

Extracted from:

Deploying Rails

Automate, Deploy, Scale, Maintain, and Sleep at Night

This PDF file contains pages extracted from *Deploying Rails*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2010 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The
Pragmatic
Programmers

Deploying Rails

Automate, Deploy,
Scale, Maintain,
and Sleep at Night



Anthony Burns
and Tom Copeland

Edited by Brian P. Hogan

Deploying Rails

Automate, Deploy, Scale, Maintain, and Sleep at Night

Anthony Burns

Tom Copeland

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at <http://pragprog.com>.

Copyright © 2012 Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.

ISBN-13: 978-1-93435-695-1

Printed on acid-free paper.

Book version: B1.0—January 19, 2012

6.6 Monitoring Applications

The monitoring techniques we've seen have all been useful with any application; `check_disk` is helpful for a Ruby on Rails application, a Java application, or a static web site. But Nagios can also monitor an application's specific data thresholds. Let's see how that's done.

We've seen how Nagios uses plugins to bring in new functionality and we've written our own plugin to monitor Passenger memory usage. Now we'll write another small plugin to check MassiveApp's activity. Actually, most of the logic will be in MassiveApp itself; we'll write just enough of a plugin to connect to MassiveApp and report a result.

For the specifics of this check, consider MassiveApp's daily growth rate in terms of accounts. In any 24 hour span we get around a dozen new accounts. If we get many more than that, we want to get an alert so we can think about firing up more servers.

We could do this check in a few different ways. We could query the MySQL database directly, but that would mean we couldn't use our ActiveRecord models with their handy named scopes and such. We could use HTTP to hit a controller action, but then we'd want to ensure that the action could only be accessed by Nagios. So we'll keep it simple by using a Rake task. First we'll declare the task; we can put this in `lib/tasks/monitor.rake`. We're namespacing the task inside nagios; this keeps all our Nagios-related tasks in one place:

Download `monitoring/monitor1.rake`

```
namespace :nagios do
  desc "Nagios monitor for recent accounts"
  task :accounts => :environment do
    end
end
```

Next let's count the number of "recently" created accounts; in this case "recently" means "in the past 24 hours". We can do this with a straightforward ActiveRecord query:

Download `monitoring/monitor2.rake`

```
namespace :nagios do
  desc "Nagios monitor for recent accounts"
  task :accounts => :environment do
    recent = Account.where("created_at > ?", 1.day.ago).count
    end
end
```

In this plugin we'll ask for a warning if we get more than 50 new accounts in a day and we'll consider it critical if we get more than 90 in a day.

Download monitoring/monitor.rake

```

namespace :nagios do
  desc "Nagios monitor for recent accounts"
  task :accounts => :environment do
    recent = Account.where("created_at > ?", 1.day.ago).count
    msg, exit_code = if recent > 90
      ["CRITICAL", 2]
    elsif recent > 50
      ["WARNING", 1]
    else
      ["OK", 0]
    end
    puts "ACCOUNTS #{msg} - #{recent} accounts created in the past day"
    exit exit_code
  end
end

```

We've got the Rake task in MassiveApp's codebase now; next up, we need Nagios to be able to run it. We can do this with a simple Bash script. We'll name this script `check_recent_accounts` and put in our nagios Puppet module in `modules/nagios/files/plugins/` alongside our `check_passenger` plugin. That script needs to run our Rake task using the `--silent` flag to prevent the usual "(in /path/to/the/app)" output message since that would confuse Nagios. It also needs to relay the exit code from the Rake task on to Nagios. We can do that using the Bash special parameter `$?` which holds the exit code of the last command executed:

Download monitoring/check_recent_accounts

```

#!/bin/bash
cd /home/vagrant/massiveapp/current/
RAILS_ENV=production /usr/bin/rake --silent nagios:accounts
exit $?

```

Switching back into Puppet mindset, we'll add another file resource to our `nagios::client` class that will move our script into place:

Download monitoring/check_recent_accounts.pp

```

"/usr/lib/nagios/plugins/check_recent_accounts":
  source => "puppet:///modules/nagios/plugins/check_recent_accounts",
  owner  => nagios,
  group  => nagios,
  mode   => 755;

```

We'll also need to add the new command to `nrpe.cfg`:

```

command[check_recent_accounts]=/usr/lib/nagios/plugins/check_recent_accounts

```

Let's run Puppet to get the script and the new `nrpe.cfg` in place. Then we can execute a trial run of this plugin in the same way that we've exercised other plugins; we'll just run it from the shell:

```
app $ /usr/lib/nagios/plugins/check_recent_accounts
ACCOUNTS WARNING - 70 accounts created in the past day
```

We need to tell Nagios about our new check, so we'll add it to `commands.cfg`:

[Download monitoring/commands.cfg](#)

```
define command {
    command_name check_recent_accounts
    command_line /usr/lib/nagios/plugins/check_recent_accounts
}
```

And we'll add this to our app checks:

[Download monitoring/nagios-service-definitions](#)

```
define service {
    use generic-service
    service_description Recent Accounts
    host_name app
    check_command check_nrpe_larg!check_recent_accounts
}
```

Another Puppet run and everything's in place; now Nagios will let us know if (or when) we get a mad rush of new users.

This is the sort of check that only needs to be run once for MassiveApp. That is, when MassiveApp grows to encompass a few servers, we won't run this on each server as we'd do with `check_ssh` and `check_passenger`. Instead, we'd designate one host to run this check and that would alert us if the thresholds were exceeded.