OpenVZ

Hey Python Hackers!

This project is currently in development. Please see the Getting Started section for more information about getting involved.
Weekly meetings for this project happen over **IRC** in **#openvz** on **Wednesdays @ 11pm EST** and **Saturday & Sunday @ 5pm EST**

One Name, Many Projects

The Crew OpenVZ project acts as an umbrella title for many projects working towards the same goal: virtualized Linux environments for CCIS students. This project is not to be confused with the efforts of the developers of the OpenVZ project. The work they have done is the core to our solution; we are merely building a management layer on top of it.

Crew OpenVZ is broken down into three sub-projects. Each has a specific role in the solution.

- If you need a new slice, talk to the The Pizza Man
- If you need to manage multiple slave machines hosting those slices, you need the The Middle Man
- If you want to programatically speak with OpenVZ and create slices, everything rests on Atlas

Getting Started

Phase One: Gather Information

Before you dive head first into this project, we suggest you do your reading and research. All pieces of the Crew OpenVZ project are written in the **Python programming language**. If you are unfamiliar with it, please see Resources. The project also relies heavily on the **OpenVZ project**. Reading about their work and the tools they have developed is essential. Each sub-project will have its own set of required reading, so remember to check their pages as well.

Phase Two: ???

Phase Three: Code

If you have read through the pages and feel you have a good grasp on the project, get involved. We currently use git to distribute our source code for the projects. **The repository can be found here**. We are also trying to systematically manage tasks for the project in an attempt to keep the project focused. We are using the Jutdah Helpdesk as a hybrid bug and task tracking system. We currently have two ticketing queues, one for django-openvz and one for openvz-daemon. **Ticketing can be found here**.

The Rest

Below are some stubs for possible future subpages. The information within them is relevant and should be read. If you have more to add to these, please take the initiative to create a full page.

Development Environments

As of now, development of Crew OpenVZ tools on personal machines is recommended. OpenVZ is a custom kernel, which applies patches to the normal Linux kernel, and as such, is easier to manage by normal users on their own hardware than that of CCIS. However, we do have two machines set aside for the purposes of development. One machine is **blaine.ccs.neu.edu**, and the other is **criss.ccs.neu.edu**. (Bonus points for noting the naming convention)

Blaine is set up as both a master machine, as well as a slave machine. It is running the last stable version of OpenVZ, as well as Django. It has the previous snapshot of django-openvz running on it, which can be accessed through a web browser. Blaine contains some development slices for people in Crew for various applications.
Criss is set up as a slave machine only. In that aspect, it is a clone of Blaine. No applications are deployed on this machine.

The goal is to have these machines sandboxed, allowing for openvz-daemon, openvz.py and django-openvz development. With the way these tools can drastically change the environment they inhabit, providing partitions between the files each touches is key. My gut feeling is that this will be solved with OpenVZ slices, each slice running OpenVZ itself.

History

Knowing where you have been can only help guide you to where you are going.
Crew OpenVZ started as a small project to add Ruby on Rails to the CGI Box, back in late Fall 2008. Ruby on Rails seemed like the new programming language framework of choice, but it was slowly revealed people wanted more than just Ruby. In addition to that, securely managing applications developed by students started to look like a nightmare, with no easily implementable solution in sight. The OpenVZ project was suggested as a solution to both problems, being both secure and highly configurable.

CGI is an old paradigm in today's database-backed, framework-driven web. The CGI Box is limited, because well, you can't log in into it for security reasons. With virtualization, each user will get a machine to themselves with which they could potentially launch any framework in a secure manner.

django-openvz steps in to provide a web-based management system for virtualization. With django-openvz, students in the college of CCIS can register for access to a virtual environment hosted on one of many slave machines located in the college, and do as they please (with limits).

django-openvz is actually an umbrella term for two main components: django-openvz, a web application written using the Djagno framework for managing machines and virtual environments, and openvz-daemon, a C daemon which runs on machines to connect the management software to machines being managed.

Key Terms

Before continuing, some key terms should be identified and defined to save Google some bandwidth. (These are interpretations of real definitions)

Virtualization	Using hardware and specialized software to run multiple instances of operating systems
OpenVZ	An open-source software providing virtualization. Uses Virtual Environments.
Virtual Environment	A sectioned off set of resources which work together with the virtualization software to mimic a machine.

django-openvz

openvz-daemon

openvz-daemon is a set of tools written in C to provide a layer of communication for the central host machine and each slave machine running OpenVZ.

