

MUNKI MISTAKES MADE RIGHT

TOM BRIDGE
TECHNOLUTIONARY LLC

   @tbridge

- Hi everyone, I'm Tom Bridge, I'm a partner at Technolutionary LLC in Washington DC. We're an Apple Consultants Network member, and we serve small to medium businesses. Our target market is 10 to 200.
- We're five or six years into Munki at this point. It's a mature platform that is managing probably millions of Macs across the enterprise. It's a very stable piece of work, for which we have to thank all of the contributors to the project. If you've ever committed code to the Munki Project, please raise your hand.
- Munki manages around 500 of our Macs across our client base. That's approximately 90% of our fleet.
- We started using Munki for deployment primarily back in 2013 at one client. Since then, we've done more than 30 Munki installs for groups as small as a single user, and as large as 150.

MUNKI IN A BOX

<https://github.com/tbridge/munki-in-a-box>

- Every Munki install we've done in the last two years has been largely the product of Munki in a Box. That doesn't mean that they were always done right the first time, and some of the choices that we've made have influenced choices that have been made in Munki in a Box.
- For example, we released last week a new version that makes overrides for the default recipes as part of the process and uses those by default.

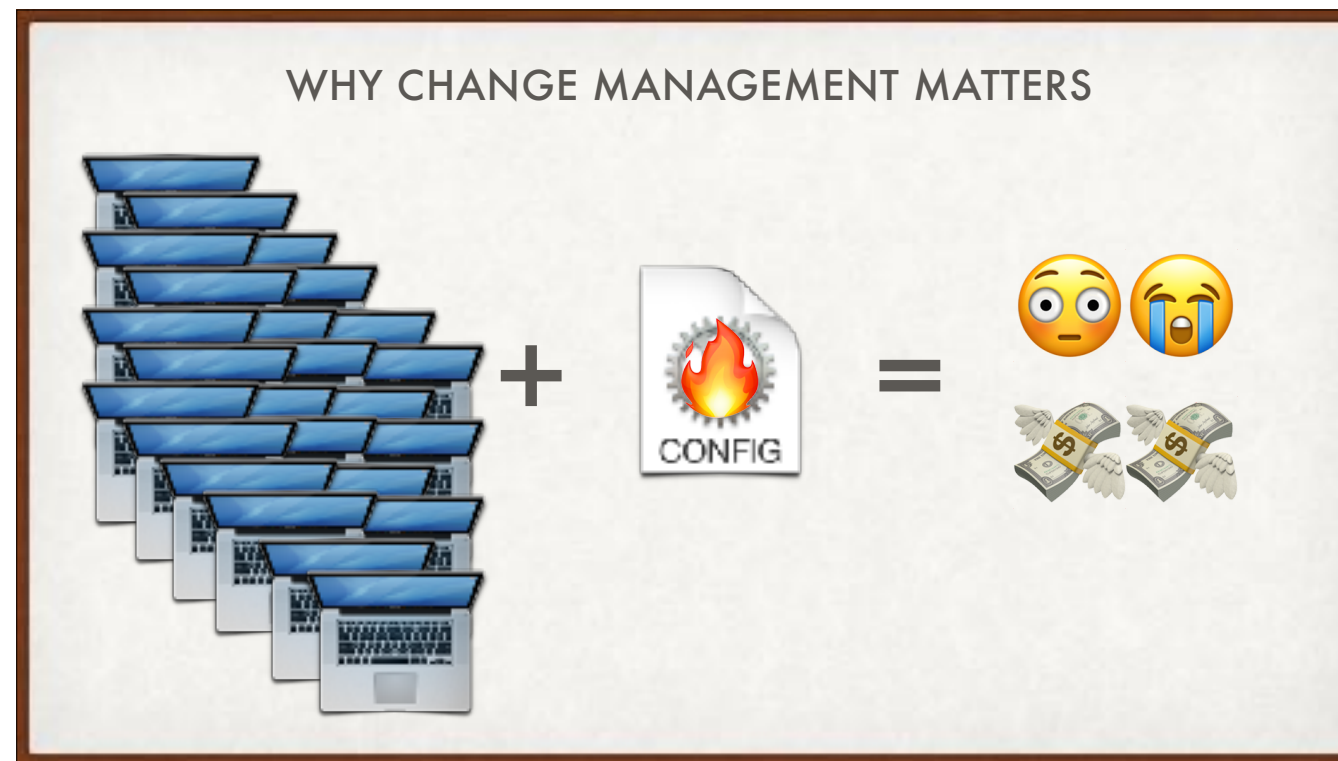
WHY CHANGE MANAGEMENT MATTERS

- Thinking carefully about fleet-wide decisions is a challenge that every IT department or consultant has whether they want it or not.
- You can go YOLO Life and just test your code in production, but that doesn't always have the best results.

WHY CHANGE MANAGEMENT MATTERS



- Do you really want to touch every machine in your fleet?
