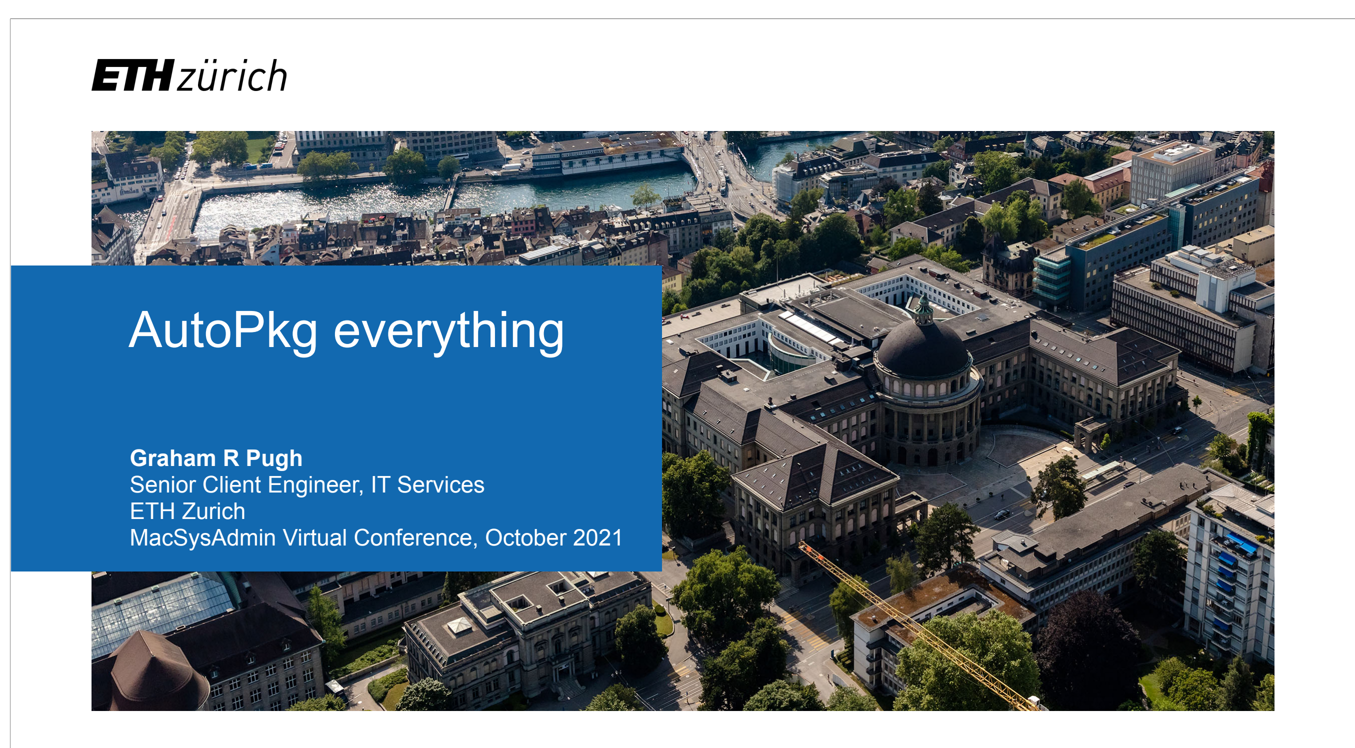
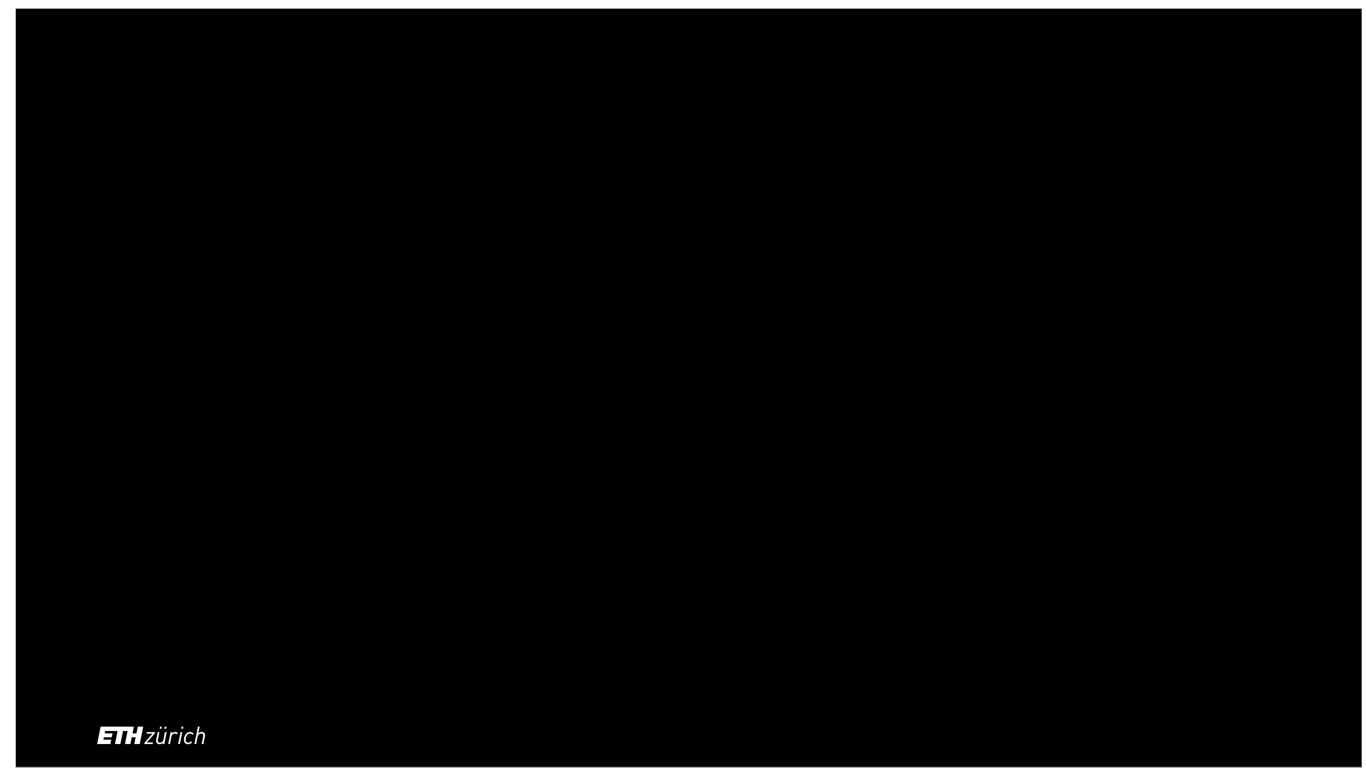
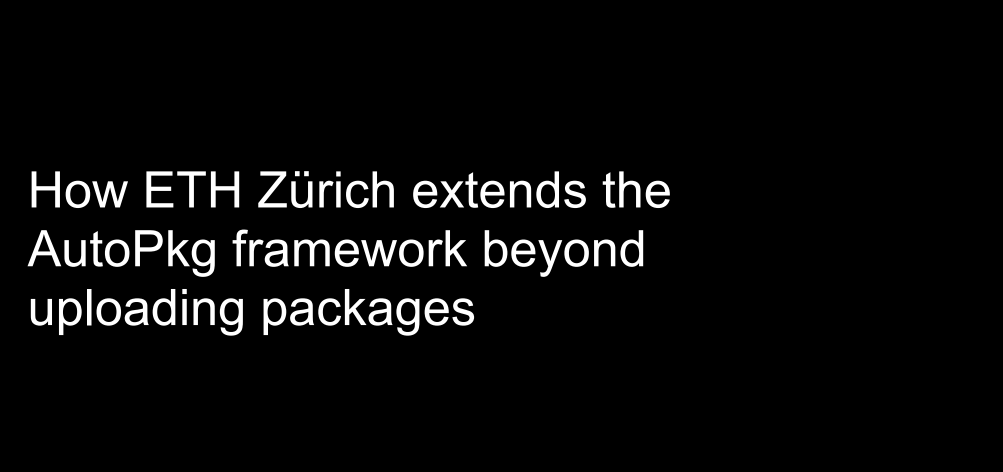

ETH zürich
AutoPkg everything
Graham R Pugh
Senior Client Engineer, IT Services
ETH Zurich
MacSysAdmin Virtual Conference, October 2021



Hello Sweden! And thank you to Patrik, I'm honoured to speak today at my first MacSysAdmin conference, albeit it in the cloud.

My name is Graham Pugh, I'm a Mac deployment engineer from the UK, and I've now been living in Switzerland for over 5 years, and working at ETH Zurich for 4 and a half of those.



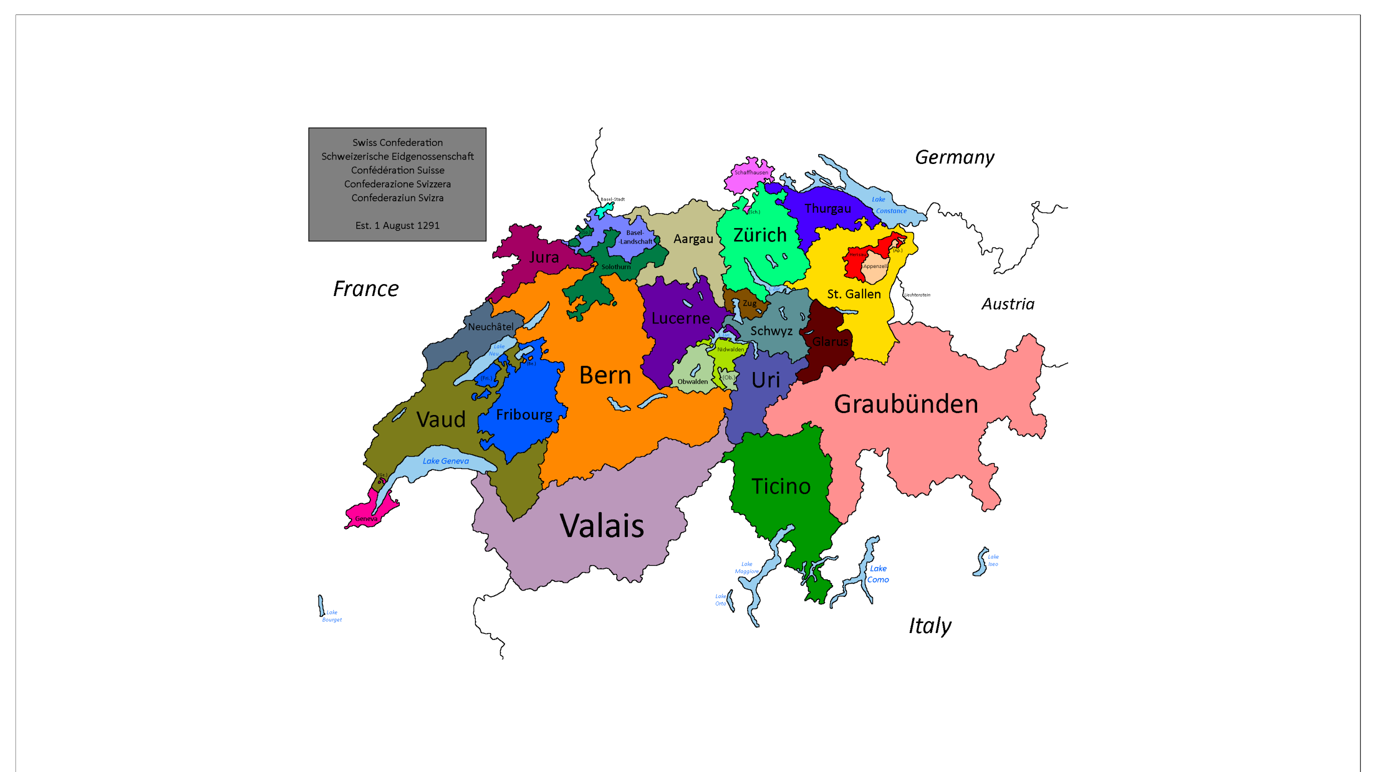
The goal of my presentation today is to talk about why we use AutoPkg, and how we have extended our use of the AutoPkg framework beyond the upload of packages to a more complete deployment, testing, staging and multi-tenant distribution workflow.

Although we are a Jamf Pro shop, I hope this will be of interest to any of you involved in package deployment to Macs, and perhaps inspire some of you to try your hand at creating your own AutoPkg processors when the supplied ones don't fit all of your needs.

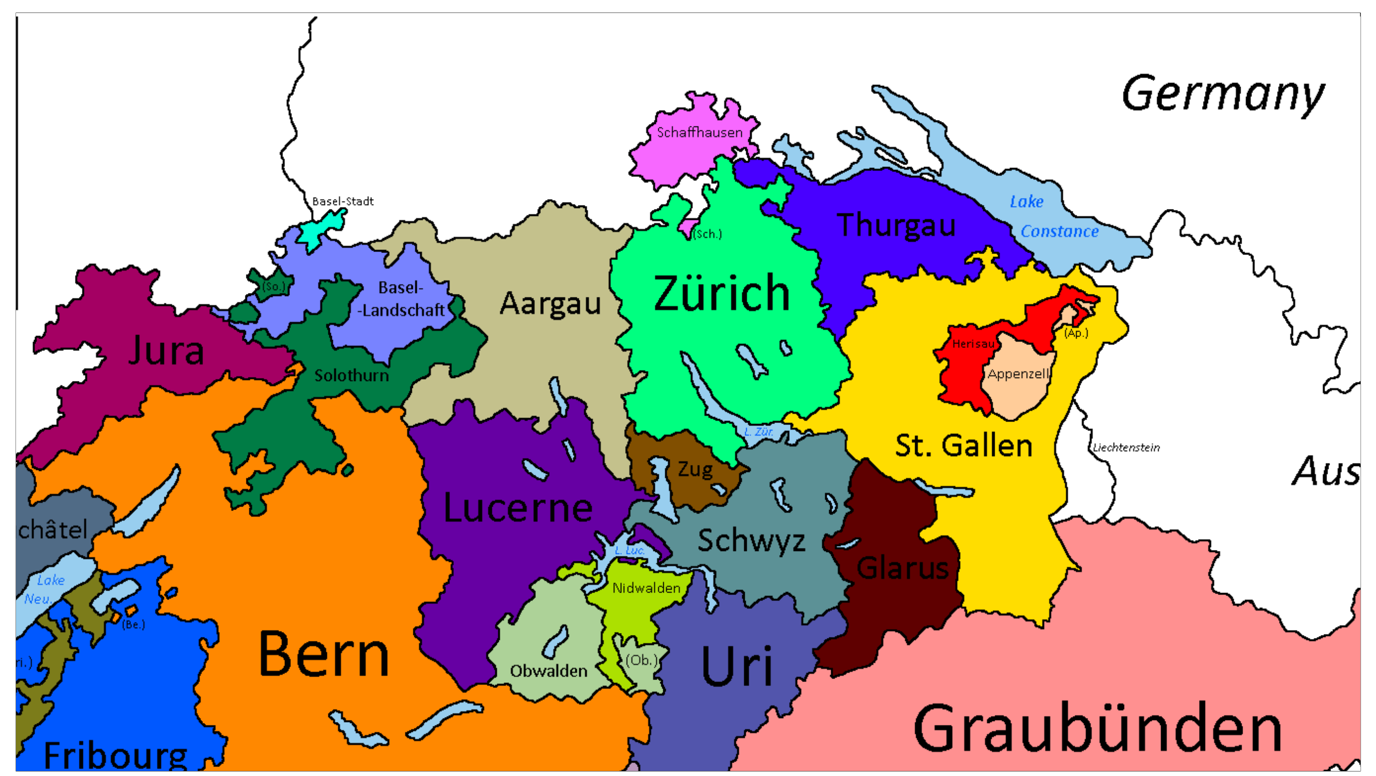
For those of you that don't know, 🍎ETH Zurich is the Swiss Federal Institute of Technology.

We are considered the best university in continental Europe.

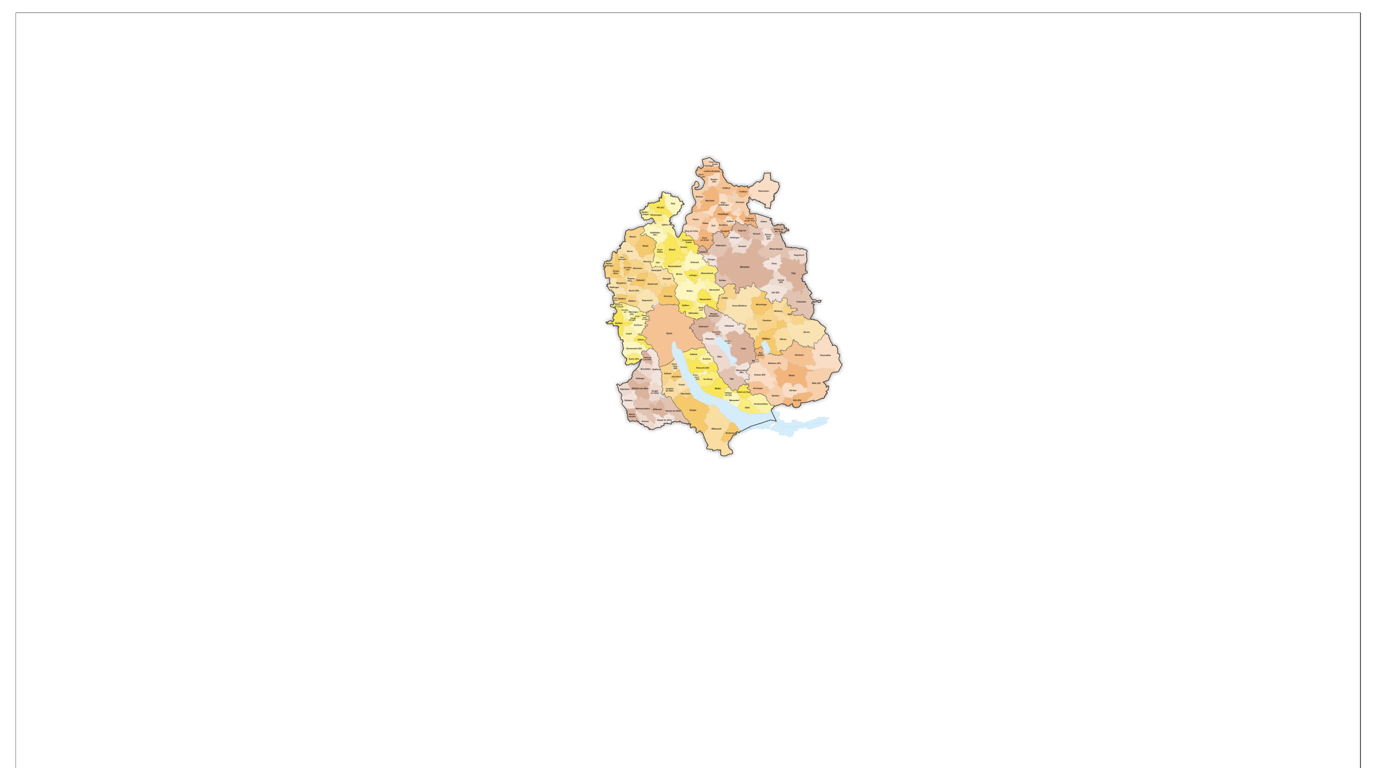
ETH
Eidgenössische
Technische
Hochschule



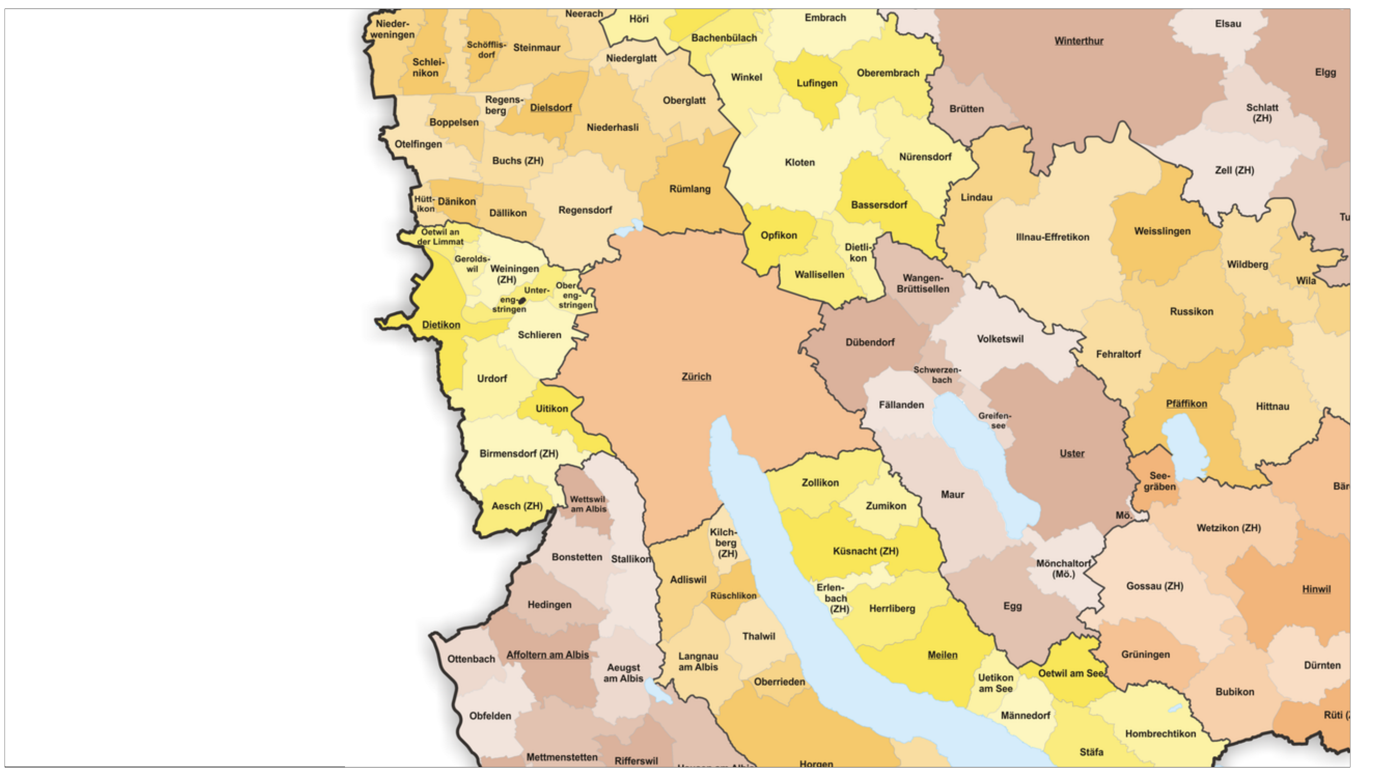
In Switzerland, everything is federated. The country is divided up into very autonomous cantons



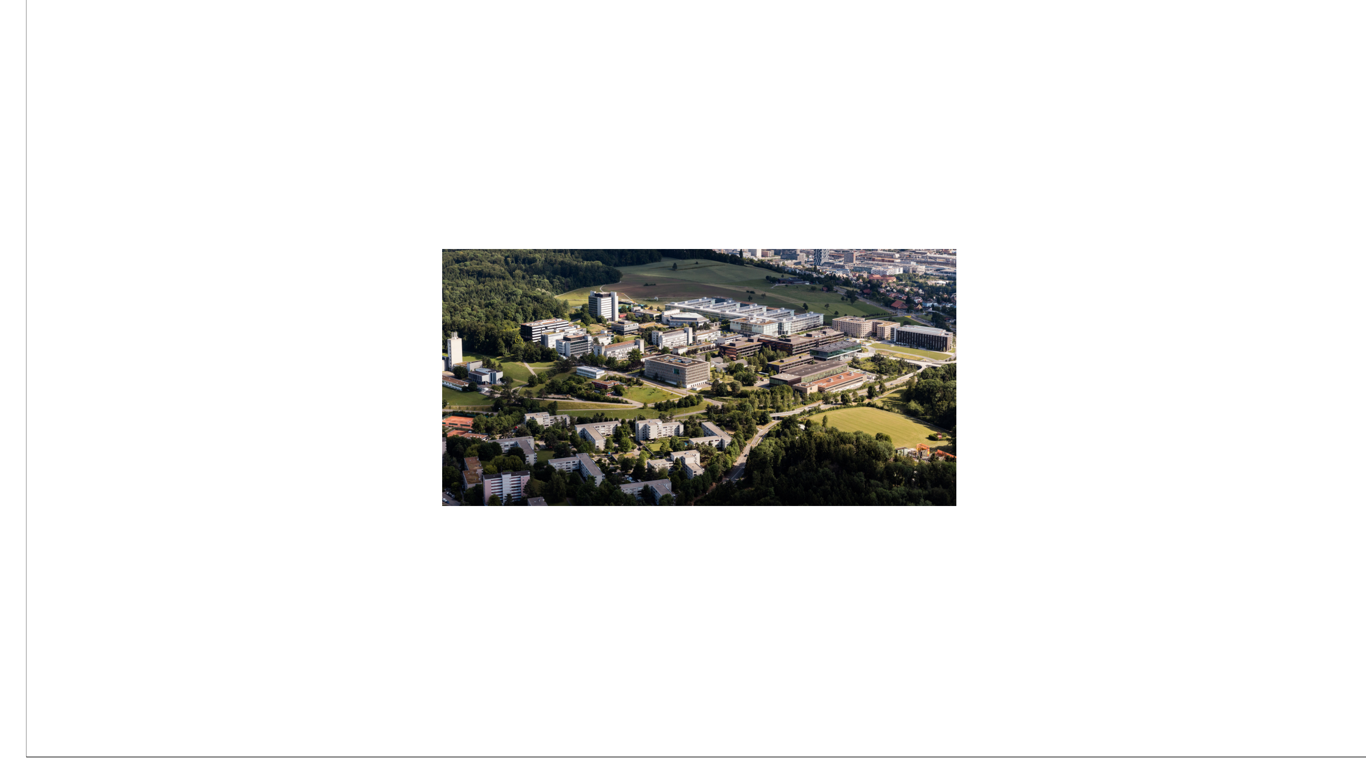
Germany
Schaffhausen
Basel-Stadt
Lake
Constance
Thurgau
(Sch.)
Zürich
Aargau
Basel-
-Landschaft
(So.)
Jura
Solothurn
Herisau
(Ap.)
Appenzell
St. Gallen
Liechtenstein
Aus
L. Zür.
Zug
Lucerne
châtel
Schwyz
Glarus
L. Luc.
Lake
Neu
Nidwalden
(Be.)
Bern
Obwalden
(Ob.)
Uri
Graubünden
Fribourg



And the cantons are divided into quite autonomous districts.



Nieder-weningen
Schöfflis-dorf
Steinmaur
Neerach
Höri
Bachenbülach
Embrach
Winterthur
Elsau
Schlei-nikon
Niederglatt
Winkel
Lufingen
Oberembrach
Elgg
Regens-berg
Dielsdorf
Oberglatt
Brütten
Schlatt (ZH)
Boppelsen
Niederhasli
Otelfingen
Buchs (ZH)
Kloten
Nürensdorf
Zell (ZH)
Rümlang
Lindau
Hütt-ikon
Dänikon
Dällikon
Regensdorf
Bassersdorf
Oetwil an der Limmat
Opfikon
Illnau-Effretikon
Weisslingen
Geroldswil
Weiningen (ZH)
Dietli-kon
Wallisellen
Wangen-Brüttisellen
Wildberg
Unter-eng-stringen
Ober-eng-stringen
Dietikon
Schlieren
Volketswil
Russikon
Dübendorf
Fehraltorf
Zürich
Schwerzen-bach
Urdorf
Uitikon
Fällanden
Greifen-see
Pfäffikon
Hittnau
Birmensdorf (ZH)
Uster
See-gräben
Wettswil am Albis
Zollikon
Maur
Aesch (ZH)
Zumikon
Wetzikon (ZH)
Kilch-berg (ZH)
Küsnacht (ZH)
Bonstetten
Stallikon
Mönchaltorf (Mö.)
Adliswil
Gossau (ZH)
Hinwil
Rüschlikon
Erlen-bach (ZH)
Herrliberg
Hedingen
Egg
Thalwil
Ottenbach
Affoltern am Albis
Grüningen
Meilen
Dürnten
Aeugst am Albis
Langnau am Albis
Uetikon am See
Oetwil am See
Oberrieden
Bubikon
Obfelden
Männedorf
Hombrechtikon
Stäfa
Mettmenstetten
Rifferswil
Horgen



And ETH itself is no different. Our professors operate on a very autonomous basis, as do our institutes and departments.

IT support tends to be organised at a departmental or institute level. Central governances and services exist of course, but often on a consensus basis, with departments able to set their own standards and processes, and to choose whether to use centrally offered services, or provide their own solutions. On the technical side, I believe only network and telephony are entirely centralised.

There are a few thousand managed Apple devices at ETH Zurich, spread across most academic departments as well as central services.

I work for a division of the central IT Services, 🍎in a small team of 3, with my colleagues Max and Kati.

We offer various services as well as Apple device expertise to all sections of the university, and one of our primary goals is to make software available to departmental IT support teams in a deployable form, and to provide a delivery mechanism for that software.

Our organisation's distributed internal structure means that each department requires their own individual admin environment. In this respect, we operate somewhat like an MSP. Therefore, our automation needs are very high, and multi-context capabilities of management solutions are required.

🍎 As I was joining ETH, the process had just begun of transitioning to using Jamf Pro for our package deployment and burgeoning MDM requirements, a decision based largely on the API and multi-context capabilities of Jamf Pro.

🍎 To get their own admin environment, each customer was to get their own Jamf Pro instance. By now, this has grown to over 35 instances.

ETH Zurich Apple Services
ETH zürich

ETH Zurich Apple Services

Katiuscia Zehnder
Max Schlapfer
Graham Pugh

ETH zürich

ETH Zurich Apple Services

Katiuscia Zehnder
Max Schlapfer
Graham Pugh

ETH zürich

ETH Zurich Apple Services
Katiuscia Zehnder
Max Schlapfer
Graham Pugh
jamf
PRO
ETH zürich

The design our software deployment workflow in Jamf Pro, is probably the same as many of you:

🍎Downloading, verifying and repackaging the software installers into a deployable form.

🍎Uploading the package into Jamf Pro, and creating the necessary policy and smart group so that there is a self service item for our testers.

🍎The people doing the testing have to do that manually - we don't have any automated software testing at present

🍎And then when that package is ready for release, we need to create or update our production policies and smart groups with the new package, and remove the testing policy.

🍎 For us, this is complicated by the need to repeat everything on each Jamf instance

Package deployment workflow

- Check for a new version of a software
- Download new version
- Verify security
- Repackage as necessary

Package deployment workflow

- Check for a new version of a software
- Download new version
- Verify security
- Repackage as necessary
- Upload package to Jamf Pro
- Create or update policies and smart groups targeted at testers

Package deployment workflow

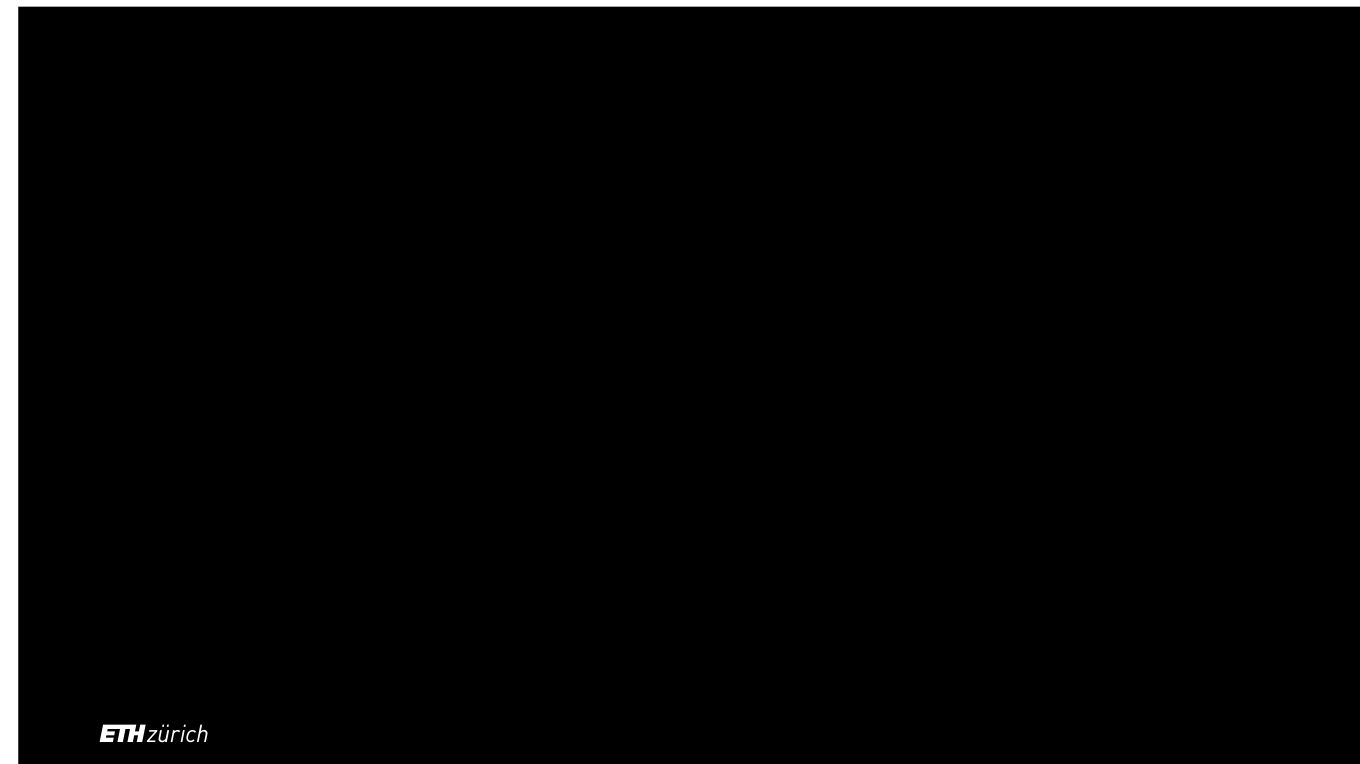
- Check for a new version of a software
- Download new version
- Verify security
- Repackage as necessary
- Upload package to Jamf Pro
- Create or update policies and smart groups targeted at testers
- Testers do the tests

Package deployment workflow

- Check for a new version of a software
- Download new version
- Verify security
- Repackage as necessary
- Upload package to Jamf Pro
- Create or update policies and smart groups targeted at testers
- Testers do the tests
- Create or update policies and smart groups available to production computers
- Delete the testing policies

Package deployment workflow

- Check for a new version of a software
- Download new version
- Verify security
- Repackage as necessary
- Upload package to Jamf Pro
- Create or update policies and smart groups targeted at testers
- Testers do the tests
- Create or update policies and smart groups available to production computers
- Delete the testing policies



Before migration to Jamf Pro, my colleagues and my predecessor were already automating parts of this workflow with in-house build scripts, though there was no API for importing packages into the deployment system.

I came into the organisation having used AutoPkg at previous organisations with Munki.
With Jamf Pro, the opportunity to use the API, and the existence of JSSImporter to upload packages, was a big impetus for us to start to benefit from using AutoPkg recipes and tap into that huge amount of work already being done in the Mac admin community.

Package deployment workflow

AutoPkg {

- Check for a new version of a software
- Download new version
- Verify security
- Repackage as necessary
- Upload package to Jamf Pro
- Create or update policies and smart groups targeted at testers
- Testers do the tests
- Create or update policies and smart groups available to production computers
- Delete the testing policies

ETH *zürich*

For the downloading and packaging of installers, we could in many cases use existing AutoPkg recipes, and we starting writing our own recipes for whatever apps didn't already have a recipe.

🍎 For uploading the package to Jamf Pro for testing, we did have to write our own JSS recipes because the standard templates were not right for us, but at least this was often just a small change from the ones that the community had already made.

🍎 But there was no obvious, existing method for creating or updating production policies and smart groups available from the community. The jamf_helper API tool was designed to do this but only worked within the constraints of standard JSS recipe design. Most of the community do this part of the workflow manually, or skip the testing phase completely.
And Jamf's Patch Management solution promised much, but turns out to be complex to maintain, and is only able to update applications on clients, not to install them afresh, so doesn't negate the need to create production policies.

