Copying and Creating Oracle® Solaris 11 Package Repositories

Part No: E21803
March 2012, E21803-03

ORACLE®

120305@25097

Contents

Preface

Copying and Creating Oracle Solaris 11 Package Repositories describes how to create a software package repository using the Oracle Solaris Image Packaging System (IPS) feature. IPS tools enable you to easily copy an existing repository or create your own repository for your own packages and easily update the packages in the repository. You can provide a file interface or a HTTP interface for users of the repository.

Who Should Use This Book

This book is for system administrators who install and manage software or assist others who install and manage software.

How This Book Is Organized

- Chapter 1, "Image Packaging System Package Repositories," discusses the benefits of providing a local IPS package repository and shows how to create a ZFS file system for your repository.
- Chapter 2, "Copying IPS Package Repositories," describes copying repositories from a file and copying repositories from an Internet location.
- Chapter 3, "Providing Access To Your Repository," explains how to enable clients to view and install packages from your repository.
- Chapter 4, "Maintaining Your Local IPS Package Repository," describes how to accomplish the following tasks:
  - Add updated packages to your repository
  - Change values of properties of your repository
  - Add packages from different sources to your repository
  - Provide access to multiple repositories on one server
  - Configure the repository depot server

Related Documentation

- *Image Packaging System Man Pages*
- Chapter 6, "Managing Services (Overview)," in *Oracle Solaris Administration: Common Tasks* describes the Oracle Solaris Service Management Facility (SMF) feature
- *Oracle Solaris Administration: ZFS File Systems*

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

TABLE P–1 Typographic Conventions

Typeface	Description	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX system prompt and superuser prompt for shells that are included in the Oracle Solaris OS. Note that the default system prompt that is displayed in command examples varies, depending on the Oracle Solaris release.

TABLE P–2 Shell Prompts

Shell	Prompt
Bash shell, Korn shell, and Bourne shell	`$`
Bash shell, Korn shell, and Bourne shell for superuser	`#`
C shell	`machine_name%`
C shell for superuser	`machine_name#`

◆ ◆ ◆ CHAPTER 1

Image Packaging System Package Repositories

Oracle Solaris 11 software is distributed in Image Packaging System (IPS) packages. IPS packages are stored in IPS package repositories, which are populated by IPS publishers.

This guide describes how to create an IPS package repository. This chapter gives reasons that you might want to create a local IPS package repository for internal use.

Local IPS Repositories

You might want a local IPS repository for the following reasons:

- **Performance and security.** You do not want your client systems to go to the Internet to retrieve new software packages or update existing packages.
- **Replication.** You want to ensure that you can perform the same installation next year that you perform today.
- **Custom packages.** You want to include your own IPS package in the same repository with Oracle Solaris OS packages.

IPS supports two types of repositories: origin repositories and mirror repositories. To achieve the performance and security goals mentioned above, the local repository you create should be an origin repository. An *origin* repository contains all of the metadata (such as catalogs, manifests, and search indexes) and content (files) for one or more packages. A *mirror* repository contains only package content (files). Clients that install and update packages from a mirror repository must still download metadata from an origin repository. IPS clients access the origin to obtain a publisher's catalog, even when the clients download package content from a mirror.

Both of the repository copying methods discussed in this document create an origin repository. An origin repository is implicitly created when you `pkgrecv` a package repository, and the repository ISO files provided by Oracle provide an origin repository.

Prepare the Repository Host System

The system that hosts the IPS package repository can be either an x86-based or a SPARC-based system.

Operating system
: The IPS repository server must be running the same or a newer version of the Oracle Solaris 11 OS as the version for which the packages you plan to copy are built. For example, if the server is running Oracle Solaris 11 Express and you want to create a copy of the Oracle Solaris 11 repository, update the server to Oracle Solaris 11 before you copy the repository.

Disk space
: To host a copy of the Oracle Solaris 11 release repository, the repository server must have 15 gigabytes of free space.

Recommended best practice is to create a separate ZFS file system for your local package repository. Using a separate ZFS file system enables you to take advantage of the following benefits:

- Achieve better performance.
- Set separate file system characteristics.
- Directly snapshot and recover specified file systems.

If one system hosts more than one IPS repository, make each repository a separate ZFS file system so that you can rollback and recover each repository separately.