- Not without a real good reason
- This is why you test. If you've got a test machine and you make a mistake, the fixes are trivial at best, and a mild inconvenience at worst, and you can learn and iterate and develop fixes without disturbing the people that count on you.
- Because that's the thing: people are counting on you. Your role is to protect the systems from failure and to protect the organization's mission. There are stakes involved, and you have a serious role to play.



- Once upon a time, before I learned this lesson, we had a client with 40 machines and a central munki server. I thought I was being clever, and I pushed a payload free package to update a few of their settings.
- Hilarity ensued. The machines stopped checking into the locally created server. This necessitated a round robin visit to every machine in their environment, taking hours of time that we couldn't bill for.
- That's when I got real serious about careful setup of Munki environments, and about change management for their environments

BEST BETS FOR CHANGE MANAGEMENT



Mobile Device Managers

No, I'm not suggesting that you have to use Profile Manager. But it's the reference installation for the MDM Specification.


- Why? Confirmation and Fixability.
- Revisions to individual preferences are more straightforward to remedy in the future
- Out of band management. If you're updating Munki's management settings via Munki, if you make a mistake, you're out of luck. You now have a system that won't update *anything*.
- If you must script it, make sure the script fails safely if it fails and returns the old values, and that if the change is successful, you have some notification method of the change. We'll talk a little bit more about a good way for handling that in a bit.

