NAME

jail - manage system jails

SYNOPSIS

From Configuration File

jail [**-cm**] [**-dqv**] [**-f** *conf_file*] [**-p** *limit*] [*jail*]
jail [**-r**] [**-qv**] [**-f** *conf_file*] [**-p** *limit*] [***** | *jail* ...]

Without Configuration File

jail [**-cm**] [**-dhilqv**] [**-J** *jid_file*] [**-u** *username*] [**-U** *username*] *param=value* ... [**command**=*command* ...]
jail [**-rR**] [**-qv**] [***** | *jail* ...]

Show Parameters

jail [**-f** *conf_file*] **-e** *separator*

Backward Compatibility

jail [**-dhilqv**] [**-J** *jid_file*] [**-u** *username*] [**-U** *username*] [**-n** *jailname*] [**-s** *securelevel*]
path hostname ip[**,**...] *command* ...

DESCRIPTION

The **jail** utility creates new jails, or modifies or removes existing jails. It can also print a list of configured jails and their parameters. A jail (or "prison") is specified via parameters on the command line, or in the jail.conf(5) file.

At least one of the options **-c**, **-e**, **-m** or **-r** must be specified. These options are used alone or in combination to describe the operation to perform:

-c Create a new jail. The jail *jid* and *name* parameters (if specified on the command line) must not refer to an existing jail.

-e *separator*
Exhibit a list of all configured non-wildcard jails and their parameters. No jail creation, modification or removal performed if this option is used. The *separator* string is used to separate parameters. Use jls(8) utility to list running jails.

-m Modify an existing jail. One of the *jid* or *name* parameters must exist and refer to an existing jail. Some parameters may not be changed on a running jail.

-r Remove the *jail* specified by jid or name. All jailed processes are killed, and all jails that are children of this jail are also removed.

-rc Restart an existing jail. The jail is first removed and then re-created, as if "**jail -r**" and "**jail -c**" were run in succession.

-cm Create a jail if it does not exist, or modify the jail if it does exist.

-mr Modify an existing jail. The jail may be restarted if necessary to modify parameters than could not otherwise be changed.

-cmr Create a jail if it doesn't exist, or modify (and possibly restart) the jail if it does exist.

Other available options are:

-d Allow making changes to a dying jail, equivalent to the *allow.dying* parameter.

-f *conf_file*
Use configuration file *conf_file* instead of the default */etc/jail.conf*.

-h Resolve the *host.hostname* parameter (or *hostname*) and add all IP addresses returned by the resolver to the list of addresses for this jail. This is equivalent to the *ip_hostname* parameter.

-i Output (only) the jail identifier of the newly created jail(s). This implies the **-q** option.

-J *jid_file*
Write a *jid_file* file, containing the parameters used to start the jail.

-l Run commands in a clean environment. This is deprecated and is equivalent to the exec.clean parameter.

-n *jailname*
Set the jail's name. This is deprecated and is equivalent to the *name* parameter.

-p *limit*
Limit the number of commands from *exec.** that can run simultaneously.

-q Suppress the message printed whenever a jail is created, modified or removed. Only error messages will be printed.

-R A variation of the **-r** option that removes an existing jail without using the configuration file. No removal-related parameters for this jail will be used -- the jail will simply be removed.

-s *securelevel*
Set the *kern.securelevel* MIB entry to the specified value inside the newly created jail. This is deprecated and is equivalent to the *securelevel* parameter.

-u *username*
The user name from host environment as whom jailed commands should run. This is deprecated and is equivalent to the *exec.jail_user* and *exec.system_jail_user* parameters.

-U *username*
The user name from the jailed environment as whom jailed commands should run. This is deprecated and is equivalent to the *exec.jail_user* parameter.

-v Print a message on every operation, such as running commands and mounting filesystems.

If no arguments are given after the options, the operation (except remove) will be performed on all jails specified in the jail.conf(5) file. A single argument of a jail name will operate only on the specified jail. The **-r** and **-R** options can also remove running jails that aren't in the jail.conf(5) file, specified by name or jid.

