Getting Started with RethinkDB

Getting Started with RethinkDB is designed to get you working with RethinkDB as quickly as possible. Starting with the installation and configuration process, you will learn how to start importing data into the database and run simple queries using the intuitive ReQL query language.

After successfully running a few simple queries, you will be introduced to other topics, such as clustering and sharding. You will learn how to work with RethinkDB using a Node.js environment and find out all about deployment techniques.

Finally, you’ll finish by working on a fully-fledged example that uses the Node.js framework and advanced features such as Changefeeds to develop a real-time web application.

What you will learn from this book
- Download and install the database on your system
- Configure RethinkDB’s settings and start using the web interface
- Import data into RethinkDB
- Run queries using the ReQL language
- Create shards, replicas, and RethinkDB clusters
- Use an index to improve database performance
- Get to know all the RethinkDB deployment techniques

Who this book is written for
Getting Started with RethinkDB is ideal for developers who are new to RethinkDB and need a practical understanding to start working with it. No previous knowledge of database programming is required, although a basic knowledge of JavaScript or Node.js would be helpful.

Community Experience Distilled

Absorb the knowledge required to utilize, manage, and deploy RethinkDB using Node.js


Gianluca Tiepolo

7.50 x 9.25
235 mm x 191 mm

Content Type: Black & White
Paper Type: White
Page Count: 176
File Type: InDesign
Request ID: CSS1563302

Open Source

PACKT

Perfect Bound Cover Template

Lightning Source

Document Size: 19” x 12”
396 x 483 mm
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Introducing RethinkDB'
- A synopsis of the book’s content
- More information on Getting Started with RethinkDB
Gianluca Tiepolo is an accomplished software engineering professional and entrepreneur with several years of experience in developing software and products on a variety of technologies, from consumer applications to revolutionary projects focused on computer vision, data engineering, and database programming. His software stack ranges from traditional platforms, such as Hadoop and OpenCV, to modern platforms, such as Node.js and Redis.

He is the founder of Defrogs, a start-up that is building a new-generation data engineering platform to handle big data called TrisDB. Bringing in innovative data process development approaches, this organization focuses on cutting-edge technologies designed to scale small to large distributed data clusters. To date, TrisDB is used by more than 3 million developers around the world. He previously co-founded Sixth Sense Solutions, a start-up that develops interactive solutions for the retail and fashion industries. In 2013, he helped produce the largest touch-enabled surface in the world. Currently, he’s working on a fashion platform called Stylobag and maintains several open source projects on his GitHub account.

In 2015, he reviewed the book called Building Web Applications with Python and Neo4j published by Packt Publishing.
Preface

Databases are all around us. In the modern web, almost every website that we visit and the web-based applications that we use have a database system working behind the frontend. Web developers are constantly looking for new database solutions that adapt to the modern web, allowing them to store data in a simpler manner.

RethinkDB is both the simplest and the most powerful document database technology available across Linux and OS X platforms. Based on robust and feature-rich software, RethinkDB provides a bunch of features that can be used to develop some real-time web applications that can be scaled incredibly easily. RethinkDB is also open source, so the source code is freely downloadable from the database GitHub repository.

This book provides an introduction to RethinkDB. The following chapters will give you some understanding and coding tips to install and configure the database and start developing web applications with RethinkDB in no time.

What this book covers

Chapter 1, Introducing RethinkDB, explains how to download and install RethinkDB on both Linux and OS X.

Chapter 2, The ReQL Query Language, explains the basics of RethinkDB’s query language and how to use it to run simple queries on the database.

Chapter 3, Clustering, Sharding, and Replication, explores the different techniques you can use to scale RethinkDB.

Chapter 4, Performance Tuning and Advanced Queries, gives out best practices to obtain optimal performance and explores more advanced queries.
Preface

Chapter 5, Programming RethinkDB in Node.js, explains how to interact with the database using the Node.js programming language.

Chapter 6, RethinkDB Administration and Deployment, teaches you how to maintain your RethinkDB database instance and how to deploy it to the cloud.