BEST BETS FOR CHANGE MANAGEMENT


```
# /etc/puppetlabs/code/environments/production/site/profiles/mariadb/jenkins/mariadb
class ::profiles::jenkins::mariadb {
  String $jenkins_port = '9090',
  String $mysql_host = 'mysql',
  String $java_version = 'latest',
}

class { 'jenkins'
  configure_firewall => true,
  install_java      => false,
  port              => $jenkins_port,
  config_path       => {
    'http_port' => { 'value' => $jenkins_port },
    'jenkins_port' => { 'value' => $jenkins_port },
  },
}

class { 'java'
  distribution => $java_dist,
  version     => $java_version,
  before      => class['jenkins'],
}
```



```
10 # java::configure::db::db
11 # java::mariadb
12 # java::mariadb::db
13 # java::mariadb::db::db
14 # java::mariadb::db::db
15 # java::mariadb::db::db
16 # java::mariadb::db::db
17 # java::mariadb::db::db
18 # java::mariadb::db::db
19 # java::mariadb::db::db
20 # java::mariadb::db::db
21 # java::mariadb::db::db
22 # java::mariadb::db::db
23 # java::mariadb::db::db
24 # java::mariadb::db::db
25 # java::mariadb::db::db
26 # java::mariadb::db::db
27 # java::mariadb::db::db
28 # java::mariadb::db::db
29 # java::mariadb::db::db
30 # java::mariadb::db::db
31 # java::mariadb::db::db
32 # java::mariadb::db::db
33 # java::mariadb::db::db
34 # java::mariadb::db::db
35 # java::mariadb::db::db
36 # java::mariadb::db::db
37 # java::mariadb::db::db
38 # java::mariadb::db::db
39 # java::mariadb::db::db
40 # java::mariadb::db::db
41 # java::mariadb::db::db
42 # java::mariadb::db::db
43 # java::mariadb::db::db
44 # java::mariadb::db::db
45 # java::mariadb::db::db
46 # java::mariadb::db::db
47 # java::mariadb::db::db
48 # java::mariadb::db::db
49 # java::mariadb::db::db
50 # java::mariadb::db::db
51 # java::mariadb::db::db
52 # java::mariadb::db::db
53 # java::mariadb::db::db
54 # java::mariadb::db::db
55 # java::mariadb::db::db
56 # java::mariadb::db::db
57 # java::mariadb::db::db
58 # java::mariadb::db::db
59 # java::mariadb::db::db
60 # java::mariadb::db::db
61 # java::mariadb::db::db
62 # java::mariadb::db::db
63 # java::mariadb::db::db
64 # java::mariadb::db::db
65 # java::mariadb::db::db
66 # java::mariadb::db::db
67 # java::mariadb::db::db
68 # java::mariadb::db::db
69 # java::mariadb::db::db
70 # java::mariadb::db::db
71 # java::mariadb::db::db
72 # java::mariadb::db::db
73 # java::mariadb::db::db
74 # java::mariadb::db::db
75 # java::mariadb::db::db
76 # java::mariadb::db::db
77 # java::mariadb::db::db
78 # java::mariadb::db::db
79 # java::mariadb::db::db
80 # java::mariadb::db::db
81 # java::mariadb::db::db
82 # java::mariadb::db::db
83 # java::mariadb::db::db
84 # java::mariadb::db::db
85 # java::mariadb::db::db
86 # java::mariadb::db::db
87 # java::mariadb::db::db
88 # java::mariadb::db::db
89 # java::mariadb::db::db
90 # java::mariadb::db::db
91 # java::mariadb::db::db
92 # java::mariadb::db::db
93 # java::mariadb::db::db
94 # java::mariadb::db::db
95 # java::mariadb::db::db
96 # java::mariadb::db::db
97 # java::mariadb::db::db
98 # java::mariadb::db::db
99 # java::mariadb::db::db
100 # java::mariadb::db::db
```



Puppet

Chef

- Why? Confirmation and Fixability.
- Revisions to individual preferences are more straightforward to remedy in the future
- Out of band management. If you're updating Munki's management settings via Munki, if you make a mistake, you're out of luck. You now have a system that won't update *anything*.
- If you must script it, make sure the script fails safely if it fails and returns the old values, and that if the change is successful, you have some notification method of the change. We'll talk a little bit more about a good way for handling that in a bit.

TEST EARLY, TEST OFTEN



Photo by [Kecko](#), Creative Commons Licensed

- If you're not testing with virtual machines, now's a really good time to start. You should be testing your common cases at a bare minimum, and you should be testing your edge cases where you know they exist.
- Testing is how you make sure it's not going to massively fail in ways that you were able to predict ahead of deployment.
- It's straight forward to make a test lab out of virtual machines.
- Don't limit your testing to just virtual machines, though, try your tests out on hardware that matches the majority of your fleet, and at least one machine that matches your critical users. This is hard, sometimes. And sometimes you may not get the luxury. But if you don't test at all, you can't protect your people.

SOURCE CONTROL



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFTSLKDFJSDKLEJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAHAHAHAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

XKCD by Randall Munroe under Creative Commons License

- Source Control for Munki repositories can be a lot of work, but if you have a sizable fleet, it can help you avoid career-ending mistakes.
- ▼ You can use a private repo on GitHub even, it will help prevent mistakes. There are whole articles on doing this process, and if your repo is large, and the stakes high, you can save yourself substantially.
 - <http://grahampugh.github.io/2015/12/07/version-control-munki-autopkg.html>
 - <https://github.com/munki/munki/wiki/Munki-With-Git>
 - <https://www.afp548.com/2014/11/24/introduction-to-git-fat-for-munki/>
 - <https://www.afp548.com/2014/12/01/git-fat-intro-part-two-setup-and-migration/>
- Moving to source control for your Munki environment is going to put a lot of strictures on your changes to the environment, but moving away from the YOLO Life should be a theme as we talk through a lot of the problems that we've worked through.
- As a consultant, we spend a lot of time addressing problems and doing the day-to-day scut work of IT. A lot of our clients are small, and a lot of them don't have a lot of resources. Source Control can be a challenge of its own to implement well, and you have your work cut out for you.

