Subject: Re: [OT] EasyVZ: An OpenVZ management GUI under the GPL
Posted by Slava Dubrovskiy on Tue, 13 Feb 2007 08:53:59 GMT
View Forum Message <> Reply to Message

> I am happy to annouce the availability of EasyVZ, an OpenVZ management GUI
> under the GPL. EasyVZ makes it simple to create,destroy and manage Virtual
> Private Servers from within a cozy and comfortable graphical user
> interface. The EasyVZ client can manage the server from anywhere over the
> network, although, currently there is no authentication available.
>
> EasyVZ is currently alpha quality software and is being released in the
> spirit of "releasing early", mainly to receive user feedback.
Thanks for the program. It that is necessary.
Wishes:
1. An opportunity to start backend as a demon. (patch it is attached)
2. Write log
3. An opportunity backup/restore VPS
4. An opportunity to set names (vzctl set VID - name)
5. Authentication

--
WBR,
Dubrovskiy Vyacheslav

```
diff -NurpP ezvz_org/backend/daemon.py ezvz/backend/daemon.py
--- ezvz_org/backend/daemon.py 1970-01-01 03:00:00 +0300
+++ ezvz/backend/daemon.py 2007-02-12 23:57:23 +0200
@@ -0,0 +1,386 @@
+"""Daemon base class
+
+Provides a framework for daemonizing a process.  Features:
+
+  - reads the command line
+
+  - reads a configuration file
+
+  - configures logging
+
+  - calls root-level setup code
+
+  - drops privileges
+
+  - calls user-level setup code
+
+  - detaches from the controlling terminal
+
```

```
+  - checks and writes a pidfile
+
+
+Example daemon:
+
+import daemon
+import logging
+import time
+
+class HelloDaemon(daemon.Daemon):
+    default_conf = '/etc/hellodaemon.conf'
+    section = 'hello'
+
+    def run(self):
+        while True:
+            logging.info('The daemon says hello')
+            time.sleep(1)
+
+if __name__ == '__main__':
+    HelloDaemon().main()
+
+
+Example hellodaemon.conf:
+
+[hello]
+uid =
+gid =
+pidfile = ./hellodaemon.pid
+logfile = ./hellodaemon.log
+loglevel = info
+
+"""
+
+import ConfigParser
+import errno
+import grp
+import logging
+import optparse
+import os
+import pwd
+import signal
+import sys
+import time
+
+
+class Daemon(object):
+    """Daemon base class"""
+
```

```
+    default_conf = ''    # override this
+    section = 'daemon'   # override this
+
+    def setup_root(self):
+        """Override to perform setup tasks with root privileges.
+
+        When this is called, logging has been initialized, but the
+        terminal has not been detached and the pid of the long-running
+        process is not yet known.
+        """
+
+    def setup_user(self):
+        """Override to perform setup tasks with user privileges.
+
+        Like setup_root, the terminal is still attached and the pid is
+        temporary.  However, the process has dropped root privileges.
+        """
+
+    def run(self):
+        """Override.
+
+        The terminal has been detached at this point.
+        """
+
+    def main(self):
+        """Read the command line and either start or stop the daemon"""
+        self.parse_options()
+        action = self.options.action
+        self.read_basic_config()
+        if action == 'start':
+            self.start()
+        elif action == 'stop':
+            self.stop()
+        else:
+            raise ValueError(action)
+
+    def parse_options(self):
+        """Parse the command line"""
+        p = optparse.OptionParser()
+        p.add_option('--start', dest='action',
+                     action='store_const', const='start', default='start',
+                     help='Start the daemon (the default action)')
+        p.add_option('-s', '--stop', dest='action',
+                     action='store_const', const='stop', default='start',
+                     help='Stop the daemon')
+        p.add_option('-c', dest='config_filename',
+                     action='store', default=self.default_conf,
+                     help='Specify alternate configuration file name')
```

```
+        p.add_option('-n', '--nodaemon', dest='daemonize',
+                     action='store_false', default=True,
+                     help='Run in the foreground')
+        self.options, self.args = p.parse_args()
+        if not os.path.exists(self.options.config_filename):
+            p.error('configuration file not found: %s'
+                    % self.options.config_filename)
+
+    def read_basic_config(self):
+        """Read basic options from the daemon config file"""
+        self.config_filename = self.options.config_filename
+        cp = ConfigParser.ConfigParser()
+        cp.read([self.config_filename])
+        self.config_parser = cp
+
+        try:
+            self.uid, self.gid = get_uid_gid(cp, self.section)
+        except ValueError, e:
+            sys.exit(str(e))
+
+        self.pidfile = cp.get(self.section, 'pidfile')
+        self.logfile = cp.get(self.section, 'logfile')
+        self.loglevel = cp.get(self.section, 'loglevel')
+
+    def on_sigterm(self, signalnum, frame):
+        """Handle segterm by treating as a keyboard interrupt"""
+        raise KeyboardInterrupt('SIGTERM')
+
+    def add_signal_handlers(self):
+        """Register the sigterm handler"""
+        signal.signal(signal.SIGTERM, self.on_sigterm)
+
+    def start(self):
+        """Initialize and run the daemon"""
+        # The order of the steps below is chosen carefully.
