Assembling Web Development Environments with Perforce

Abstract

Web development assembles a broad range of professionals, project techniques, employment roles, and levels of technical expertise, and puts them together under all of the pressures of traditional software development and advertising with even more stringent deadlines. In addition, the incredible labor demand in technology has often lowered the bar for the skills of people working on Web projects. The result is a complex medley of artistic and technical, experienced and inexperienced requiring rapid development and application of a robust yet flexible process that takes all contributors into account.

This paper will attempt to untangle many of the decisions with specific implementation techniques for a wide range of situations, including branching best practices, integrating non-technical participants, and managing sub-contractor relations, all wrapped into a configuration-managed environment using Perforce.

The Case for SCM in Web Projects

Software configuration management (SCM) plays a key role in Web development. For those engaged in large-scale e-commerce initiatives, this is an obvious statement. Software binds business needs to hardware, and should be properly managed. Corporate application development has understood this for some time..

However, the proliferation of the Web has exposed non-software practitioners of different disciplines to this arena. Some of these practitioners think of a Web page as a flexible communications medium, rather than a textual encoding of formatted text and graphics. Some think of it as a way to shop, rather than a collection of code to implement business rules.

Similarly, non-software practitioners have different perspectives on the basic aspects of configuration management. Some think of versions as the pile of papers left over from brainstorming the last advertising campaign. Others think of concurrent development as selecting the best work at the end of the day or as a new work composed of an amalgam of previous ideas. And of course, branches are something found in a forest.

A good SCM system, or possibly its close cousin the Web content management (WCM) system,[1] allows an organization to effectively manage its Web development. As even the

[1] WCM systems on the market today tend to approach the content management problem from the perspective of marketing collateral and content assets. They make provisions for source assets, but are

most common Web sites increase in sophistication, two trends appear, the use of ready-made components on one hand, and a greater reliance on custom solutions on the other. The problems posed by the latter motivate this paper.

One Web Server, Many Developers

The distributed and dynamic natures of Web development introduce many issues that are not apparent in traditional software development. A basic SCM tool addresses the concepts of concurrent development and isolated workspaces.

However, the site must be delivered through a Web server to be seen as the intended audience will see it. Viewing files from the local disk suffices for the limited category of static content, but few useful Web sites consist of strictly static content. The use of server-side technologies, including CGI programs, simple server-side includes, PHP, some uses of XML, ASP, and the Java Enterprise technologies Java Servlets, Java Server Pages and Enterprise JavaBeans, has become a matter of course for even modest sites. Each of these requires a server to render the result of its interactions.

The use of branching in Web development suggests the relevance of multiple views of the site. Add the prospect of multiple concurrent projects applying their changes to the same branch and add concurrent QA and release activities, and the potential complexity explodes.

The most obvious solution allocates a server for each individual or role on the project. If the IT environment dictates Windows for e-mail, groupware and document sharing, as is common, then the cost of a server per developer escalates depending on the target platform for the Web servers. For IIS, running a copy on the developer's workstation is effective. For servers on another x86-based platform such as Linux of FreeBSD, products like VMWare or Win4Lin allow either development or Windows to run in a virtual machine, with one set of hardware per developer. If the server platform is Solaris, the options are drastically reduced.

This paper approaches the integration of Web development with SCM from a perspective of frugality, particularly appropriate for the current business climate but useful at all times. The solutions presented assume a minimal allocation of money for development hardware and software licenses. The configurations presented can jumpstart a bootstrapped company or optimize one that is well endowed. Specifically, the configurations use the Apache server, which runs on a number of platforms, including most Unix and Windows. Much, but not all, of the functionality discussed is present in or available for other Web servers.

generally not as sophisticated in their treatment of the software aspects or of concurrent development issues.

Virtual Hosts

Virtual hosts allow multiple Web servers on minimal hardware. The Web server is configured to distinguish between virtual hosts using either the IP address or the host name associated with the request.

Name-based virtual hosts are preferred over IP-based virtual hosts. There are three primary arguments against IP-based virtual hosts:

1. They consume IP addresses, a real problem when depending on officially allocated IP addresses. The increasing scarcity of public IP addresses is motivating the migration to IPv6 with its increased address space. Private IP addresses also alleviate this resource availability problem.
2. Adding and deleting IP addresses or, even worse NIC cards, requires root privileges and possibly reboots.
3. Referencing the virtual host by name requires DNS reconfiguration. Dynamic DNS can reduce this trouble, but it is an inelegant solution and heightens the network security risks.[2]

Name-based virtual hosting solves or reduces all of these problems. The Web server distinguishes between virtual servers based on the domain name of the request using the HTTP "Host:" header. For example, when a browser requests http://www.vance.com/, the HTTP request starts with two lines like:

```
GET / HTTP/1.1
Host: www.vance.com
```
[3]

Many domain names can map to a single IP address, solving the consumption problem in a broad stroke. This involves modifying the Web server's configuration, which can require administrative privileges. However, naming tricks and various Apache modules, to be discussed, can solve this seamlessly. Judicious use of DNS wildcards eliminates the need to reconfigure DNS as sites are added and removed.

The following shows an example configuration for name-based virtual hosts,

[2] The use of an administrative tool for user directed operations is considered inelegant. Dynamic DNS has relatively weak security. Only recent versions of BIND have cryptographic security. Previously, security was by network address. Additionally, dynamic updates can only be restricted in very limited ways, allowing dynamic update access to potentially leave the entire domain open to change.

