

Continuous Delivery

Moritz Lenz <moritz@fau2k3.org>

<https://perlgeek.de/>

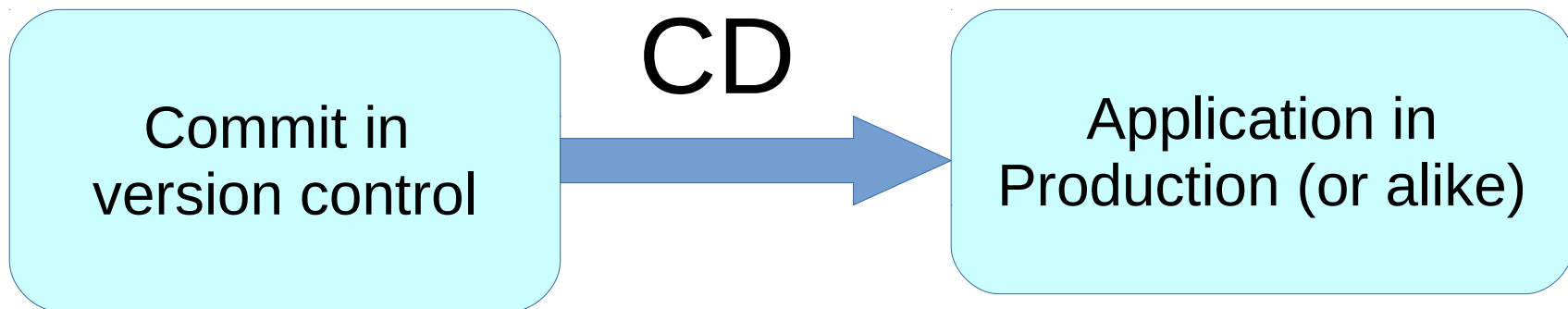
<https://deploybook.com/>

Contents

- What is Continuous Delivery
- Why?
- Requirements
- Anatomy of a Delivery system
- Example project
- Experience

What is Continuous Delivery?

Deploying every commit that is good enough to production or a production-like environment



What is Continuous Delivery?

- Building
- Determine which commits are “good enough” (CI, Various testing stages)
- Deployment to environments
- Actual deployment to production can still be a business decision

Why?

Why CD? – Save time

How much time do you spend

- Preparing releases?
- Doing the deployments?
- Fix up problems afterwards?
- Spend time in meetings on how to avoid the problems caused by deployments?

Boring, repetitive, error-prone tasks should be automated!

Why CD? – Allow Scaling

Manual deployments don't scale to:

