The SkunkWeb Operations Manual

Release 3.0

Drew Csillag and Robin Thomas

January 29, 2007

CONTENTS

CHAPTER

ONE

Introduction

This manual explains the installation and use of the software included in the Skunk software distribution.

The Skunk software distribution has one main software tool: the SkunkWeb web application server. SkunkWeb's purpose is to provide a programming environment that makes it easy to build applications that generate "dynamic documents" in response to Web (HTTP) requests. Think of SkunkWeb as doing the same job as Vignette StoryServer, ASP, ColdFusion, or regular CGI applications.

The other software tools are utilities to help you use SkunkWeb, or general-purpose software to help in web applications programming. This manual will concentrate almost exclusively on the use of SkunkWeb, with little or no mention of the other software in the distribution.

CHAPTER

TWO

Prerequisites

2.1 Hardware and Operating System

The SkunkWeb server, and all of the other software in the Skunk distribution, are designed to run on any reasonable Unix system. We have successfully built and installed Skunk software on Solaris 2.6 and 2.7, various flavors of Linux 2.x, and recent versions of FreeBSD.

A reasonable amount of disk space is obviously required, more may be needed depending on the caching configuration.

2.2 Software

- An ANSI C compiler
- Python 2.1.1 (or later) (optionally) compiled with the crypt module installed. Actually, Python 2.1 works just fine, except that the license for 2.1 isn't GPL compatible, thus I don't believe you can distribute the whole thing precompiled.
- The mxBase Python extension package, available from www.egenix.com.
- Optionally, Apache 1.3.x configured with (at least) the switch `--enable-module=so`. If you run into some weirdness with it (specifically with the root `index.html`), you may want to try adding these too: `--disable-module=dir --disable-module=autoindex`. The SkunkWeb/Apache interface will not build unless Apache has been configured for dynamic extension modules.
- Optionally, Sudo if you want users other than that which SkunkWeb would normally run as to be allowed to start/stop/restart SkunkWeb.

2.3 Users and Groups

It is recommended that the SkunkWeb server run as the same user that the Apache server runs as. Historically, we have created a user named `apache`, with a home group named `http`, for both Apache and SkunkWeb to run under. The Skunk software installation will create many directories and files owned by the user/group you choose during the installation; we recommend that you create the user apache and group http before starting the Skunk installation, so that the Skunk installer can use apache:http to set ownership of the files it installs.

CHAPTER

THREE

Build and Install

For the most part, SkunkWeb will successfully install by just typing:

```
# configure; make; make install
```

But of course, you may want to customize the installation.

3.1 Configure Script Options

Outside of the normal `configure` options, SkunkWeb provides the following:

`--with-skunkweb` Install SkunkWeb as part of this installation (default yes)

`--with-python=/path/python` Specify path to python executable

`--with-user=user` Make Skunk installation owned by user

`--with-group=group` Make Skunk installation owned by group

Additional options if building SkunkWeb (which will be built by default):

`--with-webdoc` Install some sample content (default yes)

`--with-services=svcs` Specify the list of services to install space delimited list of personas to install, valid names are:

`sessionHandler`, `requestHandler`, `remote`, `ae_component`, `remote_client` `aecgi`, `httpd`, `basicauth`, `oracle`, `pars`, `templating`, `web`, `fcgiprot`

default is to include everything.

`--with-sudo=/path/sudo` Specify path to sudo

`--without-mod_skunkweb` Don't build mod_skunkweb

`--with-apxs=/path/apxs` Specify path to the apxs program

N.B.: SkunkWeb, by default installs itself in `/usr/local/skunk`, but may be overridden by the standard `configure` option `--prefix`. The rest of this manual assumes that SkunkWeb was installed in `/usr/local/skunk`.

3.2 Installation of the Cache Reaper

If you are going to use component caching (you probably should), you probably want something to delete expired cached components periodically as they tend to build up over time. If you add the following line (editing if necessary):

```
0 0 * * *  /usr/local/skunk/util/cache_reaper.py -c /usr/local/skunk/cache
```

to the crontab of the user that SkunkWeb runs as, it will go about finding dead items and removing them from the cache.

You may want to wrap the script with a shell script to `nice` the process (such that it runs with lower priority).

3.3 Apache Configuration

If you are going to have Apache fronting SkunkWeb (the default configuration), for the most part, just appending the contents of `SkunkWeb/mod_skunkweb/httpd_conf.stub` to your existing `httpd.conf` file (editing as required) will often suffice, but if you are doing virtual hosting, obviously things get a bit more complex.

CHAPTER

FOUR

Starting/Stopping

Starting and stopping SkunkWeb is a relatively simple matter using the `swmgr` script in `/usr/local/skunk/bin`.

The `swmgr` takes one argument that has one of three different values:

`start` Start the SkunkWeb server if it is not already running. If it is already running, you will usually see an error message that says something like “Address already in use”.

