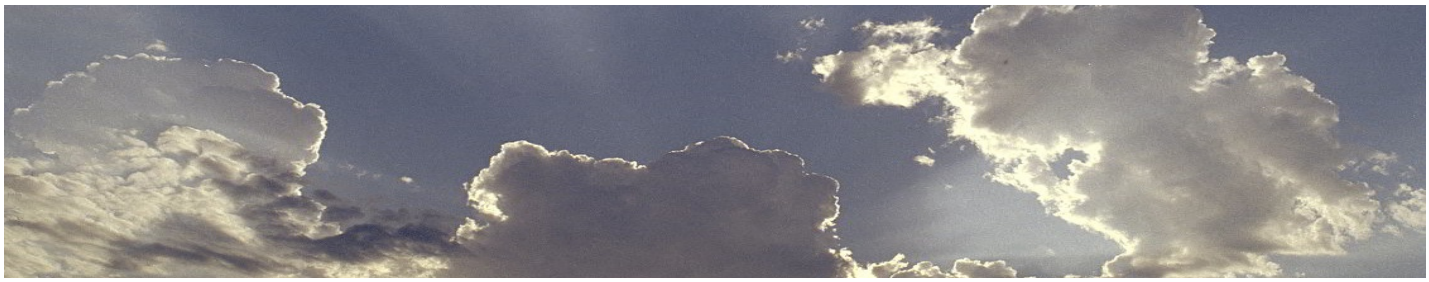
Version 1.1 August 15, 2009



Web Ops 2.0:

Achieving Fully Automated Provisioning

Contributors:

Damon Edwards, DTO Solutions

Andrew Schafer, Reductive Labs

Anthony Shortland, DTO Solutions

Alex Honor, ControlTier Project

Lee Thompson, Former VP & Chief Technologist of E*TRADE Financial

Introduction

E-commerce and software-as-a-service business models have matured quickly, but the quality of the web operations that support these businesses has lagged behind.

Outages are all too common. High variability and defect rates are bemoaned but have become an accepted reality. Key engineers spend all day (and sometimes all night) mired in deployment issues and bottlenecks. And topping it all off, what tooling that does exist are usually a custom one-offs that are brittle and expensive to maintain.

" Today's business of operating software over the Web as a revenue producing service is a dramatic departure from the days when software was primarily produced for delivery on physical mediums..."

Today's business of operating software over the Web as a revenue producing service is a dramatic departure from the days when software was primarily produced for delivery on physical mediums and IT Operations was considered a back-of-the-house support function. Shouldn't we be completely rethinking our tooling and operational capabilities to match these new innovations?

In short, we need to get out of Web Operations 1.0 -- mired in legacy tools, outdated approaches, and low expectations -- and into Web Operations 2.0 where tools and procedures are built from the ground up for highly efficient, reliable, and agile operations.

Web Operations 1.0

Web Operations 2.0

There are multiple factors that go into achieving excellence in Web Operations, but the linchpin that holds it all together is a fully automated provisioning system.

In this paper we will be:

1. Defining what we mean by "fully automated provisioning"
2. Explaining why virtualization and cloud computing efforts fail without fully automated provisioning capabilities
3. Proposing a reference open source tool chain for fully automated provisioning
4. Describing a live implementation where a leading online retailer is actively rolling out a fully automated provisioning system using all open source tools

What is Fully Automated Provisioning?

"Fully automated provisioning" means having the ability to deploy, update, and repair your application infrastructure using only pre-defined automated procedures. While this might seem like a straightforward concept, the trick is in meeting the criteria:

1. *Be able to automatically provision an entire environment -- from "bare-metal" to running business services -- completely from specification*

   Starting with bare metal (or stock virtual machine images), can you provide a specification to your provisioning tools and the tools will in turn automatically deploy, configure, and startup your entire system and application stack? This means not leaving runtime decisions or "hand-tweaking" for the operator. The specification may vary from release to release or be broken down into individual parts provided to specific tools, but the calls to the tools and the automation itself should not vary from release to release (barring a significant architectural change).