- Several environments
- Many machines
- Many applications

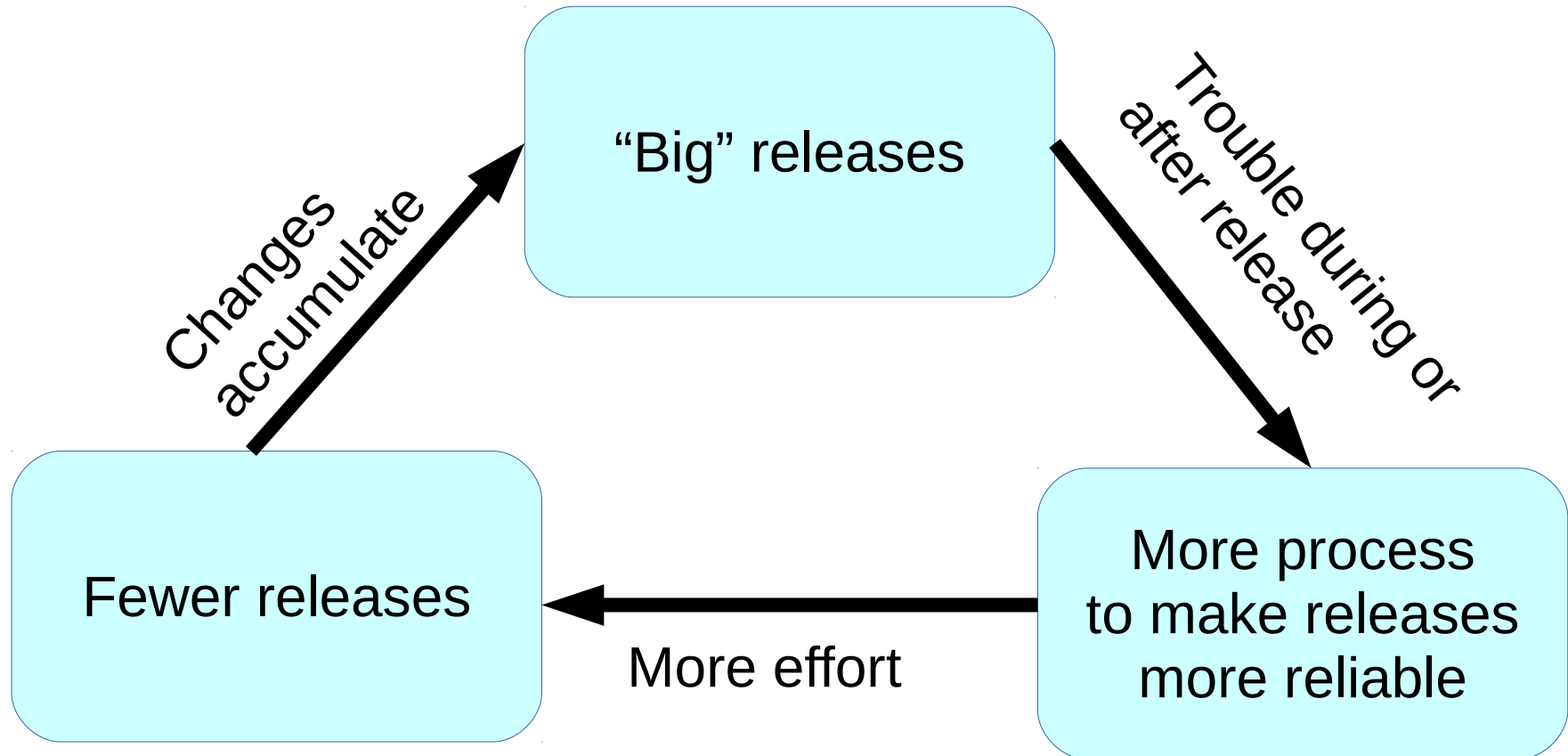
Why CD? – Time to Market

- Release cycles put a lower bound on the time from idea (or bug report) to value delivered
- Applies to **everything**, even one-line CSS changes
- (unless you bypass the normal deployment process for “small” fixes, which gives you other problems)

Why CD? – Shorter feedback cycle

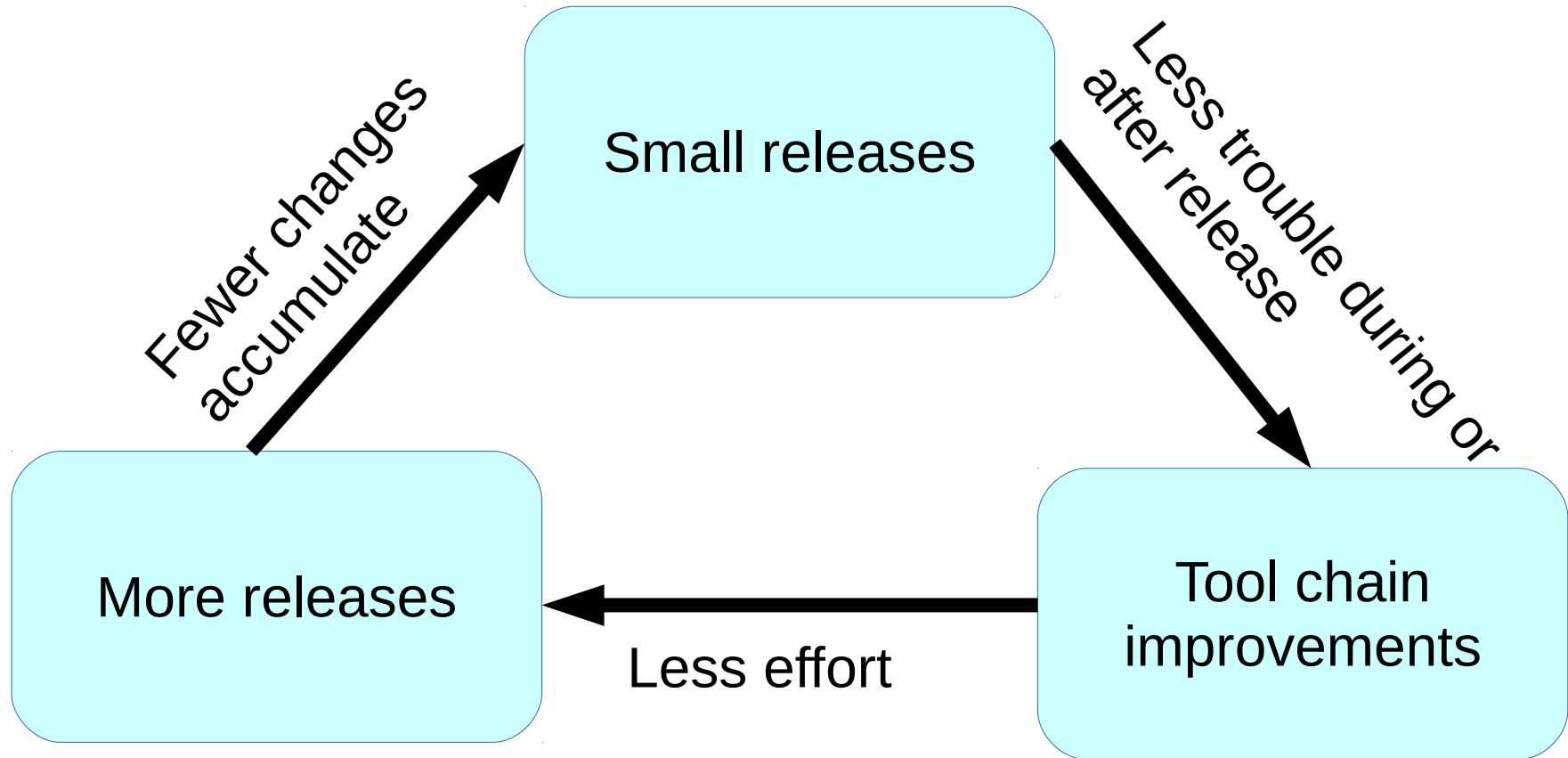
- Specifications are never perfect. Only multiple iterations of feedback and improvements produce good products
- The shorter the feedback cycle, the more feedback you can get
- If the feedback cycle is too long, users won't bother to give feedback