AND NOW FOR MY PARADE OF MISTAKES

On to my parade of mistakes and how I've fixed them.

Some of these aren't full on mistakes, but the best practice may have changed, or the use case of the installation may have changed. The result is still the same: I have to fix something in a fleet-deployed environment without upsetting the apple cart.

Some changes are easy, some changes are hard. Here's how I dealt with them.

HTTP ONLY

HTTP ONLY

DIFFICULTY: EASY TO MEDIUM
IMPORTANCE: MEDIUM TO HIGH

- Why, oh why, would you do that? Hardwired network or lab machines.
- Get a commercial cert, or a CA-signed cert.
- Trust the certificate if necessary.
- Change the endpoint to use the certificate

▼ Why'd we do it?

- Simplicity. Hardwired network or lab machines.
- Certificates? Who Needs Certificates?

▼ Why'd we change it?

• We expanded the mission. Suddenly they were using munki not just in their lab, but on their laptops. Which would've been fine if they were chained to the desk, but they weren't. Laptops left.

- Importance Level: Medium-High
- Difficulty Level: Easy to Medium

▼ Solution

- ▼ Get a commercial cert, or a CA-signed cert.
- Trust the Cert if Necessary at the client side
- Change the endpoint to use the new https:// URL using a payload free package or other configuration change, or a bunch of interns with fast fingers and admin terminal access.

HTTP ONLY

DIFFICULTY: EASY TO MEDIUM
IMPORTANCE: MEDIUM TO HIGH

- Public Repository? There's more you must do.
- HTTP Basic Auth
- or
- Client Certificates

- ▼ If you're planning to expose your repository to the internet - and that may be a good idea if you have workers that regularly work from home, or plan to use their computers outside the office ever, consider a few extra steps:
- HTTP Basic Auth for providing a mechanism to authenticate that the clients pulling installer packages from your server are the ones you intend to do so
 - Client Certificates for providing a mechanism to stop rogue clients in the future.
 - Of the two, I would argue that the former is far more important than the latter, but if you're deploying more than just standard commercial software with your munki server, this is another method of making sure the machine can't get additional resources if it falls into the wrong hands, or the Basic Auth password is compromised.
 - Theoretically, you can have individual workstation Basic Auth keys. This is probably more overhead than Client Certificates.

ONE MANIFEST

ONE MANIFEST

DIFFICULTY: EASY TO MEDIUM
IMPORTANCE: VARIABLE

- Why?! Began as a lab build-out, expanded elsewhere.
- So, how do you expand from just using site_default?
- Munki uses four possible identifiers for manifests if the value is not set:
 - Fully Qualified Domain Name (tbridge.technolutionary.com)
 - Short Domain Name (tbridge)
 - Serial Number
 - site_default

- Individual manifests, tied back to a site_default manifest or other team-based manifests that you can manage organizationally, give you massively more flexibility than you would have with just a single master manifest.
- So, what are your options for creating manifests for everyone individually?
- ▼ If you don't want to change anything on your devices, munki will use some default identifiers to assign the manifest without your intervention. These are:
 - FQDN
 - Short hostname
 - Serial Number
 - site_default
- Personally, I tend to gravitate toward the serial number. You have a record of it somewhere (I hope) and you can easily instantiate it by duplicating the manifest of another machine and then editing it.
- Spend some time reading this guide to Manifests by Victor Vrantchan: <https://groob.io/posts/manifest-guide/>

ONE MANIFEST

DIFFICULTY: EASY TO MEDIUM
IMPORTANCE: VARIABLE

- Read Victor's Manifest Guide:
<https://groob.io/posts/manifest-guide/>
- Read Alan's Manifest Guide:
<https://technology.siprep.org/another-opinionated-guide-to-munki-manifests/>
- Make new manifest files (You can — and should — script this!) and test.

