Fanery Documentation

Release 0.2.5

Marco Caramma

January 07, 2015

Contents

Contents:

CHAPTER 1

Fanery Framework

An opinionated application development framework.

1.1 Project Goals

- Strong security by default.
- Focus on being developer-oriented.
- Promote funcional pythonic style.
- Promote continuous testing+profiling.

1.2 Install

1. First make sure to install successfully the following C libraries:

```
pip install PyNaCl
pip install cxor
pip install ujson
pip install scrypt
pip install bjoern
pip install bsdiff4
pip install ciso8601
pip install python-libuuid
pip install msgpack-python
pip install linesman objgraph
```

2. Then install Fanery and run test files:

```
pip install Fanery
python tests/test_term.py
python tests/test_service.py
```

1.3 Contribute

- Issue Tracker: https://bitbucket.org/mcaramma/fanery/issues
- Source Code: https://bitbucket.org/mcaramma/fanery/src

1.4 License

The project is licensed under the ISC license.

CHAPTER 2

Before we start

This documentation is a work-in-progres ... before anything I must warn the reader that I'm not a native English speaker, my apologies in advance for all mistakes.

CHAPTER 3

Why Fanery

Fanery is the result of several years struggling around a few basic questions:

- How to build Software that really fulfill end-users expectations?
- How to build Software that really scale?
- How to build Software that is really secure?
- How to build Software at scale that can be managed successfully by a very small team?

Answers came after years of experimentation and failures, but at the end a simple method emerged from chaos. That simple method of Software development is the reason Fanery exists the way it is.

3.1 Fulfill end-users expectations

There are infinites ways to build Software, but one easy and cost effective way to create understanding and agreement around a Software project is enlightning user's expectation with the aid of Story Boards, sequences of pictures representing front-end interfaces and processes workflows through their interactions.

Once a Story Board get approved it means no misunderstanding still left about how the final product must look like and what it should do. Most important no single line of code is written at this stage, nor time is wasted choosing/arguing about which programming language, storage strategy, database engine, UI toolkit, operating system, etc, should be employed.

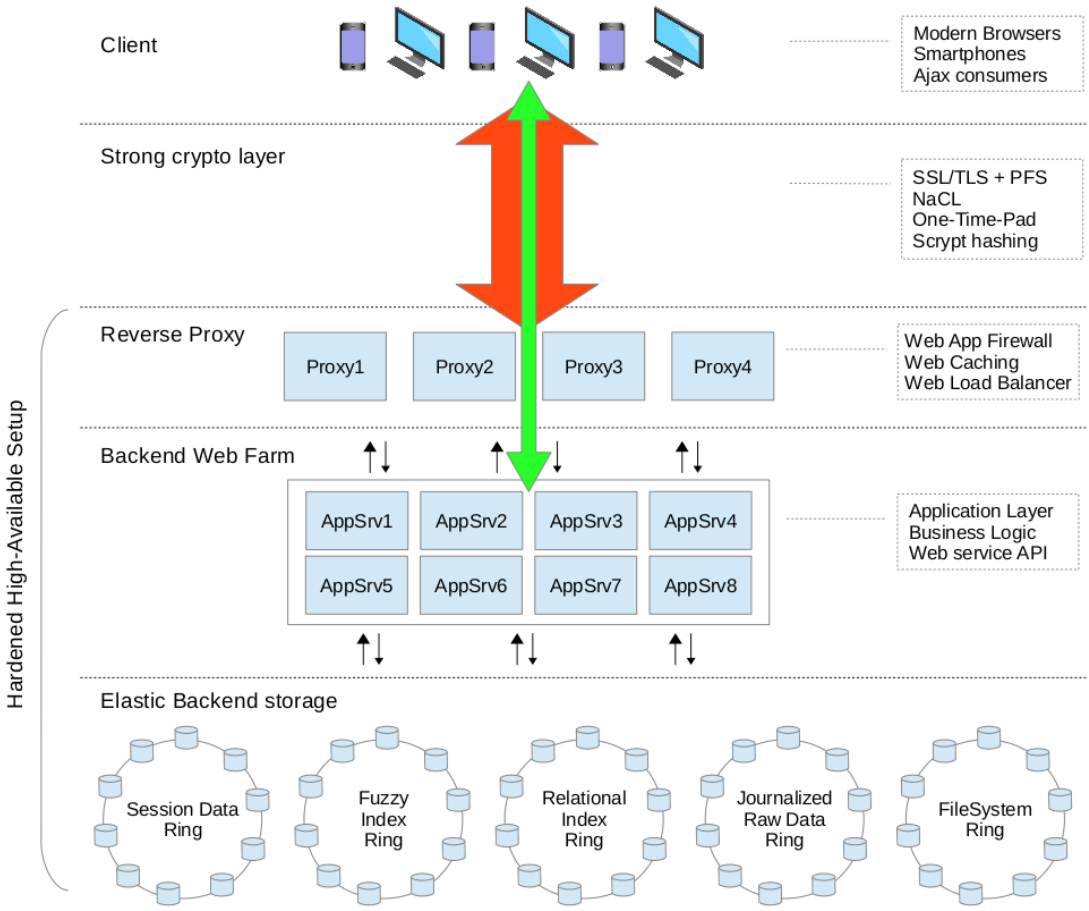
A team of interdiciplinary experts in the field of graphic design, human resources and psycology is recommended as usually end users don't know exactly what they want or have issues expressing/clarifying what they need.

3.2 Build Software that scale

Scalability is a difficult, a very tough problem to solve and deal with; no final answer exists however lots of clever solutions have been proposed and something many of them have in common is minimization of complexity.

Building Software architectures where logical elements are loosely coupled is a first step in that direction.

Fanery is designed around the idea that the following architecture is one flexible, valuable and real solution to the scalability problem and the framework should make very easy to build Software for it.



