iocage Documentation

Release 1.2

Brandon Schneider and Peter Toth

Oct 01, 2021

Contents

iocage is a jail/container manager written in Python, combining some of the best features and technologies the FreeBSD operating system has to offer. It is geared for ease of use with a simplistic and easy to learn command syntax.

FEATURES:

- Templates, basejails, and normal jails
- Easy to use
- Rapid thin provisioning within seconds
- Automatic package installation
- Virtual networking stacks (vnet)
- Shared IP based jails (non vnet)
- Dedicated ZFS datasets inside jails
- Transparent ZFS snapshot management
- Binary updates
- Export and import
- And many more!

CHAPTER 1

Documentation:

1.1 Install iocage

iocage is a jail and container manager merging some of the best features and technologies from the FreeBSD operating system. It is geared for ease of use with simple command syntax. Visit the iocage github for more information.

1.1.1 Using binary packages

To install using binary packages on a FreeBSD system, run:

```
sudo pkg install py38-iocage
```

1.1.2 Using github

If installing from github, the FreeBSD source tree **must** be located at `$SRC_BASE` (`/usr/src` by default).

To install from github, run these commands:

```
pkg install python38 git-lite py38-cython py38-libzfs py38-pip
git clone https://github.com/iocage/iocage
```

`make install` as root.

Tip: To install subsequent updates run: `make install` as root.

1.1.3 Using pkg(8)

It is possible to install pre-build packages using pkg(8) if using FreeBSD 10 or above.

To install using pkg(8), run:

```
sudo pkg install py38-iocage
```

1.1.4 Building Ports

iocage is in the FreeBSD ports tree as sysutils/py-iocage.

Build the ports: `cd /usr/ports/sysutils/iocage/ ; make install clean`

1.1.5 Upgrading from `iocage_legacy`

This repository replaces `iocage_legacy` . To upgrade to the current version:

1. Stop the jails (`Service iocage stop; iocage stop ALL`)
2. Back up your data.
3. Remove the old `iocage` package if it is installed (`pkg delete iocage`)
4. Install `iocage` using one of the methods above.
5. Migrate the jails by running `iocage list` as root.
6. Start the jails (`service iocage onestart`).

1.1.6 Migrating from Ezjail to Iocage

Assumptions:

- ezjail jails are located at `/usr/jails`
- iocage jails are located at `/iocage/jails`

Create Target

Create an empty jail in iocage to act as the target for the migration. The release and networking information will be updated with information from ezjail.

```
iocage create -e -n NewJail
```

Copy Old Data

Before data can be copied, another symlink must be created in the root directory. Ezjail relies on symlinks to utlilize the basejail system, however when looking in an existing jail, it's symlinked to the root.

```
% ls -ls /usr/jails/OldJail/bin
1 lrwxr-xr-x  1 root  wheel  13 Feb 22  2017 /usr/jails/OldJail/bin@ -> /basejail/bin
```

This would work fine from within a running jail, but on the host filesystem this link doesn't currently exist. Because of this, create a symlink from the basejail to the root filesystem of the jail host.

```
ln -s /usr/jails/basejail /basejail
```

Now that the link exists, copy the data from the ezjail jail directory to the iocage jail directory.

```
rsync -a --copy-links /usr/jails/OldJail/ /iocage/jails/NewJail/root/
```

Populate iocage config.json

There are 2 main parts from ezjail that need to be copied into the iocage config:

- release information
- IP address

Release

The release info can be found in the old basejail files via the `freebsd-update` executable.

```
$ grep USERLAND_VERSION= /usr/jails/basejail/bin/freebsd-version
USERLAND_VERSION="11.1-RELEASE-p6"
```

This value goes into the "release" line of `config.json`

```
"release": "11.1-RELEASE-p6",
```

IP Address

The IP addresses used in an ezjail jail are found in `/usr/local/etc/ezjail/OldJail`

```
$ grep ip= /usr/local/etc/ezjail/OldJail
export jail_OldJail_ip="em0|192.168.1.10"
```

This goes into the "ip4_addr" line of `config.json`

```
"ip4_addr": "em0|192.168.1.10/24",
```

Remember to append the subnet mask when adding network info to the iocage config.

Start the New Jail

Make sure the old jail is shut down so there won't be any IP conflicts.

```
ezjail-admin stop OldJail
```

Start the new jail with iocage

```
iocage start NewJail
```

(Optional) Update fstab

If there are other mounts in use in ezjail, these can be easily copied into iocage as well.

Ezjail fstab entries are located at `/etc/fstab.OldJail` on the host.

```
$ cat /etc/fstab.OldJail
/usr/jails/basejail /usr/jails/OldJail/basejail   nullfs   ro   0   0
/path/on/host /usr/jails/OldJail/path/in/jail   nullfs   rw   0   0
```

The basejail line isn't needed in iocage if using the default jail type, but the remaining entries need to be updated.

