for instances or 12 hours for images. Any object exceeding its grace period is deleted. We have some exceptions to this policy, such as:

- Keeping the last known good build for future reuse.
- Preserving personal development environments located on the private cloud.
- Objects tagged with a DON'T DELETE tag.

APPENDIX: Useful OpenStack Features

Platform9 has found the following OpenStack features to be particularly useful in our own automated CI/CD workflow:

- **Instance flavors:** Platform9 spawns a large number of instances during testing, and each test requires various resources to run. Using flavors that define the CPU, RAM, and storage resources to allocate to each instance allows us to optimize our resource usage. We have a large number of flavors that we can tune for various workloads.
- **Host aggregates:** Host aggregates guide instance placement, such as for nested virtualization requirements. When deploying an instance, OpenStack uses the flavor parameters to determine the best host to use for satisfying the request, taking resource usage statistics into account for load balancing. This suffices for most cases, but some instances require particular hosts. For example, when it is required to run an instance inside another instance (nested virtualization), we use host aggregates and flavors with the `nested-virt=true` flag to specify the suitable hosts.

Sample Environment

The Platform9 build/test environment consists of:

- HP Proliant DL 385 servers with 12-core AMD Opteron CPUs, 100GB DDR3 SDRAM, and 500GB storage.
- 100GHz total computing speed, 500GB total RAM, and 2.5TB total storage
- Typical CPU over-allocation is about 5x-6x, with up to 16x supported.
- Support for 1.5x RAM over-allocation (rarely used, to avoid slowing non-test workloads.) This is one reason why we recommend running your automated CI/CD workflow in a separate tenant, if possible.
- We spin up and tear down 1,000+ instances every day.
- Common instance flavors include:
  - 1 vCPU, 1GB RAM, and 10GB storage
  - 1 vCPU, 4GB RAM, and 8GB storage

About Platform9

Platform9's OpenStack-powered service transforms an organization's existing servers into an AWS-like agile and efficient self-service private cloud at any scale within minutes while leveraging the latest open source innovations. Platform9 Managed OpenStack is the first 100% cloud-managed platform for KVM, VMware vSphere, and Docker. Founded in 2013 by a team of early VMware engineers, Platform9 is backed by Redpoint Ventures and is headquartered in Sunnyvale, CA.

- **Tagging:** This allows us to build metadata for images, such as indicating a successful build or protecting a resource from pruning. We can look up objects based on their tags.

For Further Information

This white paper described how Platform9 uses our internal OpenStack private cloud to automate our CI/CD workflow, thereby ensuring both rapid development and high product quality. If you would like more information or to schedule a demonstration, please contact us at:

- **Email:** support@platform9.com
- **Phone:** +1-650-898-7369

To learn more about Platform9, visit **www.platform9.com**.

- Blog
- Twitter
- LinkedIn
- Facebook
- YouTube
- SlideShare

PLATFORM9