



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



I work for Lookout




```
ast.rb ~/.viirc
self.fail(Puppet::ParseError, message)
end
```

Wrap a stateop in a reusable way so we always throw a parse error.

```
def parsewrap
  exceptwrap :type => Puppet::ParseError do
    yield
  end
end
```

The version of the evaluate method that should be called, because it correctly handles errors. It is called to evaluate a string because it can enable you to raise an exception if the string is not valid. It is much higher up the stack.

```
def safeevaluate(*options)
  # We duplicate code here, rather than using exceptwrap, because this
  # is called so many times during parsing.
```

```
begin
  return self.evaluate(*options)
rescue Puppet::ParseError => detail
  raise adderrorcontext(detail)
rescue => detail
  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
```

Initialize the object. Requires a hash as the argument, and takes each of the parameters of the hash and calls the setter method for them. This is probably pretty inefficient and should


```
set/parser/ast.rb [ruby,utf-8,unix]
```

32,8x28

3,64

Hacking with Ruby


(not that it's important)

The background of the slide is a composite image. The upper portion shows the tail of a white airplane with a red stripe and a 'TA' logo on the vertical stabilizer, set against a clear blue sky. The lower portion shows a close-up of a large, white rocket engine with red and blue decorative stripes around its base. A dark, semi-transparent rectangular box is centered over the image, containing the text.

Let's talk about:
continuous deployment



What it is:
NOT

A photograph of a messy room. In the foreground, a wooden table is cluttered with various items: a yellow container, a blue and white striped bag, a glass jar, and some papers. In the background, a potted plant sits on a dark surface, and a wooden chair is visible. A large, semi-transparent grey box with white text is overlaid on the center of the image.

"Release *everything*
as soon as possible!"

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

A photograph of a messy room. A light-colored wooden table is tipped over on its side. On the floor next to it is a potted plant with green leaves. Various items are scattered on the floor, including a yellow container, a blue and white striped bag, a glass jar, and some papers. A dark-colored sofa is visible in the background.

"Great! No need for a QA team"

A photograph of a messy room. A light-colored wooden table is tipped over on its side. On the floor next to it is a potted plant with green leaves. There is a lot of trash and debris scattered on the floor, including crumpled paper, a yellow container, and a glass jar. A dark-colored sofa is visible in the background.

"Our users will be our testers!"