[3] HTTP 1.1 requires either an `absoluteURI` (e.g. http://www.vance.com/index.html) in the request or a `Host` header. All HTTP/1.1-compliant servers must accept the `absoluteURI` form, but requests to proxy servers are required to use it. `Host` was introduced in HTTP 1.0, but was not required. All current browsers supply `Host`, including Netscape 4 and higher, Internet Explorer 4 and higher, Opera and Lynx. See RFC2616 "Hypertext Transfer Protocol – HTTP/1.1" at ftp://ftp.isi.edu/in-notes/rfc2616.txt for complete details.

DNS Configuration (vance.com zone file)

```
*.dev.vance.com CNAME 86400 perforce.vance.com.
```

Apache Configuration (httpd.conf)

```
NameVirtualHost *
<VirtualHost *>
     DocumentRoot /usr/local/apache/htdocs
     ServerName www.vance.com
</VirtualHost>
<VirtualHost *>
     DocumentRoot /usr/local/dev/site1/htdocs
     ServerName site1.dev.vance.com
</VirtualHost>
```

The following cautions apply to this example:

- The DNS wildcard should only fall in a subdomain that will never have individual name configurations within it. Any such names may disable the wildcard altogether.[4]
- The first virtual host becomes the default Web site. Any name that is not explicitly configured as a virtual host with a ServerName resolves to the default host. If the configuration outside of the VirtualHost blocks was previously being used, the DocumentRoot must be defined within the default VirtualHost block for a comparable effect.
- The '*' in the NameVirtualHost and VirtualHost directives assumes that all IP addresses on the machine should be serviced equally. Certain virtual hosts can be restricted to certain IP addresses, if desired. For details and a wide range of other possibilities, see the Apache Virtual Host documentation.

As a best practice, make the sub-domain the same as the client spec name. Because DNS restricts domain name components to letters, numbers and hyphens, the client spec names are necessarily limited likewise. Also, use the same name for the final path component. For example, the development Web site site1.dev.vance.com would map to the client 'site1' whose client root is /usr/local/src/dev/site1 and whose document root is /usr/local/src/dev/site1/htdocs.

Permissions should also be mentioned at this point. By default, Apache runs as user *nobody* and group *nobody* or, on Red Hat Linux, *apache.apache*. The default user is deliberately chosen to have limited permissions. relative to the other configured users and groups. In order for Apache to display the content, it should have at least read permission to the pages. CGI programs may also require executable permission.

[4] See the O'Reilly DNS and Bind book. Information on wildcard records is on pp. 377-8. Particularly, the note about wildcard entries as it pertains to MX records understates the issue. The noted pre-emption of wildcards applies when *any* specific reference in the same subdomain covered by the wildcard resolves to an address.

Superimposing a Web site on a client workspace requires adjustments to the default permissions to work properly. Although the details of permissions are left to the system administrator, suggestions to consider when solving this problem are changing developer's or the default umask, creating a group that developers and Apache share in common on the development machine, or, if running an Apache server for each developer, run the server with the developer's user ID.

Simplifying the Configuration

The basic virtual host configuration works well, except that each time a developer needs a new client, a privileged user has to manually alter the Apache configuration and restart the server.[5] The burden can rest on a small number of users, or everyone can have the required privileges. The former is too intrusive on an already aggressively scheduled staff. The latter can be a problem, depending on administrative policies and security concerns.

A good solution to this dilemma applies the techniques from the Apache document "Dynamically Configured Mass Virtual Hosting." These techniques use either mod_vhost_alias or mod_rewrite to dynamically map path components based on the domain name supplied in the host header or based on a mapping from a lookup table in a file. However, this approach conflicts with some of the later techniques for integrating Perforce contributors.

Integrating Non-Perforce Users

So far, the proposed techniques allow multiple workspaces, each with its own Web server drawing content from the local disk. This works well for developers comfortable working in the environment on which the server runs. However, the variety of skills brought to bear on a Web site also introduces professionals who may not be comfortable with, for example, a Unix command line.

Strategic use of aliases can soften the impact of a command environment for Web content viewing, but then every submission becomes a multi-step process of submit, sync the server client and view the content. The extra step can make the difference between a happy contributor and a grudging participant working around the process out of convenience.

If the contributors in question are willing and able to use Perforce on their local machines, disk-sharing technologies like Samba, DAVE or NFS can bridge the gap. But

[5] On a side note, using 'apachectl restart' will send the HUP signal to Apache. This will cause each process to end immediately, regardless of whether it is currently handling a transaction. Red Hat uses /etc/rc.d/init.d/httpd. The 'restart' argument there will completely kill Apache and start it fresh. The 'reload' argument is equivalent to 'apachectl restart'. None of these approaches is very graceful for a shared Web server. A more elegant and less intrusive approach is to use 'apachectl graceful' or to kill the Apache parent process with the USR1 signal. This allows in-progress transactions to finish before restarting their process.

licensing fees, problems with CR-LF translation and case sensitivity, and, to a lesser extent, increased local network traffic can discourage this approach.

Additionally, not all platforms have easily accessible GUIs or even command environments. For example, until recently the only available Macintosh clients for Perforce required MPW, CodeWarrior or OS X, none of which is available or appealing in this case.

Managing and Viewing Web Content: P4Web

Last year, Perforce released an application called P4Web that provides a Web-based GUI to Perforce. In and of itself, P4Web has great value by providing a Perforce GUI on platforms for which one was previously unavailable, most notably Unix and Macintosh.[6] Refer to the P4Web documentation for installation and configuration instructions, as there are some platform-specific dependencies.