Edit the fstab for the iocage jail and change the paths of the mountpoint.

```
$ cat /iocage/jails/NewJail/fstab
/path/on/host /iocage/jails/NewJail/root/path/in/jail   nullfs   rw   0   0
```

1.2 Basic Usage

This section is about basic iocage usage and is meant as a "how-to" reference for new users.

Tip: Remember, command line help is always available by typing **`iocage --help`** or **`iocage [subcommand] --help`**.

---

iocage has a basic "flow" when first used. As a new user interacts with iocage for the first time, this flow guides them through initializing iocage, then interacting with newly created jails.

Tip: iocage has an experimental "color" mode enabled by setting the environment variable **`IOCAGE_COLOR`** to **TRUE**.

1.2.1 Setting Environment Variables

iocage currently has four environment variables:

Table 1: **iocage Environment Variables**

Name	Accepted Values	Description
IOCAGE_LOGFILE	FILE	File location to have iocage log into.
IOCAGE_COLOR	TRUE|FALSE	Turns on a colored CLI output.
IOCAGE_FORCE	TRUE|FALSE	Required for any automatic migrations
IOCAGE_PLUGIN_IP	IP_ADDR	This environment variable is set in a plugin jail. Use it to quickly query it with another p

The process for setting these variables depends on the shell being used. The default FreeBSD shell **`csh/tcsh`** and the **`bash/sh`** shell are different from one another and require a slightly different process for setting environment variables. For example:

In the FreeBSD shell **`csh/tcsh`** , `setenv IOCAGE_COLOR TRUE` sets the environment variable IOCAGE_COLOR to true.

In the **`bash/sh`** shell, `export IOCAGE_COLOR=TRUE` sets the environment variable IOCAGE_COLOR to true.

1.2.2 Activate iocage

Before iocage is functional, it needs to **`activate`**. Essentially, iocage needs to link with a usable zpool. In most cases, activation is automatic to the primary system zpool, but more advanced users can use **`iocage activate`** to

designate a different zpool for iocage usage.

Once iocage is ready with an active zpool, users are able to immediately begin downloading FreeBSD releases for jail creation.

1.2.3 Fetch a Release

iocage now needs to fetch a RELEASE, which is used to create jails. By default, typing **iocage fetch** opens a menu for the user to choose which release to download, as seen in this example:

```
# iocage fetch
[0] 9.3-RELEASE (EOL)
[1] 10.1-RELEASE (EOL)
[2] 10.2-RELEASE (EOL)
[3] 10.3-RELEASE
[4] 11.0-RELEASE

Type the number of the desired RELEASE
Press [Enter] to fetch the default selection: (11.0-RELEASE)
Type EXIT to quit: 4
```

Once the desired RELEASE is downloaded, the most recent patches are also applied to it.

iocage fetch also has a number of options and properties for users to fine-tune the functionality of the command.

To fetch the latest RELEASE,

```
iocage fetch -r LATEST
```

If a specific RELEASE is required, use the **-r** option:

iocage fetch -r [11.0-RELEASE]

If a specific download mirror is required, use the **-s** option:

iocage fetch -s [ftp.hostname.org]

fetch can also pull from a specific ftp directory, using the **-d** option:

iocage fetch -d [dir/]

1.2.4 Create a Jail

With a release downloaded, iocage is now able to create jails. There are two types of jails: **normal** and **base**. More details about these jail types can be found in the *Jail Types* section of this documentation.

Depending on the user's requirements, the **create** subcommand can be adjusted to create either jail type. By default, **iocage create** creates a **normal** jail, but invoking the **-b** option changes the creation to the basejail type. iocage is able to create a jail with the latest release by adding *LATEST* to the create command.

Here is an example of creating a normal jail from the latest available release:

```
# iocage create -r LATEST -n [JAIL]
```

This creates a normal jail that is a clone of the latest release.

Here is an example of creating a normal jail from the *11.0-RELEASE*:

```
# iocage create -r 11.0-RELEASE
```

This normal jail is a clone of the specified RELEASE.

To create multiple jails, use the **-c** option:

```
# iocage create -r 11.0-RELEASE -c 2
```

This example shows the numeric value after the **-c** flag is used to designate the number of jails to create. In the above example, two jails are created.

A simple basejail is created with the **-b** option:

`iocage create -b -r [RELEASE]`

After designating the type and number of jails to create with the option flags, specific jail **properties** can also be set. For example:

```
# iocage create -r 11.0-RELEASE --name myjail boot=on
```

Creates a FreeBSD 11.0-RELEASE jail with the custom name *myjail* and sets the jail to start at system boot time.

More information about jail properties is available in the iocage(8) FreeBSD manual page, accessible on a FreeBSD system by typing **`man iocage`**.

1.2.5 Listing Jails

To list all jails, use **`iocage list`**

To see all downloaded RELEASEs, use **`iocage list -r`**

View available templates with **`iocage list -t`**

1.2.6 Start, Stop, or Restart a Jail

Jails can be started, stopped, or restarted at any time. By default, new jails are in a *down* (stopped) state. To see the status of all jails, use **`iocage list`** and read the **STATE** column.

