RStudio Connect: Admin Guide

Version 1.5.4-13

Abstract

This guide will help an administrator install and configure RStudio Connect 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 Connect allows users to share and collaborate on the results they produce with R such as R Markdown documents, Shiny applications, Plumber APIs, and plots. Source code or rendered artifacts can be deployed into RStudio Connect and selectively shared with other viewers and collaborators within the organization. Some content can even be scheduled to be re-executed and emailed on a given schedule.

RStudio Connect can also help simplify the role of the system administrator tasked with supporting R by offering:

- Detailed metrics for the server and the associated R processes
- Logs for all R processes spawned by Connect
- Secure deployments and interactions with artifacts using SSL/TLS
- Scale a Shiny application beyond a single R process to support additional visitor load

1.1 System Requirements

RStudio Connect is supported on the following distributions of the Linux operating system:

- Red Hat Enterprise Linux/CentOS Linux 6.0+
- Red Hat Enterprise Linux/CentOS Linux 7.0+
- Ubuntu 12.04
- Ubuntu 14.04
- Ubuntu 16.04

We currently only offer installers for the x86-64 architecture and require root privileges both to install and run Connect.

RStudio Connect can be used with R versions 3.1.0 or higher.

RStudio Connect is supported against the latest versions of the following browsers:

- Chrome
- Safari
- Firefox
- Internet Explorer 10
- Internet Explorer 11
- Microsoft Edge

2 Getting Started

This chapter helps you install RStudio Connect on Ubuntu or Red Hat Enterprise Linux/CentOS Linux, learn to manage the server, and perform some initial configuration.

We built this checklist to guide you through that process.

1. Install R - Ubuntu 2.1.1, Red Hat/CentOS 2.1.2

2. Download RStudio Connect installer
3. Install RStudio Connect - Ubuntu 2.1.1, Red Hat/CentOS 2.1.2
4. Set `Server.SenderEmail` - 2.2.1
5. Set `Server.Address` - 2.2.1
6. Configure Authentication - 2.2.2, 10
7. Restart RStudio Connect - 5.1
8. Sign into RStudio Connect - 2.2.3
9. Configure email sending - 2.2.4

2.1 Installation

This section explains how to install R using the public package repositories for your Ubuntu or Red Hat/CentOS server. Chapter 14 explains how to configure RStudio Connect to access multiple versions of R on the same server.

2.1.1 Ubuntu (12.04+)

Connect recommends an installation of R version 3.1.0 or higher. To install the latest version of R you should first add the CRAN repository to your system as described here:

http://cran.rstudio.com/bin/linux/ubuntu/README.html

You can then install R using the following command:

```
$ sudo apt-get install r-base
```

> **Note:** If you do not add the CRAN Ubuntu repository as described above this command will install the version of R corresponding to your current system version. This version of R may be a year or two old. It is strongly recommended that you add the CRAN repositories so you can run the most up-to-date version of R.

RStudio Connect can be configured to use versions of R other than the default system R. Please see Section 14 for details.

You will use `gdebi` to install Connect 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 Connect. 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-connect-1.5.4-13.deb
```

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

```
$ sudo gdebi rstudio-connect-1.5.4-13.deb
```

This will install Connect into `/opt/rstudio-connect/`, and create a new `rstudio-connect` user.

You can now configure the server following the instructions in Section 2.2.1. However, we recommend that you consider installing some additional system dependencies that common R packages require. Without these system dependencies, your users may not be able to use the R packages they require on the server.

Recommended Packages

The following system dependencies are required by many common R packages and nearly all deployments will need to provide these. These package names may vary slightly between different versions of Ubuntu.

```
build-essential
libcurl4-gnutls-dev
openjdk-7-* # may require also executing `R CMD javareconf`
libxml2-dev
libssl-dev
texlive-full # very large dependency, but needed to render PDF documents from R Markdown
```

Supplemental Packages

There are additional system dependencies that may be required for some R packages depending on the types of R packages your users are leveraging. You could consider providing these packages for your users now, or wait until they are requested.

```
libgmp10-dev
libgsl0-dev
libnetcdf6
libnetcdf-dev
netcdf-bin
libdigest-hmac-perl
libgmp-dev
libgmp3-dev
libgl1-mesa-dev
libglu1-mesa-dev
libglpk-dev
tdsodbc
freetds-bin
freetds-common
freetds-dev
odbc-postgresql
libtiff-dev
libsndfile1
libsndfile1-dev
libtiff-dev
tk8.5
tk8.5-dev
tcl8.5
tcl8.5-dev
libgsl0-dev
libv8-dev
```

2.1.2 Red Hat Enterprise Linux/CentOS Linux (6.0+)

Prerequisites

RStudio Connect recommends an installation of R version 3.1.0 or higher. Connect has several dependencies on packages (including R itself) found in the Extra Packages for Enterprise Linux (EPEL) repository. If you don't already have this repository available, add it to your system using the instructions found here: https://fedoraproject.org/wiki/EPEL. On some distributions of Red Hat Enterprise Linux/CentOS Linux, the R package references dependencies that are not available by default. In this case, you may need to edit the `/etc/yum.repos.d/redhat.repo` file to enable the `rhel-6-server-optional-rpms` (by setting `enabled = 1`) before you can install the R package.

After enabling EPEL, ensure that you have installed the version of R available from EPEL with the following command:

```
$ sudo yum install R
```

RStudio Connect can be configured to use versions of R other than the default system R. Please see Section 14 for details.

You can now begin the installation of RStudio Connect. You should have been provided with an RPM file which contains Connect and all of its dependencies (other than R). You can install this rpm file using `yum`. If you have only a link to the RPM file, you can use `wget` to download the file to the current directory.

```
$ sudo yum install --nogpgcheck rstudio-connect-1.5.4-13.rpm
```

This will install Connect into `/opt/rstudio-connect/` and create a new `rstudio-connect` user.

You can now configure the server following the instructions in Section 2.2.1. However, we recommend that you consider installing some additional system dependencies that common R packages require. Without these system dependencies, your users may not be able to use the R packages they require on the server.

Recommended Packages

The following system dependencies are required by many common R packages and nearly all deployments will need to provide these. These package names may vary slightly between different versions of Red Hat Enterprise Linux/CentOS Linux.

```
make
gcc
gcc-c++
libcurl-devel
libxml2-devel
java-1.7.0-openjdk-devel  # may require also executing `R CMD javareconf`
openssl-devel
texlive-*  # VERY large dependency, but needed to render PDF documents from R Markdown
```

2.2 Initial Configuration

RStudio Connect is installed, but requires additional configuration before it is ready for use. This section will help you specify the public URL of your server, configure authentication, and validate that RStudio Connect is able to send email.

2.2.1 Editing the Configuration File

RStudio Connect is controlled by the `/etc/rstudio-connect/rstudio-connect.gcfg` configuration file. You will edit this file to make server-wide configuration changes to the system. See the configuration appendix A for details about this file, its syntax, and the available settings.

Start by setting the `SenderEmail` and `Address` server properties. Both must be specified in the `[Server]` section of your configuration file.

The `Server.SenderEmail` property is the email address from which Connect sends emails. It is important that the sendmail or SMTP configuration RStudio Connect uses be willing and able to send email from this `SenderEmail` address. Otherwise, Connect will not be able to successfully send email. See Section 2.2.4 for more details about mail sending.

The `Server.Address` property is the public URL used to access the server. When accessible over a non-standard port, this URL must specify both hostname and port. This setting enables Connect to include links in emails that send users to the appropriate location on the server.

> The standard HTTP port is 80; the standard HTTPS port is 443.

Important Note

Please use a publicly available URL (like the one you set in `Server.Address`) when connecting `rsconnect` or the RStudio IDE to your RStudio Connect server. If a non- public address (e.g., `localhost`) is used for publishing content, `rsconnect` will not be able to automatically open the published content in the user's browser.

Whenever RStudio Connect is deployed behind a proxy, you *must* configure the `Server.Address` setting with the proxied location. RStudio Connect normally returns URLs that are in terms of its local address. The `Server.Address` property causes Connect to use an alternate base location when building URLs. Setting `Server.Address` to the location of your proxy will produce URLs in terms of your proxy address instead of the Connect local address.

Here is a sample configuration specifying both `SenderEmail` and `Address`.

```
[Server]
SenderEmail = rstudio-connect@company.com
Address = https://rstudio-connect.company.com/
```

Use the instructions in Section 5.1 to restart RStudio Connect after altering the `rstudio-connect.gcfg` configuration file.

2.2.2 Authentication

It is important that you specify the correct style of authentication for your organization. RStudio Connect includes a built-in authentication mechanism and supports a number of external authentication integrations, which are detailed in Section 10.

You **must** establish the correct form of authentication before using RStudio Connect. Migrating from one style of authentication to another is **NOT SUPPORTED**.

2.2.3 Sign In!

Use a web browser to visit the RStudio Connect dashboard. This has a default location of http://your-connect-server:3939/. Click the "Sign In" link. If you are using an external authentication provider, specify your login credentials. If you are using password authentication, follow the "Create a new account" link and configure your account.

The first account will be marked as an RStudio Connect administrator. Please use this account to configure mail sending. These settings are necessary in order for Connect to be able to distribute reports and notify users of errors running their content. Connect also sends confirmation messages when using the default password auth provider.

2.2.4 Email Sending

Visit the RStudio Connect dashboard and sign in as an administrator. Visit the Admin>Settings screen and configure mail sending for your organization.

RStudio Connect supports two options for sending mail:

- Sendmail - The `sendmail` command is used to send messages locally on your server. This relies on a working sendmail configuration or some equivalent replacement.
- SMTP - Mail is sent using an SMTP endpoint and supports SSL and authentication.

Please contact your system administrator if you have questions about which of these options are appropriate.

Be sure to verify your settings by sending a test message!

At this point, RStudio Connect is installed and ready for use. The rest of the administration guide covers additional configuration options.

3 Licensing & Activation

When RStudio Connect 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-connect/bin/license-manager status
```

After purchasing a license to RStudio Connect, you will receive a product key that is used to activate the license on a given system. Each product key given limits usage of RStudio Connect in the following ways...

- Number of user accounts that have signed into RStudio Connect. Once this limit is reached, additional users will not be permitted to sign into RStudio Connect. This limit is enforced the first time each user logs in. Locked users are not counted against this quota. Additionally, users that have not recently been active on the server are not counted against this quota. Users are deemed "inactive" after 365 days without visiting RStudio Connect, though this value may vary for certain licenses.
- Number of users that can access Shiny applications at one moment in time. If this number is exceeded, new anonymous users will be unable to view the Shiny application requested. This limitation does not affect logged in users.
- Whether or not API hosting is supported.

How many are allowed of each metric depends on the license purchased from RStudio.

You can activate your license key with the command:

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

After activation, we recommend restarting the RStudio Connect server. A change in license status will eventually be detected by the product; a forced restart ensures that change is seen immediately.

```
$ sudo stop rstudio-connect
$ sudo start rstudio-connect
```

Your platform may need alternate commands to restart RStudio Connect. Please see Section 5.1 for instructions specific to your operating system version.

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

```
$ sudo /opt/rstudio-connect/bin/license-manager deactivate
```

3.1 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-connect/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`.

3.2 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-connect/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 our offline activation application or send 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-connect/bin/license-manager activate-offline <activation-file>
```

After activation, we recommend restarting the RStudio Connect server. A change in license status will eventually be detected by the product; a forced restart ensures that change is seen immediately.

```
$ sudo stop rstudio-connect
$ sudo start rstudio-connect
```

Your platform may need alternate commands to restart RStudio Connect. Please see Section 5.1 for instructions specific to your operating system version.

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

```
$ sudo /opt/rstudio-connect/bin/license-manager deactivate-offline
```

Executing this command will print an offline deactivation request to the terminal which you should copy and paste into our offline activation application or send to RStudio customer support (support@rstudio.com).

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

```
$ sudo /opt/rstudio-connect/bin/license-manager status-offline
```

3.3 Licensing errors

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

3.4 Floating Licensing

If you stop and start the RStudio Connect server frequently, for instance because it's running inside a virtual machine or container, you may wish to use floating licensing instead of traditional licensing.

To use floating licensing, you run a small, lightweight server, which holds the licenses and distributes leases. When a RStudio Connect server starts, it will connect to the license server and obtain a temporary lease on a license, releasing it when the RStudio Connect server is stopped. Using this method, you can have more RStudio Connect servers than licenses, so long as you do not run more instances at once than you have licenses.

3.4.1 The RStudio Connect License Server

The RStudio License Server site contains license server downloads for all RStudio products. Download and install the license server for RStudio Connect on a machine in your network. You then activate the license server with your license key, using a command like the following:

```
$ sudo connect-license-server activate <product-key>
$ sudo connect-license-server start
```

A license key which distributes floating licenses is not the same as a traditional license key, and the two cannot be used interchangeably. If you have purchased traditional license keys and wish to exchange them for a floating license key, or vice versa, please get in touch with RStudio customer support.

The file `/etc/connect-license-server.conf` contains configuration settings for the RStudio Connect License server, including the network port to listen on and any proxy settings required for connecting to the Internet.

3.4.2 License Server Offline Activation

The `connect-license-server activate` command requires an internet connection. If your license server has no connection to the internet it's also possible to perform an offline activation. The process for doing this on the license server is identical to the process used to activate RStudio Connect offline. Generate an offline activation request as follows:

```
$ sudo connect-license-server activate-offline-request <product-key>
```

Executing this command will print an offline activation request to the terminal which you should copy and paste and then send 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 connect-license-server activate-offline <activation-file>
$ sudo connect-license-server restart
```

3.4.3 Using Floating Licensing

Once your license server is up and running, you need to tell RStudio Connect to use floating licensing instead of traditional licensing.

/etc/rstudio-connect/rstudio-connect.gcfg

```
[Licensing]
LicenseType = remote
```

The value `remote` indicates that RStudio Connect should connect to a remote licensing server to obtain a license; the `value` local can be used to explicitly specify traditional (local) activation.

Then, tell RStudio Connect which licensing server to connect to:

```
$ sudo /opt/rstudio-connect/bin/license-manager license-server <server-hostname-or-ip>
```

Restart the RStudio Connect server so that the license changes can be detected by the product.

You only need to run the `license-server` command once; RStudio Connect saves the server name and will use it on each subsequent startup.

By default, the RStudio Connect License Server listens on port 8999. If you wish to use a different port, you will need to specify the port in `/etc/connect-license-server.conf`, and specify license-server to RStudio Connect as `<server-hostname-or-ip:port>`.

3.4.4 Configuring License Leases

When using floating licenses, you can optionally determine how long the license leases last by setting the lease length value on the licensing server. This value is in seconds, so for instance to make license leases last 30 minutes you would use the following syntax:

`/etc/connect-license-server.conf`

```
<lease length="1800"/>
```

The lease length controls how frequently a RStudio Connect server needs to contact the licensing server to renew its license lease in order for the lease to remain valid.

A shorter lease length will increase tolerance to failures by making leases available for reuse more quickly. RStudio Connect will release its lease immediately if shut down normally, but if abnormally terminated, the lease will not be released until it expires.

A longer lease length will increase tolerance to transient failures on the network. Any such issues that can be resolved before the lease is due for renewal won't interrupt use of RStudio Connect.

We generally recommend using a longer lease length. Use a short lease length only if your environment routinely encounters abnormal terminations of the container/instance on which RStudio Connect runs.

3.4.5 Lease Expiration and Renewal

Under normal conditions RStudio Connect will automatically renew its license lease in a configurable interval as described above. However, there are situations in which it will be unable to do so, such as a network problem, or an issue on the host running the license server.

When RStudio Connect cannot obtain a license lease, either because there are no leases currently available or because it can't reach the licensing server, it will begin automatically attempting to acquire a lease every 10 seconds. This interval is configurable; for instance, to retry every 30 seconds instead you would set the following value:

/etc/rstudio-connect/rstudio-connect.gcfg

```
[Licensing]
RemoteRetryFrequency = 30s
```

If you don't want RStudio Connect to attempt to reestablish a license lease automatically, set the value to 0 to disable retries. In this case you will need to manually restart RStudio Connect in order to reestablish the lease. This can be useful if you often run more RStudio Connect servers than you have keys for, and wish to have more control over which RStudio Connect servers receive license leases from the limited pool on the license server.

3.4.6 Troubleshooting Floating Licensing

To validate that the license server has been successfully activated, run the activation-status command. This will report the version of the server as well as the license key and the number of available slots.

```
$ sudo connect-license-server activation-status
```

If your server is activated but you're still having trouble with floating licensing, you can tell the RStudio Connect License Server to emit more detailed logs. Change the log level to notification:

/etc/connect-license-server.conf

```
<log file="/var/log/rstudio-licensing.log" level="notification"/>
```

Then, restart the license server, tail the licensing log, and start your RStudio Connect server.

```
$ sudo connect-license-server restart
$ tail -f /var/log/rstudio-licensing.log
```

At the notification level, the licensing log will tell you the total number of licenses associated with your key, and how many are currently in use. It will also notify you when a RStudio Connect server acquires a lease, and when that lease is released, renewed, or expired. No rotation is done for this log, so it's recommended to use the warning level in production.

4 Files & Directories

4.1 Program Files

The RStudio Connect installers place all program files into the `/opt/rstudio-connect` directory.

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

4.2 Configuration

The RStudio Connect configuration file is `/etc/rstudio-connect/rstudio-connect.gcfg`. This file is initially owned by `root` with permissions `0600`. You will edit this file to properly configure RStudio Connect for your organization.

A configuration management tool like Puppet or Chef can be used to maintain the `rstudio-connect.gcfg` file. We recommend that it remain owned by `root` and have permissions `0600`, as your configuration may need to contain passwords and other sensitive information.

RStudio Connect upgrades will not overwrite customizations to the `rstudio-connect.gcfg` file.

4.3 Server Log

The RStudio Connect server log is located at `/var/log/rstudio-connect.log`. This file is owned by `root` with permissions `0600`.

If `logrotate` is available when RStudio Connect 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.

4.4 Access Logs

The RStudio Connect HTTP access logs are located at `/var/log/rstudio-connect.access.log`. This file is owned by `root` 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 Connect 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.

4.5 Application Logs

Each R process launched by RStudio Connect produces output that is retained within the `jobs` subdirectory of the RStudio Connect data directory (see Section 4.6 for details). These directories and files are managed by the server. They are retained for 30 days and subsequently removed from the system.

Application logs are available in the RStudio Connect dashboard. The dashboard settings page for deployed content contains a **Logs** section containing execution details for each launched R process. Standard output and standard error are captured and available.

4.6 Variable Data

RStudio Connect manages uploaded Shiny applications, Plumber APIs, R Markdown documents, and plots. All of the variable data associated with this content is stored within the server's data directory. This includes:

- Deployment bundles as uploaded by the user.
- Directories containing unpacked bundles, including R source code.
- R packages, as demanded by the deployed code.
- Rendered R Markdown documents.

