

Heroku Workbook

Integrating Force.com and Applications on Heroku	3
About the Workbook	3
Intended Audience	3
Before You Begin	3
Step 1: Create a Heroku account	3
Step 2: Create a Salesforce Developer Edition account	4
Step 3: Set up your local development environment	4
Step 4: Install the Heroku Toolbelt	5
Step 5: Log in to Heroku	5
Tutorial #1: Create a Web Application	6
Step 1: Create a Gemfile	6
Step 2: Create the Sinatra app	7
Step 3: Tidying up loose ends	7
Summary	9
Tutorial #2: Adding Force.com OAuth Authorization	10
Step 1: Create a Connected App in Salesforce	11
Step 2: Add OAuth logic to the web application	12
Step 3: Create another Remote App in Salesforce	14
Step 4: Deploy the application to Heroku	14
Step 5: View your application's logs on Heroku	15
Summary	15

Tutorial 3: A Tour Through Heroku	16
Step 1: Scaling	16
Step 2: Add-ons	17
Step 3: Releases and rollback	18
Step 4: One-off dynos	18
Step 5: Staging and multiple applications	20
Summary	20
Tutorial 4: Add Integration with Force.com	21
Step 1: Add session handling	21
Step 2: Use a Force.com integration library	21
Step 3: Deploy to Heroku	23
Summary	23
Tutorial 5: Tidying Up [Optional]	
Step 1: Add a security filter	24
Step 2: Add simple HTML rendering	24
Summary	24

Integrating Force.com and Applications on Heroku

Heroku www.heroku.com provides an ideal platform for building, developing and deploying applications that interact with Salesforce1 APIs. Heroku supports multiple programming languages including Ruby, Java, Node.js and Python, which makes it easy to tap into a vast number of libraries and utilities to help build applications that integrate with APIs, using languages that are very familiar to professional developers.

About the Workbook

This workbook shows one typical example of building an API integration: it creates a web application that lets users sign in (utilizing Force.com's OAuth), and then performs API calls back to Force.com. Along the way, it also demonstrates some of the Heroku platform and how it can be used.

Like Force.com, Heroku is based around applications—it removes many of the barriers around getting a production deployed and scaled—letting you concentrate on writing your app instead. Unlike Force.com, these applications can typically run locally, as well as in the cloud.

For development, this workbook uses the Ruby language and the Sinatra web framework. Ruby is an elegant language that permits us to show these integrations with the least amount of code, thus providing the most clarity.

The source code for the finished application developed in this workbook is available on GitHub at <https://github.com/jonmountjoy/heroku-force-workbook-demo>. Refer to it if you ever need any help.

Intended Audience

This workbook is intended for experienced developers and programmers who may be new to Heroku and/or the Force.com platform. For more information about Heroku see the Getting Started for the language of choice at <https://devcenter.heroku.com>. For more information about developing data-driven cloud applications using Force.com see the Getting Started at <http://developer.force.com/gettingstarteddevelopers>.

Before You Begin

Before you begin the tutorials, you'll need to set up your local development environment, and if you haven't already, get sign up for a free Heroku account.

Step 1: Create a Heroku account

You'll need a Heroku account in order to deploy your application to the cloud:

1. Navigate to <http://heroku.com>.
2. Click Sign Up.
3. Enter your email address.
4. Wait a few minutes for the confirmation email and follow the steps included in the email.

Step 2: Create a Salesforce Developer Edition account

This workbook is designed to be used with a Salesforce Developer Edition organization, or DE org for short. DE orgs are multipurpose environments with all of the features and permissions that allow you to develop, package, test, and install apps.

1. In your browser, go to <http://developer.force.com/join>.
2. In the Email Address field, make sure to use a public address you can easily check from a Web browser.
3. Fill in the rest of the details, and follow the steps included in the confirmation email.
4. Log in to your Force.com Developer Edition Environment, which should look something like this:

The screenshot shows the Salesforce Developer Edition user interface. At the top, there is a search bar and navigation links for Home, Chatter, Campaigns, Leads, Accounts, Contacts, Opportunities, Forecasts, Contracts, Cases, Solutions, Products, Reports, and Dashboards. The main content area is divided into several sections:

- Getting Started:** A "Build App" section with a "Generate a basic app with just one step, and then easily extend that app with clicks or code." and an "Add App" button. Below it is a "Recent Items" table.
- Recent Items:** A table with columns for Name, Type, and Object. It lists various items like Coaching Layout, Performance Cycle Layout, Feedback Question Set Layout, etc.
- Quick Links:** A section with three columns: Tools (App Quick Start, Schema Builder, New custom object), Users (New user, Add multiple users, Reset users' passwords), and App (Manage apps, Manage profiles, Enable Chatter feeds). There are also sections for Security and Data.
- Community:** A section with "Resources", "Find Answers", and "Contribute Ideas" buttons.
- Right Sidebar:** Contains "Resources", "Featured Content", "Events", and "Recommended Apps" sections.

Step 3: Set up your local development environment

This workbook uses Ruby 2.0.0, which you'll need to install on your local development machine.

1. Navigate to <https://www.ruby-lang.org/en/downloads/> and install Ruby 2.0.0. For windows, check out <http://rubyinstaller.org/>. For OS X, try ruby-install <https://github.com/postmodern/ruby-install>

After installation, test that everything is working. Running **ruby -v** should yield something similar to the following:

```
$ ruby -v
ruby 2.0.0p247 (2013-06-27 revision 41674) [x86_64-darwin12.4.1]
```

Step 4: Install the Heroku Toolbelt

The Heroku Toolbelt is a set of command line utilities that make it easy to deploy and manipulate Heroku applications.