`restart` Restart a running SkunkWeb server. If the SkunkWeb server is not already running, you will see the message “SkunkWeb pidfile missing. Aborting...”

`stop` Stops the running SkunkWeb server. If the SkunkWeb server is not already running, you will see the message “SkunkWeb pidfile missing. Aborting...”

CHAPTER

FIVE

Confi guring the SkunkWeb Server

5.1 Services

There are a plethora of services available that....

`ae_component` A service used by `remote` and `templating` that initializes the AE component facilities. This service, coupled with the `web`, `requestHandler` and `templating` services is what really makes the server tick.

`aecgi` The service that handles the SkunkWeb server side of the Apache (or actually anything that speaks the aecgi protocol) to SkunkWeb connectivity.

`fcgiprot` The service that allows FastCGI connections to the SkunkWeb server.

`basicauth` Allows for doing HTTP basic authentication (see section 7.3, page 20).

`httpd` Makes it so SkunkWeb can handle HTTP requests directly, without the use of Apache. Requires the `web` service.

`mysql` Service to cache MySQL connections (see section 6.3, page 18).

`oracle` Service to cache Oracle connections and stored procedure definitions (see section 6.3, page 18).

`pars` Allows the packaging of stuff that would normally be found in the docroot to be stored in parfile archives (see section 7.2, page 19).

`postgresql` Service to cache PostgreSQL connections (see section 6.3, page 18).

`remote` Allows SkunkWeb to serve remote component requests (see section 6.1, page 17); requires the `requestHandler` and `ae_component` services.

`remote_client` Allows SkunkWeb to make remote component requests (see section 6.1, page 17); requires the `templating` service.

`requestHandler` Adds a request handling framework, built on top of by some of the other services.

`sessionHandler` Handles the persistence of user session data in a MySQL database, the local filesystem, or other user-defined data store; requires the `web` service.

`templating` Hooks together the `web` and `ae_component` services, in addition to adding a few tags.

`userdir` Make it so requests under the uri */user/rest*, will go to the directory *users_home*`/public_html/`*rest*.

`web` Service to manage web requests and setup the `CONNECTION` object; requires the `requestHandler` service.

5.2 Configuration Options

5.2.1 Scoping

Global configuration information is accessed throughout SkunkWeb as attributes of the `SkunkWeb.Configuration` pseudo-module, which, although accessed like a Python module, is actually a subclass of `scope.Scopeable`, a chameleon object the values of whose attributes can be made to vary depending on the environment to which it has been told to adapt. In SkunkWeb's requestHandler and web services, scoping is used to (potentially) modify the Configuration object on the basis of values in the request per se or the request environment, thereby making possible direct support for configurable virtual hosts (based on host name, port, and/or IP address), location directives, and other more exotic options.

Scoping is configured in the sw.conf file by the use of the `Scope`, `Location`, `Host`, `Port` and `IP` configuration directives, which look like (and are) Python method calls. The `Scope` directive is always the outer container of the other directives, taking their return values (`scope.ScopeMatcher` instances) as its arguments; it applies the ScopeMatchers passed to it to the Configuration object. The first argument of the other directives is a value which is matched against a value in the request; in the case of `Port`, for instance, it should be an integer which will be compared with the port on which the request came. `Host`, `Port` and `IP` then take an arbitrary number of unnamed `ScopeMatcher` arguments, which are applied only in the case that they and all their parents match the request; `Location` is excluded from this recursive bonanza. `Host`, `Port`, `IP` and `Location` all then take an arbitrary number of named arguments, which should be configuration options and their desired values.

Take, for example, the following possible `Scope` directive:

```
Scope(IP('192.168.65.12', Port(9887, job=REMOTE_JOB)))
job=TEMPLATING_JOB
```

This causes `Configuration.job` to return the value REMOTE_JOB for the case that the ip address of the request is 192.168.65.12 and the request port is 9887, and TEMPLATING_JOB otherwise.

To turn on basic authentication for the directory '/private' for host 'www.foo.com', you could use the following:

```
Scope(Host('www.foo.com',
           Location('/private',
                    basicAuthName='privileged zone',
                    basicAuthFile='/usr/local/skunk/var/AUTHDB')))
```

5.2.2 The Options

Note: Arguments that are not *scopable* are denoted with a [N] in their description.

Core Parameters

`services` [N]A list of services you want to load

`accessLog` File path where the access log will be written. Defaults to `'/usr/local/skunk/var/log/access.log'`.

`errorLog` File path where the error log will be written. Defaults to `'/usr/local/skunk/var/log/error.log'`.

`regularLog` File path where the regular log will be written. Defaults to `'/usr/local/skunk/var/log/sw.log'`.

`debugLog` File path where the debug log will be written. Defaults to `'/usr/local/skunk/var/log/debug.log'`.

`pidFile` [N]File path where a file will be written whose sole contents is the parend process id. Defaults to `'/usr/local/skunk/var/run/sw.pid'`.

