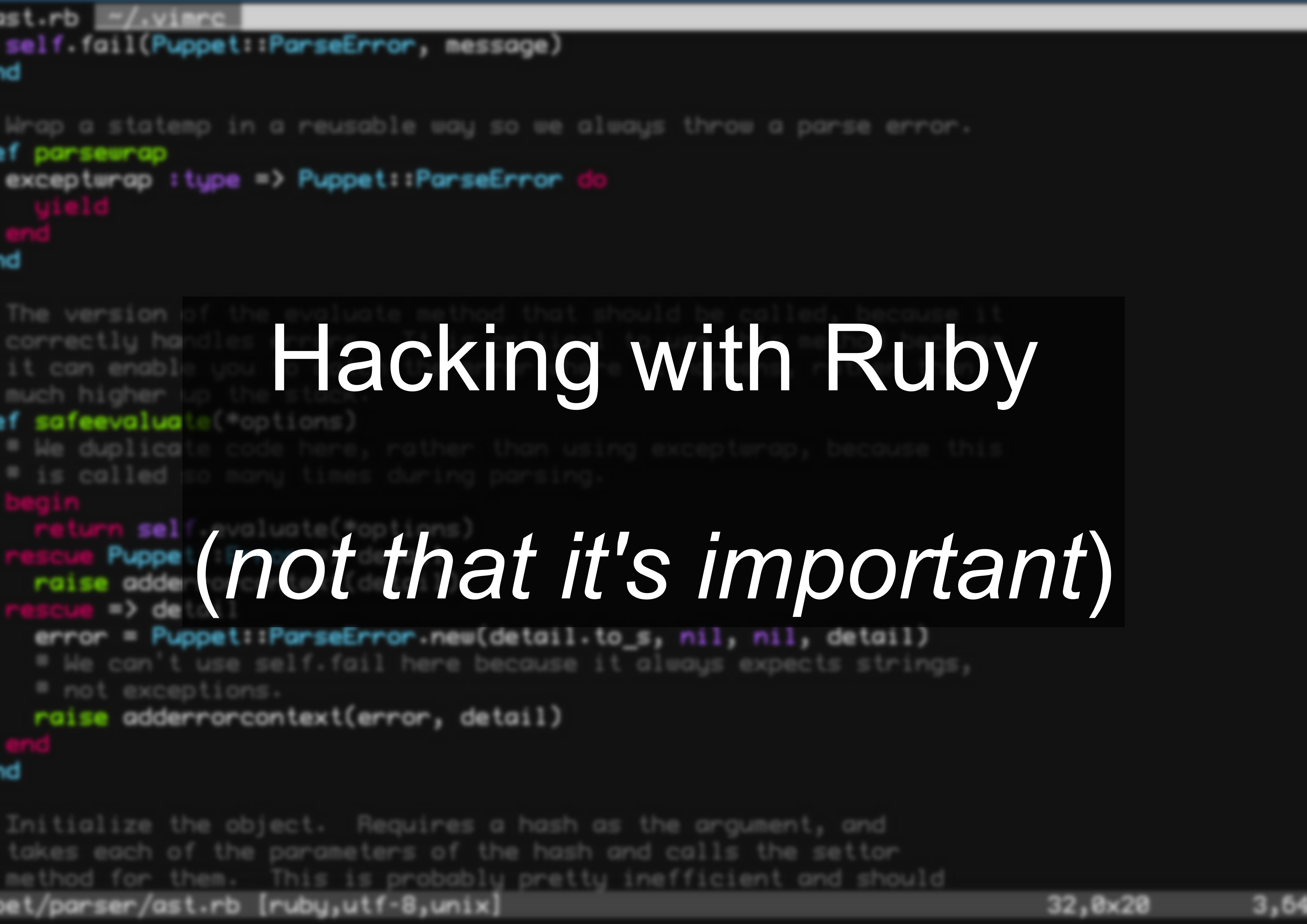
Rebuilding the airplane at 10 000m
Continuous Deployment with Jenkins and Gerrit