The most important feature of P4Web for Web development is that the client workspace can reside on a different host from the browser. This allows a contributor to control a virtual host remotely, eliminating the need for command line interaction. Note that if P4Web needs to be used remotely, it should be started at system startup.

The second important feature for Web development is the ability to view content directly from the depot and have "back-in-time" browsing. This allows direct and immediate viewing of static Web content directly from the Perforce depot. Unfortunately, content involving server-side processing requires the use of a Web server. This is not usually an issue for static content developers focusing on copy and presentation.

Managing Sub-Contractor Contributions

Employing sub-contractors on Web development projects effectively complements the skills of the development team without carrying the overhead of specialized skills. A marketing communications-oriented team typically sub-contracts highly technical development assistance. An application-oriented team commonly sub-contracts graphical design and marketing communications work. Many combinations are possible, depending on the required skills for the project. Given the application-oriented motivations for this paper, the discussion will focus on the management of non-technical contractors.

Many graphical design and marketing communications firms are entirely Windows- and/or Macintosh-based. Although most are accustomed to keeping archives of client deliverables, the concepts of full-blown configuration management are foreign to them. Requiring sub-contractor teams to learn a new tool and knowledge domain to fulfill the contract will usually meet with justifiable resistance. Additionally, internal security policies may prohibit broad access to network facilities by outside contractors.

[6] For another Unix GUI, see the tkp4 project.

Since most Web design tools support ftp to populate a Web site, simply setting the document root on the virtual host as the ftp drop point for the contractor can be very effective. The document root falls under the client root for a client workspace, with the "allwrite" option set to allow them to put files unimpeded.

Once the contractor provides notification of an update to their content, the techniques from Perforce Tech Note #2 can be used to scan the tree for changes. Watch for CR-LF changes that cause spurious diffs and for obsolete Web elements that were not deleted by the contractor.

This increases the administrative load for the project team a little, but is much better than leaving the content unmanaged. Isolating the contractor's work to its own branch eases the administration, as well. Otherwise, delays when reacting to content updates create merging difficulties when bringing the changes back into Perforce.

Improving Sub-Contractor Contribution: P4FTP

P4FTP emulates an ftp server, providing access to a client workspace using the ftp protocol. It directly targets the Web development community. P4FTP setup and configuration are straightforward and are detailed in the "P4FTP User's Guide." P4FTP treats each ftp session as a single changelist, eliminating the need to react immediately and to scan the client workspace when notified of new content. The contractor's work still should be isolated to a branch and policed for correctness, but that is simply good SCM and good management. Isolating the work to a branch also ensures that merging issues do not impede the contractor's submissions.

P4FTP has two drawbacks under some circumstances. The first is that the content only transiently populates the disk space of the Web server, so it does not suffice for viewing content on the server with the client superimposed on a virtual host. Since most sub-contractors test with their own Web servers, this is not a major issue for sub-contractor relations, except possibly for dynamic content.

Due to the fixed relationship between the user ID and the P4FTP client name, P4FTP requires a Perforce license even for the occasional contributor. Even worse, attempting to consolidate multiple contractors under one license forces them to share a client.

Providing Content in Virtual Hosts: WebKeeper

Each of the discussed methods for integrating non-Perforce users, except for P4Web, only addresses the issue of how content gets into the depot. P4Web provides direct viewing of content within the depot, but not through the production server. This is appropriate for static content because it is simply a matter of sending the file to the browser, which does all of the formatting. Even apparently dynamic JavaScript is presented to the browser and treated as simply another file by the server. However, true emulation of the production server, particularly with non-static content, requires more.

Perforce has provided an Apache module called WebKeeper to simplify access to Perforce-managed Web content. Using WebKeeper, a Web site can map directly to a portion of the Perforce depot. WebKeeper is available in the Perforce public depot at //public/perforce/webkeeper/….

WebKeeper provides views of Web content in the depot without having to login to a Unix machine and sync a Perforce client workspace. This can work in cooperation with virtual host mechanisms to provide a significant boost in usability for any content author working on a machine other than the Web server. WebKeeper can not be used to view work in progress because it is checked out in a client workspace, but this can be mitigated through judicious use of branches.

Here is a simple virtual host configuration with WebKeeper,

```
<VirtualHost *>
  ServerName site1.dev.vance.com
  <IfModule mod_webkeep.c>
    WebKeepPort perforce.vance.com:1666
    WebKeepUser steve
    WebKeepDirectoryIndex index.html
    WebKeepAlias / //depot/dev/steve/htdocs/
  </IfModule>
</VirtualHost>
```

Note that the `DocumentRoot` directive has been replaced by a series of `WebKeep*` directives. These directives define the Perforce port and user, and the mapping from the Web namespace to the depot. Three other WebKeeper directives are available, `WebKeepClient`, `WebKeepPasswd` and `WebKeepSync`. They are explained in more detail below.

The `WebKeepDirectoryIndex` directive tells WebKeeper to map requests for directories to the file index.html in the requested directory. The directive takes an arbitrary number of file names. The names will be used in order of precedence. This list of file names should correspond to the names in the normal Apache `DirectoryIndex` directive for consistency.

The second directive indicates that the entire sub-tree maps one-to-one to the depot sub-tree //depot/dev/steve/htdocs/. In this example, a request for /mypage.html will map to //depot/dev/steve/htdocs/mypage.html.

As a final note on the example, the `IfModule` directives make the configuration portable. `IfModule` only allows its content to be interpreted if the specified module is available. If the configuration were transferred to an Apache server that did not have WebKeeper installed, the server would refuse to start without the `IfModule` tags. With these tags the server would start, although the WebKeeper virtual host would not work as expected.