The RStudio Connect data directory also contains information used by the server in managing your deployed content. This includes:

- The RStudio Connect `SQLite` database and encryption key.
- R process execution information including logged output.
- Parameter overrides for R Markdown documents.

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

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

The RStudio Connect SQLite database **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-connect

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

4.6.1 Permissions

Data directory permissions are established by RStudio Connect as files are created. This section documents the general ownership patterns you will find under the RStudio Connect data directory.

Directories directly accessed from R applications will usually be owned by the `Applications.RunAs` user. This setting defaults to use an `rstudio-connect` account created during RStudio Connect installation. The `rstudio-connect` account has a default primary group also named `rstudio-connect`. We use the account and group name `rstudio-connect` throughout this section instead of referencing the property name.

Directories used during metrics collection are owned by the `rstudio-connect` user (customizable via the `Metrics.User` setting).

Learn more about customizing metrics collection in Section 16.1.

Directories not accessed by R applications or by the monitoring system will be owned by `root`.

`/var/lib/rstudio-connect` is owned by `root` with permissions `0701`.

The `R` subdirectory contains R packages used when content is deployed. The entire `R` directory hierarchy needs to be owned by `rstudio-connect`. Files must have `0600` permissions and directories need `0700` permissions.

The `packrat` subdirectory contains R packages installed on behalf of deployed content. These packages are installed when content is deployed and subsequently used when an application or report executes. The entire `packrat` directory hierarchy needs to be owned by the `rstudio-connect` and the `rstudio-connect` group. Files must have `0640` permissions while directories need `0750` permissions.

The `reports` subdirectory is owned by `root` with `0711` permissions. This contains generated output for report content deployed with source. The nested directories are written to by R processes and are owned by `rstudio-connect` with `0700` permissions. Files contained in this hierarchy will have `0600` permissions.

The `bookmarks` directory contains a bookmarking state subdirectory for each Shiny application. The top-level directory is owned by `root` with `0711` permissions. Each `bookmarks/A_ID` subdirectory is owned by `rstudio-connect` and the `rstudio-connect` group with `0770` permissions.

Learn more about server-stored Shiny bookmarking state in this article.

The `apps` directory contains directories for each deployment. The top-level directory is owned by `root` with `0711` permissions. The first level of the `apps` hierarchy is a directory for each content deployment. These `apps/A_ID` directories are owned by `rstudio-connect` with `0700` permissions.

Beneath each `apps/A_ID` directory is a set of directories for each deployed bundle. The ownership and permissions for this hierarchy depend on whether or not the content is configured with a custom `RunAs` setting. Without a custom `RunAs` setting, permissions are simple: owned by `rstudio-connect` with directories having `0700` and files having `0600` permissions.

Learn more about using a custom `RunAs` in Section 12.3.

RStudio Connect needs a more complicated permission structure when content is configured with a custom `RunAs` setting. This is because the `rstudio-connect` user (`Applications.RunAs`) is used to install the necessary packages while the content-specific custom `RunAs` is used when running the deployed R code. The `apps/A_ID/B_ID` directory is owned by the custom `RunAs` with group ownership set to `rstudio-connect`. Permissions on this directory are `0750`. The `packrat` subdirectory is owned by `rstudio-connect` with group ownership of `rstudio-connect`. File permissions on this directory and its sub-directories are `0750` while files have `0640` permissions. Other than the `packrat` directory, all files underneath `apps/A_ID/B_ID` have `0600` permissions and directories are given `0700`.

All other data subdirectories are owned by `root` with `0700` permissions.

4.7 Backups

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

A running RStudio Connect server may be writing into the data directory if there are any active deployments, applications or documents. You should stop the RStudio Connect server before taking a backup.

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

Your platform may need alternate commands to restart RStudio Connect. Please see Section 5.1 for instructions specific to your operating system version.

5 Server Management

This section describes common administrative tasks for RStudio Connect.

5.1 Stopping and Starting

Occasionally it is necessary to start and stop the RStudio Connect service. Stopping and starting is handled by `systemd` or Upstart. On stop/start or restart the following occurs:

Stop:

- The RStudio Connect process is stopped.
- R processes serving Shiny applications and Plumber APIs are stopped.
- R processes rendering R Markdown documents run through completion.
- In-progress deployments will fail. R processes running as part of the deployment may run to completion.

Start:

- RStudio Connect process is resumed.
- Shiny applications and Plumber APIs with a minimum number of R processes are started.
- Scheduled R Markdown updates missed during system downtime are run at most once.

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

5.1.1 systemd (Red Hat/CentOS 7, Ubuntu 16.04)

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

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

```
$ sudo systemctl start rstudio-connect
```

```
$ sudo systemctl stop rstudio-connect
```

You can restart the server with:

```
$ sudo systemctl restart rstudio-connect
```

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-connect
```

This 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 Connect when changing other properties.

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

```
$ sudo systemctl status rstudio-connect
```

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

```
$ sudo systemctl enable rstudio-connect
```

```
$ sudo systemctl disable rstudio-connect
```

5.1.2 Upstart (Ubuntu 12.04, Ubuntu 14.04, Red Hat 6)

Upstart is a system used to automatically start, stop and manage services. The installer writes an Upstart configuration file to `/etc/init/rstudio-connect.conf`. This instructs the Upstart to initialize RStudio Connect 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 `connect` process is respawned if the process unexpectedly terminates. However, in the event that there is an issue which consistently prevents RStudio Connect 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 Connect startup attempt before it transitions to the "stopped" state.

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

```
$ sudo start rstudio-connect
```

```
$ sudo stop rstudio-connect
```

To restart the server you can run:

```
$ sudo stop rstudio-connect
$ sudo start rstudio-connect
```

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 at`/etc/init/rstudio-connect.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-connect
```

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 Connect when changing other properties.

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

```
$ sudo status rstudio-connect
```

5.2 System Messages

Administrators can add a message to the RStudio Connect welcome page and content page.

Messages are set in the `/etc/rstudio-connect/rstudio-connect.gcfg` file. `Server.PublicWarning` defines the message for the welcome page. `Server.LoggedInWarning` defines the message for the content page. The messages are supplied as HTML snippets. For example:

```
[Server]
PublicWarning = "<strong>Warning:</strong> Scheduled downtime this weekend."
LoggedInWarning = "Data Science Team Meeting Tomorrow"
```

Messages can be added or modified without restarting the connect service. After adding the message property to the config file, use the reload commands for either systemd (Red Hat/CentOS 7, Ubuntu 16.04):

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

or Upstart (Ubuntu 12.04, Ubuntu 14.04, Red Hat 6):

```
sudo reload rstudio-connect
```

5.3 Health-Check

RStudio Connect provides a simple health-check endpoint that can be used to test if Connect is up/listening. Point your browser to `myserveraddress:myserverport/__ping__`, which returns an empty JSON response and an HTTP 200 status.

```
curl -I -X GET http://myserveraddress:myserverport/__ping__
```

5.4 Upgrading

Upgrading RStudio Connect requires limited downtime. Scheduled R Markdown documents are not interrupted. Connections to running Shiny applications and Plumber APIs are closed. We recommend upgrading during a period of downtime. Users can be warned ahead of an upgrade with system messages.

The RStudio Connect version number is visible on the lefthand navigation pane. The latest version is available on the download page along with release notes.

To upgrade:

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

   Ubuntu:

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

   Red Hat/CentOS:

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

The new version of RStudio Connect will install on top of an earlier installation. Existing configuration settings are respected. During installation the RStudio Connect service is restarted. Total downtime is less than 10 minutes.

5.5 Purging RStudio Connect

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

1. Stop the RStudio Connect service. (See 5.1 for details)
2. Uninstall the RStudio Connect package from your system.

   Ubuntu:

   ```
   sudo apt-get purge rstudio-connect
   ```

   Red Hat/CentOS:

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

6 High Availability and Load Balancing (Experimental)

Multiple instances of RStudio Connect 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.

Using Connect in a HA configuration is currently considered experimental. Please report any issues to support@rstudio.com.

6.1 HA Checklist

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

1. Install and Configure the same version of RStudio Connect on each node - 2
2. Migrate to a PostgreSQL database (if running SQLite) - 9.1. All nodes in the cluster must use the same PostgreSQL database.
3. Configure each server's `Server.DataDir` to point to the same shared location - 4.6 and 6.2.3
4. Configure each server's `Server.LandingDir` to point to the same shared location (if using a custom landing page) - C and 6.2.3
5. Configure each server's `Metrics.DataPath` directory to point to a unique-per-server location - A.18. Alternatively, you may also wish to consider using Graphite to write all metrics to a single location - 6.2.4
6. Update each server's configuration with `LoadBalancing.EnforceMinRsconnectVersion = true` to ensure that your clients use a compatible version of `rsconnect` - 6.2.6

7. Configure your load balancer to route traffic to your RStudio Connect nodes with sticky sessions - 6.2.6

6.2 HA Limitations

6.2.1 Node Management

RStudio Connect nodes in a 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

6.2.2 Database Requirements

RStudio Connect only supports HA when using a PostgreSQL database. If you are using SQLite, please switch to PostgreSQL. See 9.1.

6.2.3 Shared Data Directory Requirements

RStudio Connect manages uploaded 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 4.6 for more information on the server's data directory. We recommend and support NFS version 3 for file sharing.

6.2.4 Metrics Requirements

By default, RStudio Connect writes metrics to a set of RRD files. We do not support metrics aggregation, and each server must maintain a separate set of RRD files to avoid conflicts. The admin dashboard for a specific node will only show metrics for that node. See A.18 for information on configuring a unique `Metrics.DataPath` for each server

RStudio Connect includes optional support for writing metrics to Graphite. If you wish to aggregate metrics, consider using Graphite or any monitoring tool compatible with Carbon protocol. See 16 for more information.

6.2.5 Shiny Applications

Shiny applications depend on a persistent connection to a single server. Please configure your load-balancer to use cookie-based sticky sessions to ensure that Shiny applications function properly when using HA.

6.2.6 rsconnect Cookie Support

For cookie-based sticky session support, you will need to ensure that your clients use `rsconnect` version 0.8.3 or later. Versions of `rsconnect` prior to 0.8.3 did not include support for cookies. Please update each server's configuration with the `LoadBalancing.EnforceMinRsconnectVersion = true` setting to ensure that clients must use a version of `rsconnect` with cookie support.

If you cannot enforce a minimum `rsconnect` version, you can consider alternatives like:

- Non-cookie-based sticky sessions, or
- Providing a separate host name for deployment from `rsconnect` to a single node in the cluster. Content deployed to a specific node will be available to the cluster assuming the database and shared storage are appropriately configured.

6.3 Updating HA Nodes

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

1. Stop all RStudio Connect nodes in your cluster.
2. Upgrade one RStudio Connect node. The first update will upgrade the database schema (if necessary) and start RStudio Connect on that instance - 5.4.
3. Upgrade the remaining nodes.

If you forget to stop any RStudio Connect 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.

6.4 Downgrading

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

1. Stop all Connect 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 - 4.6
    2. If using a custom landing page, configure the server's `Server.LandingDir` to point to a location on the node, and copy the custom landing page data from the NFS share to this location - C
    3. Configure the server's `Metrics.DataPath` directory to point to an appropriate location. If necessary, copy the data from the NFS share to this location. - 6.2.4
4. If you wish to move the database to this node, install PostgreSQL on the node and copy the data. Moving the PostgreSQL database from one server to another is beyond the scope of this guide. Please note that we do not support migrating from PostgreSQL back to SQLite.
5. Start the Connect process 5.1

6.5 HA Details

6.5.1 Concurrent Scheduled Document Rendering

The `Applications.ScheduleConcurrency` configuration setting specifies the number of scheduled jobs that can run concurrently on a node. This setting defaults to `2` and can be adjusted to suit your needs. This setting will not affect ad-hoc rendering requests, hosted APIs, or Shiny applications.

6.5.2 Concurrent Shiny Applications and Ad-Hoc Rendering

Each R process associated with Shiny applications, hosted APIs, ad-hoc rendering requests, and bundle deployments runs on the server where the request was initiated. We depend on your load balancer to distribute these requests to an appropriate Connect node. The minimum and maximum process limits for Shiny applications are enforced per server. For example, if a Shiny application allows a maximum of 10 processes, a maximum of 10 process per server will be enforced. See A.16 for more information.

6.5.3 Polling

RStudio Connect nodes poll the data directory for new scheduled jobs:

- Every 5 seconds, and
- After every completed scheduled job.

6.5.4 Abandoned R Processes

While processing a scheduled job, the RStudio Connect node periodically updates the job's metadata in the database with a "heartbeat". If the node goes offline and the "heartbeat" ceases, another node will eventually claim the abandoned job and run it again. Hence, if a server goes offline or the Connect process gets shut down while a scheduled report is running, it is possible that the scheduled job could run twice.

6.5.5 Abandoned Shiny Applications

A Shiny applications depends on a persistent connection to a single server. If the server associated with a particular Shiny application session goes down, the Shiny application will fail. However, simply refreshing the application should result in a new session on an available server, assuming your load balancer detects the failed node and points you to a working one.

Shiny applications that support client-side reconnects using the `session$allowReconnect(TRUE)` feature will automatically reconnect the Shiny application to a working node. See https://shiny.rstudio.com/articles/reconnecting.html

7 Running with a Proxy

If you are running RStudio Connect 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 Connect. This section describes how to correctly configure a reverse proxy with Nginx.

When RStudio Connect is behind a proxy, it is important that we send the original request URL information to Connect so that it can generate FQDN URLs and return them the requester. For this reason, when proxying to Connect, we recommend adding a header, `X-RSC-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://connect.company.com/some/path`)

Some proxies (like Amazon Web Services Elastic Load Balancer, for example), 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-RSC-Request`. If both `X-RSC-Request` and `X-Forwarded` headers are supplied, `X-RSC-Request` takes precedence.

If your proxy secures traffic with SSL, but uses an insecure (HTTP) connection to Connect, Connect Dashboard users will see a warning about accessing Connect over an insecure connection. You can disable this warning by using the `Http.NoWarning = true` configuration setting. See A.2.

7.1 Nginx Configuration

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

```
sudo apt-get install nginx
```

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

```
sudo yum install nginx
```

To enable an instance of Nginx running on the same server to act as a front-end proxy to RStudio Connect you would add commands like the following to your `nginx.conf` file. This configuration assumes RStudio Connect is running on the same host as Nginx and listening for HTTP requests on the `:3939` port. If you are proxying to RStudio Connect on a different machine or port, replace the `localhost:3939` references with the correct address of the server where RStudio Connect is hosted.

```
http {
    map $http_upgrade $connection_upgrade {
        default upgrade;
        ''      close;
    }
    server {
        listen 80;

        client_max_body_size 0; # Disables checking of client request body size

        location / {
            proxy_set_header    X-RSC-Request $scheme://$host:$server_port$request_uri;
            proxy_pass http://localhost:3939;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection $connection_upgrade;
            proxy_http_version 1.1;
            proxy_buffering off; # Required for XHR-streaming
        }
    }
}
```

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

```
http {
    map $http_upgrade $connection_upgrade {
        default upgrade;
        ''      close;
    }
    server {
        listen 80;

        client_max_body_size 0; # Disables checking of client request body size

        location /rsconnect/ {
            rewrite ^/rsconnect/(.*)$ /$1 break;
            proxy_set_header    X-RSC-Request $scheme://$host:$server_port$request_uri;
            proxy_pass http://localhost:3939;
            proxy_redirect / /rsconnect/;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection $connection_upgrade;
            proxy_http_version 1.1;
        }
    }
}
```

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

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

```
sudo systemctl restart nginx
```

On `upstart` systems (Ubuntu 12.04, Ubuntu 14.04, Red Hat 6):

```
sudo restart nginx
```

7.2 Apache Configuration

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

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

The following configuration will permit proxying to RStudio Connect 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-RSC-Request "%{REQUEST_SCHEME}s://%{HTTP_HOST}s%{REQUEST_URI}s"
    RewriteCond %{HTTP:Upgrade} =websocket
    RewriteRule /(.*) ws://172.17.0.1:3939/$1 [P,L]
    RewriteCond %{HTTP:Upgrade} !=websocket
    RewriteRule /(.*) http://172.17.0.1:3939/$1 [P,L]
    ProxyPass / http://172.17.0.1:3939/
    ProxyPassReverse / http://172.17.0.1:3939/
</VirtualHost>
```

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

```
Listen 3737

<VirtualHost *:3737>
    RewriteEngine on
    RedirectMatch ^/rsconnect$ /rsconnect/
    RequestHeader set X-RSC-Request "%{REQUEST_SCHEME}s://%{HTTP_HOST}s%{REQUEST_URI}s"
    RewriteCond %{HTTP:Upgrade} =websocket
    RewriteRule /rsconnect/(.*) ws://172.17.0.1:3939/$1 [P,L]
    RewriteCond %{HTTP:Upgrade} !=websocket
    RewriteRule /rsconnect/(.*) http://172.17.0.1:3939/$1 [P,L]
    ProxyPass /rsconnect/ http://172.17.0.1:3939/
    ProxyPassReverse /rsconnect/ http://172.17.0.1:3939/
    Header edit Location ^/ /rsconnect/
</VirtualHost>
```

8 Security & Auditing

8.1 Browser Security

There are a variety of security settings that can be configured in RStudio Connect. 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.

8.1.1 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 Connect 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 Connect cookies to a server without an HTTPS-secured connection.

8.1.2 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.

8.1.3 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.

For the purposes of the `X-Frame-Options` header, RStudio Connect distinguishes between "dashboard" and "user" content. Dashboard content are any of the internal services or assets that are shipped with RStudio Connect. User content is anything uploaded by a user (reports, Shiny applications, Plumber APIs, etc.)

`Server.FrameOptionsContent` configures the `X-Frame-Options` header value for user-uploaded content. By default it is empty, meaning that the header will not be set. This allows user-provided content to be embedded in iframes from any location. If you do not intend for others to embed user content on their sites, you can set this to a value of `SAMEORIGIN` to ensure that only sites on the same server will be able to embed your users' content. The RStudio Connect dashboard itself uses iframes to present user content in the dashboard, so it is not recommended to set this option to `DENY`.

`Server.FrameOptionsDashboard` configures the `X-Frame-Options` header value for internal services and assets provided with RStudio Connect and defaults to a value of `DENY`. This means that other sites will not be able to embed the RStudio Connect dashboard. This setting is more secure in that it protects against clickjacking attacks against the dashboard, but if you plan to embed the dashboard elsewhere you may need to tune this setting.

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

8.1.4 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 Connect 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 Connect. 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 Connect, by Plumber, or by Shiny. You should not depend on a custom header that conflicts with a header already in use by RStudio Connect.

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"
```

8.2 Audit Logs

The following events are logged by the auditing system:

Event	Description
`add_user`	Create a user
`edit_user`	Change an existing user
`update_lock_user`	Set or remove a lock for an existing user
`add_application`	Add new content
`upload_bundle`	Upload a bundle for a content
`deploy_application`	Deploy content to the server. Note that content still needs to published after deployment.
`edit_application`	Change content settings
`remove_application`	Delete content
`activate_token`	Activate a token. Tokens are used by the `rsconnect` package to authenticate a user.
`add_group`	Create a group
`remove_group`	Delete a group
`add_group_member`	Add a user to a group
`remove_group_member`	Remove a user from a group
`assign_user_app_role`	Give a user view or edit access to content
`remove_user_app_role`	Remove a user from view or edit access list
`assign_group_app_role`	Give a group view or edit access to content
`remove_group_app_role`	Remove a group from view or edit access list
`clear_app_viewer_acl`	Change from a specific list of viewers to “just me”
`add_api_key`	Added API key
`remove_api_key`	Removed API key
`add_vanity`	Add vanity url
`update_vanity`	Update vanity url
`remove_vantiy`	Remove vanity url

Event	Description
`remove_bundle`	Remove a bundle
`download_bundle`	Download a bundle
`add_tag`	Create a tag/category
`remove_tag`	Delete a tag/category
`update_tag`	Update a tag/category
`assign_tag_to_parent`	Associate a tag with some parent tag/category
`add_app_tag`	Associate a tag with content
`remove_app_tag`	Disassociate a tag with content

8.3 Audit Logs Command-Line Interface

See Appendix B for more information on using the `usermanager` CLI to dump audit logs.

