RStudio Package Manager: Admin Guide

Version 0.6.0-5

Abstract

This guide will help an administrator install and configure RStudio Package Manager on a managed server. You will learn how to install the product on different operating systems, configure authentication, and monitor system resources.

Contents

1 Introduction

RStudio Package Manager organizes and centralizes R packages across your team, department, or entire organization.

Traditionally, R packages entered the organization from a variety of sources including CRAN, Bioconductor, Github, and even internally developed package sources. RStudio Package Manager empowers R users to access packages and reproduce environments while giving IT control and visibility into package use.

2 Getting Started with Installation

This chapter helps you install RStudio Package Manager on Ubuntu (Section 2.2.1) or Red Hat Enterprise Linux (Section 2.2.2)/CentOS Linux (Section 2.2.2)/ SUSE Linux Enterprise (Section 2.2.2), learn to manage the server (Section 3.1), and perform some initial configuration (Section 2.3).

We built this checklist to guide you through the initial installation process.

1. Check the OS, Privileges, and Internet Access Requirements
2. Download RStudio Package Manager Installer
3. Install RStudio Package Manager - Ubuntu 2.2.1, Red Hat/CentOS/SUSE 2.2.2,
4. Initial Configuration - 2.3

5. Activate RStudio Package Manager License - 4
6. Start RStudio Package Manager - 3.1. At this point, RStudio Package Manager will be available at your specified URL.
7. Add Repositories and Packages - 6. After this step, users will be able to install packages and repositories will be available in the web client.

Once RStudio Package Manager is installed, licensed, and started, visit the quick start configuration guides to quickly configure RStudio Package Manager for common scenarios.

2.1 Requirements

Please review the necessary system requirements, account privileges, and internet access requirements prior to installing RStudio Package Manager.

2.1.1 System Requirements

- Red Hat Enterprise Linux/CentOS Linux 6.0+
- Red Hat Enterprise Linux/CentOS Linux 7.0+
- SUSE Enterprise Linux 12 SP3+
- Ubuntu 14.04
- Ubuntu 16.04
- Ubuntu 18.04

RStudio Package Manager should run on a server with a minimum of 2GB of RAM. Package sources can be lazily cached from CRAN or eagerly downloaded. RStudio Package Manager recommends 50-200GB of disk storage.

2.1.2 Internet Access Requirements

RStudio Package Manager acts as a "bridge" between offline servers running R and upstream package sources like CRAN. RStudio Package Manager should be installed on a server in the network with outbound access to:

`https://rspm-sync.rstudio.com`

See Repository Syncing for more details on what information is downloaded from the RStudio CRAN service.

RStudio Package Manager can be configured to use a HTTP proxy to access the internet, see Outbound Proxy for more details.

Additional steps are necessary if your internal servers can not access an online server through your internal network. See the appendix on running RStudio Package Manager in an air-gapped environment.

2.1.3 Root Requirements

RStudio Package Manager does not run as root, see 2.1.3.1. By default, root privileges are required to:

- Install RStudio Package Manager
- Start and Stop RStudio Package Manager via the service daemons
- Activate the RStudio Package Manager license

RStudio Package Manager can be installed and licensed without requiring root privileges. Instructions are available in the appendix.

2.1.3.1 RunAs User

RStudio Package Manager runs under an unprivileged account.

The installer creates a user account and group named `rstudio-pm` and runs the RStudio Package Manager service under this account. If you wish to change the account under which the service runs, please see C.

2.2 Installation

Please follow the installation instructions specific to your operating system.

2.2.1 Install RStudio Package Manager - Ubuntu

You will use `gdebi` to install RStudio Package Manager and its dependencies. It is installed via the `gdebi-core` package.

```
sudo apt-get install gdebi-core
```

You should have been provided with a `.deb` installer for RStudio Package Manager. If you only have a link to this file, you can use `wget` to download the file to the current directory.

```
wget https://download-url/rstudio-pm_0.6.0-5.deb
```

Once the `.deb` file is available locally, run the following command to install RStudio Package Manager.

```
sudo gdebi rstudio-pm_0.6.0-5.deb
```

This will install RStudio Package Manager into `/opt/rstudio-pm`

2.2.2 Install RStudio Package Manager - RedHat/SUSE

You should have been provided with an RPM file which contains RStudio Package Manager. You can install this rpm file using `yum`/`zypper`. If you have only a link to the RPM file, you can use `wget` to download the file to the current directory.

For RedHat:

```
sudo yum install --nogpgcheck rstudio-pm-0.6.0-5.x86_64.rpm
```

For SUSE:

```
sudo zypper --no-gpg-checks install rstudio-pm-0.6.0-5.x86_64.rpm
```

This will install RStudio Package Manager into `/opt/rstudio-pm/`.

2.3 Initial Configuration

Before RStudio Package Manager can run, you will need to update the configuration file located at `/etc/rstudio-pm/rstudio-pm.gcfg`. Complete the `Address` property within the `Server` section by specifying the URL used to access RStudio Package Manager by clients. For example:

```
Address = http://r-packages.example.com
```

The other configuration properties may also be set, see: A for details.

After updating the configuration file, follow the steps in 4 to activate the RStudio Package Manager license and then restart the server 3.1.

At this point, the web client is available at the server address you selected. The default address is `http://<server_ip>:4242`. Though the client is available, no repositories are available yet. The next step is to configure sources and repositories using the admin CLI 6. We recommend visiting the quickstart configuration guides.

3 Server Management

This section describes common administrative tasks for RStudio Package Manager.

3.1 Stopping and Starting

Occasionally it is necessary to start and stop the RStudio Package Manager service. Stopping and starting is handled by `systemd` or Upstart.

The specific stop/start commands depend on the service daemon. Commands for `systemd` and Upstart are listed below.

After a restart, any scheduled syncs that were missed during downtime will automatically begin. See 7.3 for details.

3.1.1 systemd

> Distributions using `systemd` include Red Hat/CentOS 7, SUSE 12, Ubuntu 16.04, and Ubuntu 18.04.

`systemd` is a management and configuration platform for Linux. The newest versions of most major Linux distributions have adopted `systemd` as their default init system.

The RStudio Package Manager installer installs a `systemd` service called `rstudio-pm`, which causes the RStudio Package Manager to be started and stopped automatically when the machine boots up and shuts down. The `rstudio-pm` service is also automatically launched during installation.

Use the following commands to manually start and stop the server:

```
sudo systemctl start rstudio-pm
```

```
sudo systemctl stop rstudio-pm
```

You can restart the server with:

```
sudo systemctl restart rstudio-pm
```

If you wish to keep the server running without interruption, but reload the configuration, you can use the systemctl command to send a `HUP` signal:

```
sudo systemctl kill -s HUP --kill-who=main rstudio-pm
```

The `HUP` signal causes the server to re-initialize but does not interrupt the current processes or any of the open connections to the server.

> Use a `HUP` signal when your configuration changes are limited to properties marked as reloadable. See Appendix A to learn which settings may be reloaded via `HUP`. Perform a full restart of RStudio Package Manager when changing other properties.

You can check the status of the `rstudio-pm` service using:

```
sudo systemctl status rstudio-pm
```

And finally, you can use the `enable/disable` commands to control whether RStudio Package Manager should be run automatically at boot time:

```
sudo systemctl enable rstudio-pm
```

```
sudo systemctl disable rstudio-pm
```

3.1.2 Upstart (Ubuntu 14.04, Red Hat 6)

Distributions using Upstart include Red Hat/CentOS 6 and Ubuntu 14.04.

Upstart is a system used to automatically start, stop and manage services. The installer writes an Upstart configuration file to `/etc/init/rstudio-pm.conf`. This instructs the Upstart to initialize RStudio Package Manager as soon as the network is activated on the machine and stop when the machine is being shut down.

The Upstart configuration also ensures that the `rstudio-pm` process is respawned if the process unexpectedly terminates. However, in the event that there is an issue which consistently prevents RStudio Package Manager from being able to start (such as a bad configuration file), Upstart will give up on restarting the service after approximately 5 failed attempts within a few seconds. For this reason, you may see multiple repetitions of a bad RStudio Package Manager startup attempt before it transitions to the "stopped" state.

To start or stop the server, run the following commands, respectively.

```
sudo start rstudio-pm
```

```
sudo stop rstudio-pm
```

To restart the server you can run:

```
sudo stop rstudio-pm
sudo start rstudio-pm
```

The `restart` command re-initializes the server.

> We recommend `stop` and `start` over `restart` because some configuration changes are not incorporated into a restart. In particular, `restart` *does not* `re-read the Upstart definition a`t/etc/init/rstudio-pm.conf`. Changes to this file need a`stop`and`start‘ to take effect.

If you wish to reload the configuration and keep the server and all R processes running without interruption, you can use the `reload` command:

```
sudo reload rstudio-pm
```

This command causes the server to re-initialize but does not interrupt the current processes or any of the open connections to the server.

> Use a `HUP` signal when your configuration changes are limited to properties marked as reloadable. See Appendix A to learn which settings may be reloaded via `HUP`. Perform a full restart of RStudio Package Manager when changing other properties.

To check the status or retrieve the process ID associated with `rstudio-pm`, run the following:

```
sudo status rstudio-pm
```

3.2 Upgrading

Upgrading RStudio Package Manager requires limited downtime. During an upgrade users will not be able to install packages. We recommend upgrading during a period of downtime.

The latest version is available on the download page along with release notes. The current version is available by running:

```
cat /opt/rstudio-pm/VERSION
```

To upgrade:

1. Download the latest `.rpm` or `.deb` file
2. Run the install command:

   Ubuntu:

   ```
   sudo gdebi <rstudio-pm-version.deb>
   ```

   Red Hat/CentOS:

   ```
   sudo yum install --nogpgcheck <rstudio-pm-version.rpm>
   ```

   SUSE:

   ```
   sudo zypper --no-gpg-checks install <rstudio-pm-version.rpm>
   ```

The new version of RStudio Package Manager will install on top of an earlier installation. Existing configuration settings are respected. During installation the RStudio Package Manager service is restarted.

3.3 Purging RStudio Package Manager

You can fully remove RStudio Package Manager and all its data from your server using the following steps:

1. Stop the RStudio Package Manager service. (See 3.1 for details)
2. Uninstall the RStudio Package Manager package from your system.

   Ubuntu:

   ```
   sudo apt-get purge rstudio-pm
   ```

   Red Hat/CentOS:

   ```
   sudo yum remove rstudio-pm
   ```

   SUSE:

   ```
   sudo zypper remove rstudio-pm
   ```

3. Remove `/opt/rstudio-pm` if it still exists.
4. Remove logs from `/var/log/rstudio-pm*`
5. Purge the databases
   - When using SQLite, remove the `SQLite.Dir` directory. This has a default location of `/var/lib/rstudio-pm/db`.
   - When using PostgreSQL, drop the databases used by RStudio Package Manager. You may also wish to remove the PostgreSQL user associated with RStudio Package Manager.
6. Remove the `Server.DataDir` directory. By default, this is `/var/lib/rstudio-pm`.
7. Remove configuration files from `/etc/rstudio-pm` if they still exist.

3.4 Backups

We recommend including the RStudio Package Manager configuration file in `/etc/rstudio-pm` as well as the variable data directory which defaults to `/var/lib/rstudio-pm` in your system backups. If you have configured the databases to be stored outside the data directory, ensure that it is also included in the backup.

A running RStudio Package Manager server may be writing into the data directory. You should stop the RStudio Package Manager server before taking a backup.

```
sudo stop rstudio-pm
# Run appropriate backup steps here.
sudo start rstudio-pm
```

Your platform may need alternate commands to restart RStudio Package Manager. Please see Section 3.1 for instructions specific to your operating system version.