+        # - don't proceed if another instance is already running.
+        self.check_pid()
+        # - start handling signals
+        self.add_signal_handlers()
+        # - create log file and pid file directories if they don't exist
+        self.prepare_dirs()
+
+        # - start_logging must come after check_pid so that two
+        # processes don't write to the same log file, but before
+        # setup_root so that work done with root privileges can be
+        # logged.
+        self.start_logging()
+        try:
```

```
+            # - set up with root privileges
+            self.setup_root()
+            # - drop privileges
+            self.set_uid()
+            # - check_pid_writable must come after set_uid in order to
+            # detect whether the daemon user can write to the pidfile
+            self.check_pid_writable()
+            # - set up with user privileges before daemonizing, so that
+            # startup failures can appear on the console
+            self.setup_user()
+
+            # - daemonize
+            if self.options.daemonize:
+                daemonize()
+        except:
+            logging.exception("failed to start due to an exception")
+            raise
+
+        # - write_pid must come after daemonizing since the pid of the
+        # long running process is known only after daemonizing
+        self.write_pid()
+        try:
+            logging.info("started")
+            try:
+                self.run()
+            except (KeyboardInterrupt, SystemExit):
+                pass
+            except:
+                logging.exception("stopping with an exception")
+                raise
+        finally:
+            self.remove_pid()
+            logging.info("stopped")
+
+    def stop(self):
+        """Stop the running process"""
+        if self.pidfile and os.path.exists(self.pidfile):
+            pid = int(open(self.pidfile).read())
+            os.kill(pid, signal.SIGTERM)
+            # wait for a moment to see if the process dies
+            for n in range(10):
+                time.sleep(0.25)
+                try:
+                    # poll the process state
+                    os.kill(pid, 0)
+                except OSError, why:
+                    if why[0] == errno.ESRCH:
+                        # process has died
```

```
+                    break
+                else:
+                    raise
+            else:
+                sys.exit("pid %d did not die" % pid)
+        else:
+            sys.exit("not running")
+
+    def prepare_dirs(self):
+        """Ensure the log and pid file directories exist and are writable"""
+        for fn in (self.pidfile, self.logfile):
+            if not fn:
+                continue
+            parent = os.path.dirname(fn)
+            if not os.path.exists(parent):
+                os.makedirs(parent)
+                self.chown(parent)
+
+    def set_uid(self):
+        """Drop root privileges"""
+        if self.gid:
+            try:
+                os.setgid(self.gid)
+            except OSError, (code, message):
+                sys.exit("can't setgid(%d): %s, %s" %
+                (self.gid, code, message))
+        if self.uid:
+            try:
+                os.setuid(self.uid)
+            except OSError, (code, message):
+                sys.exit("can't setuid(%d): %s, %s" %
+                (self.uid, code, message))
+
+    def chown(self, fn):
+        """Change the ownership of a file to match the daemon uid/gid"""
+        if self.uid or self.gid:
+            uid = self.uid
+            if not uid:
+                uid = os.stat(fn).st_uid
+            gid = self.gid
+            if not gid:
+                gid = os.stat(fn).st_gid
+            try:
+                os.chown(fn, uid, gid)
+            except OSError, (code, message):
+                sys.exit("can't chown(%s, %d, %d): %s, %s" %
+                (repr(fn), uid, gid, code, message))
+
```

```
+    def start_logging(self):
+        """Configure the logging module"""
+        try:
+            level = int(self.loglevel)
+        except ValueError:
+            level = int(logging.getLevelName(self.loglevel.upper()))
+
+        handlers = []
+        if self.logfile:
+            handlers.append(logging.FileHandler(self.logfile))
+            self.chown(self.logfile)
+        if not self.options.daemonize:
+            # also log to stderr
+            handlers.append(logging.StreamHandler())
+
+        log = logging.getLogger()
+        log.setLevel(level)
+        for h in handlers:
+            h.setFormatter(logging.Formatter(
+                "%(asctime)s %(process)d %(levelname)s %(message)s"))
+            log.addHandler(h)
+
+    def check_pid(self):
+        """Check the pid file.
+
+        Stop using sys.exit() if another instance is already running.
+        If the pid file exists but no other instance is running,
+        delete the pid file.
+        """
+        if not self.pidfile:
+            return
+        # based on twisted/scripts/twistd.py
+        if os.path.exists(self.pidfile):
+            try:
+                pid = int(open(self.pidfile).read().strip())
+            except ValueError:
+                msg = 'pidfile %s contains a non-integer value' % self.pidfile
+                sys.exit(msg)
+            try:
+                os.kill(pid, 0)
+            except OSError, (code, text):
+                if code == errno.ESRCH:
+                    # The pid doesn't exist, so remove the stale pidfile.