The *Application Layer* is where the real *product* lives and the core point is:

> It should not care about front-end technologies or transport channels nor about storage strategies and database engines; it should just focus on the core business logic, processes and functionalities.

This simple idea unleash the freedom to switch at will all supporting pieces to our *product*, letting this way, load/stress-test different storage and indexing engines, UI toolkits, filesystems, messages queues, etc ... to consciously peak the bests for the job, taking decisions based on measurable and reproducible results.

Easy debugging, testing and profiling of every single line of written code is another important aspect of true scalability;

> Slow unoptimized sloppy code doesn't scale.

3.3 Build Software that is really secure

Fanery approach to security is:

- Strong cryptography must be transparent and enabled by default.

  Building secure Software is hard, understanding cryptography is harder.

  Fanery pretend to provide transparent cryptography done correctly, trying hard to make it developer friendly (easy and unobtrusive).

- Encryption must not rely on cryptographic keys generated client side.

Mistrust is a fundamental aspect of Software security, Fanery assume client-side cryptography is weak and for such reason all key-pairs must be generated exclusively server side by proper trong cryptographic primitives.

- Encryption must only rely on unbroken high-quality ciphers/algorithms/implementations.

  Fanery crypto strategy is build on top of NaCL (libsodium), Scrypt and One-Time Pad.

- Session security must not rely on SessionIDs, bizare URLs, secure cookies, secret tokens, magic keys or any other piece of information that can be guessed or stolen during transmission.

  Authentication and session security must stand even when security weakness are presents in SSL/TLS.

- Capture and re-transmission of encrypted messages must be pointless.

  Every encrypted message is unique and unforgeable, build with cryptographically secure random key-pairs and sign-keys that are destroyed after transmission.

- Transparent protection against brute-force and authenticated sessions abuse.

  Fight agains authentication abuse, message forging, unauthorized access, session hijacking, priviledge escalation, CSRF, MitM, RFD, XSS, etc.

- Transparent (de)serialization to harmless/built-in only object types.

  Automatic input/output (de)serialization help dramatically during development, avoiding boilerplates code and hacky constructs, but doing it correctly is not easy; Software development and frameworks have been historically plagued by critical security issues related to data (de)serialization.

- Carefull urlpaths validation and sanitization.

  Prevent directory traversal, file inclusion and similar attacks.

The described approach offer a solid fundation to build secure Software that stand against the majority of common attacks.

3.4 How to build Software at scale that can be managed successfully by a very small team?

If we forget about end-user support team for a moment and just focus on DevOps responsibility, a squeezed and over-simplified answer may be:

> Build Software on top of an hardened, massively scalable, almost zero configuration, shared-nothing architecture that can be fully understood by a single person.

That's a dreamed scenario, that may sound absurd or impossible to achieve, however it's not that crazy if we manage to remove all unecesary complexity from the full picture.

This guide pretend to show and explain one cost-effective way to build such architecture; of course there are many other ways, but just for a moment try to forget about current FUD, hype and greedy vendors *"best practice"*.

3.5 Disclaimer

The choice of third-party FLOSS tools, programs and libraries is deliberately subjective, based on my personal experience and taste.

Every decision is always influenced by:

1. *Costs*: we are in a limited budget.
2. *Security*: security garanties must be preserved.
3. *Scalability*: the solution must be truly elastic.
4. *Flexibility*: every single piece should be replaced with easy.
5. *Easy of management*: a single person must be able to hadle it.

CHAPTER 4

Hardened System

Hardened Systems are environments that focus on high security and reliability at all levels they can control.

Let's start our journey building the basement that will support our elastic Software solution.

4.1 Physical environment

The market is rich in alternatives, looking for private cloud virtual machines helps keeping the budget low without sacrifing to much freedom. While choosing you may consider observing the following guidelines:

- Stick to Tier 3/4 companies that offers multiple datacenter options, in multiple geo-locations/continents.
- Make sure to get support for backup/snapshots, unmetered private LAN between VMs, HTTP(S) load balancing and truthful statistics about bandwidth usage.
- Be sure that you get full root access and they let you install customized kernels, or even better, they let you start a fresh minimal install from official netinstall ISO.
- Use NetCraft data and investigate deeply the Web, looking for satified and disatisfied customers.
- Prefers proven FLOSS virtualization technologies like KVM or XEN.

4.2 Automation

From now on installation and configuration steps are presented as shell commands and scripts for the purpose of automation and easy of management.

Intermediate knowledge of Debian GNU/Linux and Shell language is required.

4.3 Operating System

The objective here is walking through the hardening process of Debian GNU/Linux 7 (Wheezy).

4.3.1 Netinstall

As starting point this documentation suppose that a fresh new install has been successfully completed following this criterias:

1. Shadow password: enabled.
2. Root login: disabled.
3. Disk partitioning:

```
PRIMARY     1GB     ext4            /boot
PRIMARY     32GB    LVM     vg0
PRIMARY     *       LVM     vg1

LogVolume   1GB     xfs     vg0     /
LogVolume   8GB     xfs     vg0     /usr
LogVolume   8GB     xfs     vg0     /var
LogVolume   8GB     xfs     vg0     /var/log
LogVolume   2GB     xfs     vg0     /tmp
LogVolume   4GB     swap    vg0
LogVolume   *       xfs     vg0     /home
```

   Partitioning is actually a matter of taste, feel free to perform a different style of formatting.