Chapter 7, Developing Real-Time Web Applications, explores how to develop a full-fledged Node.js web application based on RethinkDB.
1

Introducing RethinkDB

RethinkDB is an open source, distributed, and document-oriented database built to store JSON documents and is used to scale multiple machines with very little effort. It's easy to set up and use, and it has a powerful query language that supports advanced queries such as table joins, aggregations, and MapReduce.

This chapter covers major design decisions that have made RethinkDB what it is now including the unique features it offers for real-time application development. We're going to start by looking at the basics of RethinkDB, why it is different, and why the new approach has everybody excited about using it to build the next generation web apps.

In this chapter, you will also learn the following:

- Installing the database on Linux and OS X
- Configuring it
- Running a query using the web interface

The RethinkDB development team provides prepackaged binaries for some platforms, whereas the source code is available on GitHub. You will learn to install the database using both methods. The choice of which type of package to use depends on which one is more appropriate for your system; Ubuntu, Debian, CentOS, and OS X users may prefer using the provided binaries, whereas users using different platforms can install RethinkDB by downloading and compiling the source code.
Rethinking the database

Traditional database systems have existed for many years, and they all have a familiar structure and common methods of communicating, inserting, and querying for information; however, the relatively recent rise and diffusion of NoSQL databases have given developers an increasingly large amount of choice on what to use for their data storage.

Although, new scalability capabilities have most certainly revolutionized the performance that these databases can deliver, most NoSQL systems still rely on the creation of a specific structure that is organized collectively into a record of data. Additionally, the access model of these systems has not changed to adapt today's modern web applications; to get in information, you add a record of data, and to get the information out, you query the database by polling specific values or fields as illustrated by the following diagram:

However, as technology evolves, it's often worth rethinking how we do tasks. RethinkDB takes a completely different approach to the database structure and methods of storing and retrieving information.

What follows is an overview of RethinkDB's main features along with accompanying considerations of how it differs from other NoSQL databases.
**Changefeeds**

RethinkDB is designed for building real-time applications. Using a feature called **Changefeeds**, developers can program the database to continuously push data updates to applications in real time. This fundamental architecture choice solves all the problems generated by continuously polling the database, as it is the database itself that serves data to applications in real time by reducing the time and complexity required to develop scalable web apps. The following diagram illustrates how this works:

![Diagram illustrating Changefeeds](image)

The best part about how RethinkDB handles Changefeeds is that you don't need to particularly modify your queries to implement them. They look identical to a normal query apart from the `changes()` command that gets appended to it. Currently, the `changes` command works on a large subset of queries and allows a client to receive updates on a table, a single document, or even the results from a specific query as they happen.

**Horizontal scalability**

RethinkDB is a very good solution when flexibility and rapid iteration are of primary importance. Its other big strength is its ability to scale horizontally with very little effort or changes required to how you interact with the database. Horizontal scalability consists of expanding the storage capacity and processing power of a database by adding more servers to a cluster. A single database node is greatly limited by the capacity of the server that hosts it. So, if the dataset exceeds available capacity, data must be **sharded** among multiple database instances that are connected to each other.
Thankfully, the RethinkDB team set out to make scaling really easy for developers. Users should not have to worry about these issues at all wherever possible. So, with RethinkDB, you can set up a cluster, create table-level shards, and run cross-shard joins and aggregations in less than five minutes using the web interface.

**Powerful query language**

The RethinkDB query language, ReQL, is a data-driven, abstract, advanced language that embeds itself perfectly in the programming language that you use to build your applications; in fact, in ReQL, queries are constructed simply by making function calls in any programming language that you prefer. ReQL is designed to be pragmatic and works like a fluent API—a set of functions that you can chain together to compose queries. It supports advanced queries including massively parallelized distributed computation. All queries are automatically parallelized on the database server and, whenever possible, query execution is split across multiple cores and datacenters. RethinkDB will automatically break large queries into stages and execute each stage in parallel by combining intermediate data to return a complete query result.

Official RethinkDB client drivers are available for JavaScript, Python and Ruby; however, support for other programming languages is available through community-supported drivers.

**Developer-oriented**

RethinkDB is different by design. In fact, it aims to be both developer friendly and operations-oriented, combining an easy-to-use query language with simple controls for operating at scale, while still maintaining an operations-oriented approach of being highly available and extremely scalable.