2. *No direct management of individual boxes*

   This is as much a cultural change as it is a question of tooling. Access to individual machines for purposes other than diagnostics or performance analysis should be highly frowned upon and strictly controlled. All deployments, updates, and fixes must be deployed only through specification-driven provisioning tools that in turn manages each individual server to achieve the desired result.

3. *Be able to revert to a "previously known good" state at any time*

   Many web operations lack the capability to rollback to a "previously known good" state. Once an upgrade process has begun, they are forced to push forward and firefight until they reach a functionally acceptable state. With fully automated provisioning you should be able to supply your provisioning system with a previously known good specification that will automatically return your applications to a functionally acceptable state. The most successful rollback strategy is what can be described as "rolling forward to a previous version". Database issues are generally the primary complication with any rollback strategy, but it is rare to find a situation where a workable strategy can't be achieved.

4. *It's easier to re-provision than it is to repair*

   This is a litmus test. If your automation is implemented correctly, you will find it is easier to re-provision your applications than it is to attempt to repair them in place. "Re-provisioning" could simply mean an automated cycle of validating and regenerating application and system configurations or it could mean a full provisioning cycle from the base OS up to running business applications.

5. *Anyone on your team with minimal domain specific knowledge can deploy or update an environment*

   You don't always want your most junior staff to be handling provisioning, but with a full automated provisioning system they should be able to do just that. Once your domain specific experts collaborate on the specification for that release, anyone familiar with a few basic commands (and having the correct security permissions) should be able to deploy that release to any integrated development, test, or production environment.

"Fully automated provisioning" means having the ability to deploy, update, and repair your application infrastructure using only pre-defined automated procedures.

The Toss Test!*

An alternate (and purely hypothetical!) test to determine how successful your provisioning automation is:

1. Grab any machine, rip it out of the rack, and throw it out of the window. Can you automatically re-provision your systems and return the affected application services to their previous state in minutes? (no cheating by failing over to a standby cluster or alternate facility)
2. Grab any senior engineer and throw him or her out of the same window. Can your operations proceed as normal?

*adapted from the "The 10th Floor Test" by by Steve Traugott (www.infrastructures.org)

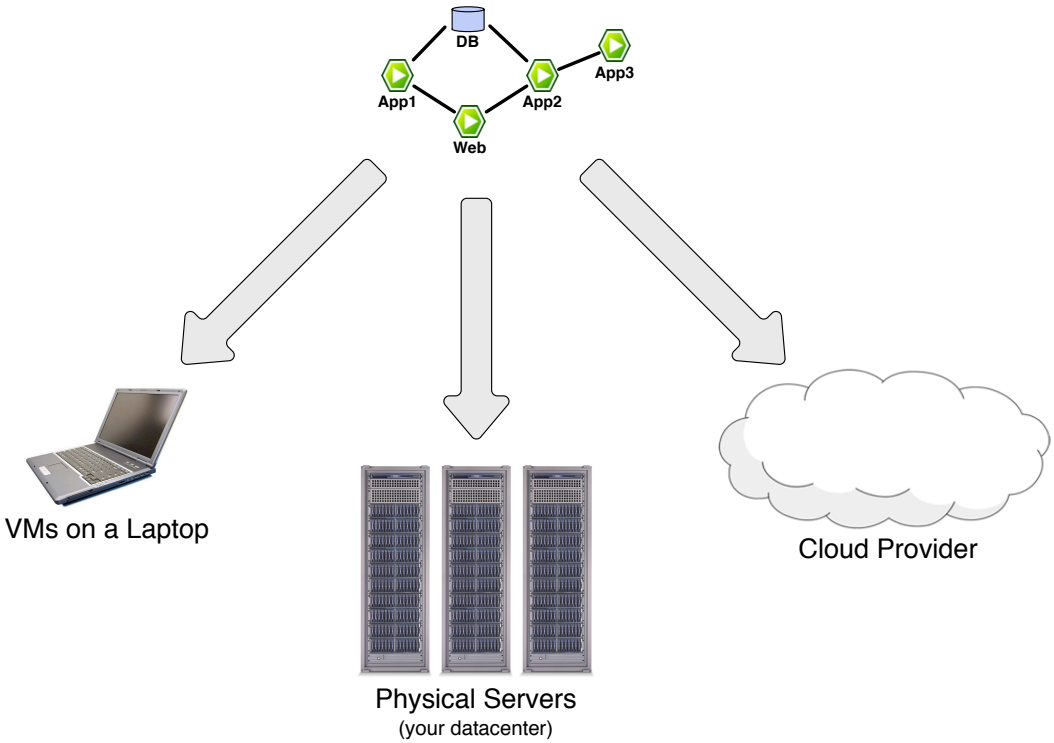
Virtualization and Cloud Computing Fail without Fully Automated Provisioning