Package deployment workflow

AutoPkg {
- Check for a new version of a software
- Download new version
- Verify security
- Repackage as necessary

AutoPkg + JSSImporter {
- Upload package to Jamf Pro
- Create or update policies and smart groups targeted at testers

- Testers do the tests
- Create or update policies and smart groups available to production computers
- Delete the testing policies

Package deployment workflow

AutoPkg
- Check for a new version of a software
- Download new version
- Verify security
- Repackage as necessary

AutoPkg + JSSImporter
- Upload package to Jamf Pro
- Create or update policies and smart groups targeted at testers
- Testers do the tests

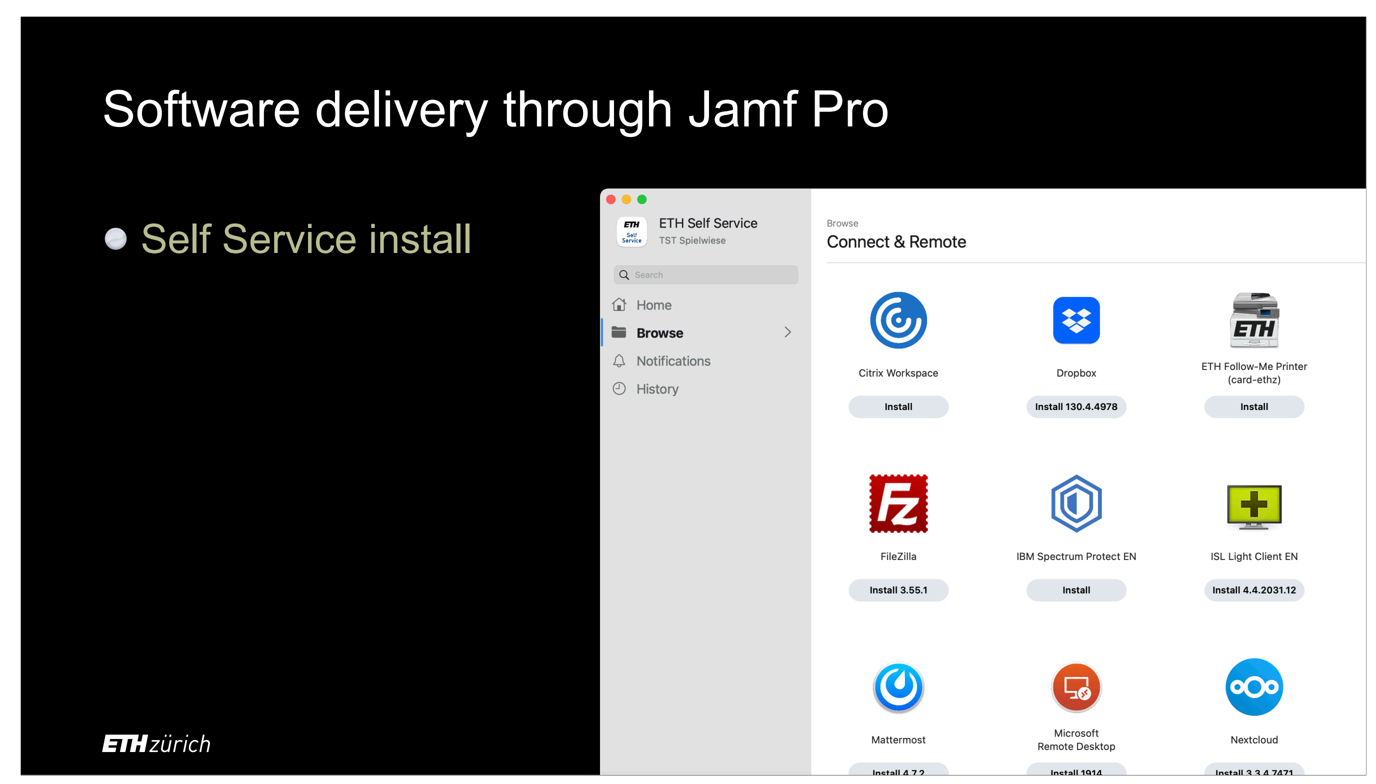
?
- Create or update policies and smart groups available to production computers
- Delete the testing policies

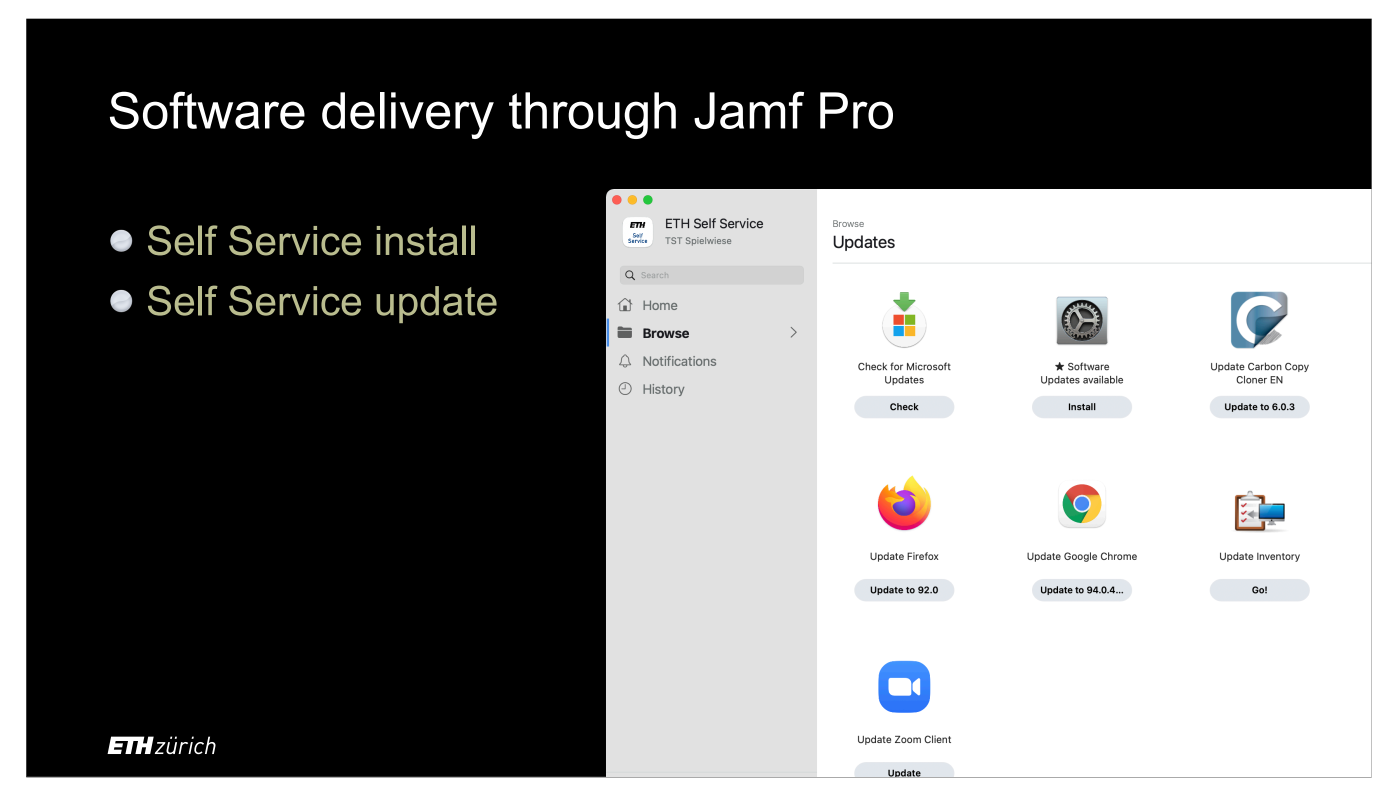
When discussing the transition to Jamf with our customers, we knew that we had to do provide more than just uploaded packages to Jamf. Many admins are sole operators who couldn't manually create all the policies and smart groups required themselves, and, on the other hand, our small team can't handle all the scoping to computers on behalf of each and every IT support team - most didn't want that anyway.

It also became apparent that a single deployment strategy was not going to be enough.

🍎Some groups wanted to use the Self Service kiosk application for installs, 🍎some for updates.

🍎Others wanted to automatically deliver software to their users, which was closer to how things had been done before.
Some wanted a combination of both options, for example self service install but automatic update.
And their needs were sometimes different for each software title.


Software delivery through Jamf Pro
Self Service install
ETH Self Service
TST Spielwiese
Search
Home
Browse
Notifications
History
Browse
Connect & Remote
Citrix Workspace
Install
Dropbox
Install 130.4.4978
ETH Follow-Me Printer (card-ethz)
Install
FileZilla
Install 3.55.1
IBM Spectrum Protect EN
Install
ISL Light Client EN
Install 4.4.2031.12
Mattermost
Microsoft Remote Desktop
Nextcloud
ETH zürich


Software delivery through Jamf Pro
Self Service install
Self Service update
ETH Self Service
TST Spielwiese
Home
Browse
Notifications
History
Browse
Updates
Check for Microsoft Updates
Check
★ Software Updates available
Install
Update Carbon Copy Cloner EN
Update to 6.0.3
Update Firefox
Update to 92.0
Update Google Chrome
Update to 94.0.4...
Update Inventory
Go!
Update Zoom Client
Update
ETH zürich

Software delivery through Jamf Pro

- Self Service install
- Self Service update
- Auto-install
- Auto-update

Jamf policies

- Firefox (Testing)

TESTING

PRODUCTION

- Update Firefox
- Auto-install Firefox
- Auto-update Firefox
- Install Firefox
- Uninstall Firefox
- Trigger-uninstall Firefox

Jamf smart groups

- Firefox test users
- Firefox test version installed

TESTING

PRODUCTION

- Firefox users
- Firefox auto-install
- Firefox auto-update
- Firefox current version installed
- Firefox installed

ETH zürich

To give our customers all those options, we designed a set of policies and smart groups for each software title.

I won't go into an explanation of how this all connects together as it's very Jamf-specific and would take too long. I have a blog post about it which I'll link to at the end. The important part of this design is that there is a separation between 🍎 the items that need to be updated when a new version is released to test or production,
🍎 And the items that our customers need to edit themselves in order to scope software to devices.

Jamf policies

- Firefox (Testing)

- Update Firefox
- Auto-install Firefox
- Auto-update Firefox
- Install Firefox
- Uninstall Firefox
- Trigger-uninstall Firefox

Jamf smart groups

- Firefox test users
- Firefox test version installed

TESTING

PRODUCTION

- Firefox users
- Firefox auto-install
- Firefox auto-update
- Firefox current version installed
- Firefox installed

Jamf policies

- Firefox (Testing)

TESTING

PRODUCTION

- Update Firefox
- Auto-install Firefox
- Auto-update Firefox
- Install Firefox
- Uninstall Firefox
- Trigger-uninstall Firefox

Jamf smart groups

- Firefox test users
- Firefox test version installed

- Firefox users
- Firefox auto-install
- Firefox auto-update
- Firefox current version installed
- Firefox installed

Jamf policies

- Firefox (Testing)

TESTING

PRODUCTION

- Update Firefox
- Auto-install Firefox
- Auto-update Firefox
- Install Firefox
- Uninstall Firefox
- Trigger-uninstall Firefox

Jamf smart groups

- Firefox test users
- Firefox test version installed

TESTING

PRODUCTION

- Firefox users
- Firefox auto-install
- Firefox auto-update
- Firefox current version installed
- Firefox installed

ETH zürich

We designed this with automation in mind. Creating all these items in Jamf manually for each software title, plus any other necessities like scripts and extension attributes - even the items that don't need to change once created - would be a mammoth and frankly impossible task across so many Jamf instances for each of our hundred plus software titles.

We absolutely had to automate it all. So we did. Some things have evolved, some are pretty much how originally designed.

This presentation is about how we did it, ultimately involving AutoPkg for everything.

Before I go on to talk about those workflows, it's important to mention one of the major things that has evolved while scaling up our service.

We started off heavily using JSSImporter, and writing JSS recipes. Early on though, I needed to go beyond its primary use case of supplying packages for testing to a Jamf Pro instance, with policies in a specific form.

I ended up taking over its maintenance, which had otherwise stalled, and this helped us a bit, as I was able to tweak it a little to suit our needs to some extent.

But as we continued to scale up, and as the world moved on with things like python 2 deprecation, I have become more frustrated with JSSImporter, and not really been able to conceptualise a good evolution for it. I needed something more flexible and, frankly, easier to maintain.

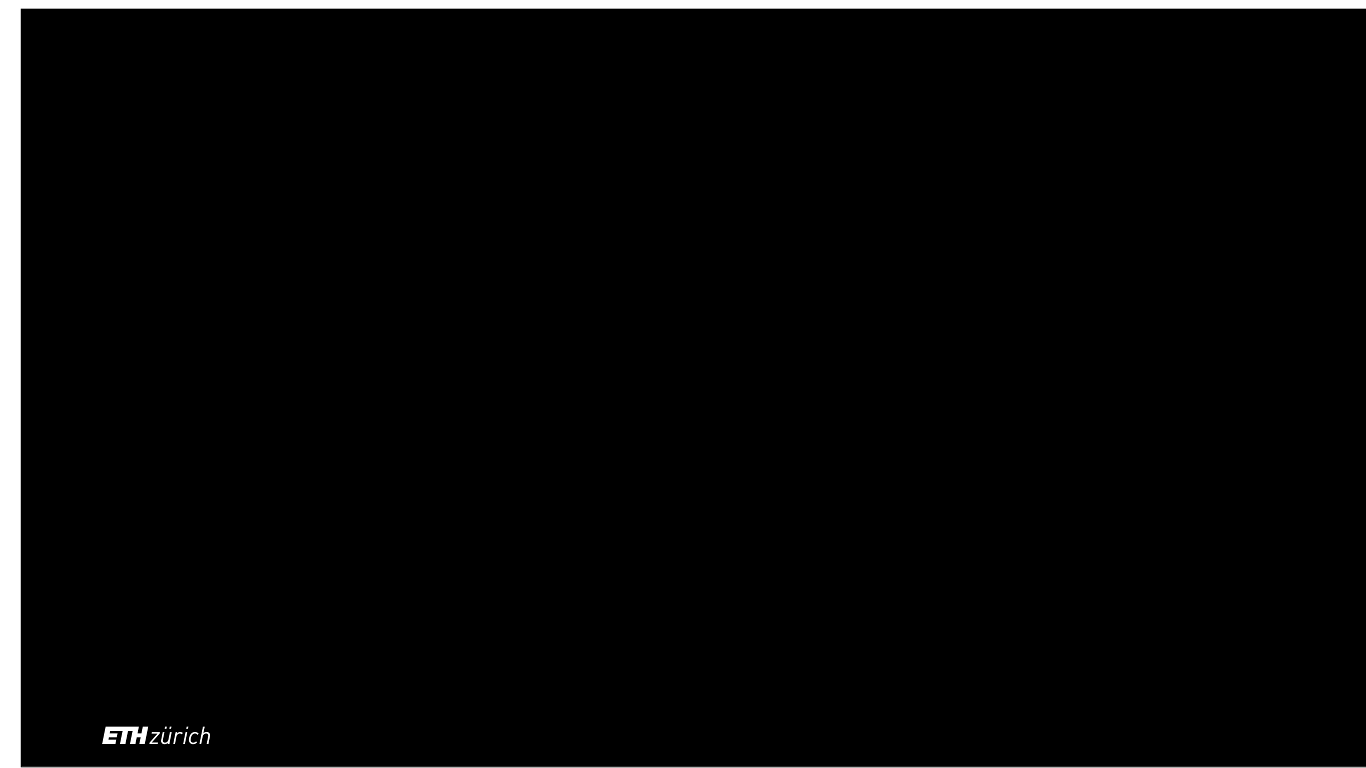
🍎So, I eventually bit the bullet, and wrote a new set of processors that I collectively call JamfUploader. We've been using these in production for almost a year now, and I'm transitioning all my recipes to .jamf recipes to use the new processors.

This presentation is not going to concentrate on those processors. Anthony Reimer and I have recorded a presentation about JamfUploader for the Virtual JNUC Conference this year, which will air in just a couple of weeks' time, on October 21.

JamfUploader

`Firefox.jamf`

ETH zürich



Just very briefly for context, because you'll see them crop up during the session, they are a suite of processors to make individual requests to the Jamf Pro APIs to achieve single tasks like upload a package, a script or a policy, for example.
Unlike JSSImporter, which has to be installed as a separate package on top of AutoPkg, JamfUploader processors are ordinary Shared Processors which require no installation. You just use the ones you need for your recipe.
I do encourage any of you currently using JSSImporter to check out this project. Since I am the maintainer of both, you can guess which one is going to get more maintenance in the future.

JamfCategoryUploader

JamfExtensionAttributeUploader

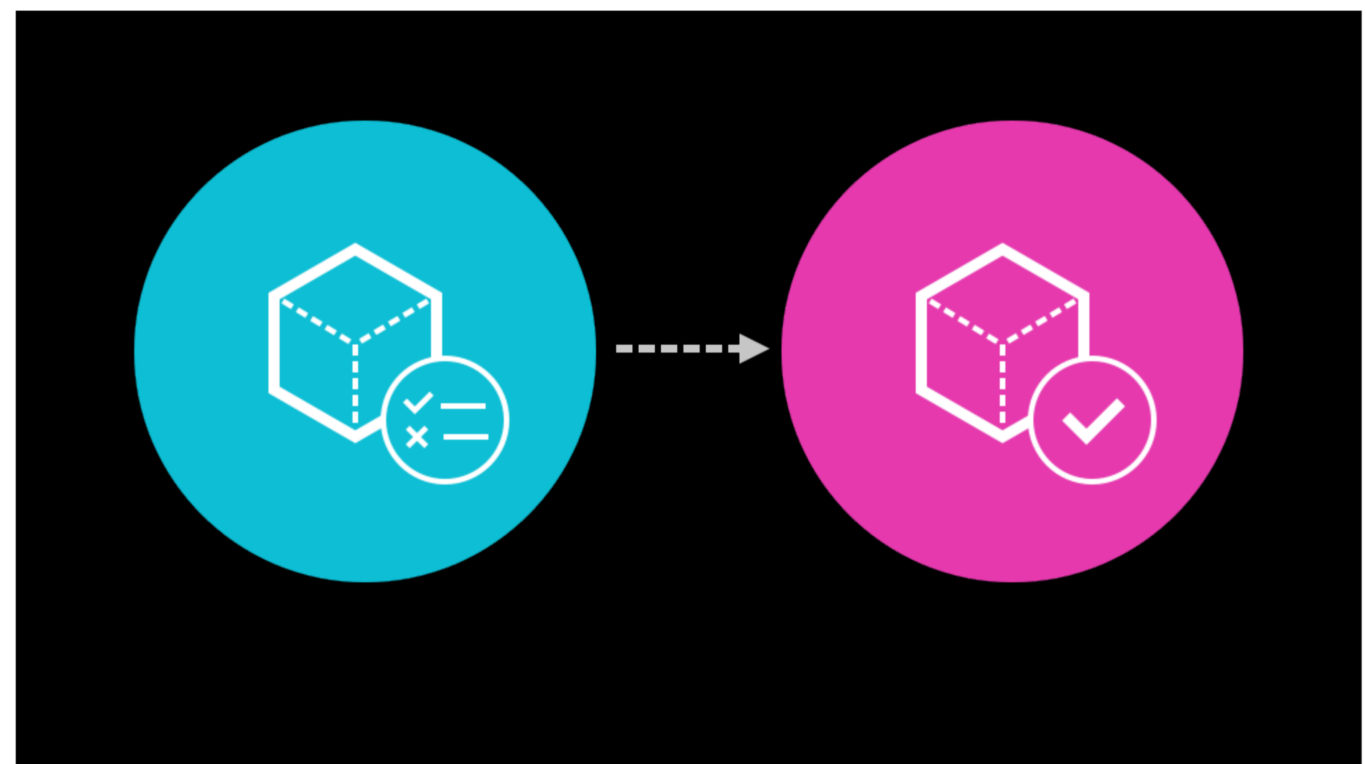
JamfPackageUploader

JamfScriptUploader

JamfComputerGroupUploader

JamfPolicyUploader

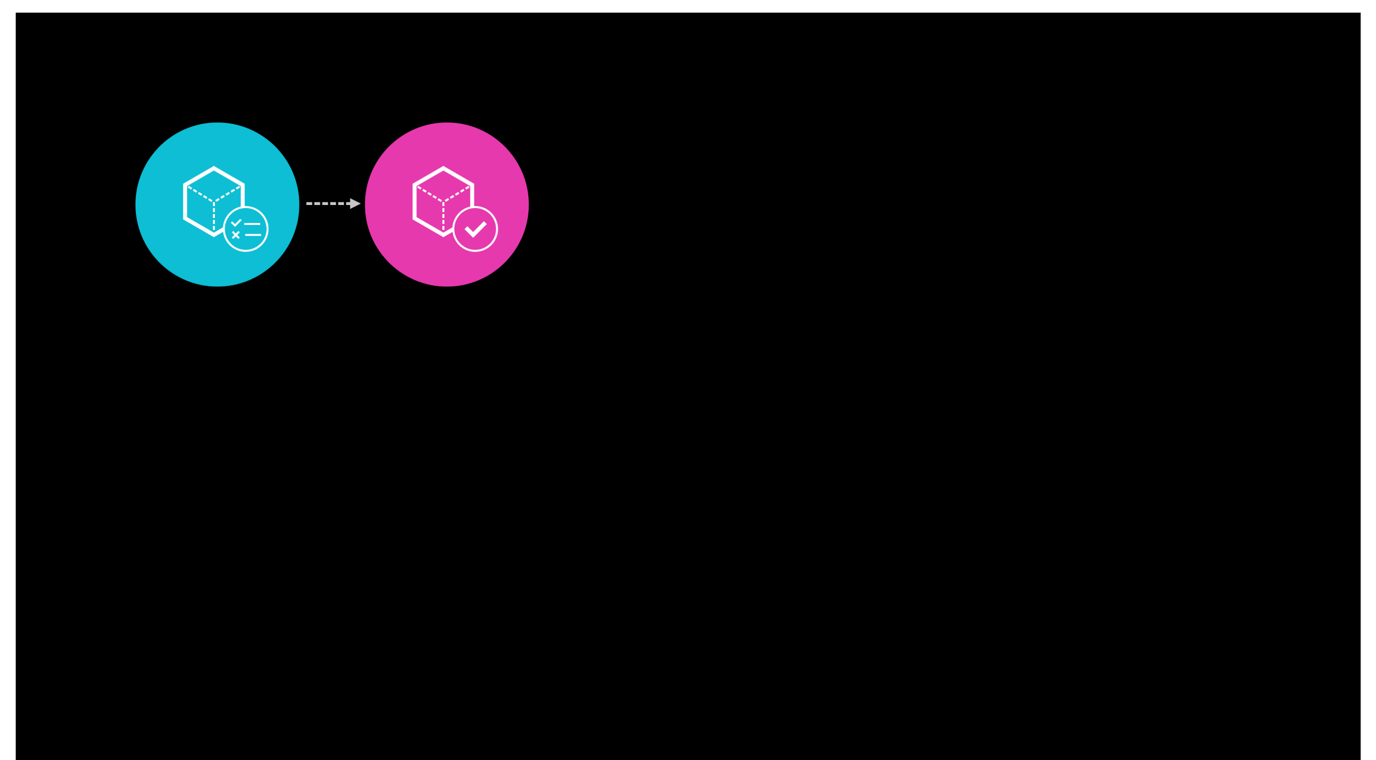
JamfComputerProfileUploader

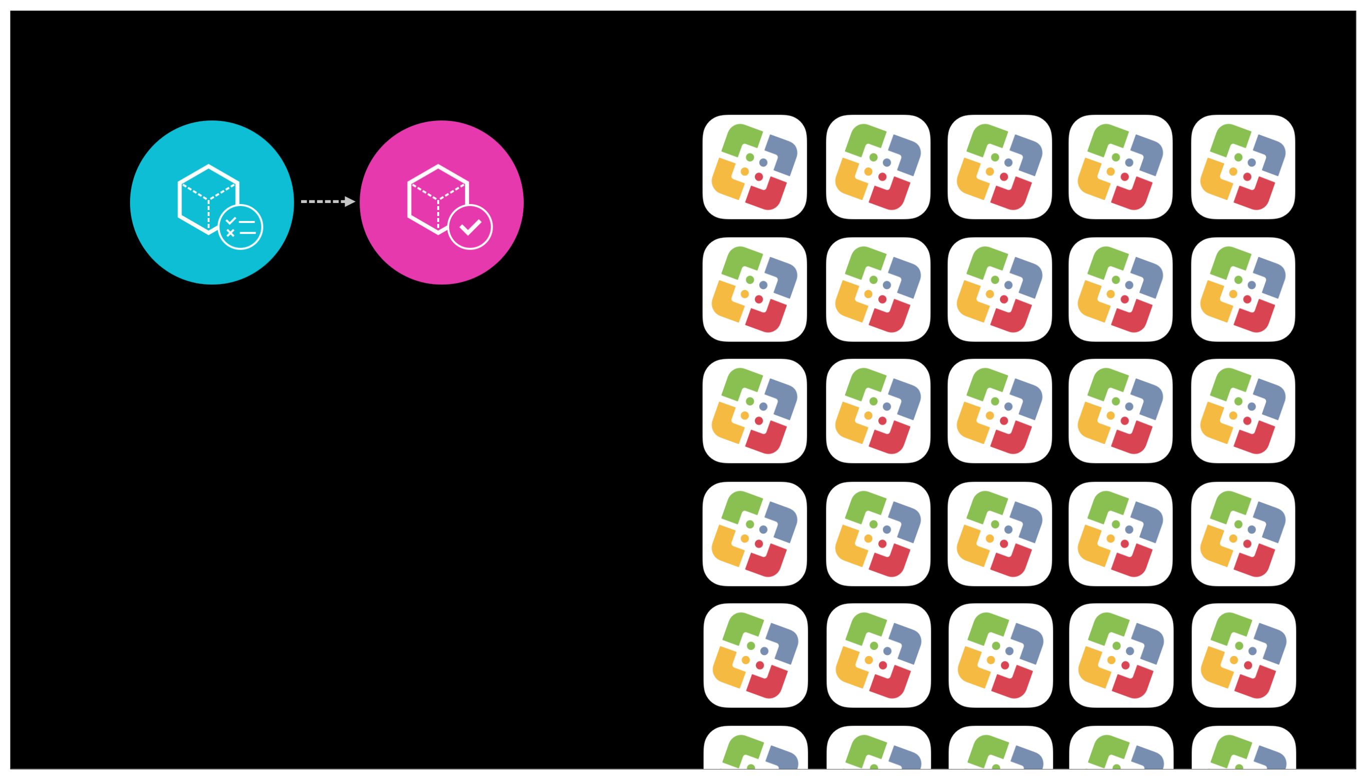


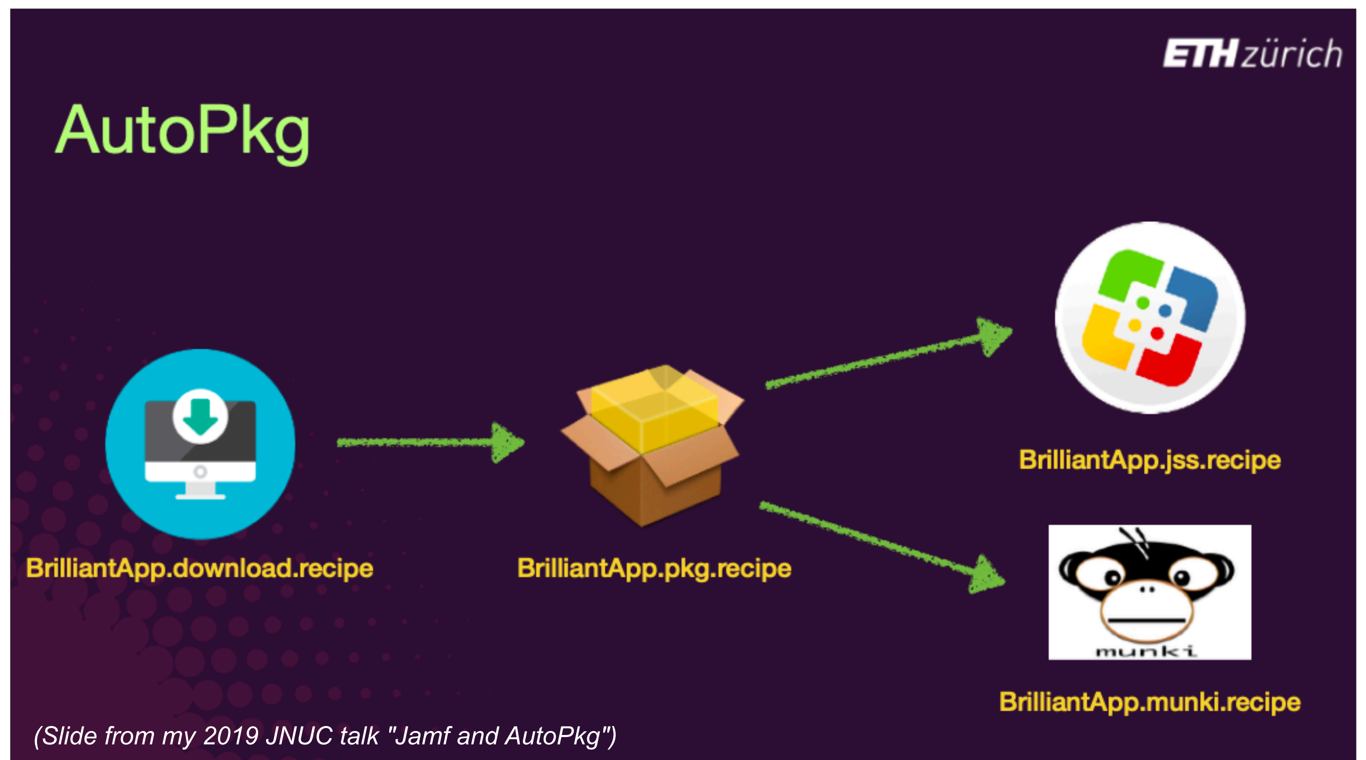
OK, so back to our automation. There's two parts to the automation that we have built that I would like to talk about today. The first is the process of promoting a software title from test to production,

🍎The second is the process of duplicating any changes across all our multiple Jamf Pro instances.

At the end, I'll mention some challenges we've encountered, and things we are still trying trying to optimise.



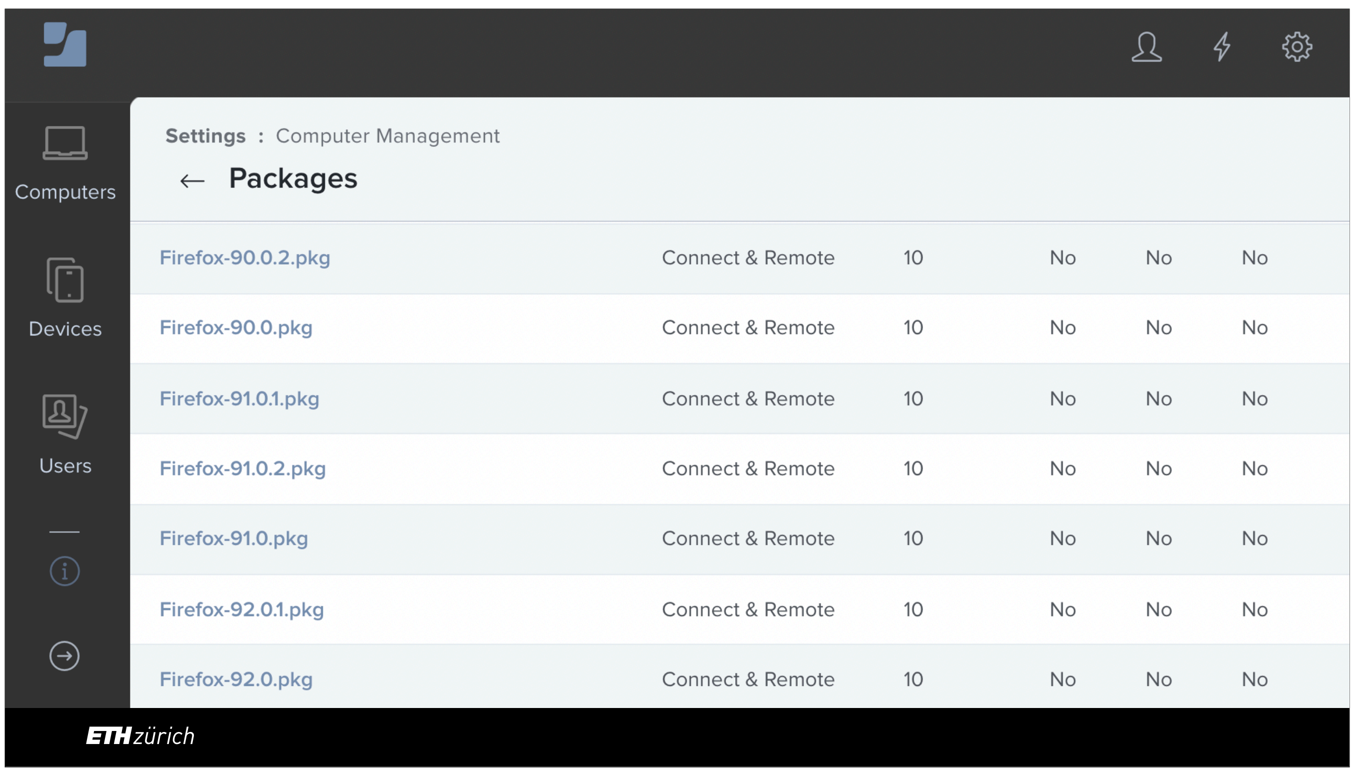




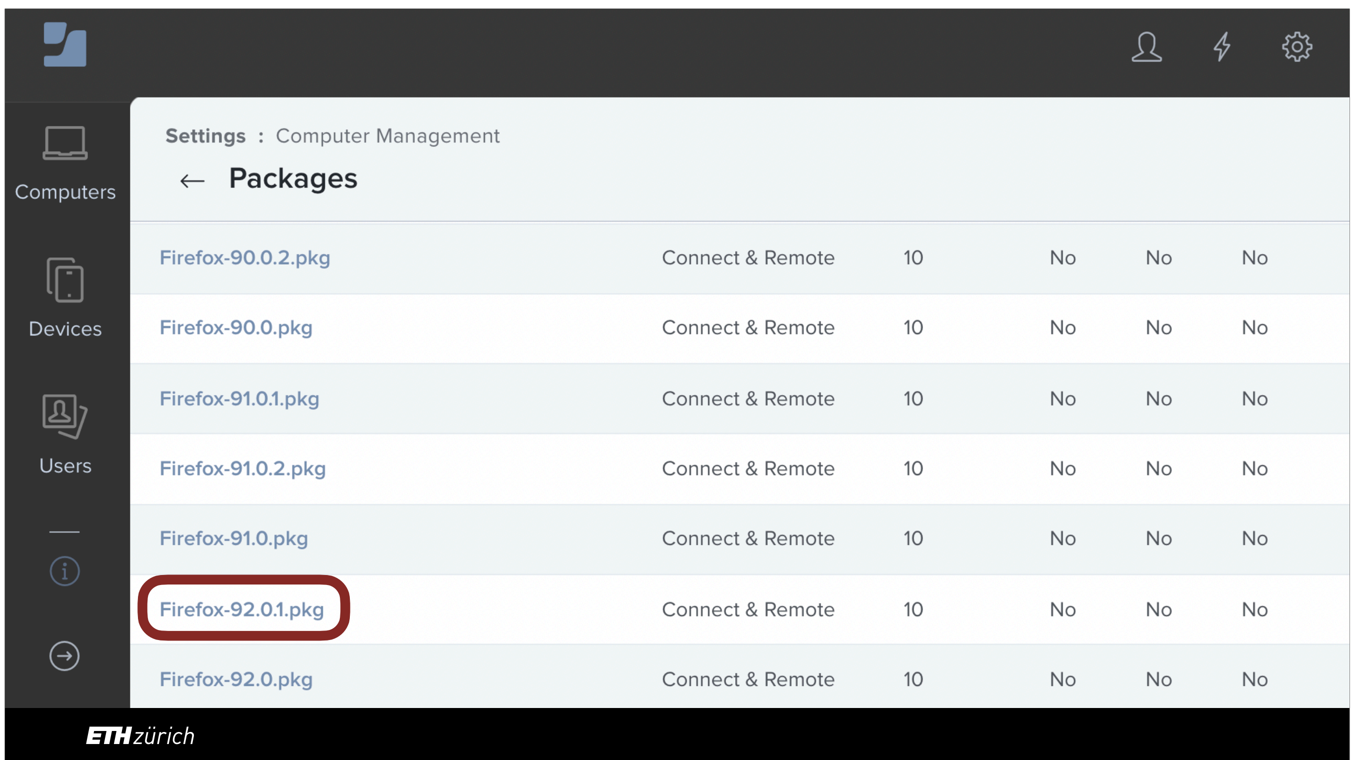
First, I'll talk about the process of promoting from test to production.

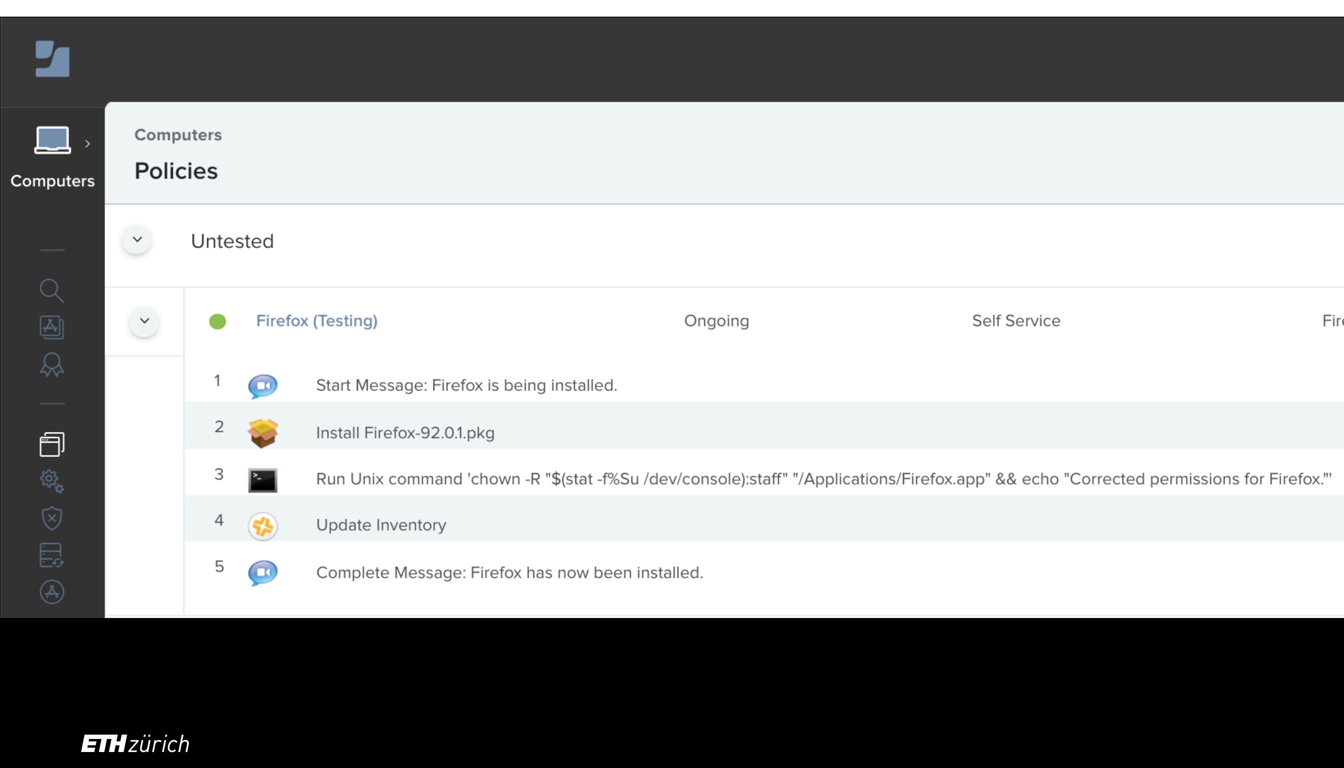
This is a slide from my 2019 JNUC presentation about using Jamf and AutoPkg together. It shows the usual workflow, where information from processors in download recipe is output and used by processors in the PKG recipe, and in turn, that information is used in a JSS recipe.

Normally, the only information that the JSS recipe needs from the parent recipes are the package path in the AutoPkg cache, the package name, and its version.