Originally, communication was achieved over SSH. The host machine would have a private key for a particular user on each slave machine, with the matching public key in the `accepted_keys` file for the user on the slave. Thus, the host machine could masquerade as the user on the slave, issuing commands as the user to the slaves. However, due to the superuser requirements of the commands needed to be executed, many commands were added to the `sudoers` file for that user. To make things simpler for communication, prompts were to never be expected over the SSH link, and thus, these commands were given the `nopasswd` flag in the `sudoers` file. This essentially created a bloated `sudoers` file for all the small commands needed, and a possible security hole on all slave machines, providing near root access if any private key from the host machine be compromised.

openvz-daemon addresses this issue. It runs as a daemon started from runlevel 0, allowing it access to superuser commands without bloating sudo. However, it only responds to a certain API, with commands issued to it from the host machine encryped with SSL. This means, if the SSL key for a slave machine is compromised, a malacious user can only perform a small set of tasks on the slave machines, and never own the box in question.

Message Specification

The daemon responds to commands issued by the host machine sent to it over the internet encrypted in SSL. Certain tasks are performed on the machine running the daemon, and then information must be returned to the host machine. To do this, the JSON specification was chosen to encapsulate data being sent back and forth between the machines. To minimize any damage that can be done by malacious users, the daemon responds to a certain API written in JSON form. These specific forms are outlined below, with their effect on the machine running the daemon process, along with the content the machines return upon execution.

All commands sent to the daemon are of the following form:

```
{ 'command' : command, 'arguments' : [arg1, arg2,...] }
```

All commands return JSON of the following form:

```
{ 'code' : success_code, 'returned' : [return1, return2,...] }
```

where success_code is always 0 for a successful execution of the command, and non-zero for a failure; and returned is an array of optional elements which a command may return to the sender.
NOTE: If the execution fails, returned is always an array of string representing the STDERR.

The following table outlines all commands which can be sent to the daemon, any optional arguments to these commands, what the daemon then does on the server when it receives this command, and what, if any, is returned after execution.

Command	Arguments	Executed by Daemon	Returned

create_env	id - The id of the environment to make template - A template file on the remote machine to use for this new environment username - CCIS user associated with environment	vzctl create id --ostemplate template; Generate ssh key pair for username and copy public to environment Generates login shell file for username Adds username to /etc/passwd file Adds username mapping to Apache config vzctl start id;	N/A
sync_templates	N/A	Causes slave machine to check template files for changes and fetch updates from host machine	N/A
start_env	id - The id of the environment to start	vzctl start id	N/A
destroy_env	id - The id of the environment to remove	vzctl stop id; vzctl remove id	N/A

Web Hosting via Virtualization

CGI is an old paradigm in today's database-backed, framework-driven web. The CGI Box is limited, because well, you can't log in into it for security reasons. With virtualization, each user will get a machine to themselves with which they could potentially launch any framework in a secure manner.

Parts

1. OpenVZ
2. Apache + DNS
3. Administration of #1 and #2
4. Default images/templates for the VE's in OpenVZ

TODO

1. Administration of OpenVZ
    - How would users request a VE?
    - How would the creation process of a VE be automated?
    - quotas?
2. Default templates
    - requirements?
3. firewall to close all ports except 22 and 80 (443)

OpenVZ

Installation

On Ubuntu,

```
sudo apt-get install linux-openvz vzctl vzquota
```

linux-image-openvz is the precompiled kernel metapackage.
vzctl is the package for the OpenVZ control tool.
vzquota is the package for the OpenVZ quota management.

To build from source, click here.

To start, reboot into the new kernel.

Creating a Virtual Environment (VE)

```
sudo vzctl create <veid> --os-template <template_name> --ipadd <addr>
```

<veid> is the unique id for your VE.
<template_name> is the name of the template you want to use. All templates should reside in [/var/lib/vz/template/cache]. A template is a tar-gzip'ed file of a minimal / (root) directory.
<addr> is an ip address

Configuring the VE

```
sudo vzctl set <veid> --nameserver <addr> --searchdomain <domain> --save
```

<domain> should be equivalent to the search parameter in resolv.conf

```
sudo vzctl set <veid> --privvmpages P[:P]
```

where P is the number of pages (memory), the two numbers mean barrier:limit

/proc/user_beancounters

user_beancounters (or UBC) file is critical to how OpenVZ manages resources.

