## Contents:

1 Installation .............................. 3  
   1.1 PKG .................................. 3  
   1.2 ports ................................ 3  

2 Network Requirements .................... 5  
   2.1 Local Area Network .................... 5  
   2.2 Public Network ......................... 6  
   2.3 /etc/pf.conf ......................... 6  

3 Usage ................................ 9  

4 Targeting ................................ 11  

5 Examples: Containers ...................... 13  

6 Examples: Releases ....................... 15  

7 Bastille sub-commands .................... 17  
   7.1 bootstrap .......................... 17  
   7.2 Releases .......................... 17  
   7.3 Templates .......................... 18  
   7.4 cmd ................................ 18  
   7.5 console .......................... 19  
   7.6 cp ................................ 19  
   7.7 create .......................... 20  
   7.8 destroy .......................... 20  
   7.9 htop .......................... 21  
   7.10 pkg .......................... 21  
   7.11 restart .......................... 24  
   7.12 service .......................... 24  
   7.13 start .......................... 24  
   7.14 stop .......................... 25  
   7.15 sysrc .......................... 25  
   7.16 top .......................... 25  
   7.17 update .......................... 26  
   7.18 upgrade .......................... 26  
   7.19 verify .......................... 26
8 Template
  8.1 Applying Templates .................................................. 30

9 jail.conf
  9.1 interface ............................................................... 33
  9.2 host.hostname .......................................................... 34
  9.3 exec.consolelog ......................................................... 34
  9.4 path ................................................................. 34
  9.5 securelevel ............................................................ 34
  9.6 devfs_ruleset ............................................................ 35
  9.7 enforce_statfs ........................................................... 35
  9.8 exec.start ............................................................. 35
  9.9 exec.stop ............................................................... 36
  9.10 exec.clean ............................................................. 36
  9.11 mount.devfs ............................................................ 36
  9.12 mount.fstab ............................................................. 36

10 Copyright .................................................. 37
Welcome to the official Bastille documentation. This collection of documents will outline installation and usage of Bastille.

The latest version of this documentation can always be found at https://docs.bastillebsd.org.
Bastille is available in the official FreeBSD ports tree at sysutils/bastille. Binary packages available in quarterly and latest repositories.

Current version is 0.5.20191125.

To install from the FreeBSD package repository:

• quarterly repository may be older version
• latest repository will match recent ports

1.1 PKG

pkg install bastille

To install from source (don’t worry, no compiling):

1.2 ports

make -C /usr/ports/sysutils/bastille install clean
Network Requirements

Here’s the scenario. You’ve installed Bastille at home or in the cloud and want to get started putting applications in secure little containers, but how do I get these containers on the network?

Bastille tries to be flexible about how to network containerized applications. The two most common methods are described here. Consider both options to decide which design work best for your needs. One of the methods works better across clouds while the other is simpler if used in local area networks.

As you’ve probably seen, Bastille containers require certain information when they are created. An IP address has to be assigned to the container through which all network traffic will flow.

When the container is started the IP address assigned at creation will be bound to a network interface. In FreeBSD these interfaces have different names, but look something like `em0`, `bge0`, `re0`, etc. On a virtual machine it may be `vtnet0`. You get the idea…

Note: if you are running in the cloud and only have a single public IP you may want the Public Network option. See below.

### 2.1 Local Area Network

I will cover the local area network (LAN) method first. This method is simpler to get going and works well in a home network (or similar) where adding alias IP addresses is no problem.

Bastille allows you to define the interface you want the IP attached to when you create it. An example:

```
bastille create alcatraz 12.1-RELEASE 192.168.1.50 em0
```

When the `alcatraz` container is started it will add `192.168.1.50` as an IP alias to the `em0` interface. It will then simply be another member of the hosts network. Other networked systems (firewall permitting) should be able to reach services at that address.

This method is the simplest. All you need to know is the name of your network interface and a free IP on your current network.
(Bastille does try to verify that the interface name you provide it is a valid interface. This validation has not been exhaustively tested yet in Bastille’s beta state.)

### 2.2 Public Network

In this section I’ll describe how to network containers in a public network such as a cloud hosting provider (AWS, digital ocean, vultr, etc.)

In the public cloud you don’t often have access to multiple private IP addresses for your virtual machines. This means if you want to create multiple containers and assign them all IP addresses, you’ll need to create a new network.

What I recommend is creating a cloned loopback interface (bastille0) and assigning all the containers private (rfc1918) addresses on that interface. The setup I develop on and use Bastille day to day uses the 10.0.0.0/8 address range. I have the ability to use whatever address I want within that range because I’ve created my own private network. The host system then acts as the firewall, permitting and denying traffic as needed.

I find this setup the most flexible across all types of networks. It can be used in public and private networks just the same and it allows me to keep containers off the network until I allow access.

Having said all that here are instructions I used to configure the network with a private loopback interface and system firewall. The system firewall NATs traffic out of containers and can selectively redirect traffic into containers based on connection ports (ie; 80, 443, etc.)