1. Navigate to <https://toolbelt.heroku.com/>.
2. Download the Toolbelt appropriate for your operating system, and run the installer.

The Toolbelt installs the CLI—the command line interface that lets you interact with Heroku, as well as Git—a powerful decentralized revision control system, and the means for deploying apps to Heroku, and Foreman, which lets you easily run applications locally as they would run on Heroku.

Test that it's working as follows:

```
$ heroku version
heroku-toolbelt/3.0.0 (x86_64-darwin10.8.0) ruby/1.9.3
```

Type `heroku help` at any time to list all the available commands.

Step 5: Log in to Heroku

At this point you have all the tools installed for developing applications locally, as well as a Heroku and Force.com account. After installing the Toolbelt, you'll have access to the **heroku** command from your command shell. Authenticate using the email address and password you used when creating your Heroku account:

```
$ heroku login
Enter your Heroku credentials.
Email: jon@example.com
Password:
Could not find an existing public key.
Would you like to generate one? [Yn]
Generating new SSH public key.
Uploading ssh public key /Users/jon/.ssh/id_rsa.pub
```

When first run, the **heroku login** command creates a SSH key which it then uploads to Heroku.

You're now ready to start development.

Tutorial #1: Create a Web Application

In this tutorial you will create a minimal web application, which you'll later extend with Force.com integration. This web application will make use of the Sinatra www.sinatrarb.com web framework. You can write similar apps in any number of other frameworks, including Ruby on Rails.

Carry out all the work in a new local directory. Call it **workbook**.

Step 1: Create a Gemfile

A `Gemfile` lets you specify any external dependencies for your application. These will typically be libraries that you will make use of when you write your app. Create such a file, and call it **Gemfile**, with the following content:

```
source 'https://rubygems.org'

ruby '2.0.0'

gem 'thin'
gem 'sinatra'
gem 'omniauth-salesforce'
gem 'force'
```

This ensures that we'll have the **thin** web server available, together with the **sinatra** web framework, as well as a library for the OAuth work we'll do later (**omniauth-salesforce**) and Force.com integration (**force**).

Make sure you have bundler, Ruby's dependency management tool, installed:

```
$ gem install bundler
```

Now instruct bundler to download and install the dependencies:

```
$ bundle install
Fetching gem metadata from https://rubygems.org/..
Resolving dependencies...
Installing daemons (1.1.9)
Using eventmachine (1.0.3)
...
```

That's it. Now that you have all the gems in place, you're ready to start coding.

Step 2: Create the Sinatra app

Here's a simple Sinatra application. Copy this into a file **myapp.rb**:

```
require "sinatra/base"

class MyApp < Sinatra::Base

  get '/' do
    'Home'
  end

  get '/hello' do
    'Hello World'
  end

  run! if app_file == $0
end
```

This web application responds to two different URLs, or routes: The root URL (/) will display "Home", while the **/hello** URL will display "Hello World".

You can now run the application locally. Running it will create the Sinatra web application, which will respond to the two URLs. Run it as follows:

```
$ bundle exec ruby myapp.rb
== Sinatra/1.4.3 has taken the stage on 4567 for development with
backup from Thin
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on localhost:4567, CTRL+C to stop
```

Now open a browser and visit <http://localhost:4567> and <http://localhost:4567/hello>.

Once you have verified your app is running use the **Ctrl-C** keystroke in your terminal to stop the app.

Step 3: Tidying up loose ends

Different applications and language frameworks have different ways of starting the main process. In the above example, we used the **bundle exec ruby myapp.rb** command to start the web process.

To inform Heroku in a standard way as to how to start a particular command, create a file named **Procfile** in the root directory of your application. It should read as follows:

```
web: bundle exec ruby myapp.rb -p $PORT
```

The name on the left (**web**) is called a process type, and the rest of the line defines the command to execute when we ask Heroku to create a dyno running this process type.

You can now run the application locally using the Foreman tool that you installed as part of the Heroku Toolbelt. In your terminal, type **foreman start**:

```
$ foreman start
12:12:31 web.1 | started with pid 27482
12:12:32 web.1 | == Sinatra/1.4.3 has taken the stage on 5000 for
development with backup from Thin
```

This will start up the web application much like before, though the port number has changed. You can view it running at <http://localhost:5000>. Foreman automatically supplied a value for the `$PORT` environment variable. When deployed to Heroku, the `$PORT` variable is set by Heroku.

Step 4: Deploy to Heroku

Now get the app up and running on Heroku. Deploying to Heroku involves sending the source code for the application to Heroku, using Git. Heroku doesn't require any changes to your app - it will run in a very similar manner in the cloud as it does locally. In particular, the Gemfile will be used by Heroku to pull down the dependencies, and the Procfile will be used to understand what command to execute to start the application.

