Appium Bootcamp 6: Running Your Tests

Appium Bootcamp is a series of articles prepared by Selenium guru Dave Haeffner, and leading Appium contributor Matthew Edwards, for Sauce Labs. Dave also authors the Elemental Selenium website, which includes tips for using Selenium, and where you can sign up for his weekly email on the topic of Selenium testing. He is also the author of the Selenium Guidebook.

- Quick Setup
- Simple Rake Tasks
- Running Your Tests In Sauce
- Specifying Sauce Credentials
- Making Your Sauce Runs Descriptive
- Outro

Now that we have our tests written, refactored, and running locally it's time to make them simple to launch by wrapping them with a command-line executor. After that, we'll be able to easily add in the ability to run them in the cloud.

Quick Setup

`appium_lib` comes pre-wired with the ability to run our tests in Sauce Labs, but we're still going to need two additional libraries to accomplish everything; `rake` for command-line execution, and `sauce_whisk` for some additional tasks not covered by `appium_lib`.

Let's add these to our `Gemfile` and run `bundle install`.

```
# filename: Gemfile
source 'https://rubygems.org'
gem 'rspec', '~> 3.0.0'
gem 'appium_lib', '~> 4.0.0'
gem 'appium_console', '~> 1.0.1'
gem 'rake', '~> 10.3.2'
gem 'sauce_whisk', '~> 0.0.13'
```

Simple Rake Tasks

Now that we have our requisite libraries let's create a new file in the project root called `Rakefile` and add tasks to launch our tests.

```
# filename: Rakefile
desc 'Run iOS tests'
task :ios do
  Dir.chdir 'ios'
  exec 'rspec'
end
desc 'Run Android tests'
task :android do
  Dir.chdir 'android'
  exec 'rspec'
end
```

Notice that the syntax in this file reads a lot like Ruby -- that's because it is (along with some Rake specific syntax). For a primer on Rake, read this.

In this file we've created two tasks. One to run our iOS tests, and another for the Android tests. Each task changes directories into the correct device folder (e.g., `Dir.chdir`) and then launches the tests (e.g., `exec 'rspec'`).

If we save this file and run `rake -T` from the command-line, we will see these tasks listed along with their descriptions.

```
> rake -T
rake android  # Run Android tests
rake ios      # Run iOS tests
```

If we run either of these tasks (e.g., `rake android` or `rake ios`), they will execute the tests locally for each of the devices.

Running Your Tests In Sauce

As I mentioned before, `appium_lib` comes with the ability to run Appium tests in Sauce Labs. We just need to specify a Sauce account username and access key. To obtain an access key, you first need to have an account. After that, log into the account and go to the bottom left of your dashboard; your access key will be listed there.

We'll also need to make our apps available to Sauce. This can be accomplished by either uploading the app to Sauce, or, making the app available from a publicly available URL. The prior approach is easy enough to accomplish with the help of `sauce_whisk`.

Let's go ahead and update our `spec_helper.rb` to add in this new upload capability (along with a couple of other bits).

```
# filename: common/spec_helper.rb
require 'rspec'
require 'appium_lib'
require 'sauce_whisk'
def using_sauce
  user = ENV['SAUCE_USERNAME']
  key  = ENV['SAUCE_ACCESS_KEY']
  user && !user.empty? && key && !key.empty?
end
def upload_app
  storage = SauceWhisk::Storage.new
  app = @caps[:caps][:app]
  storage.upload app
  @caps[:caps][:app] = "sauce-storage:#{File.basename(app)}"
end
def setup_driver
  return if $driver
  @caps = Appium.load_appium_txt file: File.join(Dir.pwd, 'appium.txt')
  if using_sauce
    upload_app
    @caps[:caps].delete :avd # re: https://github.com/appium/ruby_lib/issues/241
  end
  Appium::Driver.new @caps
end
def promote_methods
  Appium.promote_singleton_appium_methods Pages
  Appium.promote_appium_methods RSpec::Core::ExampleGroup
end
setup_driver
promote_methods
RSpec.configure do |config|
  config.before(:each) do
    $driver.start_driver
  end
  config.after(:each) do
    driver_quit
  end
end
```

Near the top of the file we pull in `sauce_whisk`. We then add in a couple of helper methods (`using_sauce` and `upload_app`). `using_sauce` checks to see if Sauce credentials have been set properly. `upload_app` uploads the application from local disk and then updates the capabilities to reference the path to the app on Sauce's storage.