An argument of "*" is a wildcard that will operate on all jails, regardless of whether they appear in jail.conf(5); this is the surest way for **-r** to remove all jails. If hierarchical jails exist, a partial-matching wildcard definition may be specified. For example, an argument of "foo.*" would apply to jails with names like "foo.bar" and "foo.bar.baz".

A jail may also be specified via parameters directly on the command line in "name=value" form, ignoring the contents of jail.conf(5). For backward compatibility, the command line may also have four fixed parameters, without names: *path*, *hostname*, *ip*, and *command*.

Jail Parameters

Parameters in the jail.conf(5) file, or on the command line, are generally of the form "name=value". Some parameters are boolean, and do not have a value but are set by the name alone with or without a "no" prefix, e.g. *persist* or *nopersist*. They can also be given the values "true" and "false". Other parameters may have more than one value, specified as a comma-separated list or with "+=" in the configuration file (see jail.conf(5) for details).

The **jail** utility recognizes two classes of parameters. There are the true jail parameters that are passed to the kernel when the jail is created, which can be seen with jls(8), and can (usually) be changed with "**jail -m**". Then there are pseudo-parameters that are only used by **jail** itself.

Jails have a set of core parameters, and kernel modules can add their own jail parameters. The current

set of available parameters can be retrieved via "**sysctl -d** *security.jail.param*". Any parameters not set will be given default values, often based on the current environment. The core parameters are:

jid The jail identifier. This will be assigned automatically to a new jail (or can be explicitly set), and can be used to identify the jail for later modification, or for such commands as jls(8) or jexec(8).

name The jail name. This is an arbitrary string that identifies a jail (except it may not contain a '.'). Like the *jid*, it can be passed to later **jail** commands, or to jls(8) or jexec(8). If no *name* is supplied, a default is assumed that is the same as the *jid*. The *name* parameter is implied by the jail.conf(5) file format, and need not be explicitly set when using the configuration file.

path 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.

ip4.addr
A list of IPv4 addresses assigned to the jail. If this is set, the jail is restricted to using only these addresses. Any attempts to use other addresses fail, and attempts to use wildcard addresses silently use the jailed address instead. For IPv4 the first address given will be used as the source address when source address selection on unbound sockets cannot find a better match. It is only possible to start multiple jails with the same IP address if none of the jails has more than this single overlapping IP address assigned to itself.

ip4.saddrsel
A boolean option to change the formerly mentioned behaviour and disable IPv4 source address selection for the jail in favour of the primary IPv4 address of the jail. Source address selection is enabled by default for all jails and the *ip4.nosaddrsel* setting of a parent jail is not inherited for any child jails.

ip4 Control the availability of IPv4 addresses. Possible values are "inherit" to allow unrestricted access to all system addresses, "new" to restrict addresses via *ip4.addr*, and "disable" to stop the jail from using IPv4 entirely. Setting the *ip4.addr* parameter implies a value of "new".

ip6.addr, *ip6.saddrsel*, *ip6*
A set of IPv6 options for the jail, the counterparts to *ip4.addr*, *ip4.saddrsel* and *ip4* above.

vnet Create the jail with its own virtual network stack, with its own network interfaces, addresses, routing table, etc. The kernel must have been compiled with the **VIMAGE option** for this to be available. Possible values are "inherit" to use the system network stack, possibly with restricted IP addresses, and "new" to create a new network stack.

host.hostname

The hostname of the jail. Other similar parameters are *host.domainname*, *host.hostuuid* and *host.hostid*.

host Set the origin of hostname and related information. Possible values are "inherit" to use the system information and "new" for the jail to use the information from the above fields. Setting any of the above fields implies a value of "new".

securelevel

The value of the jail's *kern.securelevel* sysctl. A jail never has a lower securelevel than its parent system, but by setting this parameter it may have a higher one. If the system securelevel is changed, any jail securelevels will be at least as secure.