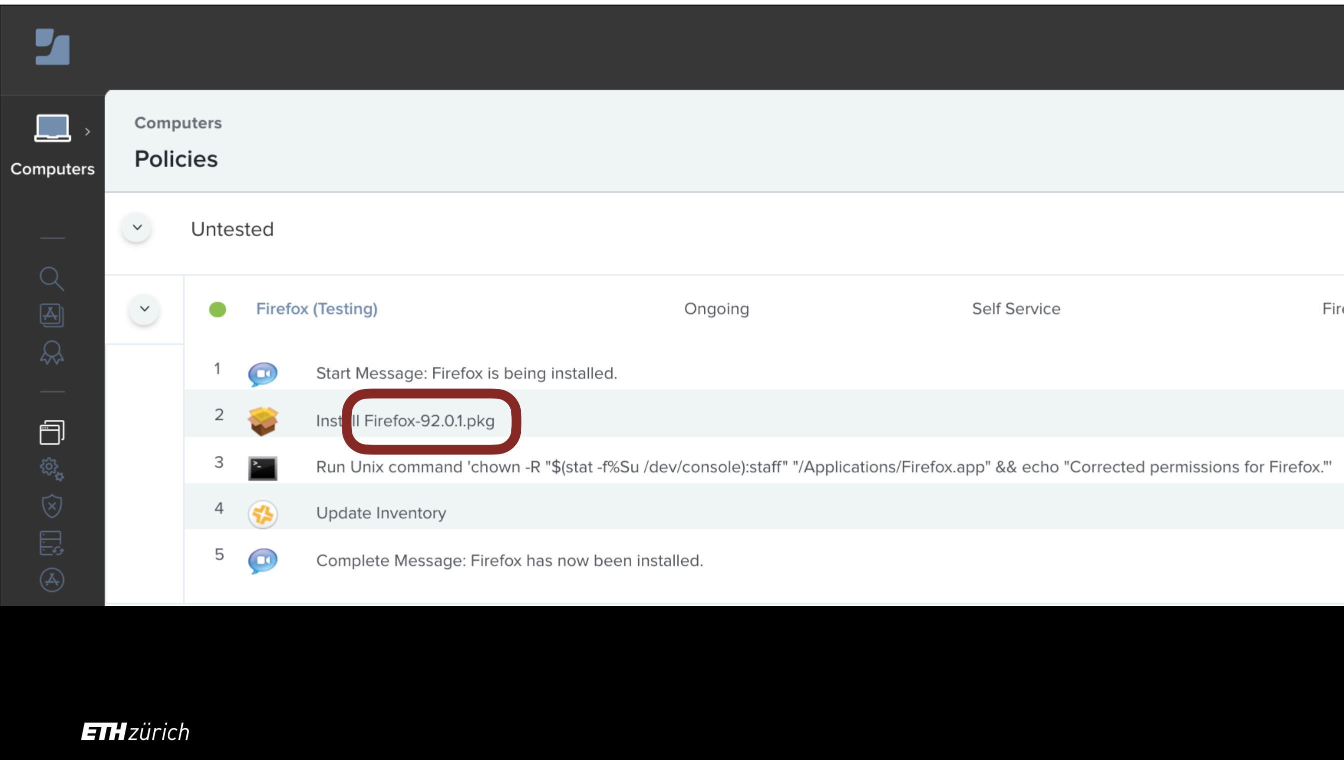


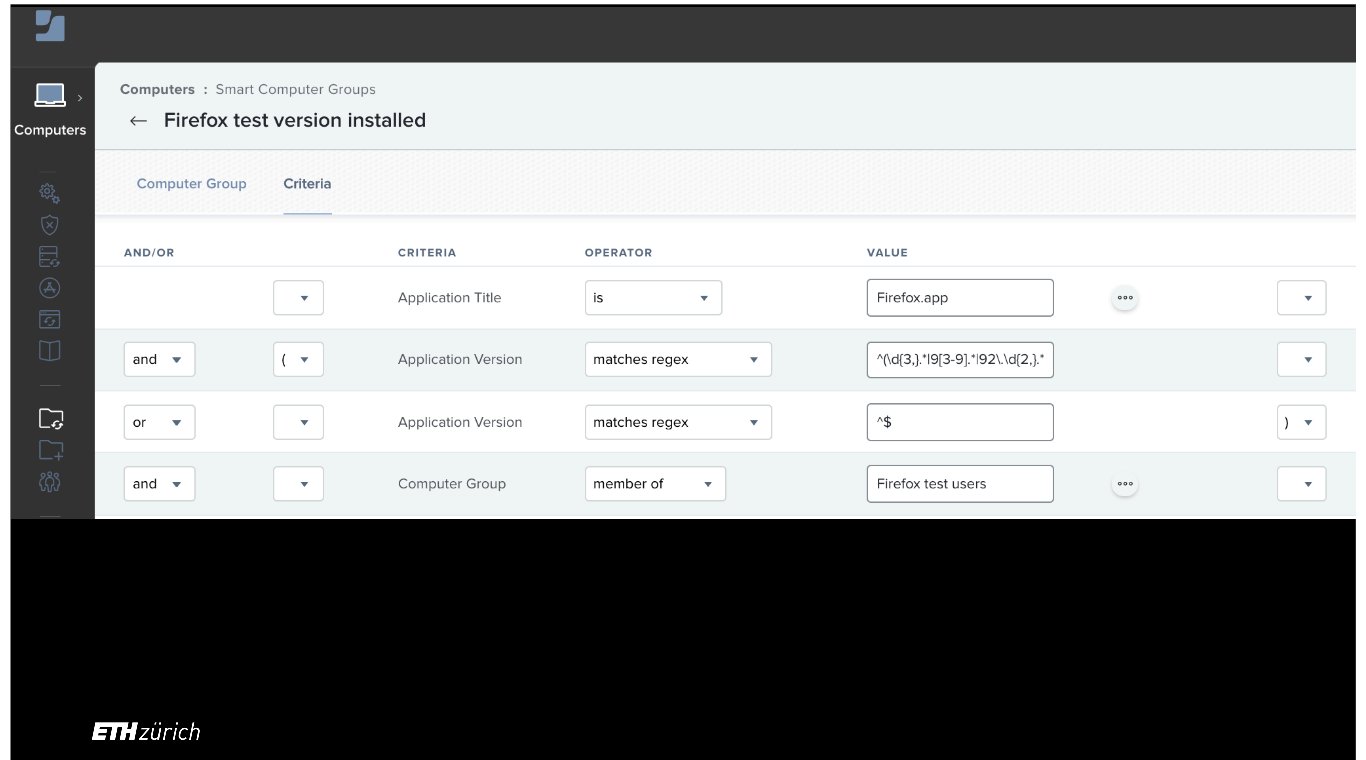
For example, 🍎 with one of our jss or jamf recipes, the package path value is used by JSSImporter or the new JamfPackageUploader processor, to upload the package into Jamf.


Computers
Devices
Users
Settings : Computer Management
Packages
Firefox-90.0.2.pkg Connect & Remote 10 No No No
Firefox-90.0.pkg Connect & Remote 10 No No No
Firefox-91.0.1.pkg Connect & Remote 10 No No No
Firefox-91.0.2.pkg Connect & Remote 10 No No No
Firefox-91.0.pkg Connect & Remote 10 No No No
Firefox-92.0.1.pkg Connect & Remote 10 No No No
Firefox-92.0.pkg Connect & Remote 10 No No No
ETHzürich



The package name is added to the new testing policy,

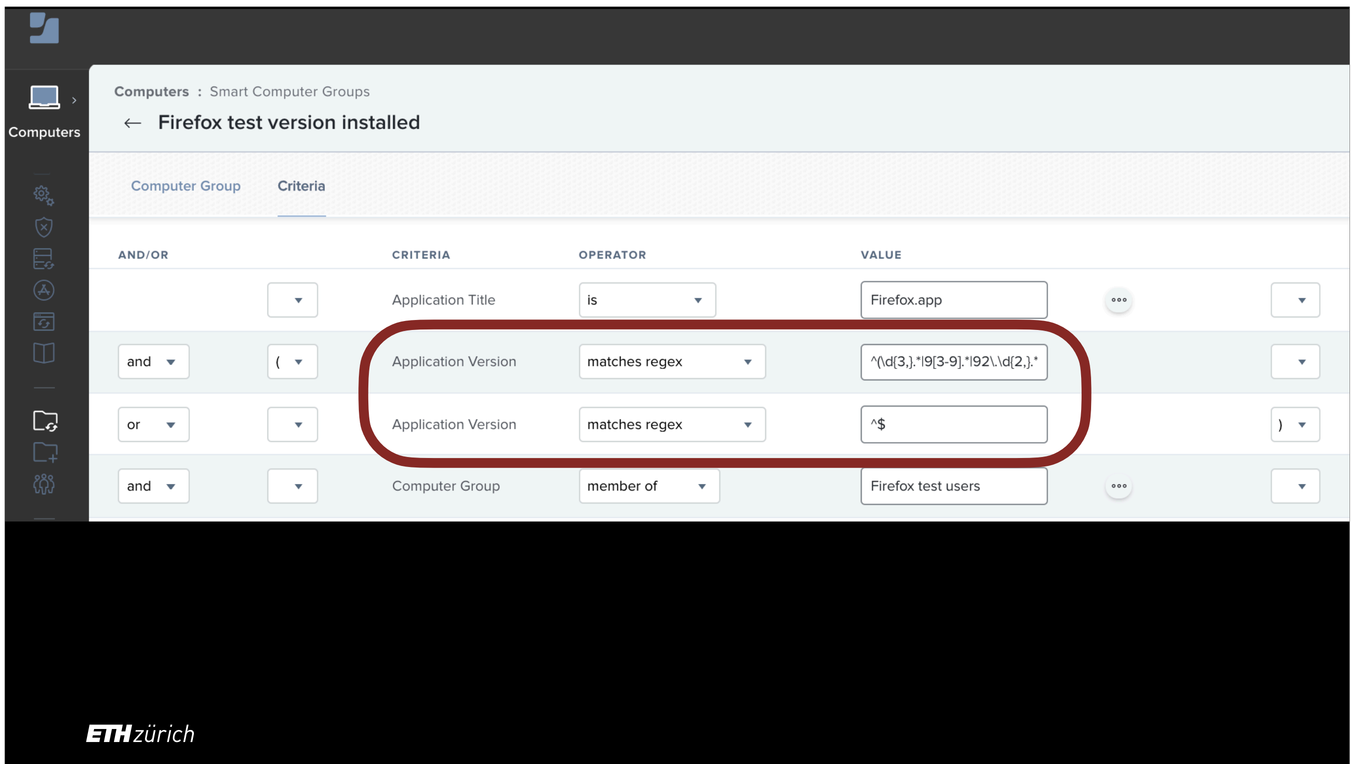

Computers
Computers
Policies
Untested
Firefox (Testing)
Ongoing
Self Service
1 Start Message: Firefox is being installed.
2 Install Firefox-92.0.1.pkg
3 Run Unix command 'chown -R "$(stat -f%Su /dev/console):staff" "/Applications/Firefox.app" && echo "Corrected permissions for Firefox."'
4 Update Inventory
5 Complete Message: Firefox has now been installed.
ETH zürich



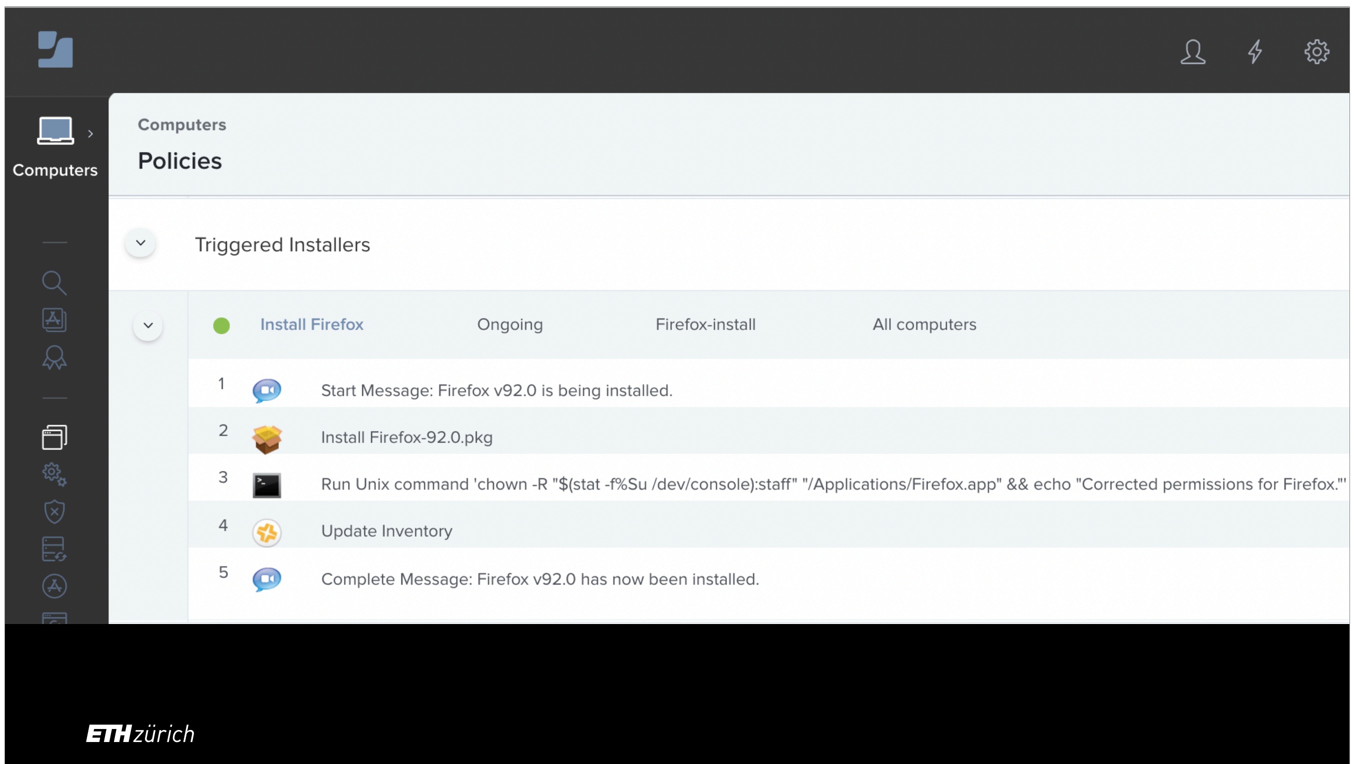
And the version string is added to a smart group, which determines if the new, untested version of the application is already installed or not.

🍎In our case we are generating a regex based on the version string, which matches this version and any conceivable newer version. This ensures package installation attempts are only scoped to computers with an older version, rather than any computer that has a different version newer or older.

If you're interested in how to go about this, checkout my presentation links for Bill Smith's CurrentVersionOrHigher bash script, or my VersionRegexGenerator AutoPkg processor.



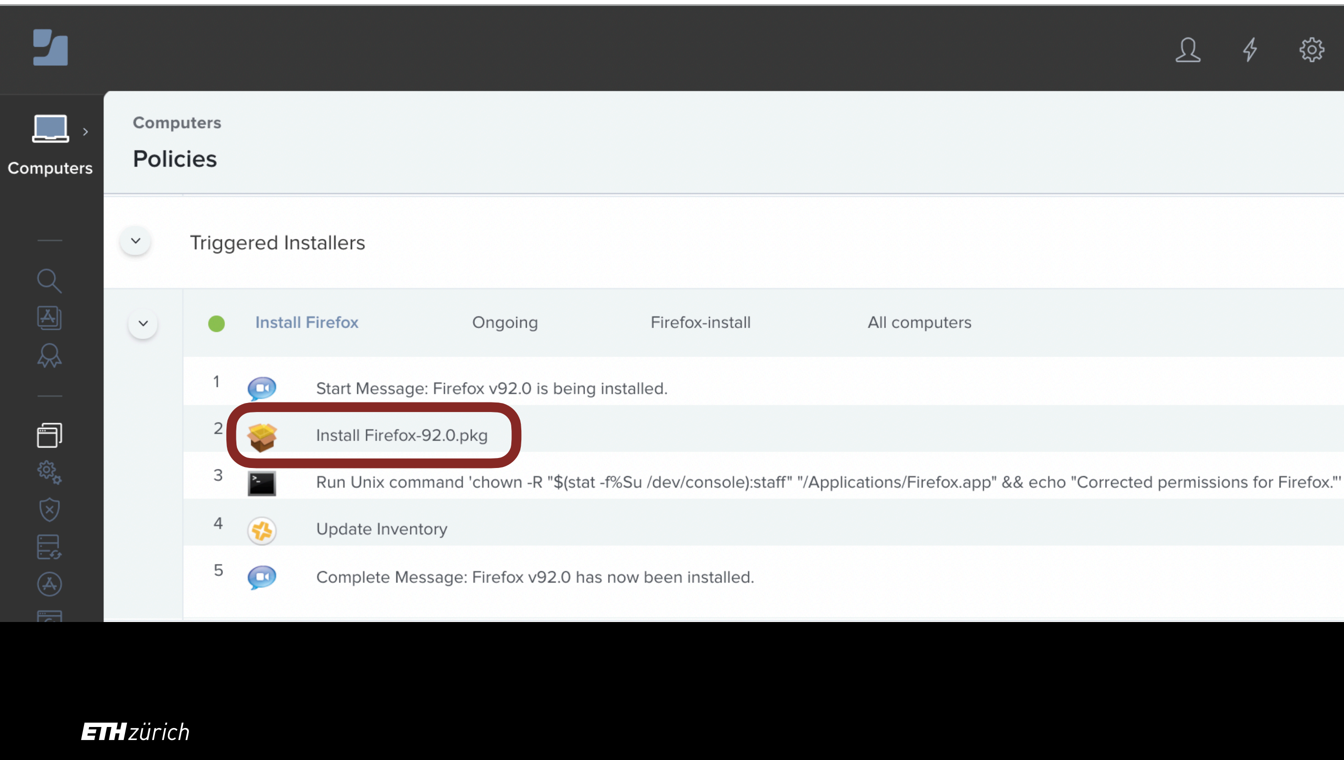
Computers
Computers : Smart Computer Groups
Firefox test version installed
Computer Group
Criteria
AND/OR
CRITERIA
OPERATOR
VALUE
Application Title
is
Firefox.app
and
(
Application Version
matches regex
^(\d{3,}.*|9[3-9].*|92\.\d{2,}.*
or
Application Version
matches regex
^$
)
and
Computer Group
member of
Firefox test users
ETH zürich

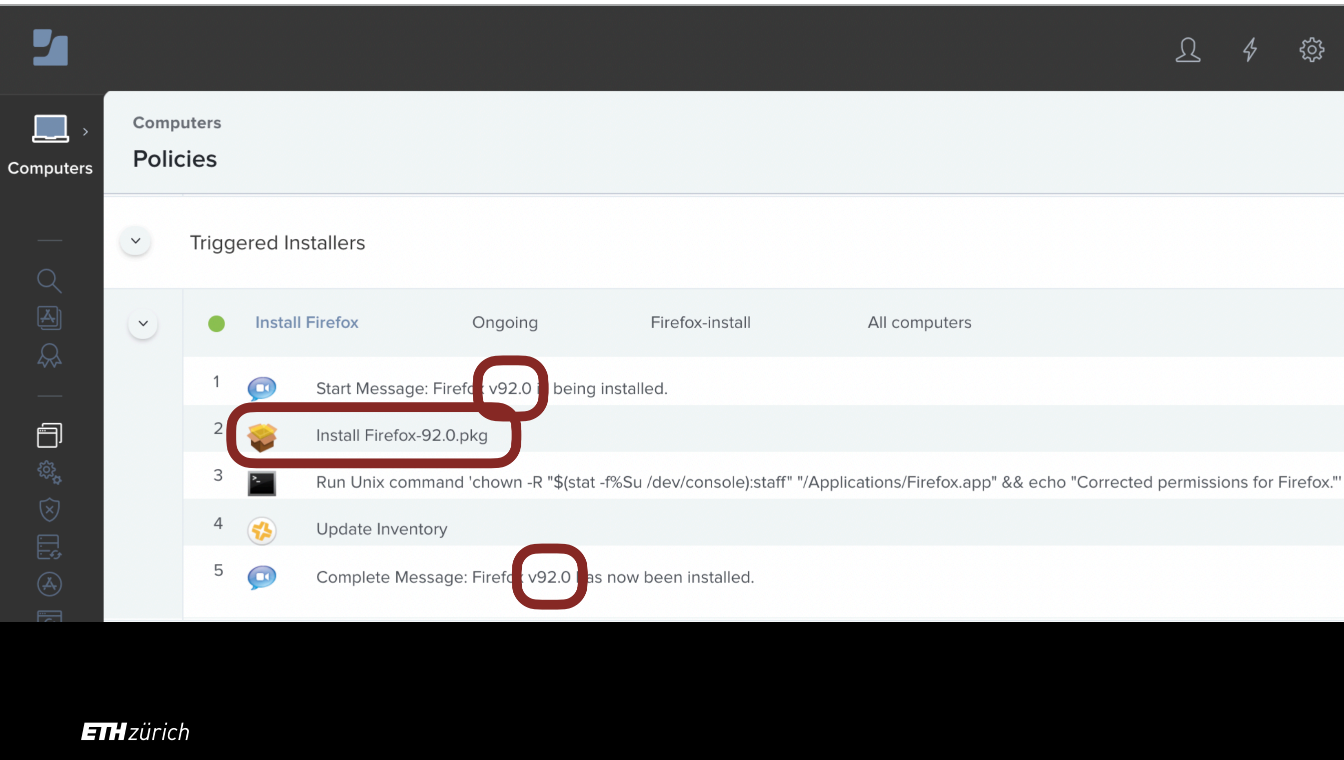


In our production policies, we also need the package name and version.

🍎 The package has to be added to the relevant policy, in our case the "Install Firefox" policy, and since we are also showing a notification to end users about which version is being installed, so 🍎we need the version string here too.

Everything else in this policy is static.


Computers
Computers
Policies
Triggered Installers
Install Firefox
Ongoing
Firefox-install
All computers
1 Start Message: Firefox v92.0 is being installed.
2 Install Firefox-92.0.pkg
3 Run Unix command 'chown -R "$(stat -f%Su /dev/console):staff" "/Applications/Firefox.app" && echo "Corrected permissions for Firefox."'
4 Update Inventory
5 Complete Message: Firefox v92.0 has now been installed.
ETH zürich


Computers
Computers
Policies
Triggered Installers
Install Firefox
Ongoing
Firefox-install
All computers
1
Start Message: Firefox v92.0 is being installed.
2
Install Firefox-92.0.pkg
3
Run Unix command 'chown -R "$(stat -f%Su /dev/console):staff" "/Applications/Firefox.app" && echo "Corrected permissions for Firefox."'
4
Update Inventory
5
Complete Message: Firefox v92.0 has now been installed.
ETH zürich

Computers : Smart Computer Groups

← Firefox current version installed

Computer Group Criteria

AND/OR | CRITERIA | OPERATOR | VALUE

Application Title | is | Firefox.app

and | (| Application Version | matches regex | ^(\d{3,}).*|9[3-9].*|92\.\d{2,

or | Application Version | matches regex | ^$ |)

and | Computer Group | member of | Firefox users

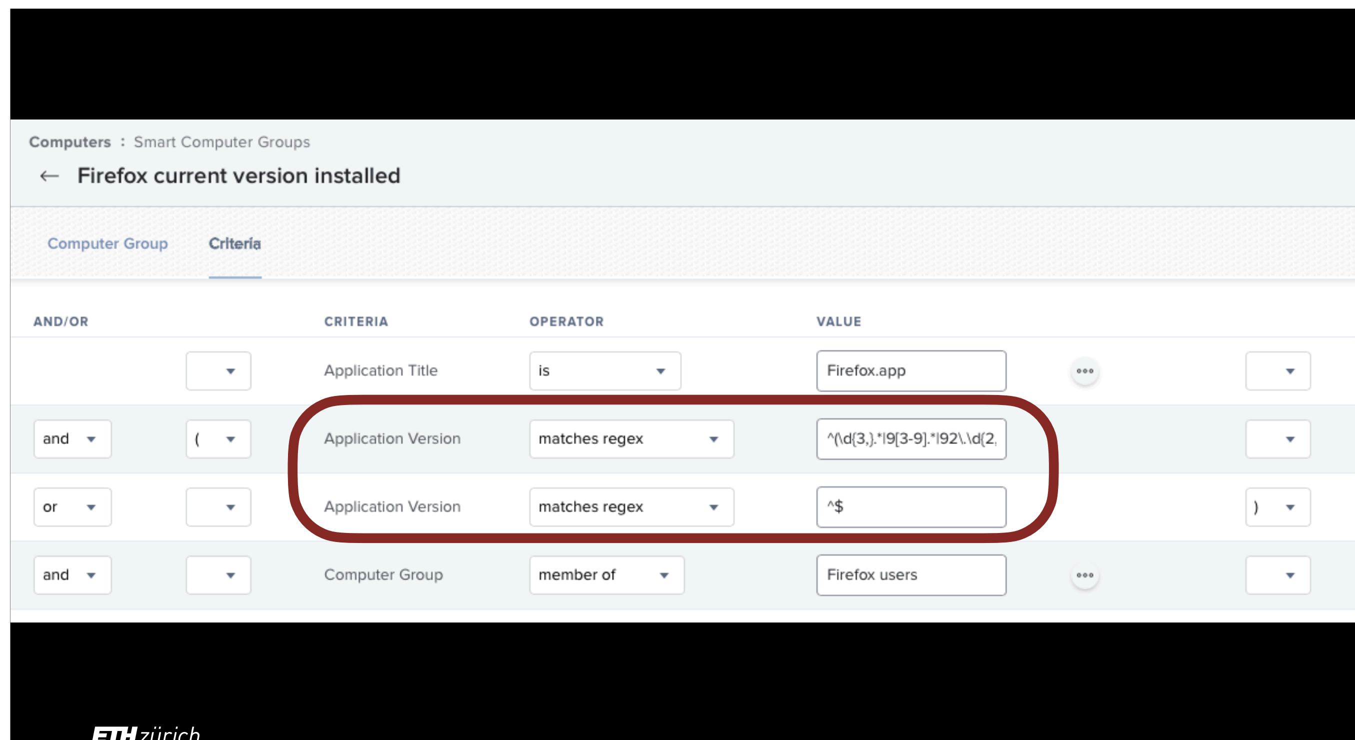
ETH zürich

And we need to put the correct version string into a smart group that determines if the current production version is already installed. 🍎 Again, here we are deriving a regex from that version string.

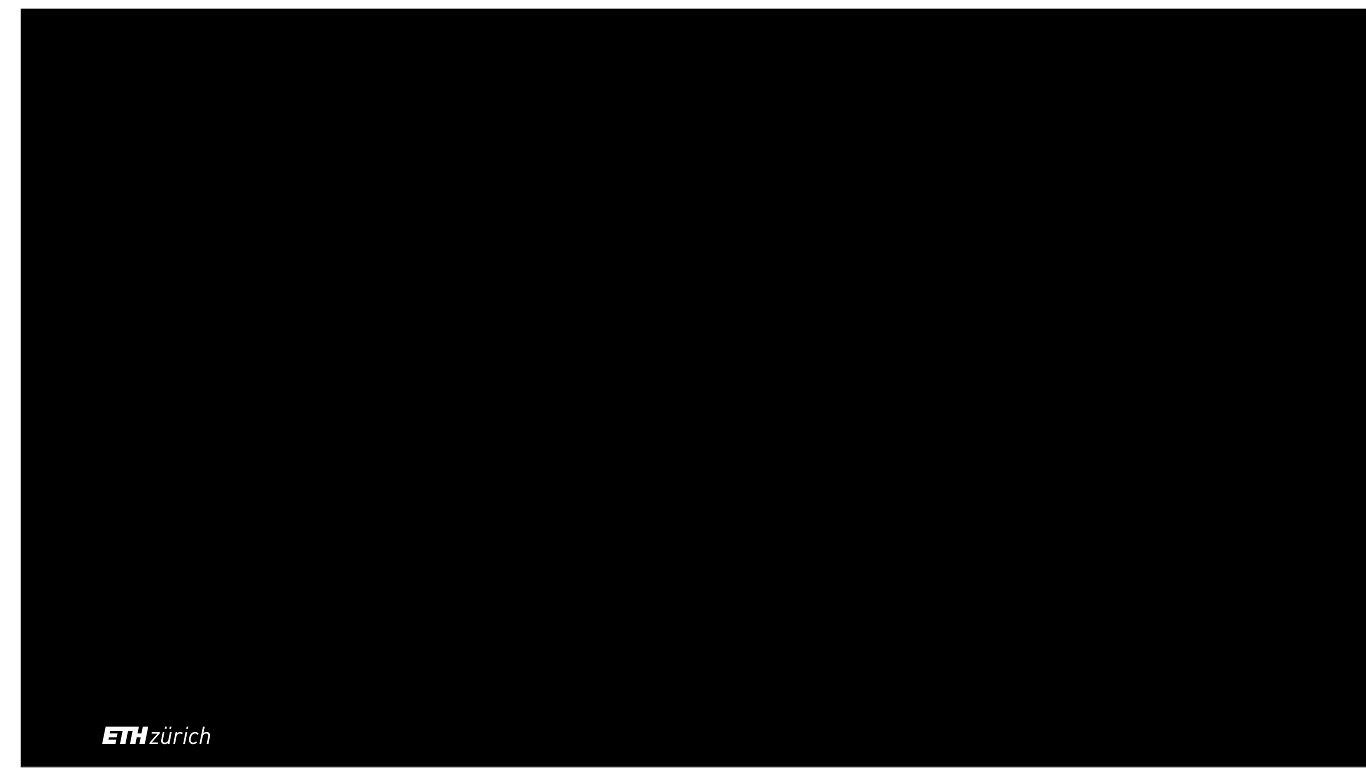
The important thing to note here, is that for production policies, we need the package name and version, but we don't need to upload the package itself. The package is already in Jamf Pro, of course, as it was already uploaded for testing.

So we don't want to go through the whole process of re-downloading the installer, packaging it up again, and uploading it to Jamf again.

But how can we get the package name and version string without running the download and PKG recipes again?



Computers : Smart Computer Groups
Firefox current version installed
Computer Group
Criteria
AND/OR
CRITERIA
OPERATOR
VALUE
Application Title
is
Firefox.app
and
(
Application Version
matches regex
^(\d{3,}.*|9[3-9].*|92\.\d{2,
or
Application Version
matches regex
^$
)
and
Computer Group
member of
Firefox users
ETH zürich

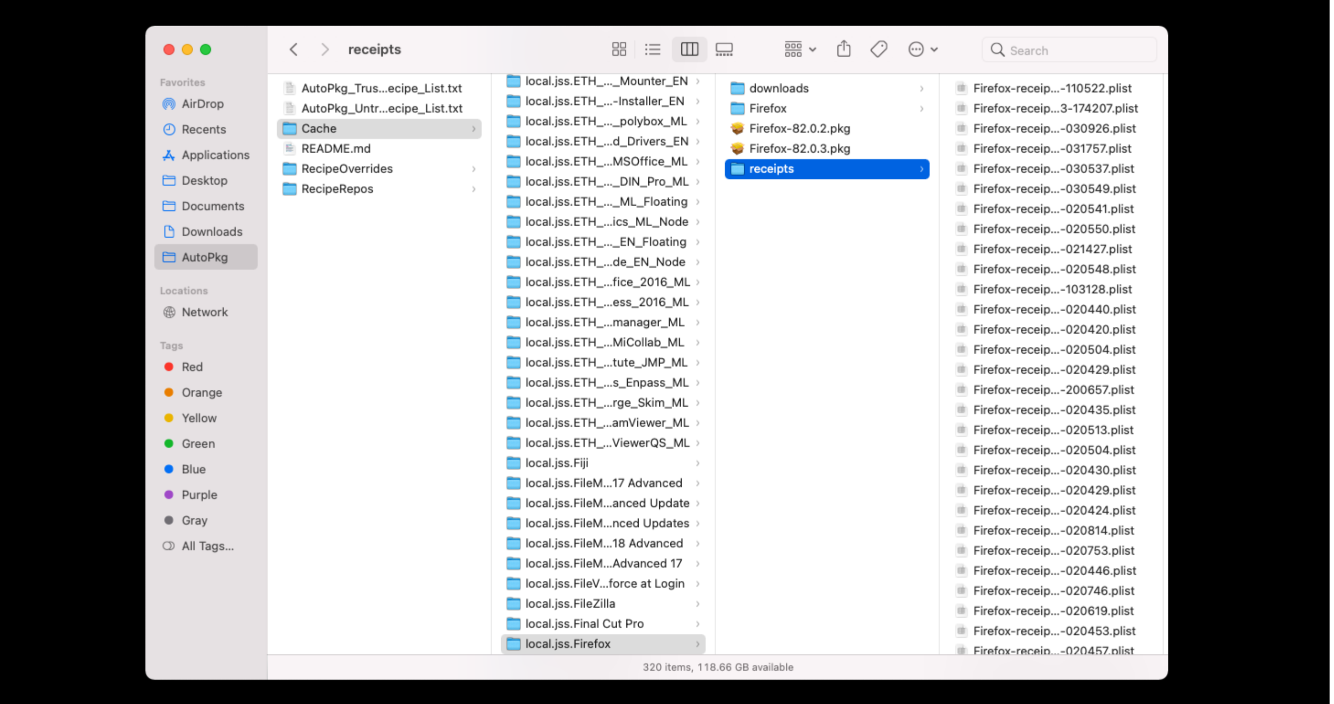


Well, I did figure out a way. This is one of the processes that I've changed over time, but I'll explain the two different solutions that I've used, because one or other might suit others' workflows better.

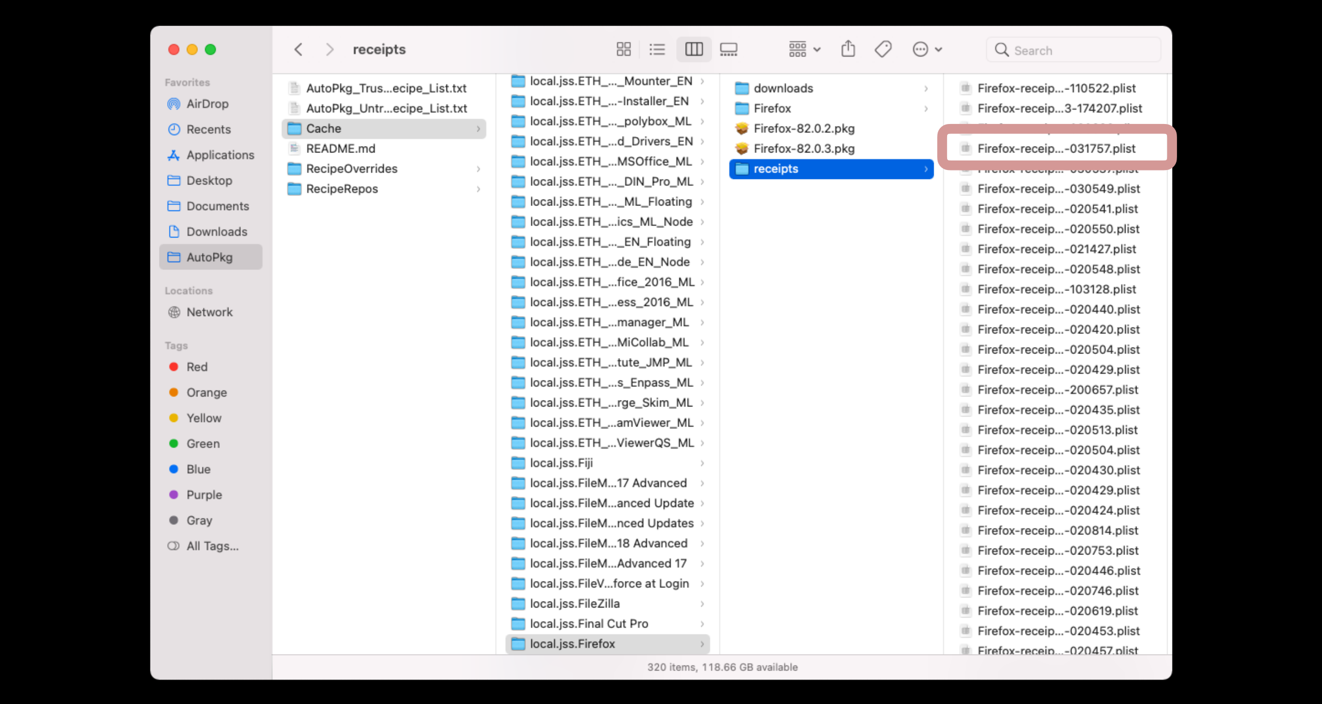
🍎This is the folder structure of the AutoPkg cache. We're looking here at the cache of the Firefox.jss recipe.
🍎Each recipe run generates a receipt, which is named after the recipe's identifier, with a date string added.

🍎And if we take a quick look inside one of them, you can see that the receipt is a plist file which contains comprehensive output of the recipe, including any output values from each processor.