Since its first release, RethinkDB has gained a large, vibrant, developer community quicker than almost any other database; in fact, today, RethinkDB is the second most popular database on GitHub and is becoming the database of choice for many big and small companies with hundreds of technology start-ups already using it in production.
Document-oriented

One of the reasons behind RethinkDB's popularity among developers is its data model. **JSON** has become the de-facto standard for data interchange for modern web applications and a persistence layer that naturally stores, queries, and manages JSON. It makes life easier for developers. RethinkDB is a document database built from the ground up to take advantage of JSON's feature set. When developers have to work with objects in databases, it can be troublesome at times due to data mapping and impedance issues. Document-oriented databases solve these issues by replacing the concept of a *row* with a more flexible model called the *document*, as documents are objects. After all, programmers who tend to work with objects are going to be much more familiar with storing and querying such data in RethinkDB. If you've never worked with a document before, consider the following example that represents a person using JSON:

```json
{
  "firstName": "Alex",
  "lastName": "Jones",
  "yearOfBirth": 1991,
  "phoneNumbers": {
    "home": "02-345678",
    "mobile": "345-12345678"
  },
  "interests": [
    "programming",
    "football",
    "chess"
  ]
}
```

As you can see from the preceding example, a document always begins and ends with curly braces, keys and values are separated by colons, and key/value pairs are separated by commas. The key is always a string. A typical JSON document lets you represent values as numbers, strings, bools, arrays, and objects; however, RethinkDB adds other data types that you can use to model your data—binary data, dates and times and the null value. Since version 1.15, RethinkDB also supports geospatial queries for you to include geometry within your JSON documents.

By allowing embedded objects and arrays in JSON, the document-oriented approach used by RethinkDB lets you represent complex relationships with a single document. This fits naturally into the way in which web developers think and model their data.
Lock-free architecture

Traditional, relational, and document databases, more often than not, use locks at various levels to ensure proper data consistency during concurrent access to the database. In a typical NoSQL database that uses locking, once a write request comes in, all readers are blocked until the write completes. What this means is that in some use cases that require large volumes of writes, this architecture could eventually lead to reads to the database getting queued up, resulting in significant performance degradation.

RethinkDB solves this problem by implementing block-level Multi-Version Concurrency Control (MVCC)—a method commonly used by database management systems that provides concurrent access to the database without locking it. Whenever a write operation occurs while there is an ongoing read, the database takes a snapshot of the data block for each relevant shard and temporarily maintains different versions of the blocks in order to execute both read and write operations at the same time.

The main difference between MVCC and lock models is that in MVCC, locks acquired for reading data don't conflict with locks acquired for writing data, and so, reading never blocks writing and vice versa. The concurrency model used by RethinkDB ensures, for example, that you can run an hour-long MapReduce job without blocking the database.

Immediate consistency

For distributed databases, consistency models are a topic of huge importance and RethinkDB makes no exception. A database is said to be consistent when a series of operations or transactions performed on it are applied in a consistent order. What this means is that if we insert some data into a table, it will immediately be available to any other client that wishes to read it. Likewise, if we read some data from the database, we want this data to be the most recently updated version. This is called immediate consistency and is a property of most traditional databases as MySQL.

Some databases as Cassandra decide to prioritize high availability and give up on immediate consistency in the favor of eventual consistency. In this case, if the network goes down, the database will still be able to accept reads and writes; however, applications built at the top of these systems will have to deal with various complexities, such as conflict resolutions and potential out-of-date reads.

RethinkDB, on the other hand, always maintains strong data consistency as all reads and writes get routed to the primary database shard where queries are executed. This results in immediately consistent and conflict-free data, and all reads on the database are guaranteed to return the most recent data.
The CAP theorem by Eric Brewer states that a database can only have two of the following guarantees at the same time: consistency, availability, and tolerance of network partitions. In distributed systems as RethinkDB, network partitioning is inevitable and must be tolerated, so essentially, what the theorem means is that a tradeoff has to be made between consistency and high availability.