The `WebKeepClient` directive, if present, takes the name of a client as its single argument. In this configuration, the client plays no role in displaying the file. However, if a client is specified, the `WebKeepAlias` mappings can use the client name space, which may result in clearer configurations.

Installing WebKeeper

The latest version of WebKeeper builds using the Apache 1.3 APACI build structure. Set P4PORT to public.perforce.com:1666 and client view to map on the depot side from "//public/perforce/webkeeper/…". Directions for building WebKeeper with Apache are located in the README.WEBKEEP file. This version of WebKeeper builds a static Apache server that includes WebKeeper in its built-in modules. Execute the resulting server with 'httpd –l' to report its modules for verification.

With a pre-packaged binary distribution of Apache, trying to replicate the Apache configuration when rebuilding may be undesirable. In this case, WebKeeper also builds as a loadable module. This allows the existing server to load WebKeeper without having to reverse engineer and replicate the build configuration. To do this, build the module by building the entire Apache tree and taking the results, copy libwebkeep.so[7] into the installed modules directory (commonly <apache root>/libexec), and adding the following directives to httpd.conf,

```
LoadModule webkeep_module libexec/libwebkeep.so
AddModule mod_webkeep.c
```

For several versions, Red Hat Linux has had a method of safely including module directives in the Apache configuration files. The problem is that Apache will refuse to start if there are `LoadModule` directives for modules that are not available.[8] A typical Red Hat configuration to load WebKeeper dynamically and safely will look like

```
<IfDefine HAVE_WEBKEEP>
LoadModule webkeep_module modules/libwebkeep.so
</IfDefine>
…
<IfDefine HAVE_WEBKEEP>
AddModule mod_webkeep.c
</IfDefine>
```

The `HAVE_*` symbols are defined by the Red Hat Apache start script /etc/rc.d/init.d/httpd based on the contents of the modules directory. Therefore, the module will only be loaded if an appropriately named file is present.

[7] Or libwebkeep.sl on HP-UX or libwebkeep.dll on Windows.
[8] Note also that Red Hat puts the Apache modules under /etc/httpd/modules. This is really a symlink to /usr/lib/apache, but it puts the modules next to the configuration files and the symlink to the logs, which is convenient.

Addressing Content Types

Static content types

WebKeeper can be used in its simplest form for static HTML pages without incurring any overhead for local disk space. Since mod_mime uses the extension in the file name to identify the file type, almost all requests will be correctly typed in the response to the browser.

The HTTP protocol defines two common types of requests for page retrieval. The GET request is usually used for static HTML. GET requests can also be used for dynamic requests, but best practices suggest they only be used for requests that do not change state on the server. It should be noted that for the basic static content configuration, WebKeeper only handles GET requests. Since traditionally, POST requests invoked CGIs, and PUT and DELETE requests were heavily restricted because they modify local file systems, this is not a significant imposition for static content.

However, several types of non-static content have become available since WebKeeper was first developed. Restricting its use to only GET requests prevents the effectiveness of dynamic content types.

Dynamic content types

Dynamic content types fall into two categories for purposes of using WebKeeper. The first category consists of text-based literal or interpreted scripts, such as PHP, Perl scripts, ASP, JSP, server-processed XML and server-side includes.[9] The key characteristics of these technologies are that their source is presented to the Web server as a text file, and that the text file is automatically processed into some executable or presentation form through an embedded or external file processor.

PHP, Perl and ASP are programming languages that can be embedded in or wrapped around an HTML document with a special file extension. The extension is associated with the appropriate command processor, either embedded in Apache (e.g. mod_php, mod_perl) or external, to interpret the scripting instructions and output the final HTML. Server-side XML provides a method to separate content from its presentation, typically using a stylesheet to transform the XML data file to its presentation form in HTML. Server-side includes are a rudimentary conditional expression and include syntax handled by Apache's mod_include. JSP takes HTML augmented with special additional tags and hands the text file to a Java application server. The application server then automatically compiles the JSP file into a Java servlet, which executes to produce the HTML.

The second category consists of binary elements that require some form of compilation and/or deployment before being accessible through the Web server. This includes C or

[9] Perl and JSP are actually compiled on-the-fly, but still share the property that a textual form is presented to Apache.

C++ CGIs and other server-side Java technologies, such as EJBs and Servlets. Java applets, although they are served up to the browser like a piece of static content, fall into the second category if they are a product of the development environment and need to be compiled first. Java applets such as those downloaded from a public site can be handled as plain static content.

Text-based Dynamic Content

WebKeeper has recently been enhanced to address the first category of dynamic content. This occurs at the expense of syncing the client workspace used by the server. Here is an example of such a configuration,

```
<IfModule mod_webkeep.c>
  WebKeepPort  perforce:1666
  WebKeepUser  perforce_user
  WebKeepDirectoryIndex index.html
  WebKeepAlias /  //my_client/htdocs/
  WebKeepClient my_client
  WebKeepSync On
</IfModule>
```

For simplicity, this example does not include the virtual host context.

There are two new directives in this example. The first is `WebKeepClient`, which gives the client spec to use for the Web site. This was not important in the previous configuration, since the depot file was effectively being output from a ‘p4 print’ command internally and never being synced to the local disk. The use of depot name syntax to retrieve the file made the client name unimportant, but with it the `WebKeepAlias` directive can be simplified.

