



# WHY ENGINE YARD IS BETTER THAN DO IT YOURSELF

Comparisons between Platform as a  
Service and self-administered infrastructure

## Introduction

Those unfamiliar with PaaS options may at times ask, “what’s the true benefit of using a Platform as a Service?” They elaborate by saying, “heck, I can install Ruby (or Node.js, PHP, MySQL, PostgreSQL, etc.), deploy my application and monitor the systems myself!” This is definitely true. There are thousands of companies doing their own DevOps today and that pattern works for them.



Write code or get tied down building a new cluster?

Where a PaaS like Engine Yard really shines is when a company doesn’t have the developer resources, in-house expertise, or contractor budget to properly manage their production infrastructure. A PaaS allows a development team of any size to focus on their application instead of their infrastructure, thus making them more productive and providing more “bang for the buck” with development dollars spent. Which would you rather do as a developer: write code, or get tied down in several days of **yak shaving** while building a new production cluster? And what if that cluster has a hardware failure at 4AM—how do you feel about being “on call”?

We are going to lay out some explanation on what it takes to build, monitor, support and manage, a production web application cluster at medium scale, and then contrast that with equivalent steps when using Engine Yard PaaS.

Running your own production setup on your own has its place and its merits. Developers can learn so much by running their own production cluster, and those lessons will help them write more stable and efficient applications. However, as with all things, there are tradeoffs of time and effort (and therefore, budget) to consider, and in those cases a PaaS may very well be a better option, especially for small teams. If you’re on the fence, not sure which way to go, this ebook aims to illuminate the differences in control, time and cost to help you make an informed decision that best benefits your team, your client(s), your project and your users long-term.

## The Basics

Let's start with the basics. In most modern applications, you'll want to use a cloud hosting provider due to the cost savings and disposability of virtual machines. At Engine Yard, we use Amazon EC2 for our underlying infrastructure. You have plenty of choices out there including EC2, Rackspace Cloud, and HP Cloud. Each have their pros and cons. For the purposes of this ebook, we're going to compare a "DIY" setup with Engine Yard.

Some organizations and/or applications may be better suited to running bare metal in a co-located data center. Remember that you'll likely have some complex contracts and logistics come up in the process of putting hardware in that data center. A fully managed solution will be simpler to have set up, but you won't own the hardware and such solutions can be rather expensive, also at times involving contractual obligations.

After deciding where your application will be hosted and on what type of platform, you would need to make a choice as to your Linux distribution. Our general advice would be to use whatever Linux distribution your team feels most comfortable with and that has the best overall community and/or commercial support.

# Building a Cluster

Once you’ve made the above basic decisions and assessed your traffic expectations, you can start building out an initial production cluster. For the purposes of this ebook, we’re going to assume you’ve decided to run a Rails app on Amazon EC2, or a similar cloud service, directly.

As you can see from the following comparison, you have complete and total control over the individual specific functions of setting up a cluster on your own, but at a significant trade off: time and effort. The amount of work we’ve seen put into standing up a cluster in some cases can range from a day to a week, depending on the complexity involved and how many surprises get thrown your way.

With Engine Yard, or any properly built PaaS frankly, that “day to week” timeframe is shrunk down to a matter of minutes. The process involved in the Engine Yard section can be completed in under an hour in most cases (issues of loading existing database and DNS propagation withstanding). However, there is a trade off here too: flexibility. When using any form of automation, by the very nature of the beast, you’ll have less flexibility. Which can prompt another question: “How can I get that flexibility back, and how much flexibility do I need?”

On Engine Yard, we solve this flexibility problem with Chef. An automated configuration tool written in Ruby, Chef is designed to be easy to learn and to help you automate and