4. Additional packages: none.
5. Set man suid: false.

4.3.2 System upgrade

1. Setup APT mirrors list.

```
cat > /etc/apt/sources.list <<EOF
deb http://ftp.debian.org/debian/ wheezy main
deb http://security.debian.org/ wheezy/updates main
deb http://ftp.debian.org/debian/ wheezy-updates main
deb http://ftp.debian.org/debian/ wheezy-proposed-updates main
deb http://ftp.debian.org/debian/ wheezy-backports main
# iptables SYNPROXY target
deb http://ftp.debian.org/debian/ testing main
EOF

cat >> /etc/apt/preferences << EOF
Package: *
Pin: release a=stable
Pin-Priority: 700

Package: *
Pin: release a=wheezy-backports
Pin-Priority: 650

Package: *
Pin: release a=testing
Pin-Priority: 600
EOF
```

2. System upgrade.

```
aptitude update
aptitude full-upgrade
```

3. Adminitration tools and usefull utils that may not be present after default install.

```
aptitude -t wheezy-backports install bzip2 less tmux lsof htop nmon dnsutils iputils-ping te
```

4.3.3 Kernel upgrade

1. Kernel >= 3.12 and iptables >= 1.4.21 are required for SYNPROXY support.

   ```
   aptitude -t wheezy-backports install linux-image-amd64 linux-headers-amd64
   ```

2. Reboot with new Kernel.

   ```
   reboot
   ```

4.3.4 Firewall setup

1. Install required packages.

   ```
   aptitude -t testing install xtables-addons-dkms iptables netsniff-ng
   aptitude -t wheezy-backports install ca-certificates ethtool sed wget awk ipset ipcalc geoip
   ```

2. Download advanced iptables script with integrated sysctl tuning and hardening.

   ```
   wget -c -O /sbin/firewall https://bitbucket.org/mcaramma/linux-setup/raw/master/firewall-syn

   chmod 500 /sbin/firewall
   ```

3. Download blocklists, build geoip database and load iptables rules (will take a few minutes).

   ```
   /sbin/firewall force-load
   ```

The proposed firewall script may seams intimidating at first but it's actually well organized and self explanatory, please take the time to study its internals and enjoy the simplicity; it shows howto:

- Blacklist Anonymous Proxies and Satellite Providers + top sources of internet attacks.
- Blacklist bogon/hijacked/infected/abusive hosts.
- Discard invalid/unwanted packets.
- Tarpit/slow-down spammers.
- Delude PortScan attempts.
- Prevent ssh brute-force.
- Mitigate DDoS attacks.
- Tune and harden TCP/IP Kernel Stack.

It's also important to stress that the script try hard to create a minimal amount of iptables rules for the job.

4.3.5 DNSCrypt-Proxy

DNS attacks can easily turn our hardening efforts useless, dnscrypt is one way to mitigate most common DNS security threats.

1. Preparation.

   ```
   aptitude -t wheezy-backports install build-essential checkinstall wget bzip2
   ```

2. Build libsodium deb package.

```
cd /usr/src
mkdir libsodium && cd libsodium
wget -c https://github.com/jedisct1/libsodium/releases/download/1.0.0/libsodium-1.0.0.tar.gz
tar -zxvf libsodium-1.0.0.tar.gz
cd libsodium-1.0.0
./configure
make
checkinstall --nodoc
ldconfig -v
```

3. Build dnscrypt-proxy deb package.

```
cd /usr/src
mkdir dnscrypt && cd dnscrypt
wget -c https://github.com/jedisct1/dnscrypt-proxy/releases/download/1.4.1/dnscrypt-proxy-1.
tar -jxvf dnscrypt-proxy-1.4.1.tar.bz2
cd dnscrypt-proxy-1.4.1
./configure
make
checkinstall --nodoc
```

4. Add dnscrypt user & init script.

```
adduser --system --quiet --shell /bin/false --group --disabled-password --disabled-login --h
wget -O /etc/default/dnscrypt-proxy -c https://raw.githubusercontent.com/jedisct1/dnscrypt-p
wget -O /etc/init.d/dnscrypt-proxy -c https://raw.githubusercontent.com/jedisct1/dnscrypt-pr
sed -i -e '/\(\/usr\)\(\/sbin\/dnscrypt-proxy\)/ s//\1\/local\2/g' /etc/init.d/dnscrypt-prox
sed -i -e 's/127\.0\.0\.2/127\.0\.0\.1/g' /etc/default/dnscrypt-proxy
chmod 550 /etc/init.d/dnscrypt-proxy
update-rc.d dnscrypt-proxy defaults
```

5. Set resolv configuration.

```
cat > /etc/resolv.conf <<EOF
search $(hostname -d)
nameserver 127.0.0.1
EOF
```

6. Test dnscrypt-proxy name resolution.

```
/etc/init.d/dnscrypt-proxy start
nslookup bitbucket.com
```

4.3.6 Miscelaneous hardening and help scripts

1. Disable core dumps.

```
echo 'fs.suid_dumpable = 0' >> /etc/sysctl.conf
echo '* hard core 0' >> /etc/security/limits.conf
echo 'ulimit -S -c 0 > /dev/null 2>&1' >> /etc/profile
```

2. Decrease per process/thread stack size

```
echo '* soft stack 1024' >> /etc/security/limits.conf
echo '* hard stack 2048' >> /etc/security/limits.conf
```

3. Disable CTRL+ALT+DEL reboot secuence.

```
sed -i -e '/^\(ca:.*:ctrlaltdel:.*\)/ s//#\1/' /etc/inittab
```

4. Disable crontab for non-root users.

```
echo ALL > /etc/cron.deny
echo root > /etc/cron.allow

chown root.root /etc/cron.{allow,deny}
chmod 444 /etc/cron.{allow,deny}
```