First, create the loopback interface:

```
ishmael ~ # sysrc cloned_interfaces+=lo1
nishmael ~ # sysrc ifconfig_lo1_name="bastille0"
nishmael ~ # service netif cloneup
nishmael ~ # ifconfig bastille0 inet 10.17.89.10
```

Second, enable the firewall:

```
nishmael ~ # sysrc pf_enable="YES"
```

Create the firewall rules:

#### 2.3 /etc/pf.conf

```
ext_if="vtnet0"
set block-policy return
scrub in on $ext_if all fragment reassemble
set skip on lo
nat on $ext_if from bastille0:network to any -> ($ext_if)
## rdr example
## rdr pass inet proto tcp from any to any port {80, 443} -> 10.17.89.45
block in all
pass out quick modulate state
antispoof for $ext_if inet
pass in inet proto tcp from any to any port ssh flags S/SA modulate state
```

- Make sure to change the `ext_if` variable to match your host system interface.
• Make sure to include the last line (port ssh) or you’ll end up locked out.

Note: if you have an existing firewall, the key lines for in/out traffic to containers are:

```
nat on $ext_if from bastille0:network to any -> ($ext_if)
## rdr example
## rdr pass inet proto tcp from any to any port {80, 443} -> 10.17.89.45
```

The `nat` routes traffic from the loopback interface to the external interface for outbound access.
The `rdr pass` . . . will redirect traffic from the host firewall on port X to the ip of Container Y. The example shown redirects web traffic (80 & 443) to the containers at 10.17.89.45.

Finally, start up the firewall:

```
ishmael ~ # service pf restart
```

At this point you’ll likely be disconnected from the host. Reconnect the ssh session and continue.
This step only needs to be done once in order to prepare the host.
Bastille is an open-source system for automating deployment and management of containerized applications on FreeBSD.

Usage:

```
bastille command [ALL|glob] [args]
```

Available Commands:

- **bootstrap**: Bootstrap a FreeBSD release for container base.
- **cmd**: Execute arbitrary command on targeted container(s).
- **console**: Console into a running container.
- **cp**: cp(1) files from host to targeted container(s).
- **create**: Create a new thin container or a thick container if -T|--thick option is specified.
- **destroy**: Destroy a stopped container or a FreeBSD release.
- **help**: Help about any command.
- **htop**: Interactive process viewer (requires htop).
- **list**: List containers, releases, templates, or logs.
- **pkg**: Manipulate binary packages within targeted container(s). See pkg(8).
- **restart**: Restart a running container.
- **service**: Manage services within targeted container(s).
- **start**: Start a stopped container.
- **stop**: Stop a running container.
- **sysrc**: Safely edit rc files within targeted container(s).
- **template**: Apply file templates to targeted container(s).
- **top**: Display and update information about the top(1) cpu processes.
- **update**: Update container base -pX release.
- **upgrade**: Upgrade container release to X.Y-RELEASE.
- **verify**: Compare release against a "known good" index.
- **zfs**: Manage (get|set) zfs attributes on targeted container(s).

Use "bastille -v|--version" for version information.
Use "bastille command -h|--help" for more information about a command.
Bastille uses a *command-target-args* syntax, meaning that each command requires a target. Targets are usually containers, but can also be releases.

Targeting a container is done by providing the exact container’s name.

Targeting a release is done by providing the release name. (Note: do not include the `-pX` point-release version.)

Bastille includes a pre-defined keyword `ALL` to target all running containers.

In the future I would like to support more options, including globbing, lists and regular-expressions.
## Examples: Containers

ishmael ~ # bastille ...

<table>
<thead>
<tr>
<th>command</th>
<th>target</th>
<th>args</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd</td>
<td>ALL</td>
<td>‘sockstat -4’</td>
<td>execute <code>sockstat -4</code> in ALL containers (ip4 sockets)</td>
</tr>
<tr>
<td>console</td>
<td>mariadb02</td>
<td>—</td>
<td>console (shell) access to mariadb02</td>
</tr>
<tr>
<td>pkg</td>
<td>web01</td>
<td>‘install nginx’</td>
<td>install nginx package in web01 container</td>
</tr>
<tr>
<td>pkg</td>
<td>ALL</td>
<td>upgrade</td>
<td>upgrade packages in ALL containers</td>
</tr>
<tr>
<td>pkg</td>
<td>ALL</td>
<td>audit</td>
<td>(CVE) audit packages in ALL containers</td>
</tr>
<tr>
<td>sysrc</td>
<td>web01</td>
<td>nginx_enable=YES</td>
<td>execute <code>sysrc nginx_enable=YES</code> in web01 container</td>
</tr>
<tr>
<td>template</td>
<td>ALL</td>
<td>username/base</td>
<td>apply <code>username/base</code> template to ALL containers</td>
</tr>
<tr>
<td>start</td>
<td>web02</td>
<td>—</td>
<td>start web02 container</td>
</tr>
<tr>
<td>cp</td>
<td>bastion03</td>
<td>/tmp/resolv.conf-cf etc/resolv.conf</td>
<td>copy host-path to container-path in bastion03</td>
</tr>
<tr>
<td>create</td>
<td>folsom</td>
<td>12.0-RELEASE 10.17.89.10</td>
<td>create 12.0 container named <code>folsom</code> with IP</td>
</tr>
</tbody>
</table>
Examples: Releases

