Isolierte Umgebungen für Python Anwendungen – Teil 2

Christine Koppelt, Jürgen Schackmann, Stefan Seelmann

Grenzen von virtualenv

- Anwendungen bestehen neben Python aus einer Reihe weiterer Komponenten (zusätzliche System-Bibliotheken, Suche, Datenbank, etc.)
  - Das Entwicklungs- /Serversystem soll „sauber“ bleiben, möglichst wenig Software soll dort installiert werden
  - Die Infrastruktur einer Anwendung soll auf mehreren Umgebungen reproduzierbar sein (Dev, Staging, Prod)
  - Anwendungen auf einem Server sollen sich nicht gegenseitig beeinflussen

Zwei Beispiele

- Linux Containers (LXC)
  - Isolation der Linux Systembibliotheken (Systemvirtualisierung)
- VirtualBox + Vagrant
  - Isolation des kompletten Betriebssystems (Hardwarevirtualisierung)

LXC - Anwendungsfälle

- Systemcontainer:
  - Verschiedene Anwendungen benötigen unterschiedliche Systembibliotheken
  - Nested Virtualisierung (LXC unter XEN)
  - Sandboxing zum Ausprobieren von Software
- Applikationscontainer:
  - Limitierung der Ressourcen einzelner Anwendungen

LXC - Eigenschaften

- Vollständiges Linux System innerhalb des Containers
  - oder auch nur einer Applikation (Applikationscontainer)
- Container := Konfigurationsfile + Root Filesystem
- Kein eigener Kernel, keine eigenen Kernel Module
- Isolierung mittels Kernel Namespaces
- Ressourcenmanagement mittels cgroups
- Eigener Netzwerkstack, eigener Prozessraum

LXC - Einrichtung

- Nicht trivial:
  - cgroups, Kernel Parameter für Speicherlimitierung
  - Netzwerkstack (bridge)
  - Konfigurationsdatei
  - Entzug von Capabilities
  - Anpassung des Gastsystems (udev, proc, hwclock, fstab)
- Mit Ubuntu 12.04 ist es recht einfach

LXC – Demo mit Ubuntu I

- Installation
  - `# apt-get install lxc`
- Wo liegt was?
  - `# ls /var/lib/lxc/`
  - `# ls /var/cache/lxc/`
  - `# ls /etc/lxc`
  - `# ls /usr/lib/lxc/templates/`

LXC – Demo mit Ubuntu II

- Wichtige Befehle
  - `# lxc-checkconfig`
  - `# lxc-ls`
  - `# lxc-create`
  - `# lxc-destroy`
  - `# lxc-start`
  - `# lxc-stop`
  - `# lxc-info`
  -

LXC – Demo mit Ubuntu III

- Erzeugen
  - `# lxc-create -n DEMO -t ubuntu`
- Starten
  - `# lxc-start -n DEMO -d`
  - `# lxc-info -n DEMO`
  - `# lxc-console -n DEMO`

Zusammenfassung LXC

- Weniger Overhead als Hardware-Virtualisierung
- System- oder Applikationscontainer
- Nur Linux unter Linux
- Kopieren von Containern funktioniert nur auf andere Linux-Systeme mit kompatiblem Kernel und Architektur
- Konfiguration (cgroups, capabilities) ist entscheidend
- Noch nicht „Enterprise Ready“ (Security, Live-Migration)
  - Management mittels OpenStack oder libvirt/virsh?

LXC – Resourcen

- LXC
  - http://lxc.sourceforge.net/
- Ubuntu
  - https://help.ubuntu.com/12.04/serverguide/lxc.html
- Vortrag zu LXC und cgroups (2h)
  - http://www.youtube.com/watch?v=wzrd5nSyo0o

Hardware Virtualisierung

- Vollständiges Betriebssystem wird „in einer Box“ installiert
  - Inklusive „Kernel“
  - Andere Typen und Architekturen sind möglich
- Gesamte Hardware wird virtualisiert – Hypervisor
- Kopieren auch auf Systeme mit anderen Plattformen und Betriebssystem
- Höherer Ressourcenverbrauch als LXC
- Beispiel: VirtualBox (+ Vagrant)

VirtualBox

- Wird als Anwendung auf einem Host-System installiert
  - Typ2 Hypervisor
- Von Oracle vermarktet als „Desktop-Virtualisierung“
  - VMs können graphisch oder geskriptet erstellt werden
  - Headless Mode
- Etwas langsamer als andere Hardware-Virtualisierungen (KVM)

Vagrant

- Wrapper um VirtualBox, unterstützt beim Anlegen und verwalten von virtuellen Umgebungen
- Anwendungsfall: Schnelles und reproduzierbares Setup von Entwicklungsinfrastruktur
  - Systembibliotheken, Datenbanken, Web Server, ...

Vagrant Image wird erstellt aus

- Einer Vagrant-Box
  - VirtualBox Image mit dem Basis Betriebssystem
  - Vorinstalliert sind
    - VirtualBox Guest Additions
    - Ruby
    - Puppet und Chef
    - SSH (Default-User, Default-Passwörter, Default SSH-Key)
- Konfigurationsfile (Vagrantfile)
- Anwendungsspezifische Konfiguration für Chef oder Puppet

Vorteile von Vagrant

- Die Entwicklung mit einer produktionsähnlichen Infrastruktur wird vereinfacht
- Es müssen keine kompletten Images weitergegeben werden, sondern nur die Konfiguration
  - Updates der Infrastruktur können durch Updates der Konfigurationsfiles an die Entwickler verteilt werden

Verwendung von VirtualBox/Vagrant

- Installation
  - VirtualBox, Ruby, RubyGems, Vagrant
- Download einer BasisBox
  - Alternativ: Box selbst erstellen
- Box bei lokaler Vagrant Installation registrieren
  - `# vagrant box add precise32 precise32.box`

Setup einer VirtualBox/Vagrant Umgebung

- Neues Verzeichnis anlegen
  - Vagrant Projekt auschecken oder
  - Vagrantfile erstellen
    - `# vagrant init <boxname>`
- Virtuelle Maschine starten
  - `# vagrant up`
- Aktualisieren der Konfiguration
  - `# vagrant provision`

Vagrant – Was liegt wo?

- Vagrant Base Boxes:
  - ~/vagrant.d/boxes
- Projektdateien (VagrantFile, Manifests)
  - beliebig
- VM Instanzen:
  - Im VirtualBox Machine Folder: ~/.VirtualBox/Machines