5. Stop using root.

```
# Disable local root login
aptitude -t wheezy-backports install sudo
sudo passwd -l root

# Add user operador
useradd -p "*" -U -m operador -G sudo
passwd operador
chage -M 60 -m 7 -W 7 operador

# Add user sshadmin (remote login only via SSH key ... remember to copy your key with ssh-co
useradd -p "*" -U -m sshadmin
passwd sshadmin

# Disable remote root login
sed -i -e '/^PermitRootLogin .*/ s//PermitRootLogin no\nAllowUsers sshadmin/' /etc/ssh/sshd_
sed -i -e '/^#\(Banner .*\)/ s//\1/' /etc/ssh/sshd_config
service ssh restart

# Set motd/issue.net banner text
cat > /etc/motd <<EOF
Unauthorized access to this machine is prohibited
Press <Ctrl-D> if you are not an authorized user
EOF
cat /etc/motd > /etc/issue.net
chown root.root /etc/{motd,issue.net}
chmod 444 /etc/{motd,issue.net}
```

6. Build /sbin/lock-filesystem script.

```
cat > /sbin/lock-filesystem <<EOF
#!/bin/sh

[ -d /var/tmp ] && {
    rm -rf /var/tmp
    ln -s /tmp /var/tmp
}

chattr -R +i /boot /usr /bin /sbin /lib* /root /vmlinuz* /initrd* /etc 2> /dev/null
chattr -R -i /etc/adjtime /etc/blkid.tab /etc/mtab /etc/network/run /etc/udev/rules.d 2> /de

mount -o nosuid,noexec,nodev,remount /home
mount -o nosuid,noexec,nodev,remount /tmp
mount -o ro,nodev,remount /boot
mount -o ro,nodev,remount /usr
mount -o nodev,remount /
EOF
```

```
chmod 500 /sbin/lock-filesystem
```

7. Build /sbin/unlock-filesystem script.

```
cat > /sbin/unlock-filesystem <<EOF
#!/bin/sh

mount -o exec,remount /tmp
mount -o rw,remount /boot
mount -o rw,remount /usr

chattr -R -i /boot /usr /bin /sbin /lib* /root /vmlinuz* /initrd* /etc 2> /dev/null
EOF

chmod 500 /sbin/unlock-filesystem
```

8. Build /sbin/system-upgrade script.

```
cat > /sbin/system-upgrade <<EOF
#!/bin/sh

aptitude update && \\
    /sbin/unlock-filesystem && \\
    aptitude \${1:-safe}-upgrade && \\
    aptitude -f install && \\
    apt-get autoremove && \\
    apt-get autoclean && \\
    apt-get clean && \\
    /sbin/lock-filesystem
EOF

chmod 500 /sbin/system-upgrade
```

9. Build /sbin/lock-system script.

```
cat > /sbin/lock-system <<EOF
#!/bin/sh

/sbin/lock-filesystem
/sbin/firewall force-load
EOF

chmod 500 /sbin/lock-system
```

10. Activate /sbin/lock-system after boot.

```
sed -i -e 's/^\(exit 0\)/\/root\/lock-system\n\n\1/' /etc/rc.local
```

11. Remove unnecesary packages.

```
aptitude purge at nano tasksel tasksel-data task-english
```

    No other remotely accessible network service should stand active except ssh.

12. Final clean-up.

```
aptitude -f install
apt-get autoremove
apt-get autoclean
apt-get clean
```

```
rm -rf /tmp/*
rm -f /var/log/wtmp /var/log/btmp
history -c
reboot
```

4.4 Remark note

The proposed process is just the beginning, the first step to system hardening; a lot more can be done to strengh the security of a GNU/Linux system, like using a custom grsecurity patched kernel.

From now on every private virtual machine explained is implied to have gone through all the previously described hardening steps.

CHAPTER 5

Backend Web Farm

The web application we'll build with Fanery is going to run inside a restricted and replicable environment.

5.1 Restricted environment

Schroot is a tool that offers many additional functionalities not found in chroot.

Alternatives like LXC or Docker are also valid but we are already running inside a Xen/KVM virtualized environment with more isolation garanties; adding another virtualization layer won't be beneficial in term of performance or security.

5.2 Replicable environment

Our goal here is to build a restricted environment that provides the conditions for:

- quick and easy replication to several different private virtual machines.
- quick and easy switching between different versions of the same application.
- graceful code reload.

5.2.1 Schroot bootstrap script

First let's create a simple shell script that aid and speed-up the creation of schroot bootstrapped minimal Debian Wheezy environments.

1. Install required packages.

```
aptitude -t wheezy-backports install debootstrap schroot bzip2 cpio xz-utils
```

2. Change schroot default bind setup.

   Do not share /home and /tmp partitions with main system.

```
sed -i -e '/\(^\/home\)/ s//#\1/' -e '/\(^\/tmp\)/ s//#\1/' /etc/schroot/default/fstab
```