3.5 Privileged Ports

RStudio Package Manager listens on HTTP port 4242 by default. When you modify the `HTTP.Listen` or `HTTPS.Listen` configuration properties to use a privileged port under 1024, the service will fail to start, and you will see an error like the following in '`/var/log/rstudio-pm.log`:

```
2017/11/28 13:41:59 Error: Could not initialize the HTTP listener: listen tcp :80: bind: permission deni
```

If you wish to listen with HTTP or HTTPS on a privileged port (< 1024), you can grant the RStudio Package Manager binary permission to do so by issuing the following command as root:

```
sudo setcap 'cap_net_bind_service=+ep' /opt/rstudio-pm/bin/rstudio-pm
```

After issuing the above command, restart the service (see 3.1).

4 Licensing

When RStudio Package Manager is first installed on a system it operates in an evaluation mode for a period of time and then subsequently requires activation for continued use.

To determine the current license status of your system, you can use the following command:

```
sudo /opt/rstudio-pm/bin/license-manager status
```

After purchasing a license to RStudio Package Manager, you will receive a product key that is used to activate the license on a given system.

You can activate your license key with the command:

```
sudo opt/rstudio-pm/bin/license-manager activate <product-key>
```

After activation, restart the RStudio Package Manager server.

```
sudo stop rstudio-pm
sudo start rstudio-pm
```

Your platform may need alternate commands to restart RStudio Package Manager. Please see Section 3.1 for instructions specific to your operating system version.

If you want to move your license of RStudio Package Manager to another system, you should first deactivate it on the old system.

```
sudo /opt/rstudio-pm/bin/license-manager deactivate
```

4.1 Proxy Servers

If your server is behind an internet proxy, you may need to add an additional command line flag indicating the address and credentials required to communicate through the proxy. This may not be necessary if either the `http_proxy` or `all_proxy` environment variable is defined (these are read and used by the license manager when available).

If you do need to specify a proxy server explicitly you can do so using the `--proxy` command line parameter. For example:

```
sudo /opt/rstudio-pm/bin/license-manager \
    --proxy=http://127.0.0.1/ activate <product-key>
```

Proxy settings can include a host-name, port, and username/password if necessary. The following are all valid proxy configurations:

```
http://127.0.0.1/
http://127.0.0.1:8080/
http://user:pass@127.0.0.1:8080/
```

If the port is not specified, the license manager will default to using port `1080`.

4.2 Offline Activation

If your system has no connection to the internet it's also possible to perform an offline activation. To do this, we recommend using our offline activation app which will walk you through the process: RStudio Offline Activation

You first generate an offline activation request as follows:

```
sudo /opt/rstudio-pm/bin/license-manager activate-offline-request <product-key>
```

Executing this command will print an offline activation request to the terminal which you should copy and paste into an email to RStudio customer support (support@rstudio.com). You will receive a reply with a file attachment that can be used to activate offline as follows:

```
sudo /opt/rstudio-pm/bin/license-manager activate-offline <activation-file>
```

After activation, restart the RStudio Package Manager server. Please see Section 3.1 for instructions specific to your operating system version.

If you want to renew your license of RStudio Package Manager or move it to another system you can also perform license deactivation offline. You can do this as follows:

```
sudo /opt/rstudio-pm/bin/license-manager deactivate-offline
```

Executing this command will print an offline deactivation request to the terminal which you should send to support@rstudio.com.

You can also perform an offline check of your current license status using the following command:

```
sudo /opt/rstudio-pm/bin/license-manager status-offline
```

4.3 Licensing errors

RStudio Package Manager uses the `license-manager` to determine if a valid license is available. Should an error occur when interacting with the license manager, RStudio Package Manager indicates that problem in the `/var/log/rstudio-pm.log` log. The license manager sends details about the error to the system messages (syslog). You should consult both locations should RStudio Package Manager be unable to obtain license status.

5 Files and Directories

5.1 Changing Ownership

Many of the files and directories mentioned in this chapter are, by default, owned by the `rstudio-pm` user. If you change the RunAs user for the RStudio Package Manager service, you will need to change ownership of these files and directories. See C for details on changing the RStudio Package Manager service RunAs user.

5.2 Program Files

The RStudio Package Manager installers place all program files into the `/opt/rstudio-pm` directory.

You should not need to change any files in the `/opt/rstudio-pm` hierarchy. Any alterations will be overwritten by subsequent re-installs or upgrades of RStudio Package Manager.

5.3 Configuration

The RStudio Package Manager configuration file is `/etc/rstudio-pm/rstudio-pm.gcfg`. This file is initially owned by `rstudio-pm` with permissions `0640`. You will edit this file to properly configure RStudio Package Manager for your organization.

A configuration management tool like Puppet or Chef can be used to maintain the `rstudio-pm.gcfg` file. We recommend that it remain owned by `rstudio-pm` and have permissions `0640`, as your configuration may need to contain passwords and other sensitive information.

RStudio Package Manager upgrades will not overwrite customizations to the `rstudio-pm.gcfg` file.

5.4 Server Log

The RStudio Package Manager server log is located at `/var/log/rstudio-pm.log`. This file is owned by `rstudio-pm` with permissions `0600`.

If `logrotate` is available when RStudio Package Manager is installed, a `logrotate` configuration will be installed. The default configuration is to rotate the logfile daily. The old log file will be stored alongside the original with a numeric extension, `.1`, `.2`, etc. The rotated log files are compressed after one day. The `.1` log file is retained uncompressed, but older logs are compressed. Most systems use `gzip` for compression, giving log files with extensions like `.2.gz`, `.3.gz`. Logs will be maintained for 30 days.

The manual for `logrotate` has more information.

5.5 Access Logs

The RStudio Package Manager HTTP access logs are located at `/var/log/rstudio-pm.access.log`. This file is owned by `rstudio-pm` with permissions `0600`. Log files are stored in Apache Combined Log Format. See http://httpd.apache.org/docs/2.2/logs.html#combined for a description of this format.

If `logrotate` is available when RStudio Package Manager is installed, a `logrotate` configuration will be installed. The default configuration is to rotate the logfile daily. The old logfile will be compressed and stored alongside the original log file with a `.1.gz` extension (then `.2.gz`, etc.). Logs will maintained for 30 days.

5.6 Variable Data

RStudio Package Manager manages R packages and repositories. All package source bundles are stored in the server's data directory. The RStudio Package Manager handles incoming requests for packages across repositories. Only a single copy of each package source is stored, even if the package is referenced in multiple repositories.

The RStudio Package Manager data directory also contains information used by the server to manage repositories including the RStudio Package Manager `SQLite` databases and encryption key if SQLite is used.

The default location for the RStudio Package Manager data directory is `/var/lib/rstudio-pm`. This can be customized by specifying an alternate `DataDir` in the `Server` section of your configuration file.

```
[Server]
DataDir = /mnt/rstudio-pm
```

If you customize the RStudio Package Manager data directory, make sure that the `rstudio-pm` user has permission to read, write, and create directories in the data directory.

The RStudio Package Manager SQLite databases **must** exist on local storage. If the location for `DataDir` is not local storage but a networked location over NFS, configure the `Dir` setting in the `SQLite` section of your server configuration file.

```
[Server]
DataDir = /mnt/rstudio-pm

[SQLite]
Dir = /var/lib/rstudio-pm/db
```

5.6.1 Permissions

`/var/lib/rstudio-pm` is owned by `rstudio-pm` with permissions `0700`.

6 Admin CLI

RStudio Package Manager is administered through a command-line interface (CLI). Administrators can use the CLI to create repositories and sources, add local packages to sources, and setup sync schedules for CRAN sources.

The CLI is installed at `/opt/rstudio-pm/bin/rspm`. The CLI uses the configuration defined in `/etc/rstudio-pm/rstudio-pm.gcfg` unless you specify an alternate configuration file with the `--config` flag.

6.1 Privileges

Users must be a member of the `rstudio-pm` group in order to use the RStudio Package Manager CLI. See C for instructions on changing the required group.

The RStudio Package Manager CLI uses a Unix domain socket for communicating with the RStudio Package Manager server. By default, the domain socket file is located at `/var/run/rstudio-pm/rstudio-pm.sock`. You can customize the location by configuring the `Server.SockFileDir` setting.

Any user invoking the RStudio Package Manager CLI must have read/write access to the Unix domain socket in order to communicate with the RStudio Package Manager server.

6.2 Location

The CLI is located at `/opt/rstudio-pm/bin/rspm`. We recommend that users make an alias, add this location to their PATH, or navigate to this directory if they will be running multiple commands in one session, for example:

```
alias rspm='/opt/rstudio-pm/bin/rspm'
```

6.3 Getting Help

Simply run `rspm` without any arguments to display the top-level help for the RStudio Package Manager CLI. For help with a specific command, you can use the `help` command. For example, to display help about the `create` command, you can use `rspm help create`.

6.4 Example

Here, we provide an example of configuring a new RStudio Package Manager to serve CRAN packages and local packages. See 15 for additional examples and scenarios.

We will create a CRAN source and a local source. Then we will schedule and sync the CRAN source and add internal packages to the local source. Finally, we will create a repository that subscribes to both sources. For more information on sources and repositories, refer to 7.

Throughout the example we will use the alias `rspm`, see 6.2.

6.4.1 Step 1: Create a local source

```
rspm create source --name=companyPkgs
```

Add the internal package `parcel_0.1.tar.gz`:

```
rspm add --source=companyPkgs --path=/path/to/parcel_0.1.tar.gz
```

6.4.2 Step 2: Sync with CRAN

To force an immediate sync with CRAN:

```
rspm sync
```

6.4.3 Step 3: Create a repository

```
rspm create repo --name=companyRepo
```

Subscribe the repo to the local source and to CRAN:

```
rspm subscribe --repo=companyRepo --source=companyPkgs
rspm subscribe --repo=companyRepo --source=cran
```

6.4.4 Step 4: Verify that everything is correct:

```
rspm list sources
rspm list repos
rspm list sources --repo=companyRepo
rspm list packages --repo=companyRepo --search=parcel
```

Once setup, the repository will be available from the web UI and users will be able to install both CRAN and local packages.

6.4.5 Optional: Delete a local package

Normally, upgrading a package automatically archives the previous versions. If you ever want to completely remove a package, use the remove command:

```
rspm remove --source=companyPkgs --name=parcel@0.1 --comment="accidentally included DB credentials"
```

> Note: Removing a package makes the package invisible going forward. It does not delete historical transactions for the package.

7 Repositories and Sources

R repositories contain package source tar files and are the primary vehicle for organizing and distributing R packages. For more information on packages and repositories see B.

In RStudio Package Manager, repositories are created from one or more sources. The documentation in this chapter outlines repositories as well as the types and structure of sources.

7.1 Repository Structure

R package repositories have a specific structure that enables client commands like `install.packages` to query the repository's contents and download packages.

A regular CRAN repository is just a set of files served from disk. RStudio Package Manager *does not create repositories on disk*. Instead, the Package Manager maintains a single copy of each package source and uses a database and specialized web server to handle HTTP requests from R.

Some example requests that can be served by the RStudio Package Manager:

PACKAGES file

`http://pkg-manager.example.com/repo/latest/src/contrib/PACKAGES`

This serves a PACKAGES file. The PACKAGES file for a repository is human-readable and contains information on each package available in the repository. RStudio Package Manager can also serve requests for PACKAGES.gz and PACKAGES.rds.

Package Source

`http://pkg-manager.example.com/repo/latest/src/contrib/package_2.1.0.tar.gz`

This request downloads the package source to the client.

Archived Package Source

`http://pkg-manager.example.com/repo/latest/src/contrib/archive/package/package_1.1.0.tar.gz`

This request downloads the tar file for an older, archived version of the package.

Most importantly, a RStudio Package Manager repository **is a CRAN-like repository** which means users can access and install packages using their regular R functions: `install.packages`, `available.packages`, `packrat`, and `devtools::install`.

7.2 Sources

7.2.1 About Sources

RStudio Package Manager repositories are composed of one or more **sources**. There are currently three types of sources:

1. *cran* source - A single cran source is automatically created. This source contains metadata and packages from RStudio's CRAN service. The source can be used directly in a repository to give users access to all CRAN packages, or it can used indirectly by curated-cran sources.