Use each jail's UUID or custom NAME to start, stop, or restart it. Partial entries are acceptable, as long as the given characters are enough to identify the desired jail. Alternately, use **ALL** to apply the command to all created jails.

Tip: Partial entries can also be supplied for any other **`iocage`** operation or subcommand.

Start

Use **`iocage start`** to start jails.

Examples:

Start a jail with the custom name **www01**:

```
iocage start www01.
```

If no custom NAME or UUID is provided by the user, **`iocage`** automatically assigns a complex UUID to a new jail. This UUID is always usable when doing **`iocage`** operations like starting a jail:

```
# iocage start 26e8e027-f00c-11e4-8f7f-3c970e80eb61
```

Partial entries are also acceptable:

```
# iocaget start www
```

```
# iocage start 26e8
```

Stop

iocage stop uses the same syntax as **iocage start**.

Examples:

```
# iocage stop www01
# iocage stop 26e8e027-f00c-11e4-8f7f-3c970e80eb61
# iocage stop 26e8
```

Restart

iocage restart also uses the same syntax as **start** and **stop**:

```
# iocage restart www01
# iocage restart 26e8e027-f00c-11e4-8f7f-3c970e80eb61
# iocage restart 26e8
```

1.2.7 Configure a Jail

Configuring the properties of an already created jail is best done with the **set** and **get** subcommands. Be sure to provide the NAME or UUID of the desired jail when using these subcommands.

Set Jail Property

iocage uses the **set** subcommand to configure jail properties.

To assign a custom note to a jail with the **notes** property:

```
# iocage set notes="This is a test jail." 26e8e027
```

The full list of jail properties is available in the iocage(8) manual page PROPERTIES section.

Get Jail Property

To view a specific jail property, use the **get** subcommand:

```
# iocage get notes 26e8e027
```

Get all properties:

Display the full list of a jail's properties:

```
# iocage get all 26e8e027 | more
```

1.2.8 Destroy a Jail

Destroy a specific jail using the **destroy** subcommand:

```
# iocage destroy www02
```

Warning: This irreversibly destroys the jail!

1.2.9 Rename a Jail

iocage allows jails to be renamed after creation and/or migration. The **iocage rename** subcommand is used to alter an existing jail's UUID or NAME. Type the command, then the UUID or name of the jail to be altered, then the desired name. This example shows using the **rename** subcommand:

```
# iocage rename jail1 TESTINGJAIL
```

1.3 Plugins

iocage plugins are a simple and very fast method to get application containers installed and configured. At its core, a plugin is a jail specifically running one program. Popular programs can be installed repeatedly with one line. Additionally, plugins are easily extended by users, offering a high level of customizability and functionality.

In structure, a plugin consists of `.json` manifest and `.png` icon files.

See what's available

To see a list of all currently available plugins, open a command line and type **iocage list -PR** or **iocage list --plugins --remote**. The full iocage plugin list is also available on GitHub.

Check which plugins are installed on the system with **iocage list --plugins** or **iocage list -P**.

Getting started with plugins

Note: iocage needs to be activated before plugins can be installed or modified!

To get started, open a command line and type **iocage fetch --plugins ip4_addr="IF|IP"**. This initial **fetch** also supports *dhcp* in the same manner as **iocage create**. The IP listed for the plugin needs to be a valid IP not already in use. Use the - *-name* flag to easily fetch a specific plugin:

```
$ iocage fetch --plugins --name plexmediaserver ip4_addr="igb0|192.168.0.91"
```

If available, plugins can also be fetched locally with **iocage fetch -P the/path/to/plugin.json ip4_addr="re0|192.168.0.100"**

Tip: Using **iocage fetch** locally is very useful when testing an in-development plugin.

After fetching a plugin, view of all its properties with **iocage get -a NAME|UUID | less**. Individual properties are found with **iocage get PROPERTY**:

```
$ iocage get type quasselcore
```

Adjust the plugin properties with **iocage set**:

```
$ iocage set PROPERTY quasselcore
```

iocage set is used to configure that plugin. In this example, a complete Quasselcore plugin is installed to a FreeNAS system, then the note of the plugin is changed:

```
[root@freenas ~]# iocage fetch --plugins --name quasselcore ip4_addr="em0|192.168.1.50
↪"
[root@freenas ~]# iocage set notes="Hello world" quasselcore
[root@freenas ~]# iocage get notes quasselcore
Hello world
```

Upgrading and updating plugins

The process for upgrading and updating plugins is exactly the same as normal jails. See *Updating Jails* or *Upgrading Jails* .