DO IT YOURSELF	ENGINE YARD
<div><div>1</div>Create security group and manage SSH keys for you and your staff</div> <div><div>2</div>Configure security group to allow traffic on ports 443, 80, and 22 (SSH)</div> <div><div>3</div>Choose a virtual machine size and boot from an AMI.</div> <div><div>4</div>Update package manager.</div> <div><div>5</div>Upgrade all available base packages to latest versions to avoid known security vulnerabilities.</div> <div><div>6</div>Install Ruby and RubyGems, probably from source since some distributions tend to lag behind on patchlevel vs. the latest available, and you should keep your version of Ruby patched to the latest patch level available to avoid commonly known security vulnerabilities.</div> <div><div>7</div>Create another server in your security group for your database.</div>	<div><div>1</div>Upload your SSH public key through a web-based GUI.</div> <div><div>2</div>Create an “application” in the dashboard. Specify the git URI for your application’s source code.</div> <div><div>3</div>Use the dashboard to create an environment and designate your public key as having access.</div> <div><div>4</div>Select the instance size you want for your application, database and utility servers, and how many of each you want, all on one screen.</div> <div><div>5</div>Click “Boot”.</div> <div><div>6</div>Click “Deploy”<div><div>•</div>Optional: migrate the database for a fresh app with no data by clicking the “migrate” checkbox</div><div><div>•</div>Optional: Ignore the “migrate” checkbox and load your data by hand if coming from another app (a SQL dump for example)</div></div> <div><div>7</div>Verify the app is functioning as expected.</div>
Continued Next Page	Continued Next Page

## DO IT YOURSELF

- 8 Configure security groups to allow access to the database server on its running port from only other members of the same security group and on port 22 (SSH). Deny all other access on all other ports.
- 9 Install and configure your database of choice.
  - Optional: Load the database schema and data if you have it (for apps with existing data)
  - Optional: create the database and then migrate it up and seed it later in your initial deploy
- 10 Install and configure your application server of choice (Thin, Passenger, Unicorn, Puma, etc.) on your application instance.
- 11 Install and configure a front-end web server: e.g. nginx or Apache. Integrate it with your application server (tell nginx about the Unix socket that Unicorn is storing all its requests in; Passenger is much easier but may not be appropriate for all apps).
- 12 Configure Passenger workers for global queue and “always on” workers for single apps needing 24/7/365 constant performance if using Passenger
- 13 Set up deployment tools of choice (e.g. Capistrano)
  - Remember to have it clean up after itself; leaving old releases around will slowly but surely eat up all your disk space!
- 14 Configure your application to look at your single database master.
- 15 Deploy code.
- 16 Start your front end and back end application servers.
- 17 Test your application’s functionality.
- 18 Deploy your code again to test that your deploys work as expected.
- 19 Test the application again to be sure that deploys didn’t break anything.
- 20 Obtain an Elastic IP address and attach it to the one/primary application instance you have running.
- 21 Tweak DNS entries to point your domain to that IP address.

Time Estimate: 2–4 days

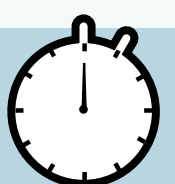


## ENGINE YARD

- 8 Adjust DNS to point to the EIP already issued and wait for propagation.
- 9 High five everyone in the office.



Time Estimate: 45 minutes–1 hour





standardize system configuration by writing code. Since virtual machines are treated as disposable commodities, you need configuration automation to make certain that all nodes in any given cluster are identical and that their configurations are repeatable in the future, even when being stood up from scratch from an empty data volume.

Using [custom Chef recipes](#), you can exercise nearly 100% total control over your cluster configuration. Once written, properly tested and in use, custom Chef recipes allow you to

#### Get flexibility with Chef recipes

automatically configure any aspect of any of your clustered instances at boot time, meaning that you can trash one instance and spin up another one and not have to worry about configuration at all. It just happens automatically when you have custom Chef recipes uploaded for a given environment.



## Scaling

Any application that's built to be successful is eventually going to need to scale. In this example we're starting out with the most basic cluster we realistically can operate in production: one application instance and one database master instance.

At some point however, you're going to need to add more application instances to handle load from end users. Additionally, you may run into performance issues using a single database master for both reads and writes. At scale, it just won't keep up and will take forever to return query results to your application instances, so you'll need to horizontally scale your database as well.

### Scaling the Application Tier

Let's start with a discussion on scaling the application tier. You'll need to duplicate your existing application server, which is easy if the entire thing is EBS-backed on EC2. However, having an entire instance based on EBS can make for slow performance, so you may not have opted for that route. Either way, you need a way to quickly and easily make another of those servers.

