Chapter 17. Jails and Containers

Table of Contents

17.1. Synopsis
17.2. Jail Types
17.3. Host Configuration
17.4. Classic Jail (Thick Jail)
17.5. Thin Jails
17.6. Jail Management
17.7. Jail Upgrading
17.8. Jail Resource Limits
17.9. Jail Managers and Containers

17.1. Synopsis

Since system administration is a difficult task, many tools have been developed to make life easier for the administrator. These tools often enhance the way systems are installed, configured, and maintained. One of the tools which can be used to enhance the security of a FreeBSD system is *jails*. Jails have been available since FreeBSD 4.X and continue to be enhanced in their usefulness, performance, reliability, and security.

Jails build upon the chroot(2) concept, which is used to change the root directory of a set of processes. This creates a safe environment, separate from the rest of the system. Processes created in the chrooted environment can not access files or resources outside of it. For that reason, compromising a service running in a chrooted environment should not allow the attacker to compromise the entire system.

However, a chroot has several limitations. It is suited to easy tasks which do not require much flexibility or complex, advanced features. Over time, many ways have been found to escape from a chrooted environment, making it a less than ideal solution for securing services.

Jails improve on the concept of the traditional chroot environment in several ways.

In a traditional chroot environment, processes are only limited in the part of the file system they can access. The rest of the system resources, system users, running processes, and the networking subsystem are shared by the chrooted processes and the processes of the host system. Jails expand this model by virtualizing access to the file system, the set of users, and the networking subsystem. More fine-grained controls are available for tuning the access of a jailed environment. Jails can be considered as a type of operating system-level virtualization.

This chapter covers:

- What a jail is and what purpose it may serve in FreeBSD installations.
- The different types of jail.
- The different ways to configure the network for a jail.
- The jail configuration file.
- How to create the different types of jail.
- How to start, stop, and restart a jail.
- The basics of jail administration, both from inside and outside the jail.
- How to upgrade the different types of jail.
- A incomplete list of the different FreeBSD jail managers.

17.2. Jail Types

Some administrators divide jails into different types, although the underlying technology is the same. Each administrator will have to assess what type of jail to create in each case depending on the problem they have to solve.

Below can be found a list of the different types, their characteristics, and considerations for use.

17.2.1. Thick Jails

A thick jail is a traditional form of FreeBSD Jail. In a thick jail, a complete copy of the base system is replicated within the jail's environment. This means that the jail has its own separate instance of the FreeBSD base system, including libraries, executables, and configuration files. The jail can be thought of as an almost complete standalone FreeBSD installation, but running within the confines of the host system. This isolation

ensures that the processes within the jail are kept separate from those on the host and other jails.

Advantages of Thick Jails:

- High degree of isolation: Processes within the jail are isolated from the host system and other jails.
- Independence: Thick jails can have different versions of libraries, configurations, and software than the host system or other jails.
- Security: Since the jail contains its own base system, vulnerabilities or issues affecting the jail environment will not directly impact the host or other jails.

Disadvantages of Thick Jails:

- Resource overhead: Because each jail maintains its own separate base system, thick jails consume more resources compared to thin jails.
- Maintenance: Each jail requires its own maintenance and updates for its base system components.

17.2.2. Thin Jails

A thin jail shares the base system using OpenZFS snapshots or NullFS mounts from a template. Only a minimal subset of base system is duplicated for each thin jail, resulting in less resource consumption compared to a thick jail. However, this also means that thin jails have less isolation and independence compared to thick jails. Changes in shared components could potentially affect multiple thin jails simultaneously.

In summary, a FreeBSD Thin Jail is a type of FreeBSD Jail that replicates a substantial portion, but not all, of the base system within the isolated environment.

Advantages of Thin Jails:

- Resource Efficiency: Thin jails are more resource-efficient compared to thick jails. Since they share most of the base system, they consume less disk space and memory. This makes it possible to run more jails on the same hardware without consuming excessive resources.
- Faster Deployment: Creating and launching thin jails is generally faster compared to thick jails. This can be particularly advantageous when rapidly deploying multiple instances.
- Unified Maintenance: Since thin jails share the majority of their base system with

the host system, updates and maintenance of common base system components (such as libraries and binaries) only need to be done once on the host. This simplifies the maintenance process compared to maintaining an individual base system for each thick jail.

- Shared Resources: Thin jails can more easily share common resources such as libraries and binaries with the host system. This can potentially lead to more efficient disk caching and improved performance for applications within the jail.