However, the addition of the `WebKeepSync` directive in its On state requires that a client be specified. The `WebKeepSync` directive instructs WebKeeper to sync content to the disk at the very beginning of the Apache content processing sequence. This allows the remainder of the Apache processing to treat the file as if it had existed on disk all along. It also allows various processing modules that only work with physical files to process the latest Perforce-managed content in the server.

Currently, there is a limitation to this approach in that only files requested through HTTP will be synced to disk. Some aspects of server-side includes, PHP includes or Perl imports may not work properly.

Text-based scripting languages that are run as CGIs still fall into this first category of dynamic content providers, but they require a slightly different configuration. Their interaction with mod_cgi requires that they also be designated as a script. WebKeeper itself does not do this, but some correspondence between WebKeeper aliases and the Apache directives that indicate scripts is required. The following shows this with a basic virtual host context.

```
<VirtualHost *>
    ServerAdmin webmaster@vance.com
    ServerName site1.dev.vance.com
    DocumentRoot /usr/local/dev/steve/htdocs
    ScriptAlias /cgi-bin /usr/local/dev/steve/cgi-bin
    <IfModule mod_webkeep.c>
        WebKeepPort perforce.vance.com:1666
        WebKeepUser steve
        WebKeepDirectoryIndex index.html
        WebKeepAlias /cgi-bin/ //depot/dev/steve/cgi-bin/
        WebKeepAlias / //depot/dev/steve/htdocs/
        WebKeepClient my_client
        WebKeepSync On
    </IfModule>
</VirtualHost>
```

There are several things to note in the example:

- A `DocumentRoot` directive has been re-introduced from the previous examples. Because of the reliance on standard Apache mechanisms like `ScriptAlias` that use paths relative to the document root, it must be specified.
- The `ScriptAlias` directive tells Apache to map requests for the /cgi-bin directory to the specified directory in the local file system, and that these requests should be handled by mod_cgi as executable scripts. This directive is defined by mod_alias. There is a corresponding `VirtualScriptAlias` for use with mod_vhost_alias.
- The `WebKeepAlias` for /cgi-bin/ tells WebKeeper to draw content for the /cgi-bin/ from the specified portion of the depot. This is the similar to the previous `WebKeepAlias` directives.
- The final note has to do with the order of the `WebKeepAlias` directives. WebKeeper processes `WebKeepAlias` directives in the order they are presented in the configuration file. If the mapping for /cgi-bin/ were to come after the mapping for /, the cgi-bin mapping would never be used and the content would never be synced.

Handling Compiled Elements

Another way of characterizing the second category of dynamic Web content is that the source that is modified in development is not directly touched in any way by the Web server. Three examples, each requiring slightly different handling, are compiled CGIs (e.g. written in C/C++), Java applets, and EJBs and Java servlets.

Before discussing how to handle compiled elements, it should be clear where and how they will be compiled. Two important details are that the elements may be platform-dependent, and that they need to end up on the Web server machine.

For example, Java is platform-independent, but a C++ CGI is platform-dependent. Also, Java that uses JNI is platform dependent because it calls into loadable shared libraries in a native compiled shared library. The architecture of a local machine is not important in determining the desired end result. Rather, it is the architecture of the Web or application server.

Consolidating the development environment onto the fewest machines possible motivates much of this discussion. A cost of this with respect to compiled elements is that it may require a method to copy the compiled elements onto the Web server. Fortunately, most developers generating and testing compiled elements are also comfortable working on the required server platform or, as possible with Java, ftp-ing the class or jar file to the server.

CGIs

Handling compiled CGIs is relatively straightforward. The configuration for script-based CGIs also works for compiled CGIs by omitting the `WebKeepAlias` that aliases cgi-bin. If the CGI directory mixes script-based and compiled elements, simply leave the `WebKeepAlias` directive and prepare for additional error log entries as WebKeeper tries to sync files that are not in the depot. In the build configuration, set the target directory of the build as the cgi-bin directory under the virtual host's directory tree.

Java Applets

Java applets are even simpler to manage than CGIs, since they do not require special Apache configuration directives to be served up properly. They only require copying the class or jar file to the correct location in the virtual host's directory tree. Note that the environment will copy after the build in almost all cases, rather than build directly into the location, due to the integration of packaging into the Java compilation rules. This will not always be the case, but is likely in most non-trivial Java development environments.

EJBs and Java Servlets

EJBs and servlets are J2EE technologies that run under a Java application server. Although the details of their SCM integration require deeper investigation, the techniques to address their use will resemble that for applets and CGIs with a greater level of complexity. Deployment of Java application server components carries with it several packaging steps, including compilation into class or jar files, authoring or modification of deployment descriptor files, and restarting the application server. Most application server development environments come with personal application servers to simulate the environment for testing, reducing the need for integrated external solutions.

Should Binaries Be Archived during Development?

The discussion of compiled elements focuses on the development process. Although checking in binaries is a feasible method of deploying them to the development server that works seamlessly with WebKeeper, it is an inefficient use of storage and can be cumbersome for the developer. Regardless of how cheap storage may be or become, binaries are stored in their entire form, rather than using an incremental difference format. Storing each revision for each developer throughout a development effort will consume excessive disk space and complicate merges or branch specs.

An alternative would be to store binaries with the +S file type, indicating that only the head revision should be stored. This increases the storage efficiency, but requires a branch for each developer's project to avoid cross-contamination of developer's sites. This is not a bad idea and is almost a requirement for virtually hosted development Web sites, but integration of these versions can complicate Web branching strategies. Also, having to edit, build and submit for each change is awkward.