Why CD? – Reliability



The Vicious Cycle of big, manual releases

Why CD? – Reliability



The Virtuous Cycle of small, automated releases

Why CD? – Architectural Freedom

The more work a deployment is, the more pressure you will feel to put a new feature into an existing component.

Microservices: dozens or hundreds of separately deployed components.

Why CD? – Fancy QA

You could:

- Deploy two version of a service
- Route requests to both versions, compare results
- Record differences and timings

... but not practical if you deploy it all manually

Every good commit to production?

- Regulations might prevent that
- Schedules in contracts might prevent that
- The need for manual review might prevent that

You can still do Continuous Delivery by deploying to a staging environment.

Can I do
Continuous
Delivery?

Requirements: Version Control

- Your code must be under version control
- Your build tools must be under version control
- Your requirements and environment should be under version control (but you can start without it).

Requirements: Infrastructure

You must have or create the infrastructure to

- Build your software
- Test it
- Deploy it

If it's not there yet, make sure to talk to the operators!

In case of lock-down: deploy to a non-locked-down environment at least.

Requirements: Tests

You need automated tests. Enough to make you feel confident that a new version doesn't break everything/too much/anything.

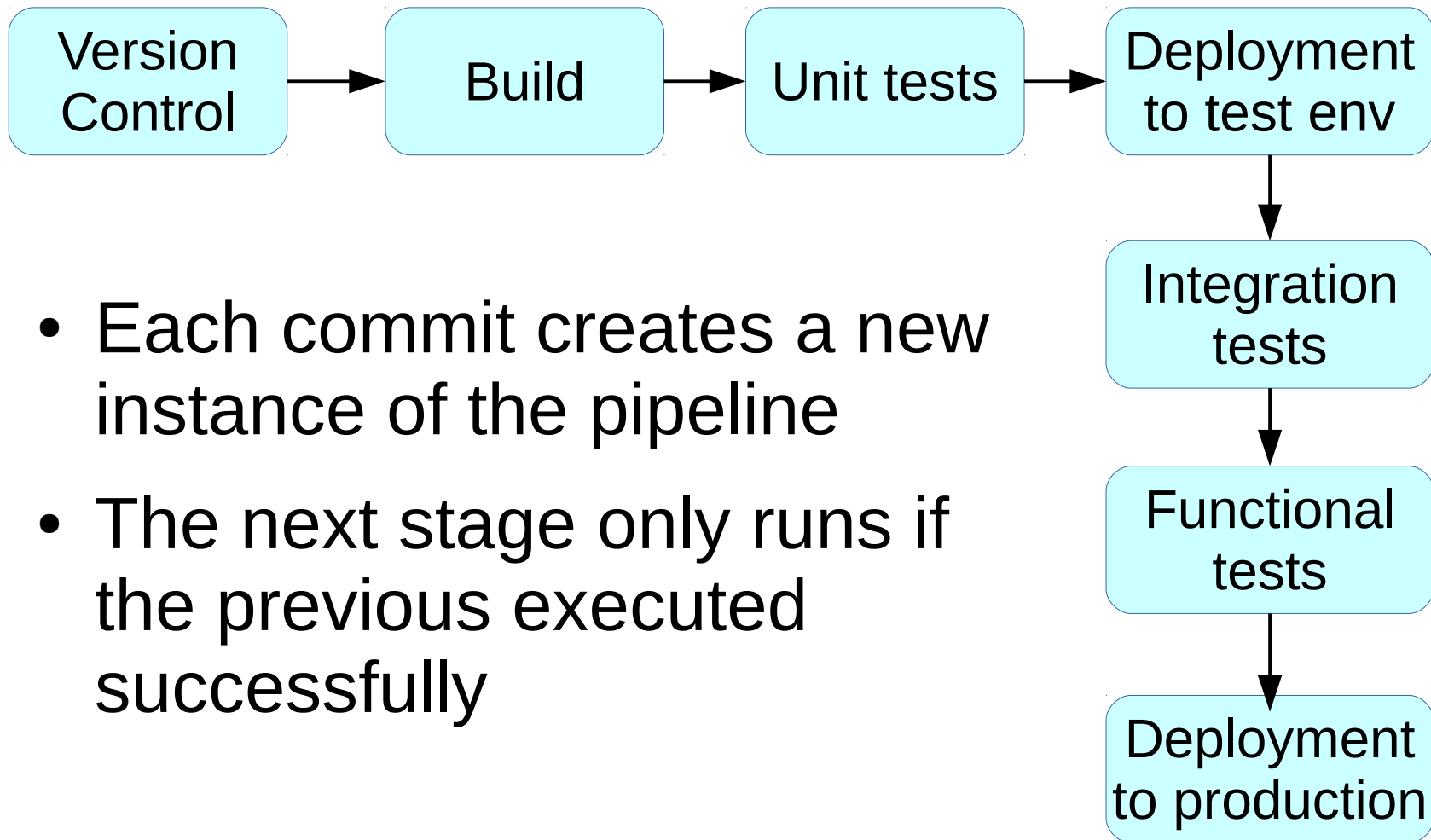
Integration tests and functional tests are very valuable here.