Disadvantages of Thin Jails:

- Reduced Isolation: The primary disadvantage of thin jails is that they offer less isolation compared to thick jails. Since they share a significant portion of the template's base system, vulnerabilities or issues affecting shared components could potentially impact multiple jails simultaneously.
- Security Concerns: The reduced isolation in thin jails could pose security risks, as a compromise in one jail might have a greater potential to affect other jails or the host system.
- Dependency Conflicts: If multiple thin jails require different versions of the same libraries or software, managing dependencies can become complex. In some cases, this might require additional effort to ensure compatibility.
- Compatibility Challenges: Applications within a thin jail might encounter compatibility issues if they assume a certain base system environment that differs from the shared components provided by the template.

17.2.3. Service Jails

A service jail shares the complete filesystem tree directly with the host (the jail root path is **/**) and as such can access and modify any file on the host, and shares the same user accounts with the host. By default it has no access to the network or other resources which are restricted in jails, but they can be configured to re-use the network of the host and to remove some of the jail-restrictions. The use case for service jails is automatic confinement of services/daemons inside a jail with minimal configuration, and without any knowledge of the files needed by such service/daemon. Service jails exist since FreeBSD 15.

Advantages of Service Jails:

- Zero Administration: A service jail ready service needs only one config line in **/etc/rc.conf**, a service which is not service jails ready needs two config lines.
- Resource Efficiency: Service jails are more resource efficient than thin jails, as

they do not need any additional disk space or network resource.

- Faster Deployment: Creating and launching service jails is generally faster compared to thin jails if only distinct services/daemons shall be jailed and no parallel instances of the same service/daemon is needed.
- Shared Resources: Service jails share all resources such as libraries and binaries with the host system. This can potentially lead to more efficient disk caching and improved performance for applications within the jail.
- Process Isolation: Service jails isolate a particular service, it can not see processes which are not a child of the service jail, even if they run within the same user account.

Disadvantages of Service Jails:

- Reduced Isolation: The primary disadvantage of service jails is that they offer no filesystem isolation compared to thick or thin jails.
- Security Concerns: The reduced isolation in service jails could pose security risks, as a compromise in one jail might have a greater potential to affect everything on the host system.

Most of the configuration of jails which is discussed below is not needed for service jails. To understand how jails work, it is recommended to understand those configuration possibilities. The details about what is needed to configure a service jail is in Configuring service jails.

17.2.4. VNET Jails

A FreeBSD VNET jail is a virtualized environment that allows for the isolation and control of network resources for processes running within it. It provides a high level of network segmentation and security by creating a separate network stack for processes within the jail, ensuring that network traffic within the jail is isolated from the host system and other jails.

In essence, FreeBSD VNET jails add a network configuration mechanism. This means a VNET jail can be created as a Thick or Thin Jail.

17.2.5. Linux Jails

A FreeBSD Linux Jail is a feature in the FreeBSD operating system that enables the use of Linux binaries and applications within a FreeBSD jail. This functionality is achieved by incorporating a compatibility layer that allows certain Linux system calls

and libraries to be translated and executed on the FreeBSD kernel. The purpose of a Linux Jail is to facilitate the execution of Linux software on a FreeBSD system without needing a separate Linux virtual machine or environment.

17.3. Host Configuration

Before creating any jail on the host system it is necessary to perform certain configuration and obtain some information from the host system.

It will be necessary to configure the jail(8) utility, create the necessary directories to configure and install jails, obtain information from the host's network, and check whether the host uses OpenZFS or UFS as its file system.

> **Tip**
> The FreeBSD version running in the jail can not be newer than the version running in the host.

17.3.1. Jail Utility

The jail(8) utility manages jails.

To start jails when the system boots, run the following commands:

```
# sysrc jail_enable="YES"
# sysrc jail_parallel_start="YES"
```

> **Tip**
> With `jail_parallel_start`, all configured jails will be started in the background.

17.3.2. Networking

Networking for FreeBSD jails can be configured several different ways:

Host Networking Mode (IP Sharing)

In host networking mode, a jail shares the same networking stack as the host system. When a jail is created in host networking mode it uses the same network interface and IP address. This means that the jail does not have a separate IP address, and its network traffic is associated with the host's IP.

Virtual Networks (VNET)

Virtual Networks are a feature of FreeBSD jails that offer more advanced and flexible networking solutions than a basic networking mode like host networking. VNET allows the creation of isolated network stacks for each jail, providing them with their own separate IP addresses, routing tables, and network interfaces. This offers a higher level of network isolation and allows jails to function as if they are running on separate virtual machines.

The netgraph system

netgraph(4) is a versatile kernel framework for creating custom network configurations. It can be used to define how network traffic flows between jails and the host system and between different jails.

17.3.3. Setting Up the Jail Directory Tree

There is no specific place to put the files for the jails.

Some administrators use **/jail**, others **/usr/jail**, and still others **/usr/local/jails**. In this chapter **/usr/local/jails** will be used.

Apart from **/usr/local/jails** other directories will be created:

- **media** will contain the compressed files of the downloaded userlands.
- **templates** will contain the templates when using Thin Jails.
- **containers** will contain the jails.