Virtualization and Cloud Computing are designed to cut costs and allow for on-demand capacity models. The technical implementations and business theory behind these movements are sound. Virtualized hardware and operating systems are production ready and there are a myriad of business waiting to sell on-demand infrastructure as a service. If all signs point to "go", then what is standing in the way of mass production environment adoption of these technologies?

To take advantage of this new dynamic infrastructure, enterprises must have the ability to automatically provision full application stacks.

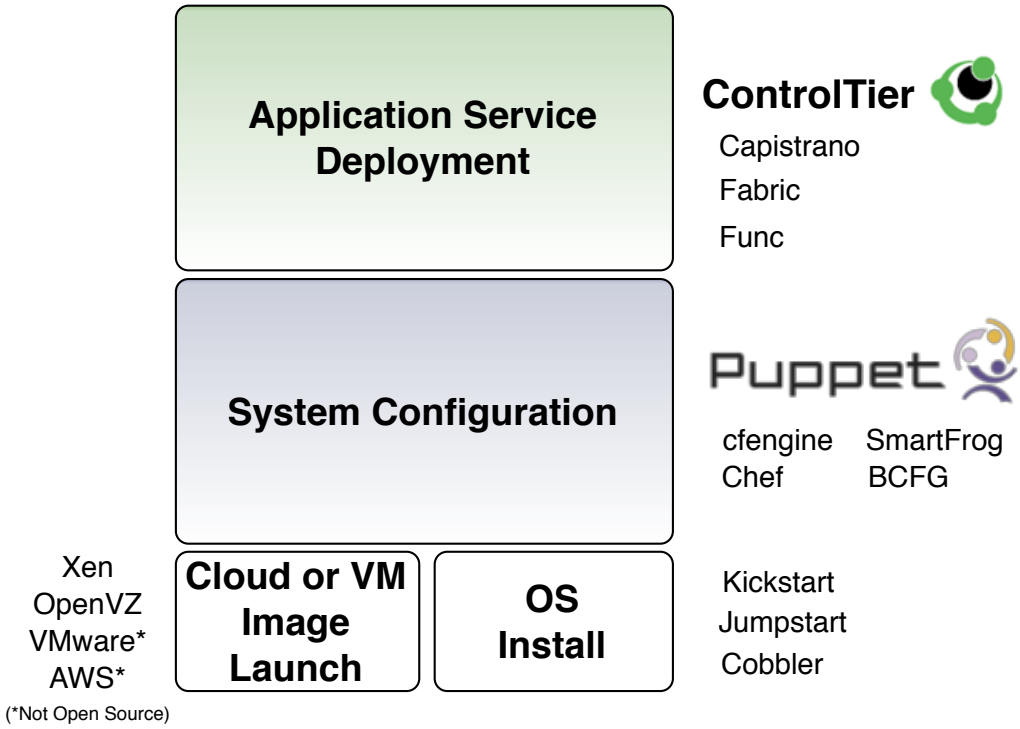
To take advantage of this new dynamic infrastructure, enterprises must have the ability to automatically provision full application stacks. Without fully automated provisioning, trying to deploy and manage applications on top of a dynamic and elastic infrastructure creates a bigger problem than existed before.