<table>
<thead>
<tr>
<th>command</th>
<th>target</th>
<th>args</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap</td>
<td>12.0-RELEASE</td>
<td>—</td>
<td>bootstrap 12.0-RELEASE release</td>
</tr>
<tr>
<td>update</td>
<td>11.3-RELEASE</td>
<td>—</td>
<td>update 11.2-RELEASE release</td>
</tr>
<tr>
<td>upgrade</td>
<td>11.2-RELEASE</td>
<td>11.3-RELEASE</td>
<td>update 11.2-RELEASE release</td>
</tr>
<tr>
<td>verify</td>
<td>11.3-RELEASE</td>
<td>—</td>
<td>update 11.2-RELEASE release</td>
</tr>
</tbody>
</table>
7.1 bootstrap

The bootstrap sub-command is used to download and extract releases and templates for use with Bastille containers. A valid release is needed before containers can be created. Templates are optional but are managed in the same manner. Note: your mileage may vary with unsupported releases and releases newer than the host system likely will NOT work at all. Bastille tries to filter for valid release names. If you find it will not bootstrap a valid release, please let us know.

In this document we will describe using the bootstrap sub-command with both releases and templates. We begin with releases.

7.2 Releases

7.2.1 Example

To bootstrap a release, run the bootstrap sub-command with the release version as the argument.

```
ishmael ~ # bastille bootstrap 11.3-RELEASE [update]
ishmael ~ # bastille bootstrap 12.0-RELEASE
nishmael ~ # bastille bootstrap 12.1-RELEASE
```

This command will ensure the required directory structures are in place and download the requested release. For each requested release, bootstrap will download the base.txz. These files are verified (sha256 via MANIFEST file) before they are extracted for use.

7.2.2 Tips

The bootstrap sub-command can now take (0.5.20191125+) an optional second argument of “update”. If this argument is used, bastille update will be run immediately after the bootstrap, effectively bootstrapping and applying security patches and errata in one motion.
7.2.3 Notes

The bootstrap subcommand is generally only used once to prepare the system. The only other use case for the bootstrap command is when a new FreeBSD version is released and you want to start deploying containers on that version.

To update a release as patches are made available, see the `bastille update` command.

Downloaded artifacts are stored in the `bastille/cache/version` directory. “bootstrapped” releases are stored in `bastille/releases/version`.

To manually bootstrap a release (aka bring your own archive), place your archive in `bastille/cache/name` and extract to `bastille/releases/name`. Your mileage may vary; let me know what happens.

7.3 Templates

Bastille aims to integrate container automation into the platform while maintaining a simple, uncomplicated design. Templates are git repositories with automation definitions for packages, services, file overlays, etc.

To download one of these templates see the example below.

7.3.1 Example

```
ishmael ~ # bastille bootstrap https://gitlab.com/bastillebsd-templates/nginx
ishmael ~ # bastille bootstrap https://gitlab.com/bastillebsd-templates/mariadb-server
ishmael ~ # bastille bootstrap https://gitlab.com/bastillebsd-templates/python3
```

7.3.2 Tips

See the documentation on templates for more information on how they work and how you can create or customize your own. Templates are a powerful part of Bastille and facilitate full container automation.

7.3.3 Notes

If you don’t want to bother with git to use templates you can create them manually on the Bastille system and apply them.

Templates are stored in `bastille/templates/namespace/name`. If you’d like to create a new template on your local system, simply create a new namespace within the templates directory and then one for the template. This namespacing allows users and groups to have templates without conflicting template names.

Once you’ve created the directory structure you can begin filling it with template hooks. Once you have a minimum number of hooks (at least one) you can begin applying your template.

7.4 cmd

To execute commands within the container you can use `bastille cmd`. 
ishmael ~ # bastille cmd folsom 'ps -auxw'
[folsom]:
USER     PID %CPU %MEM  VSZ RSS  TT  STAT STARTED    TIME     COMMAND
root    71464  0.0   0.0  14536  2000 -IsJ  4:52PM 0:00.00 /usr/sbin/syslogd -ss
root    77447  0.0   0.0  16632  2140 -SsJ  4:52PM 0:00.00 /usr/sbin/cron -J 60 -s
root    80591  0.0   0.0  18784  2340  1  R+J  4:53PM 0:00.00 ps -auxw

7.5 console

This sub-command launches a login shell into the container. Default is password-less root login.

ishmael ~ # bastille console folsom
[folsom]:
FreeBSD 12.1-RELEASE-p1 GENERIC

Welcome to FreeBSD!