First, add your application to Git:

```
$ git init .
$ git add .
$ git commit -m "First commit"
```

Now create the application on Heroku:

```
$ heroku create
Creating still-cliffs-2432 in organization heroku... done, region is us
http://still-cliffs-2432.herokuapp.com/ | git@heroku.com:still-
cliffs-2432.git
```

This command does a few things. Firstly, it creates a domain for the app (in this case, <http://still-cliffs-2432.herokuapp.com/>). You can visit your domain, but you'll just see an error page displayed. Secondly, it creates a Git remote (with a default name of **heroku**), which you can utilize for deployment. Use **git push heroku master** to deploy the app:

```
$ git push heroku master
Counting objects: 1202, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (391/391), done.
Writing objects: 100% (1078/1078), 374.90 KiB | 412.00 KiB/s, done.
Total 1078 (delta 761), reused 990 (delta 684)

-----> Ruby/Rails app detected
-----> Using Ruby version: ruby-2.0.0
```



```

-----> Installing dependencies using Bundler version 1.3.2 Counting
objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.19 KiB | 0 bytes/s, done.
Total 4 (delta 2), reused 0 (delta 0)

-----> Ruby/Rack app detected
-----> Using Ruby version: ruby-2.0.0
-----> Installing dependencies using Bundler version 1.3.2
      Running: bundle install --without development:test --path
vendor/bundle --binstubs vendor/bundle/bin --deployment
      Using daemons (1.1.9)
      Using eventmachine (1.0.3)
...
-----> Discovering process types
      Procfile declares types      -> web
      Default types for Ruby/Rack -> console, rake

-----> Compiled slug size: 26.9MB
-----> Launching... done, v6
      http://still-cliffs-2432.herokuapp.com deployed to Heroku

```

The long log output shows Heroku receiving your application, fetching the dependencies, discovering the Procfile, and deploying the bundled application (called a slug).

Now visit your application on the web:

```
$ heroku open
```

Your browser should open to a URL that corresponds to the domain for the app. In our case it was <http://still-cliffs-2432.herokuapp.com>. You should see “Hello” displayed.

Congratulations. You've deployed your first application to Heroku.

Summary

In this tutorial you created a simple web application, and ran it locally. You also created a Procfile, which instructs Heroku how to run the application. Finally you deployed it to Heroku using standard Git commands. Along the way you learned:

- How a Procfile defines the command Heroku can use to start your application.
- How to run an application locally using Foreman.
- How to create a Heroku application with `heroku create`.`
- How to add changes to your code to a git repository with **git add.** and **git commit.**
- How to deploy an application with **git push heroku master.**

Tutorial #2: Adding Force.com OAuth Authorization

To get your app talking to Force.com, you'll need to set up a "Connected App" on the Salesforce side, then configure your web app to connect to it through OAuth. In Force.com parlance, a Connected App refers to an external application that can securely connect to Force.com over Identity and Data APIs. Connected Apps essentially describe a relationship between an external application and a Salesforce user, and define an entry point into the platform. Connected Apps use standard OAuth 2.0 (or simply, OAuth) to authenticate, provide Single Sign-On, and acquire access tokens for use with Salesforce APIs.

In this tutorial you will create a Connected App and add OAuth to the web application you just built, using OAuth to authorize users of the web application to access Force.com resources. OAuth lets users authorize applications to access Force.com resources (via the Force.com REST and SOAP APIs) on their behalf without revealing their passwords or other credentials to those applications.

For this example, your application doesn't need to handle authentication explicitly; it simply needs to ensure that a valid access token accompanies all interactions with the API. You can typically download an OAuth library to do the heavy lifting. We're using one called **omniauth**.

Step 1: Create a Connected App in Salesforce

Your first step is to configure your Force.com development environment to allow OAuth. You do this by creating a "Remote App."

1. Log in to your Salesforce Developer Edition environment at: <https://login.salesforce.com>
2. Click on Setup in the upper right-hand menu.
3. Under **Build** click **Create** -> **Apps**.
4. Scroll to the bottom and click **New** under **Connected Apps**.

Enter the following details for the remote application:

1. For **Connected App Name**, enter a name for your Connected App.
2. The **API Name** will be filled in automatically. Accept this value.
3. Enter an email address for **Contact Email**.
4. Select **Enable OAuth Settings** under the API dropdown.
5. For **Callback URL** provide the callback URL of your local app, <http://localhost:5000/auth/salesforce/callback> You will configure your web application to respond to this URL in the next step.

6. Give the app **Full access** scope

The screenshot shows the 'API (Enable OAuth Settings)' configuration page. It includes a checked 'Enable OAuth Settings' checkbox, a 'Callback URL' text input with the value 'http://localhost:5000/auth/salesforce/callback', and an unchecked 'Use digital signatures' checkbox. Below these are two columns: 'Selected OAuth Scopes' (empty) and 'Available OAuth Scopes' (containing a list of permissions). The 'Full access (full)' scope is highlighted in the 'Available OAuth Scopes' list. Between the two columns are 'Add' and 'Remove' buttons.

7. Click **Save**

After saving, you'll see that Force.com provides you two values, a consumer key and a consumer secret. Save these in a text file called **.env** in the root of your workbook project using the following format:

```
SALESFORCE_KEY=<insert key here>
SALESFORCE_SECRET=<insert secret here>
```

When you run your application with Foreman, these variables are made available to the application as environment variables. A similar mechanism exists in Heroku called config vars. This ability to separate configuration from code is a powerful paradigm, that facilitates running the same application in multiple contexts (such as locally or remote, or staging and production).

Step 2: Add OAuth logic to the web application

You're now ready to add OAuth logic to the web application.

Add these lines to the top of **myapp.rb** (after **require "sinatra/base"**), to ensure you have the necessary libraries available at run time:

```
require "omniauth"
require "omniauth-salesforce"
```

After the class declaration (`class MyApp < Sinatra::Base`), add a configuration block to support logging and sessions. It also ensures we can catch exceptions.

```
class MyApp < Sinatra::Base

  configure do
    enable :logging
    enable :sessions
    set :show_exceptions, false
    set :session_secret, ENV['SECRET']
  end
end
```

...

In most web frameworks, you need to provide a "secret" that's used to help encrypt the session information. In this case you instruct Sinatra to use the secret specified in the **SECRET** environment variable.

