RStudio Server Administrator's Guide

RStudio Server Professional v0.99.482

Contents

Chapter 1

Getting Started

1.1   Introduction

RStudio Server enables you to provide a browser based interface (the RStudio IDE) to a version of R running on a remote Linux server. Deploying R and RStudio on a server has a number of benefits, including:

- The ability to access R sessions from any computer in any location;
- Easy sharing of code, data, and other files with colleagues;
- Allowing multiple users to share access to the more powerful compute resources (memory, processors, etc.) available on a well equipped server; and
- Centralized installation and configuration of R, R packages, TeX, and other supporting libraries.

This manual describes *RStudio Server Professional Edition*, which adds many enhancements to the open-source version of RStudio Server, including:

- An administrative dashboard that provides insight into active sessions, server health, and monitoring of system-wide and per-user performance and resource metrics;
- Authentication using system accounts, ActiveDirectory, LDAP, or Google Accounts;
- Full support for PAM (including PAM sessions for dynamically provisioning user resources);
- Ability to establish per-user or per-group CPU priorities and memory limits;
- HTTP enhancements including support for SSL and keep-alive for improved performance;
- Ability to restrict access to the server by IP;
- Customizable server health checks; and
- Suspend, terminate, or assume control of user sessions; Impersonate users for assistance and troubleshooting.

1.2   Installation

1.2.1   Debian (8+) / Ubuntu (12.04+)

Installing R

RStudio requires a previous installation of R version 2.11.1 or higher. To install the latest version of R you should first add the CRAN repository to your system as described here:

- Debian: http://cran.rstudio.com/bin/linux/debian/README.html
- Ubuntu: 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 Debian or Ubuntu repository as described above this command will install the version of R corresponding to your current system version. Since 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.

Installation Commands

After downloading the appropriate Debian/Ubuntu package for RStudio Server Professional you should execute the following commands to complete the installation:

```
$ sudo apt-get install gdebi-core
$ sudo gdebi <rstudio-server-package.deb>
```

1.2.2   RedHat / CentOS (5+)

Prerequisites

RStudio Server has some 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 you should add it to your system using the instructions found here: https://fedoraproject.org/wiki/EPEL

After enabling EPEL you should then ensure that you have installed the version of R available from EPEL. You can do this using the following command:

```
$ sudo yum install R
```

Installation Commands

After downloading the appropriate RedHat/CentOS package for RStudio Server Professional you should execute the following command to complete the installation:

```
sudo yum install --nogpgcheck <rstudio-server-package.rpm>
```

1.2.3   openSUSE / SLES (11+)

Installing R

You can install R for openSUSE or SLES using the instructions on the CRAN SUSE download page: http://cran.rstudio.com/bin/linux/suse/.

Note that the binaries linked to from this page have one additional requirement that isn't satisfied using the default repositories. Before installing R you should install the `libgfortran43` package. This package is available from the SUSE Linux Enterprise SDK. If the SDK repository is available in your environment you can install `libgfortran43` as follows:

```
$ sudo zypper install libgfortran43
```

Installation Commands

After downloading the appropriate RPM package for RStudio Server Professional you should execute the following command to complete the installation:

```
$ sudo zypper install <rstudio-server-package.rpm>
```

1.3   Management Script

RStudio Server management tasks are performed using the `rstudio-server` utility (installed under `/usr/sbin`). This utility enables the stopping, starting, and restarting of the server, enumeration and suspension of user sessions, taking the server offline, as well as the ability to hot upgrade a running version of the server.

For example, to restart the server you can use the following command:

```
$ sudo rstudio-server restart
```

Note that on some systems (including RedHat/CentOS 5 and SLES 11) the `sudo` utility doesn't have the `/usr/sbin` directory in it's path by default. For these systems you can use a full path to the management script. For example:

```
$ sudo /usr/sbin/rstudio-server restart
```

1.4   Activation

After completing the installation steps described in the previous section you may need to activate the product before using it. Alternatively, if you haven't previously installed RStudio Server on a system then it will run in evaluation mode for a period of time before requiring activation. To determine the current license status of your system you can use the following command:

```
$ sudo rstudio-server license-manager status
```

To activate the product you obtain a product key and then use the following commands:

```
$ sudo rstudio-server license-manager activate <product-key>
$ sudo rstudio-server restart
```

Note that you need to restart the server in order for licensing changes to take effect.

Additional details on license management (including discussions of offline activation and activating through a proxy server) can be found in the License Management section.

1.5 Accessing the Server

1.5.1 Logging In

By default RStudio Server runs on port 8787 and accepts connections from all remote clients. After installation you should therefore be able to navigate a web browser to the following address to access the server:

```
http://<server-ip>:8787
```

RStudio will prompt for a username and password and will authenticate access using the PAM authentication scheme configured for the server. Some notes related to user authentication:

- RStudio Server will not permit logins by system users (those with ids < 100).
- By default on Debian/Ubuntu the system default PAM profile (`/etc/pam.d/other`) will be used (this can be customized by creating an RStudio PAM profile at `/etc/pam.d/rstudio`).
- By default on RedHat/CentOS and SLES an RStudio PAM profile (`/etc/pam.d/rstudio`) that authenticates using the system username/password database will be used (this can be customized by editing the profile as appropriate).
- User credentials are encrypted using RSA as they travel over the network.

Additional details on customizing RStudio Server authentication are provided in Authenticating Users. Details on customizing the port and enabling SSL are covered in Access and Security.

1.5.2 Troubleshooting Problems

If you are unable to access the server after installation, you should run the `verify-installation` command to output additional diagnostics:

```
$ sudo rstudio-server verify-installation
```

This command will start the server and run and connect to an R session. Note that this will test the correct installation of RStudio Server and ensure that it can connect to a locally installed version of R. However, it won't test whether networking or authentication problems are preventing access to the server.

If problems persist, you can also consult the system log to see if there are additional messages there. On Debian/Ubuntu systems this will typically be located at:

```
/var/log/syslog
```

On RedHat/CentOS systems this will typically be located at:

```
/var/log/messages
```

Chapter 2

Server Management

2.1 Core Administrative Tasks

2.1.1 Configuration Files

RStudio Server uses several configuration files all located within the `/etc/rstudio` directory. Configuration files include:

`rserver.conf`	Core server settings
`rsession.conf`	Settings related to individual R sessions
`profiles`	User and group resource limits
`ip-rules`	IP access rules (allow or deny groups of IP addresses)
`load-balancer`	Load balancing configuration
`health-check`	Template for content to return for server health checks
`google-accounts`	Mappings from Google accounts to local accounts

The `rserver.conf` and `rsession.conf` files are created by default during installation however the other config files are optional so need to be created explicitly.

Whenever making changes to configuration files you need to restart the server for them to take effect. You can do this using the `restart` command of the server management utility:

```
$ sudo rstudio-server restart
```

2.1.2 Stopping and Starting

During installation RStudio Server is automatically registered as a daemon which starts along with the rest of the system. On Debian, Ubuntu, and RedHat/CentOS 6 this registration is performed using an Upstart script at `/etc/init/rstudio-server.conf`. On other systems including RedHat/CentOS 5 an init.d script is installed at `/etc/init.d/rstudio-server`.

To manually stop, start, and restart the server you use the following commands:

```
$ sudo rstudio-server stop
$ sudo rstudio-server start
$ sudo rstudio-server restart
```

To check the current stopped/started status of the server:

```
$ sudo rstudio-server status
```

2.1.3 Managing Active Sessions

There are a number of administrative commands which allow you to see what sessions are active and request suspension of running sessions.

To list all currently active sessions:

```
$ sudo rstudio-server active-sessions
```

2.1.3.1 Suspending Sessions

When R sessions have been idle (no processing or user interaction) for a specified period of time (2 hours by default) RStudio Server suspends them to disk to free up server resources. When the user next interacts with their session it is restored from disk and the user resumes right back where they left off. This is all done seamlessly such that users aren't typically aware that a suspend and resume has occurred.

To manually suspend an individual session:

```
$ sudo rstudio-server suspend-session <pid>
```

To manually suspend all running sessions:

```
$ sudo rstudio-server suspend-all
```

The suspend commands also have a "force" variation which will send an interrupt to the session to request the termination of any running R command:

```
$ sudo rstudio-server force-suspend-session <pid>
$ sudo rstudio-server force-suspend-all
```

The `force-suspend-all` command should be issued immediately prior to any reboot so as to preserve the data and state of active R sessions across the restart.