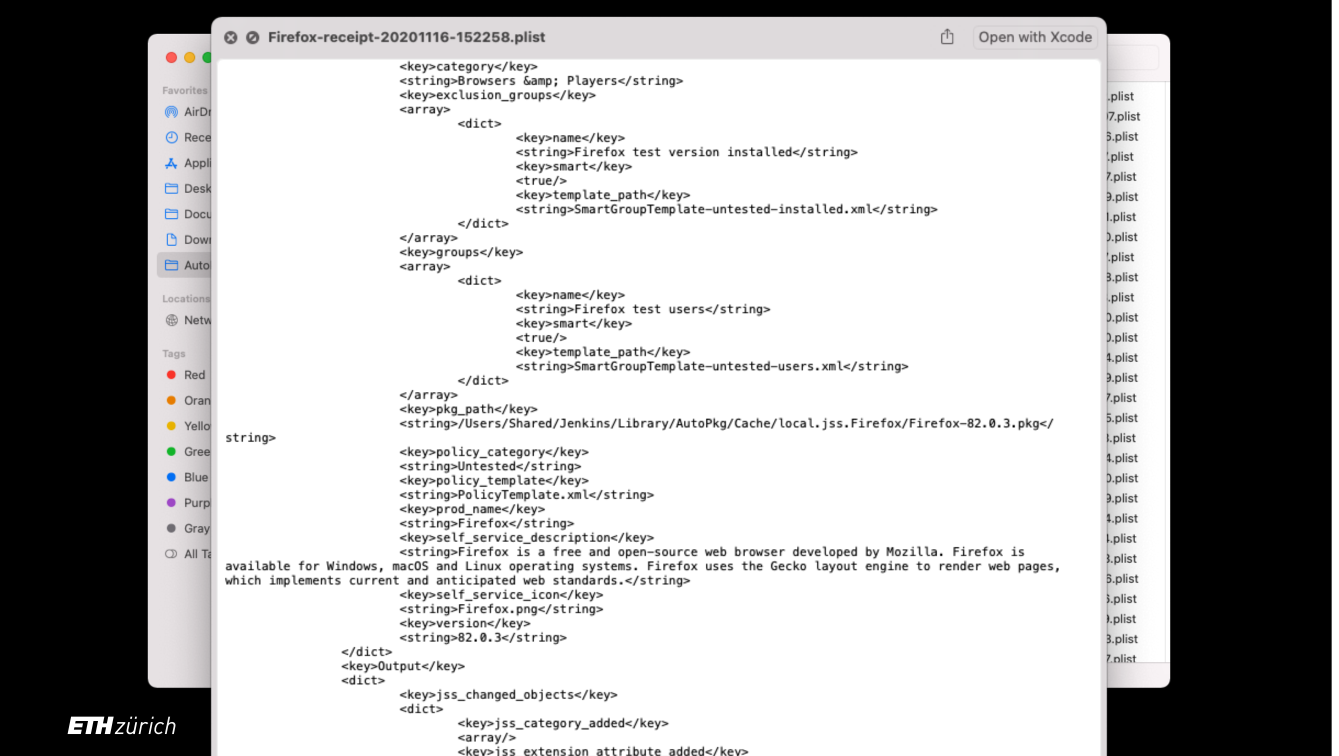
🍎We can use this receipt's contents to get the name and version of the package, plus any other information that might be useful for the policies we want to write for promoting to production.

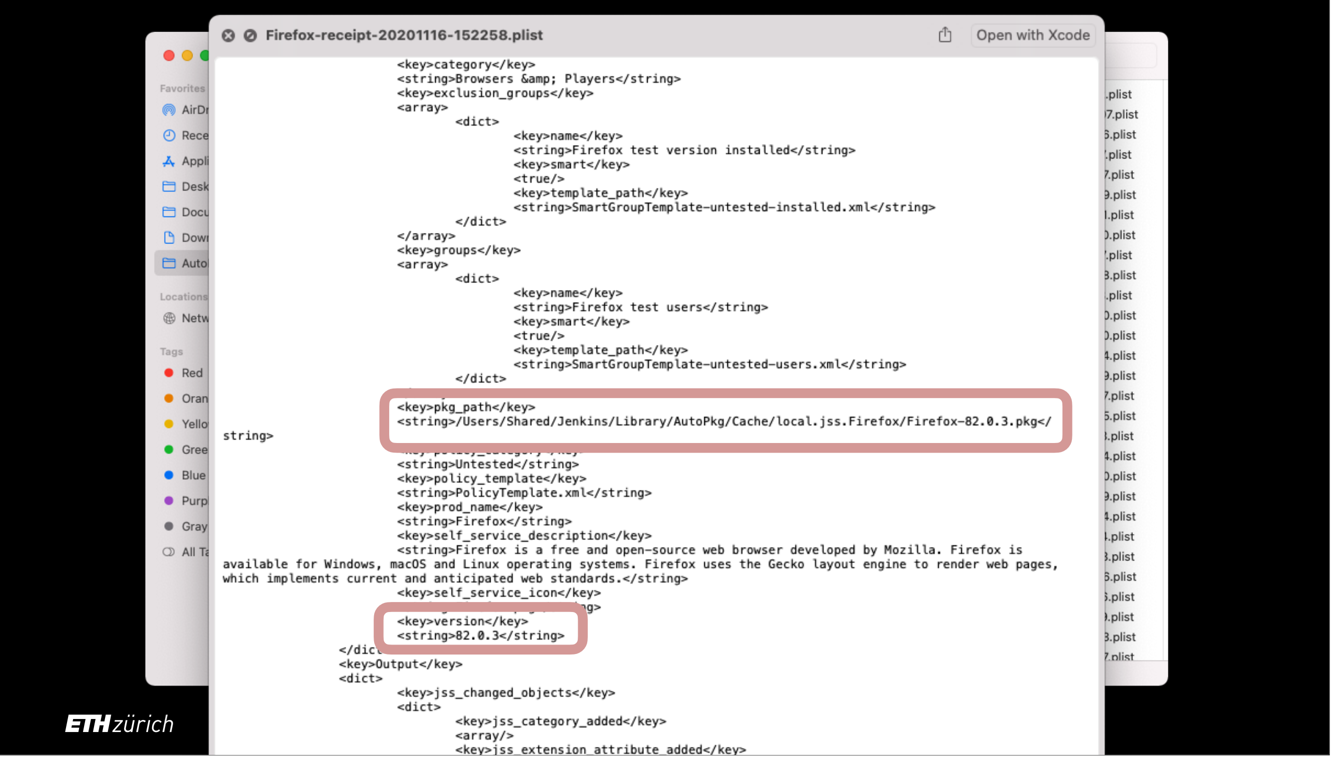


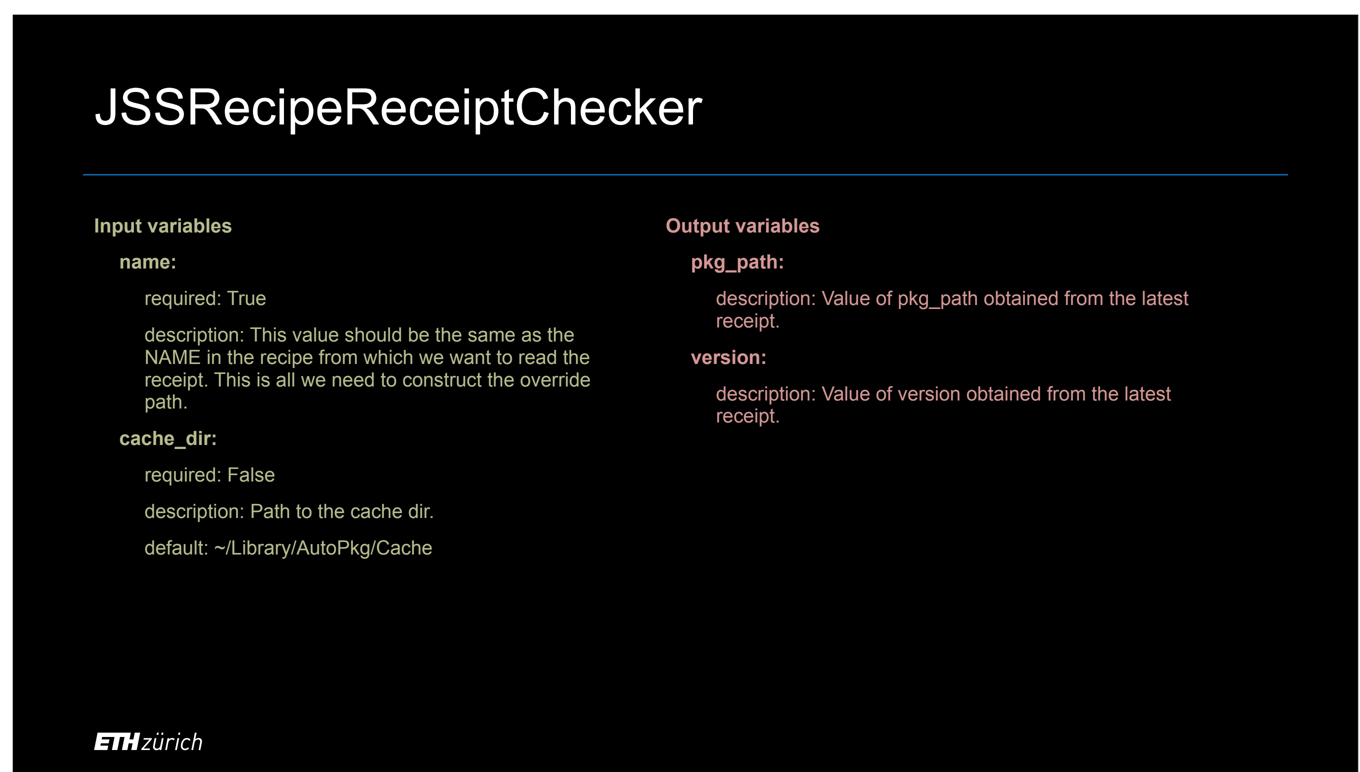
ETH zürich



ETH zürich


Firefox-receipt-20201116-152258.plist
Open with Xcode
```
			<key>category</key>
			<string>Browsers &amp; Players</string>
			<key>exclusion_groups</key>
			<array>
				<dict>
					<key>name</key>
					<string>Firefox test version installed</string>
					<key>smart</key>
					<true/>
					<key>template_path</key>
					<string>SmartGroupTemplate-untested-installed.xml</string>
				</dict>
			</array>
			<key>groups</key>
			<array>
				<dict>
					<key>name</key>
					<string>Firefox test users</string>
					<key>smart</key>
					<true/>
					<key>template_path</key>
					<string>SmartGroupTemplate-untested-users.xml</string>
				</dict>
			</array>
			<key>pkg_path</key>
			<string>/Users/Shared/Jenkins/Library/AutoPkg/Cache/local.jss.Firefox/Firefox-82.0.3.pkg</
string>
			<key>policy_category</key>
			<string>Untested</string>
			<key>policy_template</key>
			<string>PolicyTemplate.xml</string>
			<key>prod_name</key>
			<string>Firefox</string>
			<key>self_service_description</key>
			<string>Firefox is a free and open-source web browser developed by Mozilla. Firefox is
available for Windows, macOS and Linux operating systems. Firefox uses the Gecko layout engine to render web pages,
which implements current and anticipated web standards.</string>
			<key>self_service_icon</key>
			<string>Firefox.png</string>
			<key>version</key>
			<string>82.0.3</string>
		</dict>
		<key>Output</key>
		<dict>
			<key>jss_changed_objects</key>
			<dict>
				<key>jss_category_added</key>
				<array/>
				<key>jss_extension_attribute_added</key>
```
Favorites
Red
Blue
Gray
ETH zürich


Firefox-receipt-20201116-152258.plist
Open with Xcode
<key>category</key>
<string>Browsers & Players</string>
<key>exclusion_groups</key>
<array>
<dict>
<key>name</key>
<string>Firefox test version installed</string>
<key>smart</key>
<true/>
<key>template_path</key>
<string>SmartGroupTemplate-untested-installed.xml</string>
</dict>
</array>
<key>groups</key>
<array>
<dict>
<key>name</key>
<string>Firefox test users</string>
<key>smart</key>
<true/>
<key>template_path</key>
<string>SmartGroupTemplate-untested-users.xml</string>
</dict>
<key>pkg_path</key>
<string>/Users/Shared/Jenkins/Library/AutoPkg/Cache/local.jss.Firefox/Firefox-82.0.3.pkg</string>
<string>Untested</string>
<key>policy_template</key>
<string>PolicyTemplate.xml</string>
<key>prod_name</key>
<string>Firefox</string>
<key>self_service_description</key>
<string>Firefox is a free and open-source web browser developed by Mozilla. Firefox is available for Windows, macOS and Linux operating systems. Firefox uses the Gecko layout engine to render web pages, which implements current and anticipated web standards.</string>
<key>self_service_icon</key>
<key>version</key>
<string>82.0.3</string>
<key>Output</key>
<dict>
<key>jss_changed_objects</key>
<dict>
<key>jss_category_added</key>
<array/>
<key>jss_extension_attribute_added</key>
ETH zürich



So, for my first attempt, I decided to write a shared processor which finds the latest receipt by date from the Cache folder, and then parses the receipt for the package name and version.

! Atom.jss-prod.recipe.yaml U ×

id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > ! Atom.jss-prod.recipe.yaml >

```yaml
1   Description: Creates all production policies for a particular title.
2   Identifier: ch.ethz.id.jss-prod.Atom
3   MinimumVersion: "2.3"
4
5 > Input: ⋯
50
51  Process:
52    - Processor: ch.ethz.autopkg.commonprocessors/JSSRecipeReceiptChecker
53      Arguments:
54        name: "%NAME%"
55
56    - Processor: ch.ethz.autopkg.commonprocessors/CheckSharepointToStage
57
58    - Processor: StopProcessingIf
59      Arguments:
60        predicate: "%PROD_PREDICATE%"
61
62    - Processor: JSSImporter
63 >    Arguments: ⋯
83      Comment: Trigger-only policy
84
```

With this approach, a production jss recipe 🍎doesn't need a parent recipe.

🍎The first process in the recipe is the JSSRecipeReceiptChecker processor. This processor outputs the package name and version, and these values are used in the JSSImporter processors below.

! Atom.jss-prod.recipe.yaml U ×

id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > ! Atom.jss-prod.recipe.yaml >

```yaml
1   Description: Creates all production policies for a particular title.
2   Identifier: ch.ethz.id.jss-prod.Atom
3   MinimumVersion: "2.3"
4
5 > Input: ...
50
51  Process:
52    - Processor: ch.ethz.autopkg.commonprocessors/JSSRecipeReceiptChecker
53      Arguments:
54        name: "%NAME%"
55
56    - Processor: ch.ethz.autopkg.commonprocessors/CheckSharepointToStage
57
58    - Processor: StopProcessingIf
59      Arguments:
60        predicate: "%PROD_PREDICATE%"
61
62    - Processor: JSSImporter
63 >    Arguments: ...
83      Comment: Trigger-only policy
84
```

! Atom.jss-prod.recipe.yaml U ×

id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > ! Atom.jss-prod.recipe.yaml >

```yaml
1   Description: Creates all production policies for a particular title.
2   Identifier: ch.ethz.id.jss-prod.Atom
3   MinimumVersion: "2.3"
4
5 > Input: …
50
51  Process:
52    - Processor: ch.ethz.autopkg.commonprocessors/JSSRecipeReceiptChecker
53      Arguments:
54        name: "%NAME%"
55
56    - Processor: ch.ethz.autopkg.commonprocessors/CheckSharepointToStage
57
58    - Processor: StopProcessingIf
59      Arguments:
60        predicate: "%PROD_PREDICATE%"
61
62    - Processor: JSSImporter
63 >    Arguments: …
83      Comment: Trigger-only policy
84
```

The challenge for this approach is that every recipe run generates a receipt, even if no package is generated or uploaded.
If no package is generated, the receipt does not contain any information about the package.
So, the JSSRecipeReceiptChecker processor has to iterate back in time through all previous receipts until it eventually finds the last run that created a package.
If no new version appears for a long time, this process takes longer and longer as it has to read more files until it gets the values required.
And it won't work if you clear out your cache all the time.
Nonetheless, we used this pretty successfully with our prod jss recipes for a couple of years.

As I transitioned to using JamfUploader, I decided to take another look at this workflow, and came up with an alternative.

```
JSSRecipeReceiptChecker: Checking: Unarchiver
JSSRecipeReceiptChecker: Checking: CodeSignatureVerifier
JSSRecipeReceiptChecker: Checking: StopProcessingIf
JSSRecipeReceiptChecker: No version found in receipt
JSSRecipeReceiptChecker: Receipt: /Users/Shared/Jenkins/Library/AutoPkg/Cache/local.jss.Atom/receipts/Atom-
receipt-20210830-160611.plist
JSSRecipeReceiptChecker: Checking: URLDownloader
JSSRecipeReceiptChecker: Checking: EndOfCheckPhase
JSSRecipeReceiptChecker: Checking: Unarchiver
JSSRecipeReceiptChecker: Checking: CodeSignatureVerifier
JSSRecipeReceiptChecker: Checking: StopProcessingIf
JSSRecipeReceiptChecker: No version found in receipt
JSSRecipeReceiptChecker: Receipt: /Users/Shared/Jenkins/Library/AutoPkg/Cache/local.jss.Atom/receipts/Atom-
receipt-20210827-151058.plist
JSSRecipeReceiptChecker: Checking: URLDownloader
JSSRecipeReceiptChecker: Checking: EndOfCheckPhase
JSSRecipeReceiptChecker: Checking: Unarchiver
JSSRecipeReceiptChecker: Checking: CodeSignatureVerifier
JSSRecipeReceiptChecker: Checking: StopProcessingIf
JSSRecipeReceiptChecker: Checking: PkgRootCreator
JSSRecipeReceiptChecker: Checking: Unarchiver
JSSRecipeReceiptChecker: Checking: Versioner
JSSRecipeReceiptChecker: Checking: PkgCreator
JSSRecipeReceiptChecker: Checking: JSSImporter
JSSRecipeReceiptChecker: Version: 1.60
JSSRecipeReceiptChecker: Package: /Users/Shared/Jenkins/Library/AutoPkg/Cache/local.jss.Atom/Atom-4.67.pkg
JSSRecipeReceiptChecker: Category: Productivity
JSSRecipeReceiptChecker: Self Service Description: Atom is a text and code editor produced by GitHub.
```

LastRecipeRunResult

Input variables

- **RECIPE_CACHE_DIR:**
  - required: True (assumed from AutoPkg)
  - description: AutoPkg Cache directory.
- **output_file_path:**
  - required: False
  - description: Full path name to write JSON file of results to.
  - default: RECIPE_CACHE_DIR
- **output_file_name:**
  - required: False
  - description: Name of output file.
  - default: latest_version.json
- **url**:

Output variables

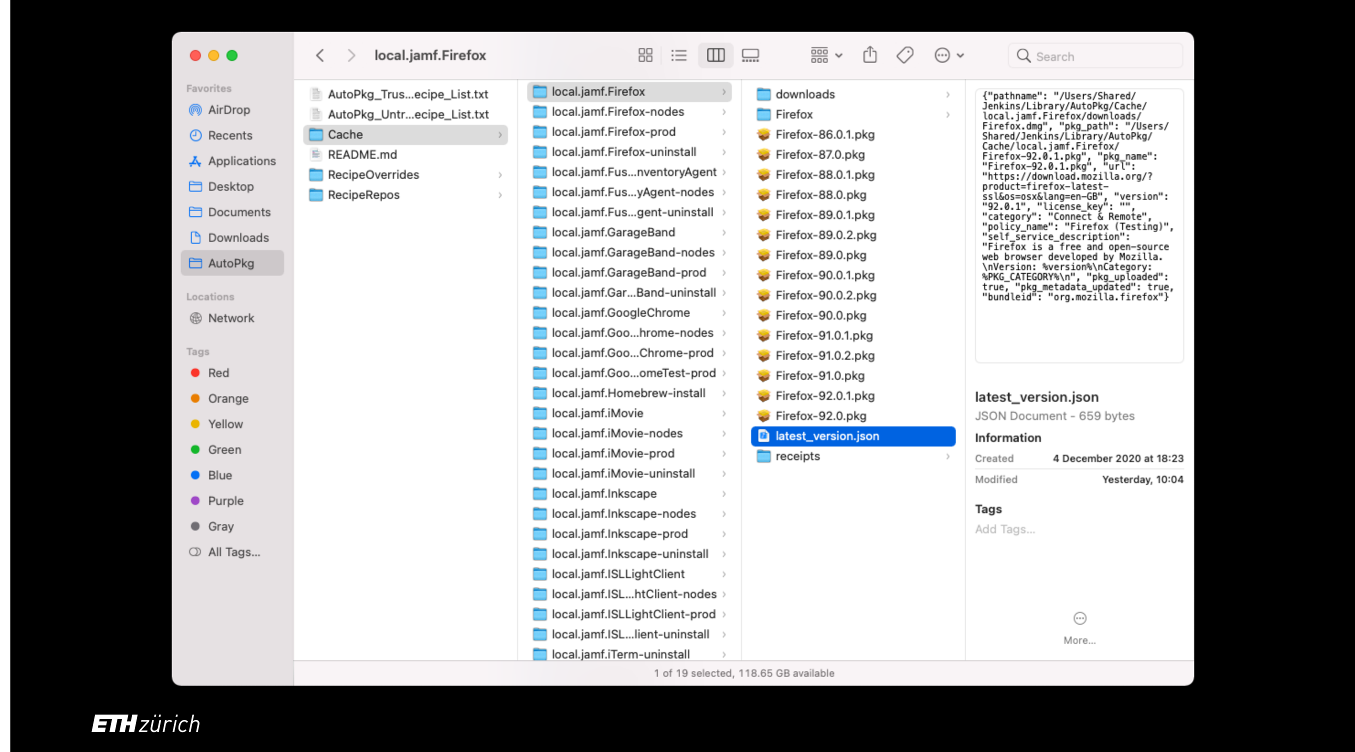
- **url:**
  - description: The value of url from the recipe run.
- **pkg_path:**
  - description: The value of pkg_path from the recipe run.
- **pathname:**
  - description: The value of pathname from the recipe run.
- **version:**
  - description: The value of version from the recipe run.
- **PKG_CATEGORY:**
  - description: The value of PKG_CATEGORY from the recipe run.
- **SELFSERVICE_DESCRIPTION:**
  - description: The value of

ETH zürich

This second method involves two processors.

The first processor is a post-processor called LastRecipeRunResult, and is designed to run at the end of a jamf recipe, the one that obtains new package installers for testing. I'll call these recipes "testing jamf recipes" from now on.

This processor takes the package name and version, plus other values that are useful to us, and writes them to a JSON file in the Recipe Cache directory.



Because these values are always written to the same file, we don't have to search through multiple files back in time to find the latest values. And it's easier to parse, because we are only collecting the values that are useful to us.

Note that this processor is agnostic to Jamf, I know of at least one organisation using this as a post-processor for PKG recipes, to integrate with their own completely different deployment system.

LastRecipeRunChecker

Input variables

- **recipeoverride_identifier**:
  - required: True
  - description: The identifier of the recipe from which the information is required.
- **cache_dir:**
  - required: False
  - description: Path to the cache dir.
  - default: ~/Library/AutoPkg/Cache
- **info_file:**
  - required: False
  - description: Name of input file.
  - default: latest_version.json

Output variables

- **url:**
  - description: The value of url from the recipe run.
- **pkg_path:**
  - description: The value of pkg_path from the recipe run.
- **pathname:**
  - description: The value of pathname from the recipe run.
- **version:**
  - description: The value of version from the recipe run.
- **PKG_CATEGORY:**
  - description: The value of PKG_CATEGORY from the recipe run.
- **SELFSERVICE_DESCRIPTION:**
  - description: The value of

ETH zürich

The second processor is called LastRecipeRunChecker, and this reads the values from the JSON file, so can be run as a pre-processor.

g-mac-recipes-yaml • Jamf_Staging_Recipes ! Firefox-prod.jamf.recipe.yaml id-mac-autopkg-recipes-yaml • _Temp_Presentation_Recipes

id-mac-autopkg-recipes-yaml > _Temp_Presentation_Recipes > ! Firefox-prod.jamf.recipe.yaml > ...

```
1   Description: Uploads the pkg to the JSS, and creates a Self-Service Policy available to members of a
    Testing group.
2   Identifier: com.github.eth-its-recipes.jamf.Firefox-prod
3   MinimumVersion: "2.3"
4
5   Input:
6     NAME: Firefox
7     UNTESTED_RECIPE_IDENTIFIER: local.jamf.Firefox
8     SELFSERVICE_ICON: Firefox 91.png
9     SELFSERVICE_DESCRIPTION: "%LAST_RUN_SELFSERVICE_DESCRIPTION%"
10    JSS_INVENTORY_NAME: "%NAME%.app"
11    TEST_USERS_GROUP_NAME: "%NAME% test users"
12    USERS_GROUP_NAME: "%NAME% users"
13    USERS_GROUP_TEMPLATE: SmartGroup-users.xml
14    AUTOINSTALL_ALL_SOFTWARE_GROUP_NAME: Software Gets Auto-installed
15    AUTOINSTALL_ALL_SOFTWARE_GROUP_TEMPLATE: SmartGroup-autoinstall-all-software.xml
16    AUTOUPDATE_ALL_SOFTWARE_GROUP_NAME: Software Gets Auto-updated
17    AUTOUPDATE_ALL_SOFTWARE_GROUP_TEMPLATE: SmartGroup-autoupdate-all-software.xml
18    AUTOINSTALL_GROUP_NAME: "%NAME% auto-install"
19    AUTOINSTALL_GROUP_TEMPLATE: SmartGroup-autoinstall.xml
20    AUTOUPDATE_GROUP_NAME: "%NAME% auto-update"
21    AUTOUPDATE_GROUP_TEMPLATE: SmartGroup-autoupdate.xml
22    PROD_VERSION_INSTALLED_GROUP_NAME: "%NAME% current version installed"
```

I use this with our prod jamf recipes. Just as with the older prod JSS recipes, we don't need a parent recipe.

🍎I supply the recipe identifier of the testing jamf recipe, which tells the processor where to look for the correct JSON file.

g-mac-recipes-yaml • Jamf_Staging_Recipes   ! Firefox-prod.jamf.recipe.yaml id-mac-autopkg-recipes-yaml • _Temp_Presentation_Recipes

id-mac-autopkg-recipes-yaml > _Temp_Presentation_Recipes > ! Firefox-prod.jamf.recipe.yaml > ...

```yaml
Description: Uploads the pkg to the JSS, and creates a Self-Service Policy available to members of a Testing group.
Identifier: com.github.eth-its-recipes.jamf.Firefox-prod
MinimumVersion: "2.3"

Input:
  NAME: Firefox
  UNTESTED_RECIPE_IDENTIFIER: local.jamf.Firefox
  SELFSERVICE_ICON: Firefox 91.png
  SELFSERVICE_DESCRIPTION: "%LAST_RUN_SELFSERVICE_DESCRIPTION%"
  JSS_INVENTORY_NAME: "%NAME%.app"
  TEST_USERS_GROUP_NAME: "%NAME% test users"
  USERS_GROUP_NAME: "%NAME% users"
  USERS_GROUP_TEMPLATE: SmartGroup-users.xml
  AUTOINSTALL_ALL_SOFTWARE_GROUP_NAME: Software Gets Auto-installed
  AUTOINSTALL_ALL_SOFTWARE_GROUP_TEMPLATE: SmartGroup-autoinstall-all-software.xml
  AUTOUPDATE_ALL_SOFTWARE_GROUP_NAME: Software Gets Auto-updated
  AUTOUPDATE_ALL_SOFTWARE_GROUP_TEMPLATE: SmartGroup-autoupdate-all-software.xml
  AUTOINSTALL_GROUP_NAME: "%NAME% auto-install"
  AUTOINSTALL_GROUP_TEMPLATE: SmartGroup-autoinstall.xml
  AUTOUPDATE_GROUP_NAME: "%NAME% auto-update"
  AUTOUPDATE_GROUP_TEMPLATE: SmartGroup-autoupdate.xml
  PROD_VERSION_INSTALLED_GROUP_NAME: "%NAME% current version installed"
```

g-mac-recipes-yaml • Jamf_Staging_Recipes   ! Firefox-prod.jamf.recipe.yaml id-mac-autopkg-recipes-yaml • _Temp_Presentation_Recipes

id-mac-autopkg-recipes-yaml > _Temp_Presentation_Recipes > ! Firefox-prod.jamf.recipe.yaml > ...

```yaml
1  Description: Uploads the pkg to the JSS, and creates a Self-Service Policy available to members of a
   Testing group.
2  Identifier: com.github.eth-its-recipes.jamf.Firefox-prod
3  MinimumVersion: "2.3"
4
5  Input:
6    NAME: Firefox
7    UNTESTED_RECIPE_IDENTIFIER: local.jamf.Firefox
8    SELFSERVICE_ICON: Firefox 91.png
9    SELFSERVICE_DESCRIPTION: "%LAST_RUN_SELFSERVICE_DESCRIPTION%"
10   JSS_INVENTORY_NAME: "%NAME%.app"
11   TEST_USERS_GROUP_NAME: "%NAME% test users"
12   USERS_GROUP_NAME: "%NAME% users"
13   USERS_GROUP_TEMPLATE: SmartGroup-users.xml
14   AUTOINSTALL_ALL_SOFTWARE_GROUP_NAME: Software Gets Auto-installed
15   AUTOINSTALL_ALL_SOFTWARE_GROUP_TEMPLATE: SmartGroup-autoinstall-all-software.xml
16   AUTOUPDATE_ALL_SOFTWARE_GROUP_NAME: Software Gets Auto-updated
17   AUTOUPDATE_ALL_SOFTWARE_GROUP_TEMPLATE: SmartGroup-autoupdate-all-software.xml
18   AUTOINSTALL_GROUP_NAME: "%NAME% auto-install"
19   AUTOINSTALL_GROUP_TEMPLATE: SmartGroup-autoinstall.xml
20   AUTOUPDATE_GROUP_NAME: "%NAME% auto-update"
21   AUTOUPDATE_GROUP_TEMPLATE: SmartGroup-autoupdate.xml
22   PROD_VERSION_INSTALLED_GROUP_NAME: "%NAME% current version installed"
```

g-mac-recipes-yaml • Jamf_Staging_Recipes   ! Firefox-prod.jamf.recipe.yaml id-mac-autopkg-recipes-yaml • _Temp_Presentation_Re

id-mac-autopkg-recipes-yaml > _Temp_Presentation_Recipes > ! Firefox-prod.jamf.recipe.yaml > {} Input

```yaml
1   Description: Uploads the pkg to the JSS, and creates a Self-Service Policy available to members of a
    Testing group.
2   Identifier: com.github.eth-its-recipes.jamf.Firefox-prod
3   MinimumVersion: "2.3"
4   ParentRecipe: com.github.eth-its-recipes.jamf.template-prod
5
6 > Input:…
52
53  Process:
54    - Processor: com.github.grahampugh.recipes.preprocessors/LastRecipeRunChecker
55      Arguments:
56        recipeoverride_identifier: "%UNTESTED_RECIPE_IDENTIFIER%"
57
58    - Processor: com.github.eth-its-recipes.processors/JamfUploadSharepointStageCheck
59
60    - Processor: StopProcessingIf
61      Arguments:
62        predicate: "%PROD_PREDICATE%"
63
64    - Processor: com.github.grahampugh.recipes.commonprocessors/VersionRegexGenerator
65
66    - Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader
67      Arguments:
```

I place the processor as the first process in a prod jamf recipe, and the values outputted are used in the subsequent processors.

g-mac-recipes-yaml • Jamf_Staging_Recipes   ! Firefox-prod.jamf.recipe.yaml id-mac-autopkg-recipes-yaml • _Temp_Presentation_Re

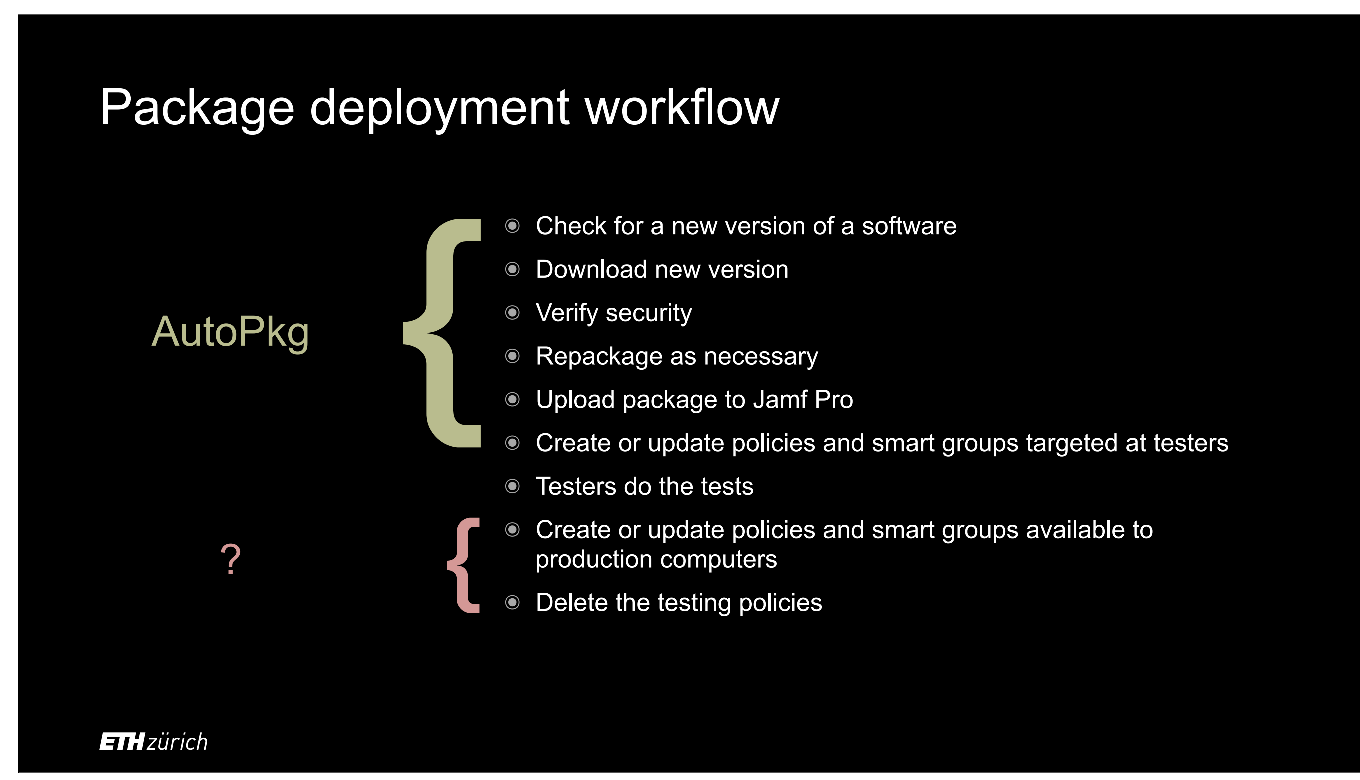
id-mac-autopkg-recipes-yaml > _Temp_Presentation_Recipes > ! Firefox-prod.jamf.recipe.yaml > {} Input

```
1   Description: Uploads the pkg to the JSS, and creates a Self-Service Policy available to members of a
    Testing group.
2   Identifier: com.github.eth-its-recipes.jamf.Firefox-prod
3   MinimumVersion: "2.3"
4   ParentRecipe: com.github.eth-its-recipes.jamf.template-prod
5
6 > Input: …
52
53  Process:
54    - Processor: com.github.grahampugh.recipes.preprocessors/LastRecipeRunChecker
55      Arguments:
56        recipeoverride_identifier: "%UNTESTED_RECIPE_IDENTIFIER%"
57
58    - Processor: com.github.eth-its-recipes.processors/JamfUploadSharepointStageCheck
59
60    - Processor: StopProcessingIf
61      Arguments:
62        predicate: "%PROD_PREDICATE%"
63
64    - Processor: com.github.grahampugh.recipes.commonprocessors/VersionRegexGenerator
65
66    - Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader
```

```
      Arguments:
        policy_template: "%AUTOUPDATE_POLICY_TEMPLATE%"
        policy_name: "%AUTOUPDATE_POLICY_NAME%"
        icon: ""
        replace_policy: "False"

    - Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader
      Comment: Self Service install policy
      Arguments:
        SELFSERVICE_POLICY_CATEGORY: "%PKG_CATEGORY%"
        policy_template: "%SELFSERVICE_POLICY_TEMPLATE%"
        policy_name: "%SELFSERVICE_POLICY_NAME%"
        icon: "%SELFSERVICE_ICON%"
        replace_policy: "True"

    - Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader
      Comment: Self Service update policy
      Arguments:
        policy_template: "%UPDATE_POLICY_TEMPLATE%"
        policy_name: "%UPDATE_POLICY_NAME%"
        icon: "%SELFSERVICE_ICON%"
        replace_policy: "True"

    - Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyDeleter
      Comment: Delete the untested policy
      Arguments:
        policy_name: "%LAST_RUN_POLICY_NAME%"

```

As the last process in the prod jamf recipes, I use a processor called JamfPolicyDeleter to remove the Self Service testing policy, which we don't want anymore.

```
141        Arguments:
142          policy_template: "%AUTOUPDATE_POLICY_TEMPLATE%"
143          policy_name: "%AUTOUPDATE_POLICY_NAME%"
144          icon: ""
145          replace_policy: "False"
146
147      - Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader
148        Comment: Self Service install policy
149        Arguments:
150          SELFSERVICE_POLICY_CATEGORY: "%PKG_CATEGORY%"
151          policy_template: "%SELFSERVICE_POLICY_TEMPLATE%"
152          policy_name: "%SELFSERVICE_POLICY_NAME%"
153          icon: "%SELFSERVICE_ICON%"
154          replace_policy: "True"
155
156      - Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader
157        Comment: Self Service update policy
158        Arguments:
159          policy_template: "%UPDATE_POLICY_TEMPLATE%"
160          policy_name: "%UPDATE_POLICY_NAME%"
161          icon: "%SELFSERVICE_ICON%"
162          replace_policy: "True"
163
164      - Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyDeleter
165        Comment: Delete the untested policy
166        Arguments:
167          policy_name: "%LAST_RUN_POLICY_NAME%"
168
```



So now we have recipes for bringing in new packages for testing, 🍎and for staging them to production.

But hold on - 🍎how do we know when a package is ready for production, and more importantly how do we automate the process of staging at the right time?

We could just run recipes manually when it has been determined that something is ready for production, but that also can get difficult to keep up with, so we wanted to automate this step as well.