Modify your `.env` file and add a new line:

```
SECRET=some_random_string
```

After the configuration block, set up **omniauth** to use the key and secret that you're supplying in the environment:

```
configure do
  enable :logging
  enable :sessions
  set :session_secret, ENV['SECRET']
end

use OmniAuth::Builder do
  provider :salesforce, ENV['SALESFORCE_KEY'], ENV['SALESFORCE_SECRET']
end
```

Doing this makes your app automatically respond to a few routes, such as `/auth/salesforce`. Finally, add a few callback methods to support OAuth.

```
get '/authenticate' do
  redirect "/auth/salesforce"
end

get '/auth/salesforce/callback' do
  logger.info "#{env["omniauth.auth"]["extra"]["display_name"]}
  authenticated"
  env["omniauth.auth"]["extra"]["display_name"]
end

get '/auth/failure' do
  params[:message]
end

get '/unauthenticate' do
  session.clear
  'Goodbye - you are now logged out'
end
```

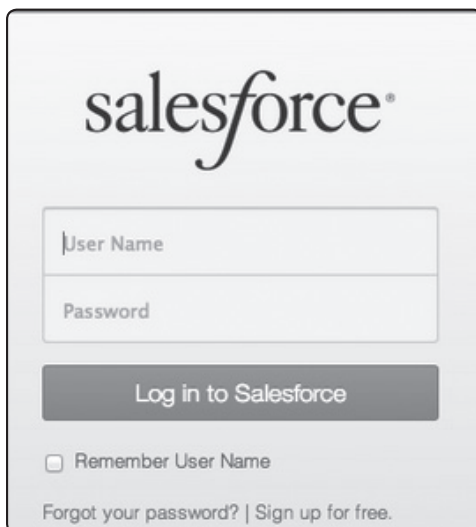
The route `/authenticate` starts the whole OAuth process, which **omniauth** handles for us. `/auth/failure` will only be called in the event of an error. The most important route is `/auth/salesforce/callback` which will be called on a successful authentication. After a successful authorisation, we log and output the display name of the person who logged in.

Finally, let's redefine the one existing route to add a little logging

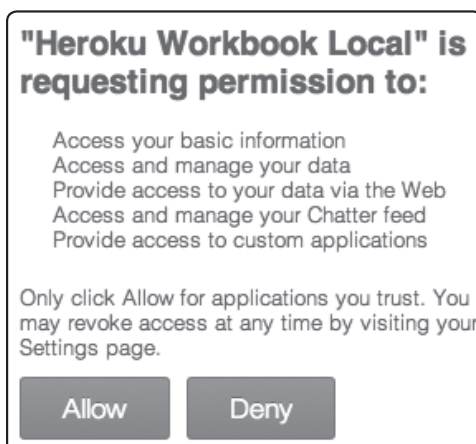
```
get '/' do
  logger.info "Visited home page"
  'Home'
end
```

Start the application again by executing `foreman start`, and then navigate to <http://localhost:5000/authenticate>

You will now be redirected to the Salesforce login screen if you are not already logged in.



After authenticating, you'll be asked to authorize your web application. When creating the Connected App you set the OAuth scope to **Full**. As a result, Force.com is now asking you to grant it full access to data on your Force.com org - thereby allowing you to later use the API and much more.



Click Allow.

Your name will then be displayed. By just implementing a few methods, and making use of a standards-based authorization method, you have added a vital piece of integration to your application.

Although at this point all you see is your name, a considerable number of things happened in the background. After navigating to **/authenticate** your app redirected to **/auth/salesforce**, part of the **omniauth** OAuth implementation, which then kicked off the entire OAuth process. That process included you being redirected to Force.com for login and OAuth authorization granting, and back again to the **/auth/salesforce/callback** in your application after the successful authorization. That authorization also transfers information about the Force.com user back to your application - and here you just extracted the name portion of the person who authorized. Phew!

Step 3: Create another Remote App in Salesforce

In order for the application to work when deployed to Heroku, you need to create another Remote Application in Force.com, because OAuth is typically URL based and the secret and key you created in Step 1 are tied to your localhost URL. You now need to create another set for use with your Heroku application's domain.

1. Repeat Step 1, but for the callback URL, use a URL based on your application's name. For example, if your application is "still-cliffs-2432", then your URL will be <https://still-cliffs-2432.herokuapp.com/auth/salesforce/callback>. If you've forgotten your application's name, type in `heroku info`. Note you must use "https" here.
2. Take a note of the new key and secret.

Step 4: Deploy the application to Heroku

You are now ready to deploy this revision of the application to Heroku. Earlier, you created a **.env** file for providing environment variables that can be used during local execution. You don't want this file stored in a repository, so as a first step, add this file to a **.gitignore** file - which prevents git from processing it.

Do this by creating a new file, **.gitignore** and adding a single line to it, **.env**.

Now perform the same actions as in Tutorial 1: updating your local git repository, and pushing it to Heroku for deployment:

```
$ git add .
$ git commit -m 'Add OAuth'
$ git push heroku master
```

When executing your application locally, you use the **.env** file to hold its configuration. When executing it on Heroku, you use the "config variables", which you can set through the command line.

Take the key and secret you generated in Step 2 for the new connected app, and configure your Heroku application to use them:

```
$ heroku config:set SECRET="some_random_string"
$ heroku config:set SALESFORCE_KEY=xxxxxxx SALESFORCE_SECRET=yyyyy
```

Now navigate to the **/authenticate** route of your deployed application. For example:
<http://still-cliffs-2432.herokuapp.com/authenticate>

Your application, running on Heroku, will now behave just like it did in Step 2 - taking you through the Salesforce login and OAuth authorization.