9 Database

RStudio Connect 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.5 for details

Here is a partial configuration which chooses to use SQLite

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

9.1 Changing Database Provider

Connect includes a `migrate` command for migrating data from one database to another.

The migration utility is installed at `/opt/rstudio-connect/bin/migrate`. It uses the configuration defined in `/etc/rstudio-connect/rstudio-connect.gcfg` unless you specify an alternate configuration file with the `--config` flag.

The `migrate` utility must be run as `root`.

The `migrate` utility can only be run when Connect is stopped. See Section 5.1 for information on stopping and restarting Connect.

Note: Migration from PostgreSQL to SQLite is not supported at this time.

9.1.1 Migration Checklist

Use this checklist to guide your migration process:

1. Shut down Connect - 5.1
2. Back up your data - 4.7
3. Ensure that you have a `[Postgres]` configuration section - 9.3
4. Run the migration
5. Update the `Database.Provider` configuration setting to point to the new database - A.5
6. Restart Connect - 5.1

9.1.2 Commands

The `migrate` utility supports two commands

- `db`: Migrate data between databases
- `help`: Displays help

9.1.3 Flags

Configuration for `migrate`:

- `--config`: The full or relative path to a Connect configuration file (`.gcfg`). Defaults to `/etc/rstudio-connect/rstudio-connect.gcfg`.

Flags for the `migrate db` command:

- `--verify`: Verify migration only.
- `--drop-all`: Drop all existing data in the target before migrating.

By default, the `migrate db` command will perform a full migration from SQLite to PostgreSQL and verify the migration. We assume that the PostgreSQL destination does not contain any data unless the `--drop-all` flag is included.

Data migration copies data from the SQLite database to the PostgreSQL database. Data in the SQLite database remains after the migration; it is not removed. A verification step runs after the data copy completes and confirms the integrity of the migration:

- Row counts for all tables are verified.
- Each record is checked for the correct values.

Data verification will fail if Connect is started prior to the completion of data verification. Please ensure that Connect remains down until the data migration and verification are complete.

9.1.4 Configuration Requirements

When migrating data, the configuration file must contain valid configuration sections for both `[SQLite]` and `[Postgres]`. The migration utility will connect to the SQLite and PostgreSQL databases specified in the configuration.

9.1.5 Examples:

Display help:

```
/opt/rstudio-connect/bin/migrate help
```

Migrate SQLite data to an empty PostgreSQL database:

```
sudo /opt/rstudio-connect/bin/migrate db
```

Migrate SQLite data to a PostgreSQL database, first dropping all data in the PostgreSQL database:

```
sudo /opt/rstudio-connect/bin/migrate db --drop-all
```

Perform data verification only:

```
sudo /opt/rstudio-connect/bin/migrate db --verify
```

Specify a custom configuration file:

```
sudo /opt/rstudio-connect/bin/migrate --config /etc/connect/mycustomconfig.gcfg db
```

9.2 SQLite

SQLite is the default database provider.

RStudio Connect 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/connect/sqlite
```

If this field is not specified, it will default to `{Server.DataDir}/db`.

9.3 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 box from your RStudio Connect 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 Connect, 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.

```
[Database]
Provider = postgres

[Postgres]
URL = "postgres://username:password@db.seed.co/connect"
```

10 Authentication

RStudio Connect supports a variety of user authentication options. Without customization, a locally-backed password scheme is used. You can learn more about password authentication in Section 10.3.

When signing into RStudio Connect, a session cookie is used to keep a user logged in for 30 days. The lifetime of these sessions can be altered using the Authentication.Lifetime setting.

External authentication is available through the following integrations:

- LDAP and Active Directory (Section 10.4)
- OAuth 2.0 using Google Apps accounts (Section 10.5)
- PAM (Section 10.6)
- Proxied Authentication (Section 10.7)

Customize the `Authentication.Provider` property with an authentication scheme appropriate for your organization. See Section A.6 for details

Here is a partial configuration which chooses to use LDAP.

```
[Authentication]
Provider = ldap
```

10.1 Changing Authentication Provider

Migrating from one authentication provider to another (for example, switching from password to LDAP) is **NOT SUPPORTED**. If changing the style of authentication is absolutely necessary, you will need to completely purge and reinstall RStudio Connect. See Section 5.5 for instructions.

10.2 Username requirements

When using OAuth or Password authentication, users are required to choose a valid username when first logging in. Usernames must:

- be 3-64 characters in length,
- start with a letter, and
- contain only alphanumeric characters, underscores, and periods.

The LDAP, PAM, and Proxy authentication providers require by default that a valid username is received from the provider. If a valid username is not received from the provider, an error will be thrown. If you wish for less restrictive behavior where the user is prompted for a valid username, please use the `RequireExternalUsernames = false` configuration setting for your auth provider. When using LDAP, the `LDAP.UsernameAttribute` configuration setting specifies how Connect can find the username. When using PAM and Proxy authentication, Connect receives the exact username specified during login.

See A.9, A.10, and A.11 for usage of the `RequireExternalUsernames` setting.

The LDAP, PAM, and Proxy authentication providers use relaxed username requirements. These providers accept any username, excepting a list of blacklisted usernames. The list of blacklisted usernames follows:

- connect
- apps
- users
- groups
- setpassword
- user-completion
- confirm
- recent
- reports
- plots
- unpublished
- settings
- metrics
- tokens
- help
- login
- welcome
- register
- resetpassword
- content

10.3 Password

Password authentication is the default authentication provider used by RStudio Connect. This is a local user account backed by the RStudio Connect database and is not integrated with a third-party service.

Users will be able to create accounts when they first visit the system and will provide profile details at that time. An administrator will also be able to create new accounts.

Password authentication may be appropriate in small organizations without centralized IT systems.

RStudio Connect will use password authentication if the `Authentication.Provider` setting has a value of `password` or if `Provider` is not present in the configuration file.

```
[Authentication]
Provider = password
```

10.4 LDAP and Active Directory

RStudio Connect can integrate with your company's LDAP or Active Directory (AD) infrastructure. User authentication and user search requests will be directed to the LDAP/AD server.

LDAP and Active Directory support in RStudio Connect has the following constraints:

- Your LDAP/AD user objects must contain a user's first name, last name, email address, and username.
- Changes to a user (e.g. their name, email address, or username) will not propagate to RStudio Connect once the user is created internally.
- When using single bind, the DN of a user must contain their username (i.e. must utilize the `UsernameAttribute`). For example, it is not supported if the DN for a user is `cn=SueJacobs,ou=People,dc=company,dc=com` but their actual username is stored in the `uid` or `SAMAccountName` LDAP attribute. You must use double bind when the DN does not contain the username.
- When using a single bind configuration, searches will only include users who have previously logged into RStudio Connect.
- When using a single bind configuration, groups will be unavailable.
- A username or DN containing a forward slash (`/`) is not supported.

When attempting to troubleshoot a problem relating to LDAP, you can enable more verbose logging by adding `ldap` to `Debug.Log` section in the configuration.

```
[Debug]
Log = ldap
```

10.4.1 Defining an LDAP or AD section

RStudio Connect does support the notion of having multiple LDAP or AD servers. This can be utilized by defining multiple LDAP sections.

To define an LDAP or AD section in the configuration file, add a header like the following:

```
[LDAP "European LDAP Server"]
...
```

An LDAP/AD configuration section header is always bounded by square brackets (`[]`). After the section type LDAP is the effective name of the LDAP or AD server (`"European LDAP Server"` in the example). Make sure that this text is unique per LDAP or AD section you configure. The LDAP section name is treated *case sensitively.*

RStudio Connect can support more than one LDAP server through multiple, uniquely named LDAP configuration sections. Other complex LDAP configurations can also be achieved by using multiple LDAP sections.

> If multiple LDAP sections have the same name, they will be combined as described in Appendix A. As this is unlikely your intent, please take care to give unique names to each LDAP configuration section.

Here is an sample configuration using two LDAP sections.

```
[LDAP "European LDAP Server"]
...

[LDAP "Statistics Department LDAP Server"]
...
```

Each of these sections will have a variety of configuration settings, which are explained below.

10.4.2 Complete Configuration Example

Here is a complete LDAP configuration as an example. Here, we are communicating with an OpenLDAP server on the local host; see the documentation for `ServerAddress` to learn how to direct requests elsewhere. The other settings will probably need adjustment for your environment. Talk to your LDAP administrator if you need help with your organization's LDAP hierarchy.

```
[LDAP "Sample OpenLDAP Configuration"]
ServerAddress = 127.0.0.1:389
BindDN = "cn=admin,dc=example-openldap"
BindPassword = "XXXXXXXX"
UserSearchBaseDN = "ou=People,dc=example-openldap"
UsernameAttribute = "uid"
UserObjectClass = "posixAccount"
UserEmailAttribute = mail
UserFirstNameAttribute = givenName
UserLastNameAttribute = sn
```

This sample configuration assumed a very simple OpenLDAP structure; here is a sample user record to show the mapping between LDAP records and RStudio Connect LDAP configuration.

```
dn: uid=john,ou=People,dc=example-openldap
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: john
sn: Doe
givenName: John
cn: John Doe
displayName: John Doe
uidNumber: 10000
gidNumber: 5000
userPassword: johnldap
gecos: John Doe
loginShell: /bin/bash
homeDirectory: /home/john
mail: john@example.com
```

More LDAP configuration scenarios can be found in Appendix D.

10.4.3 LDAP or AD Configuration Settings

10.4.3.1 `ServerAddress`

`ServerAddress` (required) is used to define the location of the LDAP/AD server. This should contain an IP address or DNS address, and a port (colon separated). Most LDAP/AD servers operate on port `389` or `636` (for SSL). But you can specify any port that fits your environment.

Examples

```
ServerAddress = 127.0.0.1:389
ServerAddress = ldap.company.com:389
ServerAddress = ldaps.company.com:636
ServerAddress = private.internal.local:7554
```

10.4.3.2 `TLS`

`TLS` is a Boolean (true/false) attribute that causes all connections to your LDAP/AD server to use TLS (SSL). The default value for this is `false`. This cannot be enabled if `StartTLS` is `true`.

Examples

```
TLS = true
TLS = false
```

10.4.3.3 `StartTLS`

`StartTLS` is a Boolean (true/false) attribute that causes connections to your LDAP/AD server to initially use an unencrypted channel but then upgrade to a TLS connection using "Opportunistic TLS". The default value for this is `false`. This cannot be enabled if `TLS` is `true`.

Examples

```
StartTLS = true
StartTLS = false
```

At present, the error messages associated with StartTLS problems can be cryptic. If you're encountering issues while configuring StartTLS, consider adding debug logging for LDAP by including the following line in your configuration file.

```
[Debug]
Log = ldap
```

10.4.3.4 `TLSCACertificate`

`TLSCACertificate` is a file location that is a certificate authority that is used to connect to an LDAP server securely. This file should be in PEM format.

Examples

```
TLSCACertificate= /etc/ssl/cert/ca.pem
```

10.4.3.5 `ServerTLSInsecure`

`ServerTLSInsecure` is a Boolean (true/false) attribute that allows insecure TLS connections. This controls whether a client will verify the server's certficate chain and host name. If this is `true`, RStudio Connect will accept any certificate presented by the server and any host name in that certificate. Setting to `true`

is susceptible to main-in-the-middle attacks, but is required if, for example, your server uses a self-signed certificate. The default value is `false`.

Examples

```
ServerTLSInsecure = true
ServerTLSInsecure = false
```

10.4.3.6 `BindDN` and `BindPassword`

`BindDN` and `BindPassword` are credentials used to connect to an LDAP/AD server to authenticate, search for users, and other functionality. While it is encouraged to specify these two attributes (a.k.a. "double bind"), it is not required (a.k.a. "single bind"). These credentials should have read-only administrator's rights, if configured.

If you do not specify these attributes, some functionality of RStudio Connect will not work. For example, searching for users to add as collaborators, or sending email documents will only work partially.

The `BindDN` can be a DN, UPN, or NT-style login.

Examples

```
# Example DN
BindDN = uid=john,ou=People,dc=company,dc=com
BindPassword = johnpassword

# Example UPN
BindDN = admin@company.com
BindPassword = adminpassword

# Example NT-style login
BindDN = COMPANY\\admin  # we use double slashes (\\) to character escape the last slash
BindPassword = adminpassword
```

10.4.3.7 `AnonymousBind`

`AnonymousBind` instructs RStudio Connect to establish an anonymous bind to your LDAP/AD server. For organizations that support anonymous binds, you may use this option instead of `BindDN` and `BindPassword`.

For this to work properly, your LDAP server must allow anonymous binds to search and view all pertinent groups, group memberships, and users.

Examples

```
AnonymousBind = true
```

10.4.3.8 `UserSearchBaseDN`

`UserSearchBaseDN` (required) is the starting point from which RStudio Connect will search for user entries in your LDAP/AD server.

Examples

```
UserSearchBaseDN = dc=company,dc=com
UserSearchBaseDN = ou=People,dc=company,dc=com
```

10.4.3.9 `UserObjectClass`

`UserObjectClass` (required) is the `objectClass` that a user in your LDAP/AD structure will have. Common examples of this are `user`, `posixAccount`, `organizationalPerson`, `person`, and `inetOrgPerson`.

Examples

```
UserObjectClass = user
UserObjectClass = posixAccount
```

10.4.3.10 `UsernameAttribute`

`UsernameAttribute` (required, case-sensitive) is the LDAP entry attribute that contains the username of a user.

Examples

```
UsernameAttribute = uid
UsernameAttribute = sAMAccountName
```

10.4.3.11 `UserFirstNameAttribute`

`UserFirstNameAttribute` (required, case-sensitive) is the LDAP entry attribute that contains the first name of a user.

Examples

```
UserFirstNameAttribute = givenName
```

10.4.3.12 `UserLastNameAttribute`

`UserLastNameAttribute` (required, case-sensitive) is the LDAP entry attribute that contains the last name of a user.

Examples

```
UserLastNameAttribute = sn
```

10.4.3.13 `UserEmailAttribute`

`UserEmailAttribute` (required, case-sensitive) is the LDAP entry attribute that contains the email address of a user.

Examples

```
UserEmailAttribute = mail
```

10.4.3.14 `WhitelistedLoginGroup`

`WhitelistedLoginGroup` defines a group DN that a user must be a member of in order to login into Connect. You can specify this attribute multiple times. Be aware that this feature restricts *only* the ability for users to login. Users not in this group could still be referenced when setting access controls for content or as email recipients. Because the users could not login, they would not be able to access content even if they were added as a viewer or collaborator, but they might still be able to receive emailed versions of reports.

Examples

```
WhitelistedLoginGroup = cn=admins,ou=group,dc=company,dc=com
WhitelistedLoginGroup = cn=scientists,ou=group,dc=company,dc=com
```

10.4.3.15 `GroupObjectClass`

`GroupUserObjectClass` is the `objectClass` that a group in your LDAP/AD structure will have. Common examples of this are `group`, and `posixGroup`.

Examples

```
GroupObjectClass = group
GroupObjectClass = posixGroup
```

10.4.3.16 `GroupNameAttribute`

`GroupNameAttribute` (case-sensitive) is the LDAP entry attribute that contains the name of a group.

Examples

```
GroupNameAttribute = cn
GroupNameAttribute = sAMAccountName
```

10.4.3.17 `GroupSearchBaseDN`

`GroupSearchBaseDN` is the starting point from which RStudio Connect will search for group entries in your LDAP/AD server.

Examples

```
GroupSearchBaseDN = dc=company,dc=com
GroupSearchBaseDN = ou=Groups,dc=company,dc=com
```

10.5 OAuth2 (Google)

OAuth2 authentication is available to authenticate against the Google OAuth2 service.

RStudio Connect will use OAuth2 authentication if the `Authentication.Provider` setting has a value of `oauth2`.

```
[Authentication]
Provider = oauth2
```

Appendix A.8 contains information about each OAuth2 configuration option.

In order for RStudio Connect to use Google as an OAuth2 service, you will need a client ID and client secret.

10.5.1 Obtaining a Client ID and Client Secret

These instructions tell you how to obtain an OAuth2 client ID and client secret. We recommend a distinct set of credentials for each application you configure to use the Google OAuth2 service.

1. Visit the Google Developers Console and create a new project. Give it a name of your choosing, such as "rstudio-connect".
2. Once the project is created, locate and enable the "Google+ API".
3. In the left navigation window, click on "Credentials", then goto the "OAuth consent screen" tab, fill in the information requested and click "Save".
4. Once again, click "Credentials" in the left navigation window. Then click the dropdown button "New credentials", then "OAuth client ID".

5. For "Application Type", select "Web Application". Then give your client ID a descriptive name. For "Authorized JavaScript origins", enter your RStudio Server URL (i.e. `https://HOST:PORT`). For "Authorized redirect URIs", use your RStudio Connect server address with `/__login__/callback` (i.e. `https://HOST:PORT/__login__/callback`).
6. Click "Create". Your client ID and client secret will be shown to you.

Add the client ID and secret to your configuration file as shown in the example below.