Plugin Manifest Example

Following is an example of a plugin manifest:

```
{
    "name": "default_jail_name_here",
    "release": "11.3-RELEASE",
    "artifact": "https://github.com/git_path_to_plugin_repo",
    "official": false,
    "properties": {
        "nat": 1,
        "nat_forwards": "tcp(7878:7878)"
    },
    "devfs_ruleset": {
        "paths": {"bpf*": null},
        "includes": []
    },
    "pkgs": [
    ],
    "packagesite": "http://pkg.FreeBSD.org/${ABI}/latest",
    "fingerprints": {
        "iocage-plugins": [
            {
                "function": "sha256",
                "fingerprint":
↪"b0170035af3acc5f3f3ae1859dc717101b4e6c1d0a794ad554928ca0cbb2f438"
            }
        ]
    },
    "revision": 0
}
```

- **devfs_ruleset**: It should be a valid dictionary object where "paths" must be specified. Value of "paths" is a dictionary where keys are the path to be added and the value is the mode to be used. *null* translates to *unhide*. For any include specified, please refer to the following example which shows how each path specified is added as iocage dynamically generates a devfs ruleset and how an include is added to the ruleset:

```
devfs rule -s ruleset_number add path bpf* unhide
devfs rule -s ruleset_number add include include_value
```

1.4 Networking

Jails have multiple networking options to better serve a user's needs. Traditionally, jails have only supported IP alias based networking. This is where an IP address is assigned to the host's interface and then used by the jail for network communication. This is typically known as "shared IP" based jails.

Another recently developed option is called VNET or sometimes VIMAGE. VNET is a fully virtualized networking stack which is isolated per jail. VNET abstracts virtual network interfaces to jails, which then behave in the same way as physical interfaces.

By default, iocage does not enable VNET, but users can enable and configure VNET for a jail by configuring that jail's properties using the instructions in the *Configure a Jail* section of this documentation.

The rest of this section shows more depth of the **Shared IP** and **VNET** networking options, along with instructions for *Configuring Network Interfaces*.

Warning: In the examples in this section, **em0** is used as the network adapter. **em0** is a placeholder and must be replaced with the user's specific network adapter. A network adapter is a computer hardware component that connects a computer to a computer network. In order to find the network adapter on the system run `ifconfig`.

1.4.1 Shared IP

The *Shared IP* networking option is rock solid, with over a decade of heavy use and testing.

It has no specific system requirements, as everything needed is built directly into the default GENERIC kernel.

Using Shared IP

There are a few steps to follow when setting up *Shared IP*:

Check the VNET property status

```
# iocage get vnet examplejail1
```

If **vnet** is on, disable it:

```
# iocage set vnet=off examplejail1
```

Configure an IP address

```
# iocage set ip4_addr="em0|10.1.1.10/24" examplejail1
```

If multiple addresses are desired, separate the configuration directives with a `,`:

```
# iocage set ip4_addr="em0|10.1.1.10/24,em0|10.1.1.11/24" examplejail1
```

Start the jail

```
iocage start examplejail1
```

Verify visible IP configuration in the jail

```
# iocage exec examplejail1 ifconfig
```

1.4.2 VIMAGE/VNET

VNET is considered experimental. Unexpected system crashes can occur. More details about issues with VNET are available in the *Known Issues* section of this documentation.

There are a number of required steps when configuring a jail to use VNET:

Kernel

Tip: If not required, disable SCTP.

Rebuild the kernel with these options:

```
nooptions       SCTP    # Stream Control Transmission Protocol
options         VIMAGE  # VNET/Vimage support
options         RACCT   # Resource containers
options         RCTL    # same as above
```

/etc/rc.conf

On the host node, add this bridge configuration to `/etc/rc.conf`:

```
# set up bridge interface for iocage
cloned_interfaces="bridge0"

# plumb interface em0 into bridge0
ifconfig_bridge0="addm em0 up"
ifconfig_em0="up"
```

/etc/sysctl.conf

Add these tunables to `/etc/sysctl.conf`:

```
net.inet.ip.forwarding=1        # Enable IP forwarding between interfaces
net.link.bridge.pfil_onlyip=0   # Only pass IP packets when pfil is enabled
net.link.bridge.pfil_bridge=0   # Packet filter on the bridge interface
net.link.bridge.pfil_member=0   # Packet filter on the member interface
```

Enable vnet for the jail

```
# iocage set vnet=on examplejail
```

Configure jail's default gateway

```
# iocage set defaultrouter=10.1.1.254 examplejail
```

Configure an IP address

```
iocage set ip4_addr="vnet0|10.1.1.10/24" examplejail
```

Start jail and ping the default gateway

Start the jail:

```
# iocage start examplejail
```

Open the system console inside the jail:

```
iocage console examplejail
```

Ping the previously configured default gateway:

```
# ping 10.1.1.254
```

Tips

Routes

Be sure the default gateway knows the route back to the VNET subnets.