I work for Lookout



```
ast.rb  ~/.vimrc
self.fail(Puppet::ParseError, message)
nd

Wrap a statemp in a reusable way so we always throw a parse error.
ef parsewrap
exceptwrap :type => Puppet::ParseError do
  yield
end
nd

The version
correctly ha
it can enabl
much higher
ef safeevalua
# We duplica
# is called
begin
  return sel
rescue Puppe
  raise adde
rescue => de
  error = Puppet::ParseError.new(detail.to_s, nil, nil, detail)
  # We can't use self.fail here because it always expects strings,
  # not exceptions.
  raise adderrorcontext(error, detail)
end
nd

Initialize the object.  Requires a hash as the argument, and
takes each of the parameters of the hash and calls the settor
method for them.  This is probably pretty inefficient and should
et/parser/ast.rb [ruby,utf-8,unix]                         32,0x20       3,6
```

Hacking with Ruby

(*not that it's important*)

Let's talk about: **continuous deployment**

What it is: NOT

"Release *everything* as soon as possible!"

(photo by thomen: <http://www.flickr.com/photos/thomen/364890522/>)

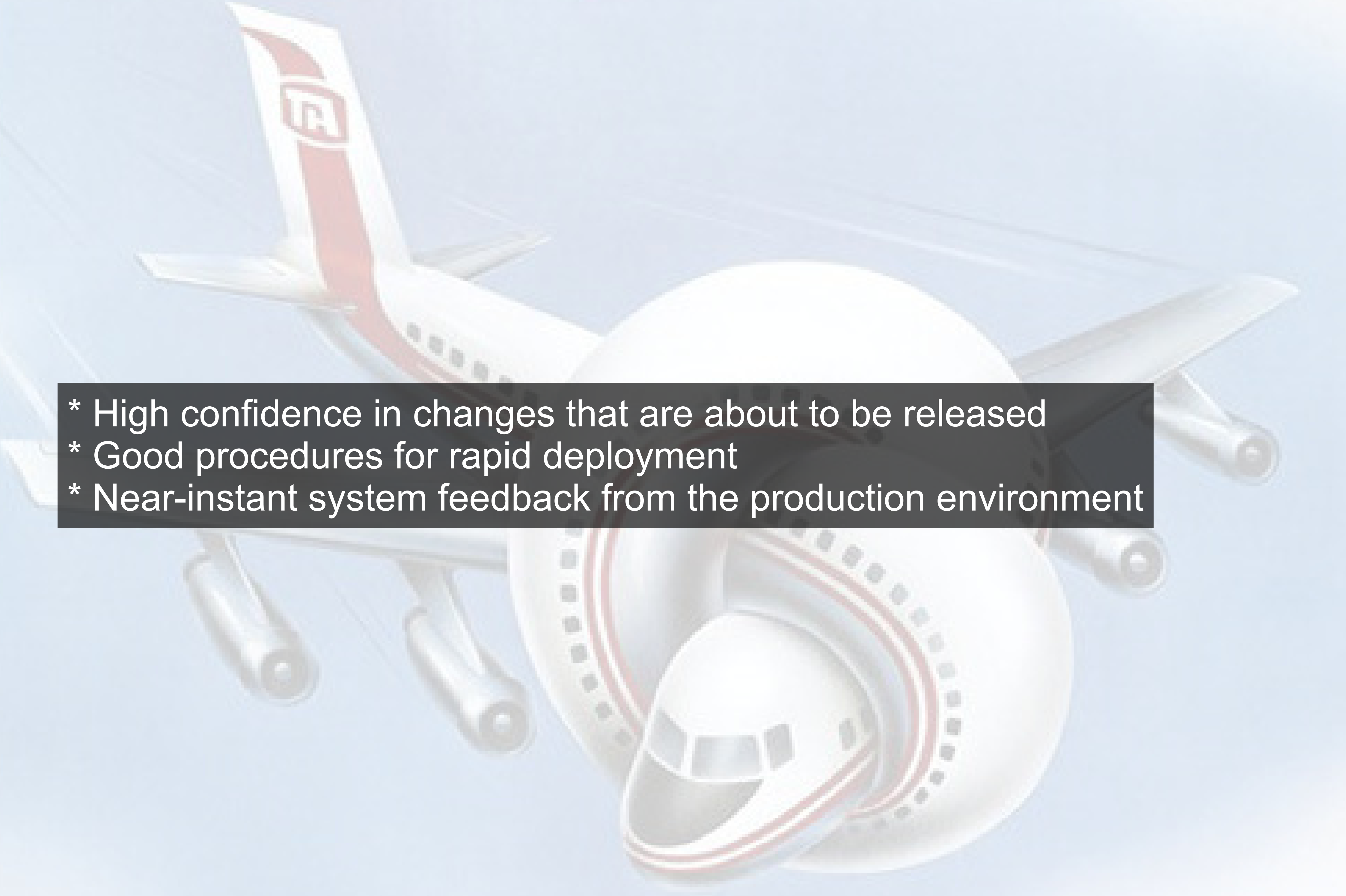
"Great! No need for a QA team"