```
[OAuth2]
DiscoveryEndpoint = https://accounts.google.com/.well-known/openid-configuration
ClientId = <CLIENT ID>
ClientSecret = <CLIENT SECRET>
```

With `DiscoveryEndpoint`, `ClientId` and either `ClientSecret` or `ClientSecretFile` configured, you can use your Google Apps account to sign into RStudio Connect!

10.5.2 Restricting Access

The default configuration allows all Google account holders to access RStudio Connect. We recommend that you limit access to specific domains that are used by your organization.

> Verify that you can use your Google Apps account to sign into RStudio Connect before attempting to configure access restrictions.

The `OAuth2.AllowedDomains` setting specifies the set of domains that are allowed to access your RStudio Connect server. Multiple domains should be space-separated.

```
[OAuth2]
AllowedDomains = company.com subsidiary.com
```

You may also restrict access by email address if using domain alone is insufficient. The `OAuth2.AllowedEmails` setting specifies the set of email addresses that are allowed to access your RStudio Connect server. Multiple addresses should be space-separated.

```
[OAuth2]
AllowedEmails = jdoe@company.com asmith@subsidiary.com
```

It is important to understand how the `AllowedDomains` and `AllowedEmails` properties interact.

If only `AllowedDomains` is configured, *only* email addresses with a listed domain will be permitted access.

If only `AllowedEmails` is configured, *only* listed email addresses will be permitted access.

When both `AllowedDomains` and `AllowedEmails` are specified, email addresses given in `AllowedEmails` are permitted access *in addition* to email addresses with a domain listed in `AllowedDomains`.

10.5.3 Searches

RStudio Connect allows users to search for collaborators against the user directory associated with your Google Apps account. That search is performed on behalf of the current user. Different accounts may have different visibility within the user directory and therefore will see different results. This is most obvious when you have configured RStudio Connect to allow access to two different domains. Users in `company.com`, for example, will likely not be able to search for colleagues in `subsidiary.com`.

RStudio Connect augments the Google Apps user directory search with a local search across its set of known accounts. Once your colleague has created their own RStudio Connect account, they will become discoverable.

10.6 PAM

RStudio Connect can use PAM for user authentication. PAM authentication is used if the `Authentication.Provider` setting has a value of `pam`.

```
[Authentication]
Provider = pam
```

See Section 12.5 for information about using PAM sessions when launching R processes.

You can change the PAM service name used for authentication by customizing the `PAM.Service` setting. The default PAM service name used for authentication is `rstudio-connect`.

```
[PAM]
Service = rstudio-connect
```

We assume that RStudio Connect is configured to use the `rstudio-connect` PAM service name in the examples that follow.

10.6.1 Ubuntu

RStudio Connect does not create a PAM service on Ubuntu systems. When RStudio Connect attempts to use the `rstudio-connect` service name for authentication, PAM will recognize that there is no service with that name and fall back to the default `other` service located at `/etc/pam.d/other`.

The default Ubuntu `other` service is configured to inherit from a set of common PAM services:

```
# Ubuntu default "other" PAM service.
@include common-auth
@include common-account
@include common-password
@include common-session
```

If the `other` service is appropriate for your organization, no further configuration is needed.

You need a custom `rstudio-connect` PAM service for RStudio Connect only if the `other` service is not fitting for your users. Create and configure `/etc/pam.d/rstudio-connect` to prevent PAM from falling back to the `other` service. PAM will use this service for subsequent authentication attempts using the `rstudio-connect` service name.

10.6.2 Red Hat/CentOS

Red Hat/CentOS systems deny access to unknown PAM service names by default. This is because the `other` configuration in `/etc/pam.d/other` contains only "deny" rules.

```
#%PAM-1.0
# The Red Hat/CentOS default "other" PAM service.
auth     required       pam_deny.so
account  required       pam_deny.so
password required       pam_deny.so
session  required       pam_deny.so
```

The RStudio Connect RPM installs an `rstudio-connect` PAM service at `/etc/pam.d/rstudio-connect`. This service is configured to require a user-id greater than 500 and authenticates against local system accounts.

```
#%PAM-1.0
# The RStudio Connect default PAM service.
auth      requisite      pam_succeed_if.so uid >= 500 quiet
```

```
auth      required       pam_unix.so nodelay
account   required       pam_unix.so
```

This default PAM service may not reflect the authentication behavior that you want for RStudio Connect. Feel free to customize this service for your organization.

10.6.3 Configuring a PAM service

This section may be helpful if your organization has different requirements from the default behavior of the `rstudio-connect` PAM service name. Please consult with your PAM/systems administrator to be sure that the RStudio Connect PAM service configuration fits your needs.

If your system already has a PAM service (e.g. `/etc/pam.d/login`) with the desired behavior, it may be enough to simply include that service from within the RStudio Connect service. For example:

```
# RStudio Connect PAM service that defers to the existing login service.
@include login
```

You could also copy that existing service into the RStudio Connect service, meaning the copy can be changed and evolve independently from the source service.

```
$ sudo cp /etc/pam.d/login /etc/pam.d/rstudio-connect
```

Lastly, you could configure the `PAM.Service` setting to reference that PAM service. This would be appropriate if you have a common `rstudio` service that you use across all the RStudio products, for example.

```
[PAM]
Service = rstudio
```

If you change the `PAM.Service` setting from its default `rstudio-connect` value, the PAM service defined in `/etc/pam.d/rstudio-connect` will not be used.

10.6.4 PAM Credential Caching

> Note: RStudio Connect's PAM cache is **encrypted** and **is not stored on disk**. The credentials **must expire** after a certain period of time.

RStudio Connect can be configured to securely cache a user's PAM credentials when they log in to RStudio Connect. This enables RStudio Connect to let users run R processes as their current UNIX account when the PAM profile requires a user's credentials.

The following config settings are required for credential caching to be enabled:

```
[Applications]
RunAsCurrentUser = true

[PAM]
UseSession = true  ; Enable PAM sessions
ForwardPassword = true ; Forward the current user's password into the PAM session
PasswordLifetime = 12h ; Cache passwords for 12 hours after login
```

Replace `12h` with the amount of time you would like credentials to be cached. Credential lifetime is counted from the moment the user logs into RStudio Connect. It is not tied to the user's web session, except that logging in again will restart the timer for that user's credentials.

10.6.5 Groups

Groups are not supported when using PAM authentication.

10.7 Proxied Authentication

RStudio Connect supports proxied authentication. This allows an external system to intercept requests and handle the authentication of users visiting the Connect dashboard or applications Connect is hosting.

10.7.1 How this Works

A service (like Apache, for example) runs as your customized authentication server. It is responsible for intercepting all requests to RStudio Connect and performing the required authentication and authorization. Requests from authenticated users will have a custom HTTP header added before the request is proxied through to RStudio Connect. That HTTP header contains the username of that visitor. RStudio Connect will take the value from the HTTP header and treat the current user as the username specified in the header.

We have no means of validating that this HTTP header was added by your authentication server and not by the user directly. So it is very important from a security perspective that the RStudio Connect server is properly firewalled off in your network and that all access to the Connect server is proxied through your authentication server.

> **Important Note**
>
> The username HTTP header should never be set by the requester. In all cases, your authentication server should delete that header if it exists before authenticating the user and adding the header itself.

RStudio Connect does not currently support directing users to a login page when using proxied authentication. Therefore, we recommend that your proxy prevent anonymous access to RStudio Connect; only allow authenticated users.

10.7.2 Deployment from the RStudio IDE

Deploying from the RStudio IDE is a unique situation. The IDE uses an R package `rsconnect` to obtain deployment credentials from RStudio Connect. Those credentials are used to sign deployment requests.

Deployment requests are signed with credentials obtained during an earlier, authenticated session, and should pass through your proxy without alteration.

The following three headers when used together identify deployment requests and should pass through your proxy without attempting to authenticate the user:

- `X-Auth-Token`
- `X-Auth-Signature`
- `X-Content-Checksum`

10.7.3 Configuring Proxied Authentication

To configure RStudio Connect to use proxied authentication, set `Authentication.Provider` to `proxy`.

```
[Authentication]
Provider = proxy
```

Proxied authentication requires that you set `Server.Address` to point at your proxy server. If you do not configure `Server.Address`, the browser may not have all its requests routed through your authenticating proxy. See Section 2.2.1 for more information about `Server.Address`.

```
[Server]
Address = https://myproxy.company.com/
```

You can customize the name of the header that your authentication server will send upon a successful authentication. By default, this key name is `X-Auth-Username`.

```
[ProxyAuth]
UsernameHeader = X-Auth-Username
```

10.7.4 Groups

Groups are not supported when using proxied authentication.

11 User Management

11.1 Self Registration

When using password authentication, users can self-register by clicking "Create a new account" on the login page. Self-registered accounts will be created with the role specified in the `DefaultUserRole` property (see 11.2).

If you wish to disable self-registration, please use the configuration setting `SelfRegistration = false` in the `Password` configuration section. See A.7 for more information on the `Password.SelfRegistration` setting.

When self-registration is disabled, the first account (the admin) is still created using self-registration. All other accounts must be created by an administrator.

11.2 User Roles

Every RStudio Connect user account is configured with a role that controls their default capabilities on the system. Data scientists, analysts and others working in R will most likely want "publisher" accounts. Other users are likely to need only "viewer" accounts.

The `DefaultUserRole` property within the `Authorization` configuration section specifies the role for new accounts and defaults to `viewer`. The `DefaultUserRole` may be either `viewer` or `publisher`; new accounts are not permitted to automatically have the `administrator` role.

Administrator RStudio Connect administrator accounts have permissions which allow them to manage the service. This includes setting the role of an account and configuring email settings. Administrators may or may not be system administrators. The specific capabilities of an administrator are documented here.

Publisher Accounts with a "publisher" role are allowed to deploy content into RStudio Connect. They can also help manage another user's content when made a "collaborator" of that content.

Viewer "Viewer" accounts can be added as a viewer to specific content. They can discover that content through the RStudio Connect dashboard and see its settings. Viewers can also email themselves copies of documents they are permitted to see.

Anonymous An anonymous visitor to RStudio Connect who is not authenticated with the system can view content that has been marked as viewable by "Everyone".

11.3 User Permissions

Administrators and Publishers can be assigned permissions for content published to RStudio Connect.

11.3.1 All Content

Anonymous Visitors Anonymous users can access content listed for `Everyone`. Anonymous viewers access content through direct URLs and will not have any view into Connect.

Viewers "Viewers" can sign into the Connect dashboard and discover and access content listed for `Everyone`, `All logged-in users`, and content for which they are granted access.

Collaborators "Collaborators" can change access controls and add Viewers and other Collaborators.

Administrators "Administrators" have all the permissions of Collaborators. Administrators are not automatically added to content and will not see all content on their homepage. Administrators can proactively add themselves as Collaborators or Viewers to any content. Administrators can set vanity URLs and change the `RunAs` user. Administrators and the original content owner can delete content.

11.3.2 R Markdown Reports

Access controls and user privileges apply to every public version of a report. For example, if the default version of a report is accessible to `Everyone`, all public versions will be accessible to `Everyone`.

Anonymous Visitors Every version of a report has a unique URL (accessible by opening the content with 'Open Solo'). Reports must be listed for `Everyone` for the URL to be available to anonymous users.

Viewers "Viewers" have the ability to view a report through the Connect dashboard. They can discover and toggle between public versions of a report. They can email themselves the current version of a report. They can not see parameters for different versions of a report. They can see the distribution and schedule for public versions.

Collaborators "Collaborators" have the privileges of Viewers and additionally can: view parameters for public versions, change parameters and run ad hoc reports, create new versions, schedule versions, setup distribution lists, and request reports to be refreshed. Collaborators can also create private versions that are not discoverable or accessible by any other user.

11.3.3 Shiny Applications & Plumber APIs

Note: Plumber APIs are currently in Beta.

Collaborators "Collaborators" can change the runtime settings for Shiny applications and Plumber APIs.

11.4 Administrator Capabilities

Administrative users on RStudio Connect are empowered to inspect and manage various settings on the server. Regardless of their level of privilege on some piece of content (viewer, collaborator, or neither), administrators can manage collaborators and viewers on content, manage the runtime settings for Shiny applications and Plumber APIs, and adjust the schedules for R Markdown documents. Additionally, only administrators can modify the Vanity Path and RunAs settings for content through the web dashboard; they can do so even on content that they don't have the ability to view the content.

Administrators do not have implicit rights to view content or download the source bundles. If an administrator visits a report without viewership privileges to the report, they will see an error message rather than the report's content. Despite being unable to see the contents of the report, administrators can still manage the

settings for all content. Because an administrator has the ability to manage the collaborators and viewers of others' content on the system, they can choose to add themselves as a viewer or collaborator on the report to gain access. Administrative overrides of permissions on content require that the administrator take an explicit action which is captured in the audit log.

11.5 Locked Accounts

You can prohibit a user from accessing RStudio Connect by "locking" their account. This control is available to administrative users when editing user profile information in the RStudio Connect dashboard.

Locked users are prohibited from signing into RStudio Connect, deploying content, and otherwise interacting with the service.

A locked account is not deleted and deployed content continues to be available. A non-personal report configured with scheduling and distribution will continue to execute according to its schedule. A locked user no longer receives scheduled content at their email address.

Content owned by a locked user can be deleted by a collaborator or by an administrative user. Each piece of deployed content must be deleted individually; there is no bulk removal.

A locked user can be subsequently unlocked. All their previously allowed abilities are immediately restored.

11.6 Username Requirements

Connect's username requirements vary depending upon the authentication provider. Please see 10.2 for more information on username requirements.

11.7 User Renaming

Administrators may alter the usernames of existing users on the system regardless of the current authentication system. Users will still be able to access their deployed content and content that has been shared with them. If they have existing vanity URLs with their username incorporated, none of those will be altered. They will, of course, need to use the new username when logging in.

If the user has authenticated inside of the RStudio IDE, they will still be able to deploy using a previous connection; however, the IDE will continue displaying their old username during deployments. To minimize the risk of future ambiguity, we recommend that the user disconnect and reconnect their IDE to RStudio Connect so that the valid username is displayed.

11.8 Command-Line Interface

Connect includes a `usermanager` command for some basic user management tasks. This utility helps you list users and modify user roles in the event that no one can access a Connect administrative user account.

See Appendix B for more information on using the `usermanager` CLI to manage users.

12 Process Management

RStudio Connect launches R to perform a variety of tasks. This includes:

- Installation of R packages
- Rendering of R Markdown documents

- Running Shiny Applications
- Running a Shiny application to customize a parameterized R Markdown document.
- Running APIs using Plumber (Beta)

The location of R defaults to whatever is in the path. Customize the `Server.RVersion` setting to use a specific R installation. See Chapter 14 for details.

12.1 Sandboxing

The RStudio Connect process runs as the `root` user. It needs escalated privileges to allow binding to protected ports and to create "unshare" environments that contain the R processes.

RStudio Connect runs its R processes as an unprivileged user; both a system default and content-specific overrides are supported. See Section 12.3 for details.

The "unshare" environment created for R execution involves first establishing a number of bind mounts and then switching to the target unprivileged user. RStudio Connect uses `unshare` to alter the execution context available to R processes. Within this newly established environment, a number of `mount` calls are made in order to hide or isolate parts of the filesystem.

You can learn more about `unshare` here. The `mount` call is detailed here. Your local man pages will document their behavior specific to your system.

The following locations are masked during R execution:

- The `Server.DataDir` directory containing all variable data used by RStudio Connect.
- The `SQLite.Dir` directory, which can optionally be placed outside the data directory.
- Configuration directories, including `/etc/rstudio-connect`.
- The `/tmp` and `/var/tmp` directories.

The following information is exposed during R execution:

- The `packrat` data directory (read-only except when installing packages).
- The `R` data directory (only when installing packages).
- The directory containing the unpackaged R code (Shiny, Plumber, and R Markdown).
- The document rendering destination directory (only for R Markdown).
- A per-process temporary directory (exposed over the original `/tmp` and `/var/tmp`).

When `Applications.HomeMounting` is enabled, the contents of `/home` are masked by an additional bind mount as follows:

- The contents of `/home` are masked by the home directory of the `RunAs` user.
- If the `RunAs` does not have a home directory, an empty directory masks `/home`.

The path to the home directory is always available through the `HOME` environment variable. With `Applications.HomeMounting`, the mounted path to the `HOME` directory is subject to change. Avoid hard-coding paths to either `/home` and `/home/username`.

Running R applications, like Shiny apps and Plumber APIs, have write access to the directory containing the unpackaged R code. This application directory is the working directory when launching an application. Data written here will be visible to all processes associated with that application but are not visible to other R processes. Application directory data remains available until that application is next deployed to RStudio Connect. A deployment creates a new application directory containing only the deployed content.

> RStudio Connect may launch multiple processes to service requests for an application. There is no coordination between these processes. Applications that write to local files could experience problems when different processes attempt to write to a single file.
>
> For example, two different processes writing to the same file may see output incorrectly interleaved or even overwritten.

We **do not** recommend using the file system for data persistence.

R Markdown documents have write access to the rendering destination directory and read access to a directory containing the unpackaged R code. The source directory is the working directory when calling `rmarkdown::render`. The destination directory is passed as the `output_dir` while a temporary directory is passed as the `intermediates_dir`. The intermediate directory is transient and not available after rendering completes. A new output directory is created whenever the document is rendered. Data created during one rendering is *not* visible to another.

R Markdown multi-document sites have a slightly different rendering pipeline than standalone documents. RStudio Connect uses the `rmarkdown::render_site` function, which does its rendering in-place. The content from the source directory is copied into the rendering destination directory in preparation for rendering. Site rendering has write access to the destination directory. Access to the original source directory is not provided because the source content is duplicated in the destination directory

The `rmarkdown::render_site` call usually places its output into a subdirectory (typically, '_site'). The contents of this output subdirectory will be moved to the root of the rendering destination directory, replacing any other content. No post-rendering file movement occurs if `rmarkdown::render_site` is instructed to render into the current directory instead of a subdirectory. This means that both source and output files will be available for serving.

> We recommend against configuring `rmarkdown::render_site` to write its output into the current directory. Rendering the site into a subdirectory (the default) allows RStudio Connect to remove source from the output directory.

RStudio Connect serves rendered content from the document output directory. This content remains available until a subsequent rendering is successful and activated (if requested). Neither incomplete nor unsuccessful document renderings affect the availability of previously rendered content.

12.2 Shiny Applications & Plumber APIs

> Note: Plumber APIs are currently in Beta.

Most of the R processes started by RStudio Connect are batch-oriented tasks. R is invoked, does a narrow set of work, and then exits. Shiny applications and Plumber APIs are different and may see an R process handle many requests for many users over their lifetimes. Both Shiny Applications and Plumber APIs are live applications that react to user requests on-demand.

RStudio Connect launches an R process tied to a live application when the first request arrives for that application. That R process will continue to service requests until it becomes idle and eventually terminated. If there is sufficient traffic against that application, RStudio Connect may launch additional processes to service those requests.

There are a number of configuration parameters which control the conditions under which processes for applications are launched and eventually reaped. The default values are appropriate for most applications but occasionally need customization in specialized environments. Section A.16 explains each of the options.

We recommend that adjustment to these runtime properties be done gradually.

12.3 User Account for R Processes

The RStudio Connect installation creates a local `rstudio-connect` user account. This account runs all the R processes; `root` does not invoke R. If you would like a different user to run R, customize the `Applications.RunAs` property.

Administrators can customize the `RunAs` user on a content-specific level. This means that different applications and R Markdown reports can be run using different Unix accounts. This setting can be found on the *Access*

tab when editing content settings. Publishers and Viewers are prohibited from changing the `RunAs` user on a content-specific level.

If you choose to specify a custom `RunAs` user for content, that user *must* be a member of the Unix group that is the primary group of the `Applications.RunAs` user.

> The `rstudio-connect` user, for example, has a primary group also named `rstudio-connect`. Any Unix account configured as a custom `RunAs` user for a Shiny application, Plumber API, or R Markdown report *must* be a member of the `rstudio-connect` group.

Installation of R packages always happens as the `Application.RunAs` user. An application or R Markdown report may override its `RunAs` setting; this alters how the deployed code is executed and does not impact package installation. See Section 12.1 for more information about process sandboxing.

12.4 Current user execution

RStudio Connect can use a local Unix account associated with the currently logged-in user when executing R. This feature requires that user authentication use PAM.

> See Section 10.6 for information about using PAM for user authentication.

The `Applications.RunAsCurrentUser` property specifies that content can be configured to execute as the currently logged-in user.