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*.

children.max

The number of child jails allowed to be created by this jail (or by other jails under this jail). This limit is zero by default, indicating the jail is not allowed to create child jails. See the *Hierarchical Jails* section for more information.

children.cur

The number of descendants of this jail, including its own child jails and any jails created under them.

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.

persist
Setting this boolean parameter allows a jail to exist without any processes. Normally, a command is run as part of jail creation, and then the jail is destroyed as its last process exits. A new jail must have either the *persist* parameter or *exec.start* or *command* pseudo-parameter set.

cpuset.id
The ID of the cpuset associated with this jail (read-only).

dying This is true if the jail is in the process of shutting down (read-only).

parent The *jid* of the parent of this jail, or zero if this is a top-level jail (read-only).

osrelease
The string for the jail's *kern.osrelease* sysctl and uname -r.

osreldate
The number for the jail's *kern.osreldate* and uname -K.

*allow.**
Some restrictions of the jail environment may be set on a per-jail basis. With the exception of *allow.set_hostname* and *allow.reserved_ports*, these boolean parameters are off by default.

allow.set_hostname
The jail's hostname may be changed via hostname(1) or sethostname(3).

allow.sysvipc
A process within the jail has access to System V IPC primitives. This is deprecated in favor of the per-module parameters (see below). When this parameter is set, it is equivalent to setting *sysvmsg*, *sysvsem*, and *sysvshm* all to "inherit".

allow.raw_sockets
The jail root is allowed to create raw sockets. Setting this parameter allows utilities like ping(8) and traceroute(8) to operate inside the jail. If this is set, the source IP addresses are enforced to comply with the IP address bound to the jail, regardless of whether or not the IP_HDRINCL flag has been set on the socket. Since raw sockets can be used to configure and interact with various network subsystems, extra caution should be used

where privileged access to jails is given out to untrusted parties.

allow.chflags
: Normally, privileged users inside a jail are treated as unprivileged by chflags(2). When this parameter is set, such users are treated as privileged, and may manipulate system file flags subject to the usual constraints on *kern.securelevel*.

allow.mount
: privileged users inside the jail will be able to mount and unmount file system types marked as jail-friendly. The lsvfs(1) command can be used to find file system types available for mount from within a jail. This permission is effective only if *enforce_statfs* is set to a value lower than 2.

allow.mount.devfs
: privileged users inside the jail will be able to mount and unmount the devfs file system. This permission is effective only together with *allow.mount* and only when *enforce_statfs* is set to a value lower than 2. The devfs ruleset should be restricted from the default by using the *devfs_ruleset* option.

allow.quotas
: The jail root may administer quotas on the jail's filesystem(s). This includes filesystems that the jail may share with other jails or with non-jailed parts of the system.

allow.read_msgbuf
: Jailed users may read the kernel message buffer. If the *security.bsd.unprivileged_read_msgbuf* MIB entry is zero, this will be restricted to the root user.

allow.socket_af
: Sockets within a jail are normally restricted to IPv4, IPv6, local (UNIX), and route. This allows access to other protocol stacks that have not had jail functionality added to them.

allow.mlock
: Locking or unlocking physical pages in memory are normally not available within a jail. When this parameter is set, users may mlock(2) or munlock(2) memory subject to *security.bsd.unprivileged_mlock* and resource limits.

allow.nfsd
: The mountd(8), nfsd(8), nfsuserd(8), gssd(8) and rpc.tlsservd(8) daemons are permitted to run inside a properly configured vnet-enabled jail. The jail's root must be a file system

mount point and *enforce_statfs* must not be set to 0, so that mountd(8) can export file systems visible within the jail. *enforce_statfs* must be set to 1 if file systems mounted under the jail's file system need to be exported by mount(8). For exporting only the jail's file system, a setting of 2 is sufficient. If the kernel configuration does not include the **NFSD** option, *nfsd.ko* must be loaded outside of the jails. This is normally done by adding "nfsd" to *kld_list* in the rc.conf(5) file outside of the jails. Similarily, if the gssd(8) is to be run in a jail, either the kernel **KGSSAPI** option needs to be specified or "kgssapi" and "kgssapi_krb5" need to be in *kld_list* in the rc.conf(5) file outside of the jails.

allow.reserved_ports
The jail root may bind to ports lower than 1024.

allow.unprivileged_proc_debug
Unprivileged processes in the jail may use debugging facilities.

allow.suser
The value of the jail's *security.bsd.suser_enabled* sysctl. The super-user will be disabled automatically if its parent system has it disabled. The super-user is enabled by default.