Secondary indexes
Simply put, a secondary index is a data structure that improves the lookup of documents by an attribute other than their primary key at the expense of write performance. This type of index is heavily used in web applications, as it is extremely common to efficiently retrieve all documents based on a field that is not a primary key. RethinkDB also supports compound indexes that are based on multiple fields and other indexes based on arbitrary expressions. Support for secondary indexes was added in version 1.5.

Distributed joins
Most relational databases allow us to perform queries that define explicit relationships between different pieces of data often contained in multiple tables. These queries are called joins and are not supported by most NoSQL databases. The reason for this is that the need for joins is not a function of the data model, but it is a function of the data access. If data is structured in such a way that it conforms structurally to the queries that are being executed, joins can be avoided. The drawback with this approach is that it requires you to structure your data in advance and knowing beforehand how you will access your data often proves to be very tricky.

RethinkDB not only supports joins but automatically compiles them to distributed programs and executes them across the cluster without further intervention from the client. When you use join queries in RethinkDB, what happens is that you connect two sequences of data based on some type of equality; the query then gets routed to the appropriate nodes and the data is combined into a final result that is returned to the client.

Now that you know what RethinkDB is and you’ve got a comprehensive understanding of its powerful feature set, it’s time to take a step forward and start using it. We’ll start by downloading and installing the database.
Introducing RethinkDB

Installing RethinkDB
So far, you've learned all about RethinkDB's features, and I'm sure that you're curious to start working with this amazing database. Now, we're ready to take a closer look at how to install RethinkDB on your system. Currently, the database is compatible with OS X and most Linux-based operating systems. So, this final part of the chapter will walk you through how to install RethinkDB on both these operating systems.

The RethinkDB source code can be compiled to run on all compatible systems, but there are also precompiled binaries available for some Linux distributions, which make the installation much easier and quicker.

Official packages are available for these platforms:

- Ubuntu
- Debian
- CentOS
- OS X

If you do not run one of these operating systems, you can still check the community-supported packages, or you can build RethinkDB by downloading and compiling the source code. Linux-based operating systems are extremely popular choices at the moment for hosting web services and, more specifically, database services. In the next section, we'll go through how to get RethinkDB running on a few popular Linux distributions: Ubuntu, Debian, CentOS, and Fedora.

Installing RethinkDB on Ubuntu/Debian Linux

There are two ways of installing RethinkDB under Ubuntu. You can install the packages automatically using the so-called repositories, or you can install the server manually. The next couple of sections will walk you through both these methods. In the following example, we will be installing RethinkDB on Ubuntu Linux using the first method. The installation procedure is easy as we will be using Ubuntu's APT package manager.

Before you install RethinkDB, you need to know which version of Ubuntu you are running as the prepackaged binaries are only available for versions 10.04 and above. If you do not know this, an easy way to find out is to run the following command:

```
cat /etc/lsb-release
```
Typing the command into the terminal will give you an output very similar to the following, depending on which version you're running:

```
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04.2 LTS"
```

This output shows that my machine is running Ubuntu 14.04 LTS "Trusty", so we can proceed with the installation of RethinkDB using `apt-get`. To install the server, we first need to add RethinkDB's repository to the list of repositories in our system. We can do this by running the following commands in the terminal:

```
source /etc/lsb-release && echo "deb http://download.rethinkdb.com/apt $DISTRIB_CODENAME main" | sudo tee /etc/apt/sources.list.d/rethinkdb.list
wget -qO- http://download.rethinkdb.com/apt/pubkey.gpg | sudo apt-key add -
```

You may wonder what exactly these commands do. The first line uses the `source` command to export the variables contained in the file `/etc/lsb-release`, whereas the `echo` command constructs the repository string using the `DISTRIB_CODENAME` variable to insert the correct codename for your system. The `tee` command is then used to save the repository URL to the list of repositories in your system. Finally, the last line downloads the GPG key that is used to sign the RethinkDB packages and adds it to the system.

We are now ready to install the server. We can do so by running the following commands in the terminal:

```
sudo apt-get update
sudo apt-get install rethinkdb
```
Introducing RethinkDB

The first line downloads the package list from all the repositories installed on your system and updates them to get information on new packages or updates, whereas the second command actually downloads and installs the server. Once the second `apt-get` command finishes, you will get an output similar to the following screenshot:

If you get no errors, RethinkDB will be installed correctly. Congratulations!