`stampEveryLine` If you don't want every line of a multiline log entry to be prefixed with a logstamp (the default behavior), set this to a false value. Default is 0.

`initialDebugServices` [N]List of services that are INITIALLY set to be debugged. To determine what services are being debugged at runtime, use SkunkWeb.ServiceRegistry.getDebugServices(). Default is the empty list.

`numProcs` [N]The number of child processes to be started. This is the max number of concurrent page views (although if Apache sits in front, it may look like quite a few more). Default is 15.

`maxKillTime` [N]The number of seconds that the parent will wait after having sent SIGTERM to the children before killing the children off with SIGKILL during a shutdown or restart. Default is 5.

`pollPeriod` [N]You probably don't need to change this, but it's the interval at which the parent looks for corpses that need to be buried and replaced with new, living kids. Default is 5.

`maxRequests` [N]The number of requests that a child server process will handle before committing suicide and having the parent respawn. This is mainly to make sure we aren't leaking memory, but you can probably set it to something a bit higher. Default is 256.

`userModuleCleanup` [N]For development, this will cause changes to sitelibs to be reflected on the next request, without a bounce of the server. Turn it off for production environments. Default is 0.

The `ae_component` service

`documentRoot` The directory that is used to hold templates, images, msgcats, comps, etc. Default is `'/usr/local/skunk/docroot'`.

`compileCacheRoot` The location on disk of where the compile cache should reside. This should be a local filesystem. Default is `'/usr/local/skunk/cache'`.

`useCompileMemoryCache` Store compiled templates and Python code in memory. Increases the memory footprint a bit, but reduces I/O a bit and CPU utilization a bit as it doesn't have to load and deserialize the compiled images from disk.

`componentCacheRoot` The location of the component cache on disk. Can be the same as the compileCacheRoot. Default is `'/usr/local/skunk/cache'`.

`failoverComponentCacheRoot, numServers, failoverRetry` If using NFS (or other shared filesystem) for the component cache, set `numServers` to the number of shared filesystems that the component cache will be on, `failoverRetry` to the number of seconds to wait before trying the shared filesystem again (it should be soft mounted!!!), and `failoverComponentCacheRoot` to an area on local disk to use as a temporary replacement for the server that failed in the meantime. If `numServers` is 0, the `failover*` options are ignored. Default for `numServers` is 0, the default for `failoverComponentCacheRoot` is `'/usr/local/skunk/failoverCache'` and the default for `'failoverRetry'` is 30.

`mimeTypesFile` [N]The location of your mimetypes file. Default is `'/usr/local/skunk/etc/mime.types'`

`componentCommentLevel` Level of commenting to be inserted in HTML documents when components are entered/exited.

0 no commenting (default)

1 only show component name in comments

2 show component name and argument names

3 show component name, argument names and values

Default is 0

`runOutOfCache` causes the system to not actually read any of the source files as long as there is a compiled version in the cache, even if they change.

`dontCacheSource` causes the system to not include any source code in the compile cache files.

The `aecgi` Service

`AecgiListenPorts` [N]List of ports (specified as string(s) of either `TCP:`*hostname*`:`*portnumber* or `UNIX:`*filepath*`:`*octal_socket_permissions*) that the SkunkWeb server will listen for AECGI protocol (the protocol used by `mod_skunkweb`) connections . Default is `['TCP:localhost:9888']`.

The `fcgiprot` service

`FCGIListenPorts` [N]List of ports (specified as string(s) of either `TCP:`*hostname*`:`*portnumber* or `UNIX:`*filepath*`:`*octal_socket_permissions*) that the SkunkWeb server will listen for FastCGI protocol. Default is `['TCP:localhost:9999']`.

The `httpd` Service

`lookupHTTPRemoteHost` [N]If you set this to a true value, the CGI environmental variable HTTP_REMOTE_HOST will always be set, involving a potentially costly DNS lookup. If you don't want it, you can still use HTTP_REMOTE_ADDR and perform the lookup on demand. Default is 0 (no lookup and no HTTP_REMOTE_HOST variable in the environment). N.B. This only affects the httpd service, and has no bearing on environmental variables SkunkWeb receives from Apache or elsewhere. Default is 0.

`HTTPKeepAliveTimeout` [N]How long (in seconds) the http keepalive timeout should be between HTTP requests. Default is 15.

`HTTPListenPorts` [N]List of port(s) for the httpd service to listen on (specified in the same way as the `AecgiListenPorts` socket is). Default is `['TCP::8080']`.

The `requestHandler` Service

`DocumentTimeout` The time (in seconds) allowed for a request to complete before timing it out (by raising an exception). Default is 30.

`PostResponseTimeout` The amount of time (in seconds) allowed for the PostRequest hooks to execute before timing them out. Default is 20.

`job` The job that the server should perform. TEMPLATING_JOB and REMOTE_JOB are likely to the only ones you need to specify among those available in the core services, and they are pre-imported into the namespace in which the sw.conf file is executed. It is essential that the sw.conf file specify this constant, as no default is specified anywhere else. For more information about job strings, see the developer's manual.