Once that's done, you also need a way to tell incoming traffic to be load balanced between those servers. You can do this with a virtual load balancing appliance such as an [Elastic Load Balancer](#), or with software, such as [haproxy](#).

Next, you have to deploy your code to the second (or third, fourth, fifth, and so on) application server in your cluster and verify that it's been set up correctly, is secured, has been added to the load balancer pool, and is actually serving up the right code.

Finally, you have to alter your deployment process to push your new code to all application servers in your cluster, and devise a strategy for doing it with minimal downtime.

If you have an application that takes uploaded assets and does something with them, please note: Moving from a single application server to a multiple application server cluster is going to cause you pain no matter how you do it. The reason being that a request will go to one application server with the uploaded data (say, a forum avatar for example) but not the other(s). Then it gets saved on disk on one of the application instances, but not on the others, meaning that future requests to `/path/to/wherever/your/static/assets/are/avatar.png` returns a 404 on all servers except the one that processed the original upload.

This is why you really need to upload your assets to Amazon S3 or a similar storage service (Rackspace Cloud Files for example). Many gems support this pattern already, and for non-Ruby users, I'm sure there are libraries and/or examples on how to achieve similar results in your language of choice.

- 1 Add another application server and do everything you already did before above when standing up the server in the first place, minus the database related parts. That may take you an entire day, or more if you run into surprises.
  - Optional: If you did set it up to be fully EBS backed, you could snapshot the application server volume, then boot another instance and create a separate volume for it based off that snapshot. That would definitely be far faster, but performance may not be adequate if you have to frequently read or write a lot of information to the disk.
- 2 Add an Elastic Load Balancer or install a software based load balancer on your primary application instance (where the EIP is attached).
  - If you add an ELB, you have to move the EIP to the ELB, or modify DNS with the IP address of the ELB. (Sorry about the TLA's, it's just part of DIY.)
  - If you use a software based load balancer, you'll need to configure it with the host-name and port of the other servers in the cluster. This means you'll likely have to run the load balancer on port 80/443, and then change your front-end web server to run on, for example, 81/444 and forward traffic to those ports on the other machines. This is how we do it with haproxy right now, for example.

- 1 Go to the environment screen, click "add", select an application instance and click "add to cluster".
- 2 No, seriously, that's it. Off to the beach!



Time Estimate: 1-2 days



Time Estimate: 20 minutes



## Scaling the Database Tier

Now that you've added your application instances, you'll want to scale out your database tier with one or more replicas. To do this, your application first should be configured via whatever gem or library you choose to perform reads from replicas and writes to the master. This applies primarily to standard SQL databases (MySQL, PostgreSQL).



Let's start by examining what you'd have to do to add a single database slave to a MySQL cluster. PostgreSQL is discussed on the next page.

## DO IT YOURSELF

- 1 Create a new instance, update all packages, and install MySQL; verify that you're running the same version on both Master and Slave.
- 2 Configure the master to use binlogs and restart MySQL.
- 3 On the Master, flush tables to disk and lock the master so that no further data gets written. Wait for the flush to finish.
- 4 Issue `SHOW MASTER STATUS\G` on the master and record the binlog file name and position.
- 5 Initiate a snapshot in your AWS console, or start an rsync of data from the master to the slave.
- 6 Once the snapshot starts and you're certain that it's running, you can release the database lock.
  - If you went the rsync route, you have to wait for the entire rsync to finish before you can release the database lock. Otherwise the data on disk will change right underneath rsync and you won't be able to reliably re-establish replication.
- 7 On the master, create a replication user with replication privileges.
- 8 On the slave, use the `CHANGE MASTER` statement in the console to specify the master hostname/IP address, port, username, password, binlog file and position.
- 9 Issue `START SLAVE`
- 10 Check that it's indeed replicating by issuing `SHOW SLAVE STATUS\G` every few minutes to check that it's advancing (look at "seconds behind master"). Eventually, depending on the size of the data, it should catch up and continue being in-step with the master.

Repeat this for each slave you want to add to your cluster.

## ENGINE YARD

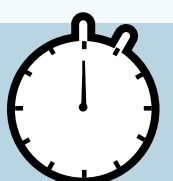
- 1 On the environment page, click "Snapshot".
- 2 Once complete, click "Add".
- 3 Add a database slave and select the latest snapshot, then click "Add to Cluster".
- 4 Break for pizza.