Use the `zfs list` command to view your current ZFS datasets.

```
$ zfs list
NAME                    USED  AVAIL  REFER  MOUNTPOINT
rpool                  75.2G   108G  5.00G  /rpool
rpool/ROOT             23.0G   108G    31K  legacy
rpool/ROOT/solaris     44.8G   108G  3.52G  /
rpool/dump             1.97G   108G  1.97G  -
rpool/export           43.0G   108G  30.5G  /export
rpool/export/home      12.6G   108G    32K  /export/home
rpool/export/home/bob  12.6G   108G  12.6G  /export/home/bob
rpool/swap             2.09G   108G  1.97G  -
```

Assume the root role:

```
$ su - root
```

Create a ZFS file system for the package repository in the root pool:

```
# zfs create rpool/export/repoSolaris11
# zfs list
NAME                        USED  AVAIL  REFER  MOUNTPOINT
rpool                      75.2G   108G  5.00G  /rpool
rpool/export/repoSolaris11   31K   108G    31K  /export/repoSolaris11
...
```

Tip – For better performance when updating the repository, set `atime` to `off`.

```
# zfs set atime=off rpool/export/repoSolaris11
```

The `atime` property controls whether the access time for files is updated when the files are read. Turning this property off avoids producing write traffic when reading files.

◆ ◆ ◆ CHAPTER 2

Copying IPS Package Repositories

This chapter describes two ways to create a copy of the Oracle Solaris 11 release IPS package repository: You can use the repository file from media or from the Oracle Solaris 11 download site, or you can retrieve a repository from the Internet.

Copying a Repository From the Internet

This section describes how to make a local copy of the Oracle Solaris 11 release package repository by copying the repository from an Internet location.

Create the Infrastructure for the Local Repository

Create the required pkg(5) repository infrastructure so that you can copy the repository. See the pkg(5) and pkgrepo(1) man pages.

```
# pkgrepo create /export/repoSolaris11
```

Copy the Repository

Use the pkgrecv(1) command to copy the repository. This operation could affect your network performance. The time required for this operation to complete depends on your network bandwidth and connection speed. To copy the Oracle Solaris 11 release repository, approximately 7 GB of data is transferred.

Tip – For better performance, close applications that use a large amount of memory, and make sure your zpool capacity is less than 80%.

Use the `zpool list` command to view your zpool capacity.

```
$ zpool list
NAME    SIZE  ALLOC  FREE  CAP  DEDUP  HEALTH  ALTROOT
rpool   186G  75.2G  111G  40%  1.00x  ONLINE  -

# pkgrecv -s http://pkg.oracle.com/solaris/release/ -d /export/repoSolaris11 '*'
Processing packages for publisher solaris ...
Creating Plan
Retrieving and evaluating 4288 package(s)...
PROCESS                     ITEMS       GET (MB)        SEND (MB)
developer/build/cmake   446/4288   332.1/4589.7   1000.2/14511.8
...
Completed              4288/4288  4589.7/4589.7  14511.8/14511.8
```

After the repository is copied, the process does some finish work. After you see the "Completed" line, wait another few minutes until you get your prompt back. If you update this repository later, only the changes are copied, and the process might take much less time.

If the `pkgrecv` operation is interrupted, use the `-c` option to retrieve content that was already downloaded and resume the content download. The value of *cache_dir* is supplied in an informational message when the transfer is interrupted, as shown in the following example:

```
PROCESS                     ITEMS       GET (MB)        SEND (MB)
...
pkgrecv: http protocol error: code: 503 reason: Service Unavailable
URL: 'http://pkg.oracle.som/solaris/release/file/file_hash

pkgrecv: Cached files were preserved in the following directory:
        /var/tmp/pkgrecv-fOGaIg
Use pkgrecv -c to resume the interrupted download.
# pkgrecv -c /var/tmp/pkgrecv-fOGaIg \
-s http://pkg.oracle.com/solaris/release/ -d /export/repoSolaris11 '*'
Processing packages for publisher solaris ...
Creating Plan
Retrieving and evaluating 156 package(s)...
PROCESS                     ITEMS       GET (MB)        SEND (MB)
desktop/compiz              1/156        0/395.0         0/1100.2
```

Copying a Repository From a File

This section describes how to make a local copy of the Oracle Solaris 11 release package repository from a repository file that is on media or is available on the Oracle Solaris 11 download site.