We put these to use in `setup_driver` by wrapping them in a conditional to see if we are using Sauce. If so, we upload the app. We're also removing the a `vd` capability since it will cause issues with our Sauce run if we keep it in.

Next we'll need to update our `appium.txt` files so they'll play nice with Sauce.

```
# filename: android/appium.txt
[caps]
appium-version = "1.2.0"
deviceName = "Android"
platformName = "Android"
platformVersion = "4.3"
app = "../../../apps/api.apk"
avd = "training"
[appium_lib]
require = ["./spec/requires.rb"]
```

```
# filename: ios/appium.txt
[caps]
appium-version = "1.2.0"
deviceName = "iPhone Simulator"
platformName = "ios"
platformVersion = "7.1"
app = "../../../apps/UICatalog.app.zip"
[appium_lib]
require = ["./spec/requires.rb"]
```

In order to work with Sauce we need to specify the `appium-version` and the `platformVersion`. Everything else stays the same. You can see a full list of Sauce's supported platforms and configuration options here.

Now let's update our Rake tasks to be cloud aware. That way we can specify at run time whether to run things locally or in Sauce.

```
desc 'Run iOS tests'
task :ios, :location do |t, args|
  location_helper args[:location]
  Dir.chdir 'ios'
  exec 'rspec'
end
desc 'Run Android tests'
task :android, :location do |t, args|
  location_helper args[:location]
  Dir.chdir 'android'
  exec 'rspec'
end
def location_helper(location)
  if location != 'sauce'
    ENV['SAUCE_USERNAME'], ENV['SAUCE_ACCESS_KEY'] = nil, nil
  end
end
```

We've updated our Rake tasks so they can take an argument for the location. We then use this argument value and pass it to `location_helper`. The `location_helper` looks at the location value -- if it is not set to `'sauce'` then the Sauce credentials get set to `nil`. This helps us ensure that we really do want to run our tests on Sauce (e.g., we have to specify both the Sauce credentials AND the location).

Now we can launch our tests locally just like before (e.g., `rake ios`) or in Sauce by specifying it as a location (e.g., `rake ios['sauce']`)

But in order for the tests to fire in Sauce Labs, we need to specify our credentials somehow. We've opted to keep them out of our `Rakefile` (and our test code) so that we can maintain future flexibility by not having them hard-coded; which is also more secure since we won't be committing them to our repository.

Specifying Sauce Credentials

There are a few ways we can go about specifying our credentials.

Specify them at run-time

```
SAUCE_USERNAME=your-username SAUCE_ACCESS_KEY=your-access-key rake ios['sauce']
```

Export the values into the current command-line session

```
export SAUCE_USERNAME=your-username export SAUCE_ACCESS_KEY=your-access-key
```

Set the values in your bash profile (recommended)

```
# filename: ~/*.bash_profile
...
export SAUCE_USERNAME=your-username
export SAUCE_ACCESS_KEY=your-access-key
```

Making Your Sauce Runs Descriptive

It's great that our tests are now running in Sauce. But it's tough to sift through the test results since the name and test status are nondescript and all the same. Let's fix that.

Fortunately, we can dynamically set the Sauce Labs job name and test status in our test code. We just need to provide this information before and after our test runs. To do that we'll need to update the RSpec configuration in `common/spec_helper.rb`.

```
# filename: common/spec_helper.rb
...
RSpec.configure do |config|
  config.before(:each) do |example|
    $driver.caps[:name] = example.metadata[:full_description] if using_sauce
    $driver.start_driver
  end
  config.after(:each) do |example|
    if using_sauce
      SauceWhisk::Jobs.change_status $driver.driver.session_id, example.exception.nil?
    end
    driver_quit
  end
end
```

In `before(:each)` we update the `name` attribute of our capabilities (e.g., `caps[:name]`) with the name of the test. We get this name by tapping into the test's metadata (e.g., `example.metadata[:full_description]`). And since we only want this to run if we're using Sauce we wrap it in a conditional.

In `after(:each)` we leverage `sauce_whisk` to set the job status based on the test result, which we get by checking to see if any exceptions were raised. Again, we only want this to run if we're using Sauce, so we wrap it in a conditional too.

Now if we run our tests in Sauce we will see them execute with the correct name and job status.

Outro

Now that we have local and cloud execution covered, it's time to automate our test runs by plugging them into a Continuous Integration (CI) server.