When using OpenZFS, execute the following commands to create datasets for these directories:

```
# zfs create -o mountpoint=/usr/local/jails zroot/jails
# zfs create zroot/jails/media
# zfs create zroot/jails/templates
# zfs create zroot/jails/containers
```

Tip

> **Tip**
> In this case, `zroot` was used for the parent dataset, but other datasets could have been used.

When using UFS, execute the following commands to create the directories:

```
# mkdir /usr/local/jails/
# mkdir /usr/local/jails/media
# mkdir /usr/local/jails/templates
# mkdir /usr/local/jails/containers
```

17.3.4. Jail Configuration Files

There are two ways to configure jails.

The first one is to add an entry for each jail to the file **/etc/jail.conf**. The other option is to create a file for each jail in the directory **/etc/jail.conf.d/**.

In case a host system has few jails, an entry for each jail can be added in the file **/etc/jail.conf**. If the host system has many jails, it is a good idea to have one configuration file for each jail in the **/etc/jail.conf.d/** directory.

The files in **/etc/jail.conf.d/** must have `.conf` as their extension and have to be included in **/etc/jail.conf**:

```
.include "/etc/jail.conf.d/*.conf";
```

A typical jail entry would look like this:

```
jailname {  1
  # STARTUP/LOGGING
  exec.start = "/bin/sh /etc/rc";  2
  exec.stop = "/bin/sh /etc/rc.shutdown";  3
  exec.consolelog = "/var/log/jail_console_${name}.log";  4

  # PERMISSIONS
  allow.raw_sockets;  5
  exec.clean;  6
  mount.devfs;  7

  # HOSTNAME/PATH
```

```
  host.hostname = "${name}";  8
  path = "/usr/local/jails/containers/${name}";  9

  # NETWORK
  ip4.addr = 192.168.1.151;  10
  ip6.addr = ::ffff:c0a8:197  11
  interface = em0;  12
}
```

1 `jailname` - Name of the jail.

2 `exec.start` - Command(s) to run in the jail environment when a jail is created. A typical command to run is "/bin/sh /etc/rc".

3 `exec.stop` - Command(s) to run in the jail environment before a jail is removed. A typical command to run is "/bin/sh /etc/rc.shutdown".

4 `exec.consolelog` - A file to direct command output (stdout and stderr) to.

5 `allow.raw_sockets` - Allow creating raw sockets inside the jail. Setting this parameter allows utilities like ping(8) and traceroute(8) to operate inside the jail.

6 `exec.clean` - Run commands in a clean environment.

7 `mount.devfs` - Mount a devfs(5) filesystem on the chrooted **/dev** directory, and apply the ruleset in the devfs_ruleset parameter to restrict the devices visible inside the jail.

8 `host.hostname` - The hostname of the jail.

9 `path` - The directory which is to be the root of the jail. Any commands that are run inside the jail, either by jail or from jexec(8), are run from this directory.

10 `ip4.addr` - IPv4 address. There are two configuration possibilities for IPv4. The first is to establish an IP or a list of IPs as has been done in the example. The other is to use `ip4` instead and set the `inherit` value to inherit the host's IP address.

11 `ip6.addr` - IPv6 address. There are two configuration possibilities for IPv6. The first is to establish an IP or a list of IPs as has been done in the example. The other is to use `ip6` instead and set the `inherit` value to inherit the host's IP address.

12 `interface` - A network interface to add the jail's IP addresses. Usually the host interface.

More information about configuration variables can be found in jail(8) and jail.conf(5).

17.4. Classic Jail (Thick Jail)

These jails resemble a real FreeBSD system. They can be managed more or less like a normal host system and updated independently.

17.4.1. Creating a Classic Jail

In principle, a jail only needs a hostname, a root directory, an IP address, and a userland.

The userland for the jail can be obtained from the official FreeBSD download servers.

Execute the following command to download the userland:

```
# fetch https://download.freebsd.org/ftp/releases/amd64/amd64/14.2-RELEASE/base.txz -c
```

Once the download is complete, it will be necessary to extract the contents into the jail directory.

Execute the following commands to extract the userland into the jail's directory:

```
# mkdir -p /usr/local/jails/containers/classic
# tar -xf /usr/local/jails/media/14.2-RELEASE-base.txz -C /usr/local/jails/containers/
```

With the userland extracted in the jail directory, it will be necessary to copy the timezone and DNS server files:

```
# cp /etc/resolv.conf /usr/local/jails/containers/classic/etc/resolv.conf
# cp /etc/localtime /usr/local/jails/containers/classic/etc/localtime
```

With the files copied, the next thing to do is update to the latest patch level by executing the following command:

```
# freebsd-update -b /usr/local/jails/containers/classic/ fetch install
```

The last step is to configure the jail. It will be necessary to add an entry to the configuration file **/etc/jail.conf** or in **jail.conf.d** with the parameters of the jail.