Get the Package Repository File

Download the Oracle Solaris 11 IPS package repository `.iso` files from the same location where you downloaded the system installation image, or locate the repository DVD in the media packet. The repository is in two files and is approximately 7 gigabytes total.

In addition to the repository `.iso` files, two other files are provided.

- Checksum file. Click the "MD5 checksum" link near the top of the Downloads page. Checksums are provided for the two repository files and for the concatenation of those two files. Compare the output from the following command to the appropriate value from the checksum file to confirm that your download was successful.

  ```
  $ digest -a md5 iso_file
  ```

- README file. The README file contains the information in this section, along with additional information such as how to copy the repository to USB or DVD media.

Copy the repository files to the file system you created in the last step. Concatenate the files into one file.

```
# cat sol-11-1111-repo-full.iso-a sol-11-1111-repo-full.iso-b > \
sol-11-1111-repo-full.iso
# ls /export/repoSolaris11
sol-11-1111-repo-full.iso
```

Make the Contents of the Repository File Available

Make the contents of the repository `.iso` file available.

```
# mount -F hsfs /export/repoSolaris11/sol-11-1111-repo-full.iso /mnt
# ls /mnt
COPYRIGHT  NOTICES    README     repo
```

If you receive an error message from the `mount` command, make sure you specified a full absolute path to the `.iso` file.

Check your work:

```
# df -k /mnt
Filesystem                                        1024-blocks     Used  Available Capacity  Mounted on
/export/repoSolaris11/sol-11-1111-repo-full.iso      6778178  6778178          0   100%    /mnt
```

You will need to remount the `.iso` image each time the repository server system restarts. To avoid the need to remount the `.iso` each time the system restarts, copy the repository files as described in the next section.

Copy the Repository Files

To increase the performance of repository accesses and to avoid the need to remount the `.iso` image each time the system restarts, copy the repository files from `/mnt/repo/` to a ZFS file system. You can do this copy with `rsync` or with `tar`.

- If you use the `rsync` command, be sure to specify `/mnt/repo/` (including the trailing slash character) and not `/mnt/repo` to copy the files and subdirectories in the `repo` directory. See the `rsync`(1) man page.

  ```
  # rsync -aP /mnt/repo/ /export/repoSolaris11
  ```

- Using the `tar` command as shown in the following example can be a faster way to move the repository from the mounted file system to the repository ZFS file system.

  ```
  # cd /mnt/repo; tar cf - . | (cd /export/repoSolaris11; tar xfp -)
  # cd /export/repoSolaris11
  ```

Check your work:

```
# ls /export/repoSolaris11
pkg5.repository            README
publisher                  sol-11-1111-repo-full.iso
# df -k /export/repoSolaris11
Filesystem                  1024-blocks      Used  Available Capacity  Mounted on
rpool/export/repoSolaris11    191987712  13733450   75787939    16%    /export/repoSolaris11
```

Unmount the Image

Unmount the image.

```
# umount /mnt
```

Build a Search Index

The repository creation commands do not build a search index by default. To enable clients to search for packages in the local repository, use the following command to catalog packages in the repository and update search indexes.

```
# pkgrepo -s /export/repoSolaris11 refresh
Initiating repository refresh.
```

◆ ◆ ◆ CHAPTER 3

Providing Access To Your Repository

This chapter describes how to enable clients to retrieve packages in your local repository by using a file interface or by using an HTTP interface. One repository can be set up for both types of access.

Retrieving Packages Using a File Interface

This section describes how to serve the local repository packages from a directory on your local network.

Configure an NFS Share

To enable clients to access the local repository via NFS, set the `sharenfs` property to create and publish the share.

```
# zfs create -o mountpoint=/export/repoSolaris11 rpool/repoSolaris11
# zfs set share=name=s11repo,path=/export/repoSolaris11,prot=nfs rpool/repoSolaris11
name=s11repo,path=/export/repoSolaris11,prot=nfs
# zfs set sharenfs=on rpool/repoSolaris11
```

Use one of the following tests to confirm that the share is published:

- Search for the repository in the shared file system table.

  ```
  # grep repo /etc/dfs/sharetab
  /export/repoSolaris11   s11repo nfs     sec=sys,rw
  ```

- Check whether the repository is accessible from a remote system.

  ```
  # dfshares solaris
  RESOURCE                          SERVER ACCESS    TRANSPORT
  solaris:/export/repoSolaris11    solaris -         -
  ```

Set the Publisher Origin To the File Repository URI

To enable client systems to get packages from your local file repository, you need to reset the origin for the `solaris` publisher. Execute the following command on each client:

```
# pkg set-publisher -G '*' -M '*' -g /net/host1/export/repoSolaris11/ solaris
```

`-G '*'` Removes all existing origins for the `solaris` publisher.