The `mysql` Service

`MySQLConnectParams` [N]To use cached connections with the pylibs/MySQL module set this to a dictionary of 'name': 'connection_string' pairs. Default is {}.

he `oracle` Service

`OracleConnectStrings` [N]To use cached connections with the pylibs/Oracle module set this to a dictionary of 'name': 'connection_string' pairs Default is {}.

`OracleProcedurePackageLists` [N]To speed up the use of stored procedures (saves one query per sp fetch) we can preload the procedure signatures ahead of time. Set to a dictionary of 'name': ['list', 'of', 'package-names'] where name matches that used in OracleConnectStrings. Default is {}.

The `postgresql` Service

`PostgreSQLConnectParams` [N]To use cached connections with the pylibs/PostgreSql module set this to a dictionary of 'name': 'connection_string' pairs. Default is {}.

The `remote` Service

`RemoteListenPorts` [N]List of port(s) for the remote service to listen on (specified in the same way as the `AecgiListenPorts` socket is). Default is `['TCP:localhost:9887']`.

The `sessionHandler` Service

`SessionTimeout` The session timeout, in seconds. Sessions which have seen no action for this period of time are ripe for reaping. Default is 1800 (30 minutes).

`SessionReapInterval` [N]Session reap interval, in seconds. When the reap interval has elapsed, the server may reap old sessions in the aftermath of its next request handling cycle. To turn off reaping (if more than one server is using the same session store, or if another process is performing reaping separately), set this to zero or a negative value. Default is 300 (5 minutes).

`SessionStore` The session store implementation class (full package name). This defaults to None, which, if the sessionHandler service is imported, will cause an error. To use sessionHandler, you must specify a session store implementation; e.g.:

```
SessionStore='sessionHandler.MySQLSessionStore.MySQLSessionStoreImpl'
```

Default is `None`; two SessionStore implementations are currently available, `MySQLSessionStore.MySQLSessionStoreImpl` and `FSSessionStore.FSSessionStoreImpl`. SessionStore can be scoped by host, port, or ip; it should not be scoped by location.

`SessionIDKey` [N]The key under which the session is kept (in a cookie or in URL rewriting). Default is `'sessionID'`.

The following `sessionHandler` config variables specify names of the session table and its columns and the MySQL host, user and password.

`SessionHandler_MySQLHost` ='localhost' [N]

`SessionHandler_MySQLUser` ='sessionHandler' [N]

`SessionHandler_MySQLPass` ='sessionPass' [N]

`SessionHandler_MySQLDB` ='sessionStore' [N]

`SessionHandler_MySQLTable` ='Sessions' [N]

`SessionHandler_MySQLIDColumn` ='id' [N]

`SessionHandler_MySQLPickleColumn` ='pickle' [N]

`SessionHandler_MySQLTimestampColumn` ='accessTime' [N]

The `templating` Service

`errorTemplate` [N]path to a template in the docroot (or parfile), dependent on scope that should be shown if the we send back an error (500) page The exception text will show up in an additional template variable ERROR_TEXT. Default is `None`.

`notFoundTemplate` [N]Path to a template in the docroot (or parfile), dependent on scope that should be shown if the we send back a 404 Not Found response. Default is `None`.

`indexDocuments` List of document names that will be recognized as index documents; Default is `['index.html']`

`hideMimeTypes` A list of mime types that should not be accessible via web request. Default is

```
[
    "text/x-stml-component",
    "text/x-stml-python-component",
    "text/x-stml-data-component",
    "text/x-stml-python-data-component",
]
```

`interpretMimeTypes` Mime types of things that should be rendered as templates when called via web request. Default is `["text/html", "application/x-python"]`.

`defaultIndexHtml` Path to a template (or other item) to be used for a directory if no index document is present (ála `indexDocuments`). Defaults to `None` — thus this feature is disabled by default.

The following parameters control how mail is sent from SkunkWeb. They are used by the `sendmail` function from `templating.MailServices` which in turn is used by the `sendmail`-tag.

`MailMethod` The method by which to send the mail. The default value is `'sendmail'`.

Possible values for MailMethod:

sendmail
: This is the standard value for MailMethod. The mail is sent to the program configured by `SendmailCommand`. It should be sendmail or a program with similar functionality. The sendmail function opens a pipe to the configured command and catches it's stderr output. The envelope sender is set by using the parameter `-f` and the (envelope-) receivers are given on the commandline.

qmail
: This value is used for delivering mail through qmail. The mail is sent to the program configured by `QmailInject`. It should be the qmail program qmail-inject or new-inject from D.J. Bernsteins mess822 package. The sendmail function opens a pipe to the configured command and catches it's stderr output. The envelope sender is set by using the parameter `-f` and the (envelope-) receivers are given on the

commandline.
You may notice that this sounds very similar to the description for the value `sendmail` - that's absolutely right. At the moment there is no real difference. It is planned to support per receiver and per message VERP (see the qmail homepage for details). When this is implemented, the both methods will be different.

relay
This value sets the MailService so that mail is delivered through SMTP to a Relay-Host (also called SmartHost sometimes). The RelayHost is configured by the parameter `MailHost`. SMTP authentication or TLS are not supported.