Step 5: View your application's logs on Heroku

Heroku makes it easy to view the logs generated by your application. It treats logs as a stream of events, and aggregates all the logs from your application and other Heroku components (such as the router) into a single, consolidated stream.

View the consolidated logs, tailing it to see all new log events as they come in from your terminal:

```
$ heroku logs --tail
2013-10-17T10:18:06.459201+00:00 app[web.1]: 81.31.127.166 - - [17/Oct/2013 10:18:06] "GET /authenticate HTTP/1.1" 302 - 0.0015
2013-10-17T10:18:06.620394+00:00 heroku[router]: at=info method=GET path=/auth/salesforce host=still-cliffs-2432.herokuapp.com fwd="81.31.127.166" dyno=web.1 connect=6ms service=23ms status=302 bytes=345
2013-10-17T10:18:06.619535+00:00 app[web.1]: 81.31.127.166 - - [17/Oct/2013 10:18:06] "GET /auth/salesforce HTTP/1.1" 302 345 0.0052
2013-10-17T10:18:10.955579+00:00 app[web.1]: 81.31.127.166 - - [17/Oct/2013 10:18:10] "GET /auth/salesforce/callback?display=page&code=aPrxV...HTTP/1.1" 302 - 0.8447
2013-10-17T10:18:10.954778+00:00 app[web.1]: I, [2013-10-17T10:18:10.954593 #2] INFO -- : Visited home page
```

If you access the home page for your application now, you should see the "Viewing home page" message logged.

Summary

In this tutorial you added the keystone of any integration—authentication and authorization. You used an off-the-shelf library, `omniauth``, which implements the OAuth 2.0 standard that Force.com supports. Along the way you learned:

- How Heroku lets you externalize the configuration for an application in config vars. The local application and the application on Heroku are deployed with a different set of credentials, but the application source code is exactly the same.
- How to set up a Connected App in Salesforce to facilitate OAuth.
- How Heroku consolidates logs into a single stream, and how to view them using `heroku logs``.

Tutorial 3: A Tour Through Heroku

Heroku is designed from the ground up for developer productivity, focused on removing the pains of managing infrastructure and operations so you and your team can focus on delivering amazing apps.

You've already seen in the previous two tutorials how Heroku is focused on the application: you provided Heroku with the application source code and it builds and deploys and manages the application for you.

In this tutorial you will explore some of the other features of the platform. You can skip this section if you want to get directly to adding Force.com integration in the next chapter.

Step 1: Scaling

Heroku uses a simple, powerful process model to support fast, efficient and tunable scale. “Dynos” are the basic unit of scale on Heroku. A dyno is a lightweight, virtualized container running a single user-specified command.

The Procfile you created earlier, tells Heroku how to start up a web dyno. You could easily add other process types to that Procfile, so that you can tell Heroku how to run the queueing system for example. Any web dynos you start are automatically connected to Heroku's routers, and will receive web traffic.

Run `heroku ps` to view the state of your running dynos:

```
$ heroku ps
=== web (1X): `bundle exec ruby myapp.rb -p $PORT`
web.1: up 2013/10/16 00:55:50 (~ 30s ago)
```

This indicates you have a single dyno running.

Scale this up to two dynos, and run `heroku ps` again:

```
$ heroku ps:scale web=2
Scaling web dynos... done, now running 2
$ heroku ps
=== web (1X): `bundle exec ruby myapp.rb -p $PORT`
web.1: up 2013/10/16 11:41:32 (~ 40s ago)
web.2: up 2013/10/16 11:41:32 (~ 9s ago)
```

You now have twice as many instances of your application running.

Note: Scaling could in theory trigger costs (see below), so Heroku prevents scaling or provisioning add-ons until your account is verified by the addition of a credit card. Trying to execute the command above will result in directions to verify your account. Please remember to scale your application back to one dyno to avoid incurring charges.

Finally, scale it back to 1:


```
$ heroku ps:scale web=1
```

Note: Heroku automatically credits each app with 750 free dyno-hours per month. By running two web dynos continuously, you will exceed your monthly allowance, so it's worthwhile scaling it back to one. In addition, if your app has only a single web dyno running (which you may do while using the free dyno hours), that web dyno will sleep after 30 minutes of inactivity. Simply visiting the application again will unidle it after a short delay. This doesn't happen in production, where you typically have more than two web dynos running.

Step 2: Add-ons

Add-ons addons.heroku.com are fully managed third-party services created by top providers, integrated into the Heroku platform so they can be easily added, scaled and consumed by your application.

Add-ons can be added through one command, and come in a variety of plans. There are a number of add-ons, providing all sorts of services ranging from database persistence to caching, monitoring, email delivery and logging. In this step you'll add a logging add-on.

Heroku only stores the last 1500 lines of log history, but the logging infrastructure has an API and several add-on partners have built services around it, which persist the logs, lets you search them and set up alerts.