`-M '*'` Removes all existing mirrors for the `solaris` publisher.

`-g` Adds the URI of the newly-created local repository as the new origin for the `solaris` publisher.

Retrieving Packages Using an HTTP Interface

This section describes how to serve the local repository packages using the package depot server.

See "Serving Multiple Repositories Using Multiple Depot Server Instances" on page 24 for information about serving multiple repositories using multiple `pkg.depotd` daemons running on different ports. See "Multiple Repositories Under One Domain" on page 29 for information about running multiple repositories under one domain name with different prefixes.

Configure the Repository Server Service

To enable clients to access the local repository via HTTP, enable the `application/pkg/server` Service Management Facility (SMF) service.

```
# svccfg -s application/pkg/server setprop pkg/inst_root=/export/repoSolaris11
# svccfg -s application/pkg/server setprop pkg/readonly=true
```

Check your work:

```
# svcprop -p pkg/inst_root application/pkg/server
/export/repoSolaris11
```

Use `pkg.depotd` to serve the repository to clients. By default, `pkg.depotd` listens for connections on port 80. You can change the port by resetting the `pkg/port` property.

```
# svccfg -s application/pkg/server setprop pkg/port=port_number
```

For a complete list of `application/pkg/server` properties, see the `pkg.depotd(1m)` man page.

To set multiple service properties, use the following command to open a `vi` session where you can edit all the properties at once:

```
# svccfg -s pkg/server editprop
```

Remember to remove the comment marker (#) from the beginning of any lines you change.

Start the Repository Service

Restart the `pkg.depotd` repository service.

```
# svcadm refresh application/pkg/server
# svcadm enable application/pkg/server
```

To check whether the repository server is working, open a browser window on the `localhost` location. By default, `pkg.depotd` listens for connections on port 80. If you have changed the port, open a browser window on the `localhost:`*port_number* location.

Set the Publisher Origin To the HTTP Repository URI

To enable client systems to get packages from your local `pkg.depotd` repository, you need to reset the origin for the `solaris` publisher. Execute the following command on each client:

```
# pkg set-publisher -G '*' -M '*' -g http://localhost:port_number/ solaris
```

`-G '*'` Removes all existing origins for the `solaris` publisher.

`-M '*'` Removes all existing mirrors for the `solaris` publisher.

`-g` Adds the URI of the newly-created local repository as the new origin for the `solaris` publisher.

◆ ◆ ◆ CHAPTER 4

Maintaining Your Local IPS Package Repository

This chapter describes how to update packages in an IPS repository, how to set or update properties of a repository, and how to add packages to a repository from a second source.

This chapter also discusses Apache configuration of the depot server, including caching and load balancing.

Updating Your Local Repository

Before you transfer newer packages into your local repository, make sure your repository server is running the same or a newer version of the Oracle Solaris 11 OS as the version for which the packages you plan to copy are built. For example, if the server is running Oracle Solaris 11 and you want to update your repository to the Oracle Solaris 11 Update 1 repository, update the server to Oracle Solaris 11 Update 1 before you update your repository.

Whether you used the `pkgrecv` command or `.iso` files to create your local IPS package repository, use the `pkgrecv(1)` command to update the repository. Only packages that have changed are updated. See the performance tips in "Copy the Repository" on page 13.

```
# pkgrecv -s http://pkg.oracle.com/solaris/release/ -d /export/repoSolaris11 '*'
```

If you are going to be doing this update on a regular basis, you might want to use the `PKG_SRC` and `PKG_DEST` environment variables.

```
# export PKG_SRC=http://pkg.oracle.com/solaris/release/
# export PKG_DEST=/export/repoSolaris11
# pkgrecv '*'
```

After you have updated your repository, run the following command to catalog any new packages found in the repository and update all search indexes.

```
# pkgrepo -s /export/repoSolaris11 refresh
Initiating repository refresh.
```

Checking and Setting Repository Properties

This section describes how to display information about an IPS repository and how to set repository and publisher properties. See the pkgrepo(1) man page.