Installing RethinkDB on CentOS and Fedora

The procedure for installing RethinkDB on Fedora and CentOS uses the Yellowdog Updater, Modified (YUM) package manager. The installation procedure consists of two steps: first, add the RethinkDB repository, and second, install the server. The CentOS RPMs are compatible with Fedora, so if you’re running Fedora, you can follow the same instructions.

We can add the RethinkDB `yum` repository to the list of repositories in our system by running the following command in the terminal:

```
sudo wget http://download.rethinkdb.com/centos/6/`uname -m`/rethinkdb.repo -O /etc/yum.repos.d/rethinkdb.repo
```

Next, we can install RethinkDB by executing the following command:

```
sudo yum install rethinkdb
```
The `yum` packet manager will check all the dependencies for RethinkDB and present us with a list of packages to install. Confirm by answering `y` and the installation will start. This will take some time depending on your system's hardware and the number of dependencies to install. If you don’t get any errors, RethinkDB will be installed and ready to be started!

**Installing RethinkDB on OS X**
RethinkDB is compatible with OS X versions 10.7 and above, so be sure to check your version of the operating system before continuing. There are two methods for installing the database on OS X. The first method uses native binaries, whereas the second method uses the Homebrew package manager.

The simplest way to get started is to download the prebuilt binary package and install RethinkDB. The package is available for download from the web at http://www.rethinkdb.com/. Just click on the install link on the home page and choose OS X to download the disk image. Once the download has finished, proceed to mount the image, and you will see a window very similar to this one:

![RethinkDB installer window](image)

The final step is to run the `rethinkdb.pkg` package and follow the instructions. As you may have guessed, the package installs the latest version of RethinkDB. That’s all there is to it! At this point, RethinkDB has been installed and is almost ready to use.
Installing RethinkDB using Homebrew

If you prefer, you can also install RethinkDB using Homebrew, an open source package manager for OS X. Homebrew is a recent addition to the package management tools available for OS X, and its ambition is to require no configuration and automatically optimize packages. However, this requires first installing Xcode—the integrated development environment produced by Apple. If you've already got Xcode installed on your system, you can install Homebrew. If you need full instructions, they are available on the Homebrew website at http://brew.sh/. However, the basic install procedure is to run the following commands from the terminal:

```
ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/install)"
```

This command runs a Ruby script that downloads and installs Homebrew. Once the installation is completed, run the following command to make sure everything is working correctly:

```
brew doctor
```

The output of the `doctor` command will point out all potential issues with Homebrew along with suggestions for how to fix them. We're now ready to install the RethinkDB package; you can do so by running the following commands in the terminal:

```
brew update
brew install rethinkdb
```

As you can imagine, the first command updates the packet manager, whereas the second line installs RethinkDB.

One of the advantages of using a packet manager as Homebrew is that you can update your software extremely easily. When a new version of RethinkDB is released, you can simply update by running the following commands:

```
brew update
brew upgrade rethinkdb
```

By now, you will have RethinkDB installed; if, however, you prefer building it by compiling the source code, we're going to cover that in the next section.
Building RethinkDB from source

Given how easy it is to install RethinkDB using a packet manager, you might wonder why you would want to install the software manually by compiling the source code. There are many reasons why you might want to do so. First, not all Linux distributions support a packet manager as `apt` or `yum`. Installing the database manually also gives you the possibility to specify some custom build flags that may, in some cases, optimize the software that is being installed. Finally, this type of installation also gives you the possibility of running multiple versions of RethinkDB at the same time. Although, installing from source is not a complicated process, it generally makes it more difficult to update RethinkDB when a new version is released.