Soft requirements: Server software

- It is easier to deploy a server-side application (web app, service).
- If users install your software on their own machines, you can only deploy to a repository and have the software auto-update.

Anatomy of a Deployment System

Architecture: Pipeline



- Each commit creates a new instance of the pipeline
- The next stage only runs if the previous executed successfully

Technology: Packaging Format

- Use something that resolves dependencies for you (not just plain tar balls/rsync). Or fat-package them.
- If you have operators: something that your operators understand

Good choices: System packages (.deb, .rpm, .msi), Containers

Technology: Repository

- Dependency resolution requires a pull model
- Repository tooling depends on the actual package format
- Aptly (.deb), Pulp (.rpm, .deb, python, ...), CPAN::Mini + CPAN::Mini::Inject, ...

Technology: Installation

- “apt-get install” “zypper install” or “cpanm” or ...
- Or a tool that automates those installers:
ansible, rex, salt

Technology: Integration tests

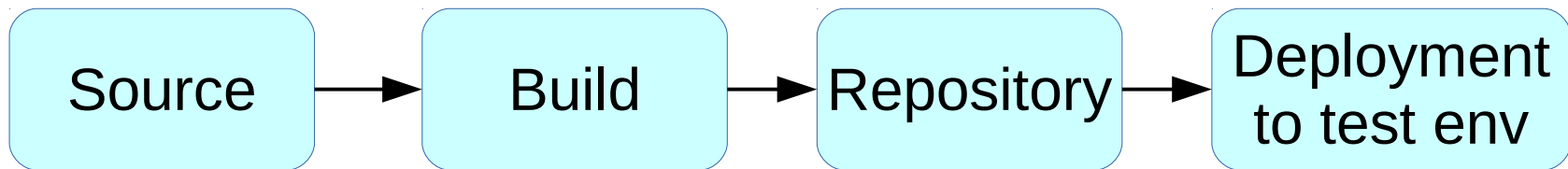
- WWW::Mechanize::Test
- wget
- curl
- expect
- Various monitoring plugins
- ...

Technology: Pipeline Organization

- Go Continuous Delivery
- Jenkins with plugins
- Atlassian Bamboo

An Example Project

<https://github.com/moritz/package-info/>



Example Project: Code

```
#!/usr/bin/perl
```

```
use Mojolicious::Lite;
```

```
plugin 'Config';
```

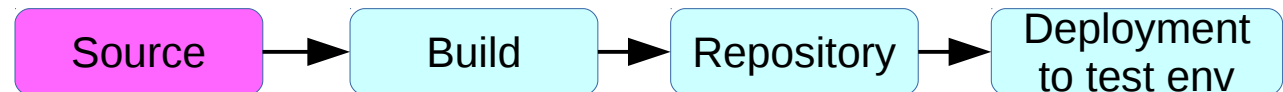
```
get '/' => sub {
```

```
    my $c = shift;
```

```
    $c->render(text => scalar qx/dpkg -l/, format => 'text');
```

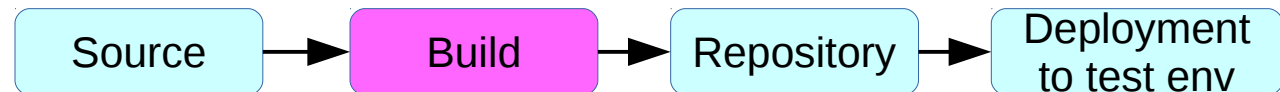
```
};
```

```
app->start;
```



Debian packaging

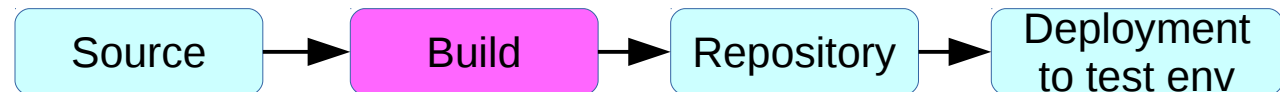
- `dh_make --createorig -p package-info_0.1`
- `debian/install`: Files to be copied to package
- `debian/control`: Meta data (name, deps)
- `debian/postinst`: Create user here
- `debian/$package.service`: system service file



Building

debuild -b -us -uc

- Problem: version (from debian/changelog) recycled



Building without version reuse

```
#!/bin/bash
```

```
set -e
```

```
set -o pipefail
```

```
version=$(git describe --long | sed 's/-g[a-f\d]*$//')
```

```
# Remove commit hash with leading "g"
```

```
version=$(sed 's/-g[A-Fa-f0-9]*$//' <<< "$version")
```

```
version="$version.${GO_PIPELINE_COUNTER:-0}.${GO_STAGE_COUNTER:-0}"
```

```
DISTRIBUTION=${DISTRIBUTION:-jessie}
```