Using VLANs

To assign a jail's traffic to a VLAN, add the VLAN interface as a bridge member, but not the VLAN's parent. For example:

```
sysrc vlans_em0="666"
sysrc ifconfig_em0_666="up"
iocage set vnet_default_interface="em0.666" examplejail
iocage set interfaces="vnet1:bridge1" examplejail
```

If using VLAN interfaces for the jail host only, on the other hand, add the parent as a bridge member, but not the VLAN interface.

```
sysrc vlans_em0="666"
sysrc ifconfig_em0_666="1.2.3.4/24"
iocage set vnet_default_interface="auto" examplejail # "em0" would also work
iocage set interfaces="vnet1:bridge1" examplejail
```

1.4.3 Configuring Network Interfaces

iocage transparently handles network configuration for both *Shared IP* and *VNET* jails.

Configuring a Shared IP Jail

IPv4

```
# iocage set ip4_addr="em0|192.168.0.10/24" examplejail
```

IPv6

```
# iocage set ip6_addr="em0|2001:123:456:242::5/64" examplejail
```

These examples add IP alias *192.168.0.10/24* and *2001:123:456::5/64* to interface *em0* of the shared IP jail, at start time.

Configuring a VNET Jail

To configure both IPv4 and IPv6:

```
# iocage set ip4_addr="vnet0|192.168.0.10/24" examplejail
# iocage set ip6_addr="vnet0|2001:123:456:242::5/64" examplejail
# iocage set defaultrouter6="2001:123:456:242::1" examplejail
```

Note: For VNET jails, a default route has to also be specified.

To create a a jail with a DHCP interface add the *dhcp=on* property:

```
# iocage create -r 11.0-RELEASE --name myjail dhcp=on
```

The *dhcp=on* property implies creating a VNET virtual network stack and enabling the Berkley Packet Filter. DHCP cannot work without VNET. More information about VNET is available in the VNET(9) FreeBSD manual page.

Tips for Configuring VNET

To start a jail with no IPv4/6 address, **set** the *ip4_addr* and *ip6_addr* properties, then the *defaultrouter* and *defaultrouter6* properties:

```
# iocage set ip4_addr=none ip6_addr=none examplejail
```

```
# iocage set defaultrouter=none defaultrouter6=none examplejail
```

Force iocage to regenerate the MAC and HW address (e.g.: after cloning a jail). This will cause the MAC and HW addresses to be regenerated when the jail is next started.

```
# iocage set vnet0_mac=none examplejail
```

1.5 Jail Types

iocage supports several different jail types:

- Clone (default)
- Basejail
- Template
- Empty
- Thickjail

All jail types have specific benefits and drawbacks, serving a variety of unique needs. This section describes and has creation examples for each of these jail types.

1.5.1 Clone (default)

Clone jails are created with:

```
# iocage create -r 11.0-RELEASE
```

Clone jails are duplicated from the appropriate RELEASE at creation time. These consume a small amount of space, preserving only the changing data.

1.5.2 Basejail

The original basejail concept was based on nullfs mounts. It was popularized by ezjail, but **iocage** basejails are a little different. Basejails in **iocage** are mounts in a jail **fstab** that are mounted at jail startup.

Create a basejail by typing:

```
iocage create -r [RELEASE] -b
```

Basejails mount their **fstab** mounts at each startup. They are ideal for environments where immediate patching or upgrading of multiple jails is required.

1.5.3 Template

Template jails are customized jails used to quickly create further custom jails.

For example, after creating a jail, the user customizes that jail's networking properties. Once satisfied, the user then changes the jail into a template with:

```
# iocage set template=yes examplejail
```

After this operation the jail is found in the *templates* list:

```
# iocage list -t
```

And new jails with the user customized networking can be created:

```
# iocage create -t examplejail -n newexamplejail
```

Template jails are convertable by setting the *template*= property.

1.5.4 Empty

Empty jails are intended for unsupported jail setups or testing. Create an empty jail with **`iocage create -e`**.

These are ideal for experimentation with unsupported RELEASES or Linux jails.

1.5.5 Thickjail

Thickjails jails are created with:

```
# iocage create -T -r 11.2-RELEASE
```

Thickjails are copied from the appropriate RELEASE at creation time. These consume a large amount of space, but are fully independent.

These are ideal for transmission or synchronization between different hosts with **`zfs send`** and **`zfs receive`**.

1.6 Best Practices

This section provides some generic guidelines and tips for working with **`iocage`** managed jails.

Use PF as a module

> This is the default setting in the *GENERIC* kernel. There seems to be a VNET bug which is only triggered when PF is directly compiled into the kernel.

Always name jails and templates!

> Use the -n option with **`iocage create`** to set a name for the jail. This helps avoid mistakes and easily identify jails.
>
> Example: `iocage create -r 11.0-RELEASE -n testjail`

Set the notes property

> Set the **notes** property to something meaningful, especially for templates and jails used infrequently.
>
> Example:

```
[root@tester ~]# iocage set notes="This is a test jail." testjail
Property: notes has been updated to This is a test jail.

[root@tester ~]# iocage get notes testjail
This is a test jail.
```

VNET

VNET provides more fine control and isolation for jails. VNET also allows jails to run their own firewalls. See *Known Issues* for more about VNET.