Continuous Deployment
is about



stability



**Faster
and
More Often**

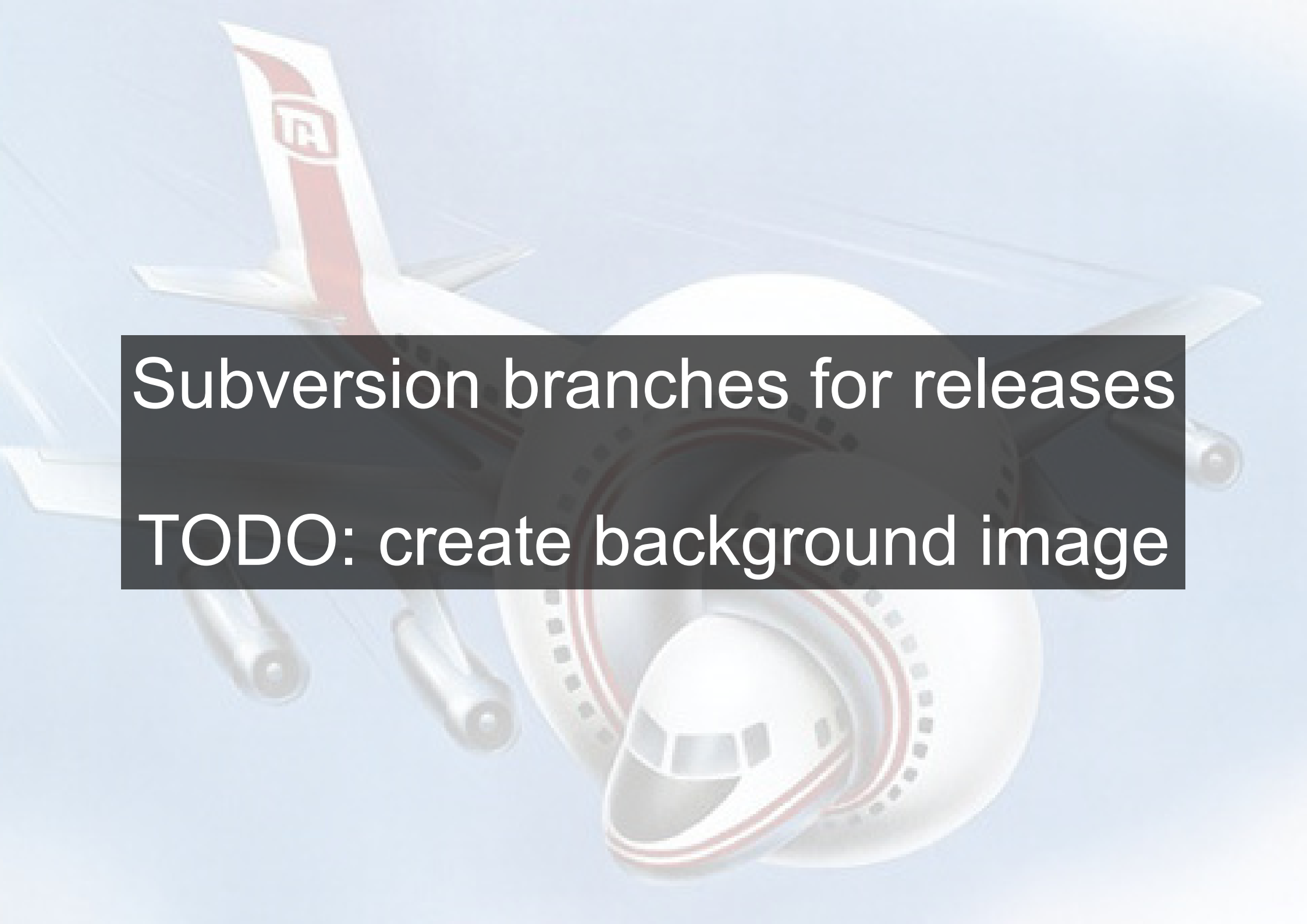
- 
- * High confidence in changes that are about to be released
 - * Good procedures for rapid deployment
 - * Near-instant system feedback from the production environment



Continuous Deployment is
GOOD

A composite image featuring a vintage airplane and a globe. The airplane, which has a white body with red and blue stripes and a 'TA' logo on the tail, is shown in flight against a light blue sky with motion blur. A large, semi-transparent globe is positioned in the foreground, partially obscuring the airplane. The globe has a similar red and blue stripe design. Overlaid on the center of the image is a black rectangular box containing the text 'in the olden days' in a white, italicized serif font.