FreeBSD FAQ: https://www.FreeBSD.org/faq/
FreeBSD Forums: https://forums.FreeBSD.org/

Documents installed with the system are in the /usr/local/share/doc/freebsd/ directory, or can be installed later with: pkg install en-freebsd-doc
For other languages, replace "en" with a language code like de or fr.

Show the version of FreeBSD installed: freebsd-version ; uname -a
Please include that output and any error messages when posting questions.
Introduction to manual pages: man man
FreeBSD directory layout: man hier

Edit /etc/motd to change this login announcement.
root@folsom:~ #

At this point you are logged in to the container and have full shell access. The system is yours to use and/or abuse as you like. Any changes made inside the container are limited to the container.

7.6 cp

This command allows efficiently copying files from host to container(s).

ishmael ~ # bastille cp ALL /tmp/resolv.conf-cf etc/resolv.conf
[bastion]:
[unbound0]:
[unbound1]:
[squid]:

(continues on next page)
Unless you see errors reported in the output the `cp` was successful.

## 7.7 create

Bastille create uses any available bootstrapped release to create a lightweight container system. To create a container simply provide a name, bootstrapped release and a private (rfc1918) IP address.

- name
- release
- ip
- interface (optional)

```
ishmael ~ $ bastille create folsom 11.3-RELEASE 10.17.89.10 [interface]
```

RELEASE: 11.3-RELEASE.
NAME: folsom.
IP: 10.17.89.10.

This command will create a 11.3-RELEASE container assigning the 10.17.89.10 ip address to the new system.

I recommend using private (rfc1918) ip address ranges for your container. These ranges include:

- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16

Bastille does its best to validate the submitted ip is valid. This has not been thoroughly tested–I generally use the 10/8 range.

## 7.8 destroy

Containers can be destroyed and thrown away just as easily as they were created. Note: containers must be stopped before destroyed.

```
ishmael ~ $ bastille stop folsom
[folsom]:
folsom: removed

ishmael ~ $ bastille destroy folsom
Deleting Container: folsom.
Note: containers console logs not destroyed.
/usr/local/bastille/logs/folsom_console.log
```
7.9 htop

This one runs htop inside the container. note: won’t work if you don’t have htop installed in the container.

```
1  [ ]  0.5%]  5  [ ]  0.0%
2  [ ]  0.0%]  6  [ ]  0.0%
3  [ ]  0.0%]  7  [ ]  0.0%
4  [ ]  0.0%]  8  [ ]  0.0%
Mem[------------------------------------------] 7.14G/24.0G
Swp[------------------------------------------] 0K/2.00G
```

Tasks: 8, 0 thr; 1 running
Load average: 0.04 0.10 0.08
Uptime: 10 days, 22:27:09

```
F1 Help  F2 Setup  F3 Search  F4 Filter  F5 Sorted  F6 Collapse  F7 Nice  F8 Nice  F9 Kill  F10 Quit
```

7.10 pkg

To manage binary packages within the container use bastille pkg.

```
ishmael ~ # bastille pkg folsom 'install vim-console git-lite zsh'
[folsom]:
The package management tool is not yet installed on your system.
Do you want to fetch and install it now? [y/N]: y
```

```
Bootstrapping pkg from pkg+http://pkg.FreeBSD.org/FreeBSD:10:amd64/quarterly, please wait...
Verifying signature with trusted certificate pkg.freebsd.org.2013102301...
done
[folsom] Installing pkg-1.10.5_5...
[folsom] Extracting pkg-1.10.5_5: 100%
```

```
 Updating FreeBSD repository catalogue...
pkg: Repository FreeBSD load error: access repo file(/var/db/pkg/repo-FreeBSD.sqlite) failed: No such file or directory
```

```
[folsom] Fetching metatx.z: 100% 944 B 0.9kB/s 00:01
[folsom] Fetching packagesite.txz: 100% 6 MiB 3.4MB/s 00:02
 Processing entries: 100%
 FreeBSD repository update completed. 32550 packages processed.
 All repositories are up to date.
 Updating database digests format: 100%
 The following 10 package(s) will be affected (of 0 checked):
```

New packages to be INSTALLED:

(continues on next page)
The PKG sub-command can, of course, do more than just install. The expectation is that you can fully leverage the pkg manager. This means, install, update, upgrade, audit, clean, autoremove, etc., etc.
ishmael ~ # bastille pkg ALL upgrade

[bastion]:
Updating pkg.bastillebsd.org repository catalogue...
[bastion] Fetching meta.txz: 100% 560 B 0.6kB/s 00:01
[bastion] Fetching packagesite.txz: 100% 118 KiB 121.3kB/s 00:01
Processing entries: 100%
pkg.bastillebsd.org repository update completed. 493 packages processed.
All repositories are up to date.
Checking for upgrades (1 candidates): 100%
Processing candidates (1 candidates): 100%
Checking integrity... done (0 conflicting)
Your packages are up to date.