3. Build /sbin/bootstrap-schroot script.

```
cat > /sbin/bootstrap-schroot <<EOF
#!/bin/sh

[ \$# -lt 4 ] && {
    echo "Usage: \$0 SYSTEM VGNAME LVNAME LVSIZE [LVTYPE]"
    exit 1
}

SYSTEM=\$1
VGNAME=\$2
LVNAME=\$3
LVSIZE=\$4
LVTYPE=\${5:-xfs}

lvcreate -n \$LVNAME -L\$LVSIZE \$VGNAME && \\
    mkfs.\$LVTYPE -f /dev/\$VGNAME/\$LVNAME && \\
    mount -t \$LVTYPE /dev/\$VGNAME/\$LVNAME /mnt && \\
    debootstrap --include apt-utils,aptitude,locales \\
                --exclude tasksel,tasksel-data,nano,isc-dhcp-client \\
                --variant=minbase wheezy /mnt http://ftp.debian.org/debian || \\
    exit 1

grep -v ^mount /sbin/unlock-filesystem > /mnt/sbin/unlock-filesystem
grep -v ^mount /sbin/lock-filesystem > /mnt/sbin/lock-filesystem
echo "# schroot nssdatabases
chattr -i /etc /etc/passwd /etc/shadow /etc/group /etc/gshadow /etc/services /etc/protocols
" >> /mnt/sbin/lock-filesystem
chmod 500 /mnt/sbin/{lock,unlock}-filesystem
cp -a /sbin/system-upgrade /mnt/sbin/
cp -a /etc/apt/preferences /mnt/etc/apt/
grep -v testing /etc/apt/sources.list > /mnt/etc/apt/sources.list
cp -a /usr/src/libsodium/libsodium*/libsodium*.deb /mnt/usr/src/
cp -a /etc/skel /mnt/home/operador
chown -R operador.operador /mnt/home/operador

umount /mnt

cat >> /etc/schroot/schroot.conf <<EOC

[\$SYSTEM]
type=lvm-snapshot
users=operador
root-users=operador
source-root-users=operador
device=/dev/\$VGNAME/\$LVNAME
lvm-snapshot-options=--size 2G
EOC

schroot -c source:\$SYSTEM -u root --directory /root -- sh -c "
    dpkg-reconfigure locales
    dpkg-reconfigure tzdata
    aptitude update
    aptitude full-upgrade
    aptitude -f install
    apt-get autoremove
    apt-get autoclean
    apt-get clean
    rm -f /etc/ssh_host_*
```

```
    rm -rf /var/tmp /tmp/*
    ln -s /tmp /var/tmp
    rm -f /var/log/wtmp /var/log/btmp
    /sbin/lock-filesystem
    history -c"
EOF

chmod 500 /sbin/bootstrap-schroot
```

5.2.2 Build Fanery compressed archive

Now it's time to build our fanery node base compressed archive, a single xzipped files that we'll be able to replicate as many times as required.

1. Bootstrap Wheezy minbase.

```
/sbin/bootstrap-schroot wheezy-fanery VG-NAME LV-NAME 1G
```

2. Enter schroot in write mode.

```
schroot -c source:wheezy-fanery -u root
```

3. Install packages required to build Fanery dependencies.

```
aptitude install build-essential pkg-config graphviz-dev uuid-dev libffi-dev libev-dev pytho
dpkg -i /usr/src/libsodium*.deb
ldconfig -v
```

4. Install Python virtualenv and wrappers.

```
aptitude -t wheezy-backports install python-setuptools git
easy_install pip
pip install setuptools --no-use-wheel --upgrade
pip install virtualenv virtualenvwrapper
```

5. Setup virtualenvwrappers.

```
su - operador
mkdir ~/.virtualenvs
cat >> .bashrc <<EOF

VIRTUALENVWRAPPER_PYTHON=$(which python2.7)
export WORKON_HOME=\$HOME/.virtualenvs
export PIP_VIRTUALENV_BASE=\$WORKON_HOME
export PIP_RESPECT_VIRTUALENV=true
source /usr/local/bin/virtualenvwrapper.sh
EOF
. .bashrc
```

6. Create project virtualenv.

```
mkvirtualenv MyProject
pip install fanery gunicorn rainbow-saddle
python .virtualenvs/MyProject/lib/python*/site-packages/fanery/tests/test_term.py
python .virtualenvs/MyProject/lib/python*/site-packages/fanery/tests/test_service.py
pip install git+https://bitbucket.org/USER-NAME/MyProject.git@v0.0.1
deactivate
```

7. Build project start script.

```
mkdir ~/bin
cat > ~/bin/project <<EOF
#!/bin/sh

[ \$# -lt 2 ] && {
    echo "Usage: \$0 {start|stop|restart|reload} PROJECT"
    echo "           upgrade PROJECT GITREPO"
    exit 1
}

VIRTUALENVWRAPPER_PYTHON=$(which python2.7)
export WORKON_HOME=\$HOME/.virtualenvs
export PIP_VIRTUALENV_BASE=\$WORKON_HOME
export PIP_RESPECT_VIRTUALENV=true
source /usr/local/bin/virtualenvwrapper.sh

ACTION=\$1
shift
PROJECT=\$1
shift
GITREPO=\$1
PIDFILE=/var/run/\${PROJECT}.pid

case "\${ACTION}" in
    start)
        workon \${PROJECT} && {
            CORES=\$(grep ^processor /proc/cpuinfo | wc -l)
            WORKERS=\$((\${CORES} * 2 + 1))
            rainbow-saddle --pid \${PIDFILE} gunicorn -w \${WORKERS} $@
        }
        ;;
    stop)
        kill -TERM \$(cat \${PIDFILE})
        ;;
    restart|reload)
        kill -HUP \$(cat \${PIDFILE})
        ;;
    upgrade)
        workon \${PROJECT} && \\
            pip install \${GITREPO} --upgrade && \\
            kill -HUP \$(cat \${PIDFILE})
        }
        ;;
esac

exit \$?
EOF

chmod 500 ~/bin/project
```

8. Cleanup.

```
exit
aptitude -f install
apt-get autoremove
apt-get autoclean
apt-get clean
rm -rf /tmp/*
```

```
rm -f /var/log/wtmp /var/log/btmp
/sbin/lock-system
history -c
```

9. Exit schroot and create compressed archive.

```
exit
mount -t auto /dev/VG-NAME/LV-NAME /mnt
cd /mnt
mkdir /var/lib/schroot/archives
find . -depth -print \
    | cpio --create --quiet --format=crc \
    | xz -6 -Csha256 \
    > /var/lib/schroot/archives/MyProject-node_$(date +%F_%T).cpio.xz
cd /var/lib/schroot/archives
umount /mnt
ls -lh
```