An example would be the following:

```
classic {
  # STARTUP/LOGGING
  exec.start = "/bin/sh /etc/rc";
  exec.stop = "/bin/sh /etc/rc.shutdown";
  exec.consolelog = "/var/log/jail_console_${name}.log";

  # PERMISSIONS
  allow.raw_sockets;
  exec.clean;
  mount.devfs;

  # HOSTNAME/PATH
  host.hostname = "${name}";
  path = "/usr/local/jails/containers/${name}";

  # NETWORK
  ip4.addr = 192.168.1.151;
  interface = em0;
}
```

Execute the following command to start the jail:

```
# service jail start classic
```

More information on how to manage jails can be found in the section Jail Management.

17.5. Thin Jails

Although Thin Jails use the same technology as Thick Jails, the creation procedure is different. Thin jails can be created using OpenZFS snapshots or using templates and NullFS. The use of OpenZFS snapshots and templates using NullFS have certain advantages over classic jails, such as being able to create them faster from snapshots or being able to update multiple jails using NullFS.

17.5.1. Creating a Thin Jail Using OpenZFS Snapshots

Due to the good integration between FreeBSD and OpenZFS it is very easy to create

new Thin Jails using OpenZFS Snapshots.

To create a Thin Jail using OpenZFS Snapshots the first step is to create the jail directory tree by following the instructions in Setting up the Jail Directory Tree.

Next, create a template. Templates will only be used to create new jails. For this reason they are created in "read-only" mode so that jails are created with an immutable base.

To create the dataset for the template, execute the following command:

```
# zfs create -p zroot/jails/templates/14.2-RELEASE
```

Then execute the following command to download the userland:

```
# fetch https://download.freebsd.org/ftp/releases/amd64/amd64/14.2-RELEASE/base.txz -o
```

Once the download is complete, it will be necessary to extract the contents in the template directory by executing the following command:

```
# tar -xf /usr/local/jails/media/14.2-RELEASE-base.txz -C /usr/local/jails/templates/1
```

With the userland extracted in the templates directory, it will be necessary to copy the timezone and DNS server files to the template directory by executing the following command:

```
# cp /etc/resolv.conf /usr/local/jails/templates/14.2-RELEASE/etc/resolv.conf
# cp /etc/localtime /usr/local/jails/templates/14.2-RELEASE/etc/localtime
```

The next thing to do is update to the latest patch level by executing the following command:

```
# freebsd-update -b /usr/local/jails/templates/14.2-RELEASE/ fetch install
```

Once the update is finished, the template is ready.

To create an OpenZFS Snapshot from the template, execute the following command:

```
# zfs snapshot zroot/jails/templates/14.2-RELEASE@base
```

Once the OpenZFS Snapshot has been created, infinite jails can be created using the OpenZFS clone function.

To create a Thin Jail named `thinjail`, execute the following command:

```
# zfs clone zroot/jails/templates/14.2-RELEASE@base zroot/jails/containers/thinjail
```

The last step is to configure the jail. It will be necessary to add an entry to the configuration file **/etc/jail.conf** or in **jail.conf.d** with the parameters of the jail.

An example would be the following:

```
thinjail {
  # STARTUP/LOGGING
  exec.start = "/bin/sh /etc/rc";
  exec.stop = "/bin/sh /etc/rc.shutdown";
  exec.consolelog = "/var/log/jail_console_${name}.log";

  # PERMISSIONS
  allow.raw_sockets;
  exec.clean;
  mount.devfs;

  # HOSTNAME/PATH
  host.hostname = "${name}";
  path = "/usr/local/jails/containers/${name}";

  # NETWORK
  ip4 = inherit;
  interface = em0;
}
```

Execute the following command to start the jail:

```
# service jail start thinjail
```

More information on how to manage jails can be found in the section Jail

Management.

17.5.2. Creating a Thin Jail Using NullFS

A jail can be created with reduced duplication of system files by using the Thin Jail technique and using NullFS to selectively share specific directories from the host system into the jail.

The first step is to create the dataset to save the template, execute the following command if using OpenZFS:

```
# zfs create -p zroot/jails/templates/14.2-RELEASE-base
```

Or this one if using UFS:

```
# mkdir /usr/local/jails/templates/14.2-RELEASE-base
```

Then execute the following command to download the userland:

```
# fetch https://download.freebsd.org/ftp/releases/amd64/amd64/14.2-RELEASE/base.txz -c
```

Once the download is complete, it will be necessary to extract the contents in the template directory by executing the following command:

```
# tar -xf /usr/local/jails/media/14.2-RELEASE-base.txz -C /usr/local/jails/templates/1
```

Once the userland is extracted in the templates directory, it will be necessary to copy the timezone and DNS server files to the template directory by executing the following command:

```
# cp /etc/resolv.conf /usr/local/jails/templates/14.2-RELEASE-base/etc/resolv.conf
# cp /etc/localtime /usr/local/jails/templates/14.2-RELEASE-base/etc/localtime
```

With the files moved to the template, the next thing to do is update to the latest patch level by executing the following command:

```
# freebsd-update -b /usr/local/jails/templates/14.2-RELEASE-base/ fetch install
```

In addition to the base template, it is also necessary to create a directory where the `skeleton` will be located. Some directories will be copied from the template to the `skeleton`.