The following command displays a list of the package publishers known by the local repository. The STATUS column can tell you whether the publisher's package data is currently being processed.

```
$ pkgrepo info -s /export/repoSolaris11
PUBLISHER PACKAGES STATUS           UPDATED
solaris   4292     online           2011-10-26T17:17:30.230911Z
```

The following command displays property information about the local repository.

```
$ pkgrepo get -s /export/repoSolaris11
SECTION    PROPERTY    VALUE
publisher  prefix      solaris
repository description This\ repository\ serves\ a\ copy\ of\ the\ Oracle\ Solaris\ 11\
Build\ 175b\ Package\ Repository.
repository name        Oracle\ Solaris\ 11\ Build\ 175b\ Package\ Repository
repository version     4
```

The value of the publisher prefix specifies that `solaris` is to be used in the following cases:

- When more than one publisher's packages are present and no publisher is specified in the package name in the `pkg` command
- When packages are published to the repository and no publisher is specified

Version 4 repositories are created by default. Version 4 repositories support storage of packages for multiple publishers.

Use the `set` subcommand to specify new property values.

```
# pkgrepo set -s /export/repoSolaris11 \
repository/description="Local copy of the Oracle Solaris 11 repository" \
repository/name="Oracle Solaris 11 Package Repository"
# pkgrepo get -s /export/repoSolaris11
SECTION    PROPERTY    VALUE
publisher  prefix      solaris
repository description Local\ copy\ of\ the\ Oracle\ Solaris\ 11\ repository
repository name        Oracle\ Solaris\ 11\ Package\ Repository
repository version     4
```

The following command displays property information about the `solaris` publisher in the local repository. The parentheses indicate that the particular value can be a list of values.

```
$ pkgrepo get -p solaris -s /export/repoSolaris11
PUBLISHER SECTION    PROPERTY         VALUE
solaris   publisher  alias
solaris   publisher  prefix           solaris
```

```
solaris   repository collection-type  core
solaris   repository description      ""
solaris   repository legal-uris       ()
solaris   repository mirrors          ()
solaris   repository name             ""
solaris   repository origins          ()
solaris   repository refresh-seconds  ""
solaris   repository registration-uri ""
solaris   repository related-uris     ()
```

`collection-type` The `core` collection type indicates that the repository contains all of the dependencies declared by packages in the repository.

`legal-uris` The `legal-uris` is a list of locations for documents that provide legal information about the repository.

`origins` The `origins` is a list of locations of repositories that contain a complete copy of this repository's package metadata and content.

`related-uris` The `related-uris` is a list of locations of repositories that contain packages that users might be interested in.

See the `pkgrepo(1)` man page for descriptions of other publisher and repository properties.

The following command displays information about the specified *section/property* in the `pkg.oracle.com` repository.

```
$ pkgrepo get -p solaris -s http://pkg.oracle.com/solaris/release \
repository/name repository/description
PUBLISHER SECTION    PROPERTY         VALUE
solaris   repository description      This\ repository\ serves\ the\ Oracle\ Solaris\ 11\ Package\
repository.
solaris   repository name             Oracle\ Solaris\ 11\ Package\ Repository
```

Notice that the repository description and repository name property values are not set for the `solaris` publisher in the local repository. To provide values for publisher properties, use the `set` subcommand as shown above, specifying the publisher name as well. The publisher `repository/name` value is displayed on the browser interface near the top of the page and as the page title. The publisher `repository/description` value is displayed on the browser interface in the About section just below the name.

```
# pkgrepo set -p solaris -s /export/repoSolaris11 \
repository/description="Local copy of the Oracle Solaris 11 repository" \
repository/name="Oracle Solaris 11 Package Repository"
# pkgrepo get -p solaris -s /export/repoSolaris11
PUBLISHER SECTION    PROPERTY         VALUE
solaris   publisher  alias
solaris   publisher  prefix           solaris
solaris   repository collection-type  core
solaris   repository description      Local\ copy\ of\ the\ Oracle\ Solaris\ 11\ repository
solaris   repository legal-uris       ()
solaris   repository mirrors          ()
solaris   repository name             Oracle\ Solaris\ 11\ Package\ Repository
```

```
solaris   repository origins          ()
solaris   repository refresh-seconds  ""
solaris   repository registration-uri ""
solaris   repository related-uris     ()
```