Kernel modules may add their own parameters, which only exist when the module is loaded. These are typically headed under a parameter named after the module, with values of "inherit" to give the jail full use of the module, "new" to encapsulate the jail in some module-specific way, and "disable" to make the module unavailable to the jail. There also may be other parameters to define jail behavior within the module. Module-specific parameters include:

allow.mount.fdescfs
privileged users inside the jail will be able to mount and unmount the fdescfs file system. This permission is effective only together with *allow.mount* and only when *enforce_statfs* is set to a value lower than 2.

allow.mount.fusefs
privileged users inside the jail will be able to mount and unmount fuse-based file systems. This permission is effective only together with *allow.mount* and only when *enforce_statfs* is set to a value lower than 2.

allow.mount.nullfs
privileged users inside the jail will be able to mount and unmount the nullfs file system. This permission is effective only together with *allow.mount* and only when *enforce_statfs* is set to a value lower than 2.

allow.mount.procfs
: privileged users inside the jail will be able to mount and unmount the procfs file system. This permission is effective only together with *allow.mount* and only when *enforce_statfs* is set to a value lower than 2.

allow.mount.linprocfs
: privileged users inside the jail will be able to mount and unmount the linprocfs file system. This permission is effective only together with *allow.mount* and only when *enforce_statfs* is set to a value lower than 2.

allow.mount.linsysfs
: privileged users inside the jail will be able to mount and unmount the linsysfs file system. This permission is effective only together with *allow.mount* and only when *enforce_statfs* is set to a value lower than 2.

allow.mount.tmpfs
: privileged users inside the jail will be able to mount and unmount the tmpfs file system. This permission is effective only together with *allow.mount* and only when *enforce_statfs* is set to a value lower than 2.

allow.mount.zfs
: privileged users inside the jail will be able to mount and unmount the ZFS file system. This permission is effective only together with *allow.mount* and only when *enforce_statfs* is set to a value lower than 2. See zfs(8) for information on how to configure the ZFS filesystem to operate from within a jail.

allow.vmm
: The jail may access vmm(4). This flag is only available when the vmm(4) kernel module is loaded.

linux
: Determine how a jail's Linux emulation environment appears. A value of "inherit" will keep the same environment, and "new" will give the jail its own environment (still originally inherited when the jail is created).

linux.osname, linux.osrelease, linux.oss_version
: The Linux OS name, OS release, and OSS version associated with this jail.

sysvmsg
: Allow access to SYSV IPC message primitives. If set to "inherit", all IPC objects on the system are visible to this jail, whether they were created by the jail itself, the base system, or other jails.

If set to "new", the jail will have its own key namespace, and can only see the objects that it has created; the system (or parent jail) has access to the jail's objects, but not to its keys. If set to "disable", the jail cannot perform any sysvmsg-related system calls.

sysvsem, sysvshm
Allow access to SYSV IPC semaphore and shared memory primitives, in the same manner as *sysvmsg*.

zfs.mount_snapshot
When set to 1, jailed users may access the contents of ZFS snapshots under the filesystem's *.zfs* directory. If *allow.mount.zfs* is set, the snapshots may also be mounted.

There are pseudo-parameters that are not passed to the kernel, but are used by **jail** to set up the jail environment, often by running specified commands when jails are created or removed. The *exec.** command parameters are sh(1) command lines that are run in either the system or jail environment. They may be given multiple values, which would run the specified commands in sequence. All commands must succeed (return a zero exit status), or the jail will not be created or removed, as appropriate.