+                    os.remove(self.pidfile)
+                else:
+                    msg = ("failed to check status of process %s "
+                           "from pidfile %s: %s" % (pid, self.pidfile, text))
+                    sys.exit(msg)
```

```
+            else:
+                msg = ('another instance seems to be running (pid %s), '
+                       'exiting' % pid)
+                sys.exit(msg)
+
+    def check_pid_writable(self):
+        """Verify the user has access to write to the pid file.
+
+        Note that the eventual process ID isn't known until after
+        daemonize(), so it's not possible to write the PID here.
+        """
+        if not self.pidfile:
+            return
+        if os.path.exists(self.pidfile):
+            check = self.pidfile
+        else:
+            check = os.path.dirname(self.pidfile)
+        if not os.access(check, os.W_OK):
+            msg = 'unable to write to pidfile %s' % self.pidfile
+            sys.exit(msg)
+
+    def write_pid(self):
+        """Write to the pid file"""
+        if self.pidfile:
+            open(self.pidfile, 'wb').write(str(os.getpid()))
+
+    def remove_pid(self):
+        """Delete the pid file"""
+        if self.pidfile and os.path.exists(self.pidfile):
+            os.remove(self.pidfile)
+
+
+def get_uid_gid(cp, section):
+    """Get a numeric uid/gid from a configuration file.
+
+    May return an empty uid and gid.
+    """
+    uid = cp.get(section, 'uid')
+    if uid:
+        try:
+            int(uid)
+        except ValueError:
+            # convert user name to uid
+            try:
+                uid = pwd.getpwnam(uid)[2]
+            except KeyError:
+                raise ValueError("user is not in password database: %s" % uid)
+
```

```
+    gid = cp.get(section, 'gid')
+    if gid:
+        try:
+            int(gid)
+        except ValueError:
+            # convert group name to gid
+            try:
+                gid = grp.getgrnam(gid)[2]
+            except KeyError:
+                raise ValueError("group is not in group database: %s" % gid)
+
+    return uid, gid
+
+
+def daemonize():
+    """Detach from the terminal and continue as a daemon"""
+    # swiped from twisted/scripts/twistd.py
+    # See http://www.erlenstar.demon.co.uk/unix/faq_toc.html#TOC16
+    if os.fork():   # launch child and...
+        os._exit(0) # kill off parent
+    os.setsid()
+    if os.fork():   # launch child and...
+        os._exit(0) # kill off parent again.
+    os.umask(077)
+    null=os.open('/dev/null', os.O_RDWR)
+    for i in range(3):
+        try:
+            os.dup2(null, i)
+        except OSError, e:
+            if e.errno != errno.EBADF:
+                raise
+    os.close(null)
diff -NurpP ezvz_org/backend/server.py ezvz/backend/server.py
--- ezvz_org/backend/server.py 2007-02-12 23:56:22 +0200
+++ ezvz/backend/server.py 2007-02-12 23:57:25 +0200
@@ -17,6 +17,9 @@
 #   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111, USA.
  ############################################################################

+import daemon
+import logging
+import time
 import ezvzlib
 import os
 import sys
@@ -251,10 +254,19 @@ class EzVZbackend:
  def vps_destroy(self, vpsid):
   return ezvzlib.vps_destroy(vpsid)
```

```
-from SimpleXMLRPCServer import SimpleXMLRPCServer
-print "\n\nEasyVZ " + version +  ", (c) 2006, Binary Karma.\n(c) 2006, Shuveb Hussain
<shuveb@binarykarma.com>.\nhttp://www.binarykarma.com\n"
-server = SimpleXMLRPCServer(("localhost", 8086))
-server.register_instance(EzVZbackend())
-print "Waiting for client..."
-server.serve_forever()
+class easyvzDaemon(daemon.Daemon):
+    default_conf = '/etc/easyvz.conf'
+    section = 'easyvzd'
+
+    def run(self):
+ logging.info('\n\nEasyVZ " + version +  ", (c) 2006, Binary Karma.\n(c) 2006, Shuveb Hussain
<shuveb@binarykarma.com>.\nhttp://www.binarykarma.com\n')
+ logging.info('Waiting for client...')
+        from SimpleXMLRPCServer import SimpleXMLRPCServer
+        server = SimpleXMLRPCServer(("localhost", 8086))
+        server.register_instance(EzVZbackend())
+ server.serve_forever()
+
+if __name__ == '__main__':
+    easyvzDaemon().main()
+

diff -NurpP ezvz_org/easyvz.conf ezvz/easyvz.conf
--- ezvz_org/easyvz.conf 1970-01-01 03:00:00 +0300
+++ ezvz/easyvz.conf 2007-02-12 22:32:45 +0200
@@ -0,0 +1,6 @@
+[easyvzd]
+uid =
+gid =
+pidfile = /var/run/easyvzd.pid
+logfile = /var/log/easyvzd.log
+loglevel = info
```