While application code tends to be fairly portable, it's the way that the various components of a business application are deployed, configured, and managed that makes the application brittle and locked into a particular environment. Fully automated provisioning provides the abstraction that makes it simple to move that application stack to a different environment -- physical or virtual.



Open Source Toolchain for Fully Automated Provisioning

This toolchain is designed to rely on each type of tool for what it does best.



This toolchain is designed to rely on each type of tool for what it does best. It also recognizes a conceptual separation between the systems level management and applications level management.

The downside of not having a conceptual separation between the management of systems and applications is that you run the risk of quickly creating a brittle and inflexible management nightmare. In real-world operations, your goals for maintaining system configurations and application configurations/state are often at odds. System configurations generally need to be maintained as consistently as possible, but application configurations can constantly change based on shifting business needs and deployment cycles. These opposing needs are one of the reasons why different management strategies are required within the same toolchain.

Below you'll find a description of each layer in the full automated provisioning toolchain, in the the order they would be invoked in a common provisioning scenario.

Moonshine:
Moonshine from Rails Machine is a basic example of an integrated implementation of an open source provisioning toolchain for a specific Ruby on Rails stack.

Moonshine integrates Puppet and Capistrano behind a pure Ruby interface to deploy and configure an Ubuntu, Ruby Enterprise Edition, Apache, Passenger, and MySQL stack. If you are a pure Ruby shop using these technologies, this is a handy option.

https://github.com/railsmachine/moonshine/

Operating System Install/Imaging

The role of this tool is to provide a base OS image. The user selects an image type or system profile and the tool handles provisioning and starting each specific OS instance. How this is executed will vary substantially depending on whether or not you are working with virtualized/cloud servers or native OS instances. Once a minimal OS

instance is running and has network connectivity, this tool's final step is to install and invoke the System Configuration client. This is generally the last you will see of the Operating System Install/Imaging tool unless re-imaging the server is needed as part of a major update or disaster recovery procedure.

Example: Kickstart, Jumpstart, Cobbler, OpenQRM, native tools from cloud or virtualization vendors (note: these may not be open source depending on the vendor)

System Configuration

Each layer has its own tooling that can be used individually or as an integrated solution.

The role of this tool is to put OS instances into the correct state (according to an assigned specification) so that the system is ready to have application service instances deployed. The System Configuration tool will achieve this state by executing a variety of package updates and configuration changes. Once the initial desired state is reached, an essential function of this tool is to ensure that the system-level configuration remains compliant with its specification (and any deviations are automatically repaired). All system-level changes are made by editing the specifications provided to the System Configuration tool.

Example: Puppet, cfengine, SmartFrog, Chef, BCFG

Application Service Deployment

The role of this tool is to coordinate the deployment of application artifacts to their appropriate servers, generate any environment-specific application configurations, and then properly start-up the individual application services so that they run as an integrated business application. An Application Service Deployment tool is concerned with coordinating actions that have to take place across multiple tiers (web, application, database) and multiple servers. Like the System Configuration Tool, this tool must also be specification driven. Changes are driven through the tool by editing the specifications for a particular integrated business application and then rerunning the same consistent deployment automation commands.

Example: ControlTier, Capistrano, Fabric, Func

Tools are easy, people are difficult

Implementing a fully automated provisioning solution is simpler than it looks -- provided that you pick the right tools and make the necessary cultural changes throughout your organization. In practice, it is more often the cultural changes that trip up an organization than the tool implementation.

If you don't already have a strong program in place that prescribes, measures, and enforces a culture of operational excellence, we highly recommend that you make that a priority.

Methodologies like Visible Ops (http://www.itpi.org/home/visibleops.php) are a straightforward way to get the culture change started.

The different layers of the automated provisioning toolchain often match up to an organization's labor divisions. For example, the infrastructure focused roles may be primarily concerned with the Operating System Install/Imaging and System Configuration tools. The application or service management focused roles may be primarily concerned with the Application Service Deployment tools. Each layer of tools must be able to function both independently and as an integrated provisioning system.