The pseudo-parameters are:

exec.prepare
Command(s) to run in the system environment to prepare a jail for creation. These commands are executed before assigning IP addresses and mounting filesystems, so they may be used to create a new jail filesystem if it does not already exist.

exec.prestart
Command(s) to run in the system environment before a jail is created.

exec.created
Command(s) to run in the system environment right after a jail has been created, but before commands (or services) get executed in the jail.

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

command
A synonym for *exec.start* for use when specifying a jail directly on the command line. Unlike other parameters whose value is a single string, *command* uses the remainder of the **jail**

command line as its own arguments.

exec.poststart
: Command(s) to run in the system environment after a jail is created, and after any *exec.start* commands have completed.

exec.prestop
: Command(s) to run in the system environment before a jail is removed.

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 jail".

exec.poststop
: Command(s) to run in the system environment after a jail is removed.

exec.release
: Command(s) to run in the system environment after all other actions are done. These commands are executed after unmounting filesystems and removing IP addresses, so they may be used to remove a jail filesystem if it is no longer needed.

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. PATH is set to "/bin:/usr/bin". The environment variables from the login class capability database for the target login are also set. If a user is specified (as with *exec.jail_user*), commands are run from that (possibly jailed) user's directory.

exec.jail_user
: The user to run commands as, when running in the jail environment. The default is to run the commands as the current user.

exec.system_jail_user
: This boolean option looks for the *exec.jail_user* in the system passwd(5) file, instead of in the jail's file.

exec.system_user
: The user to run commands as, when running in the system environment. The default is to run the commands as the current user.

exec.timeout

The maximum amount of time to wait for a command to complete, in seconds. If a command is still running after this timeout has passed, the jail will not be created or removed, as appropriate.

exec.consolelog

A file to direct command output (stdout and stderr) to.

exec.fib

The FIB (routing table) to set when running commands inside the jail.

stop.timeout

The maximum amount of time to wait for a jail's processes to exit after sending them a SIGTERM signal (which happens after the *exec.stop* commands have completed). After this many seconds have passed, the jail will be removed, which will kill any remaining processes. If this is set to zero, no SIGTERM is sent and the jail is immediately removed. The default is 10 seconds.

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.

ip4.addr

In addition to the IP addresses that are passed to the kernel, an interface, netmask and additional parameters (as supported by ifconfig(8)) may also be specified, in the form "*interface*|*ip-address*/*netmask param ...*". If an interface is given before the IP address, an alias for the address will be added to that interface, as it is with the *interface* parameter. If a netmask in either dotted-quad or CIDR form is given after an IP address, it will be used when adding the IP alias. If additional parameters are specified then they will also be used when adding the IP alias.

ip6.addr

In addition to the IP addresses that are passed to the kernel, an interface, prefix and additional parameters (as supported by ifconfig(8)) may also be specified, in the form "*interface*|*ip-address*/*prefix param ...*".

vnet.interface

A network interface to give to a vnet-enabled jail after is it created. The interface will automatically be released when the jail is removed.

ip_hostname

Resolve the *host.hostname* parameter and add all IP addresses returned by the resolver to the list of addresses (*ip4.addr* or *ip6.addr*) for this jail. This may affect default address selection for outgoing IPv4 connections from jails. The address first returned by the resolver for each address family will be used as the primary address.

mount

A filesystem to mount before creating the jail (and to unmount after removing it), given as a single fstab(5) line.

mount.fstab

An fstab(5) format file containing filesystems to mount before creating a jail.

mount.devfs

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

mount.fdescfs

Mount a fdescfs(5) filesystem on the chrooted */dev/fd* directory.

mount.procfs

Mount a procfs(5) filesystem on the chrooted */proc* directory.

allow.dying

Allow making changes to a *dying* jail.

depend

Specify a jail (or jails) that this jail depends on. When this jail is to be created, any jail(s) it depends on must already exist. If not, they will be created automatically, up to the completion of the last *exec.poststart* command, before any action will taken to create this jail. When jails are removed the opposite is true: this jail will be removed, up to the last *exec.poststop* command, before any jail(s) it depends on are stopped.