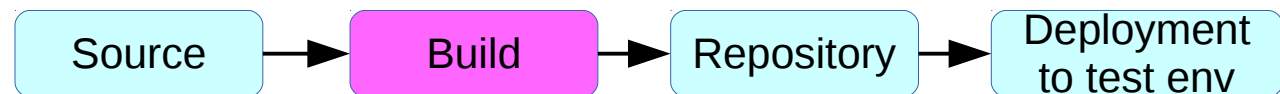
```
echo $version
```

```
echo $version > ../version
```

```
DEBFULLNAME='Go Debian Build Agent' DEBEMAIL='go-noreply@example.com' debchange
```

```
--newversion=$version --force-distribution -b --distribution="${DISTRIBUTION:-jessie}" 'New Version'
```

```
debuild -b -us -uc
```



Repo Management: aptly

- Initial setup:

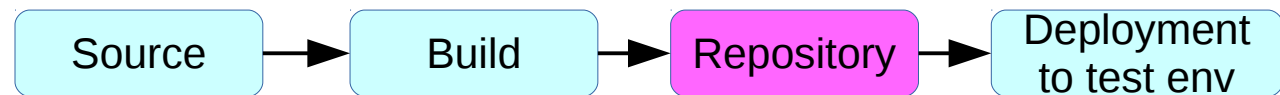
```
aptly repo create -distribution=jessie  
-architectures=amd64,all,i386 staging
```

```
aptly publish repo staging
```

- For each package:

```
aptly repo add staging $debfile
```

```
aptly publish update jessie
```



Package Installation: ansible

- Inventory (“hosts”) file:

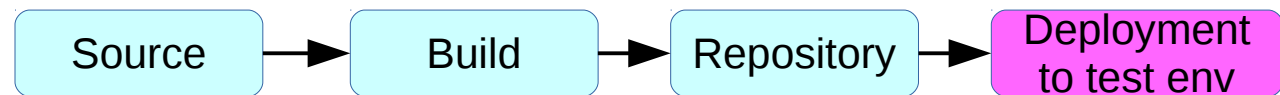
```
[web]
```

```
www01.staging.example.com
```

```
www02.staging.example.com
```

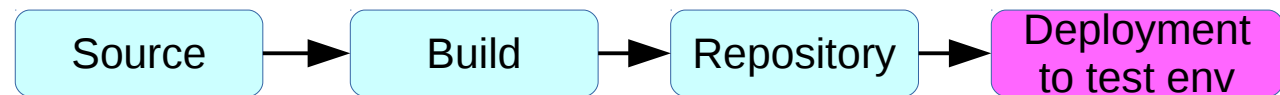
```
[all:vars]
```

```
ansible_ssh_user=root
```



Package Installation: ansible

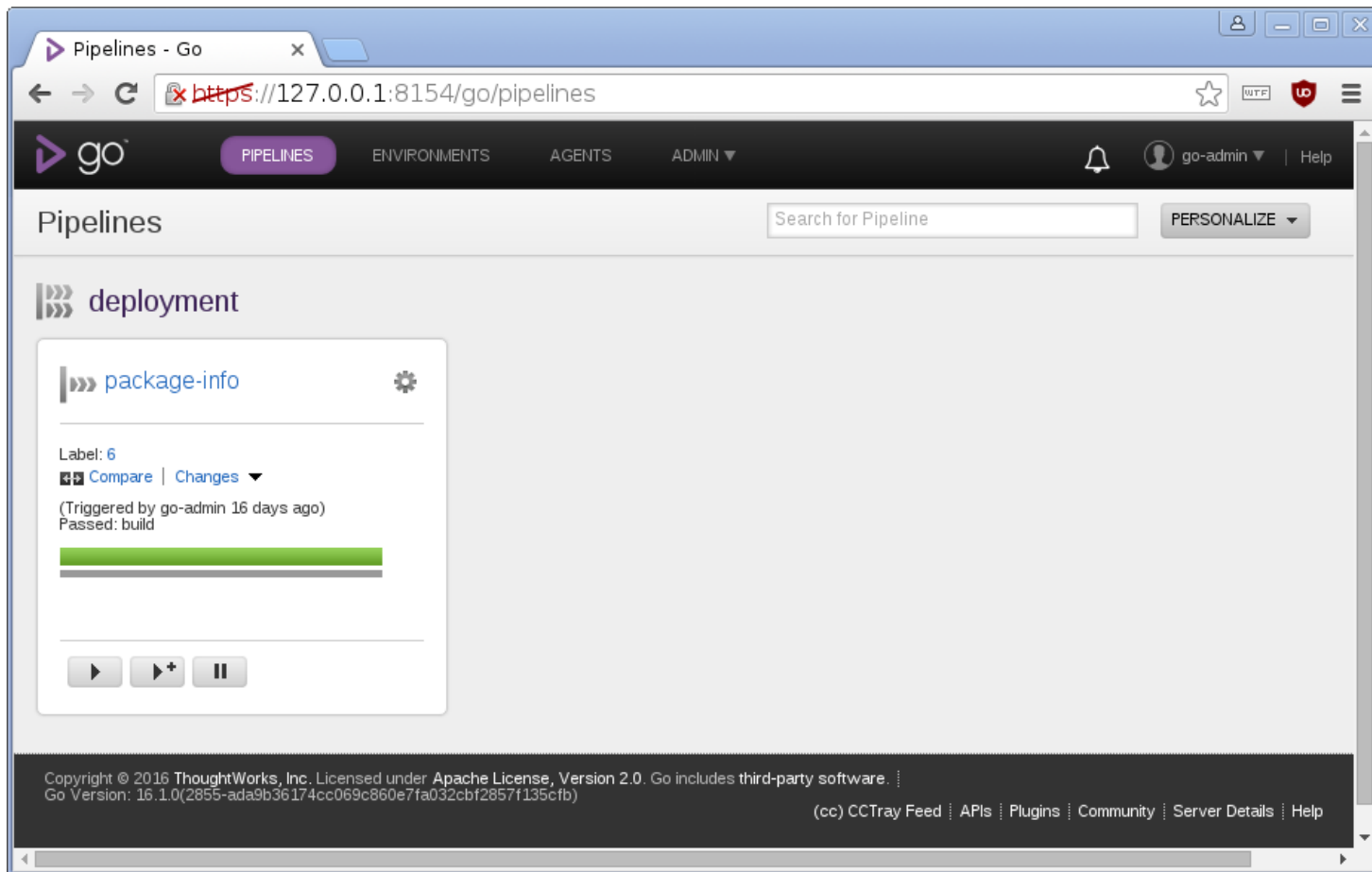
```
ansible -i staging web \
  -m apt \
  -a 'name=package-info update_cache=yes
state=latest'
```