Example Implementation (Online Retailer)

A major online retailer of clothing and equipment recently engaged both Reductive Labs (the primary sponsors of the Puppet project) and DTO Solutions. (the primary sponsors of the ControlTier project) for the purpose of improving the efficiency and reliability of their online operations.

Like most successful online retailers who have grown quickly, this retailer's web operations were feeling the strain. Critical problems included:

Like most successful online retailers who have grown quickly, this retailer's web operations were feeling the strain.

- Difficulty meeting seasonal capacity requirements and bringing new environments online
- Key engineers were constantly tied up "firefighting" deployment problems. This created both a financial drain and a tactical orientation that consistently diverted attention away from strategic work that would deliver higher business value.
- Outage incidents were common and there was a low success rate in delivering system or application updates.

DTO Solutions and Reductive Labs helped the retailer analyze these problems and identified the following root causes:

1. Servers were being built on a case-by-case basis and weren't being built to strictly defined profiles. System configuration and application deployment was mostly manual. The automation that did exist was in the form of scripts that required significant intervention before and after most runs.
2. Configuration drift was very quick to occur. Systems quickly diverged from each other as routine systems administration and deployment activity occurred. Further complicating matters, there was no formal distinction between application configuration and system configuration.
3. Application provisioning and updating was done on a file-by-file basis with little formal packaging. Backing out changes was very difficult and there was no reliable way to tell which files came out of source control from development and which were files that weren't under source control and came from other groups.
4. There was no mechanism for centrally managing configurations and no way to manage application services as logical or physical groups. Most configuration management or deployment activity had to be done on a server-by-server basis with a considerable amount of trial and error.

The first phase established an automated provisioning system built from all open source tools.

DTO Solutions and Reductive Labs are working with the retailer on a phased approached to improving the retailer's operational capabilities. The first phase established an automated provisioning system built from all open source tools. That phase is the focus of this discussion.

Building on our shared philosophy of "using the right tool for each job", a strategy was adopted that uses distinct, yet complementary, management strategies for system deployment/configuration and application deployment/configuration. Each sub-solution can be used individually to achieve specific goals for specific layers of the infrastructure or they can be used an integrated solution that provisions from bare-metal (or a basic virtual machine image) all the way to running business applications.

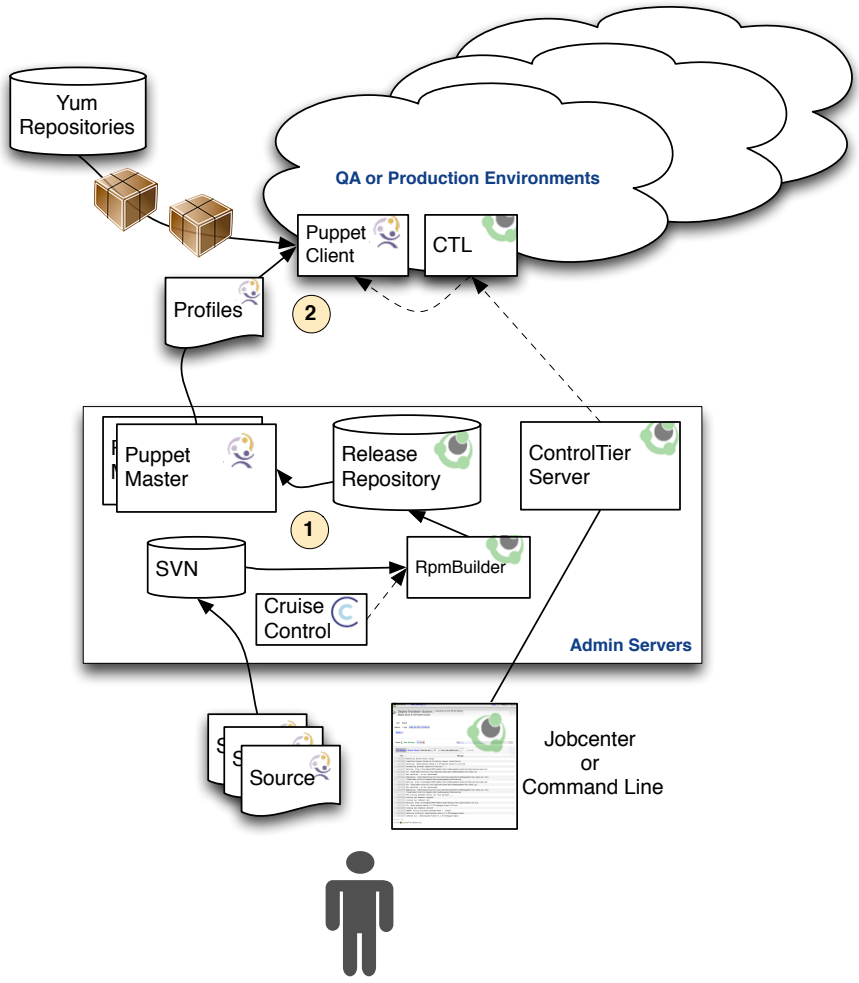
In addition to efficiency and reliability gains, the combined solution also reduces the maintenance overhead that would come with developing and maintaining a custom provisioning system. This solution uses mostly out-of-the-box Puppet and ControlTier modules that have been assembled into a system that is tailor-made for the requirements of this retailer's infrastructure.