2.1.4 Taking the Server Offline

If you need to perform system maintenance and want users to receive a friendly message indicating the server is offline you can issue the following command:

```
$ sudo rstudio-server offline
```

When the server is once again available you should issue this command:

```
$ sudo rstudio-server online
```

2.1.5   Upgrading to a New Version

If you perform an upgrade of RStudio Server and an existing version of the server is currently running, then the upgrade process will also ensure that active sessions are immediately migrated to the new version. This includes the following behavior:

- Running R sessions are suspended so that future interactions with the server automatically launch the updated R session binary
- Currently connected browser clients are notified that a new version is available and automatically refresh themselves.
- The core server binary is restarted

To upgrade to a new version of RStudio Server you simply install the new version. For example on Debian/Ubuntu:

```
$ sudo gdebi <rstudio-server-package.deb>
```

For RedHat/CentOS:

```
$ sudo yum install --nogpgcheck <rstudio-server-package.rpm>
```

For openSUSE / SLES:

```
$ sudo zypper install <rstudio-server-package.rpm>
```

2.2   Administrative Dashboard

RStudio Server includes an administrative dashboard with the following features:

1) Monitoring of active sessions and their CPU and memory utilization;
2) The ability to suspend, forcibly terminate, or assume control of any active session;
3) Historical usage data for individual server users (session time, memory, CPU, logs);
4) Historical server statistics (CPU, memory, active sessions, system load); and
5) Searchable server log (view all messages or just those for individual users)

The dashboard can be an invaluable tool in understanding server usage and capacity as well as to diagnose and resolve problems.

2.2.1   Enabling the Dashboard

The administrative dashboard is accessed at the following URL:

```
http://<server-address>/admin
```

The administrative dashboard is disabled by default. To enable it you set the `admin-enabled` option. You can also specify that only users of certain group have access to the dashboard using the `admin-group` option. For example:

/etc/rstudio/rserver.conf

```
admin-enabled=1
admin-group=rstudio-admins
```

Note that changes to the configuration will not take effect until the server is restarted.

2.2.2   Administrator Superusers

You can further designate a certain user or group of users as administrative "superusers". Superusers have the following additional privileges:

1) Suspend or terminate active sessions
2) Assume control of active sessions (e.g. for troubleshooting)
3) Login to RStudio as any other server user

Administrative superusers do not have root privilege on the system, but rather have a narrow set of delegated privileges that are useful in managing and supporting the server. You can define the users with this privilege using the `admin-superuser-group` setting. For example:

/etc/rstudio/rserver.conf

```
admin-superuser-group=rstudio-superuser-admins
```

Changes to the configuration will not take effect until the server is restarted.

2.2.2.1   Google Accounts Restrictions

Note that the ability to login as other users and assume control of existing sessions is not available if you are authenticating with Google Accounts. This is because Google authentication uses a different user-identity mechanism which isn't compatible with the way that user session impersonation is implemented.

Chapter 3

Authenticating Users

3.1   PAM Authentication

RStudio Server Professional Edition authenticates users via the Linux standard PAM (Pluggable Authentication Module) API. PAM is typically configured by default to authenticate against the system user database (`/etc/passwd`) however it can also be configured to authenticate against a wide variety of other systems including ActiveDirectory and LDAP.

The section describes the PAM configuration used for authentication by default after installation. Note that PAM can be used for both authentication as well as to tailor the environment for user sessions (PAM sessions). This section describes only authentication, see the [User Resources and Limits] section for details on how RStudio Server can be configured to use PAM sessions.

3.1.1   PAM Basics

PAM profiles are are located in the `/etc/pam.d` directory. Each application can have their own profile, and there is also a default profile used for applications without one (the default profile is handled differently depending on which version of Linux you are running).

To learn more about PAM and the many options and modules available for it see the following:

- http://en.wikipedia.org/wiki/Pluggable_authentication_module
- http://www.centos.org/docs/5/html/Deployment_Guide-en-US/ch-pam.html
- http://tldp.org/HOWTO/User-Authentication-HOWTO/x115.html
- http://linux.die.net/man/8/pam

3.1.2   Default PAM Configuration

Debian / Ubuntu

On Debian and Ubuntu systems RStudio Server does not provide an RStudio specific PAM configuration file. As a result, RStudio Server uses the `/etc/pam.d/other` profile, which by default inherits from a set of common configuration files:

/etc/pam.d/other

```
@include common-auth
@include common-account
@include common-password
@include common-session
```

If the `/etc/pam.d/other` profile reflects the authentication system and policies that you'd like RStudio Server to use then no further configuration is required. If you want to create a custom PAM profile for RStudio you would create a file named `/etc/pam.d/rstudio` and specify whatever settings are appropriate.

RedHat / CentOS / SUSE

On RedHat, CentOS and SUSE systems applications without their own PAM profiles are denied access by default. Therefore to ensure that RStudio is running and available after installation a default PAM profile is installed at `/etc/pam.d/rstudio`. This profile is configured to require a user-id greater than 500 and to authenticate users against local system accounts:

/etc/pam.d/rstudio

```
auth       requisite       pam_succeed_if.so uid >= 500 quiet
auth       required        pam_unix.so nodelay
account    required        pam_unix.so
```

This default PAM profile may not reflect the authentication behavior that you want for RStudio Server. In that case, some customization may be required. If you've already set up another PAM profile (e.g. `/etc/pam.d/login`) with the desired behavior then it may be enough to simply copy that profile over the RStudio one. For example:

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

3.1.3 Diagnosing PAM Authentication Problems

If you are unable to login to RStudio Server there may be an underlying problem with the PAM configuration. The best way to diagnose PAM configuration problems is to use the `pamtester` utility (which is bundled with RStudio Server). Using `pamtester` enables you to test authentication in an isolated environment as well as to see much more detailed diagnostics.

The `pamtester` utility is located at `/usr/lib/rstudio-server/bin/pamtester`. To invoke it you pass several arguments indicating the PAM profile to test, the user to test for, and whether you want verbose output. For example:

```
sudo /usr/lib/rstudio-server/bin/pamtester --verbose rstudio <username> authenticate
```

You can find more detailed documentation on using `pamtester` here: http://linux.die.net/man/1/pamtester.

3.1.4   Managing PAM Login Lifetimes

When logging in using PAM authentication users have am option to stay signed in across browser sessions. You can prevent this option from being shown by using the `auth-stay-signed-in` setting. For example:

/etc/rstudio/rserver.conf

```
auth-stay-signed-in=0
```

Setting this option to `0` will result in users being prompted to log in each time they start a new browser session (i.e. logins will only be valid as long as the browser process in which they originated in remains running).

3.2   Google Accounts

RStudio Server can be configured to authenticate users via Google Accounts. This enables users to login with their existing Gmail or Google Apps credentials and to be automatically authenticated to RStudio Server whenever they are already logged into their Google account.

3.2.1   Registering with Google
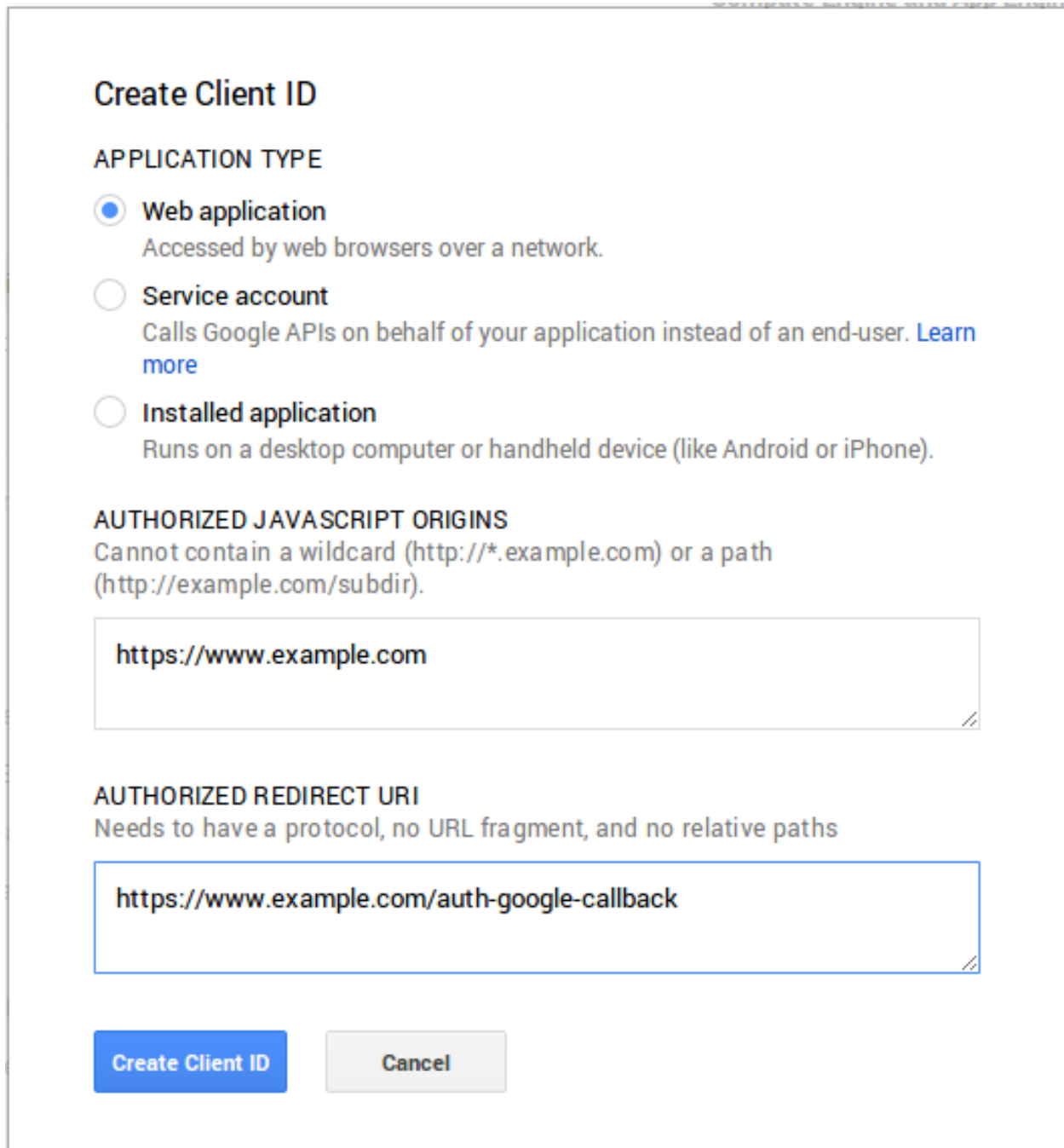
In order to use Google Accounts with RStudio Server you need to register your server with Google for OAuth 2.0 Authentication. You do this by creating a new "Project" for your server in the *Google Developer Console*:

https://console.developers.google.com/

Once you've created a project you go to the *Credentials* area of *APIs and auth* and choose to **Create New Client ID**:
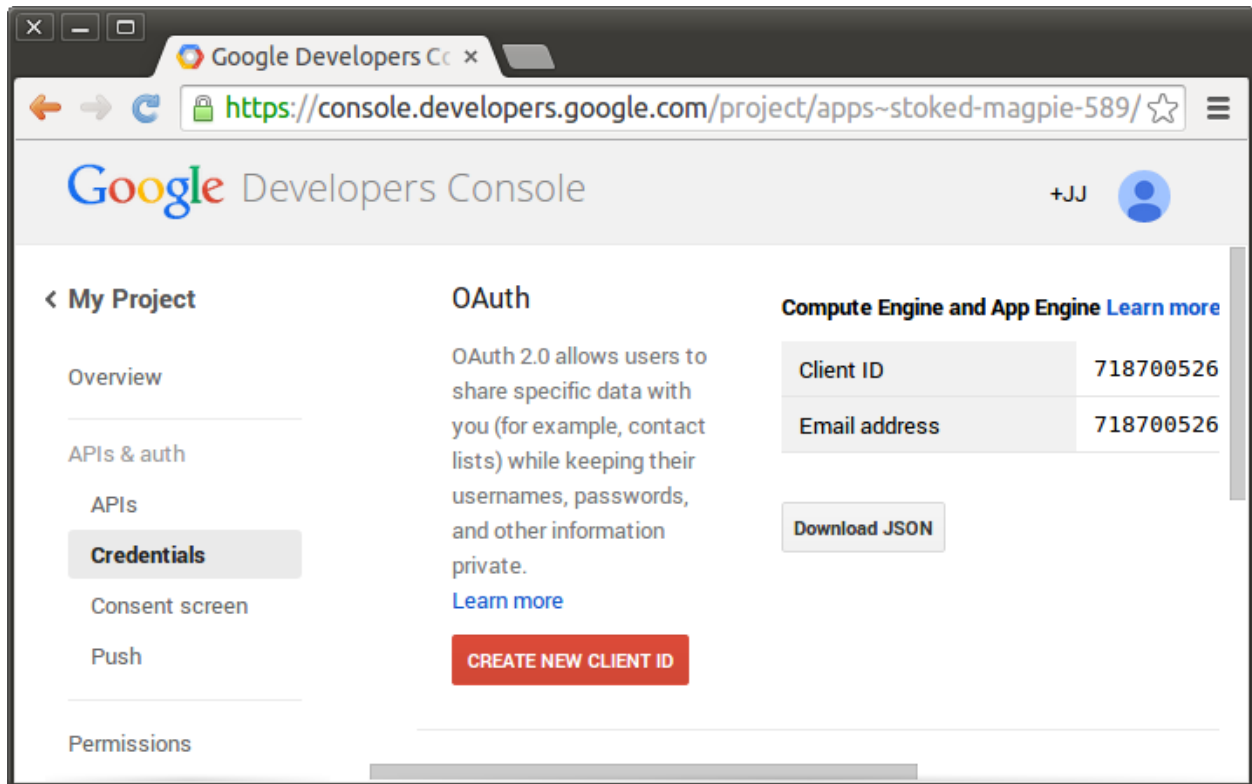
You'll then be presented with a dialog used to create a new client ID:

You should select "Web application" as the application type and provide two URLs that correspond to the server you are deploying on. The screenshot above uses `https://www.example.com` as the host, you should substitute your own domain and port (if not using a standard one like 80 or 443) in your configuration.

This will result in two values which you'll need to provide as part of the RStudio Server configuration: `client-id` and `client-secret` (they'll be displayed in the *Google Developer Console* after you complete the dialog).

3.2.2 Enabling Google Accounts

To enable authentication with Google Accounts you add the `auth-google-accounts` option to the RStudio Server configuration file:

/etc/rstudio/rserver.conf

```
auth-google-accounts=1
```

In addition, you need to add a configuration file (`/etc/rstudio/google-client-secret`) containing the `client-id` and `client-secret` that you received when registering your site with Google. For example, the configuration file might look like this:

/etc/rstudio/google-client-secret

```
client-id=llllllllllllll-xxxxxxxxxxxxxxxxxxxx.apps.googleusercontent.com
client-secret=BhCC6rK7Sj2ZtPH0ord7lO1w
```

The `/etc/rstudio/google-client-secret` file should have user read/write file permissions (i.e. 0600) to protect it's contents from other users. You can ensure this as follows:

```
$ sudo chmod 0600 /etc/rstudio/google-client-secret
```

Note that the above `client-id` and `client-secret` aren't the actual values you'll use. Rather, you should substitute the values that you obtained from Google when registering your site for OAuth authentication.

Once you enable authentication with Google Accounts that becomes the exclusive means of authentication (you can't concurrently use both PAM and Google Account authentication).

3.2.3 Translating to Local Accounts

3.2.3.1 Creating Matching Accounts

Once a user is authenticated via Google Accounts it's necessary to map their Google Accounts identity to a local system account. The default and most straightforward way to do this is to create a local account with a username identical to their Google email address.

If you choose to create local accounts that match Google email addresses be sure to use only lowercase characters in the account name, since Google email addresses are transformed to lower-case prior to matching them to local account names.

One problem with creating local accounts that match Google email addresses is that they often contain characters that are invalid by default within Linux usernames (e.g. @ or .). On Debian/Ubuntu systems it's possible to force the system to create a user with these characters. Here's an example of creating a user with a username that contains typically invalid characters:

```
$ sudo adduser --force-badname <username>
```

Note that the `--force-badname` option is only available on Debian/Ubuntu systems and is not available on RedHat/CentOS or SLES systems.

If the users you are creating will only be accessing the server via RStudio, you may also want to disable their ability to login as a normal interactive user and to specify that they have no password. For example:

```
$ sudo adduser --force-badname --disabled-login --disabled-password <username>
```

3.2.3.2 Using an Account Mappings File

Alternatively, you map create local accounts that do not match Google email addresses and then specify a mapping of Google accounts to local accounts via the `/etc/rstudio/google-accounts` configuration file. For example:

/etc/rstudio/google-accounts

```
john.smith@gmail.com=jsmith
sally.jones@gmail.com=sjones
```

Note that changes to the `google-accounts` configuration file take effect immediately and do not require a server restart.

3.3 Proxied Authentication

You can configure RStudio Server to participate in an existing web-based single-sign-on authentication scheme using proxied authentication. In this configuration all traffic to RStudio Server is handled by a proxy server which also handles user authentication.

In this configuration the proxy server adds a special HTTP header to requests to RStudio Server letting it know which authenticated user is making the request. RStudio Server trusts this header, launching and directing traffic to an R session owned by the specified user.

3.3.1 Enabling Proxied Authentication

To enable proxied authentication you need to specify both the `auth-proxy` and `auth-proxy-sign-in-url` settings (the sign-in URL is the absolute URL to the page that users should be redirected to for sign-in). For example:

/etc/rstudio/rserver.conf

```
auth-proxy=1
auth-proxy-sign-in-url=http://example.com/sign-in
```

Note that changes to the configuration will not take effect until the server is restarted.