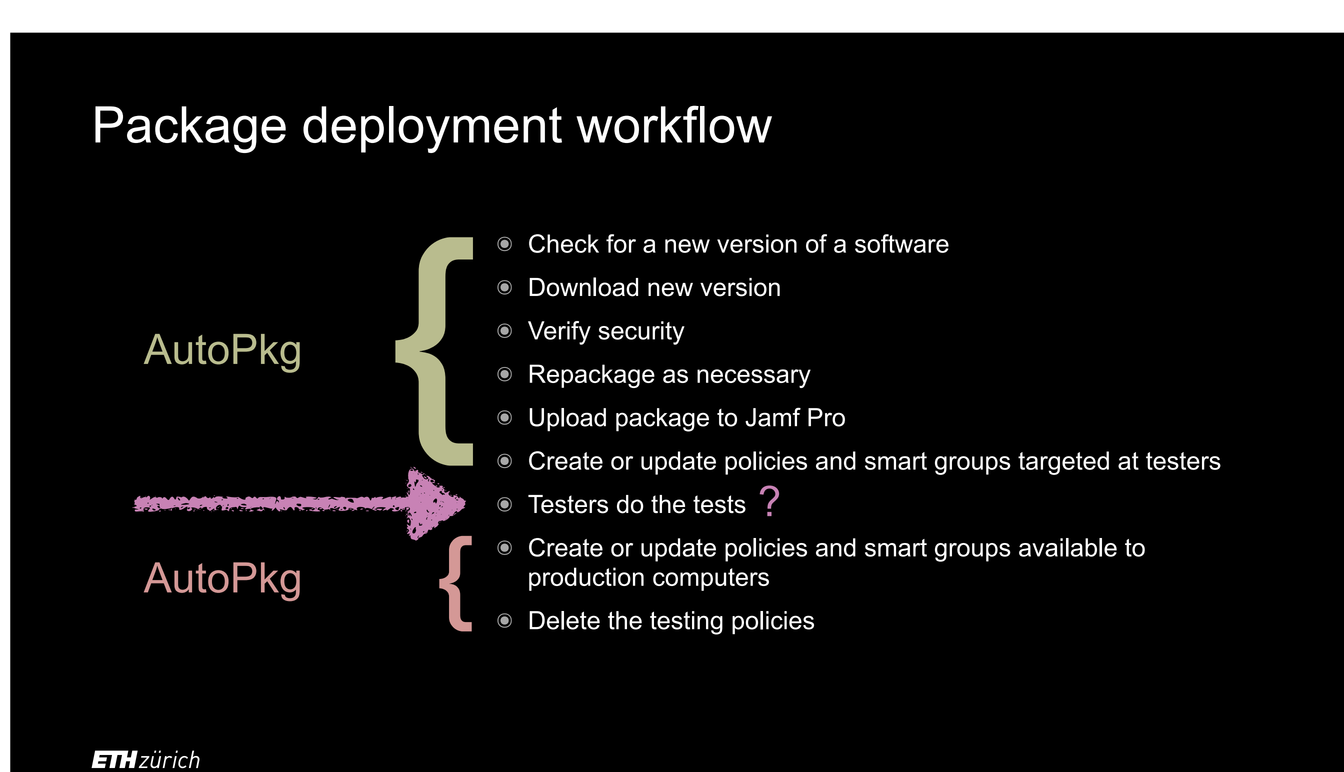
Package deployment workflow

AutoPkg

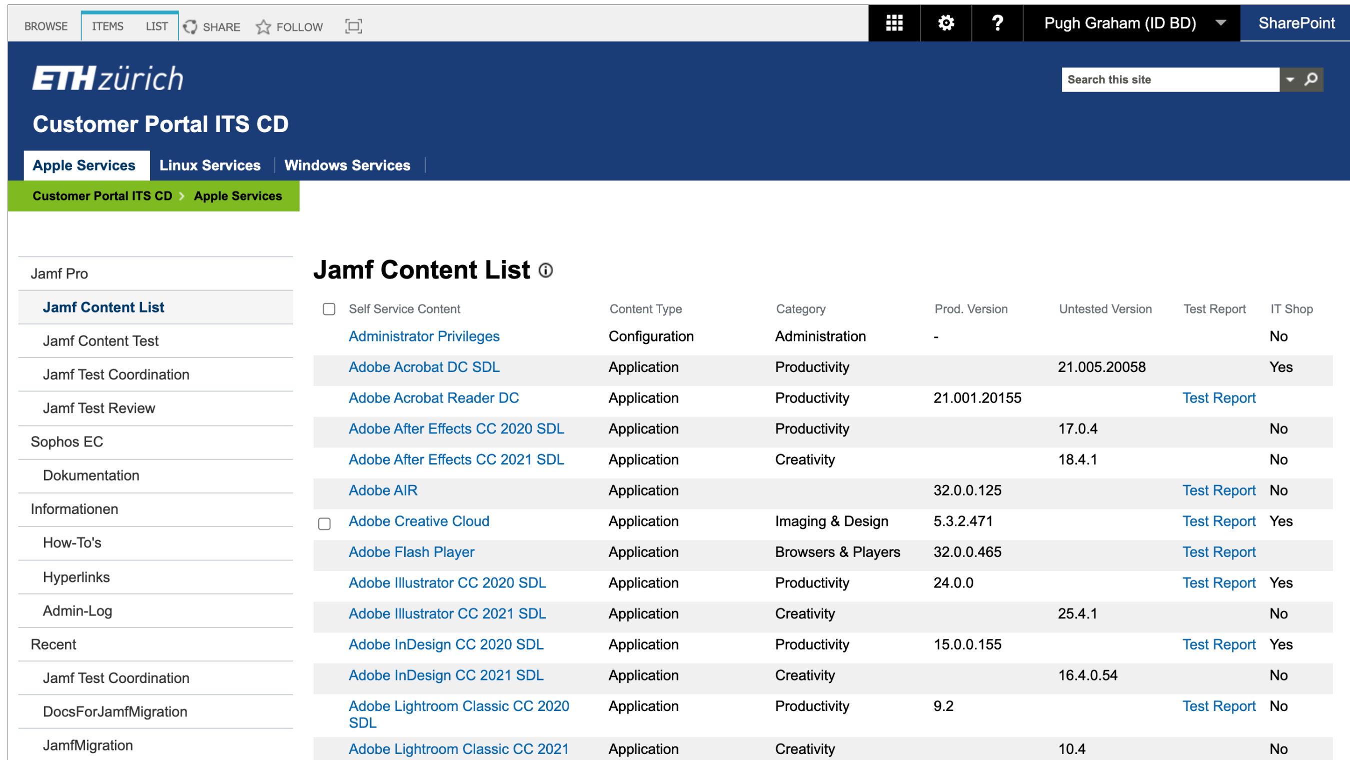
- Check for a new version of a software
- Download new version
- Verify security
- Repackage as necessary
- Upload package to Jamf Pro
- Create or update policies and smart groups targeted at testers
- Testers do the tests

AutoPkg

- Create or update policies and smart groups available to production computers
- Delete the testing policies



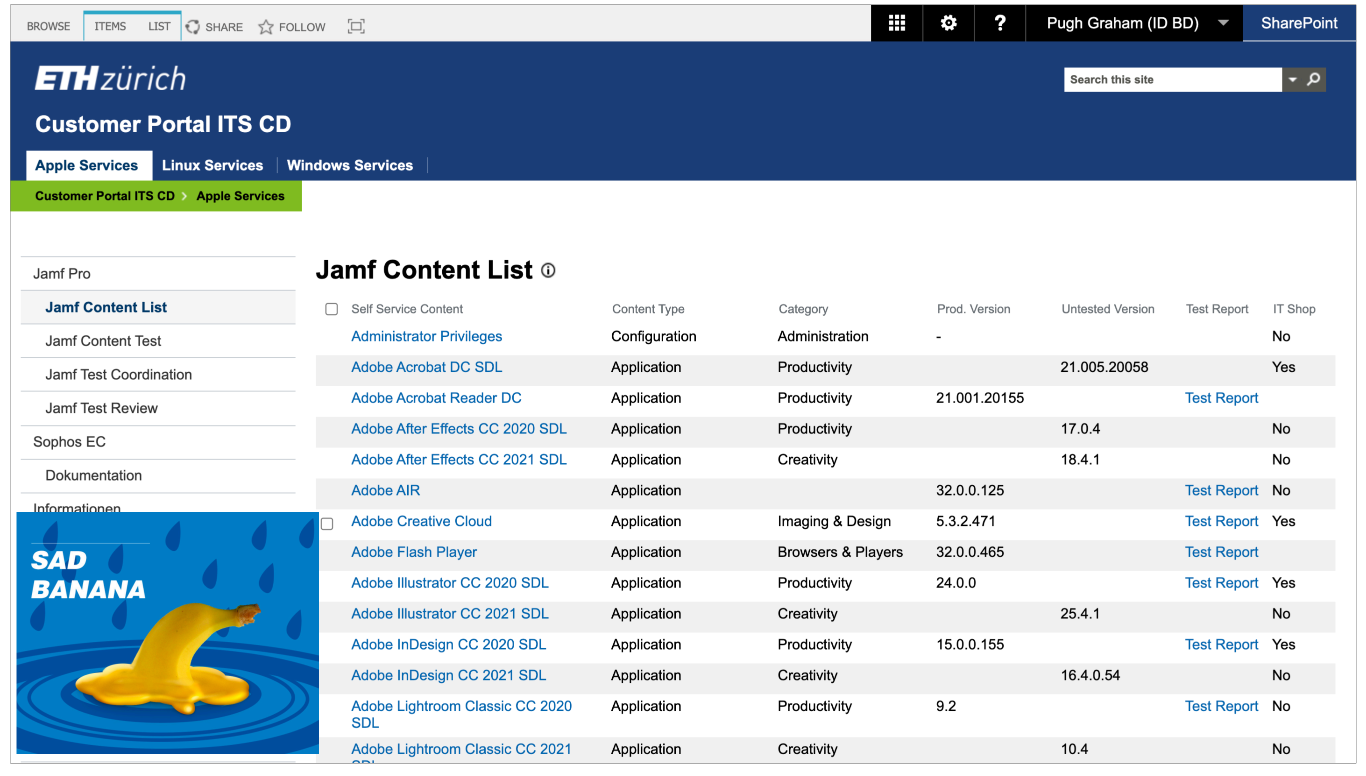
Package deployment workflow
AutoPkg
Check for a new version of a software
Download new version
Verify security
Repackage as necessary
Upload package to Jamf Pro
Create or update policies and smart groups targeted at testers
Testers do the tests ?
AutoPkg
Create or update policies and smart groups available to production computers
Delete the testing policies
ETH zürich



Self Service Content	Content Type	Category	Prod. Version	Untested Version	Test Report	IT Shop
Administrator Privileges	Configuration	Administration	-			No
Adobe Acrobat DC SDL	Application	Productivity		21.005.20058		Yes
Adobe Acrobat Reader DC	Application	Productivity	21.001.20155		Test Report	
Adobe After Effects CC 2020 SDL	Application	Productivity		17.0.4		No
Adobe After Effects CC 2021 SDL	Application	Creativity		18.4.1		No
Adobe AIR	Application		32.0.0.125		Test Report	No
Adobe Creative Cloud	Application	Imaging & Design	5.3.2.471		Test Report	Yes
Adobe Flash Player	Application	Browsers & Players	32.0.0.465		Test Report	
Adobe Illustrator CC 2020 SDL	Application	Productivity	24.0.0		Test Report	Yes
Adobe Illustrator CC 2021 SDL	Application	Creativity		25.4.1		No
Adobe InDesign CC 2020 SDL	Application	Productivity	15.0.0.155		Test Report	Yes
Adobe InDesign CC 2021 SDL	Application	Creativity		16.4.0.54		No
Adobe Lightroom Classic CC 2020 SDL	Application	Productivity	9.2		Test Report	No
Adobe Lightroom Classic CC 2021	Application	Creativity		10.4		No

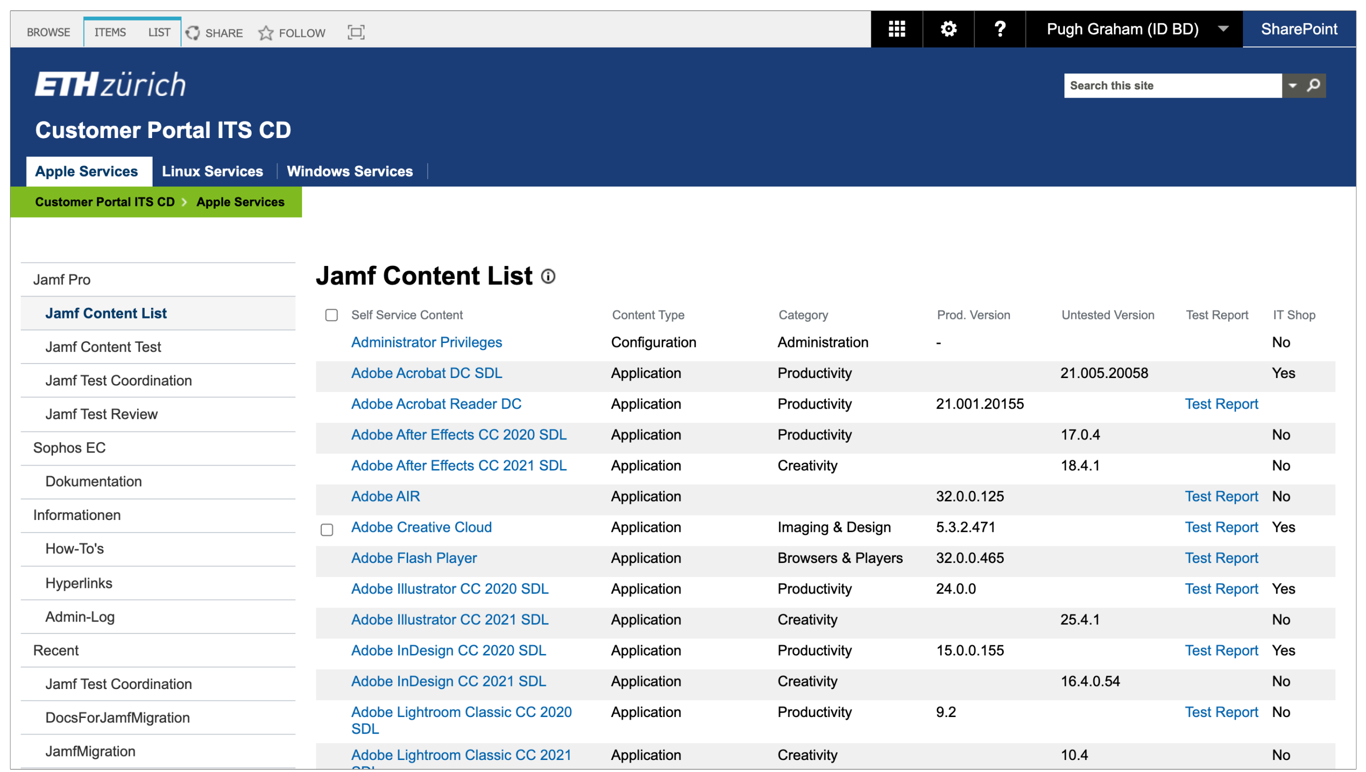
Our small Apple Client Delivery team is not responsible for testing whether a new package functions properly or not - apart from a few core apps like browsers, we outsource that responsibility to our customers, who are closer to the users of these products. To aid in this process, we maintain a software catalogue for our customers, on an internal 🍎 SharePoint site.

For all SharePoint's downsides, it does have an API. 🍎
So we can automate the filling in of information of each new version of a product into this SharePoint site.



Jamf Content List

Self Service Content	Content Type	Category	Prod. Version	Untested Version	Test Report	IT Shop
Administrator Privileges	Configuration	Administration	-			No
Adobe Acrobat DC SDL	Application	Productivity		21.005.20058		Yes
Adobe Acrobat Reader DC	Application	Productivity	21.001.20155		Test Report	
Adobe After Effects CC 2020 SDL	Application	Productivity		17.0.4		No
Adobe After Effects CC 2021 SDL	Application	Creativity		18.4.1		No
Adobe AIR	Application		32.0.0.125		Test Report	No
Adobe Creative Cloud	Application	Imaging & Design	5.3.2.471		Test Report	Yes
Adobe Flash Player	Application	Browsers & Players	32.0.0.465		Test Report	
Adobe Illustrator CC 2020 SDL	Application	Productivity	24.0.0		Test Report	Yes
Adobe Illustrator CC 2021 SDL	Application	Creativity		25.4.1		No
Adobe InDesign CC 2020 SDL	Application	Productivity	15.0.0.155		Test Report	Yes
Adobe InDesign CC 2021 SDL	Application	Creativity		16.4.0.54		No
Adobe Lightroom Classic CC 2020 SDL	Application	Productivity	9.2		Test Report	No
Adobe Lightroom Classic CC 2021	Application	Creativity		10.4		No



Self Service Content	Content Type	Category	Prod. Version	Untested Version	Test Report	IT Shop
Administrator Privileges	Configuration	Administration	-			No
Adobe Acrobat DC SDL	Application	Productivity		21.005.20058		Yes
Adobe Acrobat Reader DC	Application	Productivity	21.001.20155		Test Report	
Adobe After Effects CC 2020 SDL	Application	Productivity		17.0.4		No
Adobe After Effects CC 2021 SDL	Application	Creativity		18.4.1		No
Adobe AIR	Application		32.0.0.125		Test Report	No
Adobe Creative Cloud	Application	Imaging & Design	5.3.2.471		Test Report	Yes
Adobe Flash Player	Application	Browsers & Players	32.0.0.465		Test Report	
Adobe Illustrator CC 2020 SDL	Application	Productivity	24.0.0		Test Report	Yes
Adobe Illustrator CC 2021 SDL	Application	Creativity		25.4.1		No
Adobe InDesign CC 2020 SDL	Application	Productivity	15.0.0.155		Test Report	Yes
Adobe InDesign CC 2021 SDL	Application	Creativity		16.4.0.54		No
Adobe Lightroom Classic CC 2020 SDL	Application	Productivity	9.2		Test Report	No
Adobe Lightroom Classic CC 2021	Application	Creativity		10.4		No

JamfUploadSharepointUpdater

Input variables

- **JSS_URL:**
  - required: True
  - description: Jamf Pro server URL.
- **SP_URL:**
  - required: True
  - description: SharePoint Portal URL.
- **SP_USER:**
  - required: True
  - description: SharePoint portal username with read privileges.
- **SP_PASS:**
  - required: True
  - description: Password for the SharePoint portal

Output variables

None

ETH zürich

We do this with a processor I wrote called JamfUploadSharepointUpdater.
🍎Always catchy with the titles.

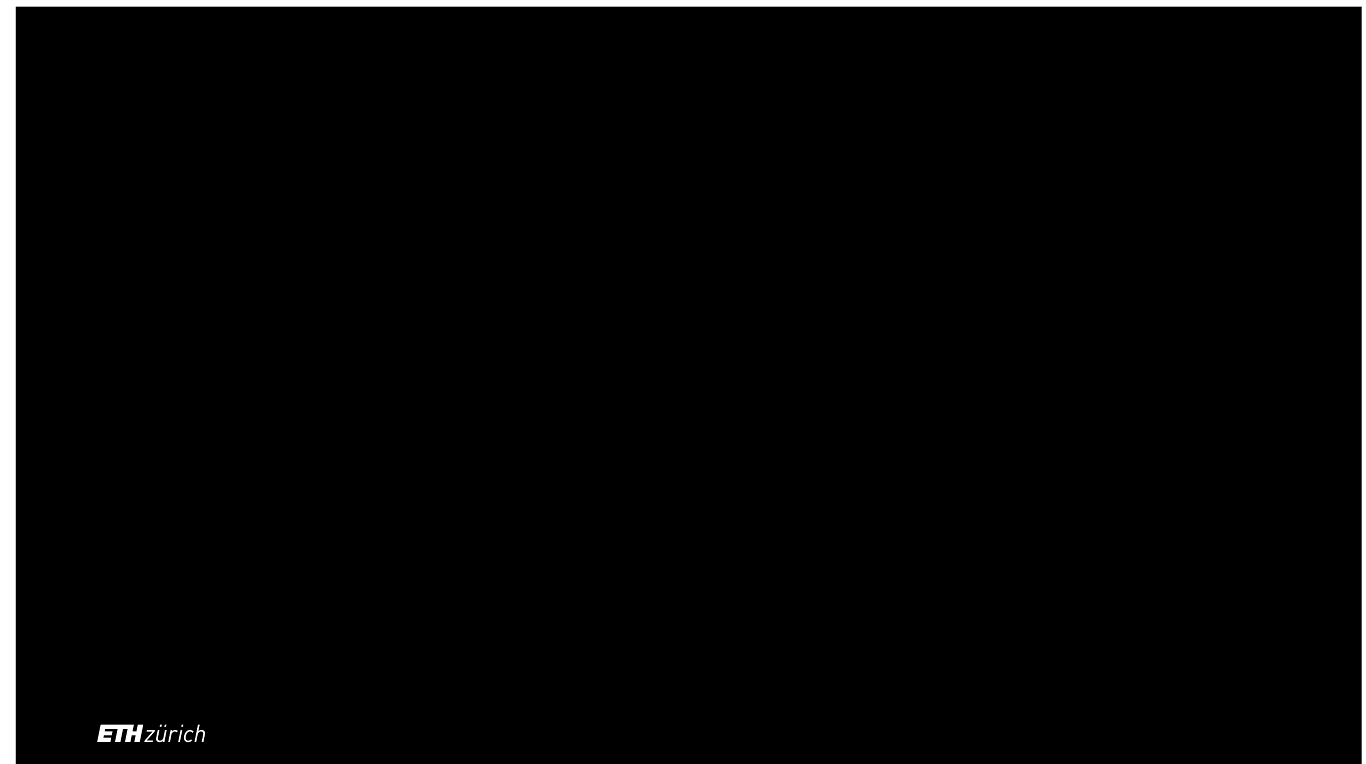
JamfUploadSharepointUpdater ✔

Input variables

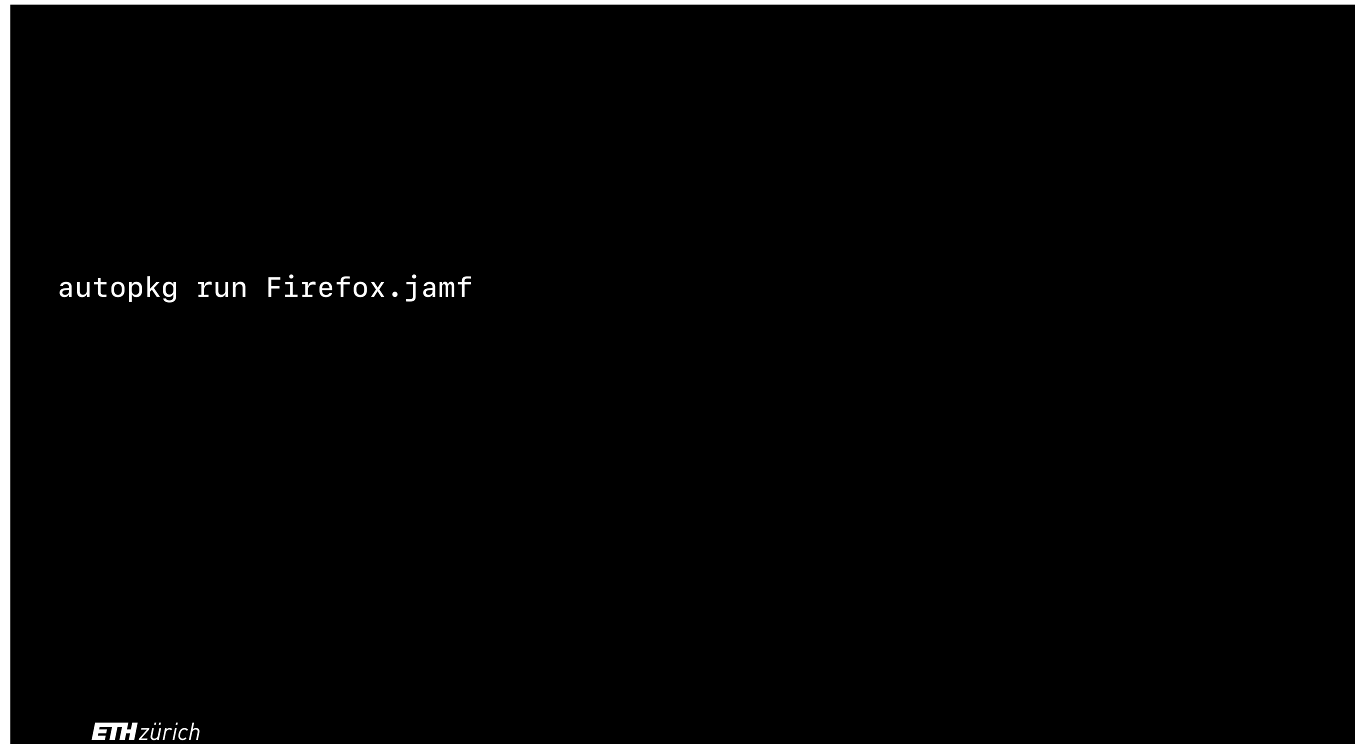
- **JSS_URL:**
  - required: True
  - description: Jamf Pro server URL.
- **SP_URL:**
  - required: True
  - description: SharePoint Portal URL.
- **SP_USER:**
  - required: True
  - description: SharePoint portal username with read privileges.
- **SP_PASS:**
  - required: True
  - description: Password for the SharePoint portal

Output variables

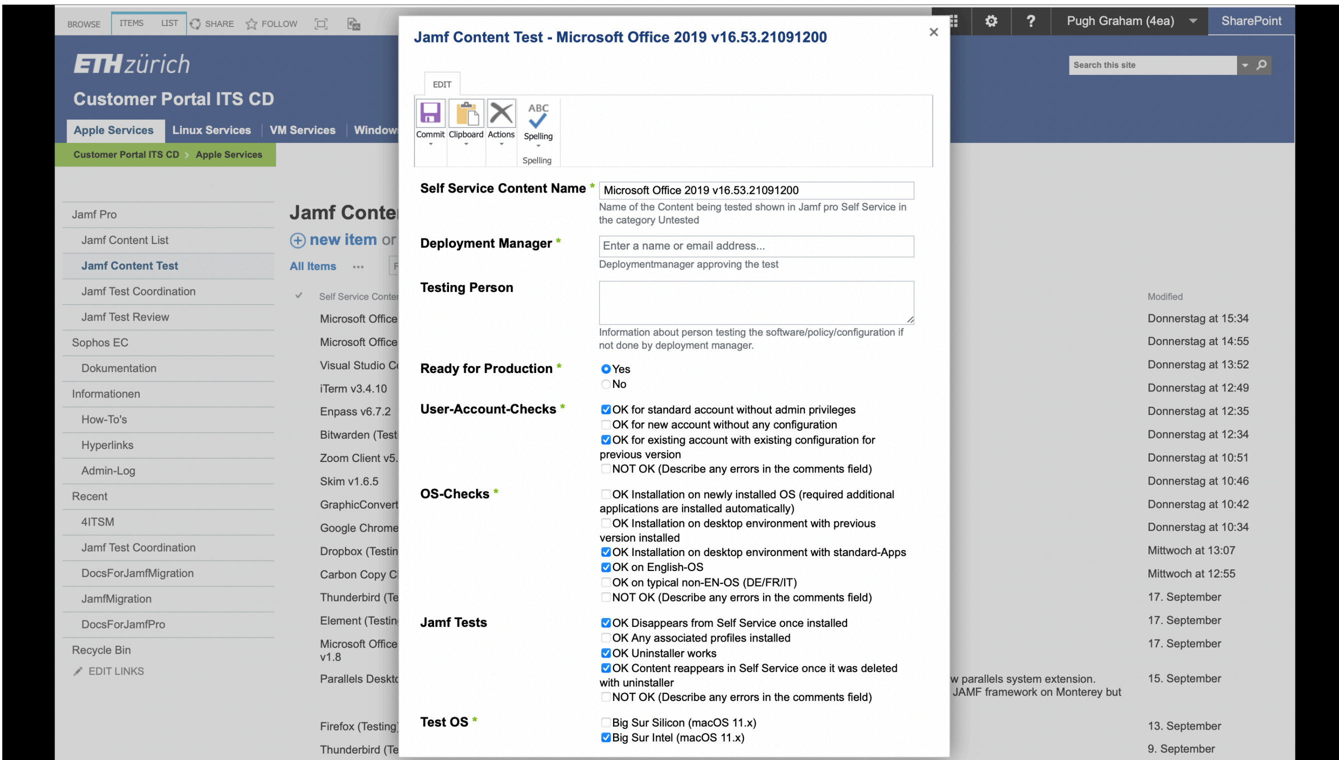
None



This is a post-processor, which we run at the end of our testing jamf recipes 🍎It updates the SharePoint site with the new information required by the testers.


autopkg run Firefox.jamf
ETH zürich

```
autopkg run Firefox.jamf
--post=com.github.eth-its-recipes.processors/JamfUploadSharepointUpdater
```



Our testers are alerted via email to the newly uploaded package, and one of them performs tests following a template that we generated in the SharePoint portal.

These tests include whether the application can be installed as a standard user, whether the item disappears from Self Service once it's installed, whether an uninstaller appears in Self Service, whether the uninstaller works properly, and some basic tests on whether the app functions.

When they have completed the test, they mark in our form whether it is Ready for Production or not. If not, they contact us to take another look at the recipe in case there's something that needs to be changed.

JamfUploadSharepointStageCheck

Input variables

- **JSS_URL:**
  - required: True
  - description: Jamf Pro server URL.
- **SP_URL:**
  - required: True
  - description: SharePoint Portal URL.
- **SP_USER:**
  - required: True
  - description: SharePoint portal username with read privileges.
- **SP_PASS:**
  - required: True
  - description: Password for the SharePoint portal

Output variables

- **ready_to_stage:**
  - description: Boolean value of whether the product is ready to be promoted to production or not.

ETH zürich

A second processor called JamfUploadSharepointStageCheck is added to the beginning of a prod jamf recipe to check the Sharepoint site and see if there is a new version ready to be released to production.

```
! Firefox-prod.jamf.recipe.yaml U ●
id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > ! Firefox-prod.jamf.recipe.yaml > [ ] Process > { } 2
1   Description: Uploads the pkg to the JSS, and creates a Self-Service Policy available to members of a Testing
    group.
2   Identifier: com.github.eth-its-recipes.jamf.Firefox-prod
3   MinimumVersion: "2.3"
4
5 > Input: ···
51
52  Process:
53    - Processor: com.github.grahampugh.recipes.preprocessors/LastRecipeRunChecker
54      Arguments:
55        recipeoverride_identifier: "%UNTESTED_RECIPE_IDENTIFIER%"
56
57    - Processor: com.github.eth-its-recipes.processors/JamfUploadSharepointStageCheck
58
59    - Processor: StopProcessingIf
60      Arguments:
61        predicate: ready_to_stage == False
62
63    - Processor: com.github.grahampugh.recipes.commonprocessors/VersionRegexGenerator
64
65    - Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader
66      Arguments:
67        category_name: "%TRIGGER_POLICY_CATEGORY%"
68
69    - Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader
```

More specifically, it is inserted after the LastRecipeRunChecker processor.

🍎The StopProcessingIf processor then tests whether previous processor determined that the app is ready to stage. If that is not the case, the recipe is stopped at this point, so the processes below are not run, which means the title is not promoted to production.

! Firefox-prod.jamf.recipe.yaml U ●

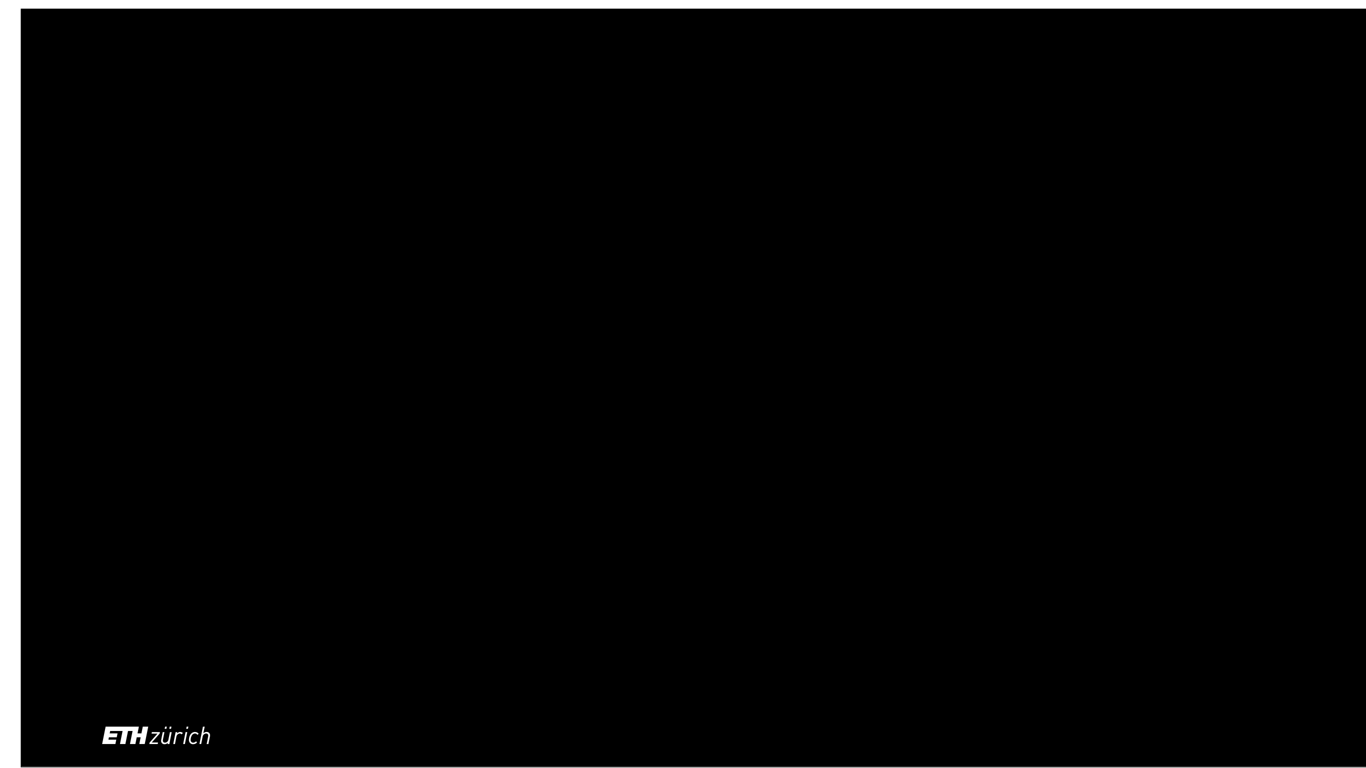
id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > ! Firefox-prod.jamf.recipe.yaml > [ ] Process > { } 2

```
1   Description: Uploads the pkg to the JSS, and creates a Self-Service Policy available to members of a Testing
    group.
2   Identifier: com.github.eth-its-recipes.jamf.Firefox-prod
3   MinimumVersion: "2.3"
4
5 > Input: ...
51
52  Process:
53    - Processor: com.github.grahampugh.recipes.preprocessors/LastRecipeRunChecker
54      Arguments:
55        recipeoverride_identifier: "%UNTESTED_RECIPE_IDENTIFIER%"
56
57    - Processor: com.github.eth-its-recipes.processors/JamfUploadSharepointStageCheck
58
59    - Processor: StopProcessingIf
60      Arguments:
61        predicate: ready_to_stage == False
62
63    - Processor: com.github.grahampugh.recipes.commonprocessors/VersionRegexGenerator
64
65    - Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader
66      Arguments:
67        category_name: "%TRIGGER_POLICY_CATEGORY%"
68
```

! Firefox-prod.jamf.recipe.yaml U ●

id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > ! Firefox-prod.jamf.recipe.yaml > [ ] Process > { } 2

```
1   Description: Uploads the pkg to the JSS, and creates a Self-Service Policy available to members of a Testing
    group.
2   Identifier: com.github.eth-its-recipes.jamf.Firefox-prod
3   MinimumVersion: "2.3"
4
5 > Input: ⋯
51
52  Process:
53    - Processor: com.github.grahampugh.recipes.preprocessors/LastRecipeRunChecker
54      Arguments:
55        recipeoverride_identifier: "%UNTESTED_RECIPE_IDENTIFIER%"
56
57    - Processor: com.github.eth-its-recipes.processors/JamfUploadSharepointStageCheck
58
59    - Processor: StopProcessingIf
60      Arguments:
61        predicate: ready_to_stage == False
62
63    - Processor: com.github.grahampugh.recipes.commonprocessors/VersionRegexGenerator
64
65    - Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader
66      Arguments:
67        category_name: "%TRIGGER_POLICY_CATEGORY%"
68
```

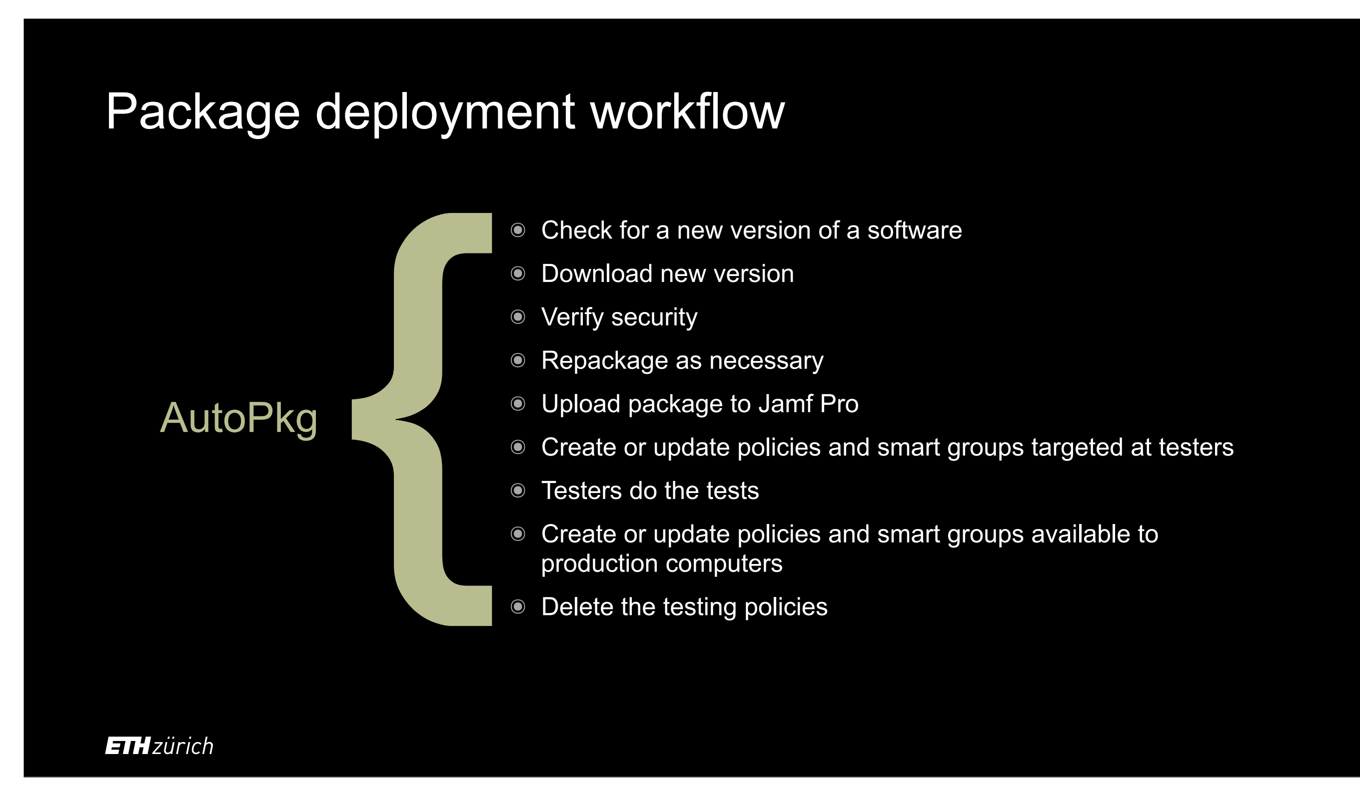


Finally, as a post-processor, 🍎we call the JamfUploadSharepointUpdater once again, this time to mark in the SharePoint that the package has been Released to Production, and to update the production version string in our Content List.