   While the cran source is created automatically, an administrator must use the CLI before any metadata or packages are downloaded to RStudio Package Manager. See the CLI section for more information on making CRAN available through RStudio Package Manager.

2. *curated-cran* source - A curated CRAN source allows administrators to specify specific sets of approved CRAN packages. Administrators can add or remove packages from the set, and they can also update the set. See 7.4 for more information.

3. *local* source - A local source is used as a mechanism to distribute locally developed packages or other packages without native support in RStudio Package Manager. Administrators add packages to local sources by specifying a path to a package's tar file.

7.2.2 Repositories with Multiple Sources

A repository can have more than one source. If you wish to serve both local packages and CRAN packages from a single repository, you can create a single repository that subscribes to multiple sources. For example:

- public (a repository)
  - internal (local source)
  - cran (CRAN source)

The "public" repository above gives users access to both local and CRAN packages, and its PACKAGES list could be accessed, for example, at http://pkg-manager.example.com/public/latest/src/contrib/PACKAGES. A repository subscribes to sources, which means that changes to a source will be reflected in the repository. For example, if an admin adds a new package to the `internal` source, users will automatically be able to access the new package via the `public` repository.

7.2.3 Package Conflicts Between Sources

If a repository has multiple sources and a package with the same name exists in both sources, RStudio Package Manager eliminates duplicates, giving preference *in the order the sources are subscribed*. In the example repository above, if a package named "plumber" exists in both the "cran" and "internal" sources, the "plumber" package from the "internal" source would be served and listed since it is the first source for the repository. The same conflict resolution occurs as sources change. For example, in the sample above, even if a new package is added to CRAN with the same name as an internal package, the internal package will continue to be served. The precedence is also maintained during updates. In the example above, the internal version of plumber will continue to be served even if the CRAN version of plumber is updated. The order of sources within a repository can be re-arranged using the `reorder` command.

7.3 The CRAN Source

A primary use case for RStudio Package Manager is making packages in public repositories, like CRAN, available to enterprise users. Administrators can elect to make all of CRAN available, or to make only curated subsets of CRAN available.

7.3.1 What is RStudio's Package Service?

RStudio Package Manager doesn't download packages directly from CRAN. Instead, RStudio maintains a curated s3 bucket that contains metadata about CRAN and package tar files. The metadata is used to track CRAN's day-to-day changes.

See D if your environment does not have access to the RStudio Package Service.

During a sync, the metadata is downloaded to RStudio Package Manager. The metadata is compared against the RStudio Package Manager database to determine what changes need to be applied. Package tarballs are then downloaded to the cache either eagerly or lazily depending on the sync mode.

Lazy sync mode is recommended.

7.3.2 Eager vs Lazy

The sync mode is configured using the `CRAN.SyncMode` property. The property defaults to "lazy", but can be set to "eager" for eager package fetching. See A.1

7.3.2.1 Eager

For eager fetching, RSPM attempts to download any package sources that could be served to users. This means that RSPM will download packages under a number of scenarios:

1. When a repository subscribes to the CRAN source.
2. If a repository is subscribed to the CRAN source, then packages will be downloaded when a sync occurs, initiated with the CLI `sync` command or initiated based on the configured schedule.
3. When a package is added to a curated CRAN source with the `add` command (and at least one repository has subscribed to the curated CRAN source).
4. When a curated CRAN source is updated with the `update` command (and at least one repository has subscribed to the curated CRAN source).

All four of these actions happen automatically. Additionally, an admin can force RSPM to download packages using the CLI `fetch` command.

If a repository subscribes to the CRAN source, then all CRAN packages will downloaded. Downloading all of CRAN during an initial sync can take a significant amount of time, bandwidth, and disk space. However,

downloading packages **does not block end users** from accessing repositories that subscribe to the source. End users can request any package as soon as the metadata is sync'd. If a user requests a package that is not already downloaded, the package will be immediately downloaded and served to the client.

7.3.2.2 Lazy

If RStudio Package Manager is set up for lazy fetching, it downloads packages as the packages are requested by end users. Package Manager will still download the metadata from CRAN on the sync schedule to keep the RStudio Package Manager database updated. The database serves as the source of truth for package availability. The benefit of lazy fetching is a smaller footprint in terms of network bandwidth and disk space.

7.3.2.3 Package Caching

In either mode, each version of a package is only downloaded once. RStudio Package Manager always checks the local cache to see if the required tar file is already available.

7.3.2.4 Changing Modes

If you change the sync mode, the following occurs:

Lazy to Eager - Eager fetching is applied to future syncs. When you next synchronize, all packages that have not yet been downloaded will be downloaded.

Eager to Lazy - Lazy fetching is applied to all future syncs. A sync with in-progress downloads will be completed.

7.3.3 Updates from CRAN

The cran source is updated according to a schedule set using the `CRAN.SyncSchedule` property in the RStudio Package Manager configuration file. This property accepts a string in crontab format. See A.1.

We recommend syncing every day. CRAN adds or updates tens to hundreds of packages each day.

The `SyncSchedule` property does not necessarily determine when a repository will make updated packages available to users. If the repository subscribes directly to the cran source, users will see updates according to the sync schedule. In contrast, if the repository subscribes to a curated CRAN source, an administrator must explicitly update the source in order for updates to become available.

In addition, updating the repository does not automatically push updates to users. **A repository specifies what packages are available, but the R user is in control of when and how to update the packages used by a project.**

See the section on managing change control for more information.

RStudio Package Manager keeps track of old versions of packages as well. Old versions of packages are available in the repository's archive, and are listed in the RStudio Package Manager web UI. This allows users to roll back updates if necessary or install packages as they existed at a prior time.

7.4 Curated CRAN Sources

Curated CRAN sources allow administrators to create and update approved subsets of CRAN. The behavior is best explained in an example.

Assume that RStudio Package Manager has been configured to sync CRAN updates daily.

January 1st - An administrator creates a curated CRAN source and is given a list of desired packages.

January 2nd - The administrator can use the `add` command with the `dryrun` flag, supplying the list of desired packages. RStudio Package Manager will identify all of the required dependencies and create a proposal. The proposal includes the set of packages to be added as well as information about each package, such as license type. This information can be used to facilitate an external review process.

January 15th - The proposal is approved. The administrator returns to RStudio Package Manager and runs the `add` command again, replacing the `dryrun` flag with a transaction ID included in the proposal. **The set of packages is installed from CRAN as they existed on January 1st, the date the source was created**.

January 20th - The administrator receives a request to add a new package to the set of approved packages. The admin uses the `add` command with the `dryrun` flag, supplying the new package as an argument. **RStudio Package Manager will create a proposal using the version of CRAN as it existed on January 1st**. In order to ensure compatibility between the packages added to the source, RStudio Package Manager will add to the set of packages by pulling from CRAN as it existed the day the source was created. As before, if the proposal is accepted, the admin can commit the changes.

January 30th - Now the administrator gets a request to update the approved packages. In order to keep all packages consistent, the entire set is updated at once using the `update` command. Like the `add` command, the `update` command supports a `dryrun` flag that will enumerate all the changes needed to update the set of packages from January 1st to January 30th.

February 1st - The proposal is approved and the administrator completes the `update` command by using the transaction ID included in the dry run update. **The set of packages is now tied to CRAN on January 30th.** Future add commands will use this pinned date, until another update sequence occurs.

To summarize, curated CRAN sources allow admins to create a subset of CRAN at a point in time. Administrators can add packages to the subset from the same frozen point in time. Administrators can also update the subset to a newer point in time. Each change supports a dry run that creates a proposal and a confirmation run that applies the proposal.

Given a list of desired packages, RStudio Package Manager automatically determines the full set of dependencies and also tracks those dependencies over time. Admins can elect to include suggested dependencies or only required dependencies by using the `include-suggests` flag. During each update, older versions of packages are archived, ensuring that tools like packrat and RStudio Connect work seamlessly with the curated CRAN subset.

The `update` command will be impacted by the sync schedule defined on the server. If the server only syncs every few weeks, `update` will only reference the latest data from CRAN **available on the server**.

8 Configuring RStudio Server

Administrators can configure RStudio Server (Pro) to automatically use RStudio Package Manager when users install packages.

The necessary configuration files are documented in the RStudio Server admin guide, example configurations are provided here for common scenarios.

8.1 A Single Repository

In the most common scenario, users will install all packages from a single RStudio Package Manager repository that may contain CRAN packages and internal packages. An admin can discourage users from changing this repository setting. In this scenario, configure RStudio Server:

1. In `/etc/rstudio/rsession.conf`, set:

```
r-cran-repos=https://<your-package-manager-server-address>/<repo-name>/latest
```

The exact URL to use is available in the RStudio Package Manager Setup page for the repository.

2. (RStudio Server Pro only) To disable the repository option menu and discourage users from changing the repository setting, also include in `/etc/rstudio/rsession.conf`:

```
allow-r-cran-repos-edit=0
```

8.2 Internal Packages and CRAN Packages

Another common scenario is to setup two repositories in RStudio Package Manager, one to serve CRAN packages and one to serve internal packages. (This setup also applies if you're only using RStudio Package Manager for internal packages and want to allow users to access a public CRAN mirror).

1. In `/etc/rstudio/rsession.conf`, remove any existing `r-cran-repos` configuration.
2. Create a file called `/etc/rstudio/repos.conf`. The file should contain:

```
CRAN=https://<your-package-manager-server-address>/<cran-repo-name>/latest
Internal=https://<your-package-manager-server-address>/<internal-repo-name>/latest
```

The name `Internal` can be replaced with whatever name makes sense for your organization, but the repo containing CRAN packages should be indexed with the keyword `CRAN`.

8.3 Allow Users to Optionally Add Additional Repos

In addition to the two scenarios described above, some organizations may have additional repositories for certain groups of users, e.g. "bleeding edge" packages available in a dedicated "dev" repository. Administrators can configure RStudio's `Global Options` menu to automatically offer RStudio Package Manager repositories as optional, secondary repos that a user can opt into using.

1. In `/etc/rstudio/rsession.conf`, set:

```
r-cran-repos-url=https://<your-package-manager-server-address>/__api__/repos
allow-r-cran-repos-edit=1
```

8.4 Precedence of Settings

If you have an `Rprofile.site` file that modifies the repository setting, that file **will take precedence** over these settings.

> Some Ubuntu and Debian R binaries (installed via `apt-get install`) include an Rprofile.site file that modifies the repository option. Unless you intended to have this file, we recommend removing /usr/lib/R/etc/Rprofile.site.

9 Managing Change Control

An important concern in any organization adopting R is code stability. Tooling is required to meet the needs of R users which include access to the latest packages and reproducible environments. In regulated environments, an additional level of control may be required to ensure certain teams or projects only use groups of packages that have undergone additional vetting. Administrators also need to worry about the availability of different versions of R and system dependencies that R packages may rely on.

There is not a single solution that meets the needs of every organization. RStudio Package Manager is designed to support a number of strategies. Three common strategies are outlined below.

9.1 Approach 1: Client Side Management

The most common approach to managing change control is for the user to manage package dependencies for a project. An easy way R users can do this is by creating project-specific libraries of packages. In this model, administrators specify what packages are available in the repository. Users create a specific library for each project and are responsible for installing the packages required for their project. Over time, users can upgrade packages or add new packages.

For reproducibility, users must maintain a record of which package versions are included in their libraries. This record allows the user to recreate the environment from a clean slate by requesting the specific version of each package from the repository. There are a number of R packages designed to help users accomplish this task, including the `packrat` package. RStudio Connect automates this approach for users when they deploy content.

This model relies on the ability of the repository to handle multiple versions of each package. R repositories handle this task using a specific structure called an archive. RStudio Package Manager automatically handles archiving packages and knows how to respond to requests for older packages from tools like `packrat` and `devtools`.