```
[Applications]
RunAsCurrentUser = true
```

Administrators can now customize the `RunAs` settings to permit current-user execution on a content-specific level. The *Access* content setting tab offers the option of executing using "The Unix account of the current user".

Content accessed anonymously will execute as the specified fallback `RunAs` user.

> See Section 12.3 for more information about `RunAs` customization.

Content execution settings are not altered when `RunAsCurrentUser` is enabled. The `RunAsCurrentUser` setting *permits* current-user execution but by itself does not change how R processes are launched. Each application or R Markdown report must explicitly request current-user execution.

All Unix accounts used to execute R *must* be members of the Unix group that is the primary group of the `Applications.RunAs` user. Applications are not permitted to launch if the Unix account associated with the logged-in user does not have the proper group membership.

> The `Applications.RunAs` setting uses the `rstudio-connect` user by default. This user has a primary group also named `rstudio-connect`. Any Unix account that may be used to execute applications or R Markdown reports *must* be a member of the `rstudio-connect` group.

12.5 PAM sessions

RStudio Connect can use PAM to establish the environment and resources available for R sessions.

> See Section 10.6 for information about using PAM for user authentication.

PAM sessions are enabled with the `PAM.UseSession` setting.

```
[PAM]
UseSession = true
```

The default PAM service name used for PAM sessions is `su`. This gives RStudio Connect the ability to launch processes as the specified user without requiring a password.

You can customize the PAM service name used for PAM sessions by customizing the `PAM.SessionService` setting.

```
[PAM]
SessionService = rstudio-connect-session
```

Any custom PAM service must contain the PAM directive that enables authentication with root privilege.

```
# Allows root to su without passwords (required)
auth sufficient pam_rootok.so
```

12.6 Path Rewriting

The sandboxing used by RStudio Connect involves bind mounts which map physical locations on disk onto different directory structures at runtime. Paths used by your R code use these sandboxed locations. If you need to find the physical file on disk, you will need to undo the path transformation.

This section gives some examples of path rewriting and offer some ways of finding the file you need.

Let's start with an `app.R` file that describes a Shiny application. This file will be in the `apps/XX/YY/` directory underneath the `Server.DataDir` location. The `XX` and `YY` path components correspond to the application ID and bundle (or deployment) ID for this version of your application. This directory is available at runtime as `/opt/rstudio-connect/mnt/app/`.

The directory structure of `/opt/rstudio-connect/mnt/` is just a number of empty directories. The "unshare" environment created during sandboxing allows RStudio Connect to associate different application directories with these mount directories.

Here are some common path transformations that may be helpful. All of the physical paths are beneath the `Server.DataDir` hierarchy that defaults to `/var/lib/rstudio-connect`. All of the sandbox paths are beneath the mount directory `/opt/rstudio-connect/mnt/`. This location is not customizable.

Physical path	Sandbox path
`DataDir/apps/XX/YY/`	`MountDir/app/`
`DataDir/reports/XX.ZZ`	`MountDir/report/`
`DataDir/R`	`MountDir/R`
`DataDir/packrat`	`MountDir/packrat`

Here are some actual path transformations using the default `Server.DataDir` location:

```
# A source Shiny application
/var/lib/rstudio-connect/apps/4/7/app.R
    => /opt/rstudio-connect/mnt/app/app.R

# A source Plumber API
/var/lib/rstudio-connect/apps/38/10/plumber.R
=> /opt/rstudio-connect/mnt/app/plumber.R

# A source R Markdown document
/var/lib/rstudio-connect/apps/8/12/index.Rmd
    => /opt/rstudio-connect/mnt/app/index.Rmd

# An HTML document rendered from that R Markdown document
/var/lib/rstudio-connect/reports/8.2/index.html
    => /opt/rstudio-connect/mnt/report/index.html
```

```
# A staticly deployed document
/var/lib/rstudio-connect/apps/17/21/index.html
    => /opt/rstudio-connect/mnt/app/index.html

# The Shiny package inside the packrat cache
/var/lib/rstudio-connect/packrat/3.2.5/v2/library/shiny/
  28d6903a44dc53bd4823fa43ccdc08e5/shiny
    => /opt/rstudio-connect/mnt/packrat/3.2.5/v2/library/shiny/
         28d6903a44dc53bd4823fa43ccdc08e5/shiny
```

12.7 Program Supervisors

You may need to modify the environment or resources available to R processes prior to R being launched. This can be accomplished using a program supervisor using the `Applications.Supervisor` configuration setting.

The supervisor command is provided the full R command-line, which *MUST* be invoked by the supervisor. The process exit code from R *MUST* be returned as the exit code of the supervisor. The file descriptors for standard input, output, and error *MUST NOT* be intercepted by the supervisor.

A supervisor is executed as the appropriate `RunAs` user. Package installation always uses the `Applications.RunAs` user. Other R processes will use the content-specific `RunAs` account, falling back to `Applications.RunAs` if no override was configured. See Section 12.3 for details.

Supervisors run within the sandbox established for any R process. See Section 12.1 for more information about process sandboxes.

RStudio Connect configures the `TMPDIR`, `HOME`, and `RSTUDIO_PANDOC` environment variables for launched R processes. RStudio Connect also manages package installation and references. Avoid altering any of this behavior in program supervisors.

12.7.1 Example Supervisors

Here is a configuration that uses the `nice` command to lower the priority of all R processes. See http://linux.die.net/man/1/nice for details about `nice`. Because process supervisors are run as a `RunAs` user and not as `root` or another super-user, you may not be permitted to assign a negative (higher priority) privilege.

```
[Applications]
Supervisor = nice -n 2
```

Here is a configuration that uses a custom script to prepare a custom execution environment before finally running R.

```
[Applications]
Supervisor = /some/script/that/prepares/an/environment.sh
```

Here is an example supervisor that echos its arguments, sets an environment variable, then invokes whatever arguments have been passed.

```
#!/bin/bash

echo arguments: "$@"
echo

export COMPANY_DATA_HOME="/data/resides/here"
```

```
exec "$@"
```

Your organization may use shell initialization scripts to establish a particular environment. This environment might not be completely compatible with how RStudio Connect attempts to launch R.

We recommend building supervisor scripts gradually and carefully. Changes to the environment can alter how your content executes or even prevent R from running correctly.

12.8 Using the `config` Package

The config package makes it easy to manage environment specific configuration values in R code. For example, you might want to use one value for a variable locally, and another value when deployed on RStudio Connect. The package vignette contains more information.

The desired configuration is identified to the config package by the `R_CONFIG_ACTIVE` environment variable. By default, R processes launched by RStudio Connect set `R_CONFIG_ACTIBE` to `rsconnect`. The value can be changed by modifying the `Applications.RConfigActive` configuration setting. Note that the value of `R_CONFIG_ACTIVE` is not available during package installation.

13 Content Management

RStudio Connect provides flexibility over how uploaded content is configured and shared.

13.1 Sharing Settings

Each deployment in RStudio Connect can have specific access controls which specify which users are allowed to view and/or edit that content.

13.1.1 Collaborators

The list of collaborators enumerates the users allowed to edit and help manage the settings for a given deployment. The content owner is always included as a collaborator. Collaborators must be either "publisher" or "administrator" accounts.

13.1.2 Viewers

A viewer is able to view content. Any type of account can be made a viewer for a given piece of content. Choose from the following options.

Everyone Any visitor to RStudio Connect will be able to view this content. This includes anonymous users who are not authenticated with the system.

All logged-in users All RStudio Connect accounts are permitted to view this content.

Specific users Specific users (or groups of users) are allowed to view this content. Other users will not have access.

Just me Only the owner of this content is able to view this content.

13.2 Vanity Paths

All content receives a URL that includes its numerical ID at at the time of deployment – something like `https://rsc.company.org/connect/#/apps/982`. Connect administrative users can create "vanity paths" for content which make the content available at an additional, customized URL.

This setting can be found at the bottom of the "Access" tab when editing a piece of content. There you can enter the path at which you want this content to be available and preview the complete URL. Once you "Save" your content, you'll be able to access your content at the new vanity URL.

Vanity URLs can not be nested inside of one another. So if a vanity URL `/finance/` already exists, you would not be able to create a new vanity URL at `/finance/budget/`. You may create sibling paths: `/finance/budget/` and `/finance/quarterly/` may both exist concurrently.

13.3 Tags

You can use tags to organize content and make it easy for users to find content that they're interested in. To begin, create a tag schema in the "Tags" section of the Admin dashboard by creating one or more tag categories. Define some tags, which can be nested any number of levels deep.

For example, if your data scientists are creating reports covering different geographical areas, you could create a category called "Geographical Area". Then, you could create tags such as "Americas" or "Asia" and nest the tags "North America" and "South America" under "Americas".

Only administrators can create and edit the tag schema. Categories and tags can be added, deleted, and renamed. Once a tag or category is deleted, all tags nested under it are also deleted.

Collaborators can associate content with one or more tags in the "Tags" tab of the content settings sidebar. Users can filter by tags to discover content, as long as they have permission to view that content.

For example, if multiple reports analyze the same set of data, those reports could be tagged with some identifier, such as "FY2016 Q3" for the third quarter of the 2016 fiscal year. A report that analyzes the third and fourth quarter could be tagged with "FY2016 Q3" and "FY2016 Q4", and would appear when a user filters for either "FY2016 Q3" or "FY2016 Q4".

13.4 Bundle Management

Content published to RStudio Connect is encapsulated in a "bundle" that contains the source code and data necessary to execute the content. An application or report is updated by uploading a new bundle. Old bundles are retained on disk until you reach the limit imposed by `Applications.BundleRetentionLimit` at which point older bundles will be deleted.

Users can manage their own bundles in the dashboard by clicking the "Source Versions" button. Collaborators can delete, download, activate, and view activation logs for their applications' bundles. Activating a different bundle is a way of "rolling back" or "rolling forward" to an older or newer version of your application, respectively.

Activating an alternative bundle for a Shiny application will cause new incoming users to be directed to the new version of the application but will not interrupt existing users of the application who are viewing the previously activated bundle. For reports, activating an alternate bundle will immediately render the newly activated bundle and promote it to be the authoritative version of that document. For parameterized reports, only the default variant will be rerendered; other instances of the report will not automatically be regenerated, but the next manual or scheduled update will be performed on the newly selected bundle.

When Activating an alternative bundle for a Plumber API, existing requests will be serviced by processes already launched running the old code. New requests will be serviced by new processes running the new code.

13.5 API Keys

RStudio Connect allows users to access hosted content outside the web browser by utilizing API Keys - e.g. via shell scripts. API Keys are enabled by default. To change this behavior please see Section 13.5.2.

13.5.1 How this Works

To retrieve static content or to invoke Plumber endpoints via API Keys an HTTP request must be made to the target URL of the published content. The request must contain an HTTP header whose key is `Authorization` and value is set to `Key API_KEY`.

```
Authorization: Key ABCDEFGHIJKLMNO
```

API Keys have the same authorization access levels as the user that owns them. Someone who uses an API Key will be able to view all content that the owner of the API Key has access to. API Keys are shared secrets and as such they should be stored securely and only be given to trusted applications. It is advisable that content requests be made securely over HTTPS. If a user believes that an API Key has been compromised, they can revoke just that key by deleting it.

For more details regarding API Keys please see the API Keys section in the User Guide.

To learn how to configure RStudio Connect to listen for HTTPS requests please see Section A.3.

13.5.2 Configuring API Keys

To disallow API Keys, set `Authentication.APIKeyAuth` to `false`.

```
[Authentication]
APIKeyAuth = false
```

14 R

RStudio Connect offers a flexible way of deploying your Shiny applications, Plumber APIs, and R Markdown content against a variety of R versions.

A compatible version of R is identified when content is deployed. That R installation is used any time R is needed for that content. Package installation, starting a Shiny application or Plumber API, and rendering R Markdown documents will all use the version of R discovered at deploy-time.

RStudio Connect allows different content to rely on different versions of R. For example, Alice's R Markdown document may require version 3.2.4 of R while Bob's Shiny application needs R version 3.3.1. Those two deployments using different R versions can coexist in RStudio Connect without conflict.

This chapter discusses RStudio Connect can be configured to support more than one version of R and how R version compatibility is determined.

Available R installations are analyzed at startup. Connect logs the discovered R versions. Connect will fail to start if no R installation can be found.

Changing any of the configuration items discussed in this chapter requires a restart of RStudio Connect.

14.1 Installing R

Ubuntu and Red Hat/CentOS Linux distributions offer a version of R; installation of the system R is detailed in sections 2.1.1 and 2.1.2.

If you are attempting to make additional R versions available side-by-side with the system version you typically need to install from source. The RStudio Server documentation has a good reference for building and installing R into alternate locations.

This RStudio Support article also includes R installation instructions.

14.2 Upgrading R

RStudio Connect supports running multiple versions of R. In most cases, upgrading R should consist of building the new version of R and retaining the previous version. We strongly recommend supporting multiple versions of R instead of upgrading and maintaining a single version of R. Supporting multiple versions of R is the best way to ensure applications or reports published with specific package dependencies will continue to run.

In cases where a single version of R is being managed, R can be upgraded using the system package manager with the following steps:

1. Stop RStudio Connect, see 5.1
2. Follow the instructions to upgrade R.

   For Ubuntu, be sure an up-to-date CRAN repo is in the source list, and then run:

   ```
   $ sudo apt-get update
   $ sudo apt-get install r-base --upgrade
   ```

   For RedHat/CentOS:

   ```
   $sudo yum update R
   ```

3. Start RStudio Connect

Following an upgrade, content dependent on R will be rebuilt on-demand. For example, during the next execution of a scheduled R Markdown document, Connect will automatically reinstall and rebuild all of the necessary packages before rendering the report. For Shiny applications and Plumber APIs, the reinstall and rebuild will occur the first time the application is requested. During the package updates, Connect will present a message and spinner indicating to the end user that the Shiny application will be available after the packages are successfully installed and built for the new version of R.

> Not all packages can be reinstalled and rebuilt on newer versions of R. Rebuilding and restoring packages can take a significant amount of time and could delay or prevent the rendering of a report or the availability of a Shiny application.

14.3 R Versions

RStudio Connect supports two ways of discovering R versions: scanning well-known locations and through direct configuration. Connect will use the PATH environment variable to discover a version of R if one is not otherwise found.

14.3.1 Scanning

RStudio Connect can automatically scan for versions of R in the following locations:

```
/usr/lib/R
/usr/lib64/R
/usr/local/lib/R
/usr/local/lib64/R
```

```
/opt/local/lib/R
/opt/local/lib64/R
```

In addition, Connect scans all subdirectories of the following directories within `/opt`:

```
/opt/R
/opt/local/R
```

For example, any of the following installed versions of R will be automatically detected:

```
/opt/R/3.1.3
/opt/R/3.2.4
/opt/local/R/3.3.1
```

Scanning happens by default. You can disable version scanning by customizing the `Server.RVersionScanning` property.

```
[Server]
RVersionScanning = false
```

14.3.2 R Versions

The `Server.RVersion` property can be used to specify alternate locations for installations of R. Specify this property once for each R installation directory.

```
[Server]
RVersion = /shared/R/3.3.1
RVersion = /shared/R/3.2.4
RVersion = /shared/R/3.1.3
```

14.3.3 /etc/rstudio/r-versions

The `/etc/rstudio/r-versions` file is an alternative way of specifying R versions and is shared with RStudio Server. List your R installations in this file. Note that the `r-versions` file is not created by default and will need to be created.

```
/shared/R/3.3.1
/shared/R/3.2.4
/shared/R/3.1.3
```

14.3.4 Excluding Versions

If you have versions of R that are picked up by automatic scanning but which you would like to exclude, disable R version scanning and explicitly specify all versions you would like to use in the `/etc/rstudio/r-versions` file or with the `Server.RVersion` configuration property.

Here is an example configuration that disables scanning, and specifies precisely two R versions that will be available for use.