EXAMPLES

Jails are typically set up using one of two philosophies: either to constrain a specific application (possibly running with privilege), or to create a "virtual system image" running a variety of daemons and services. In both cases, a fairly complete file system install of FreeBSD is required, so as to provide the necessary command line tools, daemons, libraries, application configuration files, etc. However, for a virtual server configuration, a fair amount of additional work is required so as to replace the "boot"

process. This manual page documents the configuration steps necessary to support either of these steps, although the configuration steps may need to be refined based on local requirements.

Setting up a Jail Directory Tree

To set up a jail directory tree containing an entire FreeBSD distribution, the following sh(1) command script can be used:

```
D=/here/is/the/jail
cd /usr/src
mkdir -p $D
make world DESTDIR=$D
make distribution DESTDIR=$D
```

In many cases this example would put far more in the jail than needed. In the other extreme case a jail might contain only one file: the executable to be run in the jail.

We recommend experimentation, and caution that it is a lot easier to start with a "fat" jail and remove things until it stops working, than it is to start with a "thin" jail and add things until it works.

Setting Up a Jail

Do what was described in *Setting Up a Jail Directory Tree* to build the jail directory tree. For the sake of this example, we will assume you built it in */data/jail/testjail*, for a jail named "testjail". Substitute below as needed with your own directory, IP address, and hostname.

Setting up the Host Environment

First, set up the real system's environment to be "jail-friendly". For consistency, we will refer to the parent box as the "host environment", and to the jailed virtual machine as the "jail environment". Since jails are implemented using IP aliases, one of the first things to do is to disable IP services on the host system that listen on all local IP addresses for a service. If a network service is present in the host environment that binds all available IP addresses rather than specific IP addresses, it may service requests sent to jail IP addresses if the jail did not bind the port. This means changing inetd(8) to only listen on the appropriate IP address, and so forth. Add the following to */etc/rc.conf* in the host environment:

```
sendmail_enable="NO"
inetd_flags="-wW -a 192.0.2.23"
rpcbind_enable="NO"
```

192.0.2.23 is the native IP address for the host system, in this example. Daemons that run out of inetd(8) can be easily configured to use only the specified host IP address. Other daemons will need to

be manually configured -- for some this is possible through rc.conf(5) flags entries; for others it is necessary to modify per-application configuration files, or to recompile the application. The following frequently deployed services must have their individual configuration files modified to limit the application to listening to a specific IP address:

To configure sshd(8), it is necessary to modify */etc/ssh/sshd_config*.

To configure sendmail(8), it is necessary to modify */etc/mail/sendmail.cf*.

In addition, a number of services must be recompiled in order to run them in the host environment. This includes most applications providing services using rpc(3), such as rpcbind(8), nfsd(8), and mountd(8). In general, applications for which it is not possible to specify which IP address to bind should not be run in the host environment unless they should also service requests sent to jail IP addresses. Attempting to serve NFS from the host environment may also cause confusion, and cannot be easily reconfigured to use only specific IPs, as some NFS services are hosted directly from the kernel. Any third-party network software running in the host environment should also be checked and configured so that it does not bind all IP addresses, which would result in those services also appearing to be offered by the jail environments.

Once these daemons have been disabled or fixed in the host environment, it is best to reboot so that all daemons are in a known state, to reduce the potential for confusion later (such as finding that when you send mail to a jail, and its sendmail is down, the mail is delivered to the host, etc.).

Configuring the Jail

Start any jail for the first time without configuring the network interface so that you can clean it up a little and set up accounts. As with any machine (virtual or not), you will need to set a root password, time zone, etc. Some of these steps apply only if you intend to run a full virtual server inside the jail; others apply both for constraining a particular application or for running a virtual server.

Start a shell in the jail:

```
jail -c path=/data/jail/testjail mount.devfs \
        host.hostname=testhostname ip4.addr=192.0.2.100 \
        command=/bin/sh
```

Assuming no errors, you will end up with a shell prompt within the jail. You can now run bsdconfig(8) and do the post-install configuration to set various configuration options, or perform these actions manually by editing */etc/rc.conf*, etc.