Coordination: GoCD

```
<materials>
  <git url="https://github.com/moritz/package-info.git" dest="package-info" />
  <git url="https://github.com/moritz/deployment-utils.git" dest="deployment-utils" materialName="deployment-utils" />
</materials>
<stage name="build" cleanWorkingDir="true">
  <jobs>
    <job name="build-deb" timeout="5">
      <tasks>
        <exec command="/bin/bash" workingdir="package-info">
          <arg>../deployment-utils/debian-autobuild</arg>
        </exec>
      </tasks>
      <artifacts>
        <artifact src="version" />
        <artifact src="package-info*_*" dest="package-info/" />
      </artifacts>
    </job>
  </jobs>
</stage>
```

GoCD: Pipeline overview



The screenshot displays the GoCD web interface in a browser window. The address bar shows the URL `https://127.0.0.1:8154/go/pipelines`. The navigation bar includes the GoCD logo, a search bar, and menu items for PIPELINES, ENVIRONMENTS, AGENTS, and ADMIN. The user is logged in as 'go-admin'. The main content area is titled 'Pipelines' and shows a list of pipelines. The selected pipeline is 'deployment', which contains a job named 'package-info'. The job details show it was triggered by 'go-admin' 16 days ago and passed the 'build' step. A progress bar indicates the job's status, and control buttons for play, play+, and pause are visible at the bottom of the job card. The footer contains copyright information for ThoughtWorks, Inc. and links to various resources like the CCTray Feed, APIs, Plugins, Community, Server Details, and Help.

GoCD: More Stages

The screenshot displays the GoCD web interface in a browser window. The address bar shows the URL `https://127.0.0.1:8154/go/pipelines`. The navigation bar includes the GoCD logo, a 'PIPELINES' button, and links for 'ENVIRONMENTS', 'AGENTS', and 'ADMIN'. The user is logged in as 'go-admin'. The main content area is titled 'Pipelines' and features a search bar and a 'PERSONALIZE' dropdown. A pipeline named 'deployment' is selected, showing a stage 'package-info' with a gear icon for configuration. The stage details include a 'Label: 7', a 'Compare | Changes' dropdown, and a status message: '(Triggered by go-admin 2 minutes ago) Failed: deploy-testing'. A progress bar below the message shows four segments, with the first two green and the last two red. At the bottom of the stage card are three control buttons: a play button, a play button with a plus sign, and a pause button. The footer contains copyright information for ThoughtWorks, Inc. (2016), the Apache License version (2.0), and the Go version (16.1.0). It also includes links for '(cc) CTray Feed', 'APIs', 'Plugins', 'Community', 'Server Details', and 'Help'.

Go: History View, Manual Approval

The screenshot shows a web browser window with two tabs: 'Pipelines - Go' and 'Pipeline Activity - Go'. The address bar shows the URL `https://127.0.0.1:8154/go/tab/pipeline/history/package-info`. The Go CD interface has a navigation bar with 'PIPELINES', 'ENVIRONMENTS', 'AGENTS', and 'ADMIN'. The user is logged in as 'go-admin'. The main content area is titled 'Package-Info' and has a 'PAUSE' button. Below this is a table of pipeline runs with columns for 'build', 'upload-testing', 'deploy-testing', 'upload-prod', and 'deploy-prod'. Each row represents a revision, with progress bars indicating the status of each stage. Revision 7 shows a failure in the 'deploy-testing' stage, indicated by a red bar.