Together, these open source tools reduce the retailer's burden of provisioning environments from hours of intense hands-on configuration to minutes of attending to executing automation.

Operating System Install/Imaging and Systems Configuration Implementation

Kickstart (OS install) and Yum (system package management) were already in significant use. Puppet was quickly implemented to provide centralized systems configuration management. For quality and version control, all Puppet profiles are centrally managed and stored in source control. Specific profiles are delivered by the Puppet Masters to the appropriate servers and each server is built and configured according to that server's specific purpose. Kickstart, Yum, and Puppet were combined into an integrated solution. This means that the Retailer's staff no longer needs to run a long series of scripts and manual steps in order to build a server. Now they only have to know which server profile they want and fire off a single command.

Now they only have to know which server profile they want and fire off a single command.



A typical system-level change management scenario, as illustrated in the diagram above:

1. Puppet module source is checked into Subversion. CruiseControl automatically invokes a ControlTier builder to create RPM packages of the Puppet module source and put those packages into the release repository. When they are ready, the retailer's administrators then run a ControlTier deployment command that installs the RPM of Puppet modules on each target environment's Puppet Master.
2. Each of the retailer's Puppet clients are set to check for updates from their respective Puppet Master at 30 minute intervals. If the retailer's administrators want to tell the Puppet clients to check for updates immediately, they can use the ControlTier infrastructure to dispatch a command to each Puppet Client to tell it to refresh itself. Once the Puppet Client receives the updates from the Puppet Master, the Puppet Client executes the necessary actions to bring the server into compliance with the new specification.

Application Service Deployment Implementation

A package-centric and specification-driven deployment methodology was put into place