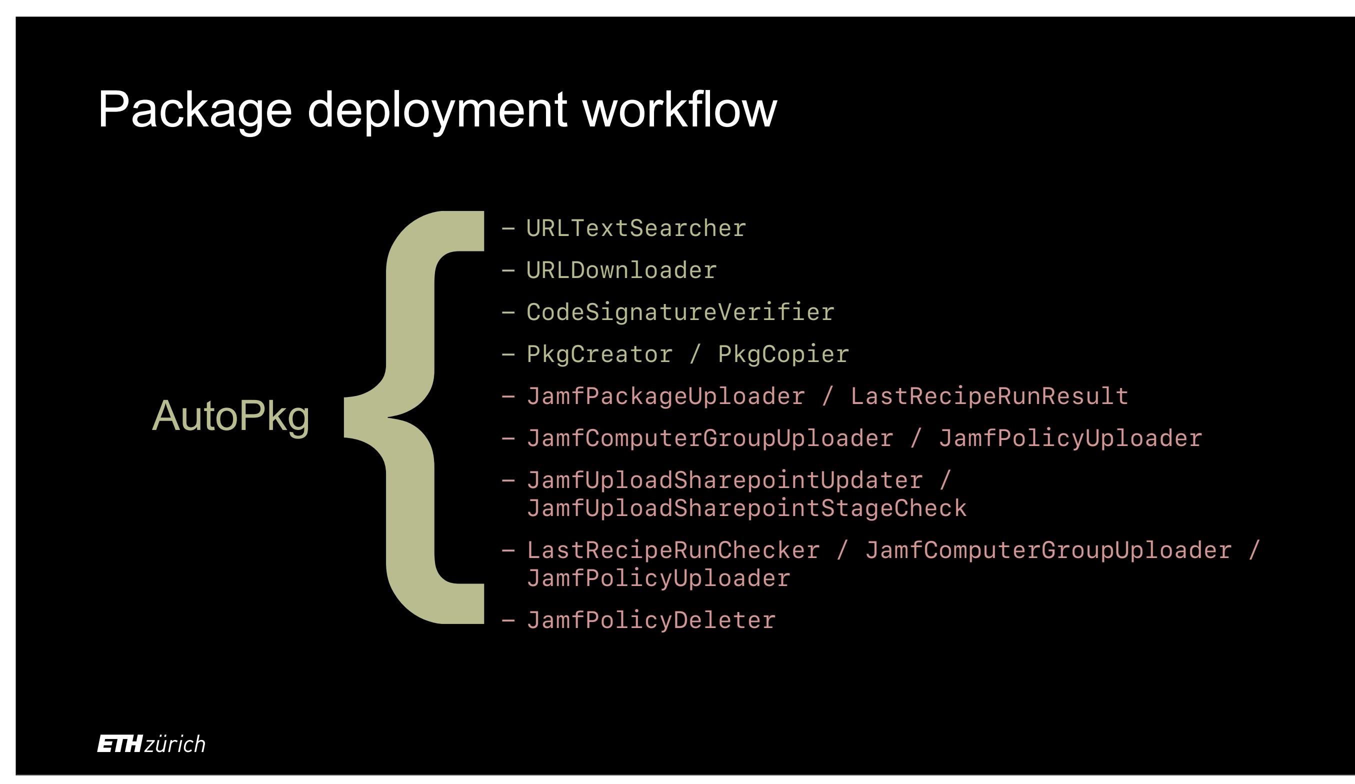
autopkg run Firefox-prod.jamf
ETH zürich

```
autopkg run Firefox-prod.jamf
--post=com.github.eth-its-recipes.processors/JamfUploadSharepointUpdater
```

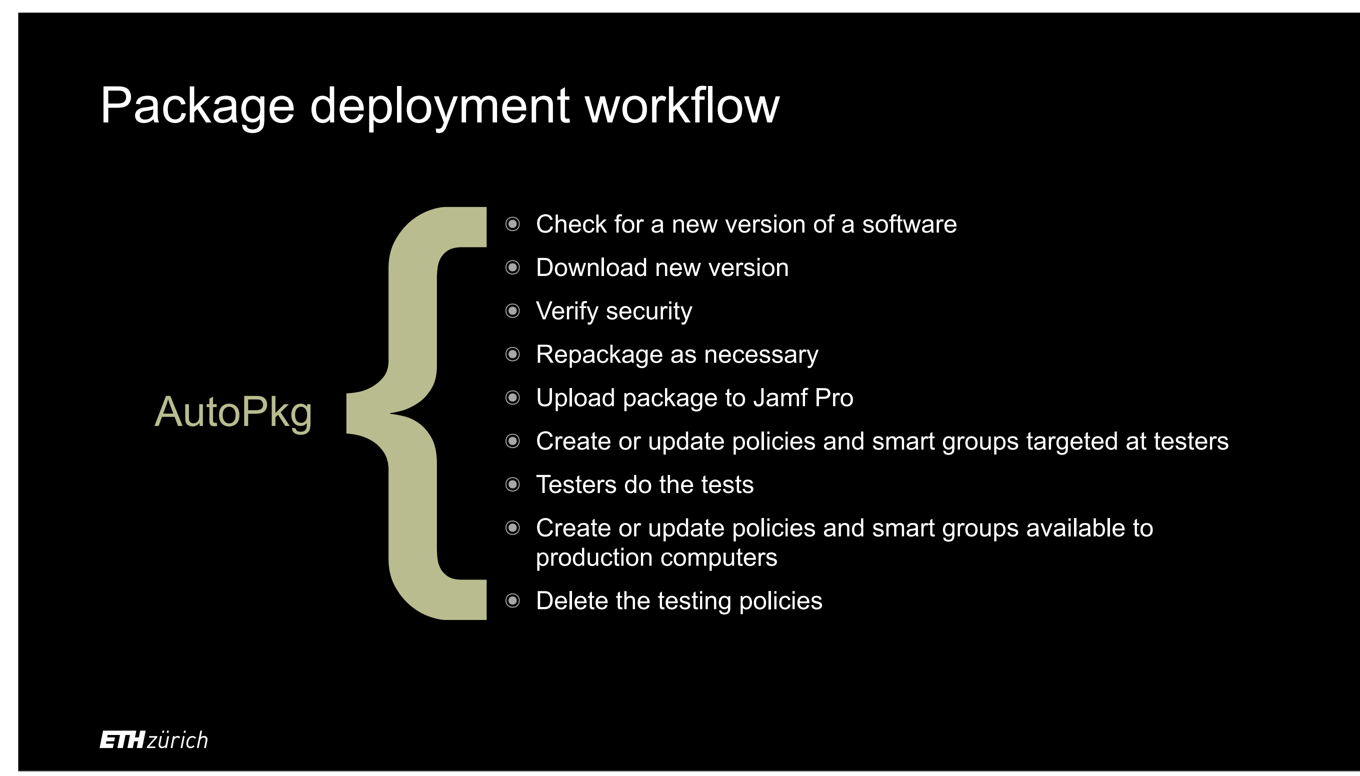
ETH zürich



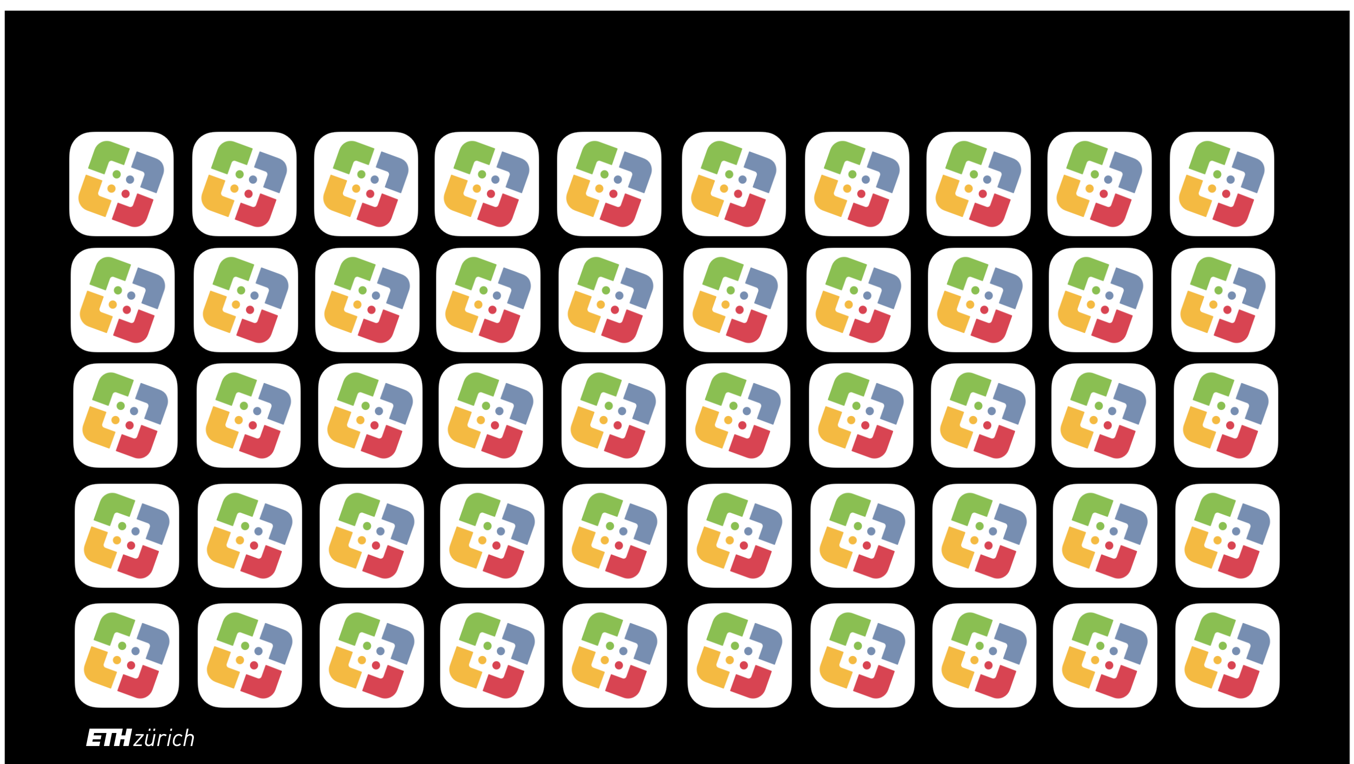
So with that in place, we have autopkg processors to do every single step of the package deployment workflow from download to promotion to production.

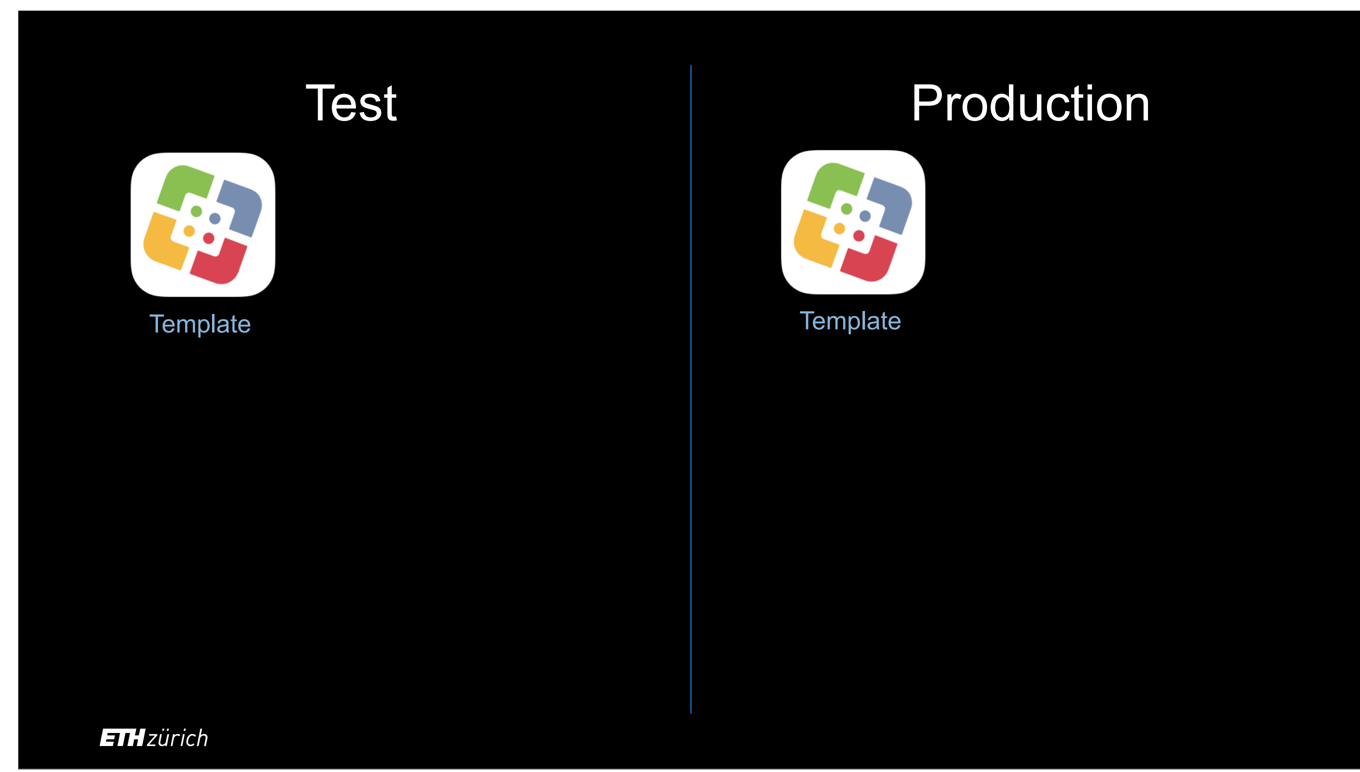


This is all the processors - the first set are the core AutoPkg processors of the download and pkg recipes, and the rest are shared processors that I've developed at ETH.



To move on to the second section of this presentation, a reminder that at ETH, we don't have only one Jamf instance, we have many, and all these need the new untested versions of software titles to be added, and all of them need the production policies at the right time.


ETH zürich

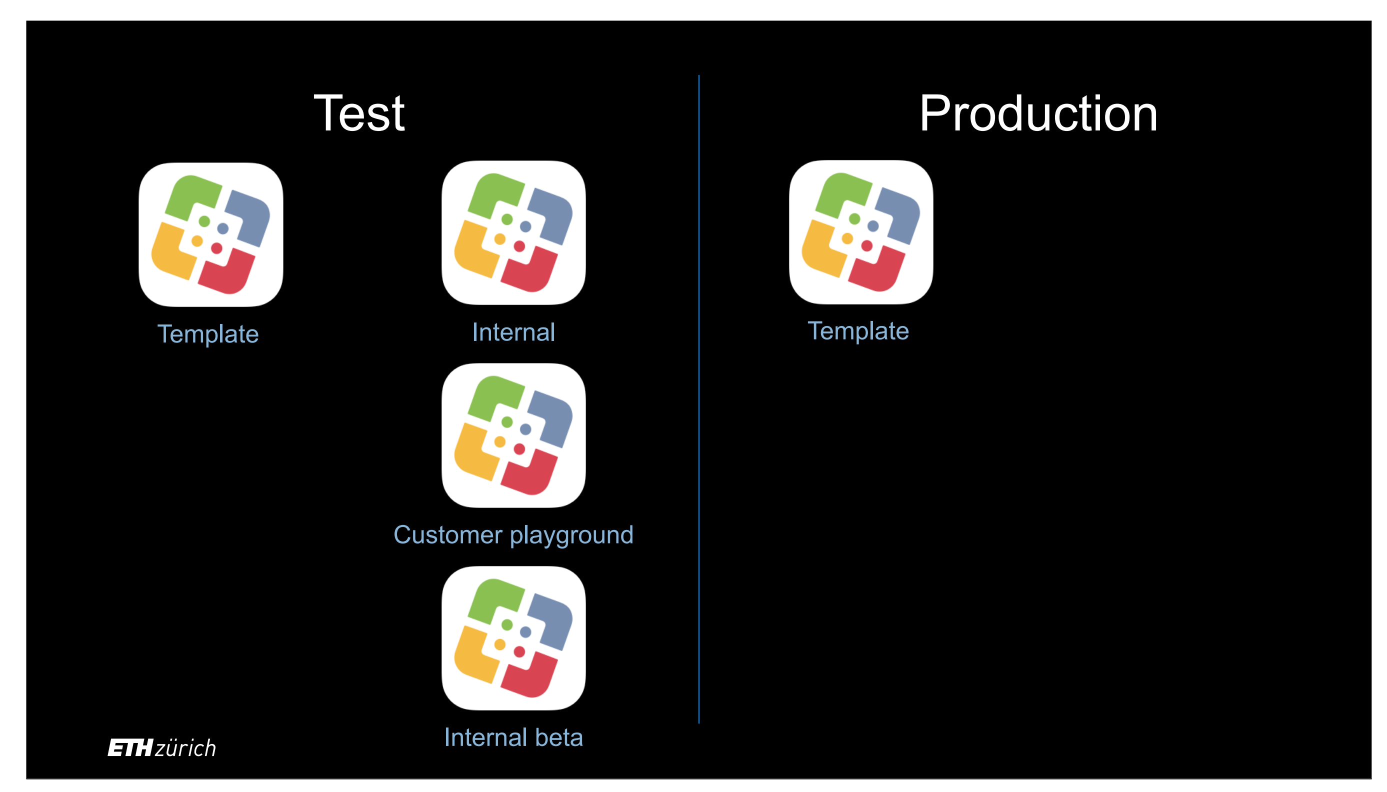


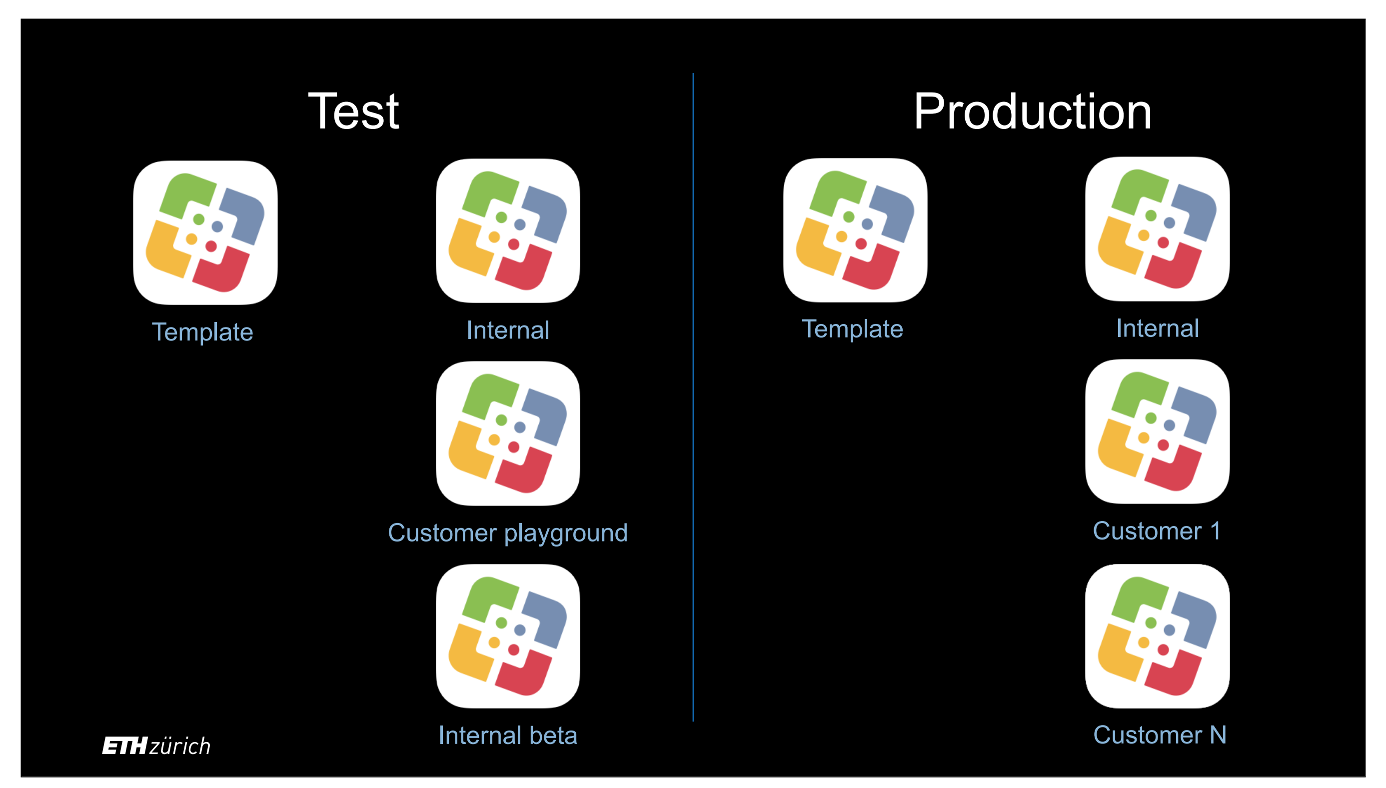
As a brief overview of our setup, we maintain template Jamf Pro instances in our test and production systems, which have no clients enrolled to them, and are not accessible by any customers. We use these as the source of truth for our centrally provided content.

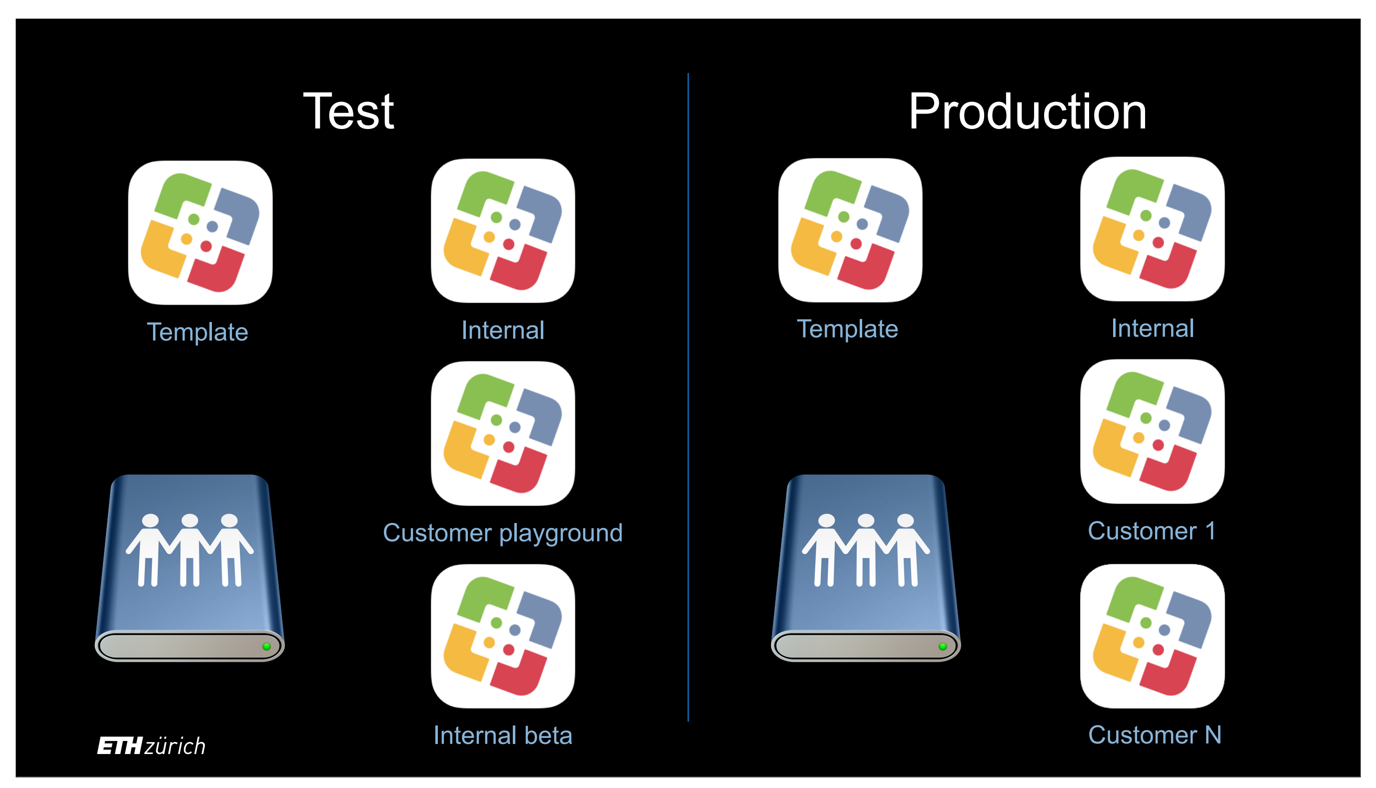
🍎Our test system has a few instances, for testing current and beta Jamf versions and to allow some of our customers to test stuff out too.

🍎On the production side, we have all our customers' main instances, as well as an instance for our own team's productive Macs.

🍎The test system and the production system each have a single FileShare Distribution point for all instances. We decided early on in designing our Jamf Pro architecture that it would not make sense for each instance to have its own entire repo.


Test
Production
Template
Internal
Customer playground
Internal beta
Template
ETH zürich


Test
Production
Template
Internal
Customer playground
Internal beta
Template
Internal
Customer 1
Customer N
ETH zürich


Test
Template
Internal
Customer playground
Internal beta
Production
Template
Internal
Customer 1
Customer N
ETH zürich

As part of the original infrastructure setup, I wrote a bash script for copying API objects from one instance to another instance, or more than one, or all other instances using the API.

The script is pretty complex, because it makes a very serious attempt to figure out all the dependencies of the object that you want to copy, and copies them all in the correct order so that conflicts don't happen.

It also has to know which smart groups should not be overwritten because they are the ones our customers edit directly.

It's mainly for that latter reason that I've never considered putting this script in a public repo, because it's very specifically designed for our workflow.

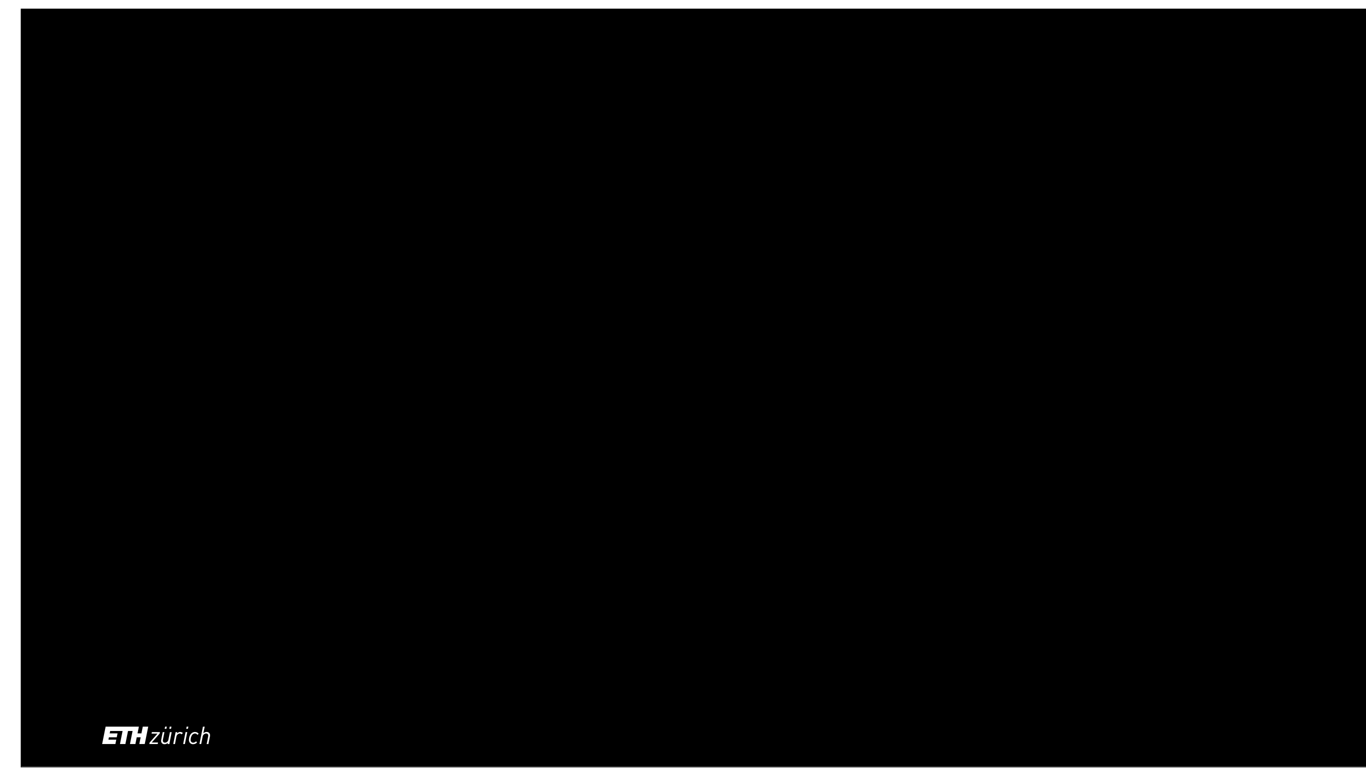
This script was our original way of promoting a title to production. It had options to create or update all the smart groups and policies required. However, this became too hard to manage once we scaled up and encountered more and more exceptions to the standard pattern, and that's when we needed to start to use AutoPkg recipes for creating and updating production policies.

But the script is still used for copying individual objects like a single policy and all its dependencies.

```
Enter the destination server name (or enter for 'prd') :

Enter the destination JSS instance name(s) to which to upload API data
   or press enter for 'template',
   or enter 'ALL' to propagate to all destination instances ('template' will be checked for
the policy's existence).
   or enter a slash ('/') for a non-context JSS : ALL

   [main] Destination Jamf Pro Server: 'prd'

API object type options:
   A - [A]dvanced Computer Search
   C - [C]onfiguration Profile
   U - Configuration Profile - specify UUID to rescue orphan profile
   E - [E]xtension Attribute
   G - Computer [G]roup
   M - [M]ac App Store App
   I - [i]OS App Store App
   P - [P]ackage object
   R - [R]estricted software
   S - [S]cript
   T - Ca[T]egory
   L or leave blank for Po[L]icy
Enter a letter from the above options:
```

ETH zürich

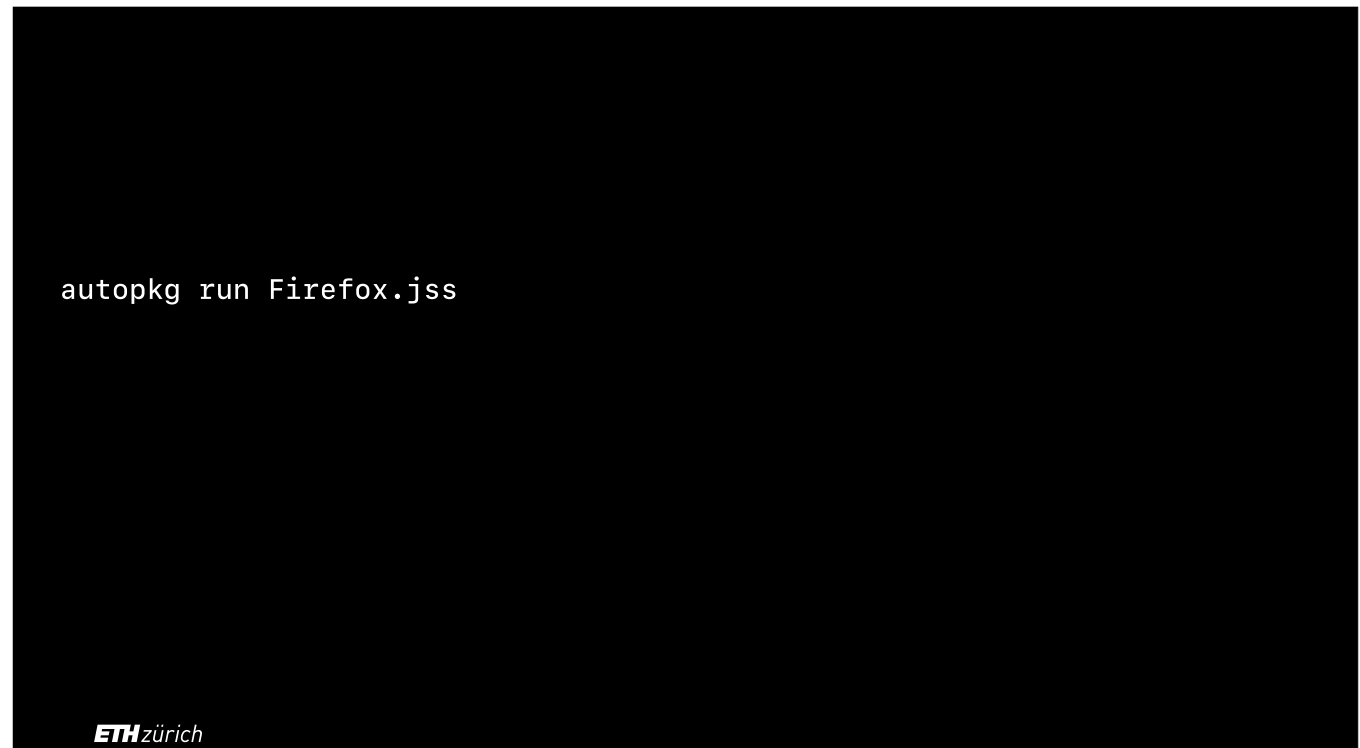
It can be run interactively, which is useful for ad hoc changes that we want to propagate across our instances, and it has a delete option, for things that we need to clean up, which of course happens from time to time, such as when obsolete applications need to be removed.



But it can also be run directly from the command line with arguments, which means it can be automated.

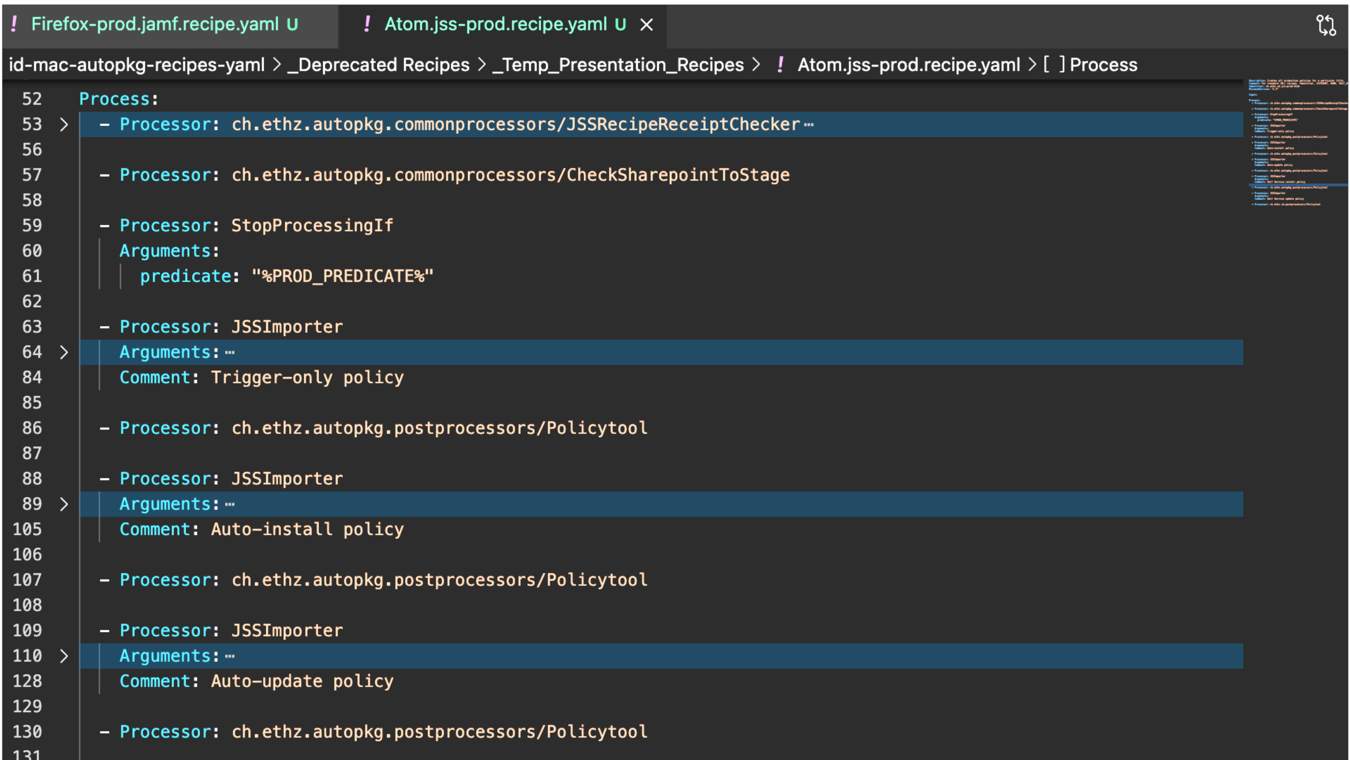
```
./jamf-api-tool.sh
  --keychain ${JAMF_KEYCHAIN_PASSWORD}
  --copy
  --server "our-prod-server"
  --policy "Install Firefox"
  --source "template"
  --dest "ALL"
```

Because of this, when we started to move to using AutoPkg for promoting software titles to production, I was able to write an AutoPkg 🍎post-processor which shells out to bash to run the script with the correct parameters. It copies a policy and all of its dependencies from the template to all the destination instances.


autopkg run Firefox.jss
ETH zürich

```
autopkg run Firefox.jss
--post=ch.ethz.autopkg.postprocessors/Policytool
```

ETH zürich



For recipes that contain multiple JSSImporter processors, we can't just add it to the command as a post-processor - the processor has to be added to the recipe itself after each of the JSSImporter processor steps, so that each individual policy and its dependencies get copied to the destination instances before the recipe moves on to the next policy.