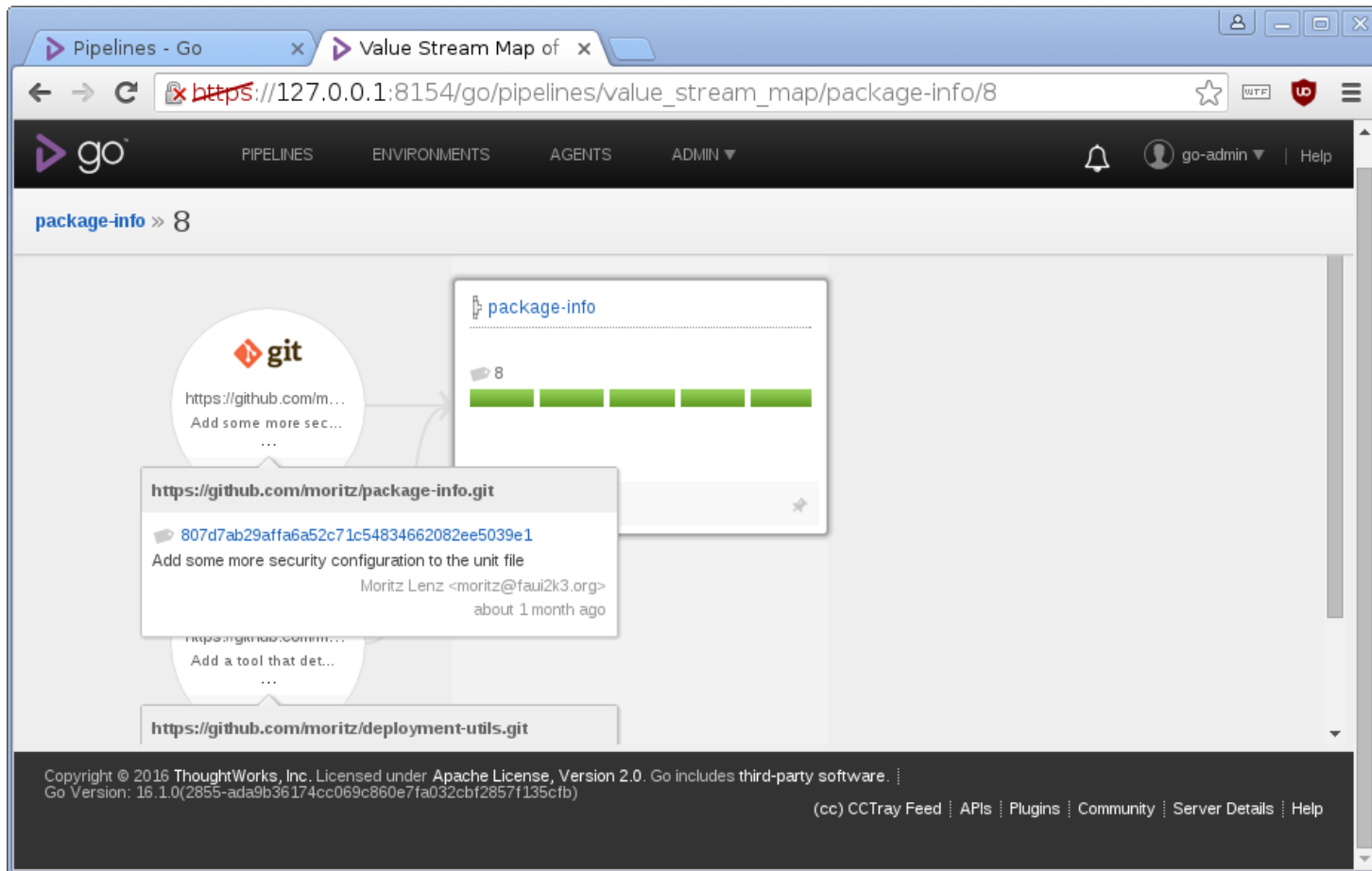
	build	upload-testing	deploy-testing	upload-prod	deploy-prod
8 revision: 807d7ab2... about 1 month ago Triggered by go-admin	Completed	Completed	Completed	Pending	Pending
7 revision: 807d7ab2... about 1 month ago Triggered by go-admin	Completed	Completed	Failed	Pending	Pending
6 revision: 807d7ab2... about 1 month ago Triggered by go-admin	Completed	Pending	Pending	Pending	Pending
5 revision: 807d7ab2... about 1 month ago Triggered by go-admin	Completed	Pending	Pending	Pending	Pending
4	Completed	Pending	Pending	Pending	Pending

GoCD: After manual approval

The screenshot shows the GoCD web interface. The browser address bar displays `https://127.0.0.1:8154/go/tab/pipeline/history/package-info`. The page title is "Package-Info" with a "PAUSE" button. The pipeline stages are: build, upload-testing, deploy-testing, upload-prod, and deploy-prod. The pipeline activity is shown for revisions 4 through 8. Revision 8 is the most recent and is highlighted with a yellow bar in the deploy-prod stage, indicating manual approval. The other stages for revision 8 are green, indicating success. Revision 7 has a red bar in the deploy-testing stage, indicating failure. Revisions 6 and 5 are greyed out, indicating they are not the current revision.