3.3.2 Implementing the Proxy

When proxying pre-authenticated traffic to RStudio Server you need to include a special HTTP header (`X-RStudio-Username`) with each request indicating which user the request is associated with. For example:

```
X-RStudio-Username: jsmith
```

It's also possible to specify both a system username and a display username (in the case where system accounts are dynamically provisioned and don't convey actual user identity). For example:

```
X-RStudio-Username: rsuser24/jsmith
```

When implementing the proxy it's important to remember that RStudio Server will always trust the `X-RStudio-Username` header to authenticate users. It's therefore critical from the standpoint of security that all requests originating from the proxy have this header set explicitly by the proxy (as opposed to the header being set by a remote client).

3.3.2.1 Customizing the Header Name

You can customize the name of the HTTP header used to determine the user's identity via the `auth-proxy-user-header` setting. For example, to use the header `X-Proxied-Username` rather than the default `X-RStudio-Username` you would use:

/etc/rstudio/rserver.conf

```
auth-proxy-user-header=X-Proxied-Username
```

3.4 Restricting Access by Group

You can specify that only users of a certain group are allowed to access RStudio Server. To do this you use the `auth-required-user-group` setting. For example:

/etc/rstudio/rserver.conf

```
auth-required-user-group=rstudio-users
```

Note that this change will not take effect until the server is restarted.

3.4.1 Creating and Managing Group Membership

To create a new group you use the `groupadd` command:

```
$ sudo groupadd <groupname>
```

To add a user to an existing group you use the **usermod** command:

```
$ sudo usermod -a -G <groupname> <username>
```

Note that it's critical that you include the **-a** flag as that indicates that the group should be added to the user rather than replace the user's group list in it's entirety.

Chapter 4

Access and Security

4.1 Network Port and Address

After initial installation RStudio accepts connections on port 8787. If you wish to listen on a different another port you can modify the `www-port` option. For example:

/etc/rstudio/rserver.conf

```
www-port=80
```

By default RStudio binds to address 0.0.0.0 (accepting connections from any remote IP). You can modify this behavior using the `www-address` option. For example:

/etc/rstudio/rserver.conf

```
www-address=127.0.0.1
```

Note that changes to the configuration will not take effect until the server is restarted.

4.2 IP Access Rules

RStudio Server can be configured to deny access to specific IP addresses or ranges of addresses. Access rules are defined in the configuration file `/etc/rstudio/ip-rules`

Access rules are established using the `allow` and `deny` directives and are processed in order, with the first matching rule governing whether a given address is allowed or denied. For example, to allow only clients within the `192.168.1.0/24` subnet but also deny access to `192.168.1.10` you would use these rules:

/etc/rstudio/ip-rules

```
deny    192.168.1.10
allow   192.168.1.0/24
deny    all
```

All clients outside of the specified subset are denied access because of the `deny all` rule at the end of the configuration.

Note that changes to the configuration will not take effect until the server is restarted.

4.3   Secure Sockets (SSL)

4.3.1   SSL Configuration

If your RStudio Server is running on a public network then configuring it to use SSL (Secure Sockets Layer) encryption is strongly recommended. You can do this via the `ssl-enabled` setting along with related settings that specify the location of your SSL certificate and key. For example:

/etc/rstudio/rserver.conf

```
ssl-enabled=1
ssl-certificate=/var/certs/your_domain_name.crt
ssl-certificate-key=/var/certs/your_domain_name.key
```

It's important when installing the certificate (.crt) file that you concatenate together any intermediate certificates (i.e. the generic one from your certificate authority) with the certificate associated with your domain name. For example you could use a shell command of this form to concatenate the CA intermediate certificate to your domain name's certificate:

```
$ cat certificate-authority.crt >> your_domain_name.crt
```

The resulting file should then be specified in the `ssl-certificate` option.

It's also important to ensure that the file permissions on your SSL certificate key are as restrictive as possible so it can't be read by ordinary users. The file should typically be owned by the `root` user and be set as owner readable and writeable. For example:

```
$ sudo chmod 600 /var/certs/your_domain_name.key
```

4.3.2   SSL Ports

When RStudio Server is configured to use SSL the default behavior with respect to ports is:

1) SSL is bound to port 443 (enabling access using the standard https protocol within the browser)
2) The server also listens on port 80 and redirects all requests to port 443 (allowing users to specify the domain without the https protocol and be automatically redirected to the secure port)

However, if SSL is bound to another port (using the `www-port` option) then the automatic redirect behavior is not enabled. It's also possible to disable automatic SSL redirects entirely using the `ssl-redirect-http` option as follows:

/etc/rstudio/rserver.conf

```
ssl-redirect-http=0
```

Note that changes to the configuration will not take effect until the server is restarted.

4.4 Server Permissions

4.4.1 Server Account

RStudio Server runs as the system root user during startup and then drops this privilege and runs as a more restricted user. RStudio Server then re-assumes root privilege for a brief instant when creating R sessions on behalf of users (the server needs to call `setresuid` when creating the R session, and this call requires root privilege).

The user account that RStudio Server runs under in the normal course of operations is `rstudio-server`. This account is automatically added to the system during installation and is created as a system rather than end user account (i.e. the `--system` flag is passed to `useradd`).

4.4.2 AppArmor

On Debian and Ubuntu systems the RStudio Server process runs under an AppArmor profile (you can find more information about AppArmor here: http://en.wikipedia.org/wiki/AppArmor).

If AppArmor is causing problems in your configuration you can disable it using the `server-app-armor-enabled` option. For example:

/etc/rstudio/rserver.conf

```
server-app-armor-enabled=0
```

Note that there aren't known scenarios where the RStudio Server AppArmor profile causes problems so it's unlikely that you'll ever need to modify this setting. Note also that this setting will not take effect until the server is restarted.

4.5 SPDY

4.5.1 Overview

RStudio Server Pro provides experimental support for the SPDY network protocol (draft 3.1 of the protocol is implemented). The implementation uses the SPDY module provided by nginx 1.8.

Note that SPDY is not supported on RedHat/CentOS 5 or SUSE 11, however is supported on RedHat/CentOS 6 and 7 as well as Ubuntu and Debian systems.

4.5.2   Configuration

Using SPDY requires that you also use SSL. See the section on Secure Sockets (SSL) for additional information on configuring SSL.

You can enable SPDY via the `spdy-enabled` setting along with related settings that configure various aspects of SPDY behavior. For example:

/etc/rstudio/rserver.conf

```
ssl-enabled=1
ssl-certificate=/var/certs/your_domain_name.crt
ssl-certificate-key=/var/certs/your_domain_name.key
spdy-enabled=1
spdy-headers-comp=0
spdy-chunk-size-kb=8
```

The `spdy-headers-comp` parameter sets the header compression level of a response in a range from 1 (fastest, less compression) to 9 (slowest, best compression). The special value 0 turns off the header compression. The default is 0 (disabled).

The `spdy-chunk-size-kb` parameter sets the maximum size of chunks into which the response body is sliced. A too low value results in higher overhead. A too high value impairs prioritization due to HOL blocking. The default is 8k.

Chapter 5

R Sessions

5.1 R Executable and Libraries

5.1.1 Locating R

RStudio Server uses the version of R pointed to by the output of the following command:

```
$ which R
```

The `which` command performs a search for the R executable using the system PATH. RStudio will therefore by default bind to the same version that is run when R is executed from a terminal.

For versions of R installed by system package managers this will be `/usr/bin/R`. For versions of R installed from source this will typically (but not always) be `/usr/local/bin/R`.

If RStudio is unable to locate R using which R, it will fall back to scanning explicitly for the R script in the `/usr/local/bin` and `/usr/bin` directories.

If you want to override which version of R is used then you can use the `rsession-which-r` setting. For example:

/etc/rstudio/rserver.conf

```
rsession-which-r=/usr/local/bin/R
```

Note that this change will not take effect until the server is restarted.

5.1.1.1 Using Multiple Versions of R

The section above describes how RStudio Server locates the global default version of R. It's also possible to specify alternate versions of R either by user or by group. The User and Group Profiles section describes this in more detail.

5.1.2   Locating Shared Libraries

You can add elements to the default `LD_LIBRARY_PATH` for R sessions (as determined by the R `ldpaths` script) by adding an `rsession-ld-library-path` entry to the server config file. This might be useful for ensuring that packages can locate external library dependencies that aren't installed in the system standard library paths. For example:

/etc/rstudio/rserver.conf

```
rsession-ld-library-path=/opt/someapp/lib:/opt/anotherapp/lib
```

Note that this change will not take effect until the server is restarted.

5.1.3   Customzing Session Launches

5.1.3.1   Profile Script Execution

RStudio Server launches R sessions under a bash login shell. This means that prior to the execution of the R session the bash shell will read and execute commands from this file if it exists:

```
/etc/profile
```

After reading that file, it looks for the following files and reads and executes commands from the *first* one that exists and is readable (it's important to note that only one of these files will be read and executed):

```
~/.bash_profile
~/.bash_login
~/.profile
```

If you have further RStudio specific initialization logic (exporting environment variables, etc.) you can optionally create an R session specific profile script at:

```
/etc/rstudio/rsession-profile
```

If it exists this script will be executed prior to the bash shell that launches the R session.