Add the free plan for the papertrail add-on:

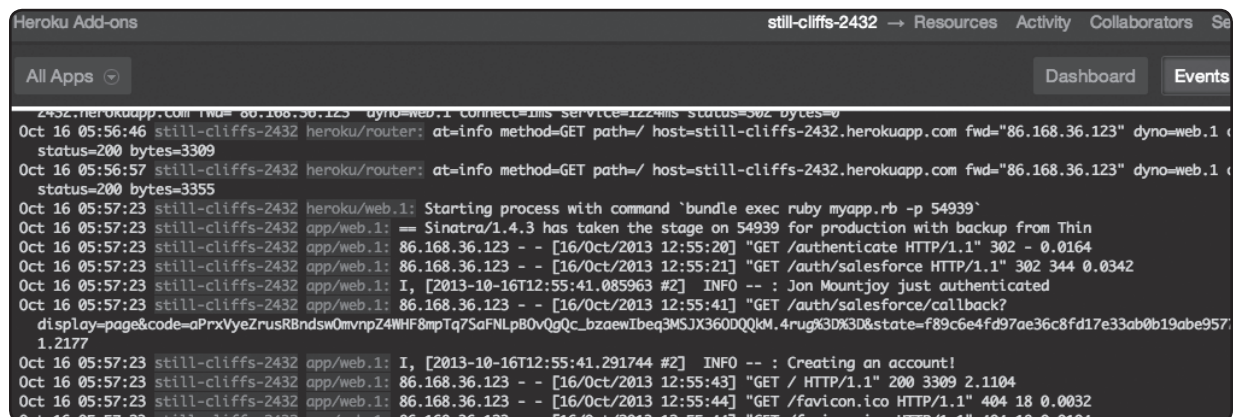
```
$ heroku addons:add papertrail
Adding papertrail on blazing-galaxy-997... done, v6 (free)
Use `heroku addons:docs papertrail` to view documentation.
```

Now visit your application a few times so that you do in fact generate logs. Just refreshing the home page of the app should cause the "Visited homepage" message to be logged. It may initially take a minute or two for the the logs to start showing up in papertrail.

Now visit the add-on's console. This console is part of papertrail, not Heroku, but can be accessed quite simply through the command line:

```
$ heroku addons:open papertrail
```

Explore the papertrail add-on functionality by clicking through to the Dashboard and Events tabs.



```
Heroku Add-ons still-cliffs-2432 → Resources Activity Collaborators Se
All Apps Dashboard Events
2432.herokuapp.com fwd=86.168.36.123 dyno=web.1 connect=ms service=122ms status=302 bytes=0
Oct 16 05:56:46 still-cliffs-2432 heroku/router: at=info method=GET path=/ host=still-cliffs-2432.herokuapp.com fwd="86.168.36.123" dyno=web.1
status=200 bytes=3309
Oct 16 05:56:57 still-cliffs-2432 heroku/router: at=info method=GET path=/ host=still-cliffs-2432.herokuapp.com fwd="86.168.36.123" dyno=web.1
status=200 bytes=3355
Oct 16 05:57:23 still-cliffs-2432 heroku/web.1: Starting process with command `bundle exec ruby myapp.rb -p 54939`
Oct 16 05:57:23 still-cliffs-2432 app/web.1: -- Sinatra/1.4.3 has taken the stage on 54939 for production with backup from Thin
Oct 16 05:57:23 still-cliffs-2432 app/web.1: 86.168.36.123 - - [16/Oct/2013 12:55:20] "GET /authenticate HTTP/1.1" 302 - 0.0164
Oct 16 05:57:23 still-cliffs-2432 app/web.1: 86.168.36.123 - - [16/Oct/2013 12:55:21] "GET /auth/salesforce HTTP/1.1" 302 344 0.0342
Oct 16 05:57:23 still-cliffs-2432 app/web.1: I, [2013-10-16T12:55:41.085963 #2] INFO -- : Jon Mountjoy just authenticated
Oct 16 05:57:23 still-cliffs-2432 app/web.1: 86.168.36.123 - - [16/Oct/2013 12:55:41] "GET /auth/salesforce/callback?
display=page&code=aPrxVyeZrusRBndsw0mvpZ4NHf8mpTq7SaFNLp80V0gQc_bzaewIbeq3MSJX360DQqkM.4rugk3DX3D&state=f89c6e4fd97ae36c8fd17e33ab0b19abe957
1.2177
Oct 16 05:57:23 still-cliffs-2432 app/web.1: I, [2013-10-16T12:55:41.291744 #2] INFO -- : Creating an account!
Oct 16 05:57:23 still-cliffs-2432 app/web.1: 86.168.36.123 - - [16/Oct/2013 12:55:43] "GET / HTTP/1.1" 200 3309 2.1104
Oct 16 05:57:23 still-cliffs-2432 app/web.1: 86.168.36.123 - - [16/Oct/2013 12:55:44] "GET /favicon.ico HTTP/1.1" 404 18 0.0032
Oct 16 05:57:23 still-cliffs-2432 app/web.1: 86.168.36.123 - - [16/Oct/2013 12:55:44] "GET /favicon.ico HTTP/1.1" 404 18 0.0184
```

As with most add-ons, we're using the free add-on plan here. Other plans will offer additional functionality. For example storing your logs for longer periods of time.

Step 3: Releases and rollback

Whenever you deploy code, change a config var, or add or remove an add-on resource, Heroku creates a new release and restarts your app. You can list the history of releases, and use rollbacks to revert to prior releases for backing out of bad deploys or config changes.

List the releases you've already made:

```
$ heroku releases
=== blazing-galaxy-997 Releases
v6  Add papertrail:choklad add-on      jon@heroku.com
    2013/10/15 11:26:59 (~ 34m ago)
v5  Add SECRET_KEY config              jon@heroku.com
    2013/10/15 12:00:10 (~ 59s ago)

v4  Deploy 9579f23                      jon@heroku.com
    2013/10/15 10:23:32 (~ 1h ago)
v3  Deploy 0eb78aa                      jon@heroku.com
    2013/10/15 10:21:33 (~ 1h ago)
v2  Enable Logplex                      heroku@herokumanager.com
    2013/10/15 09:58:21 (~ 2h ago)
v1  Initial release                    heroku@herokumanager.com
    2013/10/15 09:58:20 (~ 2h ago)
```

Rollback to one of the initial releases:

```
$ heroku rollback v2
Rolling back blazing-galaxy-997... done, v7
!    Warning: rollback affects code and config vars; it doesn't add
or remove addons. To undo, run: heroku rollback v7
```

Now if you refresh you will now hit the old version of your application - without OAuth. The rollback itself will have created another release, which you'll find when you execute `heroku releases` again. To get back to the state you were in before the rollback, rollback to the penultimate release.