10. Build schroot node setup script.

```
cat > /var/lib/schroot/archives/build-schroot <<EOF
#!/bin/sh

XZFILE=\$1
SYSTEM=\$2
VGNAME=\$3
LVNAME=\$4
LVSIZE=\$5
LVTYPE=\${6:-xfs}

[ \$# -lt 5 ] && {
    echo "Usage: \$0 XZFILE SYSTEM VGNAME LVNAME LVSIZE [LVTYPE]"
    exit 1
}

lvcreate -n \$LVNAME -L\$LVSIZE \$VGNAME && \\
    mkfs.\$LVTYPE -f /dev/\$VGNAME/\$LVNAME && \\
    mount -t \$LVTYPE /dev/\$VGNAME/\$LVNAME /mnt && \\
    cd /mnt && xzcat \$XZFILE | cpio -d --extract || exit 1

cd /var/lib/schroot/archives
umount /mnt

/sbin/unlock-filesystem

aptitude update
aptitude full-upgrade
aptitude -t wheezy-backports install schroot tar xz-utils
apt-get autoremove
apt-get autoclean
apt-get clean

sed -i -e '/\(^\/home\)/ s//#\1/' -e '/\(^\/tmp\)/ s//#\1/' /etc/schroot/default/fstab

cat >> /etc/schroot/schroot.conf <<EOC

[\$SYSTEM]
type=lvm-snapshot
users=operador
```

```
root-users=operador
source-root-users=operador
device=/dev/\$VGNAME/\$LVNAME
lvm-snapshot-options=--size 2G
EOC

/sbin/lock-filesystem
EOF

chmod 500 /var/lib/schroot/archives/build-schroot
```

5.2.3 Web farm node setup

We are now ready to replicate our project node in a few simple steps to whatever Debian based hardened virtual machine we selected as member of our *Backend Web Farm*.

1. Preparation.

```
aptitude -t wheezy-backports install schroot cpio xz-utils
cd /var/lib/schroot/
scp -r sshadmin@host:/var/lib/schroot/archives .
```

2. Build schrooted node from compressed archive.

```
cd /var/lib/schroot/archives
./build-schroot MyProject-node_VERSION.tar.xz MyProject VG-NAME LV-NAME 10G
```

5.2.4 WebApp management

Once in place our Web application is easily managed via schroot sessions.

1. Start MyProject daemon.

```
schroot -b -n MyProject -c MyProject -u operador
schroot -r -c MyProject -- /home/operador/bin/project start MyProject myproject:app --log-le
```

2. Project version upgrade and graceful code reload.

```
schroot -c source:MyProject -u operador -- /home/operador/bin/project upgrade MyProject git+
```

Read schroot-faq(7) man page for more details about schroot sessions.

CHAPTER 6

Working with Fanery

Fanery is a framework build by developers for developers.

Opinionated frameworks usually takes uncommon and controversial paths you may disagree with, that's why before starting to work with Fanery it's fundamental to understand the intrinsics ideas that give birth to the tool as it is:

- Strong security by default.
- Focus on being developer-oriented.
- Promote funcional pythonic style.
- Promote continuous testing+profiling.

This section is dedicated to make sense of that 4 ideas in the context of Web Software Development with Fanery Framework.

6.1 Strong security by default

Simplicity is key to growth and manageability of Software projects, however isn't uncommon that development frameworks get in your way during the process.

Fanery services, which are just remotely callable functions, let safely expose functionalities unobtrusively via `@service` decorator.

Let briefly look at two simple services, a public anonymously callable and another one that require authenticated encrypted connection.

6.1.1 Public anonymous service

Public anonymous exposed functions do not comply to Fanery security protocol, but as Fanery enable strong security by default it's necessary to decorate our service explicitly to be anonymous:

```
from fanery import service

@service(safe=False, ssl=False, auto_parse=False, cache=True)
def hello(name='World'):
    return "Hello, %s!" % name
```

By default call's arguments are auto-parsed to built-in/harmless data types. In this case `hello` service explicitly disable it.

Caching control through `pragma`, `cache-control` and `expires` headers can be enabled setting `cache` argument to `True` or a positive value representing cache expiration in seconds (default to `False`).

Additionally `hello` service may be called over unsecure `HTTP` connections; by default only `SSL/TLS` encrypted calls are accepted.

Disabling `safe` argument tells Fanery not to trigger session's security checks.

6.1.2 Secure authenticated service

Our service `profile_data` just return some personal information about the current session user.

```
from fanery import service, get_state

@service()
def profile_data():
    profile = get_state().profile
    return dict(name=profile.username,
                email=profile.email,
                role=profile.role)
```

The use of `@service()` without arguments imply `@service(safe=True, ssl=True, cache=False)` which means our exposed function `profile_data()` can be called only if the following conditions are honored:

- Calls must travel inside encrypted SSL/TLS channel.
- A valid/active authenticated session exists.
- Remote call goes through Fanery security protocol (`NaCL` + `One-Time pad`).

Fanery framework comes with all required JavaScript functionalities to perform such type of secure Ajax calls.

Safe Ajax calls are asynchronous and deferred:

```
Fanery.safe_call('profile_data').then(console.log).fail(console.error);
Fanery.safe_call('profile_data.json').always(console.debug);
```

In both cases all data between the client and server is secured and (de)serialized automatically to `JSON`.

All required JavaScript codes and third party libraries are mapped by default behind `jfanery/` urlpath:

```
<!-- third party FLOSS libraries jfanery depends on -->
<script src="jfanery/nacl_factory.js"></script>
<script src="jfanery/scrypt.js"></script>
<script src="jfanery/base64.js"></script>

<!-- fanery security protocol implementation -->
<script src="jfanery/jfanery.js"></script>
```

Unobtrusiveness is reflected in the following principles:

> `profile_data` know nothing and should not care about protocols (`HTTP/S`, `WSGI`), serialization formats (`JSON`), encryption (`NaCL`, `One-Time pad`), session abuse (`bruteforce`, `hijacking`, `MitM`, `CSRF`) nor anything outside Python and the job it's supposed to perform.