5.1.3.2   Environment Variables

R sessions inherit environment variables that are explicitly exported from the profile scripts described above. It's also possible to append paths to the `LD_LIBRARY_PATH` environment variable using the `rsession-ld-library-path` option (see previous section for details).

Another source of environment variables are PAM sessions. On Debian/Ubuntu systems, the default PAM profile run by RStudio Server includes the environment variables defined in `/etc/security/pam_env.conf` and `/etc/environment`. To learn more about setting environment variables with PAM you should consult the PAM Sessions section as well as the documentation on the `pam_env` module here: http://linux.die.net/man/8/pam_env.

5.1.3.3   Program Supervisors

You may also wish to run R sessions under a program supervisor that modifies their environment or available resources. You can specify a supervisor (and the arguments which control it's behavior) using the `rsession-exec-command` setting. For example:

/etc/rstudio/rserver.conf

```
rsession-exec-command=nice -n 10
```

This example uses the `nice` command to run all R sessions with a lower scheduling priority. See http://linux.die.net/man/1/nice for more details on `nice`. Note that for `nice` in particular it's possible to accomplish the same thing using user and group profiles (and even specify a custom priority level per user or group). See the User and Group Profiles section for more details.

5.2   User and Group Profiles

User and Group Profiles enable you to tailor the behavior of R sessions on a per-user or per-group basis. The following attributes of a session can be configured within a profile:

1) Version of R used
2) CPU affinity (i.e. which set of cores the session should be bound to)
3) Scheduling priority (i.e. nice value)
4) Resource limits (maximum memory, processes, open files, etc.)

5.2.1   Creating Profiles

Profiles are defined within the file `/etc/rstudio/profiles`. Note that this file is not created by default so you'll need to create it if doesn't already exist. Profiles are divided into sections of three different type:

1) Global (`[*]`)

2) Per-group (`[@groupname]`)

3) Per-user (`[username]`)

Here's an example profiles file that illustrates each of these types:

/etc/rstudio/profiles

```
[*]
cpu-affinity = 1-4
max-processes = 100
max-memory-mb = 2048

[@powerusers]
cpu-affinity = 5-16
nice = -10
max-memory-mb = 4096

[jsmith]
r-version = /opt/R/3.1.0/bin/R
```

This configuration specifies that by default users will run on cores 1 to 4 with a limit of 100 processes and 2GB of virtual memory. It also specifies that members of the `powerusers` group will run on cores 5 to 16 with an elevated nice priority and a limit of 4GB of memory. Finally, the user `jsmith` is configured to use a different version of R from the system default.

Note that the `/etc/rstudio/profiles` file is processed from top to bottom (i.e. settings matching the current user that occur later in the file always override ones that appeared prior). The settings available within `/etc/rstudio/profiles` are described in more depth below.

5.2.2 CPU Affinity and Scheduling Priority

If you have users or groups that consistently require more compute resources than others you can use profile settings to reserve CPUs (`cpu-affinity`) as well as raise overall scheduling priority (`nice`).

5.2.2.1 CPU Affinity

The `cpu-affinity` setting specifies which cores on a multi-core system should be used to schedule work for a session. This is specified as a comma-separated list of core numbers (1-based) where both individual cores and ranges of cores can be specified. For example:

```
cpu-affinity = 1,2,3,4
cpu-affinity = 1-4
cpu-affinity = 1,2,15-16
```

To determine the number of addressable cores on your system you can use the `nproc` command:

```
$ nproc
```

5.2.2.2 Scheduling Priority

The `nice` setting specifies a relative priority for scheduling session CPU time. Negative 20 is the highest nice priority and positive 20 is the lowest priority. The system default niceness for processes is typically 0. The following are all valid nice values:

```
nice = -10
nice = 0
nice = 15
```

Scheduler behavior around nice priorities varies by system. For more details see nice use and effect.

5.2.3 Resource Limits

Profiles can also be used to specify limits on available memory as well as the maximum number of processes and open files.

5.2.3.1 Available Memory

The `max-memory-mb` setting controls the maximum amount of addressable memory for R sessions (by default memory is unlimited). This example specifies a limit of 2GB:

```
max-memory-mb = 2048
```

Note that this value sets the amount of virtual memory that can be used by a process. Virtual memory includes code (i.e. shared libraries) loaded by the process as well as things like memory mapped files, so can often consume several hundred megabytes even for a vanilla R session. Therefore, you want to be sure not to set this threshold too low (in no case should you set it below 1024).

5.2.3.2   Number of Processes

The `max-processes` settings controls the maximum number of processes createable by a user. This setting is useful to prevent either inadvertent or malicious fork bombs. The following example sets a limit of 200 processes:

```
max-processes = 200
```

5.2.3.3   Number of Open Files

In most Linux environments there is a maximum of 1024 open files per process. This is typically more than enough, but if you have a particular applications that requires more open files the `max-open-files` setting can be used to increase the limit. For example:

```
max-open-files = 2048
```

5.2.4   Using Multiple Versions of R

As illustrated above, you can bind users or groups to distinct versions of R installed on your server. This is controlled by the `r-version` option. Here are several examples of it's use:

```
r-version = /usr/bin/R
r-version = /usr/lib/R/bin/R
r-version = /usr/local/R/bin/R
r-version = /opt/R/3.1.0/bin/R
```

Note that `r-version` specifies the full path to the binary used to invoke R within the shell (rather than the directory where R is installed). The first two lines in the above example are equivalent (since `/usr/bin/R` is typically either an alias to or copy of `/usr/lib/R/bin/R`).

5.2.4.1   Installing Additional Versions of R

Installing additional versions of R side-by-side with the system version requires building R from source but is very straightforward. First, ensure that you have the build dependencies required for R. On RedHat/CentOS you'd use this command:

```
$ sudo yum-builddep R
```

On Debian/Ubuntu systems you'd use this command:

```
$ sudo apt-get build-dep r-base
```

Once you've satisfied the build dependencies, you should obtain and unarchive the source tarball for the version of R you want to install. Then from within the extracted source directory execute these commands (this example assumes you are installing R 3.1.0):

```
$ ./configure --prefix=/opt/R/3.1.0 --enable-R-shlib
$ make
$ sudo make install
```

Note that the `--enable-R-shlib` option is required in order to make the underlying R runtime library available to RStudio Server.

You may also wish to link to the system BLAS libraries rather than use the R internal versions. For this you'd use the following configure command:

```
./configure --prefix=/opt/R/3.1.0 --enable-R-shlib --with-blas --with-lapack
```

This version of R could then be specified with a profile setting as follows:

```
r-version = /opt/R/3.1.0/bin/R
```

Note that the full path to the R executable rather than the R home directory is specified.

5.3 PAM Sessions

RStudio Server Professional uses PAM (Pluggable Authentication Modules) for both user authentication as well to establish the environment and resources available for R sessions. This is accomplished using the PAM session API. PAM sessions are used for a variety of purposes:

1) To initialize environment variables
2) To automatically create local users after authentication against a directory server.
3) To mount remote drives
4) To initialize and destroy Kerberos tickets

This section explains how to configure and customize PAM sessions with RStudio Server.

5.3.1 Session PAM Profile

For PAM authentication RStudio Server uses the either the `/etc/pam.d/other` profile (Debian/Ubuntu) or `/etc/pam.d/rstudio` profile (RedHat/CentOS). However, for launching R sessions a different PAM profile is used. This is because the launching of R sessions may not coincide with authentication (e.g. returning to the site with login credentials cached in a cookie or resuming a suspended session). Therefore, the PAM directive that enables authentication with `root` privilege only (`auth sufficient pam_rootok.so`) needs to be present in the PAM profile.

The behavior that RStudio Server requires is essentially same as that of the `su` command (impersonation of a user without a password). Therefore by default RStudio Server uses the `/etc/pam.d/su` profile for running R sessions.

5.3.1.1   Creating a Custom Profile

The **/etc/pam.d/su** profile has different default behavior depending upon your version of Linux and local configuration. Depending upon what type of behavior you want associated with R sessions (e.g. mounting of disks, setting of environment variables, enforcing of resource limits, etc.) you'll likely want to create a custom profile for R sessions. For example, if you wanted to use a profile named **rstudio-session** you would add this to the configuration file:

/etc/rstudio/rserver.conf

```
auth-pam-sessions-profile=rstudio-session
```

Here is in turn what the custom profile might contain in order to enable a few common features of PAM sessions (this is based on a modified version of the default **su** profile on Ubuntu):

/etc/pam.d/rstudio-session

```
# This allows root to su without passwords (this is required)
auth       sufficient pam_rootok.so

# This module parses environment configuration file(s)
# and also allows you to use an extended config
# file /etc/security/pam_env.conf.
# parsing /etc/environment needs "readenv=1"
session    required   pam_env.so readenv=1

# Locale variables are also kept into /etc/default/locale in etch
# reading this file *in addition to /etc/environment* does not hurt
session    required   pam_env.so readenv=1 envfile=/etc/default/locale

# Enforces user limits defined in /etc/security/limits.conf
session    required   pam_limits.so

# The standard Unix authentication modules
@include common-auth
@include common-account
@include common-session
```

5.3.1.2   Custom Profile with Passwords

Note that in the above configuration we rely on **pam_rootok.so** to enable authentication without a password. This is necessary because RStudio Server doesn't retain the passwords used during the authentication phase.