`MailHost` If using the `relay` mail method the relay host to send it to. Default is `'localhost'`.

`SendmailCommand` If using the `sendmail` mail method, the path to the sendmail (or sendmail compatible) program. Defaults to `'/usr/lib/sendmail'`.

`FromAddress` The fallback sender (used in the `From:` header and as envelope sender) if there is no sender given in the call to the `sendmail` function. The `sendmail` function **does not** accept an empty envelope sender - these are used for bounces and it's not likely that a web application has to send bounces.
Please note: A major annoyance to mail administrators is the use of invalid envelope senders. Many of them block sites generally if they are lazy/stupid regarding envelope senders or they do intensive checks on them. So it's not only good behaviour to set a valid envelope sender, it's also in your own interest to do so. The fallback is your defense against lazy programming, so use it. Defaults to `"root@localhost"`.

`QmailInject` If using the `qmail` mail method, the path to the injector program. Defaults to `'/var/qmail/bin/qmail-inject'`.

The `userdir` Service

`userDir` Whether to enable the `userdir` functionality. Default is 1.

`userDirPath` What directory name to append to the users home directory when utilizing this service. Default is `'public_html'`.

CHAPTER

SIX

Service Details

6.1 Remote Components

It is possible to call SkunkWeb components on other servers, thanks to the `remote` and `remote_client` services.

When the `remote` service is loaded, the Configuration variable `RemoteListenPorts` is defined is `sw.conf`, and Configuration.job is scoped as REMOTE_JOB for the appropriate requests, the server's components are available for remote access over the specified port(s). (By default, `sw.conf` uses port 9887 and scopes Configuration.job accordingly.)

The `remote_client` service makes it possible to access remote components on other servers. When loaded, it installs an alternate component handler, which will be invoked if the component location argument to the component call is a "swrc" (SkunkWeb Remote Component) url, of the form:

```
swrc://hostname[:port]/path/to/component
```

Assuming that localhost is running the remote service on the default port (9887) and that the remote_client service is also loaded, the following two component calls should produce equivalent output:

```
<:datacomp /foo/bar/spam.comp a='3' b='4' cache=yes:>

<:datacomp "swrc://localhost/foo/bar/spam.comp" a='3' b='4' cache=yes:>
```

The remote component handler works with all components whose output is pickleable: regular components, data components, and includes.

These two services have no dependencies on one another in a given installation; a database server, for instance, might run a SkunkWeb instance dedicated to serving remote components to a number of SkunkWeb instances that themselves serve web pages and do not need to run the remote service.

The remote protocol is unencrypted and relatively transparent, although not designed for legibility; it is not suited for the transport of privileged data across the internet. A secure remote protocol is planned for a future release.

6.2 `sessionHandler`

The `sessionHandler` service adds to the `web` service's `CONNECTION` object the ability to create a session object, which can contain and automatically persist pickleable Python data for the duration of a web session, maintained and tracked by a session id placed in a cookie. For sessionHandler to work, it is necessary to specify a `SessionStore`

implementation by setting the configuration variable of the same name in `sw.conf`.

6.3 Database Services

CHAPTER

SEVEN

Miscellaneous SkunkWeb Utilities

7.1 vicache.py

The `vicache.py` utility is for use mainly by the SkunkWeb maintainers and anyone doing STML hacking, but sometimes can be useful for the general user when they are getting an error that they can not explain without seeing exactly what is going on.

To invoke `vicache.py`, go into your compile cache directory and then:

```
# python /usr/local/skunk/util/vicache.py name_of_cachefile.htmlc
```

The file (in this case `name_of_cachefile.htmlc`) is the name of the original STML component/template/etc. with a 'c' appended onto the name.

What then will happen is that either `vi` or whatever editor you have in you EDITOR shell environment variable will be started with the contents of the Python source that is generated by the STML compiler.

At this point, you can edit the source text, save it (or not) and exit your editor. `vicache.py` will then ask you if you wish to save your changes into the cache file.

7.2 The par Archive Tool

The `par.py` program is used to create and extract files from `par` files. It has three modes of operation: create, list and extract.

To create `par` files, type:

```
par c parfile_to_create.par file_or_dir1 file_or_dir2 ...
```

To list the contents of `par` files, type:

```
par t parfile_to_list.par
```

To extract tho contents of `par` files, you can either extract everything or extract only certain files. To extract everything, type:

```
par x parfile_to_extract.par
```

Otherwise,

```
par x parfile_to_extract.par file_or_dir1 file_or_dir2 ...
```

7.3 The swpasswd Utility

The swpasswd utility is a simple program to maintain password files.