"Our users will be our testers!"

Continuous Deployment is about

stability

Faster and More Often



* High confidence in changes that are about to be released
* Good procedures for rapid deployment
* Near-instant system feedback from the production environment

Continuous Deployment is **GOOD**

in the olden days

Subversion branches for releases

TODO: create background image

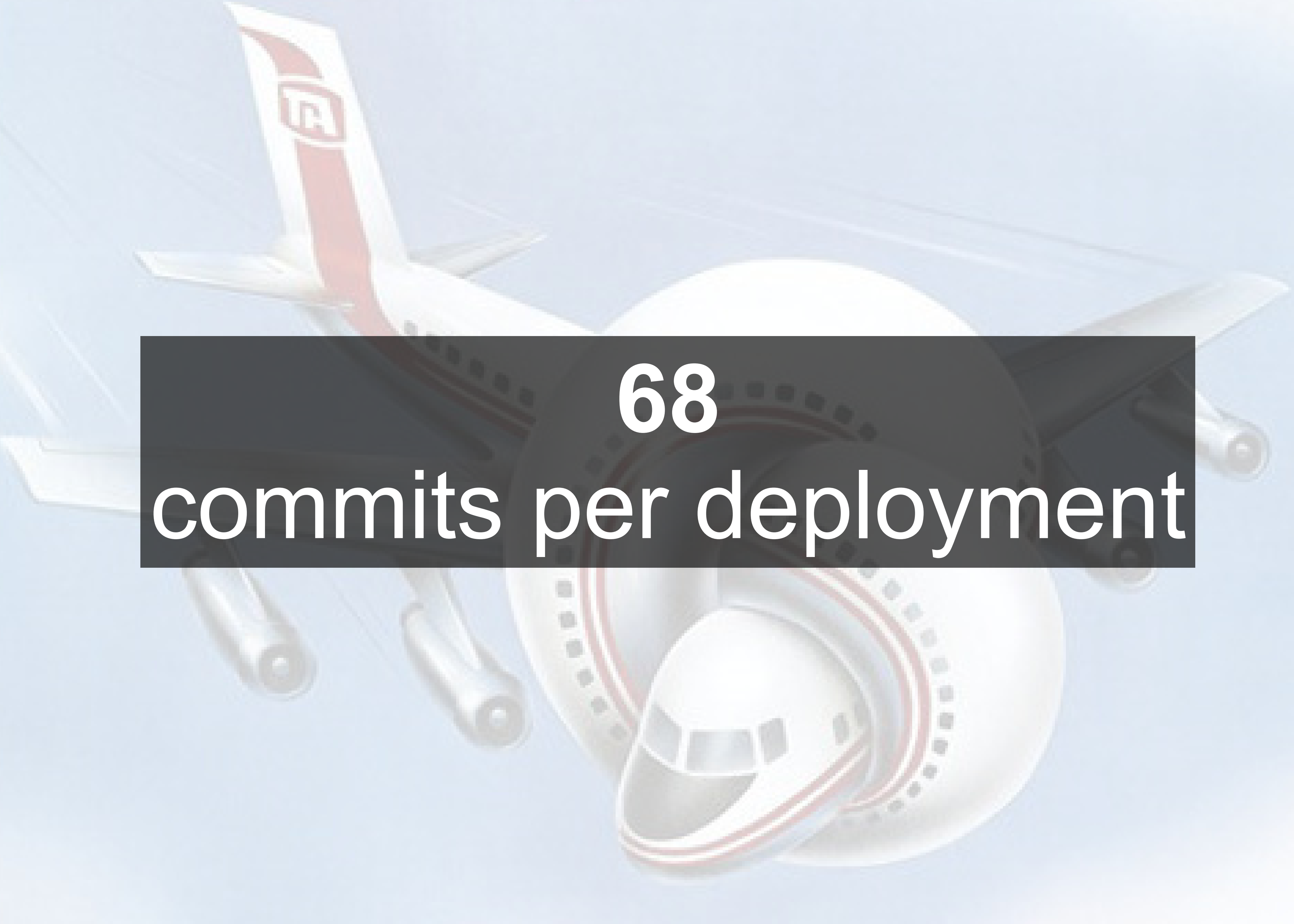
10-18 days per release branch

very little automation

Sadness with Numbers

36%

of deployments failed



68

commits per deployment

62%

of deployments missed their target date

(*that sucks*)

Let's **fix** this.

Step One: **Automate**

s/Bitten/Jenkins/

I won't tell you too much about Bitten, but it's not a great tool and we had a number of issues with it:

* Practically zero developer insight into the test/build process
* All the tests ran on *one* build machine which was hand-crafted by the Operations team for the task
* We would constantly have issues with Bitten losing track of test processes

We installed Jenkins and started to work on migration "jobs" over to Jenkins.

Since our Bitten installation was so backwards, we ended up building a number of jobs "from scratch."

"Why don't our tests pass?"

The first major issue we had was that we noticed that we had tests that didn't actually *pass* reliably. Previously this was hidden from us, but after running the tests after every commit with Jenkins, we noticed that we had some technical debt in the test suite

"We need build machines!"

Once we started running more tests more often with Jenkins we found out very quickly that we needed to start to create a build slave infrastructure.

For the first few months we relied on a hand-crafted VMWare snapshot, which we used to start up a pool of machines that all looked the same

Step Two
Die Subversion, Die

(I don't like SVN)

Git + Gerrit

I'm going to assume you know about Git or at least some kind of distributed version control system, so let me tell you about Gerrit.

Gerrit

Gerrit is a Git-based code review tool
TODO: Fill this in more

Git + Gerrit
at the same time!

We switched from Subversion directly to Git and Gerrit, all at once.