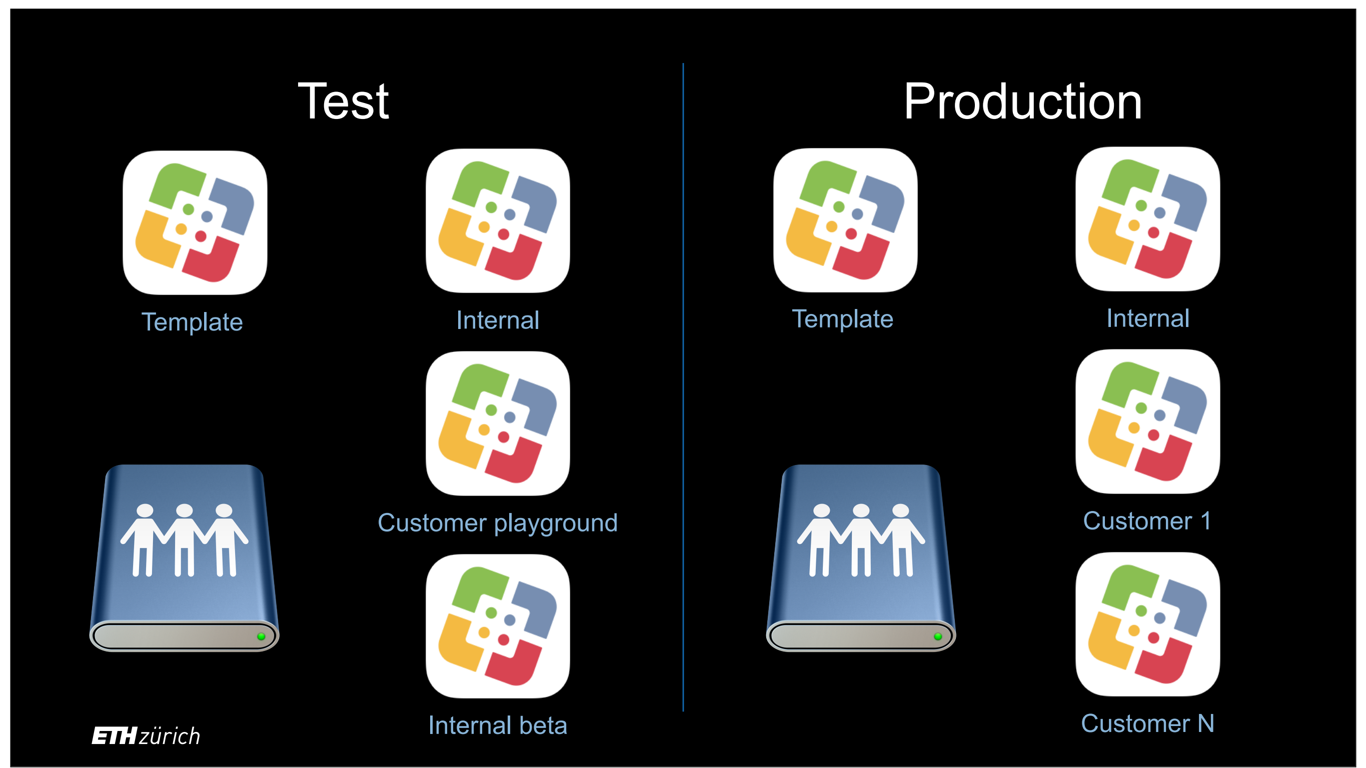
As our software portfolio continued to broaden, even copying policies became more difficult using a script, due to an increasing number of exceptions to the normal dependencies - we started to need to hard-code too many things in to the scriipt.
And shelling out from AutoPkg to a bash script never seemed like the best way to do the job.

Moving away from JSSImporter to using the new JamfUploader processors was an opportunity to look at how I could use the new processors in a way that meant I could run autopkg on each instance separately, rather than handing it over to an external script, meaning that individualities in recipes could be directly implemented across all instances.

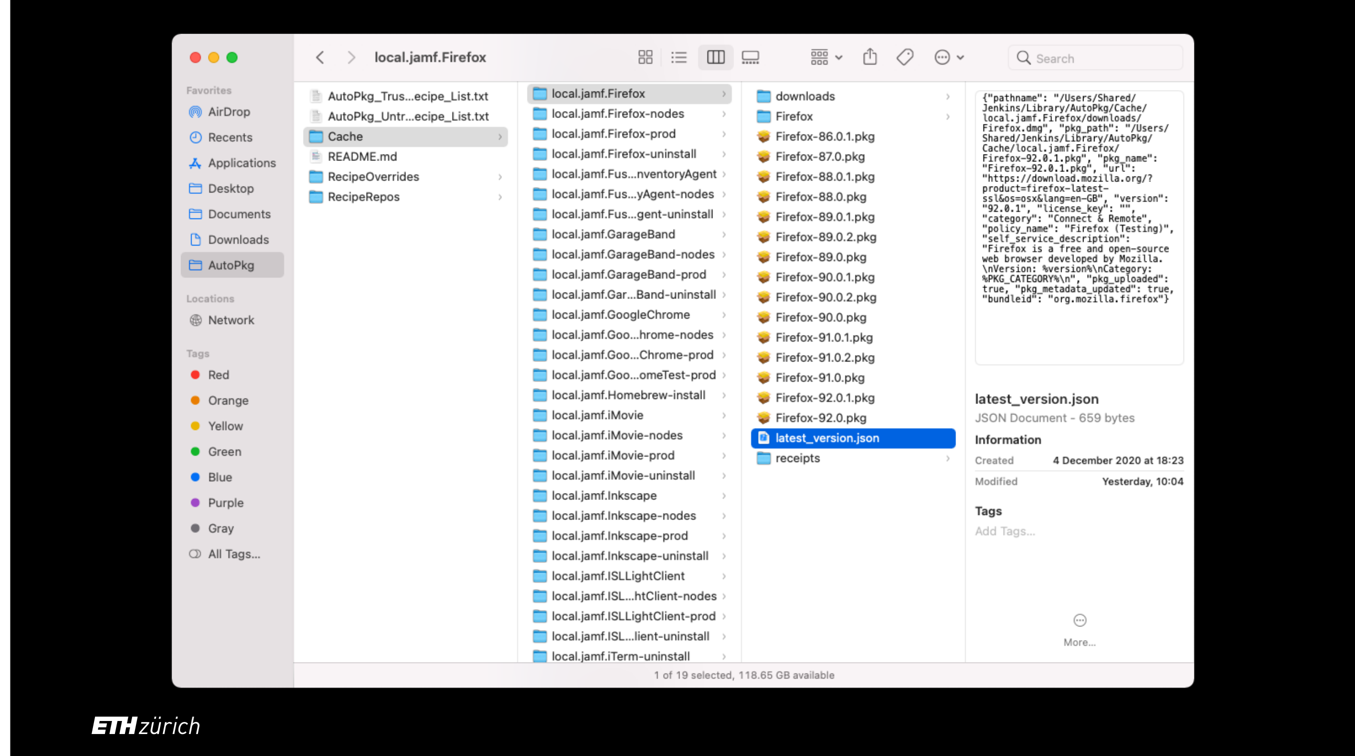
! Firefox-prod.jamf.recipe.yaml U ! Atom.jss-prod.recipe.yaml U ×

id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > ! Atom.jss-prod.recipe.yaml > [ ] Process

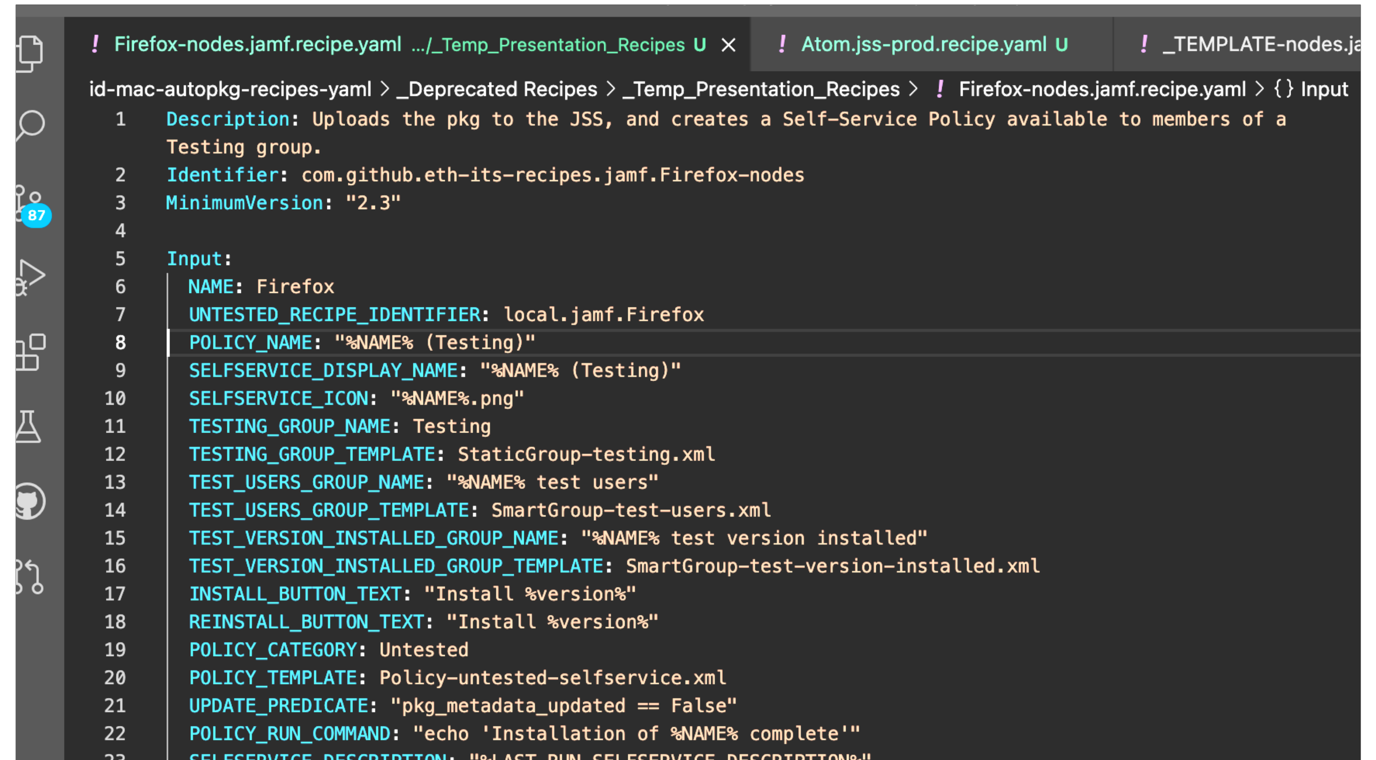
```
52   Process:
53 >   - Processor: ch.ethz.autopkg.commonprocessors/JSSRecipeReceiptChecker…
56
57     - Processor: ch.ethz.autopkg.commonprocessors/CheckSharepointToStage
58
59     - Processor: StopProcessingIf
60       Arguments:
61         predicate: "%PROD_PREDICATE%"
62
63     - Processor: JSSImporter
64 >     Arguments:…
84       Comment: Trigger-only policy
85
86     - Processor: ch.ethz.autopkg.postprocessors/Policytool
87
88     - Processor: JSSImporter
89 >     Arguments:…
105      Comment: Auto-install policy
106
107    - Processor: ch.ethz.autopkg.postprocessors/Policytool
108
109    - Processor: JSSImporter
110 >    Arguments:…
128      Comment: Auto-update policy
129
130    - Processor: ch.ethz.autopkg.postprocessors/Policytool
131
```



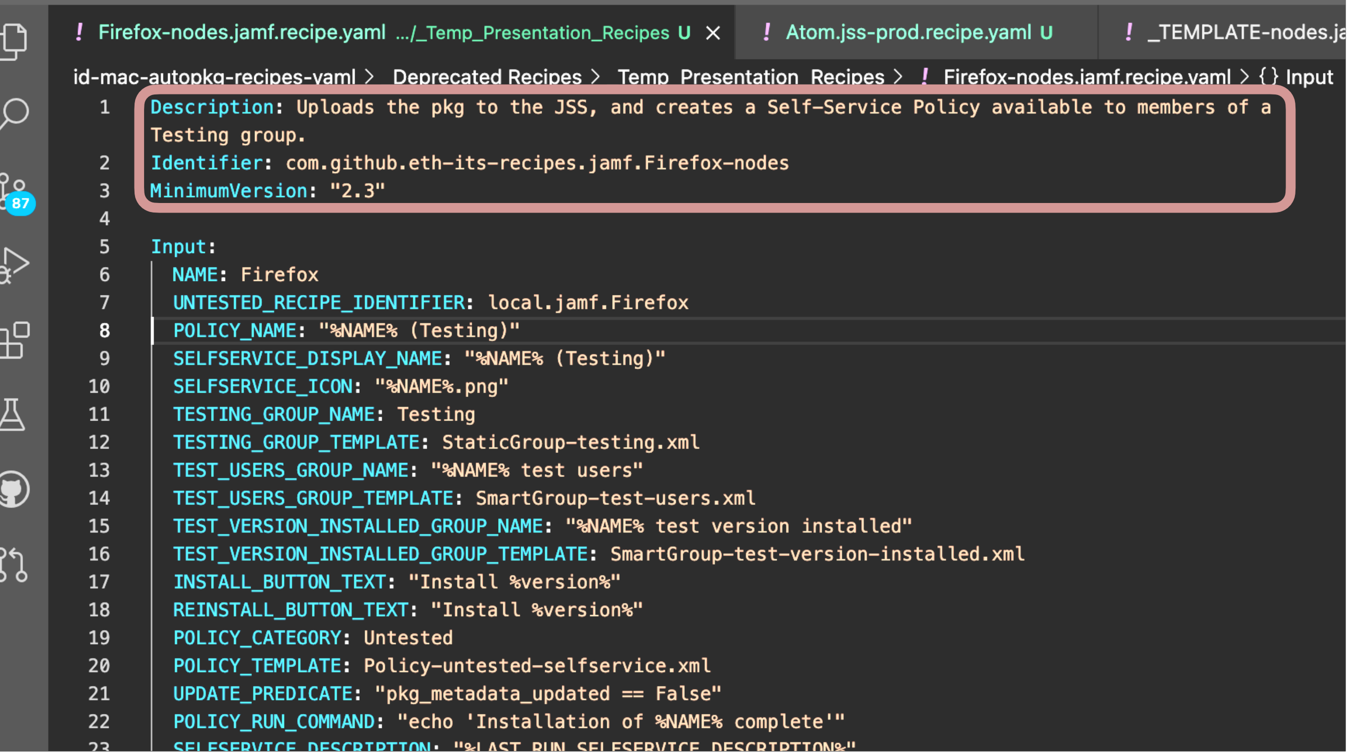
Because we only have one File Share for the test and production systems, we don't want to upload the package again and again when running a recipe on every instance, as it's already there in the repo.
We only need to write the information or metadata about the new package to each instance, as well as updating the smart groups and policies to match what we did in the Template instance.



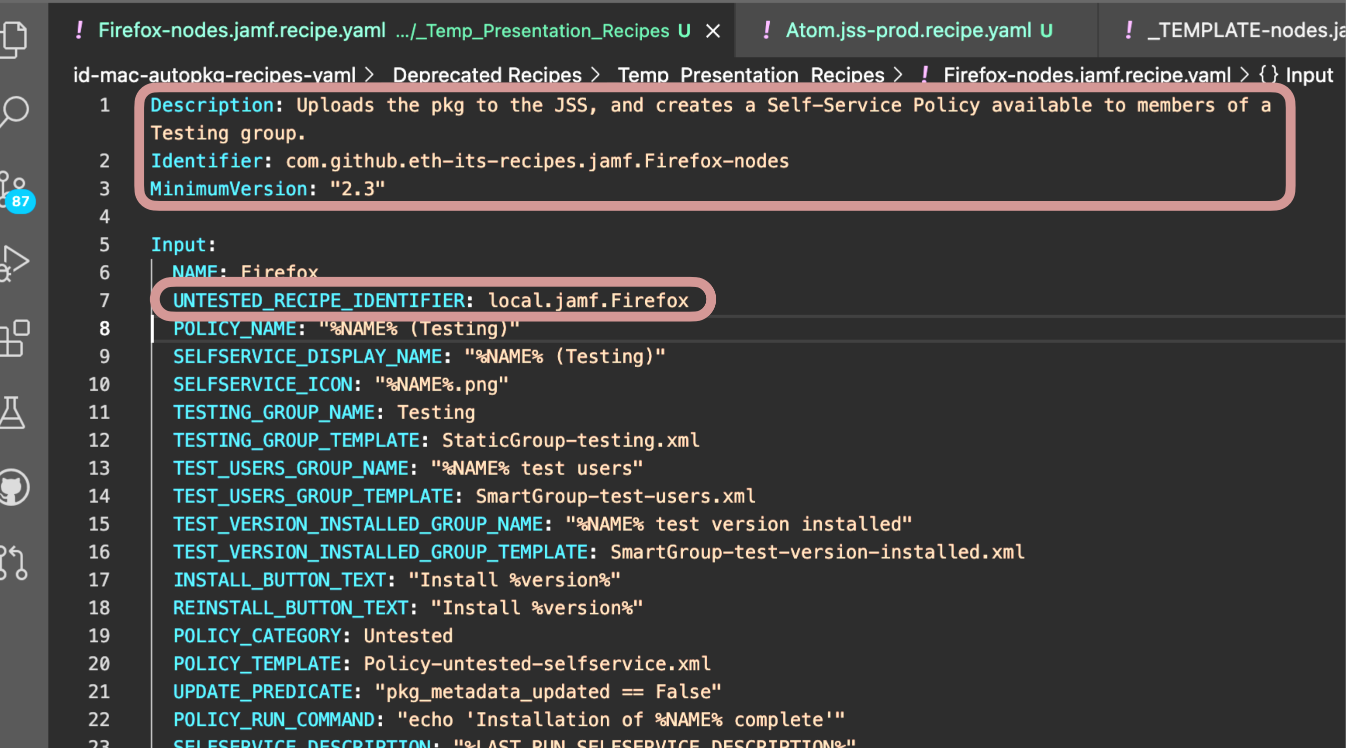
Thanks to the LastRecipeRunResult processor and the JSON file it produces, we already have the information about the package available.

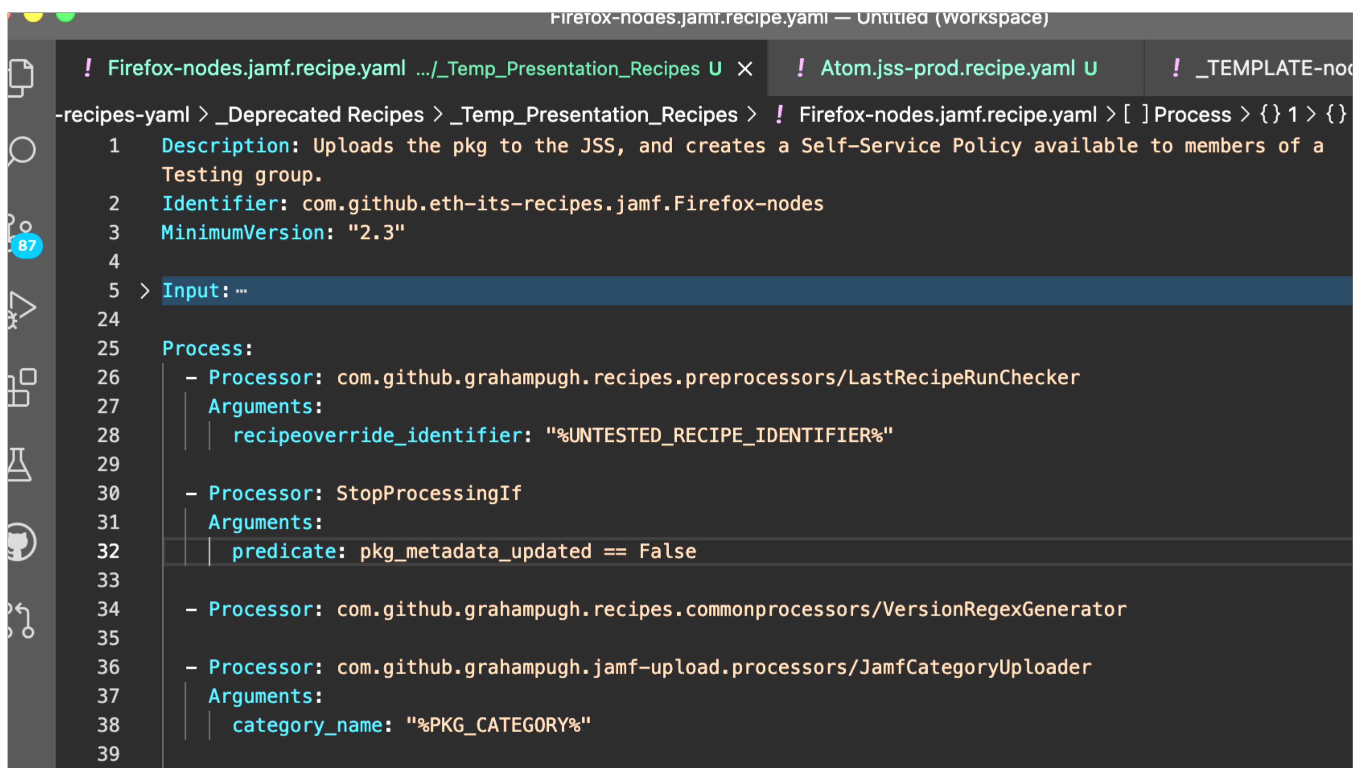


So, I came up with a separate recipe type for running on the customer instances. I call this a "nodes" jamf recipe.
This contains all the same information as a testing jamf recipe, but like the prod jamf recipes, 🍎it has no parent recipe. Instead, 🍎 we supply the identifier of the testing jamf recipe, so that we can get the package information from the correct JSON file.

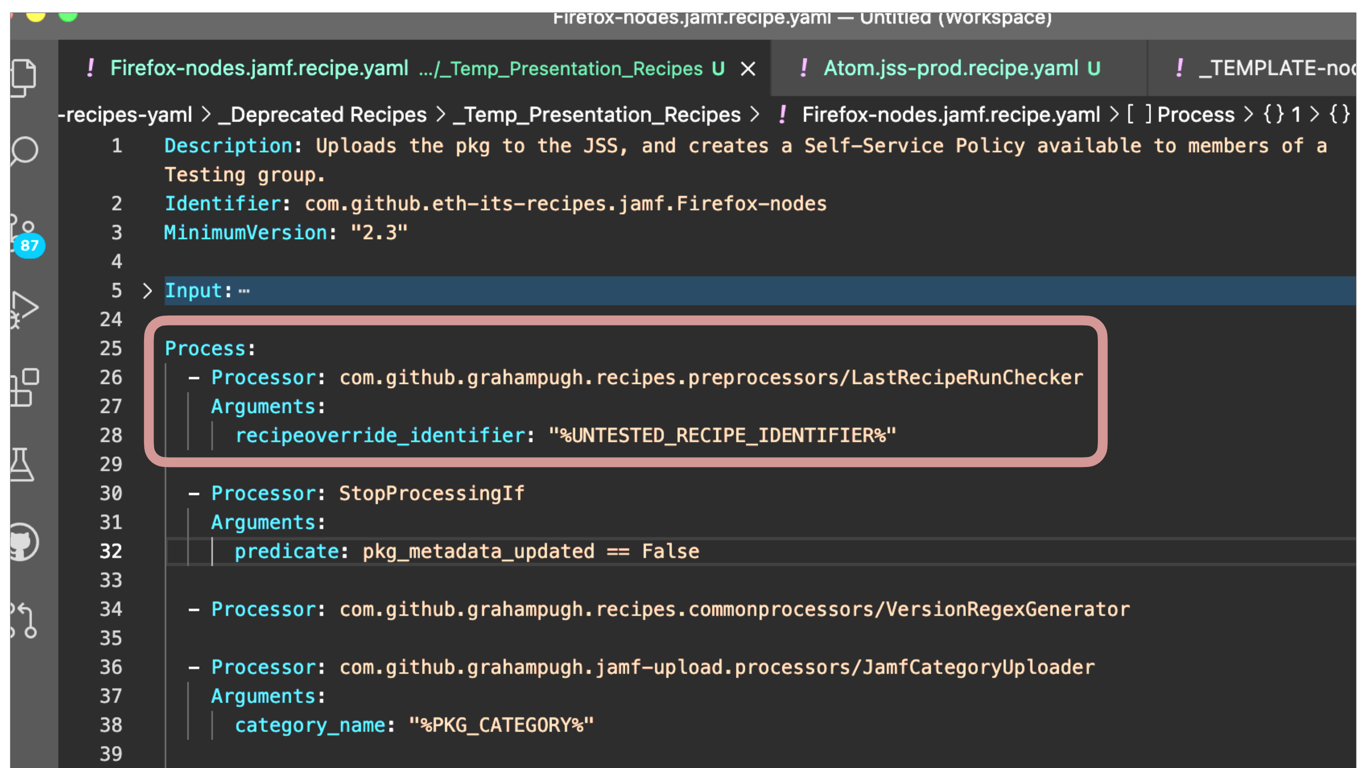


Firefox-nodes.jamf.recipe.yaml .../_Temp_Presentation_Recipes U
Atom.jss-prod.recipe.yaml U
_TEMPLATE-nodes.ja
id-mac-autopkg-recipes-yaml > Deprecated Recipes > Temp_Presentation_Recipes > Firefox-nodes.jamf.recipe.yaml > {} Input
Description: Uploads the pkg to the JSS, and creates a Self-Service Policy available to members of a Testing group.
Identifier: com.github.eth-its-recipes.jamf.Firefox-nodes
MinimumVersion: "2.3"
Input:
NAME: Firefox
UNTESTED_RECIPE_IDENTIFIER: local.jamf.Firefox
POLICY_NAME: "%NAME% (Testing)"
SELFSERVICE_DISPLAY_NAME: "%NAME% (Testing)"
SELFSERVICE_ICON: "%NAME%.png"
TESTING_GROUP_NAME: Testing
TESTING_GROUP_TEMPLATE: StaticGroup-testing.xml
TEST_USERS_GROUP_NAME: "%NAME% test users"
TEST_USERS_GROUP_TEMPLATE: SmartGroup-test-users.xml
TEST_VERSION_INSTALLED_GROUP_NAME: "%NAME% test version installed"
TEST_VERSION_INSTALLED_GROUP_TEMPLATE: SmartGroup-test-version-installed.xml
INSTALL_BUTTON_TEXT: "Install %version%"
REINSTALL_BUTTON_TEXT: "Install %version%"
POLICY_CATEGORY: Untested
POLICY_TEMPLATE: Policy-untested-selfservice.xml
UPDATE_PREDICATE: "pkg_metadata_updated == False"
POLICY_RUN_COMMAND: "echo 'Installation of %NAME% complete'"


Firefox-nodes.jamf.recipe.yaml .../_Temp_Presentation_Recipes U
Atom.jss-prod.recipe.yaml U
id-mac-autopkg-recipes-yaml > Deprecated Recipes > Temp_Presentation_Recipes > Firefox-nodes.jamf.recipe.yaml > {} Input
Description: Uploads the pkg to the JSS, and creates a Self-Service Policy available to members of a Testing group.
Identifier: com.github.eth-its-recipes.jamf.Firefox-nodes
MinimumVersion: "2.3"
Input:
NAME: Firefox
UNTESTED_RECIPE_IDENTIFIER: local.jamf.Firefox
POLICY_NAME: "%NAME% (Testing)"
SELFSERVICE_DISPLAY_NAME: "%NAME% (Testing)"
SELFSERVICE_ICON: "%NAME%.png"
TESTING_GROUP_NAME: Testing
TESTING_GROUP_TEMPLATE: StaticGroup-testing.xml
TEST_USERS_GROUP_NAME: "%NAME% test users"
TEST_USERS_GROUP_TEMPLATE: SmartGroup-test-users.xml
TEST_VERSION_INSTALLED_GROUP_NAME: "%NAME% test version installed"
TEST_VERSION_INSTALLED_GROUP_TEMPLATE: SmartGroup-test-version-installed.xml
INSTALL_BUTTON_TEXT: "Install %version%"
REINSTALL_BUTTON_TEXT: "Install %version%"
POLICY_CATEGORY: Untested
POLICY_TEMPLATE: Policy-untested-selfservice.xml
UPDATE_PREDICATE: "pkg_metadata_updated == False"
POLICY_RUN_COMMAND: "echo 'Installation of %NAME% complete'"



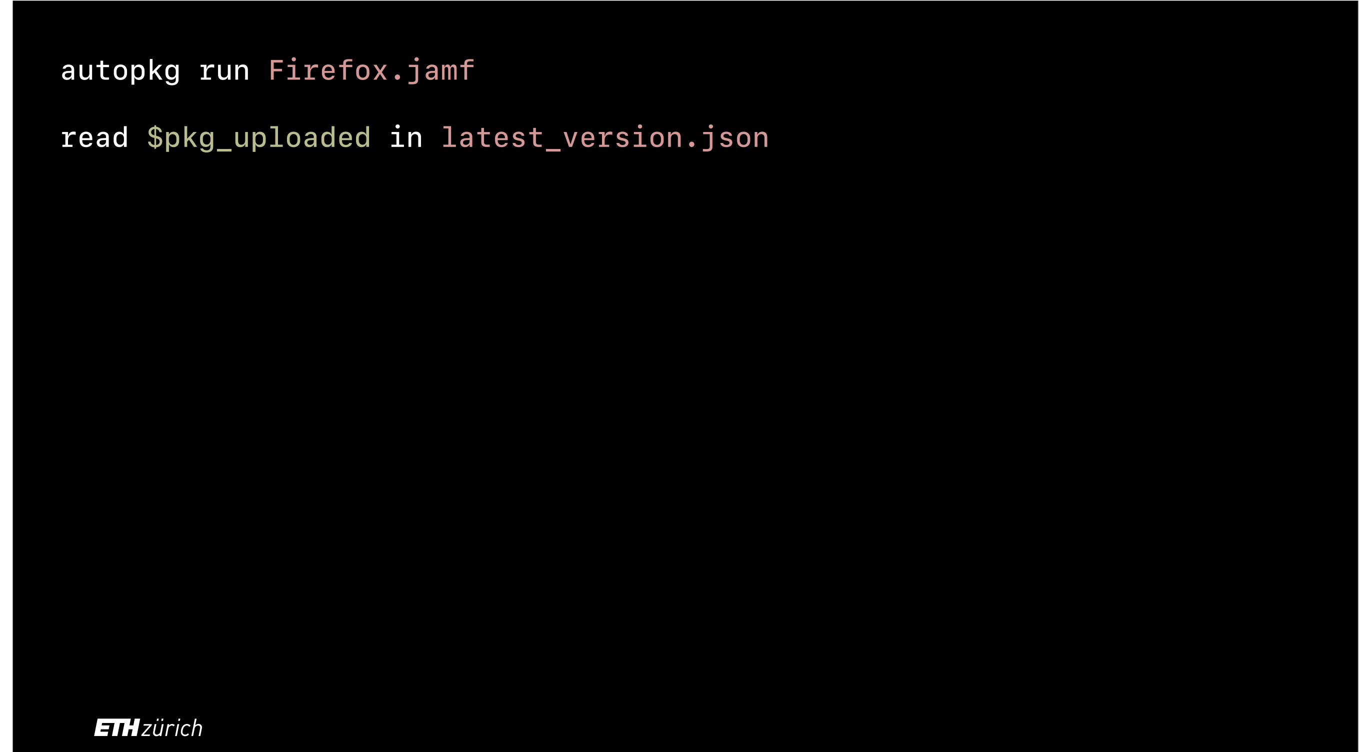
Again, just like the prod jamf recipe, the first process is the LastRecipeRunChecker processor.
The StopProcessingIf process checks the value of the key called pkg_metadata_updated from the JSON file. It's set to true only when a package has just been updated. We do this because it speeds up the recipe run, because if the package metadata wasn't updated on the template instance when the testing jamf recipe was run, chances are there's nothing else to change, so we can stop.


Firefox-nodes.jamf.recipe.yaml — Untitled (Workspace)
Firefox-nodes.jamf.recipe.yaml .../_Temp_Presentation_Recipes U
Atom.jss-prod.recipe.yaml U
-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > Firefox-nodes.jamf.recipe.yaml > [ ] Process > { } 1 > { }
Description: Uploads the pkg to the JSS, and creates a Self-Service Policy available to members of a Testing group.
Identifier: com.github.eth-its-recipes.jamf.Firefox-nodes
MinimumVersion: "2.3"
Input:
Process:
- Processor: com.github.grahampugh.recipes.preprocessors/LastRecipeRunChecker
Arguments:
recipeoverride_identifier: "%UNTESTED_RECIPE_IDENTIFIER%"
- Processor: StopProcessingIf
Arguments:
predicate: pkg_metadata_updated == False
- Processor: com.github.grahampugh.recipes.commonprocessors/VersionRegexGenerator
- Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader
Arguments:
category_name: "%PKG_CATEGORY%"

To make it possible to run AutoPkg on all the instances, we use a wrapper script to allow us to override the autopkg preferences key for the JSS server URL for each instance, and therefore run autopkg on all of the nodes in a loop.

In pseudo code, our workflow looks kind of like this:

The script runs the testing jamf recipe first



Then, to prevent unnecessary recipe runs, we check the JSON file to see if a package was uploaded or not.

```
autopkg run Firefox.jamf

read $pkg_uploaded in latest_version.json

if $pkg_uploaded is True:

    nodes_list=(
        internal
        customer_1
        customer_2
        customer_3
    )
```

ETH zürich

If it was, we supply a list of the names of the nodes to iterate through,

```
autopkg run Firefox.jamf

read $pkg_uploaded in latest_version.json

if $pkg_uploaded is True:

    nodes_list=(
        internal
        customer_1
        customer_2
        customer_3
    )

    for node in $nodes_list:
        autopkg run Firefox-nodes.jamf \
            --key JSS_URL="https://server:8443/$node"
```

ETH zürich

And then run autopkg on each instance using a for loop, overriding the JSS_URL key with the correct URL each time.

```
nodes_list=(
    template
    internal
    customer_1
    customer_2
    customer_3
)

for node in $nodes_list:
    autopkg run Firefox-nodes.jamf \
        --key JSS_URL="https://server:8443/$node"
```

ETH zürich

For our prod recipes, the recipe has no parent and doesn't upload the package, and also checks whether we are ready to stage to production or not, right there in the recipe, so we could just iterate through the template and all the nodes, running the recipe on each.

However, to speed things up, the wrapper script first runs the recipe on the template instance only.

Then, it reads the receipt of the recipe that just ran, to look for the result of the StopProcessingIf predicate.
If it finds that the value was true, then the wrapper script exits, preventing the recipe from running unnecessarily on all the other instances.

```
autopkg run Firefox-prod.jamf

read $stop_processing_recipe from StopProcessingIf in latest_receipt

if $stop_processing_recipe is False:

    nodes_list=(
        internal
        customer_1
        customer_2
        customer_3
    )

for node in $nodes_list:
    autopkg run Firefox-prod.jamf \
        --key JSS_URL="https://server:8443/$node"
done
```

ETH zürich

But, if the predicate was not matched, then we can supply the instance names and run the recipe on all of them.

Jamf policies

- Firefox (Testing)

- Firefox
- Update Firefox
- Auto-install Firefox
- Auto-update Firefox
- Install Firefox
- **Uninstall Firefox**
- **Trigger-uninstall Firefox**

Jamf smart groups

- Firefox test users
- Firefox test version installed

- Firefox users
- Firefox auto-install
- Firefox auto-update
- Firefox current version installed
- Firefox installed

ETH zürich

I didn't talk about our uninstall policies yet, but these are also important to our customers, because when they provide their users with an application via Self Service, the expectation is that it can be installed by a standard user.

If that's the case, then the app should also be able to be uninstalled by a standard user. So, we need to provide properly scoped uninstaller policies to go in Self Service to achieve that, and just like the other policies, we need to automate their distribution.

So, we use recipes for this too.

```
! Firefox-uninstall.jamf.recipe.yaml .../_Temp_Presentation_Recipes U ×   ! Atom.jss-prod.recipe.yaml U   ! _TEMPLATE-nod
id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > ! Firefox-uninstall.jamf.recipe.yaml > Mi
1  Description: Uploads a script to the Jamf Pro Server and creates a Self Service Policy and Smart Group.
2  Identifier: com.github.eth-its-recipes.jamf.Firefox-uninstall
3  MinimumVersion: "2.3"
4
5  Input:
6    NAME: Firefox
7    JSS_INVENTORY_NAME: "%NAME%.app"
8    SCRIPT_NAME: Application-uninstall.sh
9    SCRIPT_PATH: Application-uninstall.sh
10   PARAMETER4_LABEL: "Application Name"
11   PARAMETER5_LABEL: Package Receipt
12   PARAMETER6_LABEL: "Parameter 6"
13   PARAMETER7_LABEL: "Parameter 7"
14   PARAMETER8_LABEL: "Parameter 8"
15   PARAMETER9_LABEL: "Parameter 9"
16   PARAMETER10_LABEL: "Parameter 10"
17   PARAMETER11_LABEL: "Parameter 11"
18   PARAMETER4_VALUE: "%NAME%"
19   PARAMETER5_VALUE: org.mozilla.firefox.pkg
20   PARAMETER6_VALUE: "None"
21   PARAMETER7_VALUE: "None"
22   PARAMETER8_VALUE: "None"
23   PARAMETER9_VALUE: "None"
```

A jamf uninstaller policy rarely uses an uninstaller package. Normally we use a script to perform the uninstallation.

We can use the same script for many uninstaller policies, since a lot of apps are just an application bundle in the Applications folder, so the process of removing it is the same except for the name of the app, and the package receipt name if you want to remove that too.

In such a recipe, we just supply 🍎the name and path of the common script we want to use, 🍎 any parameter labels we would like to add to the script, and the values of those parameters we want to set in the policy, in this case the 🍎app name, and 🍎 the package receipt's name.