Read Victor's Opinionated Guide to Manifests, and then Read Alan's. They're two strong viewpoints that you won't get confused reading. They have multiple approaches to this depending on whether or not you've already trod the garden path away from site_default, as well as using Munki Web Admin with Serial Number based manifests in a clear and sane way.

SINGLE CATALOG

WHAT?!

SINGLE CATALOG

DIFFICULTY: EASY TO MEDIUM

IMPORTANCE: VERY %^&*ING HIGH

- Create a Testing Catalog and a Production Catalog.
- Select a group of testers
- What's a good test?

- “Oh, all this stuff is pretty safe. Nothing too funny here.”
- Then they got a design team using Creative Cloud.

Living the YOLO life might be something you need to do if updates carry more priority in your org than the possible pitfalls of those updates.

Creating a catalog is easy - just add it to a pkginfo file of an existing package, and issue make catalogs.



FIND GOOD TESTERS

- Never just one person
- Never someone whose application scope is limited
- Never someone whose work is indispensable (you get to help them decide who's indispensable. Not having testers isn't a good option either.)
- Not just IT Department testers.

MAKE GOOD TESTS

- ▼ What's a good test?
 - Launch the application. Does it crash?
 - Okay, maybe not that simplistic. If we were in education, what we'd do is to create a rubric for evaluating updates.
 - Pick some critical tests that your primary applications must pass and evaluate them against that.
 - ▼ Browser:
 - Must load organization critical websites (Okta, Timesheets, Help Desk, WashingtonPost.com, ESPN.com, Something with Flash in it if you're permitting Flash in your organization.)
 - ▼ Office Suite:
 - Pick a clear set of documents that your organization uses often. Make a copy. Test updates to these documents, save them, distribute them to another workstation and make sure they open on the old version.
 - Print something.
 - ▼ Creative Cloud:
 - The hardest of our workflows to test, because unless you are also a graphic designer, you're going to need to borrow one to test the environment more fully.
 - Work with a designer to build the rubric of tasks and workflows.
 - Cover the printing process.
 - Cover 3rd Party Tools and

SHIRLEY WILL FIND YOU.



Vet the tools carefully and don't mess it up. Shirley will find you.

IMPROPER DNS & SWITCHING SERVERS



Photo by [Philip Brewer](#)

- I remember the monkey bars on the playground well, you'd swing from one ring to another, trusting that you had enough momentum to reach the next ring or you'd fall into the tan bark at the bottom. That tan bark was brutal. Sharp corners, itchy if it got under your shirt, but I suppose it was better than falling five feet onto asphalt. The crazy things they let you do as kids.
- All of that serves to say: you don't want to have your users fall down into the tan bark as you shift between munki servers.
- I've done this wrong, and it was a huge pain, and I'm here primarily to provide you with some good guidelines for how to prepare yourself for this move.

USE A PER-SERVICE FQDN FOR SERVICES

- ▼ If you're not using pure service fully qualified domain names for your software deployment, now is the time to start. Our very first munki server shared the name of the box that served up the updates. `server.clientname.com`
 - Server had many roles at the organization. It was the OD Master, then the Wiki Server (yes, the Wiki Server. They customized it. Heavily.), then also a File Server, and later still the Host for their phone switch. This was terrifying to us, and we've been raising the alarm with this client for some time.
 - Finally, we had permission to start to migrate things away and we realized that we were stuck. We couldn't just change the DNS to match the new munki server. Too many things relied on the old `server.clientname.com` domain.
 - If only we'd used just a service record, we could move the repository to the new apache or nginx container, switch the service record, and continue about our day.
 - But we didn't do that. So we had to change the `SoftwareRepoURL` on all of our clients to get them across the gap.