in the olden days



Subversion branches for releases
TODO: create background image

10-18 days per release branch






very little automation

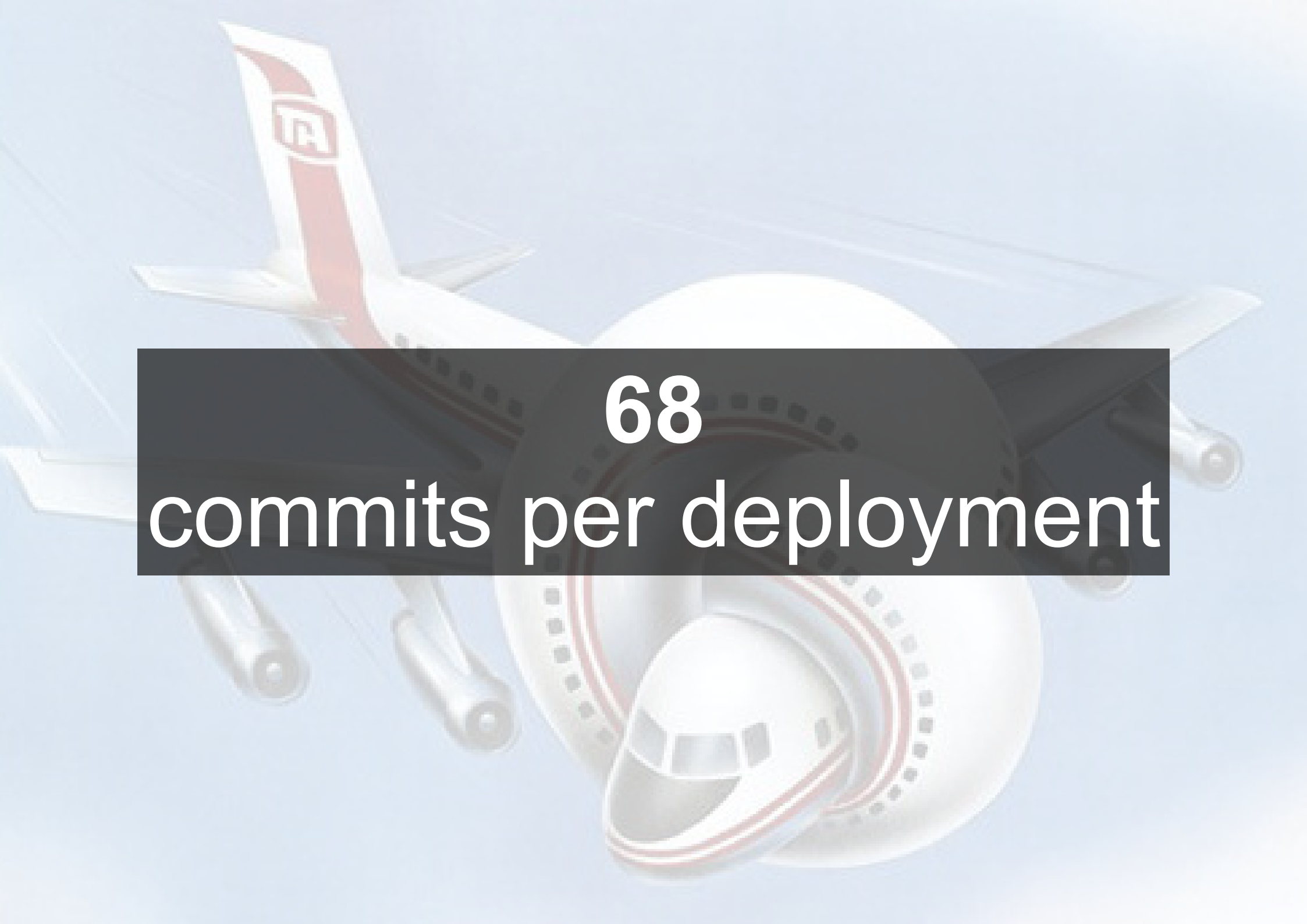




Sadness with Numbers




36%
of deployments failed



68

commits per deployment



62%
of deployments missed
their target date



(that sucks)

A large white passenger jet is shown flying upside down against a light blue sky. The aircraft has a red stripe running along the fuselage and a red tail fin with a white 'TA' logo. The text 'Let's fix this.' is displayed in white on a black rectangular background, centered over the aircraft's fuselage.

Let's fix this.

A photograph of an industrial manufacturing environment, likely a car factory. Several yellow robotic arms are visible, working on car chassis. The scene is filled with metal structures, pipes, and safety railings. The lighting is bright, typical of a large industrial hall. The text 'Step One: Automate' is overlaid in the center in a large, white, sans-serif font.

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

A large passenger jet, likely a Boeing 747, is shown from a low angle, flying over a city. The aircraft is white with red and blue stripes. The tail fin features a logo with the letters 'TA' inside a red square. A smaller aircraft is visible in the background, flying alongside the larger plane. The sky is a clear, light blue.

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

A large commercial airplane, likely a Boeing 747, is shown from a low angle, flying over a city. The plane is white with red and blue stripes along the fuselage and tail. The tail features a logo with the letters 'TA' inside a red square. The background is a light blue sky with some clouds. A dark blue banner is overlaid in the center of the image, containing the text 'Git + Gerrit Training the Team' in white, bold, sans-serif font.

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" branch



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"

A large commercial airplane, likely a Boeing 747, is shown in flight against a light blue sky. The plane is white with red and blue stripes along the fuselage and tail. The tail features a logo with the letters 'TA' inside a red square. The plane is flying over a large, stylized globe that is also white with red and blue stripes. The globe is positioned in the center of the image, and the plane is flying from the top left towards the bottom right. The overall image has a soft, ethereal quality with a light blue background.