Customizing Your Local Repository

You can create a repository that is a subset of the source repository. The following command copies all versions of the `group/feature/amp` package and all dependencies of those versions to the `amprepo` repository. The `amprepo` repository was previously created using the `pkgrepo create` command.

```
# pkgrecv -s http://pkg.oracle.com/solaris/release/ -d /export/amprepo \
-m all-versions -r group/feature/amp
```

You can add packages from different publishers to your repository. The following `pkgrecv` command adds all the packages from the `ISVproducts.p5p` package archive to the local repository. In the `pkg list` output, the publisher is shown because it is not the publisher that is highest ranked in search order in this image.

```
# pkg list -g /tmp/ISVproducts.p5p
NAME (PUBLISHER)       VERSION      IFO
isvtool (isv.com)      1.0          ---
# pkgrecv -s /tmp/ISVproducts.p5p -d /export/repoSolaris11 '*'
Processing packages for publisher isv.com ...
Retrieving and evaluating 1 package(s)...
PROCESS        ITEMS      GET (MB)      SEND (MB)
Completed        1/1       0.0/0.0        0.0/0
# pkg list -g /export/repoSolaris11 isvtool
NAME (PUBLISHER)       VERSION      IFO
isvtool (isv.com)      1.0          ---
```

Serving Multiple Repositories Using Multiple Depot Server Instances

This section shows how to extend the information provided in "Retrieving Packages Using an HTTP Interface" on page 18 to support serving multiple repositories using multiple `pkg.depotd` daemons running on different ports on the same repository server.

In this example, the `dev_repo` repository exists in addition to the `repoSolaris11` repository. The `repoSolaris11` repository is accessible from `http://localhost/` using port 80.

Make sure the publisher prefix is set on the `dev_repo` repository:

```
# pkgrepo set -s /export/dev_repo publisher/prefix=dev
```

Add a new instance of the `pkg/server` service:

```
# svccfg -s pkg/server add dev
# svccfg -s pkg/server:dev addpg pkg application
# svccfg -s pkg/server:dev setprop pkg/port=81
# svccfg -s pkg/server:dev setprop pkg/inst_root=/export/dev_repo
```

Check that you have added the new instance:

```
# svccfg -s pkg/server list
:properties
default
dev
```

Complete configuration of the new `pkg/server` instance:

```
# svccfg -s pkg/server:dev addpg general framework
# svccfg -s pkg/server:dev addpropvalue general/complete astring: dev
# svccfg -s pkg/server:dev addpropvalue general/enabled boolean: true
```

Start the new service:

```
# svcadm refresh application/pkg/server:dev
# svcadm enable application/pkg/server:dev
```

Browse the repository at `http://localhost:81/`.

See "Multiple Repositories Under One Domain" on page 29 for information about running multiple repositories under one domain name with different prefixes.

Depot Server Apache Configuration

This section discusses running the depot server behind an Apache web server instance to gain the following benefits:

- Improve performance by content caching and load balancing.
- Allow hosting multiple repositories under one domain name.

Configuring Caching For the Depot Server

Minimal configuration is required to set up the depot server behind a caching proxy. With the exception of the catalog attributes file and repository search results, which are discussed below, all files served are unique and therefore safe to cache indefinitely if necessary. Also, all depot responses contain the appropriate HTTP headers to ensure files in the cache do not become stale by mistake.

See the Apache Caching Guide for more information about configuring Apache as a caching proxy.

Use the `CacheRoot` directive to specify the directory to contain the cached files. Make sure the specified directory is writable by the Apache process. No explicit error message is output if Apache cannot write to this directory.

```
CacheRoot /tank/proxycache
```

Apache allows you to enable caching for specific directories. You probably want your repository server to cache all the content on the server, as shown in the following directive.

```
CacheEnable disk /
```

Use the `CacheMaxFileSize` directive to set the maximum size of files to be cached. The Apache default of 1 MB might be too small for most repositories. The following directive sets the maximum cached file size to 1 GB.

```
CacheMaxFileSize 1000000000
```

Adjust the directory structure of the on-disk cache for the best performance with the underlying file system. In a ZFS dataset, multiple directory levels affect performance more than the number of files in one directory. Therefore, configure one directory level with a large number of files in each directory. Use the `CacheDirLevels` and `CacheDirLength` directives to control the directory structure. Set `CacheDirLevels` to 1. Set `CacheDirLength` to a value that results in a good balance between the number of directories and the number of files per directory. The value of 2 set below will generate 4096 directories. See the Apache Disk-based Caching documentation for more information.

```
CacheDirLevels 1
CacheDirLength 2
```

Cache Considerations For the Catalog Attributes File

The repository catalog attributes file (`catalog.attrs`) contains the current status of the repository catalog. This file can be large enough to warrant caching. However, this file becomes stale if the catalog of the back-end repository has changed. You can use one of the following two methods to address this issue.