Photo by [Chuckas McFly](#)

- So how did we choose to get there? This client has not opted into using MDMs yet, so we couldn't just push a profile update.
- We opted to write a tool to do it for us. Since we were just moving the repository to a new home, and changing the URL, we only needed to alter a single setting: SoftwareRepoURL
- But how do we make sure this works?
- We thought about it generating a notification file somewhere on the server, but that sounded frustrating to check. We thought about it generating a service email, but that even sounded tedious. I hate email, I get way too much of it.
- What about Slack?
- So I wrote a script that would check the Munki Preferences for a specific key, exit normally if it was set right, but if it wasn't set right, it would make the change we were looking for, and then use a Slack web hook to tell us that it had gone okay after all. Then we could verify with munkireport or through log analysis, that everything had gone correctly.

Change Munki, Tell Slack

```
#!/bin/sh
set -e

# We've made a change, now let's tell Slack.

IDENT=$(defaults read $PPM ClientIdentifier)

read -d "" SLACK_PAYLOAD_DATA << EOF
{
  "channel": "$CHANNEL",
  "username": "MunkiBot",
  "icon_emoji": ":monkey-face:",
  "attachments": [
    {
      "fields": [
        {
          "title": "Munki client update",
          "value": "This is a client update from $IDENT. I have updated $PREPOT-CK to be $SHOULDGE successfully.",
          "short": false
        }
      ]
    }
  ]
}
EOF

SLACK_COMMAND="curl -X POST --data-urlencode 'payload=${SLACK_PAYLOAD_DATA}' ${SLACK_HOOK}"
```

<https://gist.github.com/tbridge/1906798f5957ff2559575d5190712ec0>

- So I wrote a script that would check the Munki Preferences for a specific key, exit normally if it was set right, but if it wasn't set right, it would make the change we were looking for, and then use a Slack web hook to tell us that it had gone okay after all. Then we could verify with munkireport or through log analysis, that everything had gone correctly.



- So how did we choose to get there? This client has not opted into using MDMs yet, so we couldn't just push a profile update.
- We opted to write a tool to do it for us. Since we were just moving the repository to a new home, and changing the URL, we only needed to alter a single setting: SoftwareRepoURL
- But how do we make sure this works?
- We thought about it generating a notification file somewhere on the server, but that sounded frustrating to check. We thought about it generating a service email, but that even sounded tedious. I hate email, I get way too much of it.
- What about Slack?
- So I wrote a script that would check the Munki Preferences for a specific key, exit normally if it was set right, but if it wasn't set right, it would make the change we were looking for, and then use a Slack web hook to tell us that it had gone okay after all. Then we could verify with munkireport or through log analysis, that everything had gone correctly.

MAKE VERIFIABLE CHANGES

All this is to say: Make sure that your changes verify that they're done properly. When you write scripts to change settings, make sure you get a verification that the change completes on a given host and point to that as a record.

Compare that list of clients using something verifiable. If you're using Munki Report or something similar, you can view the error logs of a given environment to make sure that machines aren't still checking in on the old server, and then re-issue those scripts as necessary.

When you write a process like this to move between two host environments, make sure you get everyone.



BUT REALLY, YOU NEED A CONFIG MANAGER

But really, these scripts and hoops that we all just jumped through are the sort of things that are only necessary when you've decided to ignore what Apple has given you in the MDM specification, or what other (larger) organizations have been giving you for some time longer: configuration management that is verifiable and auditable. Puppet, Chef, these are the kinds of professional tools that get used in large fleets for reasons: you want to make sure that your changes get made across a fleet the size of Google's or the size of Facebook's. There's no reason you can't do that for a fleet of 50 or 100. It's just more work. But it's all more work.

Better busy than bored.

MUNKI MISTAKES MADE RIGHT

TO SUM UP

- Change Management Matters
- Without Change Management, Write Scripts
- Use Service-based DNS Everywhere
- Never Leap Without a Plan.
- Use Munki's Features As Much As You Can