Step Three

Automate *Everything*

A large white passenger jet with red and blue stripes is flying over a smaller white propeller plane. The passenger jet has a red stripe running along the side of the fuselage and a red and blue stripe on the tail. The propeller plane is white with a red stripe on the nose. The background is a light blue sky with white clouds.

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/Sl testing after QA deploy



Deploying the *production* environment

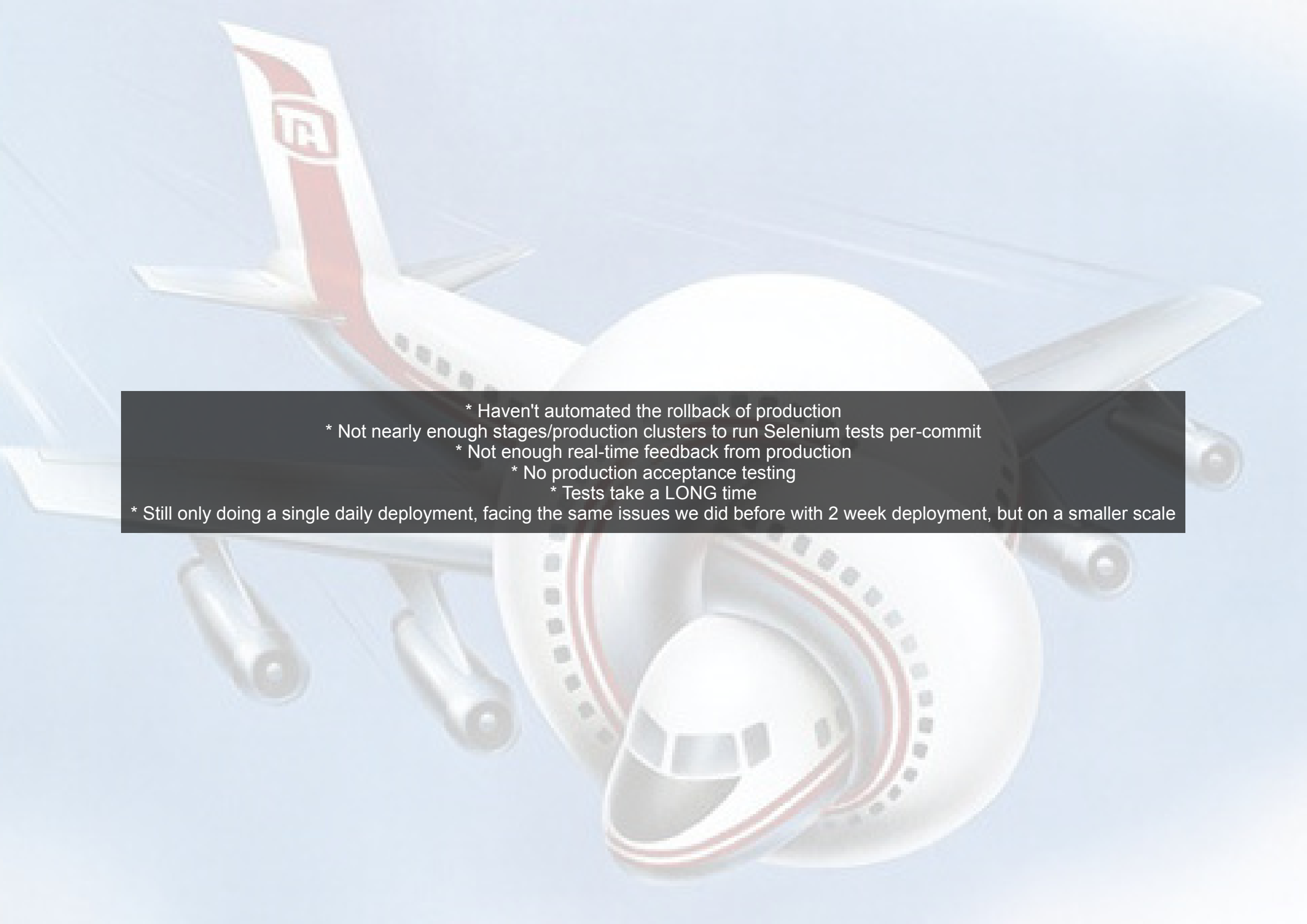
TODO: Discuss the use of build promotions to stage, deploy and finally mark the deployment as successful



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