In some situations however passwords are important for more than just authentication. PAM profiles support a **use_first_pass** directive to forward passwords used during authentication into other modules (for example, to request a Kerberos ticket with **pam_krb5.so** or to mount an encrypted or remote drive with **pam_mount.so**). For these scenarios RStudio Server supports an optional mode

to retain passwords after login and then forward them into the PAM session profile. This is enabled via the `auth-pam-sessions-use-password` setting:

/etc/rstudio/rserver.conf

```
auth-pam-sessions-use-password=1
```

In this scenario you should remove the `auth sufficient pam_rootok.so` directive and replace it with whatever authentication directives apply in your environment. You can then employ the `use_first_pass` directive to forward the password as necessary to other modules.

For example, here's a very simple RedHat/CentOS PAM configuration file that uses system default authentication and forwards the password into the `pam_mount.so` module. Note that we are no longer using `pam_rootok.so` because the password is now available when the session is created.

/etc/pam.d/rstudio-session

```
# Auth/account (use system auth and forward password to pam_mount)
auth     include    system-auth
auth     optional   pam_mount.so use_first_pass
account  required   pam_unix.so

# Session (read environment variables and enforce limits)
session  required   pam_env.so readenv=1
session  required   pam_env.so readenv=1 envfile=/etc/default/locale
session  required   pam_limits.so
```

Note that this configuration requires that RStudio Server retain user passwords in memory. This retention is done using industry best-practices for securing sensitive in-memory data including disabling ptrace and core dumps, using mlock to prevent paging into the swap area, and overwriting the contents of memory prior to freeing it.

5.3.1.3   More Resources

If you want to learn more about PAM profile configuration the following are good resources:

- http://www.linux-pam.org/Linux-PAM-html/Linux-PAM_SAG.html
- http://linux.die.net/man/8/pam.d
- http://www.linuxjournal.com/article/2120
- http://www.informit.com/articles/article.aspx?p=20968

5.3.2 Disabling PAM Sessions

If you don't want RStudio Server to utilize PAM sessions you can disable this feature using the `auth-pam-sessions-enabled` setting. For example:

/etc/rstudio/rserver.conf

```
auth-pam-sessions-enabled=0
```

5.4 Workspace Management

5.4.1 Default Save Action

When a user exits an R session they need to choose whether to save their R workspace (i.e. `.RData` file). RStudio has global and per-project settings that control what happens when a workspace has unsaved changes. Possible values are:

- `ask` – Ask whether to save the workspace file
- `yes` – Always save the workspace file
- `no` – Never save the workspace file

The default global setting is `ask` and the default project-level setting is derived from the current global setting (these options can be modified by end users via the *Global Options* and *Project Options* dialogs respectively).

The default global setting can also be changed via the `session-save-action-default` configuration parameter in the `rsession.conf` config file. For example, to change the default value to `no` you would use this:

/etc/rstudio/rsession.conf

```
session-save-action-default=no
```

Note that this setting is specified in the `rsession.conf` config file and takes effect the next time a user launches an R session (rather than requiring a full restart of the server).

5.4.2 Suspend and Resume

When R sessions have been idle (no processing or user interaction) for a specified period of time (2 hours by default) RStudio Server suspends them to disk to free up server resources. When the user next interacts with their session it is restored from disk and the user resumes right back where they left off. This is all done seamlessly such that users aren't typically aware that a suspend and resume has occurred.

5.4.2.1 Session Timeout

To configure the amount of idle time to wait before suspending sessions you can use the `session-timeout-minutes` setting in the `/etc/rstudio/rsession.conf` file. For example:

/etc/rstudio/rsession.conf

```
session-timeout-minutes=30
```

Important note: this setting and a few others discussed in this section are specified in the `/etc/rstudio/rsession.conf` file (rather than the `rserver.conf` file previously referenced).

There are some conditions where an R session will not be suspended, these include:

1) When a top-level R computation is running
2) When the R prompt is not in it's default state (e.g. during a debugging session)

You can also specify that R sessions should never be suspended by setting the `session-timeout-minutes` to zero. For example:

/etc/rstudio/rsession.conf

```
session-timeout-minutes=0
```

Note that session timeout settings take effect the next time a user launches an R session (rather than requiring a full restart of the server).

5.4.2.2 Forcing Suspends

You can force the suspend of individual sessions or even all sessions on the server. You can do this directly from the main page of the Administrative Dashboard or from the system shell as follows:

```
$ sudo rstudio-server force-suspend-session <pid>
$ sudo rstudio-server force-suspend-all
```

5.4.2.3 Resume and .Rprofile

By default the `Rprofile.site` and `.Rprofile` files are not re-run when a session is resumed (it's presumed that all of their side-effects are accounted for by simply restoring loaded packages, options, environment variables, etc.).

In some configurations it might be desirable to force the re-execution of profile files. There is an end user option that controls this on the *General* options pane which defaults to false. However, server administrators may wish to ensure that this option defaults to true. To do this you use the `session-rprofile-on-resume-default` option. For example:

/etc/rstudio/rsession.conf

```
session-rprofile-on-resume-default=1
```

Note that this setting is specified in the `rsession.conf` config file and takes effect the next time a user launches an R session (rather than requiring a full restart of the server).

5.4.3   Workspace Storage

Storage of workspaces (.RData files) in RStudio Server does not use compression by default. This differs from the behavior of base R. Compression is disabled because we've observed that for larger workspaces (> 50MB) compression can result in much lower performance for session startup and suspend/resume (on the order of 3 or 4 times slower).

The default workspace save options under RStudio Server are as follows:

```
options(save.defaults=list(ascii=FALSE, compress=FALSE))
options(save.image.defaults=list(ascii=FALSE, safe=TRUE, compress=FALSE))
```

If you wish to use different defaults you can define the `save.defaults` and/or `save.image.defaults` options in your `Rprofile.site` or per-user `.Rprofile` and RStudio Server will respect the settings you specify rather than using it's own defaults.

See https://stat.ethz.ch/R-manual/R-devel/library/base/html/save.html for additional details on how R saves objects and the storage and performance implications of using compression.

5.5   Package Installation

You can customize the location of user packages installed from CRAN as well as the default CRAN repository. You can also configure the user-interface of the RStudio IDE to discourage end-user package installation in the case where packages are deployed centrally to a site library.

Important note: The settings discusseded in this section are specified in the `/etc/rstudio/rsession.conf` file (rather than the `rserver.conf` file previously referenced).

5.5.1   User Library

By default R packages are installed into a user-specific library based on the contents of the `R_LIBS_USER` environment variable (more details on this mechanism are here: http://stat.ethz.ch/R-manual/R-devel/library/base/html/libPaths.html).

It's also possible to configure an alternative default for user package installation using the `r-libs-user` setting. For example:

/etc/rstudio/rsession.conf

```
r-libs-user=~/R/library
```

One benefit of establishing an alternative default user library path is that by doing this you can remove the R version component of the package library path (which the default path contains). This makes it possible to upgrade the major version of R on the server and have user's packages continue to work.

5.5.2   Discouraging User Installations

It may be that you've configured RStudio Server with a site package library that is shared by all users. In this case you might wish to discourage users from installing their own packages by removing the package installation UI from the RStudio IDE. To do this you use the `allow-package-installation` setting. For example:

/etc/rstudio/rsession.conf

```
allow-package-installation=0
```

Note that this setting merely discourages package installation by removing user-interface elements. It's still possible for users to install packages directly using the `utils::install.packages` function.

5.5.3   CRAN Repositories

RStudio Server uses the RStudio CRAN mirror (http://cran.rstudio.com) by default. This mirror is globally distributed using Amazon S3 storage so should provide good performance for all locales. You may however wish to override the default CRAN mirror. This can be done with the `r-cran-repos` settings. For example:

/etc/rstudio/rsession.conf

```
r-cran-repos=http://cran.at.r-project.org/
```

Whatever the default CRAN mirror is, individual users are still able to set their own default. To discourage this, you can set the `allow-r-cran-repos-edit` settings. For example:

/etc/rstudio/rsession.conf

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

Note that even with user editing turned off it's still possible for users to install packages from alternative repositories by directly specifying the `repos` parameter in a call to `install.packages`.

5.6   Feature Limits

RStudio Server has a number of other limits that can be configured. This section describes these limits. Note that these settings are specified in the `/etc/rstudio/rsession.conf` file (rather than the `rserver.conf` file previously referenced).