Instead of introducing Git as a separate tool to developers, we introduced
at the same time so developers never learned a Git-based workflow that
didn't involve Gerrit at its core.

"Pre-tested" Commits

An integral part of our Git + Gerrit workflow involved pre-testing commits.

The whole concept behind "pre-testing" a commit is that only changes which have passed the "tests" will be allowed to be integrated or merged.

Featuring the:

Gerrit Trigger plugin

Git + Gerrit
Training the Team

We scheduled 3 different 1 hour training sessions with various groups of engineers in order to provide a hands-on walk-through of the Git + Gerrit workflow

This included a fully set-up "demo" project to use for experimentation of creating commits, code reviewing them, verifying them with Jenkins and finally merging them into the "master" bracnh

Document *everything*

During the course of these training sessions, we used the feedback and common problems encountered by engineers to fill out a "getting started" wiki page which new hires now use to come up to speed with Git + Gerrit.

Recap

TODO: Diagram and cover "the world thus far"

Step Three
Automate *Everything*

More slaves!

Managing slaves?

New kinds of tests!

Automating **deployment**

First automating with `infra_update_faithful2` then moving into the `infra_deploy_qa` territory. Finally automating the actual deployment of production

Deploying the test environment

Once the deployment of our test environment was managed through Jenkins,
we created pipelines with Jenkins, chaining off of a successful
deployment to the test environment.

TODO: Discuss selenium testing/SI testing after QA deploy

Deploying the *production* environment

TODO: Discuss the use of build promotions to stage, deploy and finally mark the deployment as successful

TODO

(*nobody's perfect*)

* Haven't automated the rollback of production
* Not nearly enough stages/production clusters to run Selenium tests per-commit
* Not enough real-time feedback from production
* No production acceptance testing
* Tests take a LONG time
* Still only doing a single daily deployment, facing the same issues we did before with 2 week deployment, but on a smaller scale