Its general form is:

```
swpasswd [-cb] password_file username [password]
```

The two options are:

- `-c` Create a new password file
- `-b` Specify the password on the command line as opposed to being asked for it interactively.

7.4 swpython

swpythonis basically the SkunkWeb server without the server part. It loads the environment just as running skunkweb does, but instead of opening ports and forking, it (by default) drops you at a Python prompt.

```
Usage: swpython [-ih] [-c configfilelist] [-e script] [scriptname]

  -c,--config-files=configfilelist   list of config files to load delimited
                                     by colons
  -h,--help                          show help screen
  -i                                 if scriptname is provided, run script
                                     and then start interactive interpreter
  -e,--execute=script                Execute script then exit
  -r,--redir-logs                    Redirect logs to stderr
```

7.5 swcgi

The swcgi script is used to be able to use Apache and SkunkWeb together without the use of mod_skunkweb. Specifically, in cases where you're not allowed to install Apache modules, or things of that nature.

A typical configuration would be to add the following to your httpd.conf:

```
ScriptAlias / /usr/local/skunk/bin/swcgi
```

To really use it though you should check out the top of the script and edit it to suit your needs (right now, the thing to check out is the value of the connectInfo variable).

CHAPTER

EIGHT

Virtual Hosting

Virtual Hosting with SkunkWeb is a relatively simple matter. Mostly what you do is use `Scope` statements in the configuration file to affect the `documentRoot` and the `compileCacheRoot`. Good practice dictates that you should probably change the `componentCacheRoot` also. If you have `numServers` set to something other than zero, you should change `failoverComponentCacheRoot` and `failoverRetry` in the `Scope` statement.

For example: you want to serve both `foo.com` and bar.com sites. First you choose which is the "default" server, i.e. if you were to hit the server with just the IP address and no name, which you would get.

```
documentRoot = '/usr/local/skunk/docroot/foo.com'
compileCacheRoot = '/usr/local/skunk/cache/foo.com'

Scope(Host('bar.com',
    documentRoot = '/usr/local/skunk/docroot/bar.com',
    compileCacheRoot = '/usr/local/skunk/cache/bar.com',
))
```

Or alternatively, you could set the "global" `documentRoot` and `compilecacheRoot` to something entirely different and have an additional `Scope` statement to handle foo.com.

All of this applies regardless of whether you are using Apache/`mod_skunkweb` or the `httpd` service.

CHAPTER

NINE

Care and feeding

Luckily, there isn't much to do in terms of care and feeding if you've set up the cache reaper in cron to do it's job. The one main thing that you will probably want to do is to roll the logfiles.

With SkunkWeb, this is relatively simple. First you rename the existing log files, then restart SkunkWeb. That's it.

CHAPTER

TEN

Tuning

There are a few things in SkunkWeb that you can tune for better performance.

The first is the number of processes that the SkunkWeb server spawns. If you are running into a performance plateau and your CPU utliization is less than 100%, try increasing the number of SkunkWeb servers (by changing the `numProcs` config variable).

Another is the memory compile cache (activated by the `useCompileMemoryCache` option). What this does is keep memory images of the on-disk compile cache so that it can reduce I/O and CPU utilization that would normally be used loading and deserializing the compiled forms of compiled things (template, python code, etc.).

The big tuning parameter is by far: the cache.

10.1 Caching

Many common web tasks can be sped up by caching. Caching is activated by calling a component (data or regular) with the cache parameter set. What this does is say "If I call this component with the same arguments again before the cache expires, don't actually execute the component, but just return what's in the cache".

The obvious question to ask is "when does the cached version expire?" Luckily, this has a simple answer. If you set the cache expiration in a component (either via the `cache` tag in an STML component, or by setting `__expiration` otherwise), it will expire according to that. If you don't say when the cache expires, it defaults to the value of the `defaultExpiryDuration` configuration variable (which defaults to 30 seconds). If you use the force or old component evaluation modes, it will follow those rules instead (see the STML Reference or the Developers Manual for details).

10.1.1 Caching Architecture

The caching architecture of SkunkWeb is fundamentally rather simple from the OS's point of view and luckily doesn't get that much more complicated when you get to the upper level details.

For component output caching, which makes up the majority of the number of files in the cache directory, we take the output of the component and a hex string representation of the MD5 hash of the component arguments and write it to a temp file. We then rename it to the real file name. This is done so we don't have to worry about write atomicity and having the SkunkWeb load a half-written file, not to mention that this scheme works properly over NFS!

An example: say we have a component whose path is `/foo/bar/baz.comp` and the argument hash is `0123456789ABCDEFFEDCBA9876543210`. The temp file will be written into a file named `/foo/bar/baz.comp/01/23/`*hostname_pid*`.`*countervalue*`.tmp` and subsequently renamed (when writing is complete) to `/foo/bar/baz.comp/01/23/0123456789ABCDEFFEDCBA9876543210.cache` in the appropriate component cache directory. There is another file written for component output with the same name as the cache file, but replacing `.cache` with `.key` which contains a somewhat textual form of the component arguments

suitable for use when doing cache clears (see section 10.1.3, page 26).