5.6.1   Disabling Access to Features

Besides the limits on package installation and CRAN repository editing described in the previous section there are a number of other limits that can be specified. The following describes all of the options that can be used to limit features.

/etc/rstudio/rsession.conf

allow-vcs Allow access to Git and SVN version control features.

allow-vcs-executable-edit Allow editing of the underlying Git or SVN executable.

allow-package-installation Allow installation of packages using the Packages Pane (note that even if this is set to 0 it's still possible to install packages using `utils::install.packages` from the command line).

allow-r-cran-repos-edit Allow editing of the CRAN repository used for package downloads (note that it's still possible to specify an alternate repository using the `repos` parameter of `utils::install.packages`).

allow-shell Allow access to the Tolls -> Shell dialog (note that it's still possible to execute shell commands using the `system` function).

allow-file-downloads Allow downloading files using the Export command in the Files Pane.

allow-external-publish Allow content to be published to external (cloud) services. This includes publishing HTML documents created with R Markdown or R Presentations to RPubs (http://rpubs.com), and publishing Shiny applications and documents to ShinyApps.io (http://shinyapps.io). Note that this just removes the relevant user interface elements in the IDE, and that it may still be possible for users to publish content using the R console.

allow-publish Allow content to be published. If specified, this option removes all user interface elements related to publishing content from the IDE, and overrides `allow-external-publish`.

All of these features are enabled by default. Specify `0` to disable access to the feature.

Note that these options should be specified in the **/etc/rstudio/rsession.conf** configuration file (rather than the main rserver.conf configuration file).

5.6.2   Maximum File Upload Size

You can limit the maximum size of a file upload by using the `limit-file-upload-size-mb` setting. For example, the following limits file uploads to 100MB:

/etc/rstudio/rsession.conf

```
limit-file-upload-size-mb=100
```

The default behavior is no limit on the size of file uploads.

5.6.3 CPU Time per Computation

If you want to prevent runaway computations that consume 100% of the CPU you can set the maximum number of minutes to allow top-level R computations to run for using the `limit-cpu-time-minutes` setting. For example:

/etc/rstudio/rsession.conf

```
limit-cpu-time-minutes=30
```

This specifies that no top level computation entered at the R console should run for more than 30 minutes. This constraint is implemented by calling the R `setTimeLimit` function immediately prior to handing off console commands to R. As a result it is possible for a particular script to override this behavior if it knows that it may exceed the threshold. This would be done as follows:

```
setTimeLimit(cpu = Inf)
# Long running R code here...
```

5.6.4   XFS Disk Quotas

If your system uses the XFS file system (http://en.wikipedia.org/wiki/XFS) then RStudio Server can be configured to notify users when they come close to or exceed their disk quota. You can enable this using the `limit-xfs-disk-quota` setting. For example:

/etc/rstudio/rsession.conf

```
limit-xfs-disk-quota=1
```

The user's XFS disk quota will be checked when the RStudio IDE loads and a warning message will be displayed if they are near to or over their quota.

Chapter 6

Load Balancing

6.1   Overview

RStudio Server can be configured to load balance R sessions across two or more nodes within a cluster. This provides both increased capacity as well as higher availability.

Note that load balancing for RStudio Server has some particular "stickiness" requirements stemming from the fact that users must always return to the same R session where their work resides (i.e. their traffic can't be handled by more than one node). As a result, it's not enough to simply place multiple RStudio Servers behind a conventional hardware or software load balancer—additional intelligence and routing is required.

Key characteristics of the RStudio Server load balancer include:

1. Multiple masters for high availability—all nodes can balance traffic to all other nodes.

2. Support for several load balancing strategies including least busy server (by active sessions or system load), even distribution by user, or a custom strategy based on an external script.

3. The ability to add and remove nodes while the cluster is running.

4. Works standalone or can be integrated with other front-end load balancing environments.

6.2   Configuration

6.2.1   Requirements

There are three main requirements for nodes within RStudio clusters:

1. All nodes must run the same version of RStudio Server Pro.

2. Server configurations (i.e. contents of the `/etc/rstudio` directory) must be identical.

3. User accounts must be accessible from each node.

4. User home directories must be accessible via **shared storage** (e.g. all nodes mounting the same NFS volume).

5. The shared storage (e.g. NFS) must support file locking. If you are unsure about this see the File Locking section below for details on how to test your configuration for the required capabilities.

6.2.2 Defining Nodes

To define a cluster node, two configuration files need to be provided:

```
/etc/rstudio/load-balancer
/etc/rstudio/secure-cookie-key
```

The first of these defines the available nodes and load balancing strategy. The second defines a shared key used for signing cookies (in single node configurations this key is generated automatically, however with multiple nodes explicit coordination is required).

For example, to define a cluster with two nodes that load balances based the number of actively running R sessions you could use the following configuration:

/etc/rstudio/load-balancer

```
[config]

balancer = sessions

[nodes]

server1.example.com
server2.example.com
```

/etc/rstudio/secure-cookie-key

```
a55e5dc0-d6ae-11e3-9334-000c29635f71
```

The secure cookie key is simply a unique value (in this case a UUID). Note that this file must have user read/write file permissions (i.e. 0600) to protect it's contents from other users. You can create a secure cookie key using the `uuid` utility as follows:

```
sudo sh -c "echo `uuid` > /etc/rstudio/secure-cookie-key"
sudo chmod 0600 /etc/rstudio/secure-cookie-key
```

6.2.3 File Locking

In order to synchronize the creation of sessions across multiple nodes RStudio Server uses the advisory file locking capabilities of the Posix fctrl function. It's therefore crtical that the shared storage you use for user home directories support file locking (note that NFS file servers do support file locking via the lockd daemon).

6.2.3.1 Testing Utility

RStudio Server includes a `locktester` utilty which you can use to verify that the requisite locking capabilities are available and working correctly. To use the `locktester` you should login (e.g. via SSH or telnet) to at least two nodes using the same user account and then invoke the utilty from both sessions as follows:

```
$ /usr/lib/rstudio-server/bin/locktester
```

The first node you execute the utility from should print the following message:

```
*** File Lock Acquired ***
```

After the message is printed the process will pause so that it can retain the lock (you can cause it to release the lock by interrupting it e.g. via Ctrl+C).

The second and subsequent nodes you execute the utilty from should print the following message:

```
Unable to Acquire File Lock
```

If you interrupt the first node (e.g. via Ctrl+C) the lock will be released and you can then acquire it from the other nodes.

If either of the following occurs then there is an issue with file locking capabilites (or configuration) that should be addressed prior to using load balancing:

1) All nodes successfully acquire the file lock (i.e. more than one node can hold it concurrently).
2) No nodes are able to acquire the file lock.

If either of the above conditions hold then RStudio won't be able to correctly synchronize the creation of R sessions throughtout the cluster (potentially resulting in duplicate sessions and lost data due to sessions overriting eachother's state).

6.2.4 Managing Nodes

6.2.4.1 Starting Up

After creating your configuration files you should ensure that these files (along with all other configuration defined in `/etc/rstudio`) are copied to all nodes in the cluster. Assuming that the server is already installed and running on each node, you can then apply the load balancing configuration by restarting the server:

```
sudo rstudio-server restart
```

6.2.4.2   Current Status

Once the cluster is running you can inspect it's state (which sessions are running where) using the load balancing status HTTP endpoint. For example:

```
curl http://localhost/load-balancer/status
```

Note that the status endpoint is accessed using localhost rather than an external IP address. This is because this endpoint is IP restricted to only be accessible within the cluster, so needs to be accessed directly from one of the nodes.

6.2.4.3   Adding and Removing Nodes

To temporarily remove a node from the cluster you can simply stop it:

```
sudo rstudio-server stop
```

R sessions running on that node will be automatically moved to another active node. To restore the node you can simply start it back up again:

```
sudo rstudio-server start
```

Note that adding and removing nodes does not require changing the list of defined nodes in `/etc/rstudio/load-balancer` (traffic is automatically routed around nodes not currently running).

6.3   Access and Availablity

Once you've defined a cluster and brought it online you'll need to decide how the cluster should be addressed by end users. There are two distinct approaches to this:

1. **Single Master**. Provide users with the address of one of the nodes. This node will automatically route traffic and sessions as required to the other nodes. This has the benefit of simplicity (no additional software or hardware required) but also results in a single point of failure.

2. **Multiple Masters**. Put the nodes behind some type of system that routes traffic to them (e.g. dynamic DNS or a software or hardware load balancer). While this requires additional configuration it also enables all of nodes to serve as points of failover for each other.

Both of these options are described in detail below.

6.3.1   Single Master

In a Single Master configuration, you designate one of the nodes in the cluster as the primary one and provide end users with the address of this node as their point of access. For example:

```
[nodes]
rstudio.example.com
rstudio2.example.com
rstudio3.example.com
```

Users would access the cluster using **http://rstudio.example.com**. This node would in turn route traffic and sessions both to itself and the other nodes in the cluster in accordance with the active load balancing strategy.