```
[Server]
RVersionScanning = false
RVersion = /opt/R/3.3.1
RVersion = /opt/R/3.2.4
```

14.4 R Version Matching

RStudio Connect attempts to find an R installation that is appropriate for your content. By default, it applies a "nearest" matching approach. This algorithm attempts to always find a version of R to use with your content. More deployments will succeed but not always with the same version of R that is used by the author.

If you would prefer a strict association between authored and deployed R versions, you can choose to use an "exact" matching approach.

The "nearest" matching algorithm is the most flexible option and favors publication of your content over precise duplication of the authoring environment.

The "major-minor" algorithm is a middle ground between "nearest" and "exact". It requires exact `MAJOR.MINOR` matching but is flexible about the patch level. This is a useful option when your desktop and server may occasionally have different update cycles when installing bug fix releases.

An inconsistent version of R occasionally causes problems when installing package dependencies. For the best results, make sure that RStudio Connect has access to the same versions of R used to author content.

The R version matching approach is controlled with the `Server.RVersionMatching` configuration setting.

nearest Find an R installation that is close to the version of R used when authoring the Shiny application, Plumber API, or R Markdown document. This algorithm uses the ordered tests when looking for `MAJOR.MINOR.PATCH` version matches:

1. Use exact version match.
2. If there are matching `MAJOR.MINOR` releases, use least-greater version.
3. If there are matching `MAJOR.MINOR` releases, use latest of these.
4. Use least-greater version across all releases.
5. Use latest across all releases.

major-minor Find an R installation that is close to the version of R used when authoring the Shiny application, Plumber API, or R Markdown document requiring an exact `MAJOR.MINOR` version match. If a compatible version cannot be found, content will fail to deploy. The algorithm is a constrained "nearest" search:

1. Use exact version match.
2. If there are matching `MAJOR.MINOR` releases, use least-greater version.
3. If there are matching `MAJOR.MINOR` releases, use latest of these.

exact Finds an R installation that exactly matches the version of R used when authoring the deployed content. If a matching version cannot be found, content will fail to deploy.

15 Package Management

15.1 Package Installation

RStudio Connect installs the R package dependencies of Shiny applications, Plumber APIs, and R Markdown documents when that content is deployed. The RStudio IDE uses the `rsconnect` and `packrat` packages to bundle the relevant source code and document its dependencies. RStudio Connect then uses `packrat` to duplicate those package dependencies on the server.

Packrat attempts to re-use R packages whenever possible. The `shiny` package, for example, should be installed only when the first Shiny application is deployed. Subsequent Shiny applications can use that package and see faster deployments as a result. Packrat also allows multiple versions of a package to exist on a system. Two Shiny applications referencing different versions of `shiny` will reference the correct Shiny installation and these two packages will not conflict with each other.

Resolving which packages need installing and which are already available all happens when you deploy content to RStudio Connect.

15.1.1 External Package Installation

> **Warning**: Adding external packages decreases the reproducibility and isolation of content on RStudio Connect, and should only be done as a last resort.

You can indicate that a system-wide installation of a package should be used instead of one fetched by packrat. To do this, set each system package name to the `External` option under the `Packages` heading.

For example, `RJava` or `ROracle` are large installations, potentially with odd dependencies, such as your choice of JDK and/or Oracle InstantClient. First, you would install these packages in every R installation that RStudio Connect will be using. Then, you would configure RStudio Connect with the following parameters:

```
[Packages]
External = ROracle
External = RJava
```

This is the same as settings the packrat option `external.packages` to `c("ROracle", "RJava")` using `packrat::set_opts`. The `external.packages` option instructs `packrat::restore` to load certain packages from the user library. See the packrat documentation for more information.

15.1.2 Proxy Configuration

If the `http_proxy` and/or `https_proxy` environment variables are provided to RStudio Connect when the server starts, those variables will be passed to all R processes run by RStudio Connect, including the package installation process.

Setting the `HttpProxy` and `HttpsProxy` configuration options under the `Packages` heading will provide their values as `http_proxy` and `https_proxy` only when packages are installed during deployment. This could be useful if you have a special proxy just for downloading package dependencies; for example, you could regulate access to unapproved packages in non-CRAN repositories through the use of a url whitelist.

15.2 Private Repositories

Packrat records details about how a package was obtained in addition to information about its dependencies. Most public packages will come from a public CRAN mirror. Packrat lets RStudio Connect support alternate repositories in addition to CRAN.

> Learn how to create your own custom repository; this directory can then be shared over HTTP or through a shared filesystem.

Here are some reasons why your organization might use an alternate/private repository.

1. Internally developed packages are made available through a corporate repository. This is used in combination with a public CRAN mirror.
2. All packages (private and public) are approved before use and must be obtained through the corporate repository. Public CRAN mirrors are not used.
3. Direct access to a public CRAN mirror is not permitted. A corporate repository is used as a proxy and caches public packages to avoid external network access.

RStudio Connect supports private repositories in these situations given that the deploying instance of R is correctly configured. No adjustment to the RStudio Connect server is needed.

Repository information is configured using the `repos` R option. Your users will need to make sure their desktop R is configured to use your corporate repository.

> RStudio IDE version 0.99.1285 or greater is needed when using repositories other than the public CRAN mirrors.

We recommend using an `.Rprofile` file to configure multiple repositories or non-public repositories.

The `.Rprofile` file should be created in a user's home directory.

```
# A sample .Rprofile file with two different package repositories.
local({
  r <- getOption("repos")
  r["CRAN"] <- "https://cran.rstudio.com/"
  r["mycompany"] <- "http://rpackages.mycompany.com/"
  options(repos = r)
})
```

This `.Rprofile` creates a custom `repos` option. It instructs R to attempt package installation first from `"CRAN"` and then from the `"mycompany"` repository. R installs a package from the first repository in `"repos"` containing that package.

With this custom `repos` option, you will be able to install packages from the `mycompany` repository. RStudio Connect will be able to install these packages as code is deployed.

For more information about the `.Rprofile` file, see `help(Startup)` in R. For details about package installation, see `help(install.packages)` and `help(available.packages)`.

15.3 Private Packages

Packages available on CRAN, a private package repository, or a public GitHub repository are automatically downloaded and built when an application is deployed. RStudio Connect cannot automatically obtain packages from private GitHub repositories, but a workaround is available.

> We recommend using a private repository to host internal packages when possible. See Section 15.2 for details.

The configuration option `Server.SourcePackageDir` can reference a directory containing additional packages that Connect would not otherwise be able to retrieve. This directory and its contents must be readable by the `Applications.RunAs` user. Connect will look in this directory for packages before attempting to obtain them from a remote location.

This feature has some limitations.

- The package must be tracked in a git repository so that each distinct version has a unique commit hash associated with it.
- The package must have been installed from the git repository using the `devtools` package so that the hash is contained in the `DESCRIPTION` file on the client machine.

If these conditions are met, you may place `.tar.gz` source packages into per-package subdirectories of `SourcePackageDir`. The proper layout of these files is `<package-name>/<full-git-hash>.tar.gz`.

For example, if `Server.SourcePackageDir` is defined as `/opt/R-packages`, source bundles for the `MyPrivatePkg` package are located at `/opt/R-packages/MyPrivatePkg`. A commit hash of `28547e90d17f44f3a2b0274a2aa1ca820fd35b80` needs its source bundle stored at the following path:

`/opt/R-packages/MyPrivatePkg/28547e90d17f44f3a2b0274a2aa1ca820fd35b80.tar.gz`

When private package source is arranged in this manner, users of RStudio Connect will be able to use those package versions in their deployed content.

Be aware that this mechanism is specific to the commit hash, so you will either need to make many git revisions of your package available in the `SourcePackageDir` directory hierarchy or standardize to a particular git commit of the package.

16 Historical Metrics

This section describes the configuration and management of historical metrics, enabled with the `Metrics.Enabled` setting.

RStudio Connect uses a separate `rserver-monitor` process to record resource (CPU, memory, etc.) usage over time. It is only active when historical metrics are enabled. The customization settings described in the remainder of this section have no effect when `Metrics.Enabled` is off.

16.1 Historical Metrics Settings

Metrics data is written by default to a set of RRD files. This data is stored by default at `/var/lib/rstudio-connect/metrics`. You can specify an alternate data path by using the `DataPath` setting mentioned in Section A.18.

The `rserver-monitor` process runs (by default) with the same user account Connect uses to run its R processes. By default, this user account is `rstudio-connect` (see the `RunAs` setting in Section A.13). You can specify an alternate user account for the `rserver-monitor` process by modifying the `User` setting. See Section A.18 for details.

RStudio Connect also supports logging of metrics to Graphite, and it supports disabling its default behavior of logging to RRD. Please see Section A.18 for more options for configuring the historical metrics in Connect.

16.2 Historical Metrics Process Management

Connect automatically spawns a process (`rserver-monitor`) to help maintain historical data. If this process exits, Connect will restart it in an attempt to record as much historical information as possible. Connect will delay restarting `rserver-monitor` if it observes rapid, repeated failures.

Since the `rserver-monitor` needs permission to write data to the metrics data directory, Connect attempts to ensure the necessary permissions at startup. When Connect starts, it grants ownership of the metrics data directory to the user account that will be used to start `rserver-monitor`.

16.3 Historical Metrics Process Logging

The `rserver-monitor` process logs its output to syslog. If the process is unable to run, you can check the system log (e.g., `/var/log/messages` or `/var/log/syslog`) for messages.

A Configuration Options

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

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

The RStudio Connect 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`.

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 Connect process causes the on-disk configuration to be reread. The server is reconfigured with the latest values of these reloadable properties. See 5.1 for details about sending a `HUP` signal to your Connect process.

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

A.1 Server

The `Server` section contains configuration properties which apply across the whole of RStudio Connect 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 Connect will store its variable data.

Type: string

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

`LandingDir` Specifies an optional path from which a customized landing page is served to logged-out users. See `examples/landing-page` for a directory containing a sample landing page.

Type: string

Default: *<empty-string>*

`EnableSitemap` Specifies if RStudio Connect should provide a `/sitemap.xml` file enumerating the publicly available apps.

Type: boolean

Default: `false`

`RVersionMatching` Specifies how RStudio Connect attempts to match R version associated with uploaded content with the R versions available on the system. Allows values of `nearest` or `exact`.

Type: string

Default: `nearest`

`RVersion` Path to an R installation root. Multiple definitions can be used to provide multiple locations with R.

Type: multi-string

Default: *unspecified*

`RVersionScanning` Scan for R installations in well-known locations.

Type: boolean

Default: `true`

`CompilationConcurrency` The amount of parallelism allowed to `make` during R package installs. This value is passed to `make` as the value given to the `-jNUM` flag.

Type: integer

Default: `4`

`SourcePackageDir` A directory containing source bundles for packages that are unavailable on either CRAN or a public GitHub repository. Must be readable by the `Applications.RunAs` user.

Type: string

Default: *<empty-string>*

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

Type: string

Default: *<empty-string>*

`SenderEmail` An email address used by RStudio Connect to send outbound email. The system will not be able to send administrative email until this setting is configured.

Type: string

Default: *<empty-string>*

`ViewerKiosk` When enabled, RStudio Connect does not prompt view-only users to request elevated privileges when attempting to access restricted resources.

Type: boolean

Default: `false`

`HideEmailAddresses` When enabled, RStudio Connect will not expose email addresses in API requests or its dashboard.

Type: boolean

Default: `false`

`MailAll` When enabled, RStudio Connect will allow scheduled and on-demand documents to send email to all users of the system.

Type: boolean

Default: `false`

`PublicWarning` An HTML snippet used to inject a message into the RStudio Connect dashboard welcome pages.

Type: string

Default: *<empty-string>*

Reloadable: true

`LoggedInWarning` An HTML snippet used to inject a message into the RStudio Connect recent views.

Type: string

Default: *<empty-string>*

Reloadable: true

`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, Connect sets the `Server` HTTP header to something like `RStudio Connect v1.2.3`. This setting allows you to override that value.

Type: string

Default: *<empty-string>*

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

Type: string

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

`CustomHeader` Custom HTTP header that should be added to responses from Connect 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 Connect would otherwise have set. Multiple definitions can be used to provide multiple custom headers.

Type: multi-string

Default: *unspecified*

`FrameOptionsContent` The value for the `X-Frame-Options` HTTP header for all user-uploaded content (Shiny apps, RMDs, etc.). If empty, no header will be added.

Type: string

Default: *<empty-string>*

`FrameOptionsDashboard` The value for the `X-Frame-Options` HTTP header for the Connect dashboard and all other Connect pages. If empty, no header will be added.

Type: string

Default: `DENY`

A.2 Http

The `Http` section contains configuration properties which control the ability of RStudio Connect to listen for HTTP requests. RStudio Connect 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 Connect 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.3 Https

The `Https` section contains configuration properties which control the ability of RStudio Connect to listen for HTTPS requests. RStudio Connect 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 Connect 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.4 HttpRedirect

The `HttpRedirect` section contains configuration properties which control the ability of RStudio Connect 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 Connect 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.5 Database

The `Database` section contains configuration properties which control the location of and how RStudio Connect interacts with its database.

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

`Provider` The type of database to use

Type: string

Default: sqlite

`Dir` This property is deprecated. Please use Sqlite.Dir instead.

Type: string

Default: *<empty-string>*

`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.6 Authentication

The `Authentication` section contains configuration properties which control how users will log into RStudio Connect.

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

`Provider` Specifies the type of user authentication. Allows values of `password`, `oauth2`, `ldap`, `pam`, or `proxy`.

Type: string

Default: `password`

`Name` Specifies a meaningful name for your authentication provider. This presented on the sign-in page and gives users context about the credentials being requested. If unspecified, RStudio Connect will use a generic name for the chosen provider. Just using your company name is often a good choice.

Type: string

Default: *<empty-string>*

`Lifetime` The lifetime of an authenticated session.

Type: duration

Default: `720h` (30 days)

`Inactivity` The period of time after which inactive sessions are considered idle and therefore invalid. Effective only when non-zero and less than `Authentication.Lifetime`.

Type: duration

Default: `0`

`APIKeyAuth` Whether API key authentication is enabled.

Type: boolean

Default: true

A.7 Password

The `Password` section contains configuration properties which control how RStudio Connect's default password authentication provider behaves.

See Section 10.3 for details about configuring password authentication for RStudio connect.

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

`SelfRegistration` Allow users to self-register. Self-registered users will be created using the role specified in the `Authorization.DefaultUserRole` setting.

Type: boolean

Default: `true`

A.8 OAuth2

The `OAuth2` section contains configuration properties which control how RStudio Connect communicates with Google OAuth2 servers in order to authenticate users.

Section 10.5 contains more information about configuring RStudio Connect to use Google for authentication.

The `DiscoveryEndpoint` property should be configured as:

```
[OAuth2]
DiscoveryEndpoint = https://accounts.google.com/.well-known/openid-configuration
```

These properties must appear after `[OAuth2]` in the configuration file.

`DiscoveryEndpoint` Specifies a URL for the OAuth2 discovery endpoint. Required for all OAuth2 configurations.

Type: string

Default: *<empty-string>*

`ClientId` Identifier for OAuth2 client. Required for all OAuth2 configurations.

Type: string

Default: *<empty-string>*

`ClientSecret` Client secret for the configured client ID. Either `OAuth2.ClientSecret` or `OAuth2.ClientSecretFile` must be specified when using OAuth2. `OAuth2.ClientSecret` takes priority if both properties are set.

Type: string

Default: *<empty-string>*

`ClientSecretFile` Path to file containing the client secret. Either `OAuth2.ClientSecret` or `OAuth2.ClientSecretFile` must be specified when using OAuth2. `OAuth2.ClientSecret` takes priority if both properties are set.

Type: string

Default: *<empty-string>*

`AllowedDomains` Space-separated list of domains permitted to authenticate.

Type: string

Default: *<empty-string>*

Reloadable: true

`AllowedEmails` Space-separated list of email addresses permitted to authenticate. When used without `OAuth2.AllowedDomains`, only the email addresses listed here will be allowed access. When used with `OAuth2.AllowedDomains`, the email addresses listed here will be **added** to those with valid domains.

Type: string

Default: *<empty-string>*

Reloadable: true

A.9 LDAP

The `LDAP` section contains configuration properties which control how RStudio Connect communicates with an LDAP or Active Directory server.

See Section 10.4 for details about how to configure RStudio Connect with LDAP authentication. Section D contains many configuration examples.

The `LDAP` section is different from many other configuration sections, as it allows multiple, distinctly named configuration instances. This name is *case sensitive*. A named section looks like:

```
[LDAP "European LDAP Server"]
```

All of the LDAP configuration properties must appear after `[LDAP "<name>"]` in the configuration file.

`RequireExternalUsernames` Require LDAP, Proxy, and PAM authentication providers to provide valid usernames. User completion will be disabled; if an invalid username is received from the provider, we will throw an error.

Type: boolean

Default: `true`

`ServerAddress` Specifies the location of the LDAP/AD server. This should be of the form `<host>:<port>`. The `host` may be either an IP or DNS address. Most LDAP/AD servers operate on port `389` or `636`.

Type: string

Default: *<empty-string>*

`TLS` When enabled, all connections to your LDAP/AD server will use TLS (SSL).

Type: boolean

Default: `false`

`StartTLS` When enabled, the connection will initially be made on an insecure port then the channel will be upgraded to TLS using StartTLS.

Type: boolean

Default: `false`

`ServerTLSInsecure` This option controls if RStudio connect will verify the server's certificate chain and host name. When enabled, RStudio Connect will accept any certificate presented by the server and any host name in that certificate. Setting to `true` is susceptible to man-in-the-middle attacks but is required in some circumstances, such as when using a self-signed certificate.

Type: boolean

Default: `false`

`TLSCACertificate` Path to a certificate authority used to connect an LDAP server.

Type: string

Default: *<empty-string>*

`UserObjectClass` The name of the LDAP `objectClass` used to define users.

Type: string

Default: *<empty-string>*

`UserFirstNameAttribute` The LDAP user attribute containing a user's first name. This is often the `givenName` attribute. This attribute is case-sensitive.

Type: string

Default: *<empty-string>*

`UserLastNameAttribute` The LDAP user attribute containing a user's last name. The `sn` attribute will usually contain last name. This attribute is case-sensitive.

Type: string

Default: *<empty-string>*

`UserEmailAttribute` The LDAP user attribute containing a user's email address. Many systems use the `mail` attribute. This attribute is case-sensitive.

Type: string

Default: *<empty-string>*

`UsernameAttribute` The LDAP user attribute containing a user's username. Commonly used attributes include `uid`, `cn`, and `sAMAccountName`. This attribute is case-sensitive.

Type: string

Default: *<empty-string>*

`BindDN` A DN for a read-only admin account that is used during double-bind authentication and for certain operations that do not occur during the login sequence (such as searching). Must be paired with `BindPassword`.

Type: string

Default: *<empty-string>*

`BindPassword` The password for the `BindDN` account.

Type: string

Default: *<empty-string>*

`AnonymousBind` Enable anonymous bind. An anonymous user must have rights to search and view all pertinent groups, group memberships, and users.

Type: boolean

Default: `false`

`UserSearchBaseDN` The base DN used when performing user searches.

Type: string

Default: *<empty-string>*

A.10 PAM

The `PAM` section contains configuration properties which control how RStudio Connect interacts with the PAM (Pluggable Authentication Module) API.

See Section 10.6 for details about configuring an appropriate PAM authentication profile for RStudio connect.

See Section 12.5 for information about using PAM sessions when launching R processes.

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

`RequireExternalUsernames` Require LDAP, Proxy, and PAM authentication providers to provide valid usernames. User completion will be disabled; if an invalid username is received from the provider, we will throw an error.

Type: boolean

Default: `true`

`Service` Specifies the PAM service name that RStudio Connect will use when authenticating users.

Type: string

Default: `rstudio-connect`

`UseSession` Use PAM sessions when launching R processes.

Type: boolean

Default: `false`

`SessionService` Specifies the PAM service name that RStudio Connect will use for PAM sessions.

Type: string

Default: `su`

A.11 Proxied Authentication

The `ProxyAuth` section contains configuration properties which control how RStudio Connect utilizes an external authentication server which proxies all requests.

See Section 10.7 for details about configuring an appropriate proxied authentication for RStudio connect.

`RequireExternalUsernames` Require LDAP, Proxy, and PAM authentication providers to provide valid usernames. User completion will be disabled; if an invalid username is received from the provider, we will throw an error.

Type: boolean

Default: `true`

`UsernameHeader` Specifices the name of the header that will contain a username provided by the proxy.

Type: string

Default: `X-Auth-Username`

A.12 Authorization

The `Authorization` section contains configuration properties which control permissions and privileges when accessing RStudio Connect.

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

`DefaultUserRole` Specifies what abilities given to a newly created user. Allows values `viewer`, `publisher`, or `administrator`.

Type: string

Default: `viewer`

`AdminEditableUsernames` When enabled, allows administrators to edit usernames if using LDAP, PAM, or Proxy authentication. This setting is not applicable for the Password and OAuth2 authentication providers.

Type: boolean

Default: `true`

A.13 Applications

The `Applications` section contains configuration properties which control how RStudio Connect communicates with R processes.

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

`RunAs` User used to invoke R.

Type: string

Default: `rstudio-connect`

`RunAsCurrentUser` Allows content to execute as the logged-in user when using PAM authentication.

Type: boolean

Default: `false`

`RConfigActive` Specifies a value for the `R_CONFIG_ACTIVE` environment variable for R processes; supported by the `config` package.

Type: string

Default: `rsconnect`

`Supervisor` Specifies a command to wrap the execution of R.

Type: string

Default: *<empty-string>*

`HomeMounting` Specifies that the contents of `/home` should be hidden from R processes with additional bind mounts. The existing `/home` will have the home directory of the `RunAs` user mounted over it. If `RunAs` does not have a home directory, an empty temporary directory will mask `/home` instead. Launched R processes can discover this location through the the `HOME` environment variable.

Type: boolean

Default: `false`

`ShinyBookmarking` Toggles support for on-disk Shiny bookmarking state. Configuring Shiny applications to use server bookmarking is described in this article.

Type: boolean

Default: `true`

`ExplicitPublishing` Content requires an explicit publication step after creation.

Type: boolean

Default: `true`

`ViewerOnDemandReports` Allow logged in report viewers to generate an ad-hoc rendering. The `ViewerCustomizedReports` property is implicitly disabled when this property is disabled.

Type: boolean

Default: `false`

`ViewerCustomizedReports` Allow logged in report viewers to customize the parameters of an ad-hoc rendering.

Type: boolean

Default: `false`

`BundleReapFrequency` Time between the worker that deletes filesystem data for bundles in excess of our retention limit.

Type: duration

Default: `24 hours`

`BundleRetentionLimit` Maximum number of bundles per app for which we want to retain filesystem data. The default is 0, which means retain everything.

Type: integer

Default: `0`

`ScheduleConcurrency` Number of scheduled reports permitted to execute in parallel

Type: integer

Default: `2`

`ConnectionTimeout` Maximum time allowed without data sent or received across a client connection. A value of `0` means connections will never time-out (not recommended).

Type: duration

Default: `1h`

`ReadTimeout` Maximum time allowed without data received from a client connection. A value of `0` means a lack of client (browser) interaction will never cause the connection to close. This is useful when deploying dashboard applications which send regular updates but have no need for interactivity.

Type: duration

Default: `1h`

`DisabledProtocols` List of comma-delimited protocols to disable on the SockJS client. Allows values of `websocket`, `xhr-streaming`, `iframe-eventsource`, `iframe-htmlfile`, `xhr-polling`, `iframe-xhr-polling`, or `jsonp-polling`. Protocols `xdr-streaming` and `xdr-polling` are always disabled.

Type: string

Default: *<empty-string>*

A.14 Packages

The `Packages` section contains configuration properties which alter how R packages are installed and managed. See Section 15 for details.

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

`HttpProxy` Value to be set for the `http_proxy` environment variable during package installation when content is deployed. When set, this will override the `http_proxy` environment variable only when content is built by connect.

Type: string

Default: *<empty-string>*

`HttpsProxy` Value to be set for the `https_proxy` environment variable during package installation when content is deployed. When set, this will override the `https_proxy` environment variable only when content is built by connect.

Type: string

Default: *<empty-string>*

`External` Package to be excluded from packrat build. This can be provided multiple times, once for each package. You will need this package available in your library path.

Type: multi-string

Default: *unspecified*

A.15 Client

The `Client` section contains configuration properties which control the behavior of browsers when interacting with applications. Interactive Shiny applications are the primary example.

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

`ReconnectTimeout` The amount of time to allow a user connection to be restored. If a zero value, reconnects will be disabled. Disabling reconnects can cause instability with the `session$allowReconnects(TRUE)` feature in Shiny.

Type: duration

Default: `0`

A.16 Runtime/Scheduler

The `Scheduler` section contains configuration properties which control how RStudio Connect manages R processes for deployed Shiny applications and Plumber APIs. These properties are managed on an individual application under the `Runtime` tab.

RStudio Connect makes a determination on each new client connection about whether or not it needs to spawn an additional R process. That computation analyzes the number of current R processes and the number of active connections against those processes. If a substantial percentage of connections are consumed, RStudio Connect will create a new process rather than causing the existing processes to become more busy. That percentage of connection use is called the "load factor".

The algorithm that considers the current load factor looks like the following pseudocode.

```
// Given:
//   numProcesses
//     - The number of R processes for the current application.
//   numConnections
//     - The number of connections across all R processes associated
//       with the current application.
allowedConnections = numProcesses * Scheduler.MaxConnsPerProcess
currentLoadFactor = numConnections / allowedConnections
if currentLoadFactor > Scheduler.LoadFactor {
    // Create a new process if the new process will not exceed
    // Scheduler.MaxProcesses
}
```

The `Scheduler.InitTimeout` and `Scheduler.IdleTimeout` properties may need adjusting when a Shiny application takes a very long time to startup. Increasing `InitTimeout` will allow more time for the Shiny application to start. An increase to `IdleTimeout` lets idle R processes linger longer so they are available the next time a request arrives - avoiding the startup penalty.

The scheduler properties can be changed in the configuration file and apply to all Shiny applictions. The RStudio Connect dashboard allows custom scheduler settings for individual applications.

We recommend that `Scheduler` property adjustment be done gradually.

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

`MaxProcesses` Specifies the total number of concurrent R processes allowed for a single application.

Type: integer

Default: `3`

`MaxConnsPerProcess` Specifies the maximum number of client connections allowed to an individual R process. Incoming connections which will exceed this limit are routed to a new R process or rejected.

Type: integer

Default: `20`

`LoadFactor` Controls how aggressively new R processes will be spawned.

Type: decimal

Default: `0.5`

`InitTimeout` Maximum time to wait for an app to start.

Type: duration

Default: `60s`

`IdleTimeout` Minimum time to keep a worker process alive after it goes idle.

Type: duration

Default: `5s`

`MinProcessesLimit` Maximum value allowed for the `MinProcesses` setting on an application level. All applications default to `MinProcesses=0`, but `MinProcesses` can be increased to this limit per application.

Type: integer

Default: `20`

A.17 Jobs

The `Jobs` section contains configuration properties which control the retention of metadata associated with R process execution.

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

`MaxCompleted` The maximum number of completed jobs preserved on disk for any one application. When this limit is reached, the oldest completed jobs for an application will be deleted as new jobs are launched. On-disk job metadata is removed if either the `MaxCompleted` or `OldestCompleted` restrictions are violated.

Type: integer

Default: `1000`

`OldestCompleted` The maximum age of a completed job retained on disk. Jobs older than this setting will be deleted. Set to zero to remove restrictions on the age of a completed job. On-disk job metadata is removed if either the `MaxCompleted` or `OldestCompleted` restrictions are violated.

Type: duration

Default: `720h`

A.18 Historical Metrics

The `Metrics` section contains configuration properties which control how RStudio Connect manages the `rserver-monitor` process for monitoring the use of resources (CPU, memory, etc.) for historical metrics.

See Section 16 for more details about historical metrics in Connect.

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

`Enabled` Specifies whether or not the `rserver-monitor` process that collects historical metrics will be started.

Type: boolean

Default: `true`

`User` The user for the `rserver-monitor` process.

Type: string

Default: `{Applications.RunAs}`

`DataPath` The path for writing log entries and RRD database files.

Type: string

Default: `{Server.DataDir}`/metrics

`Interval` The frequency of historical metrics collection.

Type: duration

Default: 60s

`RRDEnabled` Enable logging of historical metrics to RRD.

Type: boolean

Default: `true`

`GraphiteEnabled` Enable logging of historical metrics to Graphite.

Type: boolean

Default: `false`

`GraphiteHost` Host to which to send Graphite historical metrics.

Type: string

Default: 127.0.0.1

`GraphitePort` Port to which to send Graphite historical metrics.

Type: integer

Default: 2003

`GraphiteClientId` Optional Client ID to include along with Graphite historical metrics.

Type: string

Default: *<empty-string>*

B Command-Line Interface

Connect includes a `usermanager` command for some basic management tasks. This utility helps you list users and modify user roles in the event that no one can access a Connect administrative user account.

Connect's `usermanager` CLI also includes the ability to dump audit logs. By default, the logs are displayed in a formatted table, but you can also choose to output comma-separated values for easy analysis in other tools.

The user management utility is installed at `/opt/rstudio-connect/bin/usermanager`. It uses the configuration defined in `/etc/rstudio-connect/rstudio-connect.gcfg` unless you specify an alternate configuration file with the `--config` flag.

The `usermanager` utility must be run as `root`.

The `usermanager` utility can only be run when Connect is stopped if you use the SQLite database provider. See Section 5.1 for information on stopping and restarting Connect. See Section 9 for information on database providers.

B.1 Commands

The `usermanager` utility supports the following commands:

- `list`: Lists users
- `alter`: Changes a user's role
- `audit`: Dumps audit logs

B.2 Flags

Configuration for `usermanager`:

- `--config`: The full or relative path to a Connect configuration file (`.gcfg`). Defaults to `/etc/rstudio-connect/rstudio-connect.gcfg`.

Flags for the `list` command:

- `--include-locked`: Includes locked user accounts in the list.

Flags for the `alter` command:

- `--username`: Specifies the user name of the user to alter.
- `--role`: Specifies the role to set for the user. Allowed roles are `viewer`, `publisher`, and `administrator`.
- `--force`: Force demotion of the last remaining administrator.

Flags for the `audit` command:

- `--csv`: Output comma-separated values

B.3 Examples:

Display help:

```
./bin/usermanager help
```

List unlocked users:

```
sudo ./bin/usermanager list
```

List all users (locked and unlocked):

```
sudo ./bin/usermanager list --include-locked
```

Specify a custom configuration file

```
sudo ./bin/usermanager --config /etc/connect/mycustomconfig.gcfg list
```

Promote the user `john` to an administrator role

```
sudo ./bin/usermanager alter --username john --role administrator
```

Demote the last remaining administrator to a non-administrative role

```
sudo ./bin/usermanager alter --username admin --role publisher --force
```

Dump audit logs to screen

```
sudo ./bin/usermanager audit
```

Dump audit logs (comma-separated) to a file:

```
sudo ./bin/usermanager audit --csv > ~/audits.txt
```

C Using a Custom Landing Page

C.1 Overview

It is possible to specify a custom landing page that your anonymous/logged-out users will see when they visit Connect.

C.2 Configuration

Use the `Server.LandingDir` configuration setting to specify the path to a custom landing page. If you do not specify an absolute path, the server will resolve the path starting at your Connect server installation directory (probably `/opt/rstudio-connect`).

Please see A for more information on the `Server.LandingDir` setting.

C.3 Custom Landing Page Assets

Include all assets (JavaScript, CSS, images, etc.) for your custom landing page in the directory you specified in the `Server.LandingDir` configuration setting. Be sure to include an `index.html`, which will be served by default.

C.4 Example

See the `/opt/rstudio-connect/examples/landing-page` directory for an example custom landing page. You can enable this example landing page by adding the following configuration setting and restarting the Connect server.

```
[Server]
LandingDir = examples/landing-page
```

D LDAP/AD Configuration Examples

This section contains sample RStudio Connect configurations to help you get started with LDAP authentication. We have provided a single bind and a double bind example (double bind is recommended).

The LDIF file contained in D.3 describes a LDAP organization used in our examples.

D.1 Single Bind

Here is a partial RStudio Connect configuration file showing how to connect using single-bind LDAP authentication. We are assuming the LDIF contained in D.3 describes the LDAP structure.

```
# using single bind
[LDAP "myLDAPserverSingle"]
ServerAddress = 127.0.0.1:389
UserSearchBaseDN = "ou=People,dc=company,dc=com"
UserObjectClass = posixAccount
UserFirstNameAttribute = givenName
UserLastNameAttribute = sn
UserEmailAttribute = mail
UsernameAttribute = uid
```

D.2 Double Bind

Here is a partial RStudio Connect configuration file showing how to connect using double bind LDAP authentication. We are assuming the LDIF contained in D.3 describes the LDAP structure.

```
# using double bind
[LDAP "myLDAPserver"]
ServerAddress = 127.0.0.1:389
BindDN = cn=admin,dc=company,dc=com"
BindPassword = "password"
UserSearchBaseDN = "ou=People,dc=company,dc=com"
UserObjectClass = posixAccount
UserFirstNameAttribute = givenName
UserLastNameAttribute = sn
UserEmailAttribute = mail
UsernameAttribute = uid
```

D.3 LDIF

Here is an LDIF (LDAP Data Interchange Format) file describing a hypothetical organization.

```
dn: ou=People,dc=company,dc=com
objectClass: organizationalUnit

dn: ou=Groups,dc=company,dc=com
objectClass: organizationalUnit

dn: cn=membera-grp,ou=Groups,dc=suba,dc=company,dc=com
objectClass: posixGroup
cn: membera-grp
gidNumber: 50000
memberUid: membera

dn: cn=memberb-grp,ou=Groups,dc=subb,dc=company,dc=com
objectClass: posixGroup
cn: memberb-grp
gidNumber: 50001
memberUid: memberb

dn: cn=memberc-grp,ou=Groups,dc=subc,dc=company,dc=com
objectClass: posixGroup
cn: memberc-grp
gidNumber: 50002
```

```
memberUid: memberc

dn: uid=membera,ou=People,dc=suba,dc=company,dc=com
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: membera
sn: A
givenName: Member
cn: Member A
displayName: Member A
uidNumber: 20000
gidNumber: 50000
userPassword: memberaldap
gecos: MemberA
loginShell: /bin/bash
homeDirectory: /home/membera
mail: membera@company.com

dn: uid=memberb,ou=People,dc=subb,dc=company,dc=com
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: memberb
sn: B
givenName: Member
cn: Member B
displayName: Member B
uidNumber: 20001
gidNumber: 50001
userPassword: memberbldap
gecos: MemberB
loginShell: /bin/bash
homeDirectory: /home/memberb
mail: memberb@company.com

dn: uid=memberc,ou=People,dc=subc,dc=company,dc=com
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: memberc
sn: C
givenName: Member
cn: Member C
displayName: Member C
uidNumber: 20002
gidNumber: 50002
userPassword: membercldap
gecos: MemberC
loginShell: /bin/bash
homeDirectory: /home/memberc
mail: memberc@company.com
```

E RStudio Connect Deployment Guide

E.1 Overview

This guide will cover the details of the deployment process in RStudio Connect. For most users, these details can be safely ignored, as the details are handled automatically via push-button publishing. However, some users may want to programmatically publish content using the rsconnect package or may have run into an error during deployment.

E.2 Programmatic Deployment

To programmatically publish content to RStudio Connect, use the functions `deployDoc`, `deployApp`, `deployApi`, and `deploySite` from the rsconnect package. Each of these functions will require a user account and a connected server. To setup an account on a server use `addConnectServer` and `connectUser`. To view currently configured accounts use `accounts`. For more details visit the rsconnect reference pages.

Each of the deployment functions listed above can be supplied with optional arguments. If additional arguments are not supplied, defaults are determined based on the content being deployed. All of the deployment functions follow a similar, underlying process. This appendix explains the process in detail.

E.3 Step 1: Building the Bundle

Connect builds an application bundle for the deployed content. The bundle contains the source code, any data files, and a manifest (JSON file) with metadata about the bundle and environment.

E.3.1 Application Metadata

rsconnect infers a number of attributes about the content including:

1. `appMode`: static, shiny, rmd-static, rmd-shiny, api
2. `hasParameters`: whether or not the R Markdown file includes parameters

In the case of an R Markdown document the YAML is parsed. Otherwise, .R files are flagged as shiny applications, html files and pdf files are flagged as static. (When a plot is published, the plot is wrapped in an html file).

E.3.2 List of Target Files

Next, rsconnect identifies the relevant files for the application. `appFiles` or `appFileManifest` can be passed as arguments to `deployApp` to specify the required files. Otherwise, rsconnect attempts to identify the required files using a number of heuristics.

For R Markdown documents and static HTML files, external dependencies are discovered using the rmarkdown function `find_external_resources`. This function searches for dependencies in the R Markdown file and the rendered HTML file. The function is able to identify files in the YAML header (if a parameter is a file), logos, images, data files used within R code chunks, and HTML dependencies. This process includes a minimal, client-side "render" of the document (the Rmd is not rendered, it is converted to plain markdown and then rendered to HTML without running any R code). Think of this rendering as creating a skeleton of the final HTML document. During push-button deployment, this initial "render" will show up in the IDE R Markdown tab.

The dependencies for R Markdown websites are identified uniquely. Websites should be deployed by calling `deploySite`.

Troubleshooting: To avoid client side rendering, deploy the content directly using `deployApp` with `appFiles` or `appFileManifest`.

For Shiny applications and Plumber APIs, rsconnect adds all the files in the project directory and subdirectories with a few exceptions: .Rproj file, the packrat directory, and the rsconnect directory. Files are added up to the specified max bundle size: `getOption("rsconnect.max.bundle.size")`.

Troubleshooting: try `rsconnect::listBundleFiles(appDir)` to see the identified dependencies

E.3.3 Lint

After identifying the target file and dependency files, rsconnect applies a series of linters. The rsconnect linters attempt to identify common problems that might prevent an application that works locally from working after deployment. These checks ensure the application code does not contain:

1. absolute paths
2. invalid relative paths
3. inconsistent capitalization among paths (the Connect server has case sensitive file paths)

The linters currently **do not** check for database connections.

Troubleshooting: You can disable the linters by passing `lint=FALSE` to the deployment function.

E.3.4 Create Temporary Folder

If the files pass the linters, RStudio Connect creates the initial bundle by copying all of the files to a temporary directory.

E.3.5 Library Dependencies

Next, rsconnect attempts to identify the package dependencies required by the app. (This step is skipped for static content). rsconnect does this by using packrat. Packrat is a dependency management tool for R designed to keep projects isolated, portable, and reproducible. rsconnect deployment does not use all of packrat's functionalities. (For example, the package sources are not installed on the client in the project's packrat subdirectory). For more information visit: https://github.com/rstudio/packrat

Packrat looks through the R code and makes note of any `library()` or `require()` calls. Packrat creates a list of the required packages and saves the list in the `packrat.lock` file. This lock file includes the package version and package dependencies. This process is recursive. In addition, the lock file also includes information on the version of R being used, the type of repository containing the package, and the specific URI for each type of repository. A few notes about this process:

Packrat searches in the order of `.libPaths`

For example, if the code includes `library(babynames)`, Packrat will look for babynames inside the first library in `.libPaths`. Imagine there are two libraries: A and B and `.libPaths(A,B)`. In A, babynames is version 1.0. In B, babynames is version 2.0. Packrat will assume the app depends on version 1.0. To understand this behavior, recall that a library is just a folder containing an installed R package. The most common scenario where this occurs is when the target directory is part of an existing packrat project.

Repositories

Most packages come from CRAN. In the packrat lockfile, packrat will record the names of packages originating from CRAN as well as a specific URL for CRAN (i.e. CRAN='https:cran.rstudio.com'). The url is determined by the state of `options("repos")` during deployment. The same process is used for other repositories: Github, BioConductor, and local repositories. In the case of a local repository, the repository URI may be a location on disk.

For the edge case of an internal package from a local repository, be sure the package's Repository option (found in the package's Description file) is mapped to a repo URI in the current `options("repos")`. For example, imagine a package called myPackage is stored in a local repo called myRepo. The myPackage Description file should include `repository:myRepo`. `options("repos")` should define a URI for myRepo during deployment runtime, i.e. `options(repos = list(myRepo="file://path_to_private_repo"))`.

> Troubleshooting: try `rsconnect:::performPackratSnapshot(appDir)`. This command will create the packrat lock file helping to identify the dependencies, corresponding repos, and URLs expected for deployment.

Once the lock file is created, rsconnect proceeds to copy all of the description files for the packages listed in the packrat lock file. The files are copied into `packrat/desc`. Normally, a packrat lockfile would be enough to fully reproduce the package environment. This additional step is necessary just in case the version of packrat on the client is significantly different from the version on the server.

E.3.6 Manifest

Next, rsconnect generates the actual manifest. This manifest includes a list of the relevant source code, package dependencies, and other metadata including the R version, the locale, the app mode, content category, etc. The R version is determined while building the manifest. The R version listed in the manifest will later be used by Connect to attempt to re-create a server-side environment consistent with the client. While creating the manifest, rsconnect will also attempt to determine the primary document (if not already listed). Checksums are stored for each file, including the packrat description files. Finally, the manifest is copied to the temporary bundle directory alongside the code and packrat directory.

For example, a target directory with the structure:

```
targetDir
- app.R
+ dataDir
  - data.csv
```

where `app.R` includes:

```
library(babynames)
library(shiny)
```

The final bundle will contain:

```
bundleDir
  - app.R
  - manifest.json
  - index.htm
  + dataDir
    - data.csv
  + packrat
    - packrat.lock
    + desc
    - babynames
    - shiny
    ...
```

The manifest.json file will include:

```
{
    "version" : 1,
    "locale" : "en_US",
    "platform" : "3.2.5",
```

```
    "metadata" : {
        "appmode" : "shiny",
        "primary_rmd" : null,
        "primary_html" : null,
        "content_category" : "application",
        "has_parameters" : false
    },
    "packages" : {
        ...
    },
    "files" : {
        "app.R" : {
            "checksum" : "bc81fad5645566fe5d228abf57bba444"
        },
        "packrat/desc/babynames" : {
            "checksum" : "ee14db463dc57f078fea1c3d74628104"
        },
        ...
    },
}
```

The `packages` entry will contain a version of each package's DESCRIPTION file. The `files` entry will include a checksum for each package description file.

> Troubleshooting: try `rsconnect::bundleApp(appDir, appFiles=rsconnect::listBundleFiles(appDir), ...)`. This command will generate a tarball containing the application bundle.

E.4 Step 2: Push Bundle to Connect

In step 2 rsconnect publishes the bundle to the server. This is done with a POST request to an HTTP endpoint. rsconnect supports multiple protocols for making HTTP requests. rsconnect looks for the server address and account information created when the IDE is linked to Connect. Publisher privileges are required for a user to link the IDE to Connect and publish content. These privileges are checked when the user sets up an account for publishing (this process creates a public-secret key pair unique to the user and Connect server).

> Troubleshooting: try `rsconnect:accounts()`

When an application bundle is successfully deployed, rsconnect generates a folder in the original target directory called rsconnect. This folder contains a DCF file with information on the deployed content (i.e. the name, title, server address, account, URL, and time). If you re-deploy the same directory, rsconnect checks for this file allowing the deployed content to be updated. Redeployments will deploy and activate the new bundle for this application. You may use the "source versions" menu option in the dashboard to revert the application to a previous bundle. Redeployment will only work if the document is the same content type. For instance, you can not redeploy an R Markdown document after adding `runtime:shiny`. Instead, deploy the document to a new endpoint by changing the `appName`.

Currently, each deployed application is tied to an account. For example, imagine user1 deploys an app and shares the code with user2. If user2 deploys the app, a new copy of the app would be deployed. This is true even if user1 shares the rsconnect folder. (The only way for a different collaborator to deploy to the same app is for both collaborators to use a service account where the username and password are shared by both users. Both users would also need to go through the steps that link the IDE to Connect - generating the public-private keypair).

In some occasions, a single user will have multiple accounts on one server, or an account on multiple servers. To deploy a bundle to a different server or under a different account, specify the account and user parameters in the `deployApp` function. After successful deployment, a new DCF file will be added to the rsconnect folder. If you deploy the same content from a new machine to the server, using the same account, rsconnect will prompt you asking whether or not the content is a redeploy. This occurs even if the rsconnect folder does not exist on the new machine.

E.5 Step 3: Bundle is deployed on Connect

Once the bundle is published to the server, Connect prepares the content to be deployed. This process follows a number of steps:

E.5.1 Parse the Manifest

The bundle is uncompressed at a unique location (assigned based on appid and bundle id). The manifest from the uncompressed bundle is parsed to determine the type of content. The R version is also identified and matched based on the available R versions on the Connect server. You can find more details here. Files are checked against the checksum listed in the manifest to ensure content was not lost or corrupted during transfer.

E.5.2 Packrat Restore

Packrat is used to ensure the required packages are available. For every package identified in the manifest:

Packrat checks to see if the required package is available in the global cache. (The cache is specific to the version of R matched previously).

If the package is available, a symlink is created that points to the package within the global cache. If a symlink is not possible, the package will be copied from the global cache.

If the package is not available, packrat attempts to install the package. The package is requested from the repo URL identified during bundling. The package is installed and built from source and the installed package is added to the global cache.

> Many R packages have system-level dependencies (Java, openssl, etc). If the package fails to install, be sure these system dependencies are installed and available.

All packages are installed as the default `[Applications].RunAs` user (typically rstudio-connect). Connect ensures that the package libraries and uncompressed bundle have the appropriate permissions based on the application specific `RunAs` user.

E.5.3 R Markdown Render

If the deployed content is an R Markdown document (excluding documents with `runtime:shiny`) the Rmd file is rendered on the server. If the document is parameterized, the default parameters are used.

The application is presented as deployed. User input is currently required to publish the application and specify any server-side attributes (such as tuning runtime settings, permissions, etc).

E.6 Other Frequently Asked Questions

1. My app deployed but does not run?

If the application is deployed but does not run, the error message will be caught and displayed in the application log (visible at the app url in Connect on the logs panel).

2. Can I get more details about the deployment failure?

Yes, set the option "Show diagnostic information after publishing" in `Tools -> Global Options -> Publishing`

3. Will database connections work once deployed?

Database connections will only work if the same drivers (and potentially DSNs) are available on the client and on Connect. At this time there is not a linter to check for connection strings.

4. I use a specific distribution of R (i.e. MRO). Will matching work?

The version of R written to the manifest will be the version used during runtime.

On the server side, Connect attempts to match the version of R in the manifest as described here.

Currently Connect only matches based on the version - no other supplemental information (such as distribution) is maintained. For that reason, to ensure a specific distribution is used on the server, install only that distribution for the desired version.

5. Are bundles compressed?

Bundles are not be compressed. Bundles do not need to be read completely into RAM during deployment. Typically the only bottleneck is upload speed. You can specify a maximum bundle size using: `getOption("rsconnect.max.bundle.size")`.

F Using Continuous Integration to Deploy Content

F.1 Overview

It is possible to use the `rsconnect` R package to programmatically deploy content to a Connect server. This is particularly useful when combined with a continuous integration (CI) server that builds and deploys your content.

F.2 Prerequisites

Currently, it is only feasible to to use a CI server to update content that you originally published from the same server. You cannot update content that you published from elsewhere. To clarify, the CI server must perform both the initial deployment and subsequent updates of the application.

Configuring a CI server to deploy content with `rsconnect` requires that you log in to the CI server with the credentials the CI server uses to run `rsconnect`. For example, if your CI server uses the `jenkins` account, you need to log in as `jenkins` to configure `rsconnect` for the CI server.

You must be familiar with deploying content with `rsconnect`. Please see E for more information.

F.3 Configuring a CI Server to Deploy Content to Connect

F.3.1 Installing rsconnect

The `rsconnect` package is used to deploy content to Connect. Install it with the following command in the R console. In practice, `rsconnect` may already be available.

```
install.packages("rsconnect")
```

F.3.2 Configuring rsconnect

> Configuring `rsconnect` requires a user home directory. In this use case, a valid home directory is required for the `jenkins` user account.

You must configure `rsconnect` for the user account that will be used by the CI server to deploy content with `rsconnect`. In this document, we assume that this user is `jenkins`.

```
sudo su jenkins
```

Next, while running as `jenkins`, run `R` and issue the following commands in the R console:

```
library(rsconnect)
addConnectServer("http://myserveraddress:3939", "mylocaldeployserver")
connectUser(server="mylocaldeployserver")
```

> The `rsconnect` server name, `mylocaldeployserver`, is an arbitrary name that is used to identify a Connect server when using rsconnect. You can choose any name you wish.

After the last command, you will see output similar to this:

> A browser window should open; if it doesn't, you may authenticate manually by visiting http://myserveraddress:3939/___login___?url=http%3A%2F%2Fmyserveraddress%3A3939%2Fconnect%2F%23%2Ftokens%2FTc8f636c59ffff521eef4888b163dcf64%2Factivate&user_id=0.
>
> Waiting for authentication...

Copy the URL in the output above, then paste it into a Web browser and authenticate with the Connect user credentials for your CI server. In this example, we assume that you wish to deploy content with the `ci-server` Connect acount.

After successfully connecting the `ci-server` Connect account to `rsconnect`, you will see this message at the R console:

```
Account registered successfully: CI Server (ci-server)
```

The server and account information are persisted to configuration files on the server in the `jenkins` user's home directory:

```
/home/jenkins/.config/R/connect/servers/mylocaldeployserver.dcf
/home/jenkins/.config/R/connect/accounts/mylocaldeployserver/connectuser.dcf
```

F.3.3 Deploying Content with rsconnect

Now `rsconnect` is configured to use the `ci-server` Connect account when running with the `jenkins` server account.

F.3.4 Package and R Version Compatibility

`rsconnect` will use the package libraries and the R installation available on the CI server to create the manifest used by Connect. It is crucial that the environment on the CI server is compatible with the content you are deploying. Ideally, you should maintain the same R version, the same available packages, and the same package versions that you use in development.

F.3.5 CRAN Note

If you don't already have it in an `.RProfile`, be sure to specify a default CRAN repository in your application before issuing the `rsconnect` command to deploy content. For example:

```
options(repos=c(CRAN="https://cran.rstudio.com"))
deployDoc(doc="out.Rmd", appName="ServerDeployedDoc",
                      account="ci-server", server="mylocaldeployserver")
```

Please note that Connect content must be published before it is publicly available. This means that you must log in to Connect and publish the content after the initial deployment. Subsequent automated deployments of the same content are automatically published and require no manual intervention.

F.4 Warning and Security Information

A CI server account that is configured to deploy content to Connect can deploy additional content to Connect without further authentication.

For example, Bob logs in to a server console as Unix user `jenkins`, which is the account used by his CI server. Bob then configures `rsconnect` to deploy content. During the authorization step, Bob signs in to Connect as a publisher with user name `ci-server`. Now, any other CI processes running on this server under the `jenkins` user account can deploy additional content using the Connect user `ci-server`.

G Programmatic Deployment with rsconnect

G.1 Overview

It is possible to use the `rsconnect` R package to programmatically deploy content to a Connect server. Furthermore, Connect-hosted content can use `rsconnect` to deploy additional content to itself or to another Connect server.

Configuring Connect to deploy content with `rsconnect` requires:

1. administrator privileges for Connect, and
2. sudo or root privileges on the server where Connect is installed.

G.2 Use Case: A Shiny Application

Here we present a use case that explains how to configure Connect for programmatic deployment. Please see G.4 for an example Shiny application for this use case.

G.2.1 Use Case Scenario

Bob White develops a Shiny application (see G.4) that:

1. Renders an R Markdown document.
2. Deploys the generated document using `rsconnect`

Bob deploys his Shiny application to Connect. The application, as noted above, can automatically deploy documents it generates to Connect. However, the Connect server must first be configured to authorize deployment from `rsconnect`.

G.2.2 Installing rsconnect

The `rsconnect` package is not yet available on Bob's Connect server, so Bob installs it by running R as root (`sudo R`) and issuing the following command in the R console. In practice, `rsconnect` may already be available.

```
install.packages("rsconnect")
```

G.2.3 Configuring a Custom "RunAs" User

Since Bob does not want to allow arbitrary Connect users to deploy content using `rsconnect`, he configures a custom `RunAs` user, `robert`, for his Shiny application. See Section 12.3 for configuring the `RunAs` user on a per-application basis in Connect.

G.2.4 Configuring rsconnect

Important Note: `rsconnect` configuration requires a user home directory. In this use case, a valid home directory is required for the `robert` user account.

Since Bob's Shiny application will be running as the `robert` user, Bob (at a server console) switches to the `robert` user:

```
sudo su robert
```

Next, while running as `robert`, Bob runs `R` and issues the following commands in the R console:

```
library(rsconnect)
rsconnect::addConnectServer('http://myserveraddress:3939', 'mylocaldeployserver')
rsconnect::connectUser(server='mylocaldeployserver')
```

> NOTE: the rsconnect server name, `mylocaldeployserver`, is an arbitrary name that is used to identify a Connect server when using rsconnect. You can choose any name you wish.

After the last command, Bob sees the following output:

> A browser window should open; if it doesn't, you may authenticate manually by visiting http://myserveraddress:3939/___login___?url=http%3A%2F%2Fmyserveraddress%3A3939%2Fconnect%2F%23%2Ftokens%2FTc8f636c59ffff521eef4888b163dcf64%2Factivate&user_id=0.
>
> Waiting for authentication...

Bob copies the URL in the output above and pastes it into a Web browser. Then Bob authenticates with his Connect user credentials. Bob's Connect user name (with publishing privileges) is `rwhite`.

After successfully connecting his Connect account to `rsconnect`, Bob sees this message at the R console:

> Account registered successfully: Bob White (rwhite)

The server and account information are persisted to configuration files on the server in Bob's home directory:

```
/home/robert/.config/R/connect/servers/mylocaldeployserver.dcf
/home/robert/.config/R/connect/accounts/mylocaldeployserver/connectuser.dcf
```

G.2.5 Deploying Content with rsconnect

Now `rsconnect` is configured to use the `rwhite` Connect account when running with the `robert` server account. If Bob's Shiny application uses `robert` as its `RunAs` user, it can deploy content using `rsconnect`.

G.2.6 CRAN Note

If you don't already have it in an `RProfile`, be sure to specify a default CRAN repository in your application before issuing the `rsconnect` command to deploy content. For example:

```
options(repos=c(CRAN="https://cran.rstudio.com"))
rsconnect::deployDoc(doc="out.Rmd", appName="ServerDeployedDoc",
                     account="rwhite", server="mylocaldeployserver")
```

G.3 Warning and Security Information

Please restrict access to any Connect content that can deploy arbitrary content via `rsconnect`. The Connect Dashboard's "Permissions" document provides details on securing content in Connect.

Do not enable deployment via `rsconnect` for the default `Applications.RunAs` user; if you do so, all your Connect users will be able to deploy content using your `rsconnect` credentials.

Once a Connect user authorizes `rsconnect` to deploy content under a particular server account, any content that runs under that server account can use `rsconnect` to deploy content without further authentication.

For example, Bob logs in to a server console as Unix user `robert`. Bob then configures `rsconnect` to deploy content. During the authorization step, Bob signs in to Connect as a publisher with user name `rwhite`. Now, any Connect application that is configured with a `RunAs` user of `robert` can deploy additional content using the Connect user `rwhite`, regardless of who owns the application.

G.4 Example Shiny Application

Below is an example Shiny application that knits R Markdown text and deploys the resulting content using `rsconnect`.

```
library(knitr)
library(rsconnect)
library(shiny)
library(shinyAce)
library(rmarkdown)

# Default text for editor
defaultMarkdown <- '
### Sample R Markdown
This is some markdown text. It may also have embedded R code
which will be executed.
'

# A Shiny UI for editing R Markdown
ui <- shinyUI(
  bootstrapPage(
    headerPanel("Embedded Deployment Example"),
    div(
      class="container-fluid",
      div(class="row-fluid",
          div(class="col-sm-6",
              h2("Source R-Markdown"),
              aceEditor("rmd", mode="markdown", value=defaultMarkdown),
              actionButton("eval", "Update")
```

```
          ),
          div(class="col-sm-6",
              h2("Knitted Output"),
              htmlOutput("knitDoc")
          )
      )
    )
  )
)

# A Shiny application that generates and deploys R Markdown content
server <- shinyServer(function(input, output, session) {

  # Only update and deploy when the 'Update' button is clicked
  rmd <- eventReactive(input$eval, {
    input$rmd
  })

  output$knitDoc <- renderUI({
    writeLines(rmd(), "out.Rmd")
    knit2html(input="out.Rmd", fragment.only = TRUE, quiet = TRUE)
    options(repos=c(CRAN="https://cran.rstudio.com"))
    rsconnect::deployDoc(doc="out.Rmd", appName="GeneratedDoc",
                         account="rwhite", server="mylocaldeployserver")
    return(isolate(HTML(
      readLines("out.html")
    )))
  })
})

# Run the application
shinyApp(ui = ui, server = server)
```