Discover templates!

Templates simplify using jail creation and customization, give it a try! See *Using Templates* to get started.

Use `iocage restart` instead of start/stop

Always restart a jail with the **`iocage restart -s`** command. This performs a soft restart and leaves the *VNET* stack alone, which is less stressful for both kernel and user.

Check the firewall rules

When using *IPFW* inside a *VNET* jail, put **firewall_enable="YES"** and **firewall_type="open"** into `/etc/rc.conf`. This excludes the firewall from accidentally blocking the user right from the beginning! Re-lock it once finished testing. It is also recommended to check the *PF* firewall rules on the host if jail and host rules are mixed.

Delete old snapshots

Remove unnecessary snapshots, especially from jails where data is constantly changing!

1.7 Advanced Usage

1.7.1 Clones

When a jail is cloned, iocage creates a ZFS clone filesystem. Essentially, clones are cheap, lightweight, and writable snapshots.

A clone depends on its source snapshot and filesystem. To destroy the source jail and preserve its clones, the clone must be promoted first.

Create a Clone

To clone **www01** to **www02**, run:

```
# iocage clone www01 --name www02
```

Clone a jail from an existing snapshot with:

```
# iocage clone www01@snapshotname --name www03
```

Promoting a Clone

> **Warning:** This functionality isn't fully added to iocage, and may not function as expected.

To promote a cloned jail, run:

`iocage promote [UUID | NAME]`

This reverses the *clone* and *source* jail relationship. The clone becomes the source and the source jail is demoted to a clone.

The demoted jail can now be removed:

`iocage destroy [UUID | NAME]`

1.7.2 Updating Jails

Updates are handled with the freebsd-update(8) utility. Jails can be updated while they are stopped or running.

Note: The command **iocage update [UUID | NAME]** automatically creates a backup snapshot of the jail given.

To create a backup snapshot manually, run:

iocage snapshot -n [snapshotname] [UUID | NAME]

To update a jail to latest patch level, run:

iocage update [UUID | NAME]

When updates are finished and the jail appears to function properly, remove the snapshot:

iocage snapremove [UUID|NAME]@[snapshotname]

To test updating without affecting a jail, create a clone and update the clone the same way as outlined above.

To clone a jail, run:

iocage clone [UUID|NAME] --name [testupdate]

Note: The **[-n | –name]** flag is optional. **iocage** assigns an UUID to the jail if **[-n | –name]** is not used.

1.7.3 Upgrading Jails

Upgrades are handled with the freebsd-update(8) utility. By default, the user must supply the new RELEASE for the jail's upgrade. For example:

```
# iocage upgrade examplejail -r 11.0-RELEASE
```

Tells jail *examplejail* to upgrade its RELEASE to *11.0-RELEASE*.

Note: It is recommended to keep the iocage host and jails RELEASE synchronized.

To upgrade a jail to the host's RELEASE, run:

iocage upgrade -r [11.1-RELEASE] [UUID | NAME]

This upgrades the jail to the same RELEASE as the host. This method also applies to basejails.

1.7.4 Auto-boot

Make sure **iocage_enable="YES"** is set in `/etc/rc.conf`.

To enable a jail to auto-boot during a system boot, simply run:

```
# iocage set boot=on UUID|NAME
```

Note: Setting **boot=on** during jail creation starts the jail after the jail is created.

Boot Priority

Boot order can be specified by setting the priority value:

```
iocage set priority=[20] [UUID|NAME]
```

Lower values are higher in the boot priority.

Depends Property

Use the `depends` property to require other jails to start before this one. It is space delimited. Jails listed as dependents also wait to start if those jails have listed `depends`.

Example: **`iocage set depends="foo bar" baz`**

1.7.5 Snapshot Management

iocage supports transparent ZFS snapshot management out of the box. Snapshots are point-in-time copies of data, a safety point to which a jail can be reverted at any time. Initially, snapshots take up almost no space, as only changing data is recorded.

List snapshots for a jail:

```
iocage snaplist [UUID|NAME]
```

Create a new snapshot:

```
iocage snapshot [UUID|NAME]
```

This creates a snapshot based on the current time.

1.7.6 Resource Limits (Legacy ONLY)

> **Warning:** This functionality is only available for legacy versions of **`iocage`**. It is not yet implemented in the current version. This applies to all subsections of *Resource Limits*.

`iocage` can enable optional resource limits for a jail. The outlined procedure here is meant to provide a starting point for the user.