Time Estimate: 1–2 days



Time Estimate: 20–40 minutes



When establishing MySQL replication, recording the position and filename of the binlog on the database master is of paramount importance because that's how your slave knows where to start replicating from. In situations where an rsync is required to move data across (as opposed to a snapshot, which can include any changes that occur during its execution in the snapshot itself) from master to slave, you'll need to keep the database locked until the rsync is complete to avoid possible data loss or other issues.

Setting up replication with PostgreSQL is different—at least from the “DIY” standpoint. There are multiple replication strategies that can be used for different reasons and architectures, and you'd first have to assess which is best for your application and cluster configuration. Once that's done, you would need to create new PostgreSQL replica servers, rsync files from the master to the replica, make configuration changes and edits on both, and then restart the servers. The PostgreSQL wiki has a great getting started tutorial [here](#). Note that on Engine Yard, all this is handled for you through the exact same interface as mentioned for MySQL above: just “add to cluster” and you're done.

Additionally, being alerted to problems with replication is also a key factor. On Engine Yard this is already monitored for you; on your own, you would need to enable monitoring of replication and the overall health of both databases on your own.

## Database Backups

Having a database replica or two in your cluster is always a good practice to help you quickly recover from a database-impacting event by promoting a replica to master status, but that shouldn't be the complete sum of your database backup strategy. You should regularly snapshot and/or perform SQL dumps of your data and store them.

### DO IT YOURSELF

- 1 Establish a database replica. At any form of scale, you probably shouldn't be executing SQL dumps or snapshots against the master because various locks and I/O performance issues can come into play, so your snapshot/sqldump should be taken from the replica.
- 2 Create an automated script in bash, ruby, python, etc. to run a SQL dump for your database of choice (e.g. mysqldump) and then ship the dumped file off to a remote storage location, Amazon S3 for example.
- 3 Test the script to ensure that it works.
- 4 Create a cron job entry to execute that script at your lowest traffic time(s) every so often—maybe once or twice per day.
- 5 Configure cron to email you if a problem occurs.
- 6 Create some form of automation to purge old SQL dumps from S3 (or your storage provider of choice) and ensure that it runs at regular intervals.
- 7 Test all of the above thoroughly to be sure that it works as expected.

### ENGINE YARD

- 1 By default, this is already done. All you have to do is tell Engine Yard Cloud how many backups to keep, and how long to keep them. That's it.
- 2 Time to kick back and relax.



Time Estimate: 1–2 days



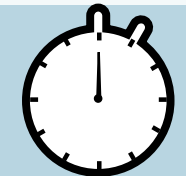


Time Estimate: None. Built-in to the product.



# Log Management

Any sufficiently sized application will result in a multitude of logs being created, and without proper log management in place, can fill a disk rather quickly.

DO IT YOURSELF	ENGINE YARD
<ol style="list-style-type: none"><li>1 Identify each log on the system that needs to be rotated by logrotate.d.</li><li>2 Create configuration files in /etc/logrotate.d for each of them.</li><li>3 Restart logrotate.d.</li></ol>	<ol style="list-style-type: none"><li>1 Grab your friends and head out for drinks, because this is already handled by default.</li></ol> 
<p>Time Estimate: 2–3 hours</p> 	<p>Time Estimate: None. Built-in to the product.</p> 

## Monitoring

The DIY sysadmin’s job is still not done. Once your cluster is built and running the way you want it, you’ll need to implement a monitoring system of some form.

There are two basic “observational vectors” for monitoring. The first is what we’ll call “internal”—monitoring that’s internal to the system, running on a host that may warn you if memory and swap become dangerously low, or if CPU usage spikes.

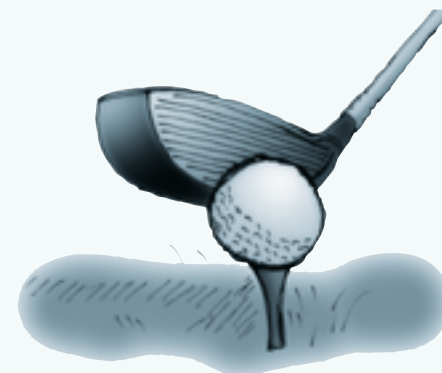
The second observational vector we consider to be “external”— e.g. a site uptime monitor. You need to know if your site goes down and you’ll need a separate service, such as [SiteUptime](#) or [Pingdom](#), or a multitude of others, to be able to alert you if that happens.