Revision	build	upload-testing	deploy-testing	upload-prod	deploy-prod
8	Success	Success	Success	Success	Manual Approval
7	Success	Success	Failure	Success	Success
6	Success	Success	Success	Success	Success
5	Success	Success	Success	Success	Success
4	Success	Success	Success	Success	Success

GoCD: Value Stream Map



The screenshot shows a web browser window displaying the GoCD interface. The address bar shows the URL `https://127.0.0.1:8154/go/pipelines/value_stream_map/package-info/8`. The GoCD logo and navigation menu (PIPELINES, ENVIRONMENTS, AGENTS, ADMIN) are visible at the top. The main content area displays a Value Stream Map for the pipeline `package-info` at stage `8`. The map shows a sequence of steps represented by green bars. A callout box highlights a commit from `https://github.com/moritz/package-info.git` with the commit hash `807d7ab29affa6a52c71c54834662082ee5039e1`. The commit message is "Add some more security configuration to the unit file" by Moritz Lenz, dated "about 1 month ago". Other callouts in the background mention "Add some more security configuration" and "Add a tool that det...". The footer contains copyright information for ThoughtWorks, Inc. and navigation links for CC Tray Feed, APIs, Plugins, Community, Server Details, and Help.

Copyright © 2016 ThoughtWorks, Inc. Licensed under Apache License, Version 2.0. Go includes third-party software. Go Version: 16.1.0(2855-ada9b36174cc069c860e7fa032cbf2857f135cfb)

(cc) CC Tray Feed | APIs | Plugins | Community | Server Details | Help

GoCD: Stage Details

The screenshot shows the GoCD web interface in a browser window. The address bar displays the URL: `https://127.0.0.1:8154/go/tab/build/detail/package-info/8/build/1/build-deb`. The navigation bar includes the GoCD logo and menu items: PIPELINES, ENVIRONMENTS, AGENTS, and ADMIN. The user is logged in as 'go-admin'. The breadcrumb trail is: `package-info >> 8 >> build / 1 >> Build-Deb`.

The main content area shows the job status: `package-info/8/build/1/build-deb job | passed`. Below this, a table provides job details:

SCHEDULED ON:	2016-03-05T13:07:18+01:00	COMPLETED ON:	2016-03-05T13:08:03+01:00 more...
DURATION:	00:00:24	AGENT:	lara (ip:192.168.2.43)
BUILD CAUSE:	Forced by go-admin		

Below the details are tabs for Console, Tests, Failures, Artifacts, Materials, and Properties. The Console tab is active, showing a log of the job execution:

```
13:07:34.556 [go] Job started: 2016-03-05 13:07:34 CET
13:07:34.556 [go] Start to prepare package-info/8/build/1/build-deb on lara [/var/lib/go-agent]
13:07:34.609 [go] Cleaning working directory "/var/lib/go-agent/pipelines/package-info" since stage is configured to clean working directory
13:07:34.610 [go] Start to update materials.
13:07:34.610 [go] Start updating package-info at revision 807d7ab29affa6a52c71c54834662082ee5039e1 from https://github.com/moritz/package-info.git
13:07:34.615 STDERR: Cloning into '/var/lib/go-agent/pipelines/package-info/package-info'...
13:07:35.511 [GIT] Fetch and reset in working directory pipelines/packag
```

On the right side, there is a 'JOB HISTORY' section listing previous jobs:

- package-info/8/build/1/build-deb (7 minutes ago)
- package-info/7/build/1/build-deb (10 minutes ago)
- package-info/6/build/1/build-deb (16 days ago)
- package-info/5/build/1/build-deb (17 days ago)
- package-info/4/build/1/build-deb (17 days ago)

Experience

CD Dreams seem unreachable

The ideals of Continuous Delivery:

- Fully automated deployments
- Fully automated rollbacks
- New environments on the press of a button

Seem to be pretty steep goals

Small steps towards CD

- Starting with automating the steps that cause most work is a good approach
- Partial automation much better than fully manual builds and deployment
- Rollbacks are really just deployments of a defined, older versions

Time to Market Matters for Bugfixes

- For bugs that affect lots of users, fixing it quickly in production is crucial
- Having a fast (minutes, not hours), automated pipeline is tremendously helpful

Managing State is HARD

- After about a year of part-time CD efforts, not yet automated at \$work
- How to handle DB schema changes? Either the change or the rollback can lose data
- Needs cooperation from the development process (introduce columns first, make them NOT NULL later)

“Big” tools have a steep learning curve

- GoCD: Quite some time getting used to
- Initial frustration
- Many secondary benefits: version tracking, notifications, permissions, operations-friendly UI

The Advantages are real

- Time savings
- Faster feedback
- Less stress
- Scaling to more components (currently 24)
- All the successful, big companies (Netflix, Google, Facebook, Spotify, LinkedIn) do it.

A worthwhile
experience.

You should try it.

Thank You

Interested? Check out
<https://deploybook.com/>