Execute the following command to create the dataset for the `skeleton` in case of using OpenZFS:

```
# zfs create -p zroot/jails/templates/14.2-RELEASE-skeleton
```

Or this one in case of using UFS:

```
# mkdir /usr/local/jails/templates/14.2-RELEASE-skeleton
```

Then create the `skeleton` directories. The `skeleton` directories will hold the local directories of the jails.

Execute the following commands to create the directories:

```
# mkdir -p /usr/local/jails/templates/14.2-RELEASE-skeleton/home
# mkdir -p /usr/local/jails/templates/14.2-RELEASE-skeleton/usr
# mv /usr/local/jails/templates/14.2-RELEASE-base/etc /usr/local/jails/templates/14.2-
# mv /usr/local/jails/templates/14.2-RELEASE-base/usr/local /usr/local/jails/templates
# mv /usr/local/jails/templates/14.2-RELEASE-base/tmp /usr/local/jails/templates/14.2-
# mv /usr/local/jails/templates/14.2-RELEASE-base/var /usr/local/jails/templates/14.2-
# mv /usr/local/jails/templates/14.2-RELEASE-base/root /usr/local/jails/templates/14.2
```

The next step is to create the symlinks to the `skeleton` by executing the following commands:

```
# cd /usr/local/jails/templates/14.2-RELEASE-base/
# mkdir skeleton
# ln -s skeleton/etc etc
# ln -s skeleton/home home
# ln -s skeleton/root root
# ln -s ../skeleton/usr/local usr/local
# ln -s skeleton/tmp tmp
```

```
# ln -s skeleton/var var
```

With the `skeleton` ready, it will be necessary to copy the data to the jail directory.

In case of using OpenZFS, OpenZFS snapshots can be used to easily create as many jails as necessary by executing the following commands:

```
# zfs snapshot zroot/jails/templates/14.2-RELEASE-skeleton@base
# zfs clone zroot/jails/templates/14.2-RELEASE-skeleton@base zroot/jails/containers/th
```

In case of using UFS the cp(1) program can be used by executing the following command:

```
# cp -R /usr/local/jails/templates/14.2-RELEASE-skeleton /usr/local/jails/containers/t
```

Then create the directory in which the base template and the skeleton will be mounted:

```
# mkdir -p /usr/local/jails/thinjail-nullfs-base
```

Add a jail entry in **/etc/jail.conf** or a file in **jail.conf.d** as follows:

```
thinjail {
  # STARTUP/LOGGING
  exec.start = "/bin/sh /etc/rc";
  exec.stop = "/bin/sh /etc/rc.shutdown";
  exec.consolelog = "/var/log/jail_console_${name}.log";

  # PERMISSIONS
  allow.raw_sockets;
  exec.clean;
  mount.devfs;

  # HOSTNAME/PATH
  host.hostname = "${name}";
  path = "/usr/local/jails/${name}-nullfs-base";

  # NETWORK
  ip4.addr = 192.168.1.153;
```

```
  interface = em0;

  # MOUNT
  mount.fstab = "/usr/local/jails/${name}-nullfs-base.fstab";
}
```

Then the create the **/usr/local/jails/thinjail-nullfs-base.fstab** file as follows:

```
/usr/local/jails/templates/14.2-RELEASE-base  /usr/local/jails/thi
/usr/local/jails/containers/thinjail     /usr/local/jails/thinjail
```

Execute the following command to start the jail:

```
# service jail start thinjail
```

17.5.3. Creating a VNET Jail

FreeBSD VNET Jails have their own distinct networking stack, including interfaces, IP addresses, routing tables, and firewall rules.

The first step to create a VNET jail is to create the bridge(4) by executing the following command:

```
# ifconfig bridge create
```

The output should be similar to the following:

```
bridge0
```

With the `bridge` created, it will be necessary to attach it to the `em0` interface and bring both of them up by executing the following commands:

```
# ifconfig bridge0 addm em0 up
# ifconfig em0 up
```

To make this setting persist across reboots, add the following lines to **/etc/rc.conf**:

```
defaultrouter="192.168.1.1"
cloned_interfaces="bridge0"
ifconfig_bridge0="inet 192.168.1.150/24 addm em0 up"
ifconfig_em0="up"
```

For more information on bridging, see Network Bridging.