- Do not cache this file. This solution works best if the repository server runs in a high-bandwidth environment where the additional traffic is not an important consideration. The following partial `httpd.conf` file shows how to specify not to cache the `catalog.attrs` file:

  ```
  <LocationMatch ".*/catalog.attrs">
          Header set Cache-Control no-cache
  </LocationMatch>
  ```

- Prune this file from the cache whenever the catalog of the back-end repository is updated.

Cache Considerations For Search

Searching a package repository generates custom responses based on the request. Therefore, search results are not well suited for being cached. The depot server sets the appropriate HTTP headers to make sure search results do not become stale in a cache. However, the expected bandwidth savings from caching are small. The following partial `httpd.conf` file shows how to specify not to cache search results.

```
<LocationMatch ".*/search/\d/.*">
        Header set Cache-Control no-cache
</LocationMatch>
```

Running the Depot Server Behind a Web Proxy

The `pkg`(5) depot server enables you to easily provide access to a repository in the local network or on the Internet. However, the depot server does not support serving multiple repositories under one domain name or sophisticated prefixes. To host multiple repositories under one domain name, run the depot server behind a web proxy. Running the depot server behind a web proxy can also improve the performance of the server by enabling load-balancing over multiple depots and enabling content caching.

The examples in this section use the Apache web server as the proxy software. The Oracle Solaris 11 OS includes the Apache web server and a basic `httpd.conf` file. You should be able to apply the principles shown in these examples to any proxy server software.

Recommended Generic Apache Configuration Settings

The following settings affect performance and security.

Enable the Apache DEFLATE filter.

HTTP clients can tell the server that they accept compressed data in an HTTP request. Enabling the Apache DEFLATE filter can dramatically reduce the over-the-wire size of metadata such as catalogs and manifests. Metadata such as catalogs and manifests often compress 90%.

```
AddOutputFilterByType DEFLATE text/html application/javascript text/css text/plain
```

Do not decode encoded forward slashes.

Packages can contain URL encoded forward slashes. To make sure these forward slashes are not interpreted as directory delimiters, instruct Apache to not decode them.

```
AllowEncodedSlashes NoDecode
```

Note – Omitting this setting very negatively impacts search functionality.

Allow more pipelined requests.

Increase the `MaxKeepAliveRequests` value to allow clients to make a larger number of pipelined requests without closing the connection. The Apache default of 100 is too low.

```
MaxKeepAliveRequests 10000
```

Set the maximum wait time for response.

The proxy timeout sets how long Apache waits for the back-end depot to respond. For most operations, 30 seconds is satisfactory. Searches with a very large number of results can take significantly longer. You might want a higher timeout value to accommodate such searches.

```
ProxyTimeout 30
```

Disable forward proxying.

Make sure that forward proxying is disabled.

```
ProxyRequests Off
```

Apache Configuration Examples

This section illustrates multiple repository, non-load-balanced, and load-balanced setups.

A Simple Prefixed Proxy Configuration

This example shows the basic configuration for a non-load-balanced depot server. This example connects `http://pkg.example.com/myrepo` to `internal.example.com:10000`.

See "Serving Multiple Repositories Using Multiple Depot Server Instances" on page 24 for instructions about setting other properties you need that are not described in this example.