## DO IT YOURSELF

- 1 Make a choice for system monitoring. There are many tools out there: commercial, open source, brand new or long-standing and battle-tested. Your choice.
- 2 Go through the documentation for the tool of your choice and install the monitoring tool.
- 3 Write configuration files to monitor what you consider to be important. This may not be a simple task whatsoever depending on the tool you choose.
- 4 Ensure that alerts are e-mailed to you at an appropriate address.

## ENGINE YARD

- 1 Enter your e-mail address at which you want to receive alerts in the environment dashboard.
- 2 Knock off work early for a round of golf—you're done.



Time Estimate: 1–2 days



Time Estimate: 3 minutes



Setting up external monitoring is quite a bit easier as it usually involves purchasing a monthly service from a third party, such as pingdom or siteuptime.com. However, depending on your choices, you may have additional fees vs. what's already built into the Engine Yard platform.

In addition to the monitoring available to you by default with Engine Yard, you can use the [AppFirst](#) and/or [New Relic](#) addons to obtain metrics about your application(s) and servers. Enabling these on Engine Yard takes a matter of minutes, whereas doing so by hand may take hours to days, depending on how billing accounts need to be set up and departmental approval that may be needed.

# Expect and Respond to the Unexpected

Every production system at one point or another experiences an unexpected event. Hardware failure is far from unheard of on systems like Amazon EC2, weather events can impact operations, user load can spike unexpectedly, a multitude of security vulnerabilities could be published or exploited, and so on. Unfortunately, these things have a tendency to happen at the worst possible times. Murphy's law applies: the least opportune time for a system to go down is when it will. So the question then becomes: how fast can you respond?

## DO IT YOURSELF

- 1 Receive an alert at an inopportune time.
- 2 Drop what you're doing; race home, get up in the middle of the night, or get to your office.
- 3 Assess the situation: how urgent is it? Is the site down or just sluggish?
- 4 Diagnose the problem.
- 5 Call in whatever assets you need to address the problem; other developers if needed, for example.
- 6 Begin work to repair the issue, possibly pushing new code to address the problem, or potentially restarting services, moving IP addresses, conducting failovers, and other items as needed.

## ENGINE YARD

There are two possible ways these situations are handled with Engine Yard.

Standard support customers:

- 1 Receive alert
- 2 File ticket with support team
- 3 Let them handle it

Premium support customers:

- 1 Receive the alert
- 2 Roll over and go back to sleep because the support team will automatically handle it for you and contact you if they need your compliance (e.g. push new code) to fix the problem.



Time Estimate: 1–2 days



Time Estimate: None. Built-in to the product.





## Summary

In summary, running your own application cluster at scale can be a very time consuming process to get set up properly. Even when “finished,” one still has to manually respond to issues and events, apply security patches and upgrades, and keep software versions up to date. A Platform as a Service offering like Engine Yard automates the vast majority of that process without sacrificing flexibility or control, allowing a development team to focus purely on their application.

At the end of the day, for any small to medium sized business, and for departments within the enterprise, it's about budget and money. You're going to spend money on hardware (physical or virtual) one way or another. The question therefore becomes, is the cost of a PaaS less than, or greater than, that of a systems administrator, and do you necessarily \*need\* a full time systems administrator for your application? In some cases you absolutely will need a dedicated systems administrator, but in most cases, a PaaS can provide agility, capabilities, and access to expertise otherwise not attainable at such a low cost.



Engine Yard, 500 Third Street, Suite 510, San Francisco, CA 94107  
[www.engineyard.com](http://www.engineyard.com) • [sales@engineyard.com](mailto:sales@engineyard.com) • 1-866-518-9273 • 1-415-624-8380

Copyright©2013 Engine Yard, Inc. All rights reserved. Engine Yard is a trademark of Engine Yard, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned may be trademarks of their respective companies. Cloud is a registered trademark or trademark of Engine Yard Inc. In the United States and/or other jurisdictions.