9.2 Approach 2: Repository Versioning

Another common approach is for users to pin a project to a specific version of the package repository. In contrast to the first approach, this approach does not rely on the repository managing an archive of package versions. Instead, the entire repository is versioned. Like the first approach, this strategy relies on the user creating a specific package library for each project. However, instead of recording the versions of each package in use, the user simply records the version of the repository and the names of the necessary packages. To recreate the package environment, the user simply has to install the requisite packages from the recorded version of the repository. The most popular example of this approach is the `checkpoint` package which relies on Microsoft's online copy of CRAN.

RStudio Package Manager supports this strategy by automatically associating every action in a repository with a versioned id.

> Future versions of RStudio Package Manager will allow users to opt in to navigating versions of the repository and installing packages from a specific version.

9.3 Approach 3: Locked Down Repositories

Some organizations prefer to manage which versions of packages are used across the organization instead of on a project-by-project basis. A common strategy entails:

1. Administrators test a set of desired packages and then freeze that set. Users are able to use only those versions of the packages.
2. A few times per year, administrators test and update the set of packages. They might also approve new packages.

RStudio Package Manager enables this strategy through curated CRAN sources. RStudio Package Manager also enhances this strategy by adding the option for a step between 1 and 2. When an administrator creates a curated CRAN source and defines a set of packages (step 1), RStudio Package Manager also records the state of all of CRAN at that moment. This enables an administrator to later add packages to the approved subset without worrying about updating the entire set.

Curated CRAN sources can be used with any of these strategies if organizations want to apply additional governance policies such as limiting packages to those with approved licenses.

9.4 What about versions of R?

In order to capture and recreate the environment for an R project, organizations must account for managing the version of R in addition to managing R packages.

In general, a library of installed packages is only compatible with a single version of R. For a new version of R, users or administrators must re-install the desired set of R packages. Luckily, managing a repository of packages facilitates re-installing packages into different libraries.

9.5 What about Docker?

Docker can play an important role in creating reproducible environments. Docker specifies the steps for creating an environment in a Dockerfile. A Docker image is created by running each of those steps.

Normally, a Dockerfile for an R project will include one or more steps that install R packages:

```
RUN Rscript -e 'install.packages(...)'
```

Using Docker in this way facilitates reproducibility, but **Docker alone is not sufficient to guarantee reproducibility**. The reason Docker is insufficient is because each time the
Docker image is recreated the R command to install packages is re-run. Just like a Dockerless environment, this command can return different results over time.

Luckily, the same approaches outlined above also work with Docker. Simply replace the `install.packages` command with a variation that uses a frozen repo or restores a specific environment using a tool like `packrat`.

10 Database

RStudio Package Manager supports multiple database options. Currently, the supported databases are SQLite and PostgreSQL.

Customize the `Database.Provider` property with a database scheme appropriate for your organization. See Section A.8 for details.

Here is a partial configuration which chooses to use SQLite.

```
[Database]
Provider = sqlite
```

10.1 SQLite

SQLite is the default database provider.

RStudio Package Manager will use SQLite database if the `Database.Provider` setting has a value of `sqlite` or if `Provider` is not present in the configuration file.

```
[Database]
Provider = sqlite
```

You can also specify the directory to store the SQLite file on your file system. This can be done by specifying `SQLite.Dir` in the configuration file.

```
[SQLite]
Dir = /mnt/rstudio-pm/sqlite
```

If this field is not specified, it will default to `{Server.DataDir}/db`. This location **must** exist on local storage.

If the location for `Server.DataDir` is not local storage but a networked location over NFS, configure the `SQLite.Dir` setting so it still resides on some local volume.

10.2 PostgreSQL

PostgreSQL is an available database provider which is more powerful and performant than SQLite.

You must provide your own Postgres server which will likely be a separate server from your RStudio Package Manager server (but not required). We currently support any 9.x version greater than or equal to 9.2. Your Postgres server does not have to be dedicated to RStudio Package Manager, but it must have its own dedicated database.

To use Postgres, select it as your provider with `Database.Provider = postgres`. You will also need to provide a fully qualified Postgres URL in `Postgres.URL`. The user credentials supplied in this URL must have read/write permissions to the database referenced at the end of url. Please ensure that you have already created a blank database with the name given at the end of your URL.

Using a separate database for usage data:

```
[Database]
Provider = postgres

[Postgres]
URL = "postgres://username:password@db.seed.co/rstudio-pm"
UsageDataURL = "postgres://username:password@db.seed.co/rstudio-pm-usage"
```

Using a single database with a separate schema for usage data:

```
[Database]
Provider = postgres

[Postgres]
URL = "postgres://username:password@db.seed.co/rstudio-pm"
UsageDataURL = "postgres://username:password@db.seed.co/rstudio-pm?search_path=metrics"
```

10.3 Usage Data

RStudio Package Manager relies on two databases by default. The primary database stores information needed to run the service including the arrangement of repositories, sources, and packages. Another database is used to record usage data like the number of times a package was downloaded.

If using SQLite, these two databases will be created automatically in the configured directory. If using PostgreSQL, you will need to define two different databases or schemas: `Postgres.URL` and `Postgres.UsageDataURL`.

If you do not wish to track usage data, you can disable this feature by setting `Server.UsageDataEnabled = false`. If disabled, usage data will not be tracked or displayed. You can use the `Server.UsageDataRetention` setting to alter the amount of usage data you wish to retain (the default is 365 days). Increasing this value will consume more disk space for the usage data database and may negatively impact performance slightly over time for busy servers.

11 Outbound Proxy

If you are syncing RStudio Package Manager to create your own CRAN repository (see 7.3), the server will need access to the internet to download manifest and package data. If you need to use an outbound proxy server, RStudio Package Manager uses the `HTTP_PROXY` and `HTTPS_PROXY` environment variables to configure HTTP or HTTPS proxy behavior.

Typically, it is best to define these environment variables in the startup script used to run RStudio Package Manager. This varies depending on the service daemon. Instructions for `systemd` and Upstart are listed below.

11.0.1 systemd (Red Hat/CentOS 7, SUSE 12, Ubuntu 16.04+)

1. Stop the RStudio Package Manager service. See 3.1.
   ```
   sudo systemctl stop rstudio-pm
   ```
2. Edit the service configuration.
   ```
   sudo vi /etc/systemd/system/rstudio-pm.service.d/user.conf
   ```
   Define an `HTTP_PROXY` or `HTTPS_PROXY` environment variable:
   ```
   [Service]
   Environment="HTTP_PROXY=http://username:password@proxy.example.com:8080"
   Environment="HTTPS_PROXY=http://username:password@proxy.example.com:8080"
   ```
3. Reload the `systemd` configuration.
   ```
   sudo systemctl daemon-reload
   ```
4. Start the RStudio Package Manager service. See 3.1.
   ```
   sudo systemctl start rstudio-pm
   ```

11.0.2 Upstart (Ubuntu 14.04, Red Hat 6)

1. Stop the RStudio Package Manager service. See 3.1.
   ```
   sudo stop rstudio-pm
   ```
2. Edit the service configuration.
   ```
   sudo vi /etc/init/rstudio-pm.override
   ```
   Define an `HTTP_PROXY` or `HTTPS_PROXY` environment variable:
   ```
   env HTTP_PROXY=http://username:password@proxy.example.com:8080
   env HTTPS_PROXY=http://username:password@proxy.example.com:8080
   ```
3. Start the RStudio Package Manager service. See 3.1.
   ```
   sudo start rstudio-pm
   ```

12 Running with a Proxy

If you are running RStudio Package Manager behind a proxy server, you need to be sure to configure the proxy server so that it correctly handles all traffic to and from RStudio Package Manager. This section describes how to correctly configure a reverse proxy with Nginx or Apache HTTPD.

When RStudio Package Manager is behind a proxy, it is important to send the original request URL information to RStudio Package Manager so that it can generate fully qualified URLs and return them the requester. For this reason, when proxying to RStudio Package Manager, we recommend adding a header, `X-RSPM-Request`, to the request. This header value should be the absolute URL of the original request made by the user or browser (i.e. `https://rspm.company.com/some/path`)

Some proxies (like Amazon Web Services Elastic Load Balancer), do not make it possible to add custom headers. Because of this, if this header is not supplied, "best efforts" are made utilizing the standard headers `X-Forwarded-Proto`, `X-Forwarded-Host`, and `X-Forwarded-Port` to parse the original request URL. If your proxy removes a server prefix from the path, `X-Forwarded` headers will not work for your use case, and you should use `X-RSPM-Request`. If both `X-RSPM-Request` and `X-Forwarded` headers are supplied, `X-RSPM-Request` takes precedence.

12.1 Nginx Configuration

On Ubuntu, a version of Nginx that supports reverse-proxying can be installed using the following command:

```
apt-get install nginx
```

On Red Hat/CentOS, you can install Nginx using the following command:

```
yum install nginx
```

On SUSE, you can install Nginx using the following command:

```
sudo zypper install nginx
```

To enable an instance of Nginx running on the same server to act as a front-end proxy to RStudio Package Manager you would use a configuration like the following in your `nginx.conf` file. This configuration assumes RStudio Package Manager is running on the same host as Nginx and listening for HTTP requests on the `:4242` port. If you are proxying to RStudio Package Manager on a different machine or port, replace the `localhost:4242` references with the correct address of the server where RStudio Package Manager is hosted.

```
http {
    map $http_upgrade $connection_upgrade {
        default upgrade;
        ''      close;
    }
    server {
        listen 80;
        location / {
            proxy_set_header    X-RSPM-Request $scheme://$host:$server_port$request_uri;
            proxy_pass http://localhost:4242;
        }
    }
}
```

If you want to serve RStudio Package Manager from a custom path (e.g. `/rspm`) you would edit your `nginx.conf` file as shown below:

```
http {
    map $http_upgrade $connection_upgrade {
```

```
        default upgrade;
        ''      close;
    }
    server {
        listen 80;
        location /rspm/ {
            rewrite ^/rspm/(.*)$ /$1 break;
            proxy_set_header    X-RSPM-Request $scheme://$host:$server_port$request_uri;
            proxy_pass http://localhost:4242;
            proxy_redirect / /rspm/;
        }
    }
}
```

After adding these entries you'll then need to reload Nginx so that the proxy settings take effect.

On `systemd` systems (Red Hat/CentOS 7, SUSE 12, Ubuntu 16.04+):

```
systemctl restart nginx
```

On Upstart systems (Ubuntu 14.04, Red Hat 6):

```
restart nginx
```

12.2 Apache Configuration

The Apache HTTPD server can act as a front-end proxy to RStudio Package Manager by first enabling three modules:

```
a2enmod rewrite
a2enmod headers
a2enmod proxy_http
```

The following configuration will permit proxying to RStudio Package Manager from the `:3737` port. Depending on the layout of your Apache installation, you may need the `Listen` and `VirtualHost` directives in different files.

```
Listen 3737

<VirtualHost *:3737>
    RewriteEngine on
    RequestHeader set X-RSPM-Request "%{REQUEST_SCHEME}s://%{HTTP_HOST}s%{REQUEST_URI}s"
    ProxyPass / http://172.17.0.1:4242/
    ProxyPassReverse / http://172.17.0.1:4242/
</VirtualHost>
```

You can serve RStudio Package Manager from a custom path (e.g. `/rspm`) with a configuration like the following:

```
Listen 3737

<VirtualHost *:3737>
    RewriteEngine on
    RedirectMatch ^/rspm$ /rspm/
    RequestHeader set X-RSPM-Request "%{REQUEST_SCHEME}s://%{HTTP_HOST}s%{REQUEST_URI}s"
    ProxyPass /rspm/ http://172.17.0.1:4242/
    ProxyPassReverse /rspm/ http://172.17.0.1:4242/
    Header edit Location ^/ /rspm/
```

```
</VirtualHost>
```

13 Security & Auditing

13.1 Browser Security

There are a variety of security settings that can be configured in RStudio Package Manager. Some of these settings are enabled by default but can be customized while others are opt-in. Below are some of the security features worth considering.

13.1.1 Guaranteeing HTTPS

If you can guarantee that your server should only ever be accessed over a TLS/SSL connection (HTTPS), then you can consider enabling the `HTTPS.Permanent` setting. This elevates the security of your server by requiring that future interactions between your users and this server must be encrypted.