In general, archiving binaries for development purposes is discouraged. Compiling on the server, using network disk sharing to put the compiled elements in the right place, or using a file transfer tool like ftp, scp, or rdist is typically recommended.

However, consider the case of the quality assurance tester or, further down the assembly line, the end user. How are compiled elements propagated to their Web sites? Some testers may have build skills, but some may not, depending on the organization. The end user site will be discussed in the Deployment section, but it can be similar to the QA treatment. Simply put, it can be useful to archive binary elements for nightly builds, for QA activities and for deployment purposes depending on the completeness of the build environment, the expected level of platform and development proficiency for QA personnel, and the chosen means of deployment.

For example, if the build environment requires special arguments or flags to be manually applied for a release build, and if the QA staff is not expected to have development or server platform skills, then something needs to bridge the gap between the developers and the QA activity. In this case archiving binaries for internal releases or having a designated release engineer may be the answer.

Deployment

Given the knowledge to develop and, at least partially, run QA on a Web project, it is logical to consider this infrastructure for deployment. Deploying a Web application is simpler than deploying a traditional client application in most ways. Similar to the old mainframe client-server configurations, updates to Web applications happen on a server or collection of servers. Some of the techniques proposed here also apply to the extreme situation of a collection of servers load-balanced with layer 4 switches or caching appliances and segregated into different roles in the application. For an example of large-scale deployment in a Perforce-managed environment, see the presentation by David Markley and Scott Money from Lycos, Inc. on "Web Content Management" from the Perforce User Conference 2000.

The most immediate method of deployment is to build a copy of the Web site locally, tar it up, transfer it to the production server, and then un-tar it. This is a familiar model for many Web site managers. However, it has several shortcomings. Depending on the size of the Web site, it may not be atomic, leading to transition problems for transactions in progress. Although tar ensures that current content is present, it does not ensure that stale content is removed. Tar carries with it the permissions from the original directory tree, which may not be correct for the target installation. In clusters of servers, the process must be repeated on each server, decreasing any semblance of atomicity. Unpacking the files over the existing files destroys the fall-back position in case of problems with the new content.

The atomicity problems are perhaps the easiest to solve using a technique analogous to double-buffering in computer graphics. Two directory trees are maintained at all times, one active and the other inactive. Regardless of how files are transferred to the production server, they should be unpackaged into the inactive directory tree, validated out of that tree, then switched active through configuration changes. The configuration changes will not take effect until the server is told to use them, which can be synchronized more closely across a collection of servers. This also provides for one level of fall-back should the deployment have problems.

The stale content issue can be addressed conventionally through complete removal of the Web tree before unpacking it. However, this adds extra steps to the deployment process, increasing the potential for complications and failure. Another possibility is to use Perforce to manage the contents of the site. A ‘p4 sync’ will add, change or remove files as appropriate according to the desired criteria (e.g. label, branch, date/time, changelist). This solution has caveats.

At this point it is reasonable to ask whether to use WebKeeper to serve the content directly. For small or lightly trafficked Web sites, it can be feasible to do so. However, there are several concerns with this approach.

The biggest concern from a failure point of view is that now content delivery is subject to the availability of the Perforce server in addition to the usual Web site availability conditions. The problem is not with Perforce itself, but with the infrastructure that makes Perforce accessible.

As fast as Perforce is, performance will also be a problem. The performance of networked systems will be bound by the slowest sequence of operations. Consider the following sequence from the customer’s browsing perspective:

1. DNS lookup for the Web server
2. Request content from the Web server
3. Web server looks up DNS of Perforce server
4. Web server syncs content from Perforce server
5. Web server processes content from local disk
6. Web server returns content to browser.

Steps 3 and 4 involve an additional DNS lookup and network transfer of the content, effectively doubling the overhead of the request. To make matters worse, if this happens on a static page with five images (which is light compared to the increasingly prominent image-based menu systems), the process happens five additional times. Each of these steps provides another opportunity for failure.

It is not always desirable for the latest content to automatically appear on the Web site. Especially for production purposes, deployment should be a conscious effort rather than incidental. WebKeeper always pulls the content from the head of the branch, leaving the exposure of the latest content to the checked in state of the release branch.

When determining whether to use WebKeeper in the production system, some factors should be evaluated. Being able to sync a Web site fully requires that all elements of the site be present in the Perforce depot. This includes binary elements, possibly compiled. One approach would be to sync the sources from which the compiled elements are built and then perform the build on the production server. However, this conflicts with the system administrator's desire to keep the production servers as lean as possible for maintenance simplicity and for security reasons. The possible addition of compilers, build tools or special shells may meet with resistance.

An approach that builds and checks in the binaries on the release branch so that they can be synced out to the production servers is recommended. Having binaries on QA and release branches is a reasonable compromise and in fact increases the explicit understanding of the versions being tested and deployed. There are more detailed arguments on this point available by perusing various SCM mailing lists.

Branching for Web Content Management

Web development can differ significantly from traditional software development. The most noteworthy difference is that a Web site is usually a continuously evolving entity, as opposed to a cleanly versioned package. This life cycle calls for a different branching structure to meet the needs of the environment.