For compiled templates and compiled python code, there is no hash (and thus, no `.key` file), so the cache filename is `path_to/templatenamec` and the rest of the process is the same.

10.1.2 Sharing the Cache

Often, it is very desirable to share the component cache across application servers for performance, managability and synchronicity issues.

The location of the component cache is a separate configuration variable (the `componentCacheRoot` configuration variable). The component cache can be segmented easily among multiple fileservers and the SkunkWeb will now fail over to a local cache if the file server goes away unexpectedly.

The component cache segmentation is done as follows. If the `numServers` configuration variable is set to something other than zero, a number of options are now effective. At the top of the component cache root (CCR), there will be N directories, where N is the number of servers, numbered 0..N-1. You can either mount your remote volumes there or have symbolic links to where they are actually mounted, or any number of similar options.

When SkunkWeb goes to operate on a cached form of component output, it takes the last 16 bits of the MD5 hash that it computed, mods it by the number of file servers and operates on the cache entry under the directory numbered by the result of the mod operation. If an operation fails when operating on the component cache (other than the usual errors of files existing already or files not existing), SkunkWeb will fail over that servers portion of the component cache to the (presumably) local failover cache (specified by the `failoverComponentCacheRoot` configuration variable) for `failoverRetry` seconds. After `failoverRetry` seconds, it will again attempt to go back to using that servers' directory under the CCR.

NOTE:The `failoverComponentCacheRoot` and `failoverRetry` options are ineffective if `numServers` is zero.

To make this work over NFS, the filesystems *must* be soft mounted!

10.1.3 `clearCache`

One word: don't use `clearCache` if you can avoid it, or at least don't come complaining to me that it's slow. Because of the way `clearCache` is implemented, what it has to do is examine *every* cached version of a component to see if it matches the arguments you passed (unless you set `matchExact`, in which case, don't worry, you won't have any problems), which if you have a large number of cached versions of a component can be exceedingly slow, in some cases I've seen, exceeding the `documentTimeout`.

If anyone can provide a version of `clearCache` that is fast and doesn't overly affect normal usage of the cache (read as: read operations), I would be greatly appreciative.

10.1.4 Filesystem Tuning

Filesystem performance can play a dramatic role in the performance of the SkunkWeb server. If filesystem performance is slow, cache operations will be slow also.

On Linux, the default filesystem settings seem to be just fine for the ext2 filesystem, but on other operating systems, the situation is a bit different.

If you are using a UFS filesystem (which is typical on BSD and Solaris), you want to set the `async` and `noatime` options or cache write operations will be *really* slow (70ms vs <.01ms). At first, some may be uneasy about setting the `async` flag on a filesystem being wary of system crashes and filesystem corruption, but worry not. Because this is a *cache*, running `mkfs` at boot time on the cache filesystem should be an entirely satisfactory solution. Given the system will be a little sluggish at first, but as the cache fills up, speed will improve. Alternatively, if you are on Solaris

(at least), you can turn on the logging option and that helps (cache writes go to about 20ms), but not quite as good as full `async` mode.

Another option is to allocate a *whole* lot of swap and put the cache in a `tempfs` filesystem.

10.1.5 Flushing the Cache

Flushing the cache is usually a pretty simple matter. What you do is make a directory in the cache that won't be stepped on and move the other directories in the cache into the newly created directory. Then remove the new directory (which may take a bit of time depending on the size of the cache).

10.2 apache

buffering for slow clients configurability

10.2.1 What Happens

1. The client connects to the Apache web server and makes an HTTP request. Apache receives the request and prepares to handle it; that is, it begins to determine how it will send back a response to the request.

2. Apache goes through several request handling phases which handle different parts of the response. When configured for use with SkunkWeb, Apache will do nothing for most of these request handling phases. When Apache reaches the main request handling phase, it uses a extension module from the Skunk software to handle the request. This extension module is called `mod_skunkweb`. This means that the request will be passed back to SkunkWeb, and that all the needed information about the request will be passed to SkunkWeb in the same general way that Apache would pass request information to CGI programs.

3. The `mod_skunkweb` module then handles the request. The first thing it does is to take all of the environment variables available, including the variables set by Apache to include request information. (For instance, the environment variable REMOTE_ADDR will contain the IP address of the client making the request. There are many other environment variables that contain request information, and `mod_skunkweb` gathers them all together. At this point it also gathers the HTTP request body also.

4. `mod_skunkweb` then encodes all of the environment variables and request body into a special format that the SkunkWeb server will understand (see the Developers Manual for details).

5. `mod_skunkweb` then opens a connection to the SkunkWeb server. The Apache configuration will tell `mod_skunkweb` on which host and on which port number the SkunkWeb server is running. The typical configuration has the SkunkWeb listening on the localhost port 9888.