Staying focused on the job to perform without wasting precious resources thinking about the external environment is fundamental to reduce complexity.

6.2 Focus on being developer-oriented

Fanery doesn't try or pretend to define *"best pratice"* about Software development. Every developer has his own style and the tool shouldn't impose boundaries, for such reason the following principles are respected:

- The framework must not depend on strict/pre-defined configuration style/format and/or directory structure.
- The framework must not tie to a particular storage or UI technology.
- The framework must provide the facilities for easy testing, debugging and profiling.
- The framework must not rely on components that inhibit elastic/horizontal scalability.

At this point must be clear that Fanery set apart from most commons Web development frameworks; indeed it's been build in compliance to the following *"unpopular"* ideas:

- Storage strategy should be the last concern during Software development.
- End-user interfaces should not be generated server-side.
- Funcional development style is superior to Object Oriented.
- Premature optimization may be evil but early optimization is prerequisite to Software quality.
- Security must not be compromised in favor to obsolete/legacy systems compatibility nor performance.

6.3 Promote functional pythonic style

Functional pythonic style here refer to the practice of building Software around a collection of functions organized and named accordingly to their being, the practice of using data as pure data, rejecting the idea of black boxes with inner state and personalized behaviours as proposed by Object Orientation.

Fanery itself is build following such principle, classes are seldom choosen as building blocks, only when it make sense in a pythonic style.

Specific classes used by Fanery are:

- `Hict`: dotted hierarchical dictionary.
- `Aict`: dotted hierarchical dictionary with terms auto-parsing.
- `DataStore`: store strategy proxy, a glue between all containers representing the *Elastic Backend storage* layer seen in our introductory setion.
- `Record`: versioned model-aware data container.
- `store`: abstraction built to handle all storage activities as a single unit of work inside a `with` statement.

Fanery data types, decorators, functions library and abstraction helpers aid developers in their quest to elegant, scalable and high available Software solutions.

6.4 Promote continuous testing+profiling

Software development is a process, a never ending quest to maturity, perfection and mankind knowledge growth; in such spirit maturity and knowledge come from experimentation and understanding after each mistake/achievement:

> *What's measurable and replicable has a really good chance to be improoved.*

This corollary stone of science is intrinsics to Software production, that's why testing and profiling must be a fundamental gear inside development machinery.

Fanery try to shorten the path required to apply testing and profiling practice to the process, providing easy access to venerable third party libraries like:

- `memory-profiler`: line-by-line memory usage.
- `line-profiler`: line-by-line execution performance.
- `profile-hooks`: code coverage and functions execution timing.
- `linesman`: wsgi middleware able to build execution stack performance metrics and graphs.
- `ipdb`: Python debugger on steroids.
- `rpdb2`: remote Python debugger.

Software testing and profiling is fundamental to garantee quality, it should be done early and iteratively; I'm not advocating one or another testing methodology here, just use the one you feel more confortable with, the one that fit best in your development process.

Another Python project that deserve great mention is `pyrasite`, it let for injection of arbitrary code into running Python process and integrate nicely with `meliae`, `pycallgraph` and `psutil`.

6.4.1 Software Performance Testing

Knowing your Software works correctly is not enough, customers have expectations that should be fulfilled, that means our Software must be load/stress tested through each architecture layers to guarantee the required level of robustness and availability.

Several tools exists for the job, some that deserve attentions are:

- multi-mechanize, funkload, locust.io: load/stress testing of Web applications.
- munin: hardware, network, system and application resource monitoring.
- bucky: web page rendering performance and timing.
- sentry, graylog2: centralized logging and events correlation.

TODO: add hyperlinks to projects pages

CHAPTER 7

Tutorial

This tutorial will show a very simple and contrived example of a simplified *TODO list* web application build with Fanery.

7.1 Before we start

We suppose the story board has been previously approved and the corresponding Web UI has already been coded.

Html, javascript, css and graphic files are teoretically stored inside a local `static` folder.

7.2 TODO list toy Web application

TODO list application has really basic server side requirements, the full API is reduced to:

- `get_all() -> [todo]`: retrieve all todos.
- `add(text) -> todo`: add a new todo.
- `update(todo) -> todo`: update an existing todo.
- `remove(todo) -> bool`: remove an existing todo.

Every `todo` object is represented by a Json object with the following attributes:

- `done`: boolean that define if todo is completed.
- `text`: todo description.
- `id`: todo unique id.
- `vsn`: todo record version.

Additionally only authenticated users will be able to manage todos.

7.2.1 Consuming Fanery services with JavaScript

First let's build a JavaScript proxy to consume Fanery services via asynchronous deferred Ajax calls.

```
window.TodoApp = (function (F, E) {

    var self = this;
```

```
    self.get_all = function () {
        return F.safe_call('get_all');
    };

    self.add = function (text) {
        return F.safe_call('add', text);
    };

    self.update = function (todo) {
        return F.safe_call('update', todo);
    };

    self.remove = function (todo) {
        var params = {'id': todo.id, 'vsn': todo.vsn};
        return F.safe_call('remove', params);
    };

    self.login = function (username, password) {
        return F.login(username, password);
    };

    self.logout = function () {
        return F.logout();
    };

    E.InvalidCredential = function () {
        alert('Sorry, invalid username or password');
    };

    E.Unauthorized = function () {
        alert('Sorry, must login first');
    };

    E.Error = function () {
        alert('Sorry, an error occurred: ' + error.exc + '\n\n' + error.err.join('\n'));
    };

    return self;
})(Fanery, Fanery.exc);
```

7.2.2 Server side TODO list setup

During development is handy to have modules auto-reload which brings us many benefits in term of productivity.