Note that in this configuration the **rstudio2.example.com** and **rstudio3.example.com** nodes can either fail or be removed from the cluster at any time and service will continue to users. However, if the main node fails or is removed then the cluster is effectively down.

6.3.2 Multiple Masters

In a Multiple Masters configuration all of the nodes in the cluster are peers and provide failover for each other. This requires that some external system (dynamic DNS or a load balancer) route traffic to the nodes. In this scenario any of the nodes can fail and service will continue (so long as the external router can respond intelligently to a node being unreachable).

For example, here's an Nginx reverse-proxy configuration that you could use with the cluster defined above:

```
http {
  upstream rstudio-server {
    server rstudio1.example.com;
    server rstudio2.example.com backup;
    server rstudio3.example.com backup;
  }
  server {
    listen 80;
    location / {
      proxy_pass http://rstudio-server;
      proxy_redirect http://rstudio-server/ $scheme://$host/;
    }
  }
}
```

In this scenario the Nginx software load balancer would be running on **rstudio.example.com** and reverse proxy traffic to **rstudio1.example.com**, **rstudio2.example.com**, etc. Note that one node is designated as primary so traffic is routed there by default. However, if that node fails then Nginx automatically makes use of the backup nodes.

This is merely one example as there are many ways to route traffic to multiple servers—RStudio Server load balancing is designed to be compatible with all of them.

6.3.3 Using SSL

If you are running an RStudio Server on a public facing network then using SSL encryption is strongly recommended. Without this all user session data (including passwords) is sent in the clear and can be intercepted by malicious parties.

The recommended SSL configuration depends on which access topology you've deployed:

1. For a Single Master deployment, you would configure each node of the cluster to use SSL as described in the Secure Sockets (SSL) section. The nodes will then use SSL for both external and intra-machine communication.

2. For a Multiple Masters deployment, you would configure SSL within the external routing layer (e.g. the Nginx server in the example above) and use standard unencrypted HTTP for the individual nodes.

6.4 Balancing Methods

There are four methods available for balancing R sessions across a cluster. The most appropriate method is installation specific and depends on the number of users and type of workloads they create.

6.4.1 Sessions

The default balancing method is `sessions`, which attempts to evenly distribute R sessions across the nodes of the cluster:

```
[config]
balancer = sessions
```

This method allocates new R sessions to the node with the least number of active R sessions. This is a good choice if you expect that users will for the most part have similar resource requirements.

6.4.2 System Load

The `system-load` balancing method distributes sessions based on the active workload of available nodes:

```
[config]
balancer = system-load
```

The metric used to establish active workload is the 5-minute load average. This is a good choice if you expect widely disparate CPU workloads and want to ensure that machines with high CPU utilization don't receive new sessions.

6.4.3 User Hash

The `user-hash` balancing method attempts to distribute load evenly and consistently across nodes by hashing the username of clients:

```
[config]
balancer = user-hash
```

The hashing algorithm used is CityHash, which will produce a relatively even distribution of users to nodes. This is a good choice if you want the assignment of users/sessions to nodes to be stable.

6.4.4 Custom

The `custom` balancing method calls out to external script to make load balancing decisions:

```
[config]
balancer = custom
```

When `custom` is specified, RStudio Server will execute the following script when it needs to make a choice about which node to start a new session on:

```
/usr/lib/rstudio-server/bin/rserver-balancer
```

This script will be passed two environment variables:

`RSTUDIO_USERNAME` — The user on behalf or which the new R session is being created.

`RSTUDIO_NODES` — Comma separated list of the IP address and port of available nodes.

The script should return the node to start the new session on using it's standard output. Note that the format of the returned node should be identical to it's format as passed to the script (i.e. include the IP address and port).

Chapter 7

Monitoring the Server

7.1   Monitoring Configuration

RStudio Server monitors the use of resources (CPU, memory, etc.) on both a per-user and system wide basis. By default, monitoring data is written to a set of RRD (http://oss.oetiker.ch/rrdtool/) files and can be viewed using the Administrative Dashboard.

The storage of system monitoring data requires about 20MB of disk space and the storage of user monitoring data requires about 3.5MB per user. This data is stored by default at `/var/lib/rstudio-server/monitor`. If you have a large number of users you may wish to specify an alternate volume for monitoring data. You can do this using the `monitor-data-path` setting. For example:

/etc/rstudio/rserver.conf

```
monitor-data-path=/monitor-data
```

You also might wish to disable monitoring with RRD entirely. You can do this using the `monitor-rrd-enabled` setting. For example:

/etc/rstudio/rserver.conf

```
monitor-rrd-enabled=0
```

Note that changes to the configuration will not take effect until the server is restarted.

7.2   Using Graphite

If you are managing several servers it might be convenient to send server monitoring data to a centralized database and graphing facility as opposed to local RRD files. You can do this by configuring the server to send monitoring data to Graphite (or any other engine compatible with the Carbon protocol). This can be done in addition to or entirely in place of RRD.

There are three settings that control interaction with Graphite:

monitor-graphite-enabled	Write monitoring data to Graphite (defaults to `0`)
monitor-graphite-host	Host running Graphite (defaults to `127.0.0.1`)
monitor-graphite-port	Port Graphite is listening on (defaults to `2003`)

For example, to enable Graphite monitoring on a remote host with the default Graphite port you would use these settings:

/etc/rstudio/rserver.conf

```
monitor-graphite-enabled=1
monitor-graphite-host=134.47.22.6
```

Note that changes to the configuration will not take effect until the server is restarted.

7.3   Server Health Checks

7.3.1   Enabling Health Checks

You may wish to periodically poll RStudio Server to ensure that it's still responding to requests as well as to examine various indicators of server load. You can enable a health check endpoint using the `server-health-check-enabled` setting. For example:

/etc/rstudio/rserver.conf

```
server-health-check-enabled=1
```

After restarting the server, the following health-check endpoint will be available:

```
http://<server-address-and-port>/health-check
```

By default, the output of the health check will appear as follows:

```
active-sessions: 1
cpu-percent: 0.0
memory-percent: 64.2
swap-percent: 0.0
load-average: 4.1
```

7.3.2   Customizing Responses

The response to the health check is determined by processing a template that includes several variables. The default template is:

```
active-sessions: #active-sessions#
cpu-percent: #cpu-percent#
memory-percent: #memory-percent#
swap-percent: #swap-percent#
load-average: #load-average#
```

You can customize this template to return an alternate format (e.g. XML or JSON) that is parseable by an external monitoring system. To do this you simply create a template and copy it to **/etc/rstudio/health-check** For example:

/etc/rstudio/health-check

```xml
<?xml version="1.0" encoding="UTF-8"?>
<health-check>
  <active-sessions>#active-sessions#</active-sessions>
  <cpu-percent>#cpu-percent#</cpu-percent>
  <memory-percent>#memory-percent#</memory-percent>
  <swap-percent>#swap-percent#</swap-percent>
  <load-average>#load-average#</load-average>
</health-check>
```

7.3.3   Changing the URL

It's also possible to customize the URL used for health checks. RStudio Server will use the first file whose name begins with **health-check** in the **/etc/rstudio** directory as the template, and require that the full file name be specified in the URL. For example, a health check template located at the following path:

```
/etc/rstudio/health-check-B64C900E
```

Would be accessed using this URL:

```
http://<server-address-and-port>/health-check-B64C900E
```

Note that changes to the health check template will not take effect until the server is restarted.

Chapter 8

License Management

8.1   Product Activation

8.1.1   Activation Basics

When RStudio Server is first installed on a system it operates in 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 rstudio-server license-manager status
```

After purchasing a license to RStudio Server you'll receive a product key that can be used to activate the license on a given system. You can perform the activation as follows:

```
$ sudo rstudio-server license-manager activate <product-key>
$ sudo rstudio-server restart
```

Note that you need to restart the server in order for licensing changes to take effect.

If you want to move your license of RStudio Server to another system you should first deactivate it on the system you are moving from. For example:

```
$ sudo rstudio-server license-manager deactivate
```

8.2   Connectivity Requirements

In order to activate or deactivate RStudio Server internet connectivity is required for communication with the licensing server. If your server is behind an internet proxy or not connected to the internet at all this section describes what's required to successfully activate.

8.2.1 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. Note however that this may not be necessary if either the `http_proxy` or `all_proxy` environment variable is defined (these are read and used by RStudio Server 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 rstudio-server 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.

8.2.2 Offline Activation

If your system has no connection to the internet it's also possible to perform an offline activation. To do this, you first generate an offline activation request as follows:

```
$ sudo rstudio-server 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 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 rstudio-server license-manager activate-offline <activation-file>
$ sudo rstudio-server restart
```

Note that you need to restart the server in order for licensing changes to take effect.

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

```
$ sudo rstudio-server license-manager deactivate-offline
```

Executing this command will print an offline deactivation request to the terminal which you should copy and paste and then 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 rstudio-server license-manager status-offline
```