Limit Cores or Threads

Limit a jail to a single thread or core #1:

```
iocage set cpuset=1 [UUID|TAG] iocage start [UUID|TAG]
```

List Applied Limits

List applied limits:

```
iocage limits [UUID|TAG]
```

Limit DRAM use

This example limits a jail to using 4 Gb DRAM memory (limiting RSS memory use can be done on-the-fly):

```
# iocage set memoryuse=4G:deny examplejail
```

Turn on Resource Limits

Turn on resource limiting for a jail with:

iocage set rlimits=on [UUID|TAG]

Apply limits

Apply limits to a running jail with:

iocage cap [UUID | TAG]

Check Limits

Check the currently active limits on a jail with:

iocage limits [UUID | TAG]

Limit CPU Usage by %

In this example, **iocage** limits *testjail* CPU execution to 20%, then applies the limitation to the active jail:

```
# iocage set pcpu=20:deny testjail # iocage cap testjail
```

Double check the jail's current limits to confirm the functionality:

```
# iocage limits testjail
```

1.7.7 Automatic Package Installation

Packages can be installed automatically at creation time!

Use the [-p | –pkglist] option at creation time, which needs to point to a JSON file containing one package name per line.

Note: An Internet connection is required for automatic package installations, as **pkg install** obtains packages from online repositories.

Create a `pkgs.json` file and add package names to it.

`pkgs.json`:

```
{
    "pkgs": [
    "nginx",
    "tmux"
    ]
}
```

Now, create a jail and supply `pkgs.json`:

`iocage create -r [RELEASE] -p [path-to/pkgs.json] -n [NAME]`

Note: The **[-n | –name]** flag is optional. **`iocage`** assigns an UUID to the jail if **[-n | –name]** is not used.

This installs **nginx** and **tmux** in the newly created jail.

1.8 Using Templates

Templates can save precious time!

Set up a jail and create a template from it. All packages and preconfigured settings remain available for deployment to new jails within seconds.

Any jail can be converted between jail and template as needed. Essentially, a template is just another jail which has the property **template** set to **yes**. The difference is templates are not started by **`iocage`**.

Create a template with iocage:

1. Create a jail: `# iocage create -r 11.0-RELEASE -n mytemplate`.
2. Configure the jail's networking.
3. Install packages and/or customize the jail as needed.
4. Once finished with customization, stop the jail: `# iocage stop mytemplate`.
5. It is recommended to add notes to the jail, so the specific jail customizations are easily remembered: `# iocage set notes="customized PHP,nginx jail" mytemplate`
6. Set the **template** property **on**: `# iocage set template=yes mytemplate`.
7. Find the new template with **`iocage list -t`**.

Use the created template:

Use **`iocage create -t`** to create a new jail from the new template:

1. `# iocage create -t mytemplate -n jailfromtemplate`
2. Find the new jail with **`iocage list`**.
3. Start the jail with `# iocage start jailfromtemplate`.

Done!

Customizing a template:

To make changes to the template, you will need to know whether any existing jails are based on the template. Since modifying the template will require converting it back into a jail, it cannot be the base for any jails.

*** No jails based on the template ***

1. Convert the template back into a jail:

   `iocage set template=no [UUID | NAME]`.

2. Start the jail:

   `iocage start [UUID | NAME]`.

3. Use any method you wish to connect to the jail and modify its contents.

4. Stop the jail:

   iocage stop [UUID | NAME].

5. Convert the jail back into a template:

   iocage set template=yes [UUID | NAME].

*** Jails based on the template ***

This process will create a new template, leaving the existing template and jails unaffected.

1. Create a 'thick' jail from the template, so that it will be independent from the template:

   iocage create -T -t [UUID | NAME] -n newtemplate.

2. Start the jail:

   iocage start newtemplate.

3. Use any method you wish to connect to the jail and modify its contents.

4. Stop the jail:

   iocage stop newtemplate.

5. Convert the jail into a template:

   iocage set template=yes newtemplate.

1.9 Create a Debian Buster Jail (native Linux)

This section shows the process to set up a Debian Linux jail. The examples in this section use a jail with the custom name **debjail**. Remember to replace **debjail** with your jail's UUID or NAME!

> **Warning:** This is not recommended for production use. The intention is to show **iocage** can do almost anything with jails.

Create an empty jail:

```
# iocage create -e -n debjail exec_start="/bin/true" exec_stop="/bin/true"
```

Install debootstrap on the host:

```
# pkg install debootstrap
```

Enable linux(4):

```
# sysrc linux_enable="YES"   # sysrc linux_mounts_enable="NO"   # service linux
start
```

Grab the mountpoint for the empty jail, append /root/ to it, and run debootstrap(8):

```
# iocage get mountpoint debjail
```

```
# debootstrap buster /iocage/jails/debjail/root/
```

Apart from Debian releases, like *buster* or *testing*, you can also use Ubuntu releases, eg *bionic*.