The next step is to create the jail as indicated above.

Either the Classic Jail (Thick Jail) procedure and the Thin Jails procedure can be used. The only thing that will change is the configuration in the **/etc/jail.conf** file.

The path **/usr/local/jails/containers/vnet** will be used as an example for the created jail.

The following is an example configuration for a VNET jail:

```
vnet {
  # STARTUP/LOGGING
  exec.consolelog = "/var/log/jail_console_${name}.log";

  # PERMISSIONS
  allow.raw_sockets;
  exec.clean;
  mount.devfs;
  devfs_ruleset = 5;

  # PATH/HOSTNAME
  path = "/usr/local/jails/containers/${name}";
  host.hostname = "${name}";

  # VNET/VIMAGE
  vnet;
  vnet.interface = "${epair}b";

  # NETWORKS/INTERFACES
  $id = "154";  1
  $ip = "192.168.1.${id}/24";
  $gateway = "192.168.1.1";
  $bridge = "bridge0";  2
  $epair = "epair${id}";

  # ADD TO bridge INTERFACE
  exec.prestart  = "/sbin/ifconfig ${epair} create up";
```

```
    exec.prestart += "/sbin/ifconfig ${epair}a up descr jail:${name}
    exec.prestart += "/sbin/ifconfig ${bridge} addm ${epair}a up";
    exec.start    += "/sbin/ifconfig ${epair}b ${ip} up";
    exec.start    += "/sbin/route add default ${gateway}";
    exec.start    += "/bin/sh /etc/rc";
    exec.stop     = "/bin/sh /etc/rc.shutdown";
    exec.poststop = "/sbin/ifconfig ${bridge} deletem ${epair}a";
    exec.poststop += "/sbin/ifconfig ${epair}a destroy";
}
```

1 Represents the IP of the Jail, it must be **unique**.

2 Refers to the bridge created previously.

17.5.4. Creating a Linux Jail

FreeBSD can run Linux inside a jail using Linux Binary Compatibility and debootstrap(8). Jails do not have a kernel. They run on the host's kernel. Therefore it is necessary to enable Linux Binary Compatibility in the host system.

To enable the Linux ABI at boot time, execute the following command:

```
# sysrc linux_enable="YES"
```

Once enabled, it can be started without rebooting by executing the following command:

```
# service linux start
```

The next step will be to create a jail as indicated above, for example in Creating a Thin Jail Using OpenZFS Snapshots, but **without** performing the configuration. FreeBSD Linux jails require a specific configuration that will be detailed below.

Once the jail has been created as explained above, execute the following command to perform required configuration for the jail and start it:

```
# jail -cm \
    name=ubuntu \
    host.hostname="ubuntu.example.com" \
    path="/usr/local/jails/ubuntu" \
    interface="em0" \
```

```
    ip4.addr="192.168.1.150" \
    exec.start="/bin/sh /etc/rc" \
    exec.stop="/bin/sh /etc/rc.shutdown" \
    mount.devfs \
    devfs_ruleset=4 \
    allow.mount \
    allow.mount.devfs \
    allow.mount.fdescfs \
    allow.mount.procfs \
    allow.mount.linprocfs \
    allow.mount.linsysfs \
    allow.mount.tmpfs \
    enforce_statfs=1
```

To access the jail, it will be necessary to install sysutils/debootstrap.

Execute the following command to access the FreeBSD Linux jail:

```
# jexec -u root ubuntu
```

Inside the jail, execute the following commands to install sysutils/debootstrap and prepare the Ubuntu environment:

```
# pkg install debootstrap
# debootstrap jammy /compat/ubuntu
```

When the process has finished and the message `Base system installed successfully` is displayed on the console, it will be necessary to stop the jail from the host system by executing the following command:

```
# service jail onestop ubuntu
```

Then add an entry in **/etc/jail.conf** for the Linux jail:

```
ubuntu {
  # STARTUP/LOGGING
  exec.start = "/bin/sh /etc/rc";
  exec.stop = "/bin/sh /etc/rc.shutdown";
  exec.consolelog = "/var/log/jail_console_${name}.log";
```

```
    # PERMISSIONS
    allow.raw_sockets;
    exec.clean;
    mount.devfs;
    devfs_ruleset = 4;

    # HOSTNAME/PATH
    host.hostname = "${name}";
    path = "/usr/local/jails/containers/${name}";

    # NETWORK
    ip4.addr = 192.168.1.155;
    interface = em0;

    # MOUNT
    mount += "devfs     $path/compat/ubuntu/dev     devfs     rw  0
    mount += "tmpfs     $path/compat/ubuntu/dev/shm tmpfs     rw,siz
    mount += "fdescfs   $path/compat/ubuntu/dev/fd  fdescfs   rw,lin
    mount += "linprocfs $path/compat/ubuntu/proc    linprocfs rw  0
    mount += "linsysfs  $path/compat/ubuntu/sys     linsysfs  rw  0
    mount += "/tmp      $path/compat/ubuntu/tmp     nullfs    rw  0
    mount += "/home     $path/compat/ubuntu/home    nullfs    rw  0
}
```

Then the jail can be started as usual with the following command:

```
# service jail start ubuntu
```

The Ubuntu environment can be accessed using the following command:

```
# jexec ubuntu chroot /compat/ubuntu /bin/bash
```

More information can be found in the chapter Linux Binary Compatibility.