If the Web development follows the pattern of a traditional software development project, characterized by large functional change sets released at well-defined intervals as binary deployments, consider a more traditional branching strategy.[10] This section focuses on the branching model for a rapid-response, continuous evolution, incremental release model. It may be well planned and have some sizeable functional change sets, but it is more evolutionary and modular than traditional software models.[11]

As always, the branching structure should serve to manage the risks associated with product development. A common branching strategy for Web development is a two or

[10] The authors paper "Advanced SCM Branching Strategies" provides a framework for identifying needs and planning a branch strategy.
[11] This characterization by no means divides the world of software development into two distinct categories. If your Web development is very rigorously structured with regular discontinuous leaps in feature content, then you probably have already determined the best strategy for your situation. Most Web development is reasonably well served by the strategy proposed.

three-tiered approach. As a quick summary, development happens on the mainline, which has a sub-branch for QA, which in turn has a sub-branch for release. It can be insightful to think of each tier in this structure as a synchronization buffer in the release pipeline, similar to the buffers used in project planning or in a manufacturing process.

The mainline centralizes coordination of development. Child branches may and should be used to isolate development projects, but these projects integrate back to the mainline on completion. Depending on the desired policies, small changes may be allowed directly on the mainline. Cursory QA activities could occur based on the mainline content to provide ongoing corrective guidance to work in progress.

The QA branch has several functions beyond simply hosting the QA activity, but they all relate to how a pre-release Web site is composed, exercised and fixed. Ideally, the pre-release version is composed from all changes that have been submitted to the mainline as of a chosen date and time.

It happens from time to time that a release candidate needs to be composed before all work in progress completes, particularly if development is allowed to occur directly on the mainline. This forces composition of the release candidate by integrating from selective changelists on the mainline. An equivalent capability can not be reliably implemented with labels, due to the possibility of multiple changelists affecting the same files.

The QA branch also allows fixes to be applied directly to the context in which their defect is found. Corrective changes are applied directly on the QA branch and later integrated back to the mainline. This avoids the possibility that contextual changes on the mainline would complicate integrating the fix to the QA branch. It also eliminates this as a source of avoidable selective integrations.

The release branch can also serve several purposes, relating to the release and deployment of a pre-release Web site. The first function is the obvious purpose of encapsulating the release configuration for the site.

Although selective release based on changelists is usually a bad idea from a release engineering perspective, selective integration based on independently releasable components has merit for incremental rollouts. For example, releasing the revised FAQs or the updated "Team" page with the new VP's picture may be easy and immediate, while testing of the new shopping cart application likely will not.

The release branch can also be used to apply necessary transformations from the release candidate before deployment. This is risky from a release engineering perspective, but may be necessary based on a real world environment.

The release branch also serves as the master from which deployment occurs. If deployment happens with Perforce sync's, it may be easier and less error prone to simply sync from the head revision than from a label. Depending on the level of automation involved in deployment, risks can include people forgetting to update labels or naming them incorrectly.

Under some conditions, the release branch can be optimized away to a label on the QA branch. Specifically, if the various purposes of the release branch are not required in the environment, consider using a two-tiered structure and labeling the QA branch to denote a release. Even when using a release branch, it is best to label the versions that are pushed live for rollback and recovery purposes.

With this structure, most integrations will be simple. Selective integration from the mainline to the QA branch holds the highest risk. Content binaries (gif, jpeg, png, pdf) will be resolved with `-at`, since the new version is the one intended for production. The more advanced Web development tools like DreamWeaver and GoLive preserve the text format of HTML closely enough for meaningful diffs. The least advanced, writing HTML by hand, has the same property.

In the discussion on archiving binaries, it was noted that it could be useful to check in binaries of compiled elements for QA and release purposes. This raises the issue of whether compiled binaries checked in for QA purposes should be integrated to the release branch like static binary content. In the presence of a good build environment, the answer to this question is a resounding, “No!”

Integrating compiled binaries introduces the risk that the released source code and the release binaries are not in sync, casting doubt on the future ability to rebuild the deployed application. Additionally, it makes the release process slightly different from the QA build process by removing the need to perform the compilation on the release branch, weakening the validity of the QA process.

An alternative approach would only integrate the binaries and not the source. This still leaves the potential that the release is not purely reproducible, although that is minimized by avoiding parallel integration of otherwise related material. The benefit of this approach is that the release branch is exclusively a deployment package. Furthermore, the exact set of binaries that were testing in QA is deployed, avoiding the possibility of differences in the build.

Although rebuilding the compiled binaries on the release branch from their sources is recommended, that method is based on the assumption of a low-risk, repeatable build system. If the repeatability of the build is considered a significant risk, or if the need to deploy exactly the system that was tested is required, consider integrating the binaries.

Conclusion

Combining Perforce and its supporting programs with open source software and a collection of best practices provides a powerful combination of tools to effectively and efficiently manage all aspects of Web development. Although most decisions depend on the needs of the organization, this paper has attempted to catalog and clarify the issues surrounding these decisions.

Several enhancements to WebKeeper, such as authentication, mass virtual hosting support and provisions to refresh included content, would significantly improve security, hands-off administration, and completeness. The complete list of proposed WebKeeper enhancements is documented on the WebKeeper Project home page.

Full applicability to large-scale e-business development requires deeper investigation into application server integration. The lion's share of application servers rely on the Java Enterprise APIs, making Java server integration a prominent concern. Non-Java application servers such as Zope also warrant investigation.

Acknowledgements

Special thanks go to those who reviewed drafts of this paper: Gavin Matthews, Sandy Currier, Robert Orenstein, Laura Wingerd, Brett Taylor and my wife, Ellena. A great thank you also goes to Perforce Software for producing a SCM solution that can be so well integrated with Web development and for their support and encouragement in my work on WebKeeper.

Appendix A: Sharing (or not) Non-Perforce Content

Although not strictly a Perforce or WebKeeper topic, when committing to a Perforce-based Web development environment, some dark corners require illumination.