However, if you still want to go ahead and install the database using the source code, you will need to install the following dependencies:

- GCC
- Protocol Buffers
- jemalloc
- Ncurses
- Boost
- Python 2
- libcurl
- libcrypto

The way in which you install these libraries depends on the system you are using. On Ubuntu, for example, you can install required packages by running the following command in the terminal:

```
sudo apt-get install build-essential protobuf-compiler python libprotobuf-dev libcurl4-openssl-dev libboost-all-dev libncurses5-dev libjemalloc-dev wget
```

Once you have installed all of the dependencies, download and extract the RethinkDB source archive. You can do so by using `wget` to get the source code and `tar` to extract the archive. At the time of writing this book, the latest release was RethinkDB v2.0.3:

```
wget http://download.rethinkdb.com/dist/rethinkdb-2.0.3.tgz

```

```
tar xf rethinkdb-2.0.3.tgz
```

---

[13]
Introducing RethinkDB

RethinkDB uses Autotools for its configuration and build process, so you will have to run the configure command within the extracted folder:

```
cd rethinkdb-2.0.3
./configure --allow-fetch
```

The `configure` command checks if all dependencies are installed and collects some details about the machine on which the software is going to be installed. Then, it uses these details to generate the makefile. The `allow-fetch` argument allows `configure` to install some dependencies if they are missing. Once the makefile has been generated, we can compile the source code:

```
make
```

You will need at least 2 GB of RAM memory to build RethinkDB by compiling the source code.

This will take a few minutes depending on your hardware; at the end of the build process, you will have a screen shown as follows:

```
[300/321] CC build/release/obj/clustering/immediate_consistency/branch/listener.o
[392/321] CC build/release/obj/clustering/immediate_consistency/branch/replier.o
[393/321] CC build/release/obj/clustering/immediate_consistency/branch/history.o
[394/321] CC build/release/obj/clustering/immediate_consistency/branch/metadata.o
[395/321] CC build/release/obj/clustering/immediate_consistency/branch/broadcaster.o
[396/321] CC build/release/obj/clustering/immediate_consistency/query/master_access.o
[397/321] CC build/release/obj/clustering/immediate_consistency/query/direct_reader.o
[398/321] CC build/release/obj/clustering/immediate_consistency/query/master.o
[399/321] CC build/release/obj/clustering/immediate_consistency/query/metadata.o
[400/321] CC build/release/obj/clustering/generic/multi_client_client.o
[410/321] CC build/release/obj/clustering/generic/nonoverlapping_regions.o
[411/321] CC build/release/obj/clustering/generic/nonoverlapping_regions.o
[412/321] BUILD v8_3.38.33.16
[413/321] BUILD v2_20140912
[414/321] ld build/release/rethinkdb.o
[415/321] BUILD build/release/rethinkdb-gdb.py
[416/321] CP .build/drivers/ruby/1.9
[417/321] CONV converts JAXB to Ruby
[418/321] CONVERT протокол drivers/ruby/1.9/lib/q12_pb.rb
[419/321] CP .build/drivers/python/rethinkdb.py
[420/321] CONVERT протокол drivers/python/rethinkdb/q2_pb2.py
[421/321] CONVERT протокол drivers/python/rethinkdb/q2_pb2.py
```

If everything is fine, we are now ready to install RethinkDB:

```
sudo make install
```

That’s it! We now have a brand new RethinkDB database server installed and ready to run. Before we start it, we have some basic configuration to do.
Configuring RethinkDB

Before you start the database server, there is a bit of fiddling to be done with the configuration file; as of now, if the database is correctly installed, you can run the `rethinkdb` command and RethinkDB will start up and create a data file in your current directory. The problem is that RethinkDB does not start up on boot by default and is not configured properly for long-term use. We will go over this procedure in the following section.

Running as a daemon

Once you've got RethinkDB installed, you'll probably want to run it as a daemon—a daemon is a software application or script that runs continuously in the background waiting to handle requests; this is how most production database servers run. You can configure RethinkDB to run like this too.

The default RethinkDB package includes various control scripts including the `init` script `/etc/init.d/rethinkdb`. These scripts are used to start, stop, and restart daemon processes. Depending on how you've installed the database, you probably already have the `init` script installed as packet managers such as `apt` and `yum`.