17.5.5. Configuring Service Jails

A service jail is configured completely via **/etc/rc.conf** or sysrc(8). The base system services are service jails ready. They contain a config line which enables networking or lift other restrictions of jails. Base system services which do not make sense to r inside jails are configured to not be started as a service jail, even if enabled in **/etc/**

rc.conf. Some examples of such a service are services which want to mount or unmount something in the start of stop method, or only configure something like a route, or firewall, or the like.

Third party services may or may not be service jails ready. To check if a service is service jail ready, the following command can be used:

```
# grep _svcj_options /path/to/rc.d/servicename
```

If there is no output, the service is not service jail ready, or does not need any additional privileges like for example, network access.

If the service is not service jail ready, and needs network access, it can be made ready by adding the necessary config to **/etc/rc.conf**:

```
# sysrc servicename_svcj_options=net_basic
```

For all possible `_svcj_options` see the rc.conf(5) man-page.

To enable a service jail for a given service, the service needs to be stopped and the `servicename_svcj` variable needs to be set to YES. To put syslogd(8) into a service jail, use the following sequence of commands:

```
# service syslogd stop
# sysrc syslogd_svcj=YES
# service syslogd start
```

If the `servicename_svcj` variable is changed, the service needs to be stopped before it is changed. If it is not stopped, the rc framework will not detect the correct state of the service and will not be able to do what is requested.

Service jails are managed only via rc.conf(5)/sysrc(8) and the service(8) command. The jail utilities, like jls(8) as described in Jail Management can be used to investigate the operation, but the jail(8) command is not supposed to be used to manage them.

17.6. Jail Management

Once the jail is created, there are a number of operations that can be performed, like starting, rebooting or deleting the jail, installing software in it, etc. In this section the different actions that can be done with jails from the host will be described.

17.6.1. List Running Jails

To list the jails that are running on the host system, the command jls(8) can be used:

```
# jls
```

The output should be similar to the following:

```
   JID  IP Address      Hostname                      Path
     1  192.168.250.70  classic                       /usr/local/j
```

jls(8) supports the `--libxo` argument, which through the libxo(3) library allows other types of formats to be displayed, such as `JSON`, `HTML`, etc.

For example, execute the following command to get the `JSON` output:

```
# jls --libxo=json
```

The output should be similar to the following:

```
{"__version": "2", "jail-information": {"jail": [{"jid":1,"ipv4":"
```

17.6.2. Start, Restart, and Stop a Jail

service(8) is used to start, reboot, or stop a jail on the host.

For example, to start a jail, run the following command:

```
# service jail start jailname
```

Change the `start` argument to `restart` or `stop` to perform other actions on the jail.

17.6.3. Destroy a Jail

Destroying a jail is not as simple as stopping the jail using service(8) and removing the

jail directory and **/etc/jail.conf** entry.

FreeBSD takes system security very seriously. For this reason there are certain files that not even the root user can delete. This functionality is known as File Flags.

The first step is to stop the desired jail executing the following command:

```
# service jail stop jailname
```

The second step is to remove these flags with chflags(1) by executing the following command, in which `classic` is the name of the jail to remove:

```
# chflags -R 0 /usr/local/jails/containers/classic
```

The third step is to delete the directory where the jail was:

```
# rm -rf /usr/local/jails/containers/classic
```

Finally, it will be necessary to remove the jail entry in **/etc/jail.conf** or in **jail.conf.d**.

17.6.4. Handle Packages in a Jail

The pkg(8) tool supports the `-j` argument in order to handle packages installed inside the jail.

For example, to install www/nginx-lite in the jail, the next command can be executed **from the host**:

```
# pkg -j classic install nginx-lite
```

For more information on working with packages in FreeBSD, see Installing Applications: Packages and Ports.

17.6.5. Access a Jail

While it has been stated above that it is best to manage jails from the host system, a jail can be entered with jexec(8).

The jail can be entered by running jexec(8) from the host:

```
# jexec -u root jailname
```

When gaining access to the jail, the message configured in motd(5) will be displayed.

17.6.6. Execute Commands in a Jail

To execute a command from the host system in a jail the jexec(8) can be used.