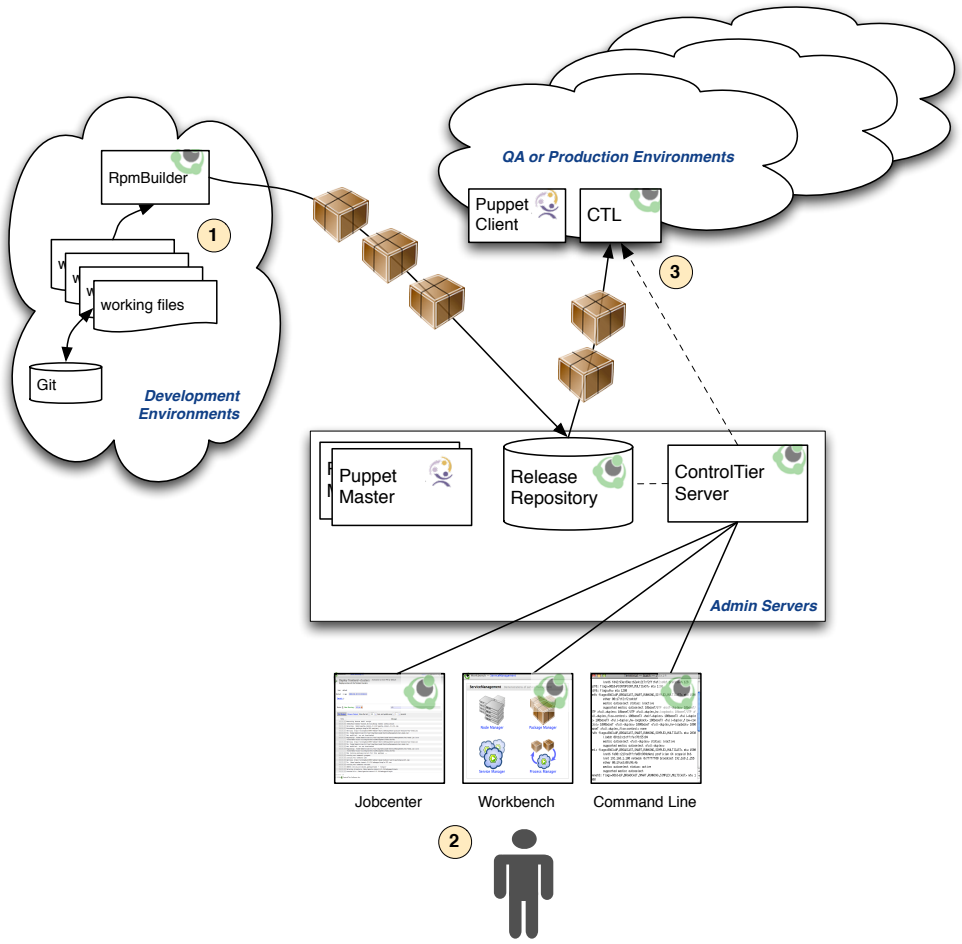
The first issue that had to be overcome was the lack of formal application packaging throughout the retailer's operations. A package-centric methodology was put into place that ensured that all application artifacts were formally packaged in the RPM format (enabling traceability, version control, simplified distribution, rollback, and easier configuration management). All application related packages are registered with the retailer's ControlTier repository (by design, system packages remained managed by Yum, and both sets of packages are installed to the standard system RPM database).

Once the packaging issues were straightened out, the rest of the application service deployment solution was a straight-forward ControlTier implementation leveraging ControlTier's standard open source automation libraries. To enable end-to-end automation, site specific workflows were implemented (again building off of pre-built automation libraries).

Application configurations were also setup to be automatically generated from templates at deployment time, (removing the need for any hacking by hand). Self-service interfaces were put into place using ControlTier's Jobcenter web application to enable the future delegation of push-button deployment capabilities to staff of any skill level (controlled by role-based job-level authorization authenticated against LDAP or Active Directory).

Similar to how the Kickstart manifests and Puppet profiles provide a complete specification of the Retailer's system level infrastructure, ControlTier contains the complete specification for assembling and controlling each instance of their application services (either from an individual service or integrated application point of view). In addition to driving the automated provisioning processes, these specifications can be shared across different teams during troubleshooting efforts.

ControlTier contains the complete specification for assembling and controlling each instance of their application services



A typical application-level change management scenario, as illustrated in the diagram above:

1. Developers work in their own environments to develop and test application changes. Developers use a ControlTier builder to create RPMs and upload them to the centralized release repository. Using the same mechanism, Build Engineers create formal releases from merged updates managed by the source code control system (GIT in this case)
2. Administrators use the ControlTier tools to select which release versions they want to deploy and specify any environment specific application configuration settings.

CTL clients coordinate the update process

3. The ControlTier server dispatches update commands to the appropriate CTL clients. The CTL clients coordinate the update process, including installing the appropriate application packages, generating the correct configurations, and coordinating the restart process. Related content updates are also managed by making calls to the appropriate tools (NAS snapshot copies, BitTorrent replicated documented root, rsync, etc..)

Stay tuned for more papers in this Web Ops 2.0 series