6. If the connection succeeds, `mod_skunkweb` then sends the encoded environment variables over the connection, and waits for SkunkWeb to return a response that can be sent back to the Web client. If the connection fails, `mod_skunkweb` will retry the connection a few times (utilizing the socket addresses listed in the `SkunkWebFailoverHosts` configuration variable). If the retries also fail, `mod_skunkweb` will return a customizable error page, with a 500 Server Error status, as the response to the client.

7. SkunkWeb, like Apache version 1, runs as a pool of processes. The SkunkWeb parent process does not listen for connections from `mod_skunkweb`; its only job is to create child processes that will do all the listening and request handling. (The amount of child processes it creates, as well as other important things, are configurable in SkunkWeb's configuration files.) Thus when we say that "SkunkWeb is running", we mean that the SkunkWeb parent process is running and has created one or more child processes that are listening for connections from `mod_skunkweb`.

8. An SkunkWeb child process, then, accepts the connection from `mod_skunkweb` and receives the encoded environment variables containing all of the request information. It then decodes the variables and checks that they are OK.

9. SkunkWeb then builds a `CONNECTION` object. It fills the `CONNECTION` object with request information from the environment variables, including an entire copy of the environment variables. The `CONNECTION` object also provides data and operations for the subsequent web application to create a well-formed HTTP response to be sent back to the client.

10. SkunkWeb then does request pre-processing. Depending on its configuration, SkunkWeb can run one or more customized functions to do common things with the request, such as authentication, rewriting of URLs, and other modifications of information in the `CONNECTION` object.

11. SkunkWeb then looks at the URI of the document requested by the client, for example `/index.html`. SkunkWeb tries to find this document in its `documentRoot`, that is, its document filesystem. If SkunkWeb cannot find the document, it generates a customizable 404 Not Found response, returns it to `mod_skunkweb`, and closes the connection.

12. If SkunkWeb does find the requested document, SkunkWeb then looks at the MIME type of the document to determine if the document is static, and should be sent as-is with no modification. If the document is static, SkunkWeb reads the contents of the document file, makes an HTTP response out of it, and sends it back to `mod_skunkweb`.

13. If the requested document is not static, it is considered dynamic, which means that the document contains programming code that should be executed to generate the HTTP response. There are two kinds of dynamic programming languages supported: the Python programming language, and a tag-based programming language called STML, or Skunk Template Markup Language. As a rule, files ending in .py can contain Python code, and files ending in .html, .txt, and other common text formats can contain STML tag code.

14. SkunkWeb then runs the dynamic documents contents through a compiler, which will turn the document and its programmming instructions into executable code. The resulting code is executable Python code; SkunkWeb is written in the Python language. SkunkWeb keeps a cache of compiled Python code for each dynamic document so that it does not have to compile the same documents repeatedly.

15. Now that SkunkWeb has executable code for the dynamic document, it sets up a special execution environment for the code. It includes the `CONNECTION` object, helpful utility functions and modules, and special objects that monitor the executable code and capture the response output that the code generates. SkunkWeb then executes the code.

16. Dynamic documents often make use of a powerful feature in SkunkWeb: they can use other special dynamic documents to help generate the needed output. These special documents are called components: they cannot be requested by `mod_skunkweb`, but they can be used as building blocks by dynamic documents and assembled together on the fly. Components can be programmed in straight Python or as text with STML tag code embedded in the text. Components also can accept arguments from the dynamic documents, just as a function can receive arguments from the code that calls the function.

17. SkunkWeb keeps track of all the components that a dynamic document tries to use. If an error occurs in any code, or if a component is asked for by a dynamic document but does not exist, SkunkWeb will catch the error, generate a customizable error page, and return a 500 Server Error response to `mod_skunkweb`.

18. SkunkWeb also offers a powerful feature for components: caching of component output. Components may indicate in their program code whether their output should be cached by SkunkWeb, and for how long. When a component whose output is cached is called, SkunkWeb does not even bother executing the component's code; SkunkWeb simply retrieves the component's output from the cache and proceeds. The component output cache is one of SkunkWeb's most important features, and it also has the most impact of server administration and performance.

19. Once all of the dynamic document's code has executed (as well as all of the components the document used) and no errors have occurred, SkunkWeb gathers all of the response output generated by the document's code, puts it in the `CONNECTION` object, and prepares to send the response back to `mod_skunkweb`.

20. SkunkWeb examines the `CONNECTION` object, extracting the needed information and creating a well-formed HTTP response, with headers and everything. SkunkWeb then sends the HTTP response over the connection to `mod_skunkweb`, and then closes the connection.

21. `mod_skunkweb` receives the HTTP response, tells Apache about the response, and finishes its job.

22. Apache, which now has the response generated by SkunkWeb, performs any post-processing on the response that it feels is necessary, and returns the response to the client that requested it.

10.3 Squid

use of Squid as accelerator cache buffering for slow clients

INDEX

Symbols