- Configure */etc/resolv.conf* so that name resolution within the jail will work correctly.

- Run newaliases(1) to quell sendmail(8) warnings.
- Set a root password, probably different from the real host system.
- Set the timezone.
- Add accounts for users in the jail environment.
- Install any packages the environment requires.

You may also want to perform any package-specific configuration (web servers, SSH servers, etc), patch up */etc/syslog.conf* so it logs as you would like, etc. If you are not using a virtual server, you may wish to modify syslogd(8) in the host environment to listen on the syslog socket in the jail environment; in this example, the syslog socket would be stored in */data/jail/testjail/var/run/log*.

Exit from the shell, and the jail will be shut down.

Starting the Jail

You are now ready to restart the jail and bring up the environment with all of its daemons and other programs. Create an entry for the jail in */etc/jail.conf*:

```
testjail {
        path = /tmp/jail/testjail;
        mount.devfs;
        host.hostname = testhostname;
        ip4.addr = 192.0.2.100;
        interface = em0;
        exec.start = "/bin/sh /etc/rc";
        exec.stop = "/bin/sh /etc/rc.shutdown jail";
}
```

To start a virtual server environment, */etc/rc* is run to launch various daemons and services, and */etc/rc.shutdown* is run to shut them down when the jail is removed. If you are running a single application in the jail, substitute the command used to start the application for "/bin/sh /etc/rc"; there may be some script available to cleanly shut down the application, or it may be sufficient to go without a stop command, and have **jail** send SIGTERM to the application.

Start the jail by running:

```
jail -c testjail
```

A few warnings may be produced; however, it should all work properly. You should be able to see inetd(8), syslogd(8), and other processes running within the jail using ps(1), with the 'J' flag appearing beside jailed processes. To see an active list of jails, use jls(8). If sshd(8) is enabled in the jail

environment, you should be able to ssh(1) to the hostname or IP address of the jailed environment, and log in using the accounts you created previously.

It is possible to have jails started at boot time. Please refer to the "jail_*" variables in rc.conf(5) for more information.

Managing the Jail

Normal machine shutdown commands, such as halt(8), reboot(8), and shutdown(8), cannot be used successfully within the jail. To kill all processes from within a jail, you may use one of the following commands, depending on what you want to accomplish:

```
kill -TERM -1
kill -KILL -1
```

This will send the SIGTERM or SIGKILL signals to all processes in the jail -- be careful not to run this from the host environment! Once all of the jail's processes have died, unless the jail was created with the *persist* parameter, the jail will be removed. Depending on the intended use of the jail, you may also want to run */etc/rc.shutdown* from within the jail.

To shut down the jail from the outside, simply remove it with:

```
jail -r
```

which will run any commands specified by *exec.stop*, and then send SIGTERM and eventually SIGKILL to any remaining jailed processes.

The */proc/pid/status* file contains, as its last field, the name of the jail in which the process runs, or "-" to indicate that the process is not running within a jail. The ps(1) command also shows a 'J' flag for processes in a jail.

You can also list/kill processes based on their jail ID. To show processes and their jail ID, use the following command:

```
ps ax -o pid,jid,args
```

To show and then kill processes in jail number 3 use the following commands:

```
pgrep -lfj 3
pkill -j 3
```

or:

killall -j 3

Jails and File Systems

It is not possible to mount(8) or umount(8) any file system inside a jail unless the file system is marked jail-friendly, the jail's *allow.mount* parameter is set, and the jail's *enforce_statfs* parameter is lower than 2.

Multiple jails sharing the same file system can influence each other. For example, a user in one jail can fill the file system, leaving no space for processes in the other jail. Trying to use quota(1) to prevent this will not work either, as the file system quotas are not aware of jails but only look at the user and group IDs. This means the same user ID in two jails share a single file system quota. One would need to use one file system per jail to make this work.