You can check if the control script is installed by running the following command in the terminal:

```bash
sudo /etc/init.d/rethinkdb status
```

If the `init` script is installed correctly, you will receive an output similar to the following:

```
rethinkdb: No instances defined in /etc/rethinkdb/instances.d/
rethinkdb: See http://www.rethinkdb.com/docs/guides/startup/ for more information
```

This message is normal and indicates that we have not yet created a configuration file for our database. You can now skip to the following section.

Depending on your operating system, the RethinkDB daemon script will be installed into a directory called `init.d` if you're using a SysV-style OS or a directory called `rc.d` for BSD-style systems. The preceding command uses `init.d`, but you must replace it with the correct directory for your system before actually running the command.
Introducing RethinkDB

If, however, the preceding command results in an error such as Command not found, it means that the control script is not installed, and we must proceed to a manual installation. Thankfully, this is as easy as running two commands from the terminal:

```
sudo wget -O /etc/init.d/ https://raw.githubusercontent.com/rethinkdb/rethinkdb/next/packaging/assets/init/rethinkdb
sudo chmod +x /etc/init.d/rethinkdb
```

The first command will download the init script to the correct directory, whereas the second command will give it execution permissions. We are now ready to proceed with the creation of a configuration file for our database.

Creating a configuration file

RethinkDB is installed with a generic configuration file sample, suitable for light and casual use. This is perfect for giving RethinkDB a try but is hardly suitable for a production database application. So, we will now see how to edit the configuration for our database instance.

On some platforms including OS X, a configuration file is not provided; however, you can specify all desired options by passing them as command-line arguments. Running RethinkDB followed by the help command will list all the available command-line options.

Instead of rewriting the full settings file, we will use the provided sample configuration as a starting point and proceed to edit it to customize the settings. The following commands copy the sample conf file into the correct directory and open the nano editor to edit it:

```
sudo cp /etc/rethinkdb/default.conf.sample /etc/rethinkdb/instances.d/instance1.conf
sudo nano /etc/rethinkdb/instances.d/instance1.conf
```

Here, I am using nano to edit the file, but you may use whatever text editor you prefer.

If you built the database by compiling the source code, you may not have the sample configuration file. If this is the case, you can download it from https://github.com/rethinkdb/rethinkdb/blob/next/packaging/assets/config/default.conf.sample.
As you can see, the configuration file is extremely well-commented and very intuitive; however, there are a couple of important entries we need to look at in the following subsections.

- **bind**: By default, RethinkDB will only bind on the local IP address `127.0.0.1`. What this means is that only the server which hosts the database will be able to access the web interface, and no other machine will be able to access the data or join the cluster. This configuration can be useful for testing, but in a production environment where the database is probably running on a separate physical server than the application code, you will need to change this setting. Another reason why you may want to change this setting is if you're running RethinkDB on a cloud server as EC2, and you're accessing the server via SSH. We're going to change this setting so that the database will bind to all IP addresses including public IPs; to set this, just set the bind setting to `all`:

  
  ```
  bind=all
  ```

  Make sure to remove the leading hash symbol (#) as doing this will uncomment the line and make the configuration active.

  Note that there are security implications of exposing your database to the internet. We'll address these issues in *Chapter 6, RethinkDB Administration and Deployment*.

- **driver-port, cluster-port**: These settings let you change the default ports on which RethinkDB will accept connections from client drivers and other servers in the cluster. Generally, they should not be changed unless these ports conflict with any other service that is being executed on the server. You may think that changing the default values could prevent someone from just *guessing* which ports you're using for your database; however, this doesn't really add any layer of security. We will discuss how to secure the database in *Chapter 6, RethinkDB Administration and Deployment*.

- **http-port**: This setting controls which port the HTTP web interface will be accessible on. As with the previous options, change this value only if this port is already in use by another service.

- **join**: The join setting allows you to connect your RethinkDB instance to another existing server to form a cluster. Suppose we have another RethinkDB instance running on a different server that has the IP address `192.168.1.100`, we could connect this database to the existing cluster by editing the join setting as in the following:

  ```
  join=192.168.1.100:29015
  ```
Introducing RethinkDB

- Always remember to activate the setting by uncommenting the line (removing the hash).

Once you've configured all of the options appropriately, save the configuration file and exit the editor. Now that we have our database configured, we're ready to start the database server!