Step 4: One-off dynos

The set of dynos declared in your Procfile and managed by the dyno manager via `heroku ps:scale` are known as the dyno formation. These dynos do the app's regular business (such as handling web requests and processing background jobs) as it runs.

But when you wish to do one-off administrative or maintenance tasks for the app, you'll want to spin up a one-off dyno using the heroku run command. This creates a new dyno with your application, but instead of running one of the usual commands you specified in the Procfile, runs something else and attaches the output to the console.

Ruby comes with a built-in REPL called **irb**. Start it now in a one-off dyno using the **heroku run** command:

```
$ heroku run irb
Running `irb` attached to terminal... up, run.9582
```

You'll be able to exec Ruby commands in that dyno. For example:

```
irb(main):001:0> puts ENV['SECRET']
some_skeycret_key
=> nil
```

Hit Ctrl + C.

One-off dynos do a great job at showing some of how Heroku works. Instead of running a Ruby command, create a one-off dyno that simply starts a Unix shell. This will allow us to look around.

```
$ heroku run bash
Running `bash` attached to terminal... up, run.9424
~ $ ls -l
total 32
drwx----- 2 u31972 31972 4096 2013-10-17 12:30 bin
-rw----- 1 u31972 31972  108 2013-10-17 12:30 Gemfile
-rw----- 1 u31972 31972 1198 2013-10-17 12:30 Gemfile.lock
-rw----- 1 u31972 31972 1698 2013-10-17 12:30 myapp.rb
-rw----- 1 u31972 31972   40 2013-10-17 12:30 Procfile
-rw----- 1 u31972 31972 1916 2013-10-17 12:30 README.md
drwx----- 5 u31972 31972 4096 2013-10-17 12:30 vendor
drwx----- 2 u31972 31972 4096 2013-10-17 12:30 views
~ $ head myapp.rb
require "sinatra/base"
require 'force'
require "omniauth"
require "omniauth-salesforce"

class MyApp < Sinatra::Base

  configure do
    enable :logging
~ $ rm myapp.rb
~ $ exit
```

As you can see, once you've started the one-off dyno you can issue standard Unix commands, look at files, and even delete them. This won't change your application in any way though - dynos are distinct and don't share anything - and every time you start a new one, the same compiled slug is used.

Step 5: Staging and multiple applications

It is a common practice to maintain more than one environment for each application. For example, a staging and production environment for each app along with any number of ephemeral test environments for features in various stages of development.

Heroku provides an efficient and productive way to create new environments, by providing **heroku fork**. This command copies an existing application, including add-ons, config vars and any database it may have.

```
$ heroku fork
Creating fork still-cliffs-2999... done
Copying slug... done
Copying config vars... done
Fork complete, view it at http://still-cliffs-2999.herokuapp.com/
```

You'll be able to visit the URL of the new application, but it should produce an error. Remember that Connected Apps, used in OAuth, are tied to URLs. This new app has a new URL. So to really set this up as a staging environment, you'd simply create a new Connected App and provide this application with a different set of config vars.

To simplify the rest of the workbook, delete the staging application (make sure you delete the staging and not the production!):

```
$ heroku apps:delete --app still-cliffs-2999
```

Summary

In this tutorial you explored a subset of Heroku functionality that increases developer productivity. Along the way you learned:

- How easy it is to scale applications using **heroku ps:scale**.
- How the large add-on marketplace can easily extend the functionality of your app.
- How to view releases, and rollback deployments of your application.
- How to run one-off dynos.
- How to create copies of your application for staging environments using **heroku fork**.

Tutorial 4: Add Integration with Force.com

In this tutorial you will make calls to Force.com from within your web application. To do this we utilize a simple Force.com integration gem, **force.rb**, which wraps the API and provides a simple interface for using the Force.com REST API. In this tutorial you will extend your application to integrate with Force.com, listing all the Accounts in your [Force.com](#) org.

Step 1: Add session handling

Your OAuth callback handler, which responds to `/auth/salesforce/callback`, doesn't do anything except print the user's name. The OAuth callback actually receives a lot more data from Force.com after a successful authentication, including an OAuth token that can be used for subsequent calls to Force.com, and an `instance_url` which tells us which Force.com API endpoint to use.

In this step you will modify the callback handler to save these data in the client's session, so that they can be used in future calls.

Change the callback method so that it reads as follows:

```
get '/auth/salesforce/callback' do
  credentials = env["omniauth.auth"]["credentials"]
  session['token'] = credentials["token"]
  session['refresh_token'] = credentials["refresh_token"]
  session['instance_url'] = credentials["instance_url"]
  redirect '/'
end
```

This takes the token, refresh token and URL and saves them into the session, redirecting the user to the home page. Note that it will no longer print the user's name, but instead redirect to the `/` route.