For example, to stop a service that is running inside a jail, the command will be executed:

```
# jexec -l jailname service nginx stop
```

17.7. Jail Upgrading

Upgrading FreeBSD Jails ensures that the isolated environments remain secure, up-to-date, and in line with the latest features and improvements available in the FreeBSD ecosystem.

17.7.1. Upgrading a Classic Jail or a Thin Jail using OpenZFS Snapshots

Jails **must be updated from the host** operating system. The default behavior in FreeBSD is to disallow the use of chflags(1) in a jail. This will prevent the update of some files so updating from within the jail will fail.

To update the jail to the latest patch release of the version of FreeBSD it is running, execute the following commands on the host:

```
# freebsd-update -j classic fetch install
# service jail restart classic
```

To upgrade the jail to a new major or minor version, first upgrade the host system as described in Performing Major and Minor Version Upgrades. Once the host has been upgraded and rebooted, the jail can then be upgraded.

> **Tip**
> In case of upgrade from one version to another, it is easier to create a new jail than to upgrade completely.

For example to upgrade from 13.1-RELEASE to 13.2-RELEASE, execute the following commands on the host:

```
# freebsd-update -j classic -r 13.2-RELEASE upgrade
# freebsd-update -j classic install
# service jail restart classic
# freebsd-update -j classic install
# service jail restart classic
```

> **Note**
> It is necessary to execute the `install` step two times. The first one upgrades the kernel, and the second one upgrades the rest of the components.

Then, if it was a major version upgrade, reinstall all installed packages and restart the jail again. This is required because the ABI version changes when upgrading between major versions of FreeBSD.

From the host:

```
# pkg -j jailname upgrade -f
# service jail restart jailname
```

17.7.2. Upgrading a Thin Jail Using NullFS

Since Thin Jails that use NullFS share the majority of system directories, they are very easy to update. It is enough to update the template. This allows updating multiple jails at the same time.

To update the template to the latest patch release of the version of FreeBSD it is running, execute the following commands on the host:

```
# freebsd-update -b /usr/local/jails/templates/13.1-RELEASE-base/ fetch install
# service jail restart
```

To upgrade the template to a new major or minor version, first upgrade the host system as described in Performing Major and Minor Version Upgrades. Once the host has been upgraded and rebooted, the template can then be upgraded.

For example, to upgrade from 13.1-RELEASE to 13.2-RELEASE, execute the following commands on the host:

```
# freebsd-update -b /usr/local/jails/templates/13.1-RELEASE-base/ -r 13.2-RELEASE upgr
# freebsd-update -b /usr/local/jails/templates/13.1-RELEASE-base/ install
# service jail restart
# freebsd-update -b /usr/local/jails/templates/13.1-RELEASE-base/ install
# service jail restart
```

17.8. Jail Resource Limits

Controlling the resources that a jail uses from the host system is a task to be taken into account by the system administrator.

Use rctl(8) to manage the resources that a jail can use from the host system.

> **Tip**
> The `kern.racct.enable` tunable must be enabled at **/boot/loader.conf**.

The syntax to limit the resources of a jail is as follows:

```
rctl -a jail:<jailname>:resource:action=amount/percentage
```

For example, to limit the maximum RAM that a jail can access, run the following command:

```
# rctl -a jail:classic:memoryuse:deny=2G
```

To make the limitation persistent across reboots of the host system, it will be

necessary to add the rule to the **/etc/rctl.conf** file as follows:

```
jail:classic:memoryuse:deny=2G/jail
```

More information on resource limits can be found in the security chapter in the Resource Limits section.

17.9. Jail Managers and Containers

As previously explained, each type of FreeBSD Jail can be created and configured manually, but FreeBSD also has third-party utilities to make configuration and administration easier.

Below is an incomplete list of the different FreeBSD Jail managers:

Table 1. Jail Managers

Name	License	Package	Documentation
BastilleBSD	BSD-3	sysutils/bastille	Documentation
pot	BSD-3	sysutils/pot	Documentation
cbsd	BSD-2	sysutils/cbsd	Documentation
AppJail	BSD-3	sysutils/appjail, for devel sysutils/appjail-devel	Documentation
iocage	BSD-2	sysutils/iocage	Documentation
ezjail	Beer Ware	sysutils/ezjail	Documentation

Last modified on: August 11, 2025 by Mark Phillips

English System

About

FreeBSD

FreeBSD Foundation

Get FreeBSD

Code of Conduct

Security Advisories

Documentation

Documentation portal

Manual pages

Presentations and papers

Previous versions

4.4BSD Documents

Wiki

Community

Get involved

Community forum

Mailing lists

IRC Channels

Bug Tracker

Legal

Donations

Licensing

Privacy Policy

Legal notices

Made with ♥ by the FreeBSD Community