/proc/user_beancounters

```
Version: 2.5
       uid  resource           held    maxheld    barrier      limit    failcnt
        1:  kmemsize        1107385    2318349   11055923   11377049          0
            lockedpages           0          0        256        256          0
            privvmpages       41480      93087     131072     139264         36
            shmpages              6          6      21504      21504          0
            dummy                 0          0          0          0          0
            numproc              18         31        240        240          0
            physpages         12565      77587          0 2147483647          0
            vmguarpages           0          0      33792 2147483647          0
            oomguarpages      12742      77765      26112 2147483647          0
            numtcpsock            3          5        360        360          0
            numflock              3          9        188        206          0
            numpty                1          2         16         16          0
            numsiginfo            0         11        256        256          0
            tcpsndbuf         26880      64960    1720320    2703360          0
            tcprcvbuf         49152     397248    1720320    2703360          0
            othersockbuf       8960      16640    1126080    2097152          0
            dgramrcvbuf           0       8384     262144     262144          0
            numothersock          5          7        360        360          0
            dcachesize        96520     114588    3409920    3624960          0
            numfile             438        581       9312       9312          0
            dummy                 0          0          0          0          0
            dummy                 0          0          0          0          0
            dummy                 0          0          0          0          0
            numiptent            10         10        128        128          0
```

The default setting was not enough to run rails, as you can see the failcnt at 36 for privvmpages.

Network Address Translation (NAT)

On the host machine,

```
sudo iptables -A POSTROUTING -t nat -s 192.168.1.0/24 -o eth0 -j MASQUERADE
```

In this example, the VE's are located on the private network 192.168.1.0/24. You can set up ip addresses of a running VE by

```
sudo vzctl set <VE_id> --ip <ip_address> --save
```

A VE is running at 192.168.1.2 and the HN is set to 192.168.1.1, any packets sent by the VE will be broadcast to venet0 (192.168.1.1/24) on the HN. Then, the broadcast address was set on venet0 to the ip of the physical eth0.

/etc/network/interfaces

```
iface venet0 inet static
        address 192.168.1.1
        netmask 255.255.255.0
        broadcast <ip_address of eth0>
```

You also need to change the kernel configuration. http://wiki.openvz.org/Quick_installation#sysctl

DNS inside the Virtual Environment

Edit /etc/resolv.conf to something like

/etc/resolv.conf from utopia

```
domain ccs.neu.edu
nameserver 129.10.116.80
nameserver 129.10.116.51
```

Apache

openvz.map

```
ve1    192.168.1.2
```

```
RewriteEngine on
RewriteMap openvz txt:/etc/apache2/openvz.map

RewriteCond %{HTTP_HOST} ^(.*)\.example\.com$
RewriteRule ^/(.*) http://${openvz:%1}/$1 [P]
```

If the client asks for ve1.example.com, the RewriteCond parses the subdomain (ve1), and then the RewriteRule searchs the RewriteMap to get a proxy ([P] is a proxy flag) to the internal ip address.

DNS

The Hardware Node will get *.<domain> and Apache will handle the *.

Web Sign-up and Automated Administration

Accessing Virtual Environments

Due to the environments being hidden behind NAT, users have no direct way to access their boxes from the outside. The only box who knows about the environments is blaine, so naturally, users should access blaine first to then access their individual environments. However, we do not want users to be able to mess around on blaine, nor do we want them to have to perform two ssh commands, remembering ofcourse the private IP address of their environment, just to do work. We want users to ssh blaine, but be transparently redirected to their own environments. This can be accomplished with the /etc/passwd file on blaine.

The cron script, upon sucessful sign-up, peforms the following:

```
$ echo "#!/bin/sh\nssh <username>@<private-ip>" > /home/users/<username>.sh
$ chmod +x /home/users/<username>.sh
$ useradd -s !$ <username>
```

When the user logs into blaine, <username>.sh will be executed, silently forwarding them to their own machine. When they are finished on their machine, the command exits, and they are automatically logged out of blaine.
Getting Started

Useful Links

1. Ubuntu OpenVZ Docs
2. Setting up a NAT
3. Installation of OpenVZ
4. iptables HOWTO
5. template pty problem