Add lines to the jail `fstab` file:

Use **iocage fstab -e [UUID | NAME]** to edit the `fstab` file of *debjail* directly. Add these lines to the file:

```
devfs    /iocage/jails/debjail/root/dev         devfs     rw          0 0
tmpfs    /iocage/jails/debjail/root/dev/shm     tmpfs     rw,size=1g,mode=1777 0 0
fdescfs  /iocage/jails/debjail/root/dev/fd      fdescfs   rw,linrdlnk 0 0
linproc  /iocage/jails/debjail/root/proc        linprocfs rw          0 0
linsys   /iocage/jails/debjail/root/sys         linsysfs  rw          0 0
```

Start the jail and attach to it:

```
# iocage start debjail
```

```
# iocage console debjail
```

The result is a 64-bit Debian Linux userland.

1.10 Known Issues

This section provides a short list of known issues.

1.10.1 Mount Path Limit

There is a known mountpoint path length limitation issue on FreeBSD. Path length has an historical 88 character limit.

This issue does not affect **iocage** jails from functioning properly, but can present challenges when diving into ZFS snapshots, like **cd** into `.zfs/snapshots`, **tar**, etc.

ZFS snapshot creation and rollback is not affected.

To work around this issue, **iocage** allows the user to assign and use a unique *NAME* for the jail. Alternately, using the **[-s | –short]** flag at jail creation tells **iocage** to assign a shortened UUID to the jail.

1.10.2 VNET/VIMAGE Issues

VNET/VIMAGE can cause unexpected system crashes when VNET enabled jails are destroyed. In other words, when the jail process is killed, removed, or stopped.

As a workaround, **iocage** allows a soft restart without destroying the jail. **iocage restart –s [UUID | NAME]** executes a soft restart of the jail.

Example:

```
# iocage restart -s examplejail
```

FreeBSD 10.1-RELEASE is stable enough to run with VNET and soft restarts. There are production machines with **iocage** and VNET jails running well over 100 days of uptime running both PF and IPFW.

VNET/VIMAGE issues w/ ALTQ

As recent as FreeBSD 10.1-RELEASE-p10, there are some *interesting* interactions between VNET/VIMAGE and the ALTernate Queueing (ALTQ) system used by PF and other routing software. When compiling a kernel, be sure these lines are **not** in the `kernconf` file (unless disabling VNET):

```
options     ALTQ
options     ALTQ_CBQ
options     ALTQ_RED
options     ALTQ_RIO
options     ALTQ_HFSC
options     ALTQ_CDNR
options     ALTQ_PRIQ
```

Otherwise, when starting a jail with VNET support enabled, the host system is likely to crash. Read a more about this issue from a 2014 mailing list post.

1.10.3 IPv6 host bind failures

In some cases, a jail with an *ip6* address may take too long adding the address to the interface. Services defined to bind specifically to the address may then fail. If this happens, add this to `sysctl.conf` to disable DAD (duplicate address detection) probe packets:

```
# disable duplicated address detection probe packets for jails
net.inet6.ip6.dad_count=0
```

Adding these lines permanently disables DAD. To set this for ONLY the current system boot, type **`sysctl net.inet6.ip6.dad_count=0`** in a command line interface (CLI). More information about this issue is available from a 2013 mailing list post.

1.11 FAQ

What is iocage? **`iocage`** is a jail management program designed to simplify jail administration tasks.

What is a jail? A *Jail* is a FreeBSD **OS virtualization** technology allowing users to run multiple copies of the operating system. Some operating systems use the term **Zones** or **Containers** for OS virtualization.

What is VNET? VNET is an independent, per jail virtual networking stack.

How do I configure network interfaces in a VNET or shared IPjail? Both are configured in the same way: **`iocage set ip4_add="[interface]|[IP]/[netmask]" [UUID | NAME]`**. For more info, please refer to the *Networking* section of this documentation.

Do I need to set my default gateway? Only if VNET is enabled. You need to assign an IP address to the **bridge** where the jail interface is attached. This IP essentially becomes the default gateway for your jail.

Can I run a firewall inside a jail? Yes, a VNET jail supports **IPFW**. *PF* is not supported inside the jail. However, you can still enable *PF* for the host. If you plan to use **IPFW** inside a jail, be sure **securelevel** is set to **2**.

Can I enable both IPFW and PF at the same time? Yes, make sure you allow traffic on both in/out for your jails.

Can I create custom jail templates? Yes, and thin provisioning is supported too!

What is a jail clone? **Clones** are ZFS clones. These are fully writable copies of the source jail.

Can I limit the CPU and Memory use? Yes, but **only** for legacy versions of **`iocage`**. Refer to the `iocage.8` manual page or *Resource Limits (Legacy ONLY)* section of this documentation for more details.

Is there a way to display resource consumption? Yes - **`iocage df`**

Is NAT supported for jails? Yes. NAT is built into FreeBSD. Treat your server as a core router/firewall. Check the FreeBSD Firewalls chapter for more details.

Will iocage work on a generic system with no ZFS pools? No. ZFS is a must. If you run a FreeBSD server, you should be using ZFS!

Is ZFS jailing supported? Yes, please refer to the `iocage.8` manual page.

1.12 Indices and tables

- genindex
- modindex
- search

Index

T

U

V