> Enabling this setting may keep users from being able to access your RStudio Package Manager instance if you later disable HTTPS or if your certificate expires. Use this setting only if you will permanently provide a valid TLS/SSL certificate on this server.

Behind the scenes, this makes two changes:

1. Introduces HTTP Strict Transport Security (HSTS) by adding a `Strict-Transport-Security` HTTP header with a `max-age` set to 30 days. HSTS ensures that your users' browsers will not trust a service hosted at this location unless it is protected with a trusted TLS/SSL certificate.
2. Enforces the `Secure` flag on cookies that are set. This prohibits your users' browsers from sending their RStudio Package Manager cookies to a server without an HTTPS-secured connection.

13.1.2 Content Sniffing

The `Server.ContentTypeSniffing` setting can be used to configure the `X-Content-Type-Options` HTTP header. This protects your users from a certain class of malicious uploads and is enabled by default.

When disabled (the default), the `X-Content-Type-Options` HTTP header will be set to a value of `nosniff` to tell browsers not to sniff the content type. If enabled, no such header will be provided.

13.1.3 Content Embedding

The `X-Frame-Options` HTTP header is used to control what content can be embedded inside other content in a web browser. The relevant attack is commonly referred to as a "clickjack attack" and involves having your users interact with a sensitive service without their knowledge.

> Some advertised values for this header are not supported across all browsers. RStudio Package Manager does not restrict the values of these headers.

13.1.4 Custom Headers

If you need to include additional HTTP headers that are not covered by any of the above features, you can include your own custom headers on all responses from RStudio Package Manager using the `Server.CustomHeader` setting.

This feature can be used to accommodate various other security practices that are not explicitly available as options elsewhere in RStudio Package Manager. For instance, X-XSS-Protection, Content Security Policy (CSP), HTTP Public Key Pinning (HPKP), and Cross-origin Resource Sharing (CORS) could all be configured using custom headers.

> Custom headers are added to the HTTP response early during request processing. Values may later be overwritten or modified by other header settings. This includes both the security preferences described earlier in this chapter and other headers used internally by RStudio Package Manager. You should not depend on a custom header that conflicts with a header already in use by RStudio Package Manager.

The `Server.CustomHeader` takes a value of the header name and its value separated by a colon. Whitespace surrounding the header name and its value are trimmed. You can use this setting multiple times as in the following example:

```
[Server]
CustomHeader = "HeaderA: some value"
CustomHeader = "HeaderB: another value"
```

14 High Availability and Load Balancing

Multiple instances of RStudio Package Manager can share the same data in highly available (HA) and load-balanced configurations. In this document, we refer to these configurations as "HA" for brevity.

14.1 HA Checklist

Follow the checklist below to configure multiple RStudio Package Manager instances for HA:

1. Ensure that all node clocks are synchronized 14.2
2. Install and Configure the same version of RStudio Package Manager on each node - 2
3. HA requires using a PostgreSQL database. All nodes in the cluster must use the same PostgreSQL database.
4. Configure each server's `Server.DataDir` (5.6) to point to the same shared location. Be sure to read 14.3.3 for additional information on the recommended settings for the shared directory.

14.2 HA Time Synchronization Requirements

The clocks on all nodes in an HA configuration must be synchronized. We recommend configuring NTP for clock synchronization.

14.3 HA Limitations

14.3.1 Node Management

RStudio Package Manager nodes in an HA configuration are not self-aware of HA. The load-balancing responsibility is fully assumed by your load balancer, and the load balancer is responsible for directing requests to specific nodes and checking whether nodes are available to accept request

14.3.2 Database Requirements

RStudio Package Manager only supports HA when using a PostgreSQL database.

14.3.3 Shared Data Directory Requirements

RStudio Package Manager manages repository content within the server's data directory. This data directory must be a shared location, and each node's `Server.DataDir` must point to the same shared location. See 5.6 for more information on the server's data directory. We recommend and support NFS version 3 or 4 for file sharing.

RStudio Package Manager relies on being able to efficiently detect new files inside of the NFS-shared `DataDir`. By default, NFS clients are configured to cache responses for up to 60 seconds, which means that it can take up to a minute before an RStudio Package Manager service is able to respond to certain requests. For most deployments, this is an unacceptably long delay.

Therefore, we strongly recommend that you modify your NFS client settings for the mount on which you'll be hosting your `DataDir`. Typically, the best way to accomplish this is to set `lookupcache=pos` for your NFS mount, which will allow existing files to be cached but will contact the NFS server directly to check for the existence of new files. If this setting is not acceptable for your mount, you could alternatively consider shortening `acdirmax` or `actimeo` so that your client becomes aware of new files within, say, 5 seconds, instead of the default of 60.

14.4 Updating HA Nodes

When applying updates to the RStudio Package Manager nodes in your HA configuration, you should follow these steps to avoid errors due to an inconsistent database schema:

1. Stop all RStudio Package Manager nodes in your cluster.
2. Upgrade one RStudio Package Manager node. The first update will upgrade the database schema (if necessary) and start RStudio Package Manager on that instance - 3.2.
3. Upgrade the remaining nodes.

If you forget to stop any RStudio Package Manager nodes while upgrading another node, these nodes will be using a binary that expects an earlier schema version, and will be subject to unexpected and potentially serious errors. These nodes will detect an out-of-date database schema within 30 seconds and shut down automatically.

14.5 Downgrading

If you wish to move from an HA environment to a single-node environment, please follow these steps:

1. Stop all RStudio Package Manager services on all nodes
2. Reconfigure your network to route traffic directly to one of the nodes, unless you wish to continue using a load balancer.
3. If you wish to move all shared file data to the node, then
    1. Configure the server's `Server.DataDir` to point to a location on the node, and copy all the data from the NFS share to this location - 5.6
4. If you wish to move the databases to this node, install PostgreSQL on the node and copy the data. Moving the PostgreSQL databases from one server to another is beyond the scope of this guide. Please note that we do not support migrating from PostgreSQL to SQLite.
5. Start the RStudio Package Manager process 3.1

15 Quick Start Configuration Guides

These guides are designed to help administrators configure RStudio Package Manager for common scenarios. Before beginning, follow the Getting Started with Installation steps to install and license RStudio Package

Manager.

The commands in the quick start guides need to be run as an appropriately privileged user. By default, the user should be a member of the `rstudio-pm` group. In addition, the instructions assume you've setup an alias: `alias rspm='/opt/rstudio-pm/bin/rspm'` or have added the binary to your path.

Once you are done setting up RStudio Package Manager, share the URL to RStudio Package Manager with users. Users will be able to setup R or RStudio to use RStudio Package Manager by following the instructions included in each repository's Setup page.

If you prefer, you can configure RStudio Server (Pro) to use RStudio Package Manager without requiring user setup, see the configuration instructions.

Quick Start Index:

15.1 Serving CRAN Packages

A common use case for RStudio Package Manager is making CRAN packages available in environments with restricted internet access. To do so, start by ensuring that RStudio Package Manager has the appropriate metadata using the `sync` command. RStudio Package Manager pulls packages and metadata from the RStudio CRAN service, it is *not* necessary to configure an upstream CRAN URL. Then, create a repository and subscribe it to the built-in source named "cran".

```
# Initiate a sync:
rspm sync --wait

# Create a repository:
rspm create repo --name=prod-cran --description='Access CRAN packages'

# Subscribe the repository to the cran source:
rspm subscribe --repo=prod-cran --source=cran
```

The behavior of the repository will depend on the whether your server is set up for lazy or eager downloading. This choice is made in the RStudio Package Manager config file. Likewise, future updates will occur on a schedule dictated in the configuration file. See A.1 for more information.

After completing these steps, the `prod-cran` repository will be available in the web interface.

15.2 Distributing Local Packages

Many teams have a handful of internally built packages. Before beginning, create the **bundled** version of each package and copy the resulting tar files to the RStudio Package Manager server. If you are unfamiliar with building the bundled version of a package, reach out to the R developer maintaining the package.

```
# Create a local source:
rspm create source --name=prod-internal-src

# Add each local package tar file to the source:
#  The tar file must be rwx by the user running the CLI and the account running
```

```
#  RSPM (rstudio-pm by default)
rspm add --source=prod-internal-src --path='/path/to/package_1.0.tar.gz'

# Create a repository:
rspm create repo --name=prod-internal --description='Stable releases of our internal packages'

# Subscribe the repository to the source:
rspm subscribe --repo=prod-internal --source=prod-internal-src
```

RStudio Package Manager automatically supports multiple versions of each package. When the R developers are ready for the next release of a package, simply run:

```
rspm add --source=prod-internal-src --path='/path/to/package_2.0.tar.gz'
```

RStudio Package Manager will ensure that version 2.0 is the default for new installations, but will keep version 1.0 available in the repository's archive for users who wish to use the older version.

Most internal packages will depend on packages from CRAN. In this case, the easiest option is to create a repository that includes the local packages and their dependencies.

See the quickstart sections for serving local packages and all of CRAN or serving local packages and a curated subset of CRAN.

15.3 Distributing Local Packages along with CRAN Packages

Often, for convenience, organizations opt to distribute their local packages along with CRAN packages in a single repository. This setup gives a single URL for all the organization's R packages. To do so, follow the quick start guides above to create a local source and cran source, then:

```
# Confirm sources exist:
rspm list sources; # should have: prod-internal-src, cran

# Create repository:
rspm create repo --name=prod --description='Production R packages from CRAN and our local packages'

# Subscribe the repository to the sources:
rspm subscribe --repo=prod --source=prod-internal-src; rspm subscribe --repo=prod --source=cran
```

In the final step, the order of subscriptions is important. Packages from the local source will be favored if there are conflicts. See 7.2.2 for details on how conflicts are resolved.

15.4 Supplementing CRAN with Packages from GitHub

In addition to a production repository, some organizations allow advanced R users to access "bleeding-edge" versions of packages available on GitHub.

Native support for accessing GitHub packages will be available in a future release.

Today, it is possible to enable bleeding-edge packages using a local source. To begin, download a tar file for each desired GitHub-based package and then copy it to RStudio Package Manager.

```
# Create a local source:
rspm create source --name=dev-pkgs-src

# For each tar file, add the package:
rspm add --source=dev-pkgs-src --path='/path/to/github_pkg.tar.gz'
```

To finish, run:

```
# Create a repository:
rspm create repo --name=dev --description='CRAN plus bleeding edge, development versions of some packag

# Subscribe the repository to the local source first:
rspm subscribe --repo=dev --source=dev-pkgs-src

# Subscribe the repository to the cran source second:
rspm subscribe --repo=dev --source=cran
```

The result of these steps is a repository that contains the GitHub version of the package in addition to CRAN packages. The GitHub package is installed from RStudio Package Manager using `install.packages` NOT `devtools`.

In the example above, the ordering of the source subscriptions is important. Imagine the desired GitHub package is ggplot2. ggplot2 will be available from cran and the local source, `dev-pkgs-src`. When ggplot2 is requested, RStudio Package Manager looks in the order of the subscriptions. First it will look for ggplot2 in `dev-pkgs-src`. Since ggplot2 is in the set of bleeding edge packages, RStudio Package Manager will find ggplot2 in `dev-pkgs-src` and serve the bleeding edge version to users. Any ggplot2 dependencies that are not in `dev-pkgs-src` will be pulled from cran. RStudio Package Manager displays whether the package came from cran or an alternative source in the package page for the repository.

15.5 Serving a Subset of Approved CRAN Packages

Some organizations only want to give access to an approved list of CRAN packages. A curated CRAN source enables administrators to serve the approved list of packages as well as any dependencies, while enabling admins to preview changes, add new packages, and run updates.