**Starting RethinkDB**

To start your RethinkDB instance, open a terminal window and run the following command:

```
sudo /etc/init.d/rethinkdb start
```

![Note: If you're running an OS that doesn't support the `init` start-stop daemon, such as OS X, you'll have to start RethinkDB by running the `rethinkdb` command followed by the appropriate command-line options. RethinkDB must be run with superuser privileges.]

If the database starts up successfully, you will get an output similar to this:

```
rethinkdb: instance1: Starting instance. (logging to `/var/lib/rethinkdb/instance1/data/log_file')
```

As you can see from the previous output, RethinkDB gives you the path of the log file. If, by any chance, the database refuses to start, check the log for any error. In a clean installation and start up, the log will contain some statements like these:

```
notice: Running rethinkdb 2.0.3-0trusty (GCC 4.8.2)...
note: Running on Linux 3.13.0-36-generic x86_64
notice: Loading data from directory /var/lib/rethinkdb/instance1/data
notice: Listening for intracluster connections on port 29015
notice: Listening for client driver connections on port 28015
notice: Listening for administrative HTTP connections on port 8080
notice: Listening on addresses: 127.0.0.1, 192.168.1.125
notice: Server ready, "rethinkdb1" fa0668f9-99cb-4516-b04a-1ee5466b572c
```

This a great deal of information. The log file tells us that the database has started running correctly, and we can access it from the specified IP addresses. It says that the server is ready and waiting for connections. Congratulations! Now, your RethinkDB server will be up and running. It's time to run your very first ReQL query.
Running a query

Now that we have a RethinkDB installation up and running, let's give it a quick try to make sure that everything is set up correctly. There are a few things that you might want to try to ensure that your database is running correctly. The first step is to try and access the web interface by browsing http://127.0.0.1:8080.

To access RethinkDB's web administration interface, you have to substitute 127.0.0.1 with the public IP that your instance is bound to.

If everything is working correctly, you'll see the RethinkDB web interface:

The web interface allows you to view the status of your cluster and manage each server independently. In the main view, we can see some standard health checks and cluster performance metrics, whereas at the bottom, we can find the most recently logged activities.
Introducing RethinkDB

If we click on the **Tables** link at the top of the page, we can see all the tables that we have added to our database:

![Tables in the cluster](image)

From this page, we can see all the databases that we have in the cluster. Within each database, we can see all the tables that we have created. As you can see from this screenshot, the cluster currently contains one database called **test** but no tables.

The web interface also provides a **Data Explorer** page that we can use to learn the query language and execute queries. If we click on the **Data Explorer** link, we are given an interface that allows us to interact with the server using the query language.

You now have an interactive shell at which you can issue commands. So, let's run our first query! Insert the following query into the **Data Explorer**:

```reql
r.db('test').tableCreate('people')
```

This simple query creates a table called **people** inside the **test** database. If you execute the query by pressing the **Run** button, RethinkDB will create a new table and acknowledge the operation with a JSON document. As you can see, the **Data Explorer** is really easy to use and provides us with a great tool for high-level management of our databases and clusters.

![Data Explorer](image)

Congratulations, you've just executed your first ReQL query! You'll get to learn the ReQL query language much better in the following chapter.
Chapter 1

Downloading the example code
You can download the example code files for this book from your account at http://www.packtpub.com. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

• Log in or register to our website using your e-mail address and password
• Hover the mouse pointer on the SUPPORT tab at the top
• Click on Code Downloads & Errata
• Enter the name of the book in the Search box
• Select the book which you're looking to download the code files
• Choose from the drop-down menu where you purchased this book from
• Click on Code Download

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

• WinRAR / 7-Zip for Windows
• Zipeg / iZip / UnRarX for Mac
• 7-Zip / PeaZip for Linux

Summary
In this first chapter, you learned all about how RethinkDB is different from other databases and what great features it offers us to efficiently store data and power real-time web apps. You also learned how to download and install the database and how to access the web interface. Finally, you learned how to run your first query!

The next chapter will be all about the ReQL query language.
Where to buy this book
You can buy Getting Started with RethinkDB from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.