[unbound0]:
Updating pkg.bastillebsd.org repository catalogue...
[unbound0] Fetching meta.txz: 100% 560 B 0.6kB/s 00:01
[unbound0] Fetching packagesite.txz: 100% 118 KiB 121.3kB/s 00:01
Processing entries: 100%
pkg.bastillebsd.org repository update completed. 493 packages processed.
All repositories are up to date.
Checking for upgrades (0 candidates): 100%
Processing candidates (0 candidates): 100%
Checking integrity... done (0 conflicting)
Your packages are up to date.

[unbound1]:
Updating pkg.bastillebsd.org repository catalogue...
[unbound1] Fetching meta.txz: 100% 560 B 0.6kB/s 00:01
[unbound1] Fetching packagesite.txz: 100% 118 KiB 121.3kB/s 00:01
Processing entries: 100%
pkg.bastillebsd.org repository update completed. 493 packages processed.
All repositories are up to date.
Checking for upgrades (0 candidates): 100%
Processing candidates (0 candidates): 100%
Checking integrity... done (0 conflicting)
Your packages are up to date.

[squid]:
Updating pkg.bastillebsd.org repository catalogue...
[squid] Fetching meta.txz: 100% 560 B 0.6kB/s 00:01
[squid] Fetching packagesite.txz: 100% 118 KiB 121.3kB/s 00:01
Processing entries: 100%
pkg.bastillebsd.org repository update completed. 493 packages processed.
All repositories are up to date.
Checking for upgrades (0 candidates): 100%
Processing candidates (0 candidates): 100%
Checking integrity... done (0 conflicting)
Your packages are up to date.

[nginx]:
Updating pkg.bastillebsd.org repository catalogue...
[nginx] Fetching meta.txz: 100% 560 B 0.6kB/s 00:01
[nginx] Fetching packagesite.txz: 100% 118 KiB 121.3kB/s 00:01
Processing entries: 100%
pkg.bastillebsd.org repository update completed. 493 packages processed.
All repositories are up to date.
Checking for upgrades (1 candidates): 100%

(continues on next page)
Processing candidates (1 candidates): 100%
The following 1 package(s) will be affected (of 0 checked):

Installed packages to be UPGRADED:
  nginx-lite: 1.14.0_14,2 -> 1.14.1,2

Number of packages to be upgraded: 1

315 KiB to be downloaded.

Proceed with this action? [y/N]: y
[nginx] [1/1] Fetching nginx-lite-1.14.1,2.txz: 100% 315 KiB 322.8kB/s 00:01
Checking integrity... done (0 conflicting)
[nginx] [1/1] Upgrading nginx-lite from 1.14.0_14,2 to 1.14.1,2...
  --> Creating groups.
  --> Creating users
  --> Creating group 'www'.
  --> Creating user 'www'.
[nginx] [1/1] Extracting nginx-lite-1.14.1,2: 100%
You may need to manually remove /usr/local/etc/nginx/nginx.conf if it is no longer needed.

7.11 restart

To restart a container you can use the bastille restart command.

ishmael ~ # bastille restart folsom
[folsom]: folsom: removed
[folsom]: folsom: created

7.12 service

The service sub-command allows for managing services within containers. This allows you to start, stop, restart, and otherwise interact with services running inside the containers.

ishmael ~ # bastille service web01 'nginx start'
ishmael ~ # bastille service db01 'mysql-server restart'
ishmael ~ # bastille service proxy 'nginx configtest'

7.13 start

To start a container you can use the bastille start command.

ishmael ~ # bastille start folsom
[folsom]: folsom: created
7.14 stop

To stop a container you can use the `bastille stop` command.

```
ishmael ~ # bastille stop folsom
[folsom]:
folsom: removed
```

7.15 sysrc

The `sysrc` sub-command allows for safely editing system configuration files. In container terms, this allows us to toggle on/off services and options at startup.

```
ishmael ~ # bastille sysrc nginx nginx_enable="YES"
[nginx]:
nginx_enable: NO -> YES
```

See `man sysrc(8)` for more info.