Create a file containing one package name per line. For example, `/tmp/packages.csv`:

```
plumber
shiny
ISLR
```

Start by creating a `curated-cran` source. Then use the `add` command to preview the changes needed to add the packages.

```
# Ensure you have cran metadata:
rspm sync --wait

# Create the curated-cran source:
rspm create source --name=subset --type=curated-cran

# Dry run to see proposed packages:
rspm add --file-in='/tmp/packages.csv' --source=subset --dryrun
```

The result will contain information on all the packages that will be added. The proposal can be saved to a csv file using the `csv-out` flag. The required dependencies for the named packages are automatically discovered included. Optionally use the `--include-suggests` flag to also discover and add suggested packages.

```
This action will install the following packages:

Name        Version  Path License                  Needs Compilation Dependency
BH          1.66.0-1      BSL-1.0                  false             true
crayon      1.3.4         MIT + file LICENSE       false             true
digest      0.6.15        GPL (>= 2)               false             true
```

```
htmltools   0.3.6         GPL (>= 2)                false             true
httpuv      1.4.3         GPL (>= 2) | file LICENSE false             true
ISLR        1.2           GPL-2                     false             false
jsonlite    1.5           MIT + file LICENSE        false             true
later       0.7.3         GPL (>= 2)                false             true
...         ...           ...                        ...               ...

To complete this installation, execute this command without the --dryrun flag.
You will need to include the --transaction-id=281 flag.
```

If the proposal is acceptable, run the command again, but use the `transaction-id` indicated in the dry run output. Finally, create a repository and subscribe the repository to the source.

```
# Commit the changes:
rspm add --file-in='/tmp/packages.csv' --source=subset --transaction-id=281

# Create a repository:
rspm create repo --name=approved-cran --description='Approved packages from CRAN'

# Subscribe the repository to the source:
rspm subscribe --repo=approved-cran --source=subset
```

When the sourceis created, RStudio Package Manager automatically pins the "subset" source to a frozen point in time on CRAN. New packages can be added from this frozen snapshot of CRAN by repeating the process.

To update the entire set of packages to the latest data available from CRAN, use the `update` command. (Note that the latest data will depend on the server's sync schedule).

```
rspm update --source=subset --dryrun

# Like the add command, a preview of the changes is printed out
# along with a transaction-id. Use this id to commit the changes

rspm update --source=subset --transaction-id=281
```

Please review the full description of curated CRAN sources to understand further which points in time `add` and `update` will use.

15.6 Serving a Subset of Approved CRAN Packages and Local Packages

An extension of the previous use case is serving a subset of CRAN and a set of internal packages from within a single repository. Start by following the steps to create a source with the subset of approved packages and a local source with the desired internal packages.

Next, for each internal package, obtain the list of the package's dependencies from the package developer. Create a csv file with one line per package name, e.g. `internal_deps.csv`:

```
xml2
jsonlite
```

Use the `add` command to add these additional dependencies to the curated CRAN source.

```
# List sources, should include
# prod-internal-source and subset:
rspm list sources
```

```
# Add the local packages' dependencies:
rspm add --source=subset --file-in='internal_deps.csv' --dryrun

# Proposed changes will be printed out, to commit the changes
# run the command again with the transaction-id:
rspm add --source=subset --file-in='internal_deps.csv' --transaction-id=283
```

Finally, create a repository and subscribe the repository to both sources. The order of the subscription commands is important, see 7.2.2 for details.

```
# Create a repo:
rspm add repo=prod-pkgs --description='Stable release of our internal packages and approved CRAN packag

# Subscribe the repo to both sources:
rspm subscribe --repo=prod-pkgs --source=prod-internal-source
rpsm subscribe --repo=prod-pkgs --source=subset
```

16 Frequently Asked Questions

16.1 Setting Up RStudio Package Manager

Where should I install RStudio Package Manager? What server specifications do you recommend?

RStudio Package Manager can be installed on a physical server, cloud instance, virtual machine, or a long-living Docker container with persistent storage. RStudio Package Manager is supported on most Linux distributions, specifically listed in the installation section of the admin guide.

RStudio Package Manager requires 2 GB of RAM and 250 GB of disk storage. There are no restrictions or requirements for CPUs.

Does RStudio Package Manager require internet access?

RStudio Package Manager can be used without internet access, but most organizations will want RStudio Package Manager to have outbound internet access to RStudio's CRAN service.

R clients using RStudio Package Manager do not need internet access, just access to RStudio Package Manager. RStudio Package Manager is designed to help organizations with restricted internet access in their R environment.

Can RStudio Package Manager be setup for High Availability?

Yes. RStudio Package Manager can be setup on 2 or more servers and placed behind a load balancer. This setup requires a PostgreSQL database and shared storage; see the admin guide for details.

Can I use RStudio Package Manager in an offline, air-gapped environment?

Yes, there are a number of options for air-gapped environments; see the instructions in the getting started guide.

What permissions are necessary to install RStudio Package Manager?

RStudio Package Manager's default installation requires root privileges in order to configure the service daemon for RStudio Package Manager, setup log files, unpack RStudio Package Manager, and activate the license. It is possible to manually install RStudio Package Manager without root; see the associated appendix.

What permissions are needed to manage repositories in RStudio Package Manager?

Repositories and packages are managed with a command line interface located at `/opt/rstudio-pm/bin/rspm`. By default, users must be a member of the `rstudio-pm` Unix group to use the CLI. The group that determines access can be customized.

How do I configure RStudio Package Manager?

RStudio Package Manager's primary configuration file is located at `/etc/rstudio-pm/rstudio-pm.gcfg` by default.

Do I need a database for RStudio Package Manager?

RStudio Package Manager uses a database to store meta-data and serve packages. Administrators do not need to interact with the database. By default, RStudio Package Manager will use an included SQLite database. Optionally, RStudio Package Manager can use an external PostgreSQL database and in a cluster RStudio Package Manager requires PostgreSQL.

16.2 Setting Up RStudio Package Manager

Can I use RStudio Package Manager to access CRAN?

Yes, see the CRAN quickstart guide. RStudio Package Manager can be configured to serve CRAN packages to servers behind your firewall. Behind the scenes, RStudio Package Manager maintains metadata on all CRAN packages, but only downloads content when necessary.

How do I use RStudio Package Manager to share local packages?

See the quickstart guide for local packages. RStudio Package Manager makes it easy to distribute internal packages and automatically handles archiving older versions of the package when new releases are added.

Can I setup multiple repositories in RStudio Package Manager?

Yes, RStudio Package Manager allows administrators to create multiple repositories with different package and source compositions. For example, you may have a repo with CRAN packages and a repo with local packages, or you may create a single repository that contains local and CRAN packages. You could create a production repo with stable releases and a development repo with the bleeding edge versions of packages. RStudio Package Manager optimizes storage to prevent needless duplication.

Where does RStudio Package Manager get packages?

RStudio Package Manager currently supports packages from CRAN and local packages. Packages from GitHub can be added by downloading the Git repo and creating a tar file. CRAN packages are managed by an RStudio service and can be downloaded eagerly or lazily on-demand.

16.3 Using RStudio Package Manager from R or RStudio

How do I tell R and RStudio to use RStudio Package Manager?

Each repository in RStudio Package Manager will contain a Setup web page with instructions for configuring R to use RStudio Package Manager. Once configured, R users can access packages using standard tools including `install.packages`, `packrat`, and `devtools`.

Can I configure RStudio Server to use RStudio Package Manager?

Yes, an administrator can configure RStudio Server (Pro) to use RStudio Package Manager by default, see the configuration instructions.

16.4 Controlling Access to Packages

How do I specify which packages should be available in RStudio Package Manager?

Administrators configure the available packages, sources, and repositories using an admin command line interface.

16.5 Package Updates

How do I update packages in RStudio Package Manager?

RStudio Package Manager can automatically update packages from CRAN on an admin-defined schedule. Local packages can be updated at any time by adding the tar file for the new version of the package.

RStudio Package Manager automatically archives older versions of packages during updates, ensuring older versions are available to users. Older versions are listed on the web page for each package.

16.6 Managing Change Control

Does RStudio Package Manager help users manage the versions of packages installed?

RStudio Package Manager was built to support all the common strategies organizations use to manage different versions of R packages. See 9 for more information.

Is RStudio Package Manager compatible with packrat?

Yes, RStudio Package Manager works with packrat for CRAN and local packages.

16.7 RStudio Package Manager and Docker

Can RStudio Package Manager run in Docker?

Yes, RStudio Package Manager can run in a Docker container that has persistent, mounted storage.

If I use Docker to manage package dependencies, should I use Package Manager?

Docker images can be used to create an environment for a certain project or analysis. However, the use of Docker is orthogonal to RStudio Package Manager. For example, inside a Dockerfile for an R project you will normally see a line that installs R packages, e.g.

```
RUN Rscript -e 'install.packages(...)'
```

This installation step benefits from RStudio Package Manager in the same way that non-Docker users benefit from RStudio Package Manager.

A Configuration Options

This appendix documents the RStudio Package Manager configuration file format and enumerates the user-configurable options.

The RStudio Package Manager configuration file is located at `/etc/rstudio-pm/rstudio-pm.gcfg`. This configuration is read at startup and controls the operation of the service.

The RStudio Package Manager configuration file uses the **gcfg** (Go Config) format, which is derived from the Git Config format.

Here is an example of that format showing the different property types:

```
; Comment
[BooleanExamples]
property1 = true
property2 = off
property3 = 1

[IntegerExamples]
Property1 = 42
Property2 = -123

[DecimalExamples]
Property1 = 3.14
Property2 = 7.
Property3 = 2
Property4 = .217

[StringExamples]
Property1 = simple
Property2 = "quoted string"
Property3 = "escaped \"quote\" string"

[MultiStringExamples]
ListProperty = black
ListProperty = blue
ListProperty = green

[DurationExamples]
Property1 = 1000000000
Property2 = 500ms
Property3 = 1m15s  ; comment with a property
```

Comments always start with a semi-colon (;) and continue to the end of the line. Comments can be on lines by themselves or on a line with a property or section definition.

Configuration sections always begin with the name of the section bounded by square brackets. A section may appear multiple times and are additive with the last value for any property being retained. The following two configuration examples are equivalent.

```
[Example]
A = aligator
B = 2

[Example]
A = aardvark
C = shining
```

```
[Example]
A = aardvark
B = 2
C = shining
```

Each configuration property must be included in its appropriate section. Property and section names are interpreted case-insensitively.

Property definitions always have the form:

```
Name = value
```

The equals sign (=) is mandatory.

If a property happens to to be given more than once, only the last value is retained. The "multi" properties are an exception to this rule; multiple entries are aggregated into a list.

```
[MultiExample]
Color = black
Color = blue

[NonMulti]
Animal = cat
Animal = dog
```

If `Color` is a multi-string property, both the "black" and "blue" values are used. If `Animal` is a normal string property, only the value "dog" is retained.

Configuration properties all have one of the following types:

string A sequence of characters. The value is taken as all characters from the first non-whitespace character after equal sign to the last non-whitespace character before the end-of-line or start of a comment. Double-quotes (") are supported, but usually unnecessary. A literal double-quote MUST be escaped and quoted itself like `QuotedValue = "J.R. \"Bob\" Dobbs"`.

multi-string A property that takes multiple string values. The property name is listed with each individual input value. For example, providing `Color = black` and `Color = blue` results in two separate values.

boolean A truth value. The values `true`, `yes`, `on`, and `1` are interpreted as true. The values `false`, `no`, `off`, and `0` are interpreted as false.

integer An integral value.

decimal A numeric value with an optional fractional component. Values with and without a decimal point are allowed.

duration A value specifying a length of time. When provided as a raw number, the value is interpreted as nanoseconds. Duration values can also be specified as a sequence of decimal numbers, each with optional fraction and unit suffix, such as `300ms`, `1.5h`, or `1m30s`.

Valid time units are `ns` (nanoseconds), `us` (microseconds), `ms` (milliseconds), `s` (seconds), `m` (minutes), and `h` (hours).

version A string representing a version. A version may have one to four numeric components, separated by periods or hyphens. Examples include `2`, `2.5`, `2.5.6`, `2.5.6.1`, and `2.5-6-11`.

bytesize A string representing a size in bytes. Valid examples include 10MB, 1 tb, 3 terabytes, and 2 GB.