There will come a time, very quickly for most, when the Web site and its application content are not the only resources involved. The most common resource is the database, but others could include directory servers, middleware brokers, DNS servers, or Web caches. At this point it becomes essential to differentiate the production resources from the internal resources. It may even become necessary to segregate the internal resources into multiple categories, such as development and QA.

There are several implementation methods to achieve the desired result. All operate on the same principal of setting a key value at some point in the build process or execution environment, and letting the applications make decisions based on that key.

One example would be to define a compile-time flag that changed the value of certain resource names. This will work, but the resources have to be configured on a per-build basis, usually leaving room for error in release builds or per-developer builds. The thought of this makes build engineers cringe

A more flexible approach would set an environment variable with a well-known name to different values corresponding to the development or QA versions of the desired resources. As with most things, there is an Apache module to set variables in the execution environment, mod_env. The SetEnv directive can be used to set an environment variable that will be visible in CGIs and other executable contexts.[12]

As an example, consider the presence or lack thereof of the following Apache directive, which can be applied within the scope of a virtual host.

```
SetEnv DBNAME devdb
```

The following C code would branch appropriately:

```
char *dbname = getenv( “DBNAME” );
if ( dbname == NULL ) dbname = “productiondb”;
open_database( dbname );
```

Although the open_database() call is obviously fictitious, the intent of the example is clear. It also raises another question that needs to be answered for each environment and perhaps each resource, how the system should behave in the absence of the significant keys.

[12] More sophisticated manipulation, including conditional variables, can be set with the directives from mod_setenvif. See Apache documentation for more details.

This example chooses to default to the production resource. This supports a configurationless deployment, but runs the risk of developers accidentally playing with production data. If the default points to a non-production resource, deployment requires additional configuration and it is unclear what the default resource should be if per-developer resources are warranted. In most cases, the safest behavior causes an error in the absence of the significant keys.

Appendix B: Managing Apache Configurations

The biggest weakness with the Virtual Host approach to isolated development environments is the privilege needed to reconfigure Apache. Reconfiguring Apache involves editing the httpd.conf (or the srm.conf or access.conf, but these are deprecated) to add, modify or remove the directives to create, configure and delete virtual hosts. Once the editing is completed, run the command 'apachectl restart'. Unfortunately, editing the file usually requires special permissions, and restarting Apache requires that either the same user that starts Apache or root. If the configuration has a mistake, the entire server dies for all virtual hosts. The use of 'apachectl graceful' will preserve transactions in progress for other developers' sites.

Several solutions are worth considering. None of them is ideal. Some are usable now, some should be promising in the near future, and some are restricted in their applicability. The most practical solution is to run a different server for each developer. This server should run under the developer's user ID, so that the developer can restart Apache without special privileges. This also prevents one developer from taking down other development and possibly non-development Web sites through configuration errors. Because this solution uses non-root IDs, the servers will have to use port numbers greater than 1024 and will each have to use a different port number. Additionally, content developers that are not comfortable with a command line environment will be either at a disadvantage or dependent upon others for their configuration. This last issue could be resolved with some relatively simple Web interface development.

The remaining possibilities all involve various public domain graphical configuration tools for Apache. None of these tools has all of the desired characteristics of being remotely hosted, feature-complete, and end-user targeted.

Comanche is feature-rich, but not remotely hosted or end-user targeted. Mohawk seems like it will satisfy all needs, but has recently become closed-source and is largely vaporware. Others include the Dark Star Apache Configuration Editor, the Apache portion of Linuxconf (bundled with many Linux distributions), and Webmin.

References

- Albitz, Paul & Liu, Cricket, DNS and Bind, Third Edition, O'Reilly & Associates, Sebastopol, CA, 1998.
- Apache Server Documentation, http://httpd.apache.org/docs/
- Apache Virtual Host documentation, http://httpd.apache.org/docs/vhosts/
- Dart, Susan, Change Management: Containing the Web Crisis, http://www.perforce.com/perforce/conf99/dart.html
- Markley, David and Scott Money, Web Content Management, available in the printed materials from the Perforce User Conference 2000.
- P4FTP Documentation, http://www.perforce.com/perforce/doc.011/manuals/p4ftp/index.html
- P4Web Documentation, http://www.perforce.com/perforce/doc.011/manuals/p4web/index.html
- Perforce Tech Note #2, http://www.perforce.com/perforce/technotes/note002.html
- Vance, Stephen, Advanced SCM Branching Strategies, http://www.vance.com/steve/perforce/Branching_Strategies.html
- Wingerd, Laura, Web Content Management with Perforce, http://www.perforce.com/perforce/wcm.html

Referenced Products

- Adobe GoLive!, www.adobe.com
- Apache, www.apache.org
- Comanche, www.covalent.net/projects/comanche
- Dark Star Apache Configuration Editor, www.darkphoton.com/darkstar/DSTACE
- DAVE, www.thursby.com
- Macromedia DreamWeaver, www.macromedia.com
- Mohawk, everythinglinux.org/Mohawk
- NFS vendors
  - Hummingbird Software, NFS Maestro, www.hummingbird.com
  - WRQ, Reflection NFS Client, www.wrq.com
  - NetManage, ViewNow InterDrive Client, www.netmanage.com
- Samba, www.samba.org
- tkp4, web.cuug.ab.ca/~macdonal/tkp4
- WebKeeper, public.perforce.com/public/perforce/webkeeper/index.html
- Webmin, www.webmin.com/webmin/
- Win4Lin, www.netraverse.com
- VMWare, www.vmware.com
- Zope, www.zope.com