7.16 top

This one runs `top` in that container.

```
last pid: 96267; load averages: 0.16, 0.17, 0.11 up 10+22:37:33 21:59:07
8 processes: 1 running, 7 sleeping
CPU: 0.0% user, 0.0% nice, 0.0% system, 0.0% interrupt, 100% idle
Mem: 6768K Active, 323M Inact, 15G Wired, 7891M Free
ARC: 8391M Total, 4004M MFU, 3376M MRU, 64K Anon, 68M Header, 943M Other
5937M Compressed, 11G Uncompressed, 1.84:1 Ratio
Swap: 2048M Total, 2048M Free

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>THR</th>
<th>PRI</th>
<th>NICE</th>
<th>SIZE</th>
<th>RES</th>
<th>STATE</th>
<th>C</th>
<th>TIME</th>
<th>WCPU</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>96267</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>7916K</td>
<td>3192K</td>
<td>CPU1</td>
<td>1</td>
<td>0:00</td>
<td>0.01%</td>
<td>top</td>
</tr>
<tr>
<td>34907</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>11236K</td>
<td>7552K</td>
<td>kqread</td>
<td>6</td>
<td>0:00</td>
<td>0.00%</td>
<td>nginx</td>
</tr>
<tr>
<td>36263</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>11236K</td>
<td>7552K</td>
<td>kqread</td>
<td>4</td>
<td>0:00</td>
<td>0.00%</td>
<td>nginx</td>
</tr>
<tr>
<td>36867</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>11236K</td>
<td>7556K</td>
<td>kqread</td>
<td>6</td>
<td>0:00</td>
<td>0.00%</td>
<td>nginx</td>
</tr>
<tr>
<td>35024</td>
<td>nobody</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>11236K</td>
<td>7540K</td>
<td>kqread</td>
<td>6</td>
<td>0:00</td>
<td>0.00%</td>
<td>nginx</td>
</tr>
<tr>
<td>40049</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>6464K</td>
<td>2372K</td>
<td>nanslp</td>
<td>4</td>
<td>0:00</td>
<td>0.00%</td>
<td>cron</td>
</tr>
<tr>
<td>10201</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>6412K</td>
<td>2368K</td>
<td>select</td>
<td>5</td>
<td>0:00</td>
<td>0.00%</td>
<td>syslogd</td>
</tr>
<tr>
<td>34902</td>
<td>root</td>
<td>1</td>
<td>52</td>
<td>0</td>
<td>11236K</td>
<td>6920K</td>
<td>pause</td>
<td>4</td>
<td>0:00</td>
<td>0.00%</td>
<td>nginx</td>
</tr>
</tbody>
</table>
```
7.17 update

The `update` command targets a release instead of a container. Because every container is based on a release, when the release is updated all the containers are automatically updated as well.

If no updates are available, a message will be shown:

```
ishmael ~ # bastille update 11.2-RELEASE
Looking up update.FreeBSD.org mirrors... 2 mirrors found.
Fetching metadata signature for 11.2-RELEASE from update4.freebsd.org... done.
Fetching metadata index... done.
Inspecting system... done.
Preparing to download files... done.

No updates needed to update system to 11.2-RELEASE-p4.
No updates are available to install.
```

The older the release, however, the more updates will be available:

```
ishmael ~ # bastille update 10.4-RELEASE
Looking up update.FreeBSD.org mirrors... 2 mirrors found.
Fetching metadata signature for 10.4-RELEASE from update1.freebsd.org... done.
Fetching metadata index... done.
Fetching 2 metadata patches... done.
Applying metadata patches... done.
Inspecting system... done.
Preparing to download files... done.

The following files will be added as part of updating to 10.4-RELEASE-p13:
...[snip]...
```

To be safe, you may want to restart any containers that have been updated live.

7.18 upgrade

This command lets you upgrade a release to a new release. Depending on the workflow this can be similar to a `bootstrap`.

```
ishmael ~ # bastille upgrade 11.2-RELEASE 12.0-RELEASE
```

7.19 verify

This command scans a bootedstrapped release and validates that everything looks in order. This is not a 100% comprehensive check, but it compares the release against a “known good” index.

If you see errors or issues here, consider deleting and re-bootstraping the release.

```
ishmael ~ # bastille verify 11.2-RELEASE
Looking up update.FreeBSD.org mirrors... 2 mirrors found.
Fetching metadata signature for 11.2-RELEASE from update1.freebsd.org... done.
Fetching metadata index... done.
Fetching 1 metadata patches... done.
```
Applying metadata patches... done.
Fetching 1 metadata files... done.
Inspecting system... done.
CHAPTER 8

Template

Bastille supports a templating system allowing you to apply files, pkgs and execute commands inside the containers automatically.

Currently supported template hooks are: PRE, OVERLAY, PKG, SYSRC, CMD. Planned template hooks include: FSTAB, PF, LOG.

Templates are created in ${bastille_prefix}/templates and can leverage any of the template hooks. Simply create a new directory named after the template. eg:

```bash
mkdir -p /usr/local/bastille/templates/username/base
```

To leverage a template hook, create an UPPERCASE file in the root of the template directory named after the hook you want to execute. eg:

```bash
echo "zsh vim-console git-lite htop" > /usr/local/bastille/templates/username/base/PKG
echo "/usr/bin/chsh -s /usr/local/bin/zsh" > /usr/local/bastille/templates/username/base/CMD
echo "etc\nroot\nusr" > /usr/local/bastille/templates/username/base/OVERLAY
```

Template hooks are executed in specific order and require specific syntax to work as expected. This table outlines those requirements:

<table>
<thead>
<tr>
<th>HOOK</th>
<th>format</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRE</td>
<td>/bin/sh command</td>
<td>mkdir -p /usr/local/my_app/html</td>
</tr>
<tr>
<td>OVERLAY</td>
<td>path(s)</td>
<td>etc root usr (one per line)</td>
</tr>
<tr>
<td>PKG</td>
<td>port/pkg name(s)</td>
<td>vim-console zsh git-lite tree htop</td>
</tr>
<tr>
<td>SYSRC</td>
<td>syrc command(s)</td>
<td>nginx_enable=YES</td>
</tr>
<tr>
<td>SERVICE</td>
<td>service command</td>
<td>‘nginx start’ OR ‘postfix reload’</td>
</tr>
<tr>
<td>CMD</td>
<td>/bin/sh command</td>
<td>/usr/bin/chsh -s /usr/local/bin/zsh</td>
</tr>
</tbody>
</table>

Note: SYSRC requires that NO quotes be used or that quotes (”) be escaped. ie; “”

In addition to supporting template hooks, Bastille supports overlaying files into the container. This is done by placing the files in their full path, using the template directory as “/”.

29
An example here may help. Think of `bastille/templates/username/base`, our example template, as the root of our filesystem overlay. If you create an `etc/hosts` or `etc/resolv.conf` inside the base template directory, these can be overlaid into your container.

Note: due to the way FreeBSD segregates user-space, the majority of your overlayed template files will be in `usr/local`. The few general exceptions are the `etc/hosts`, `etc/resolv.conf`, and `etc/rc.conf.local`.

After populating `usr/local/` with custom config files that your container will use, be sure to include `usr` in the template `OVERLAY` definition. eg:

```
echo "etc\usr" > /usr/local/bastille/templates/username/base/OVERLAY
```

The above example “etc usr” will include anything under “etc” and “usr” inside the template. You do not need to list individual files. Just include the top-level directory name. List these top-level directories one per line.

### 8.1 Applying Templates

Containers must be running to apply templates.

Bastille includes a `template` command. This command requires a target and a template name. As covered in the previous section, template names correspond to directory names in the `bastille/templates` directory.

ishmael ~ # bastille template ALL username/base
[proxy01]:
Copying files...
Copy complete.
Installing packages.
pkg already bootstrapped at /usr/local/sbin/pkg
vulnxml file up-to-date
0 problem(s) in the installed packages found.
Updating bastillebsd.org repository catalogue...
[cdn] Fetching meta.txz: 100% 560 B 0.6kB/s 00:01
[cdn] Fetching packagesite.txz: 100% 121 KiB 124.3kB/s 00:01
Processing entries: 100%
bastillebsd.org repository update completed. 499 packages processed.
All repositories are up to date.
Checking integrity... **done** (0 conflicting)
The most recent version of packages are already installed
Updating services.
cron_flags: -J 60 -> -J 60
sendmail_enable: NONE -> NONE
syslogd_flags: -ss -> -ss
Executing final command(s).
chsh: user information updated
Template Complete.

[web01]:
Copying files...
Copy complete.
Installing packages.
pkg already bootstrapped at /usr/local/sbin/pkg
vulnxml file up-to-date
0 problem(s) in the installed packages found.
Updating pkg.bastillebsd.org repository catalogue...
[poudriere] Fetching meta.txz: 100% 560 B 0.6kB/s 00:01
[poudriere] Fetching packagesite.txz: 100% 121 KiB 124.3kB/s 00:01
(continues on next page)
Processing entries: 100%
pkg.bastillebsd.org repository update completed. 499 packages processed.
Updating bastillebsd.org repository catalogue...
[poudriere] Fetching meta.txz: 100% 560 B 0.6kB/s 00:01
[poudriere] Fetching packagesite.txz: 100% 121 KiB 124.3kB/s 00:01
Processing entries: 100%
bastillebsd.org repository update completed. 499 packages processed.
All repositories are up to date.
Checking integrity... done (0 conflicting)
The most recent version of packages are already installed
Updating services.
cron_flags: -J 60 -> -J 60
sendmail_enable: NONE -> NONE
syslogd_flags: -ss -> -ss
Executing final command(s).
chsh: user information updated
Template Complete.

Note: FreeBSD introduced container technology twenty years ago, long before the industry standardized on the term “container”. Internally, FreeBSD refers to these containers as “jails.”
In this section we’ll look at the default config for a new container. The defaults are sane for most applications, but if you want to tweak the settings here they are.