Each configuration property documented in this appendix includes its description, data type, and default value.

Some properties are marked as "reloadable". Sending a `HUP` signal to the Package Manager process causes the on-disk configuration to be reread. The server is reconfigured with the latest values of these reloadable properties. See 3.1 for details about sending a `HUP` signal to your Package Manager process.

> Use a `HUP` signal when your configuration changes are limited to properties marked as reloadable. Perform a full restart of RStudio Package Manager when changing other properties.

A.1 Server

The `Server` section contains configuration properties which apply across the whole of RStudio Package Manager and are not appropriate for the other sections, which are generally narrower.

The properties which follow all must appear after `[Server]` in the configuration file.

`DataDir` The directory where RStudio Package Manager will store its variable data.

Type: string

Default: `/var/lib/rstudio-pm`

`CacheDir` The directory containing the RStudio Package Manager cache.

Type: string

Default: `{Server.DataDir}/cache`

`CacheTimeout` The amount of time we wait for files to appear in the cache directory after the database reports they are available. The default of 65 seconds should accommodate the NFS default directory caching limit of 60 seconds

Type: duration

Default: 65s

`Address` A public URL for this RStudio Package Manager server. Must be configured to enable features like including links to your content in emails.

Type: string

Default: *<empty-string>*

`ContentTypeSniffing` If disabled, sets the `X-Content-Type-Options` HTTP header to `nosniff`. When enabled, removes that header, allowing browsers to mime-sniff responses.

Type: boolean

Default: `false`

`ServerName` By default, Package Manager sets the `Server` HTTP header to something like `RStudio Package Manager v1.2.3`. This setting allows you to override that value.

Type: string

Default: *<empty-string>*

`AccessLog` Path to the file that RStudio Package Manager will use for its access logs. Disabled when empty.

Type: string

Default: `/var/log/rstudio-pm.access.log`

`CustomHeader` Custom HTTP header that should be added to responses from Package Manager in the format of `key: value`. The left side of the first colon in the string will become the header name; everything after the first colon will be the header value. Both will be trimmed of leading/trailing whitespace. This will always add a new header with the specified value; it will never override a header that Package Manager would otherwise have set. Multiple definitions can be used to provide multiple custom headers.

Type: multi-string

Default: *unspecified*

`FrameOptionsUI` The value for the `X-Frame-Options` HTTP header for the Package Manager UI and all other Package Manager pages. If empty, no header will be added.

Type: string

Default: `DENY`

`HideVersion` When `true`, the `/version` API returns a blank string and the product version is not displayed in the UI.

Type: boolean

Default: `false`

Reloadable: true

`SockFileDir` RStudio Package Manager will use this directory to create a socket file for admin connections.

Type: string

Default: /var/run/rstudio-pm

`UsageDataEnabled` If true, will enable tracking of package downloads.

Type: boolean

Default: true

`UsageDataRetention` The number of days for which usage data will be retained. If `0`, data will be preserved forever.

Type: integer

Default: 365

`UsageDataSchedule` The schedule for usage data aggregation. Must be a valid crontab string.

Type: string

Default: 0 0 * * *

Reloadable: true

A.2 API

The `API` section contains configuration properties which apply to the RStudio Package Manager API.

The properties which follow all must appear after `[API]` in the configuration file.

`MaxApiResults` Limits the maximum number of results that can be returned from API endpoints that support a `_limit` parameter. When zero, there is no limit. Please consider performance implications before raising this limit.

Type: integer

Default: `1000`

`DefaultApiResultsLimit` The default limit on the number of results returned from API endpoints that support a `_limit` parameter. When zero, there is no limit.

Type: integer

Default: `100`

A.3 HTTP

The `HTTP` section contains configuration properties which control the ability of RStudio Package Manager to listen for HTTP requests. RStudio Package Manager must be configured to listen for either HTTP or HTTPS requests (allowing both is acceptable).

These properties must appear after `[HTTP]` in the configuration file.

`Listen` RStudio Package Manager will listen on this network address for HTTP connections. The network address can be of the form `:80` or `192.168.0.1:80`. Either `HTTP.Listen` or `HTTPS.Listen` is required.

Type: string

Default: *<empty-string>*

`NoWarning` Disables warnings about insecure (HTTP) connections.

Type: boolean

Default: `false`

A.4 HTTPS

The `HTTPS` section contains configuration properties which control the ability of RStudio Package Manager to listen for HTTPS requests. RStudio Package Manager must be configured to listen for either HTTP or HTTPS requests (allowing both is acceptable).

These properties must appear after `[HTTPS]` in the configuration file.

`Listen` RStudio Package Manager will listen on this network address for HTTPS connections. The network address can be of the form `:443` or `192.168.0.1:443`. Either `HTTP.Listen` or `HTTPS.Listen` is required.

Type: string

Default: *<empty-string>*

`Key` Path to a private key file corresponding to the certificate specified with `HTTPS.Certificate`. Required when `HTTPS.Certificate` is specified.

Type: string

Default: *<empty-string>*

`Certificate` Path to a TLS certificate file. If the certificate is signed by a certificate authority, the certificate file should be the concatenation of the server's certificate followed by the CA's certificate. Must be paired with `HTTPS.Key`.

Type: string

Default: *<empty-string>*

`Permanent` Advertises to all visitors that this server should only ever be hosted securely via HTTPS. WARNING: if this is set to true – even temporarily – visitors may be permanently denied access to your server over an unsecured (non-HTTPS) protocol. This sets the `secure` flag on all session cookies and adds a `Strict-Transport-Security` HTTP header with a value of 30 days.

Type: boolean

Default: `false`

A.5 HttpRedirect

The `HttpRedirect` section contains configuration properties which control the ability of RStudio Package Manager to listen for HTTP requests and then redirect all traffic to some alternate location. This is useful when paired with an `HTTPS.Listen` configuration.

These properties must appear after `[HttpRedirect]` in the configuration file.

`Listen` RStudio Package Manager will listen on this network address for HTTP connection and redirect to either the `HttpRedirect.Target` or `Server.Address` target location. The network address can be of the form `:8080` or `192.168.0.1:8080`. Useful when you wish all requests to be served over HTTPS and send users to that location should they accidentally visit via an HTTP URL. Must be paired with either `HttpRedirect.Target` or `Server.Address`.

Type: string

Default: *<empty-string>*

`Target` The target for redirects when users visit the `HttpRedirect.Listen` HTTP service. `Server.Address` is used as a redirect target if this property is not specified.

Type: string

Default: *<empty-string>*

A.6 Licensing

The `Licensing` section contains configuration properties which control how RStudio Package Manager interacts with its licensing system.

These properties must appear after `[Licensing]` in the configuration file.

`LicenseType` Enable remote or local validation. `local` is traditional activation, whereas `remote` uses floating licensing.

Type: string

Default: `local`

`RemoteRetryFrequency` When Package Manager loses its lease, it will begin automatically attempting to acquire a lease by `RemoteRetryFrequency`. Use a value of `0` to disable retries.

Type: duration

Default: `10`

A.7 CRAN

The `CRAN` section contains configuration properties related to the CRAN source in RStudio Package Manager.

`ManifestURL` The URL to the CRAN manifest service. Do not use this setting unless you want to get CRAN data from your own server.

Type: string

Default: `https://rspm-sync.rstudio.com`

`SyncSchedule` The schedule for CRAN synchronization. Must be a valid crontab string. Leave blank for manual syncing only.

Type: string

Default: *<empty-string>*

Reloadable: true

`SyncMode` The sync mode for CRAN synchronization. Must be either `eager` or `lazy`

Type: string

Default: `lazy`

Reloadable: true

`CRANTimeout` The timeout for downloading checkpoints or changes (JSON) from the CRAN manifest

Type: duration

Default: 30m

`FetchTimeout` The timeout for downloading a CRAN package tarball.

Type: duration

Default: 10m

A.8 Database

The `Database` section contains configuration properties which control the location of and how RStudio Package Manager interacts with its databases.

These properties must appear after `[Database]` in the configuration file.

`Provider` The type of database to use

Type: string

Default: sqlite3

`MaxIdleConnections` The maximum number of database connections that should be retained after they become idle. If this value is less-than or equal-to zero, no idle connections are retained.

Type: integer

Default: `0`

`MaxOpenConnections` The maximum number of open connections to the database. If this value is less-than or equal-to zero, then there is no limit to the number of open connections.

Type: integer

Default: `0`

`ConnectionMaxLifetime` The maximum amount of time a connection to the database may be reused. If this value is less-than or equal-to zero, then connections are reused forever.

Type: duration

Default: `0`

A.9 SQLite

The `SQLite` section contains configuration properties which control the location of and how RStudio Package Manager interacts with the SQLite database.

These properties must appear after `[SQLite]` in the configuration file.

`Dir` The directory containing the RStudio Package Manager database. Must reference a directory on the local filesystem and not on a networked volume like NFS.

Type: string

Default: `{Server.DataDir}/db`

A.10 Postgres

The `Postgres` section contains configuration properties which control the location of and how RStudio Package Manager interacts with its postgres.

These properties must appear after `[Postgres]` in the configuration file.

`URL` The fully qualified URL to connect to a Postgres database

Type: string

Default: *<empty-string>*

`UsageDataURL` The fully qualified URL to connect to a Postgres database where usage information will be written.

Type: string

Default: *<empty-string>*

B Package Ecosystem

The R package ecosystem has a few key components.

B.1 Packages

Packages are the primary extension mechanism for R. They can be used to share functions, datasets, and documentation. An R package can exist in a few states:

B.1.1 Source

An R package is composed of a series of directories and files. The source of an R package is just a top-level directory containing the coponents of the package. Package authors work with source packages during development. Git(hub) repositories store source packages.

B.1.2 Bundle

A bundled package is a package thats been compressed into a single file. By convention, package bundles in R use the extension .tar.gz.

B.1.3 Binary

A binary package is the result of building a source package for a specific operating system. Binary packages are single files that are ready for installation on their specific operating systems.

B.1.4 Installed

An installed package is a binary package that has been decompressed into a package library and is ready for use by R.

B.2 Repositories

Repositories organize R packages for distribution to end users. Repositories contain package bundles and binaries that are organized in a specific way so that users can install packages from the repository using R's `install.packages` command. CRAN and Bioconductor are examples of R repositories.

B.3 Git(hub)

Many R package sources are stored in version controlled directories. A popular versioning tool is Git. Github, as an extension of Git, houses many package sources. The `devtools` R package includes convenience functions for installing packages from the package source contained on a Git repository, including Github. Used in this manner, git repositories and Github are one way to distribute R packages, but Github and Git repositories *are not* R package repositories.

B.4 Libraries

End users of R typically interact with installed packages that live in libraries. Package libraries are just directories containing installed packages. When a package is requested by R, R searches the different library directories to find the installed package.

R libraries are very flexible. In the past, R users have set up libraries for specific projects or set up a system-wide library used across multiple projects. In multi-tenant servers it has been common to have both a system library shared by all users and user-specific libraries.

A best practice is to set up per-project libraries alongside a package cache.

C Changing RunAs User

C.1 Default Account

The installer creates a user account and group named `rstudio-pm` and runs the RStudio Package Manager service under this account. See 2.1.3.1 for more information.

C.2 Changing The RunAs Account (Service)

You can configure RStudio Package Manager to run under another account. The steps below serve as a guide for reconfiguring RStudio Package Manager to run under an account named `thor` with a primary group of `heroes` instead of the default `rstudio-pm:rstudio-pm`.

1. Stop the RStudio Package Manager service. See 3.1.

```
# Ubuntu 14 and Red Hat/CentOS 6
sudo stop rstudio-pm

# Ubuntu 16+, Red Hat/CentOS 7, SUSE 12
sudo systemctl stop rstudio-pm
```

2. Create a new group and user account

```
sudo groupadd heroes
sudo useradd -r -g heroes -M -s /sbin/nologin thor
```

In order to use the CLI tool, a user must be a member of the primary group of the user that starts the RStudio Package Manager service.