You should configure the depot server with a `pkg/proxy_base` setting that names the URL at which the depot server can be accessed. Use the following commands to set the `pkg/proxy_base`:

```
# svccfg -s pkg/server add repo
# svccfg -s pkg/server:repo addpg pkg application
# svccfg -s pkg/server:repo "setprop pkg/proxy_base = astring: http://pkg.example.com/myrepo"
# svcadm refresh pkg/server:repo
# svcadm enable pkg/server:repo
```

The `pkg`(5) client opens 20 parallel connections to the depot server when performing network operations. Make sure the number of depot threads matches the expected connections to the server at any given time. Use the following commands to set the number of threads per depot:

```
# svccfg -s pkg/server:repo "setprop pkg/threads = 200"
# svcadm refresh pkg/server:repo
# svcadm restart pkg/server:repo
```

Use `nocanon` to suppress canonicalization of URLs. This setting is important for properly working search. Also, limit the number of back-end connections to the number of threads the depot server provides. The following partial `httpd.conf` file shows how to proxy one depot server:

```
Redirect /myrepo http://pkg.example.com/myrepo/
ProxyPass /myrepo/ http://internal.example.com:10000 nocanon max=200
```

Multiple Repositories Under One Domain

The most important reason to run the depot server behind a proxy is to easily run several repositories under one domain name with different prefixes. The example from "A Simple Prefixed Proxy Configuration" on page 28 can be easily extended to support multiple repositories.

In this example, three different prefixes of one domain name are connected to three different package repositories:

- `http://pkg.example.com/repo_one` is connected to `internal.example.com:10000`
- `http://pkg.example.com/repo_two` is connected to `internal.example.com:20000`
- `http://pkg.example.com/xyz/repo_three` is connected to `internal.example.com:30000`

The `pkg(5)` depot server is an SMF managed service. Therefore, to run multiple depot servers on the same host, simply create a new service instance:

```
# svccfg -s pkg/server add repo1
# svccfg -s pkg/server:repo1 addpg pkg application
# svccfg -s pkg/server:repo1 setprop pkg/property=value
# ...
```

Like the previous example, each depot server runs with 200 threads.

```
Redirect /repo_one http://pkg.example.com/repo_one/
ProxyPass /repo_one/ http://internal.example.com:10000 nocanon max=200

Redirect /repo_two http://pkg.example.com/repo_two/
ProxyPass /repo_two/ http://internal.example.com:20000 nocanon max=200

Redirect /xyz/repo_three http://pkg.example.com/xyz/repo_three/
ProxyPass /xyz/repo_three/ http://internal.example.com:30000 nocanon max=200
```

Load Balanced Configurations

You might want to run depot servers behind an Apache load balancer. This example connects `http://pkg.example.com/myrepo` to `internal1.example.com:10000` and `internal2.example.com:10000`.

Configure the depot server with an appropriate `proxy_base` setting as shown in "A Simple Prefixed Proxy Configuration" on page 28.

Limit the number of back-end connections to the number of threads each depot is running divided by the number of depots in the load-balancer setup. Otherwise, Apache opens more connections to a depot than are available and they stall, which can decrease performance. Specify the maximum number of parallel connections to each depot with the `max=` parameter. The example below shows two depots, each running 200 threads. See "A Simple Prefixed Proxy Configuration" on page 28 for an example of how to set the number of depot threads.

```
<Proxy balancer://pkg-example-com-myrepo>
        # depot on internal1
        BalancerMember http://internal1.example.com:10000 retry=5 max=100

        # depot on internal2
        BalancerMember http://internal2.example.com:10000 retry=5 max=100
</Proxy>

Redirect /myrepo http://pkg.example.com/myrepo/
ProxyPass /myrepo/ balancer://pkg-example-com-myrepo nocanon
```

Complete Load Balanced Example

The following example includes all the directives you need to add to the `httpd.conf` file for a repository server hosting a load-balanced and a non-load-balanced depot server setup.

In this example, two different prefixes of one domain name are connected to three different package repositories:

- `http://pkg.example.com/repo_one` is connected to `internal1.example.com:10000` and `internal2.example.com:10000`
- `http://pkg.example.com/repo_two` is connected to `internal1.example.com:20000`

```
AddOutputFilterByType DEFLATE text/html application/javascript text/css text/plain

AllowEncodedSlashes NoDecode

MaxKeepAliveRequests 10000

ProxyTimeout 30

ProxyRequests Off

<Proxy balancer://pkg-example-com-repo_one>
        # depot on internal1
        BalancerMember http://internal1.example.com:10000 retry=5 max=100

        # depot on internal2
        BalancerMember http://internal2.example.com:10000 retry=5 max=100
</Proxy>

Redirect /repo_one http://pkg.example.com/repo_one/
```

```
ProxyPass /repo_one/ balancer://pkg-example-com-repo_one nocanon
Redirect /repo_two http://pkg.example.com/repo_two/
ProxyPass /repo_two/ http://internal.example.com:20000 nocanon max=200
```