A jail.conf template is used each time a new container is created. This template looks like this:

```plaintext
interface = {interface};
host.hostname = {name};
exec.consolelog = /usr/local/bastille/logs/{name}_console.log;
path = /usr/local/bastille/jails/{name}/root;
ip6 = disable;
securelevel = 2;
devfs_ruleset = 4;
enforce_statfs = 2;
exec.start = '/bin/sh /etc/rc';
exec.stop = '/bin/sh /etc/rc.shutdown';
exec.clean;
mount.devfs;
mount.fstab = /usr/local/bastille/jails/{name}/fstab;

{name} {
ip4.addr = x.x.x.x;
}
```

### 9.1 interface

**interface**

A network interface to add the jail’s IP addresses (ip4.addr and ip6.addr) to. An alias for each address will be added to the interface before the jail is created, and will be removed from the interface after the jail is removed.
9.2 host.hostname

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host.hostname</td>
<td>The hostname of the jail. Other similar parameters are host.domainname, host.hostuuid and host.hostid.</td>
</tr>
</tbody>
</table>

9.3 exec.consolelog

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exec.consolelog</td>
<td>A file to direct command output (stdout and stderr) to.</td>
</tr>
</tbody>
</table>

9.4 path

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The directory which is to be the root of the jail. Any commands run inside the jail, either by jail or from jexec(8), are run from this directory.</td>
</tr>
</tbody>
</table>

9.5 securelevel

By default, Bastille containers run at securelevel = 2. See below for the implications of kernel security levels and when they might be altered.

Note: Bastille does not currently have any mechanism to automagically change securelevel settings. My recommendation is this only be altered manually on a case-by-case basis and that “Highly secure mode” is a sane default for most use cases.

The kernel runs with five different security levels. Any super-user process can raise the level, but no process can lower it. The security levels are:

-1 Permanently insecure mode - always run the system in insecure mode. This is the default initial value.

0 Insecure mode - immutable and append-only flags may be turned off. All devices may be read or written subject to their permissions.

1 Secure mode - the system immutable and system append-only flags may not be turned off; disks for mounted file systems, /dev/mem and /dev/kmem may not be opened for writing; /dev/io (if your platform has it) may not be opened at all; kernel modules (see kld(4)) may not be loaded or unloaded. The kernel debugger may not be entered using the debug.kdb.enter sysctl. A panic or trap cannot be forced using the debug.kdb.panic and other sysctl's.

2 Highly secure mode - same as secure mode, plus disks may not be opened for writing (except by mount(2)) whether mounted or not. This level precludes tampering with file systems by unmounting them, but also inhibits running newfs(8) while the system is multi-
user.

In addition, kernel time changes are restricted to less than or equal to one second. Attempts to change the time by more than this will log the message "Time adjustment clamped to +1 second".

Network secure mode - same as highly secure mode, plus IP packet filter rules (see ipfw(8), ipfirewall(4) and pfctl(8)) cannot be changed and dummynet(4) or pf(4) configuration cannot be adjusted.

9.6 devfs_ruleset

devfs_ruleset
The number of the devfs ruleset that is enforced for mounting devfs in this jail. A value of zero (default) means no ruleset is enforced. Descendant jails inherit the parent jail's devfs ruleset enforcement. Mounting devfs inside a jail is possible only if the allow.mount and allow.mount.devfs permissions are effective and enforce_statfs is set to a value lower than 2. Devfs rules and rulesets cannot be viewed or modified from inside a jail.

NOTE: It is important that only appropriate device nodes in devfs be exposed to a jail; access to disk devices in the jail may permit processes in the jail to bypass the jail sandboxing by modifying files outside of the jail. See devfs(8) for information on how to use devfs rules to limit access to entries in the per-jail devfs. A simple devfs ruleset for jails is available as ruleset #4 in /etc/defaults/devfs.rules.

9.7 enforce_statfs

enforce_statfs
This determines what information processes in a jail are able to get about mount points. It affects the behaviour of the following syscalls: statfs(2), fstatfs(2), getfsstat(2), and fhstatfs(2) (as well as similar compatibility syscalls). When set to 0, all mount points are available without any restrictions. When set to 1, only mount points below the jail's chroot directory are visible. In addition to that, the path to the jail's chroot directory is removed from the front of their pathnames. When set to 2 (default), above syscalls can operate only on a mount-point where the jail's chroot directory is located.

9.8 exec.start
exec.start

Command(s) to run in the jail environment when a jail is created.
A typical command to run is "sh /etc/rc".

9.9 exec.stop

exec.stop

Command(s) to run in the jail environment before a jail is
removed, and after any exec.prestop commands have completed. A
typical command to run is "sh /etc/rc.shutdown".

9.10 exec.clean

exec.clean

Run commands in a clean environment. The environment is
discarded except for HOME, SHELL, TERM and USER. HOME and SHELL
are set to the target login's default values. USER is set to the
target login. TERM is imported from the current environment.
The environment variables from the login class capability
database for the target login are also set.

9.11 mount.devfs

mount.devfs

Mount a devfs(5) filesystem on the chrooted /dev directory, and
apply the ruleset in the devfsrules parameter (or a default of
ruleset 4: devfsrules_jail) to restrict the devices visible
inside the jail.

9.12 mount.fstab

mount.fstab

An fstab(5) format file containing filesystems to mount before
creating a jail.
This content is copyright Christer Edwards. All rights reserved.
Duplication of this content without the express written permission of the author is not permitted.
Note: this documentation is included with the source code in docs.