Create a todoapp/_config.py file as follow:

```
from fanery import config

# enable auto-reload
config.IS_DEVELOPMENT = True

# the folder where static files lives (html, css, javascripts, etc)
STATIC_DIR = 'static'
```

It's important not to forget that setting `IS_DEVELOPMENT` to `True` disable default behaviour to force SSL, the reason for such decision is to let start quickly producing/testing/profiling code without requiring first the difficult and time consuming job of properly creating certificates and setting up a caching web reverse proxy.

Production code must always leave `IS_DEVELOPMENT` set to default `False` value, not only to enforce SSL usage, also because having modules auto-reload in production environment is a very bad idea.

Models definition file todoapp/_models.py may look like this:

```
from fanery import Hict

class Todo:

    @classmethod
    def initialize(cls, record):
        record.merge(done=False, text='')

    @classmethod
    def validate(cls, record):
        errors = Hict()

        text = record.text
        if not isinstance(text, basestring):
            errors.text.bad_type = type(text).__name__
        elif not text.strip():
            errors.text.invalid = text

        done = record.done
        if not isinstance(done, bool):
            errors.done.bad_type = type(done).__name__

        return errors

    @classmethod
    def index(cls, record):
        return dict(done=record.done, text=record.text)

    @classmethod
    def to_dict(cls, record):
        return dict(done=record.done, text=record.text,
                    id=record._uuid, vsn=record._vsn)
```

It's sane to always perform server-side data validations and never trust input sent by our users. Validation in the front-end is also a good idea but we can't rely on it.

Following the idea that storage strategy should be our last concern, we'll start using a toy proxy implementation already provided with Fanery.

Setup file todoapp/_setup.py take care of it:

```
from fanery import DataStore, dbm, auth, add_model, add_storage
from _models import Todo

db = dbm.MemDictStore()
storage = DataStore(db, permission=db,
                        state=db,
                        abuse=db,
                        profile=db,
                        settings=db)

add_model(Todo)
add_storage(storage, Todo)

auth.setup(storage)
```

```
auth.add_user('MY-USER', 'MY-SECRET', domain='localhost')
```

Fanery is build with the idea that applications should be multi-tenant, transparent multi-tenancy is achived via domains abstraction, in other words, `state` and `Record` objects belongs to a specific `domain` which define the tenancy space.

`MemDictStore` is a toy in-memory datastore implementation that define all required hooks necessary to support Fanery `storage` facility, its purpose is only to get started with a storage strategy that let experiment with data models during early development stage, until a proper and production ready stategy is choosen.

Application business logic may be defined inside todoapp/_api.py file as follow:

```
from fanery import service, storage
from _models import Todo

@service()
def get_all():
    with storage() as db:
        return map(Todo.to_dict, db.select(Todo))

@service(auto_parse=False)
def add(text):
    with storage() as db:
        record = db.insert(Todo, text=text)
    return Todo.to_dict(record)

@service(auto_parse=False)
def update(id, vsn, text, done):
    with storage() as db:
        record = db.fetch(Todo, id, vsn)
        record.text = text
        record.done = done
        db.update(record)
        # or just
        # record = db.update(Todo, id, vsn, text=text, done=done)
    return Todo.to_dict(record)

@service()
def remove(id, vsn):
    with storage() as db:
        record = db.fetch(Todo, id, vsn)
        db.delete(record)
        # or just
        # db.delete(Todo, id, vsn)
    return True
```

What's left is starting our `TODO list` behind some `WSGI` capable application server.

For development purpose start_server.py will do:

```
from fanery import server, static
from todoapp import config

static('/', config.STATIC_DIR)

server.start_wsgi_server()
```

Make todoapp directory a Python module:

```
cat > todoapp/__init__.py <<EOF
import _config as config
```

```
import _models as models
import _api as api
import _setup
EOF
```

Finally start TODO list Web application.

```
PYTHONPATH=. python start_server.py
```

Point your browser to http://localhost:9000/.

The way our `TodoApp` ajax proxy may be used to consume todoapp API should be clarified by the following JavaScript sniplet:

```
// login first
TodoApp.login("MY-USER", "MY-SECRET").then(function () {

    // create your first todo
    TodoApp.add("Play with Fanery").then(function (todo) {

        // verify got stored
        TodoApp.get_all().then(function (data) {
            if (data.length != 1) alert("invalid length");
            if (data[0].id != todo.id) alert("unexpected id");
            if (todo.done || data[1].done) alert("should not be done");

            // update todo
            todo.done = true;

            // verify got updated
            TodoApp.update(todo).then(function (updated) {
                if (todo.id != updated.id) alert("id mismatch");
                if (todo.vsn != (updated.vsn - 1)) alert("unexpected version");
                if (todo.text != updated.text) alert("text mismatch");
                if (!updated.done) alert("should be done");

                // remove updated todo
                TodoApp.remove(updated).then(function (success) {
                    if (!success) alert("couldn't remove updated todo");

                    // verify no more todo available
                    TodoApp.get_all().then(function (data) {
                        if (data.length > 0) alert("should be no more todos left");

                        // logout
                        TodoApp.logout().then(function () {

                            // no anonymous access 1
                            TodoApp.get_all();

                            // no anonymous access 2
                            Fanery.safe_call("get_all");

                            // no anonymous access 3 (detected as abusive behaviour)
                            Fanery.call("/get_all.json");

                            // no anonymous access 4 (detected as abusive behaviour)
                            jQuery.get("/get_all.json");
```

```
                                    // repeate the last 2 calls a few more times and you won't be able t
                                });
                            });
                        });
                    });
                });
            });
        });
```

Going back to transparent multi-tenancy premise, the hostname in the URI must match the `domain` value defined during `auth.add_user()` call, if they don't `TodoApp.login()` will show `InvalidCredential` message error.

CHAPTER 8

Fanery Security Protocol

TODO

CHAPTER 9

Storage strategy

TODO