In this example, RStudio Package Manager is started by the user `thor`. The primary group of the `thor` account is `heroes`, so users must be members of the `heroes` group to use the CLI.

3. Edit the service configuration

Ubuntu 14 and Red Hat/CentOS 6

```
sudo vi /etc/init/rstudio-pm.override
```

Change these lines:

```
env RSTUDIO_PM_USER=thor
env RSTUDIO_PM_GROUP=heroes
```

Ubuntu 16+, Red Hat/CentOS 7, SUSE 12

```
sudo vi /etc/systemd/system/rstudio-pm.service.d/user.conf
```

Change these lines:

```
[Service]
User=thor
Group=heroes
```

4. Change ownership of files and directories

```
# Configuration file
sudo chown thor:heroes /etc/rstudio-pm/rstudio-pm.gcfg

# Log files
sudo chown thor:heroes /var/log/rstudio-pm.*

# Data directory (or `Server.DataDir`, if configured for a custom location)
sudo chown -R thor:heroes /var/lib/rstudio-pm

# Run directory
sudo chown -R thor:heroes /var/run/rstudio-pm

# If you have a custom `Sqlite.Dir` (e.g., `Sqlite.Dir = /database/directory`)
```

```
sudo chown -R thor:heroes /database/directory

# If you have a custom `Server.CacheDir` (e.g., `Server.CacheDir = /path/to/cache`)
sudo chown -R thor:heroes /path/to/cache
```

5. Remove remaining domain socket file (if any)

```
sudo rm /var/run/rstudio-pm/rstudio-pm.sock
```

6. Start the RStudio Package Manager service. See 3.1.

```
# Ubuntu 14 and Red Hat/CentOS 6
sudo start rstudio-pm

# Ubuntu 16+, Red Hat/CentOS 7, SUSE 12
sudo systemctl daemon-reload # Reload the systemd process
sudo systemctl start rstudio-pm
```

7. Verify that the `rstudio-pm` service is running under the `thor` account.

```
ps -axj | grep `id -u thor`
```

8. Check `/var/log/rstudio-pm.log` to verify that the server started up with no errors.

C.3 Using the New RunAs Account (CLI)

After changing the service RunAs user, your CLI users must be members of the `heroes` group. For example:

```
sudo useradd -g heroes hulk
sudo passwd hulk
su hulk
/opt/rstudio-pm/bin/rspm <command>
```

D Air-Gapped RStudio Package Manager

RStudio Package Manager communicates with a RStudio CRAN service to access CRAN packages and metadata. In offline environments, it is possible to directly download the necessary data from the RStudio CRAN service and then copy it to an offline RStudio Package Manager server.

This guide includes 3 sections:

- Instructions for an initial air-gapped setup (D.1)
- Instructions for performing regular updates (D.2)
- Instructions for upgrading RStudio Package Manager in an air-gapped environment (D.3)

D.1 Initial Setup

1. Install the AWS CLI tools in an online machine. Confirm by running `aws help`.
2. Run the command `rspm air-gap` in the offline RStudio Package Manager server. See 6 for more information about the `rspm` commands.

The air-gap command will print information and output a command similar to:

```
MAJOR=`curl https://rspm-sync-staging.rstudio.com/v2/version.txt`; \
echo "aws s3 sync --no-sign-request --exclude=* --include=v2/version.txt \
--include=v2/${MAJOR}/* s3://rstudio-pm-sync/ ./"
```

3. In the online machine, create a new directory, e.g `cran-data`. Copy the command from step 2 and run it from within the directory on the online machine. A new command will be printed starting with `aws s3 sync`. Copy the entire output and run it in the the same location. The sequence of input and output looks like:

```
# Input:
MAJOR=`curl https://rspm-sync-staging.rstudio.com/v2/version.txt`; \
echo "aws s3 sync --no-sign-request --exclude=* --include=v2/version.txt \
--include=v2/${MAJOR}/* s3://rstudio-pm-sync/ ./"

# Output:
aws s3 sync --no-sign-request --exclude=* --include=v2/version.txt \
--include=v2/2/* s3://rstudio-pm-sync/ ./

# Input:
aws s3 sync --no-sign-request --exclude=* --include=v2/version.txt \
--include=v2/2/* s3://rstudio-pm-sync/ ./
```

These steps will begin the download which includes 50+ GB of data and may take some time to complete.

The result will be a directory with files prefixed by a version indicator, e.g.

```
cran-data/v2/version.txt
cran-data/v2/2
cran-data/v2/2/...
```

4. Create a directory in the offline RStudio Package Manager server. If you have a cluster of nodes, use shared storage for this directory.

```
mkdir /var/lib/rstudio-pm/cran
```

Copy the data downloaded in step 3 from the online machine to the directory on the offline RStudio Package Manager server. For completely isolated servers, you may need to copy the data to a physical drive in order to move it to the offline environment.

Confirm by checking that the offline directory has the same number of files as the original directory on the online machine.

Finally, modify the permissions on the directory in the offline RStudio Package Manager server, changing ownership to the unix account running RStudio Package Manager, `rstudio-pm` by default:

```
sudo chown -R rstudio-pm /var/lib/rstudio-pm/cran
```

5. Next, configure the offline RStudio Package Manager server to use the downloaded data. To do so, modify the RStudio Package Manager Configuration file to include the following properties in the `CRAN` configuration section:

```
[CRAN]
ManifestURL = A URL in the form, `file:///<the directory you created in step 4>`
```

For example, if your CRAN data directory is at `/var/lib/rstudio-pm/cran`, the file `/etc/rstudio-pm/rstudio-pm.gcfg` should contain:

```
[CRAN]
ManifestURL = file:///var/lib/rstudio-pm/cran
```

Once the file is updated, restart the RStudio Package Manager Service.

6. Follow the remainder of the instructions in the admin guide for setting up sources and repositories using the `rspm` commands in the offline RStudio Package Manager server. The `rspm sync` command will use the downloaded data to populate the necessary CRAN data and packages.

D.2 Regular Updates

It is important to regularly update data available on the offline server. Updating this data does not automatically make new packages available to end users. The following process pushes updates from the RStudio CRAN service to RStudio Package Manager's metadata. Follow the instructions in the admin guide to make updates available to end users. See 6 (and, specifically, 6.4.2) for more information.

1. In the offline RStudio Package Manager server, run the command `rspm air-gap`. This command will output information along with a command similar to:

```
MAJOR=`curl https://rspm-sync.rstudio.com/v2/version.txt`; \
echo "aws s3 sync --no-sign-request --exclude=* \
--include=v2/version.txt --include=v2/${MAJOR}/* s3://rstudio-pm-sync/ ./"
```

2. In the online machine, navigate to the directory created during the initial setup, e.g. `cran-data`, and paste the command from step 1. A second command will be printed. Copy and run the second command in the same location. Any newly available data will be downloaded, but existing data will not be re-downloaded.

3. Copy the directory from the online machine to the folder created in the offline RStudio Package Manager during the initial setup, e.g. `/var/lib/rstudio-pm/cran`. Ensure that the directory is still owned by the unix account running RStudio Package Manager, `rstudio-pm` by default.

D.3 Upgrading RStudio Package Manager

A new version of RStudio Package Manager may require data from a new version of the RStudio CRAN service. To ensure a smooth upgrade with limited downtime, we recommend the following steps:

1. You will need a staging environment that mirrors your offline production server. After creating this environment, begin by upgrading the offline staging server to the latest RStudio Package Manager release.

2. Follow the instructions for the *Initial Setup* of an Air-Gapped server in D.1, using the offline staging server.

3. After you have validated that everything works as expected, copy the the CRAN data, e.g. `/var/lib/rstudio-pm/cran`, from the offline staging server to the offline production server.

4. Upgrade the offline production server to the new version of RStudio Package Manager.

5. (Optional) After an upgrade, navigate to the directory storing CRAN data, e.g. `/var/lib/rstudio-pm/cran`. This directory will contain versioned folders, e.g.

```
/v2
/v3
```

Run the command `rspm air-gap` in the offline production server. The output will indicate the version of the RStudio CRAN service required by the current version of RStudio Package Manager, e.g.

> Your RStudio Package Manager uses a CRAN Manifest Schema Version of "v3".

You can safely remove all of the other folders in the directory. In this example, only `/var/lib/rstudio-pm/v3` is necessary.

E Manual Installation for Minimal Root Use

E.1 Discussion

The RStudio Package Manager installer requires root privileges and installs the RStudio Package Manager to `/opt/rstudio-pm`. Additionally, it configures services for Upstart or systemd (depending on your OS), adds log files to `/var/log`, adds a configuration file under `/etc/rstudio-pm`, creates an `rstudio-pm` user account and group, and places a logentries configuration.

It is possible to extract the RStudio Package Manager files manually and configure the service to use configuration and data files at a custom location. Most of these steps can be performed without root privileges.

E.2 Extracting Files

First, download the latest version of RStudio Package Manager from the link provided by RStudio. Then, extract the files from the installer by following the instructions for your operating system, below.

E.2.1 Extracting Files (Ubuntu)

Note you may need to install `binutils` with `sudo apt install binutils` to use the `ar` utility.

```
mkdir rspm
ar x rstudio-pm_<version>_amd64.deb
tar -xzvf data.tar.gz -C rspm/
cd rspm
```

E.2.2 Extracting Files (Red Hat/CentOS 7, SUSE 12)

```
mkdir rspm
cd rspm
rpm2cpio ../rstudio-pm-<version>.x86_64.rpm | cpio -div
```

E.3 Create Directories

Create the directories you need to run RStudio Package Manager.

```
mkdir log
touch log/rstudio-pm.log
touch log/rstudio-pm.access.log
mkdir run
mkdir data
```

E.4 SUSE Only - Create OpenSSL Softlinks

```
  ln -s -f /lib64/libssl.so.1.0.0 ./opt/rstudio-pm/ext/activation/libssl.so.10
  ln -s -f /lib64/libcrypto.so.1.0.0 ./opt/rstudio-pm/ext/activation/libcrypto.so.10
```

E.5 Licensing

The license manager used by RStudio Package Manager supports userspace license activation. If you wish to activate a system-wide license, root privileges are required. See 4 for more details on system-wide licensing.

```
# Initialize a trial in userspace
./opt/rstudio-pm/bin/license-manager initialize --userspace

# Activate a userspace license key, if you have a license key
./opt/rstudio-pm/bin/license-manager activate --userspace <key>
```

E.6 Edit config file

Next, edit the RStudio Package Manager configuration file to point to the directories and access log you created.

```
vi ./etc/rstudio-pm/rstudio-pm.gcfg
```

Add the following lines to the `[Server]` configuration section.

```
[Server]
DataDir = <path-to-unbundled-rspm-dir>/data
SockFileDir = <path-to-unbundled-rspm-dir>/run
AccessLog = <path-to-unbundled-rspm-dir>/log/rstudio-pm.access.log
```

E.7 Start the RStudio Package Manager Service

When RStudio Package Manager is installed without root, the system daemons are not used to automatically start RStudio Package Manager. Instead, the user is required to start RStudio Package Manager manually and ensure the process continues to run. The command to start RStudio Package Manager is:

```
./opt/rstudio-pm/bin/rstudio-pm --config \
./etc/rstudio-pm/rstudio-pm.gcfg >> ./log/rstudio-pm.log 2>&1
```

As an example, `nohup` can be used to launch this command without blocking or depending on the terminal using:

```
nohup ./opt/rstudio-pm/bin/rstudio-pm --config \
./etc/rstudio-pm/rstudio-pm.gcfg >> ./log/rstudio-pm.log 2>&1 &
```

E.8 Use the CLI to Manage RStudio Package Manager

```
./opt/rstudio-pm/bin/rspm --config ./etc/rstudio-pm/rstudio-pm.gcfg help
```

In this mode, every CLI command will require the config flag that points at the configuration file.