Step 2: Use a Force.com integration library

Now, set everything up so that you can call the Force.com API. Add the following line to the top of your app (the “myapp.rb” file):

```
require 'force'
```

Create a helper method that uses the `force` gem. This method, `client`, creates a new instance of the main client class, and populates it with the URL, tokens and OAuth credentials. After the class declaration (“`class MyApp < Sinatra::Base`”), add the following:

```
helpers do
  def client
    @client ||= Force.new instance_url: session['instance_url'],
                          oauth_token: session['token'],
                          refresh_token: session['refresh_token'],
                          client_id: ENV['SALESFORCE_KEY'],
                          client_secret: ENV['SALESFORCE_SECRET']
  end
end
```

Now modify the route for the home page to use this client to display Account data:

```
get '/' do
  logger.info "Visited home page"
  accounts= client.query("select Id, Name from Account")
  accounts.map(&:Name).join(',')
end
```

This simply makes a call to the `query()` method on the client, passing in some SOQL. (This in turn, will make a call using the Force.com REST API.) It then takes the resulting list of accounts, extract the `Name` fields, and concatenate them all. It's not pretty, but you get the idea!

As with most programming, it's always better to catch exceptions. The following two methods do that. The first ensures that general errors are displayed. The second ensures that if a session has expired that the user is redirected to authenticate again:

```
error Force::UnauthorizedError do
  redirect "/auth/salesforce"
end
error do
  "There was an error. Perhaps you need to re-authenticate to /
authenticate ? Here are the details: " + env['sinatra.error'].name
end
```

Start your application again:

```
$ foreman start
```

Now authenticate again (remember that now it will store data in the session, so you'll need to authenticate again or you'll just receive an error):

<http://localhost:5000/authenticate>

After authentication, you'll be redirected to the home page, which will query Force.com, and then display the account data.

GenePoint,United Oil & Gas, UK,United Oil & Gas, Singapore,Edge Communications,Burlington Textiles Corp of America,Pyramid Construction Inc.,Dickenson plc,Grand Hotels & Resorts Ltd,Express Logistics and Transport,University of Arizona,United Oil & Gas Corp.,sForce

With a single line of code (`client.query`) you have called the Force.com API. The force gem is pretty powerful, allowing you to tap into the full Force.com API using simple methods on the client, including creating, finding, searching and upserting data. See <https://github.com/heroku/force.rb> to learn more.

Step 3: Deploy to Heroku

Deploying this change to Heroku doesn't require anything new - perform the same actions as before:

```
$ git add .  
$ git commit -m "add force.com integration"  
$ git push heroku master
```

As in the previous step, now visit your application's `/authenticate`` route to start the ball rolling. For example:

<http://still-cliffs-2432.herokuapp.com/authenticate>

Well done. You've just deployed an app to Heroku that performs Salesforce authorisation and integration.

Summary

In this tutorial you added integration with Force.com. The integration uses the credentials retrieved from the OAuth to establish a connection to Force.com using a Ruby library that wraps the Force.com API. Performing a SOQL query is as simple as `client.query("select Id, Name from Account")`.

Tutorial 5: Tidying Up [Optional]

You've written a bare-bones application that authenticates and performs a simple callout to Salesforce. Let's extend it in a few very small but important ways, to give you the basis of doing a lot more.

Step 1: Add a security filter

Typically, you are going to want to make sure that your application fails a little more gracefully if someone tries to access any routes in your application without first being authorised. You can do this by using a powerful feature of Sinatra called a filter, which is run before any method that matches some regular expression signature. Add the following to your application after the class definition:

```
before /^(?!\/(auth.*))/ do
  redirect '/authenticate' unless session[:instance_url]
end
```

The regular expression `^(?!\/(auth.*))` matches all routes that do not begin with "auth". If someone tries to reach, say `/hello`, then the filter will fire, and if that user doesn't have an **instance_url** in their session, then they'll be redirected to **/authenticate**.

By now, you should know how to start your local application server, or deploy to Heroku, to view these changes, so we'll write out the instructions!

Step 2: Add simple HTML rendering

You typically wouldn't just return a bare string in a web application, as you currently do in your home page method. In this step you'll add simple HTML rendering - passing the list of accounts to a parameterised view, which will iterate through each account and display its data.

Modify the method handling the ``/`` route to read as follows:

```
get '/' do
  logger.info "Visited home page"
  @accounts= client.query("select Id, Name from Account")
  erb :index
end
```

This modification assigns the list of accounts to a local variable, `@accounts`, and then asks **erb** to render a file called "index". ERB is a simple templating system built into Ruby. By default, Sinatra will now look for a file called ``index.erb`` in a new subdirectory called **views**.

Create the subdirectory `views` and the file **index.erb**, using this simple template:

```
<table>
<tr>
  <th>Account</th>
  <th>ID</th>
</tr>
<% @accounts.each do |account| %>
<tr>
  <td><%= account.Name %></td>
  <td><%= account.Id %></td>
</tr>
<% end %>
</table>
```

This template creates an HTML table, iterating over each account in the `@account` variable, creating a row for each.

Summary

In this tutorial you learned how to add a before filter in Sinatra, and how to extend the application with basic templating.

Next Steps

Here are some suggested next steps:

- Visit <https://devcenter.heroku.com/articles/quickstart> to learn how to get started with Heroku using Ruby, Node.js, Java, Python and other languages.
- Visit <https://devcenter.heroku.com/articles/how-heroku-works> to learn many more details about how Heroku works.
- Visit <https://devcenter.heroku.com/articles/architecting-apps> to learn how to architect applications well.
- Visit <https://github.com/heroku/force.rb> to learn more about the `force` integration gem, and its capabilities.
- Visit http://wiki.developerforce.com/page/Force.com_workbook to get started with the Force.com platform, Visualforce, Apex and REST and SOAP APIs.
- Learn more about Force.com from the Technical Library at <http://wiki.developerforce.com/page/Wiki>.
- Follow Force.com Developer and Engineering blogs at <http://blogs.developerforce.com>.
- Get involved in Force.com Discussion Boards at http://boards.developerforce.com/sforce/?category_id=developers

Notes

Notes