Sysctl MIB Entries

The read-only entry *security.jail.jailed* can be used to determine if a process is running inside a jail (value is one) or not (value is zero).

The variable *security.jail.jail_max_af_ips* determines how may address per address family a jail may have. The default is 255.

Some MIB variables have per-jail settings. Changes to these variables by a jailed process do not affect the host environment, only the jail environment. These variables are *kern.securelevel*, *security.bsd.suser_enabled*, *kern.hostname*, *kern.domainname*, *kern.hostid*, and *kern.hostuuid*.

Hierarchical Jails

By setting a jail's *children.max* parameter, processes within a jail may be able to create jails of their own. These child jails are kept in a hierarchy, with jails only able to see and/or modify the jails they created (or those jails' children). Each jail has a read-only *parent* parameter, containing the *jid* of the jail that created it; a *jid* of 0 indicates the jail is a child of the current jail (or is a top-level jail if the current process isn't jailed).

Jailed processes are not allowed to confer greater permissions than they themselves are given, e.g., if a jail is created with *allow.nomount*, it is not able to create a jail with *allow.mount* set. Similarly, such restrictions as *ip4.addr* and *securelevel* may not be bypassed in child jails.

A child jail may in turn create its own child jails if its own *children.max* parameter is set (remember it is zero by default). These jails are visible to and can be modified by their parent and all ancestors.

Jail names reflect this hierarchy, with a full name being an MIB-type string separated by dots. For example, if a base system process creates a jail "foo", and a process under that jail creates another jail

"bar", then the second jail will be seen as "foo.bar" in the base system (though it is only seen as "bar" to any processes inside jail "foo"). Jids on the other hand exist in a single space, and each jail must have a unique jid.

Like the names, a child jail's *path* appears relative to its creator's own *path*. This is by virtue of the child jail being created in the chrooted environment of the first jail.

SEE ALSO

killall(1), lsvfs(1), newaliases(1), pgrep(1), pkill(1), ps(1), quota(1), jail_set(2), vmm(4), devfs(5), fdescfs(5), jail.conf(5), linprocfs(5), linsysfs(5), procfs(5), rc.conf(5), sysctl.conf(5), bsdconfig(8), chroot(8), devfs(8), halt(8), ifconfig(8), inetd(8), jexec(8), jls(8), mount(8), mountd(8), nfsd(8), reboot(8), rpcbind(8), sendmail(8), shutdown(8), sysctl(8), syslogd(8), umount(8)

HISTORY

The **jail** utility appeared in FreeBSD 4.0. Hierarchical/extensible jails were introduced in FreeBSD 8.0. The configuration file was introduced in FreeBSD 9.1.

AUTHORS

The jail feature was written by Poul-Henning Kamp for R&D Associates who contributed it to FreeBSD.

Robert Watson wrote the extended documentation, found a few bugs, added a few new features, and cleaned up the userland jail environment.

Bjoern A. Zeeb added multi-IP jail support for IPv4 and IPv6 based on a patch originally done by Pawel Jakub Dawidek for IPv4.

James Gritton added the extensible jail parameters, hierarchical jails, and the configuration file.

BUGS

It might be a good idea to add an address alias flag such that daemons listening on all IPs (INADDR_ANY) will not bind on that address, which would facilitate building a safe host environment such that host daemons do not impose on services offered from within jails. Currently, the simplest answer is to minimize services offered on the host, possibly limiting it to services offered from inetd(8) which is easily configurable.

NOTES

Great care should be taken when managing directories visible within the jail. For example, if a jailed process has its current working directory set to a directory that is moved out of the jail's chroot, then the process may gain access to the file space outside of the jail. It is recommended that directories always be copied, rather than moved, out of a jail.

In addition, there are several ways in which an unprivileged user outside the jail can cooperate with a privileged user inside the jail and thereby obtain elevated privileges in the host environment. Most of these attacks can be mitigated by ensuring that the jail root is not accessible to unprivileged users in the host environment. Regardless, as a general rule, untrusted users with privileged access to a jail should not be given access to the host environment.