! Firefox-uninstall.jamf.recipe.yaml .../_Temp_Presentation_Recipes U × ! Atom.jss-prod.recipe.yaml U ! _TEMPLATE-nod

id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > ! Firefox-uninstall.jamf.recipe.yaml > Mi

```
1  Description: Uploads a script to the Jamf Pro Server and creates a Self Service Policy and Smart Group.
2  Identifier: com.github.eth-its-recipes.jamf.Firefox-uninstall
3  MinimumVersion: "2.3"
4
5  Input:
6    NAME: Firefox
7    JSS_INVENTORY_NAME: "%NAME%.app"
8    SCRIPT_NAME: Application-uninstall.sh
9    SCRIPT_PATH: Application-uninstall.sh
10   PARAMETER4_LABEL: "Application Name"
11   PARAMETER5_LABEL: Package Receipt
12   PARAMETER6_LABEL: "Parameter 6"
13   PARAMETER7_LABEL: "Parameter 7"
14   PARAMETER8_LABEL: "Parameter 8"
15   PARAMETER9_LABEL: "Parameter 9"
16   PARAMETER10_LABEL: "Parameter 10"
17   PARAMETER11_LABEL: "Parameter 11"
18   PARAMETER4_VALUE: "%NAME%"
19   PARAMETER5_VALUE: org.mozilla.firefox.pkg
20   PARAMETER6_VALUE: "None"
21   PARAMETER7_VALUE: "None"
22   PARAMETER8_VALUE: "None"
23   PARAMETER9_VALUE: "None"
```

! Firefox-uninstall.jamf.recipe.yaml .../_Temp_Presentation_Recipes U × ! Atom.jss-prod.recipe.yaml U ! _TEMPLATE-nod

id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > ! Firefox-uninstall.jamf.recipe.yaml > Mi

```
1   Description: Uploads a script to the Jamf Pro Server and creates a Self Service Policy and Smart Group.
2   Identifier: com.github.eth-its-recipes.jamf.Firefox-uninstall
3   MinimumVersion: "2.3"
4
5   Input:
6     NAME: Firefox
7     JSS_INVENTORY_NAME: "%NAME%.app"
8     SCRIPT_NAME: Application-uninstall.sh
9     SCRIPT_PATH: Application-uninstall.sh
10    PARAMETER4_LABEL: "Application Name"
11    PARAMETER5_LABEL: Package Receipt
12    PARAMETER6_LABEL: "Parameter 6"
13    PARAMETER7_LABEL: "Parameter 7"
14    PARAMETER8_LABEL: "Parameter 8"
15    PARAMETER9_LABEL: "Parameter 9"
16    PARAMETER10_LABEL: "Parameter 10"
17    PARAMETER11_LABEL: "Parameter 11"
18    PARAMETER4_VALUE: "%NAME%"
19    PARAMETER5_VALUE: org.mozilla.firefox.pkg
20    PARAMETER6_VALUE: "None"
21    PARAMETER7_VALUE: "None"
22    PARAMETER8_VALUE: "None"
23    PARAMETER9_VALUE: "None"
```

! Firefox-uninstall.jamf.recipe.yaml .../_Temp_Presentation_Recipes U × ! Atom.jss-prod.recipe.yaml U ! _TEMPLATE-nod

id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > ! Firefox-uninstall.jamf.recipe.yaml > Mi

```
1  Description: Uploads a script to the Jamf Pro Server and creates a Self Service Policy and Smart Group.
2  Identifier: com.github.eth-its-recipes.jamf.Firefox-uninstall
3  MinimumVersion: "2.3"
4
5  Input:
6    NAME: Firefox
7    JSS_INVENTORY_NAME: "%NAME%.app"
8    SCRIPT_NAME: Application-uninstall.sh
9    SCRIPT_PATH: Application-uninstall.sh
10   PARAMETER4_LABEL: "Application Name"
11   PARAMETER5_LABEL: Package Receipt
12   PARAMETER6_LABEL: "Parameter 6"
13   PARAMETER7_LABEL: "Parameter 7"
14   PARAMETER8_LABEL: "Parameter 8"
15   PARAMETER9_LABEL: "Parameter 9"
16   PARAMETER10_LABEL: "Parameter 10"
17   PARAMETER11_LABEL: "Parameter 11"
18   PARAMETER4_VALUE: "%NAME%"
19   PARAMETER5_VALUE: org.mozilla.firefox.pkg
20   PARAMETER6_VALUE: "None"
21   PARAMETER7_VALUE: "None"
22   PARAMETER8_VALUE: "None"
23   PARAMETER9_VALUE: "None"
```

! Firefox-uninstall.jamf.recipe.yaml .../_Temp_Presentation_Recipes U × ! Atom.jss-prod.recipe.yaml U ! _TEMPLATE-nod

id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > ! Firefox-uninstall.jamf.recipe.yaml > Mi

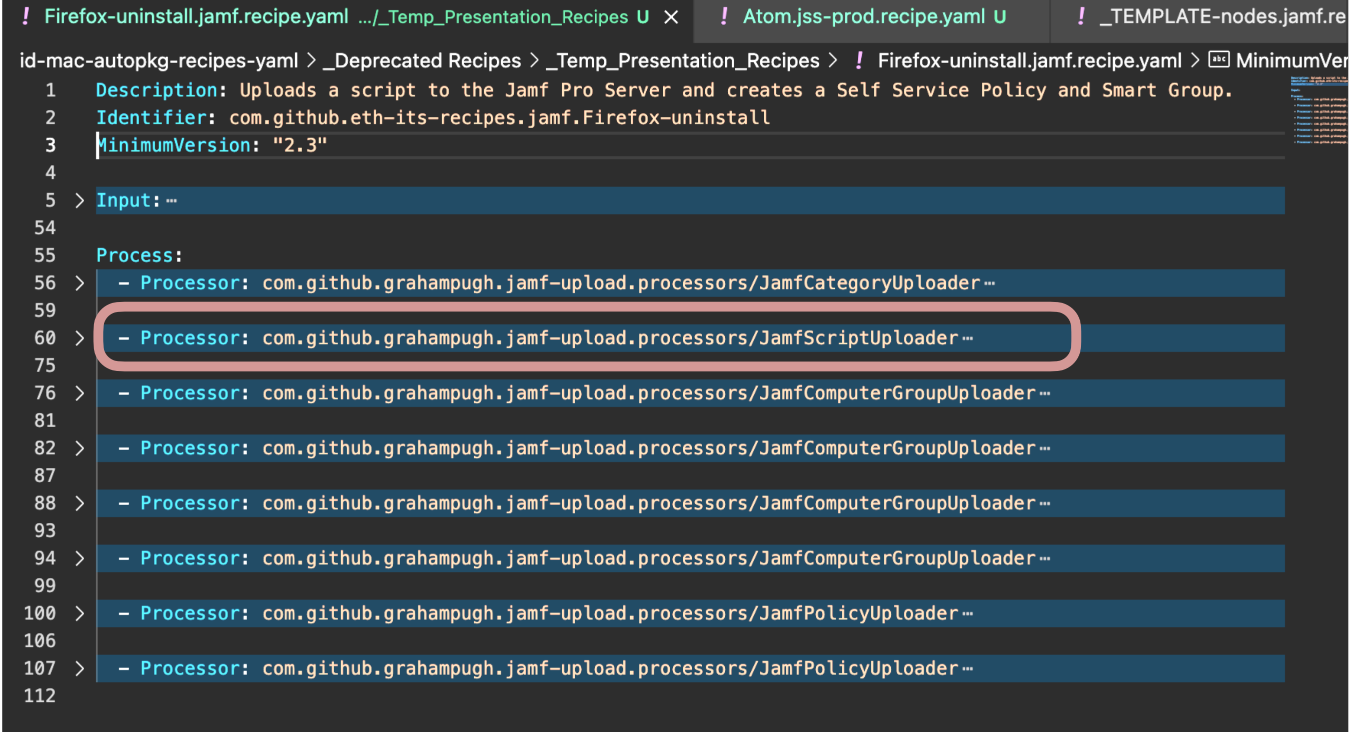
```yaml
1  Description: Uploads a script to the Jamf Pro Server and creates a Self Service Policy and Smart Group.
2  Identifier: com.github.eth-its-recipes.jamf.Firefox-uninstall
3  MinimumVersion: "2.3"
4
5  Input:
6    NAME: Firefox
7    JSS_INVENTORY_NAME: "%NAME%.app"
8    SCRIPT_NAME: Application-uninstall.sh
9    SCRIPT_PATH: Application-uninstall.sh
10   PARAMETER4_LABEL: "Application Name"
11   PARAMETER5_LABEL: Package Receipt
12   PARAMETER6_LABEL: "Parameter 6"
13   PARAMETER7_LABEL: "Parameter 7"
14   PARAMETER8_LABEL: "Parameter 8"
15   PARAMETER9_LABEL: "Parameter 9"
16   PARAMETER10_LABEL: "Parameter 10"
17   PARAMETER11_LABEL: "Parameter 11"
18   PARAMETER4_VALUE: "%NAME%"
19   PARAMETER5_VALUE: org.mozilla.firefox.pkg
20   PARAMETER6_VALUE: "None"
21   PARAMETER7_VALUE: "None"
22   PARAMETER8_VALUE: "None"
23   PARAMETER9_VALUE: "None"
```

! Firefox-uninstall.jamf.recipe.yaml .../_Temp_Presentation_Recipes U × ! Atom.jss-prod.recipe.yaml U ! _TEMPLATE-nodes.jamf.re

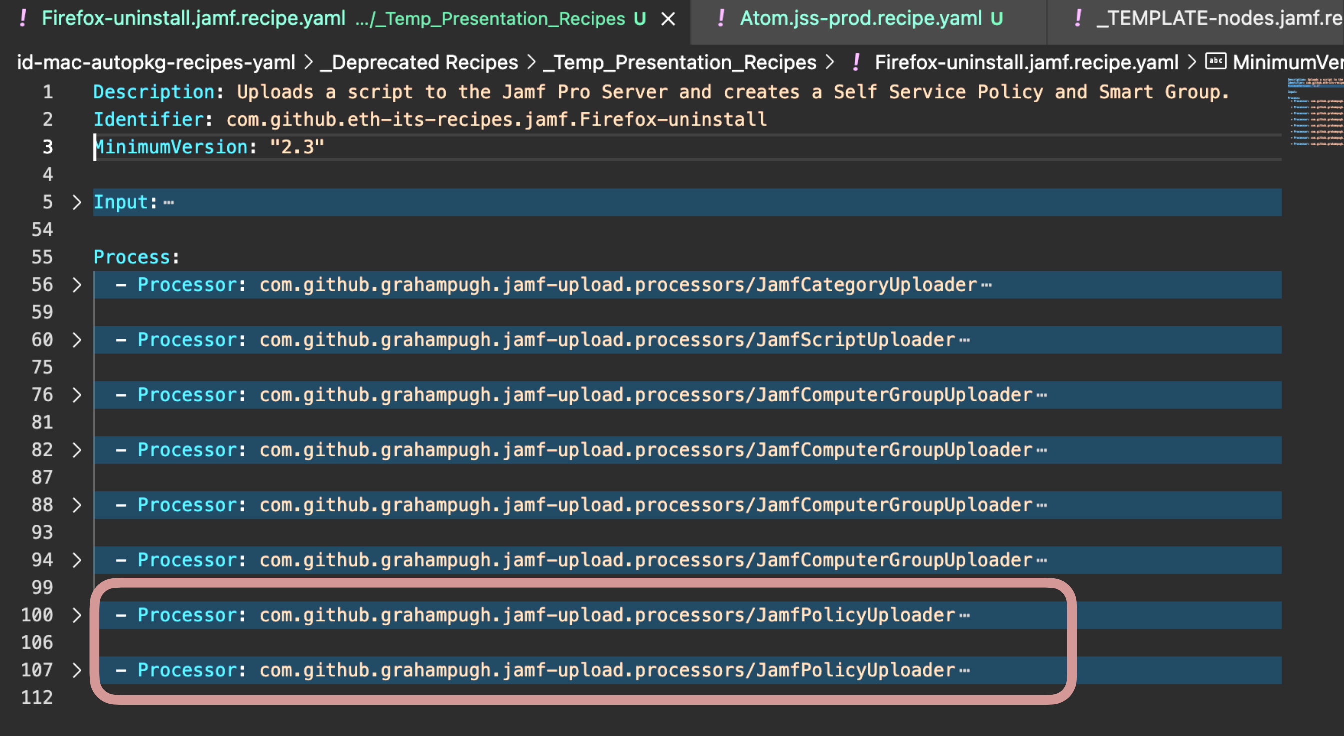
id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > ! Firefox-uninstall.jamf.recipe.yaml > MinimumVer

```
1    Description: Uploads a script to the Jamf Pro Server and creates a Self Service Policy and Smart Group.
2    Identifier: com.github.eth-its-recipes.jamf.Firefox-uninstall
3    MinimumVersion: "2.3"
4
5  > Input:…
54
55   Process:
56 >   - Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader…
59
60 >   - Processor: com.github.grahampugh.jamf-upload.processors/JamfScriptUploader…
75
76 >   - Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader…
81
82 >   - Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader…
87
88 >   - Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader…
93
94 >   - Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader…
99
100 >  - Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader…
106
107 >  - Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader…
112
```

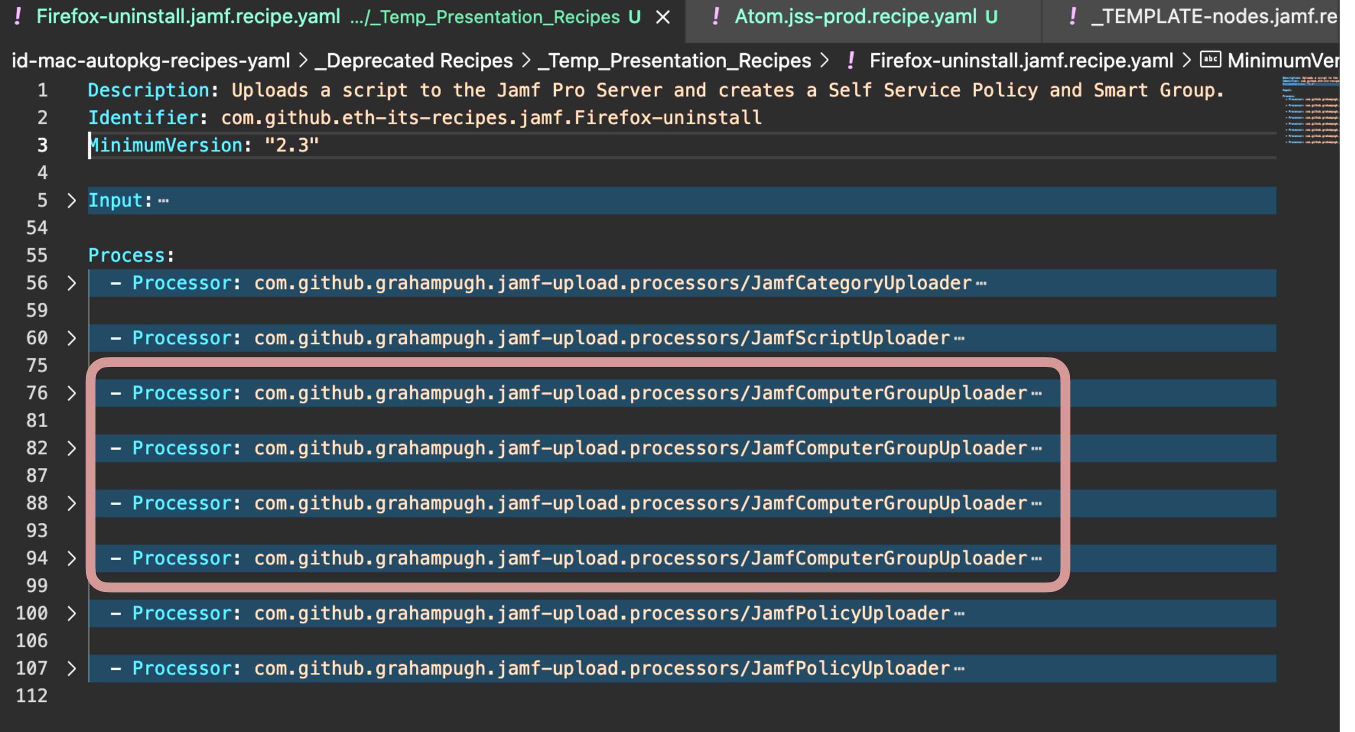
The processes in such a recipe 🍎 create or update the script, 🍎the policies, and 🍎any dependent smart groups.



Firefox-uninstall.jamf.recipe.yaml .../_Temp_Presentation_Recipes U
Atom.jss-prod.recipe.yaml U
_TEMPLATE-nodes.jamf.re
id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > Firefox-uninstall.jamf.recipe.yaml > MinimumVer
1 Description: Uploads a script to the Jamf Pro Server and creates a Self Service Policy and Smart Group.
2 Identifier: com.github.eth-its-recipes.jamf.Firefox-uninstall
3 MinimumVersion: "2.3"
4
5 Input:
54
55 Process:
56 - Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader
59
60 - Processor: com.github.grahampugh.jamf-upload.processors/JamfScriptUploader
75
76 - Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader
81
82 - Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader
87
88 - Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader
93
94 - Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader
99
100 - Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader
106
107 - Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader
112



Firefox-uninstall.jamf.recipe.yaml .../_Temp_Presentation_Recipes U
Atom.jss-prod.recipe.yaml U
_TEMPLATE-nodes.jamf.re
id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > Firefox-uninstall.jamf.recipe.yaml > MinimumVer
Description: Uploads a script to the Jamf Pro Server and creates a Self Service Policy and Smart Group.
Identifier: com.github.eth-its-recipes.jamf.Firefox-uninstall
MinimumVersion: "2.3"
Input:
Process:
- Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfScriptUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader


Firefox-uninstall.jamf.recipe.yaml .../_Temp_Presentation_Recipes U
Atom.jss-prod.recipe.yaml U
_TEMPLATE-nodes.jamf.re
id-mac-autopkg-recipes-yaml > _Deprecated Recipes > _Temp_Presentation_Recipes > Firefox-uninstall.jamf.recipe.yaml > MinimumVer
Description: Uploads a script to the Jamf Pro Server and creates a Self Service Policy and Smart Group.
Identifier: com.github.eth-its-recipes.jamf.Firefox-uninstall
MinimumVersion: "2.3"
Input:
Process:
- Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfScriptUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader
- Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader

```
nodes_list=(
    template
    internal
    customer_1
    customer_2
    customer_3
)

for node in $nodes_list:
    autopkg run Firefox-uninstall.jamf \
        --key JSS_URL="https://server:8443/$node"
```

ETH zürich

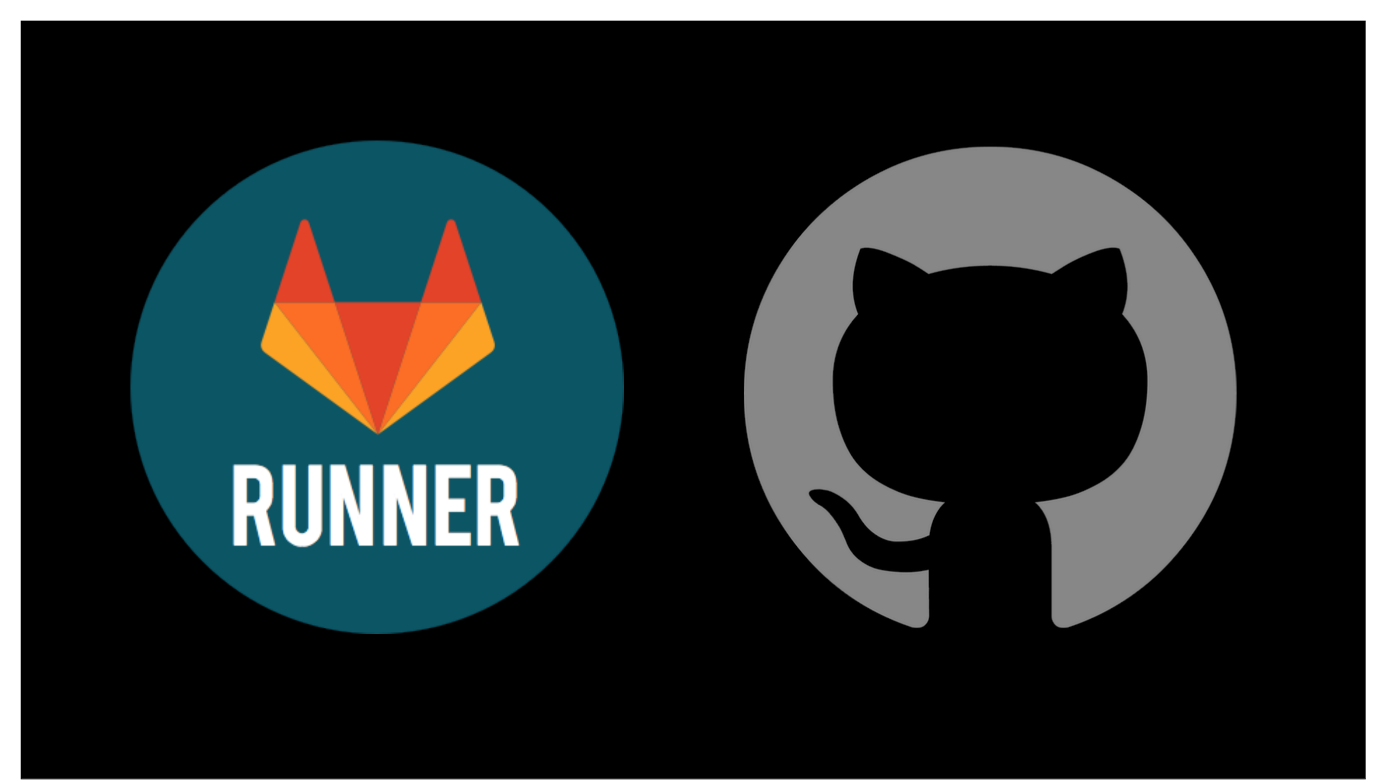
As the uninstaller recipes have no parent, and we don't need to update them regularly, we can just run our wrapper script manually when we need to make a change to an uninstaller. It doesn't need to be scheduled.

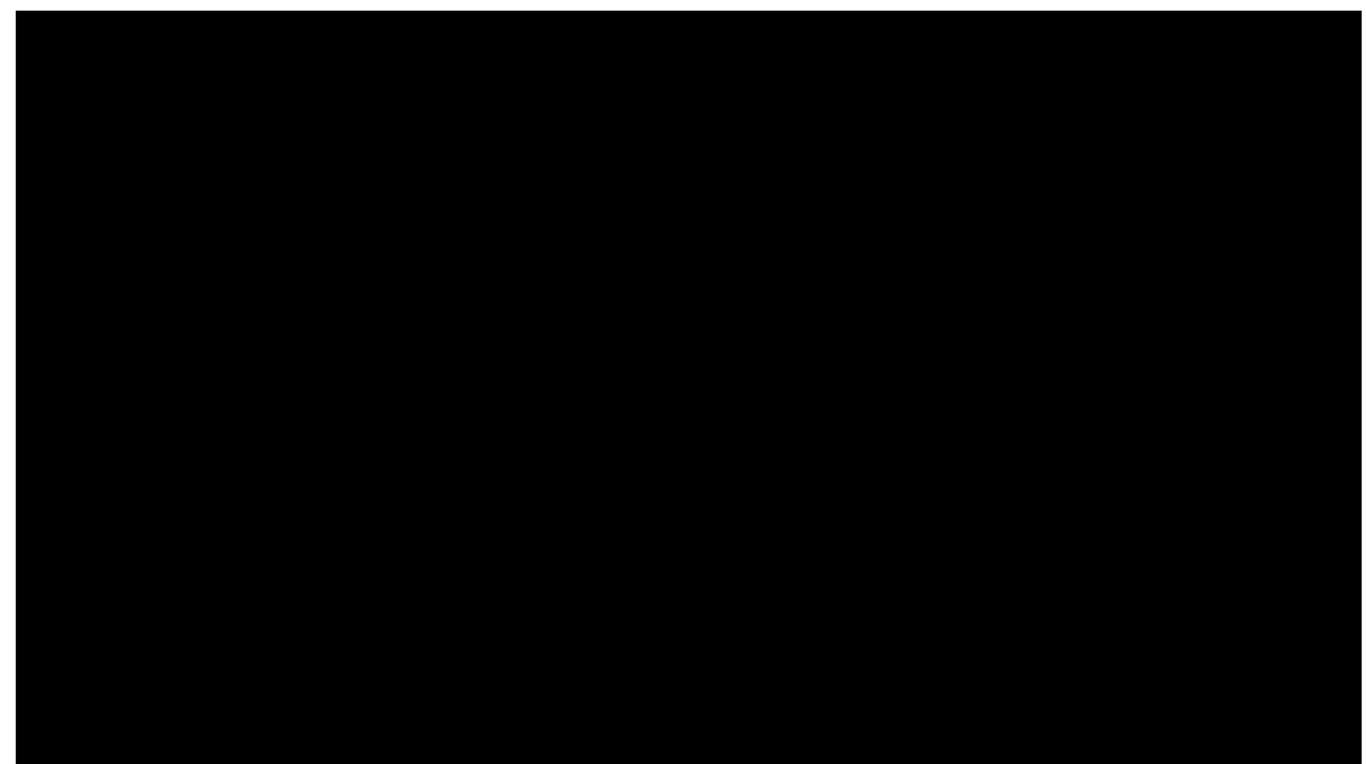
So, this is the simplest of our wrapper scripts - it can just iterate through the template and customer instances without any checks.

To schedule the verification and running of our testing, nodes and prod jamf recipes, we use GitLab Runner to manage the scheduling. We use this because it is available to us on premises, but I know that a lot of people use 🍎 **GitHub's** runner, and I believe it works in a very similar way.

These tools are most often used for compiling and building software, but they include some useful features for running any scheduled task.
One of the most useful features for us is the opportunity to store secrets securely in the repo that we need to supply to autopkg, particularly the API account details for connecting with Jamf and SharePoint.

If you want to know more about how we do this, I wrote a blog post about it some time ago, and I will include a link to this in the presentation links.


RUNNER



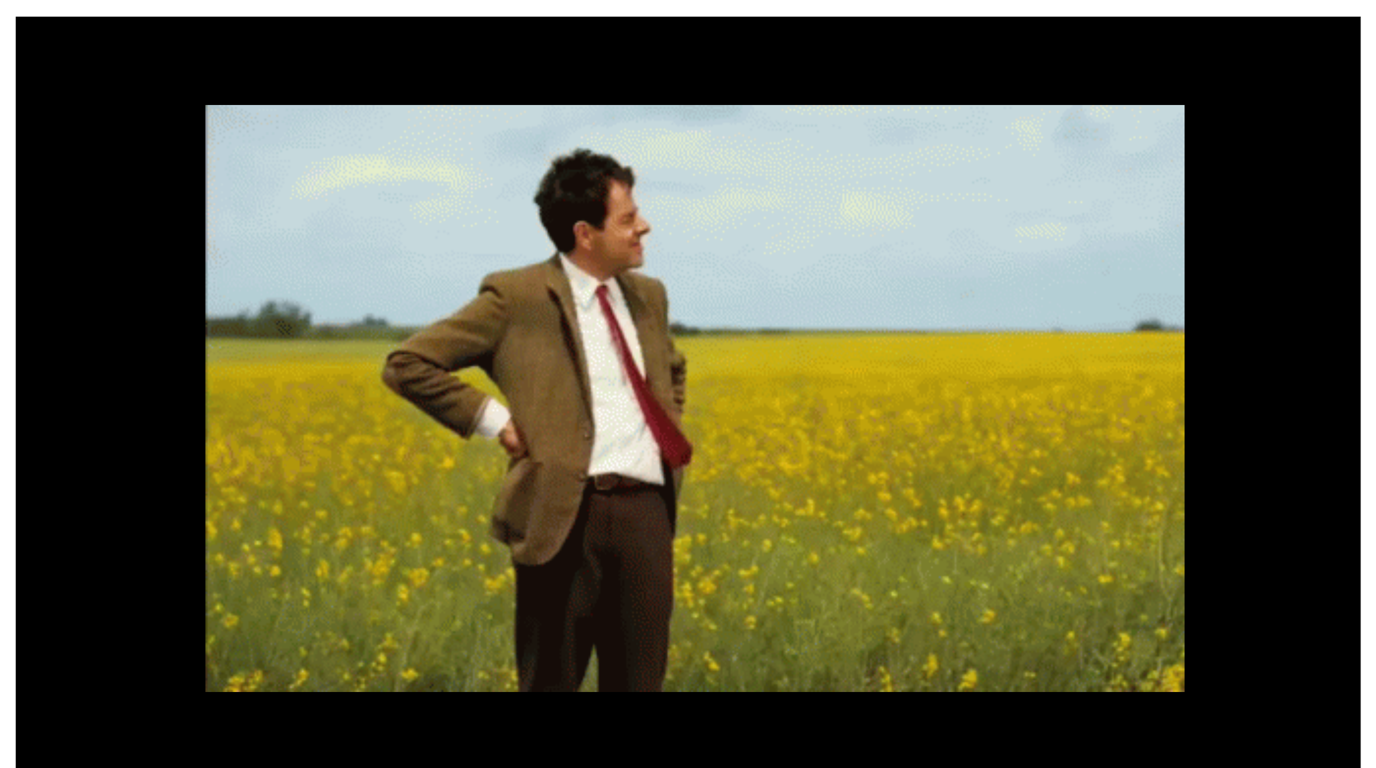
I do have to mention that running this number of AutoPkg recipes daily has brought some challenges.

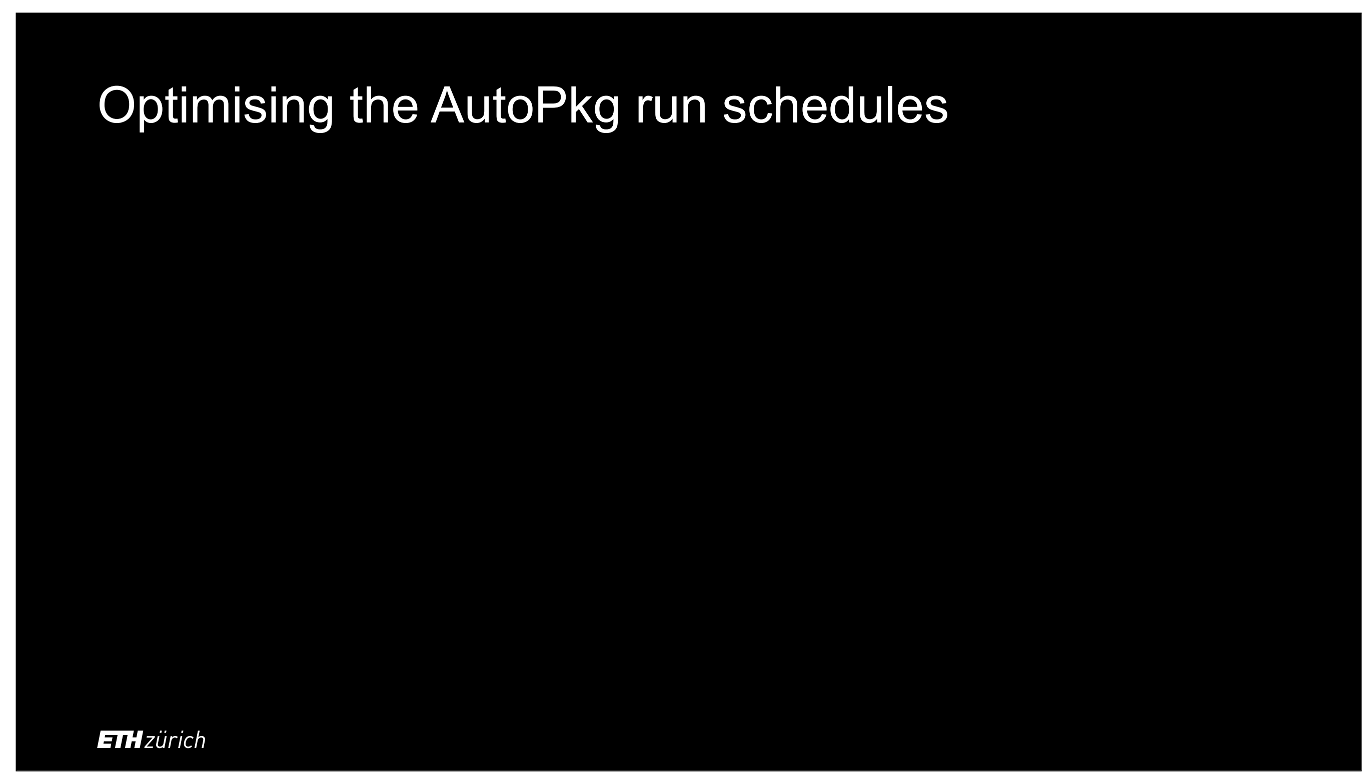
When I moved away from using my shell script to distribute policies to all the instances, to running the autopkg recipe on every instance, the total length of the jobs increased massively.

In fact, including the additional steps we make to verify trust, the total run time started easily exceeding 24h, so I couldn't even maintain a daily schedule anymore.

I suppose it was never envisaged that you would scale up to thousands of autopkg commands in one scheduled job, which we are now doing with over 100 software titles, and the nodes and prod recipes running 35 times each.

That's one of the risks of using a framework beyond its conceived use cases.





I still think it has been worth it, and I have got some way towards optimisations that are speeding up the jobs.

These include:

★ Various tricks in the wrapper scripts, some that I showed earlier, to prevent recipes from running on all instances when we can determine externally that there's nothing to update
★ Taking out software from the main recipe lists that is less important or infrequently updated - it's not really worth running a recipe every night when you know from experience that it's only updated once or twice a year. However, I haven't yet found a way to set multiple schedules using GitLab Runner, so it requires a bit of extra scripting to achieve.
★ I have found out that the length of time taken for AutoPkg to prepare before a run really gets going is heavily influenced by the number of repos you have added to your Recipe Search List. This can have an influence of more than a minute per single autopkg run command. But our prod and nodes recipes don't have any parents, and they are all in the same single repo. So, I started to use a separate prefs file for these runs, which does not contain any other repos than ours. This had a huge effect on the total run time,
★ Finally, I did some tests on our first M1 hardware, and autopkg runs were twice as fast as on the Intel T2 Mac minis we are currently using. So, we are migrating to M1 minis imminently.

Optimising the AutoPkg run schedules

- Preventing unnecessary recipe runs via the wrapper scripts

Optimising the AutoPkg run schedules

- Preventing unnecessary recipe runs via the wrapper scripts
- Separate recipe-lists and schedules for critical, frequently updated and infrequently updated software

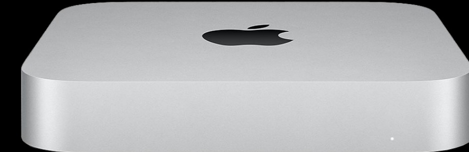
ETH zürich

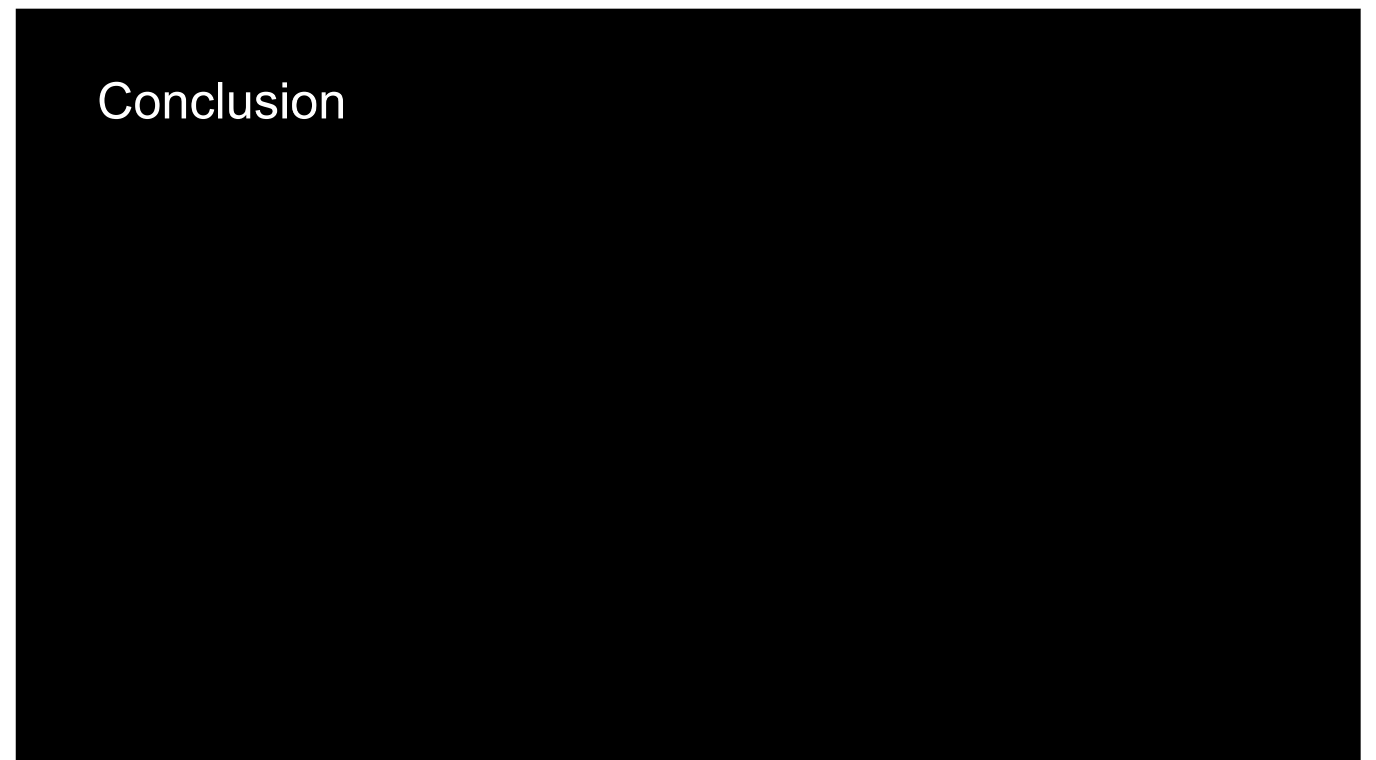
Optimising the AutoPkg run schedules

- Preventing unnecessary recipe runs via the wrapper scripts
- Separate recipe-lists and schedules for critical, frequently updated and infrequently updated software
- Rationalised preferences file for -prod and -nodes recipes

Optimising the AutoPkg run schedules

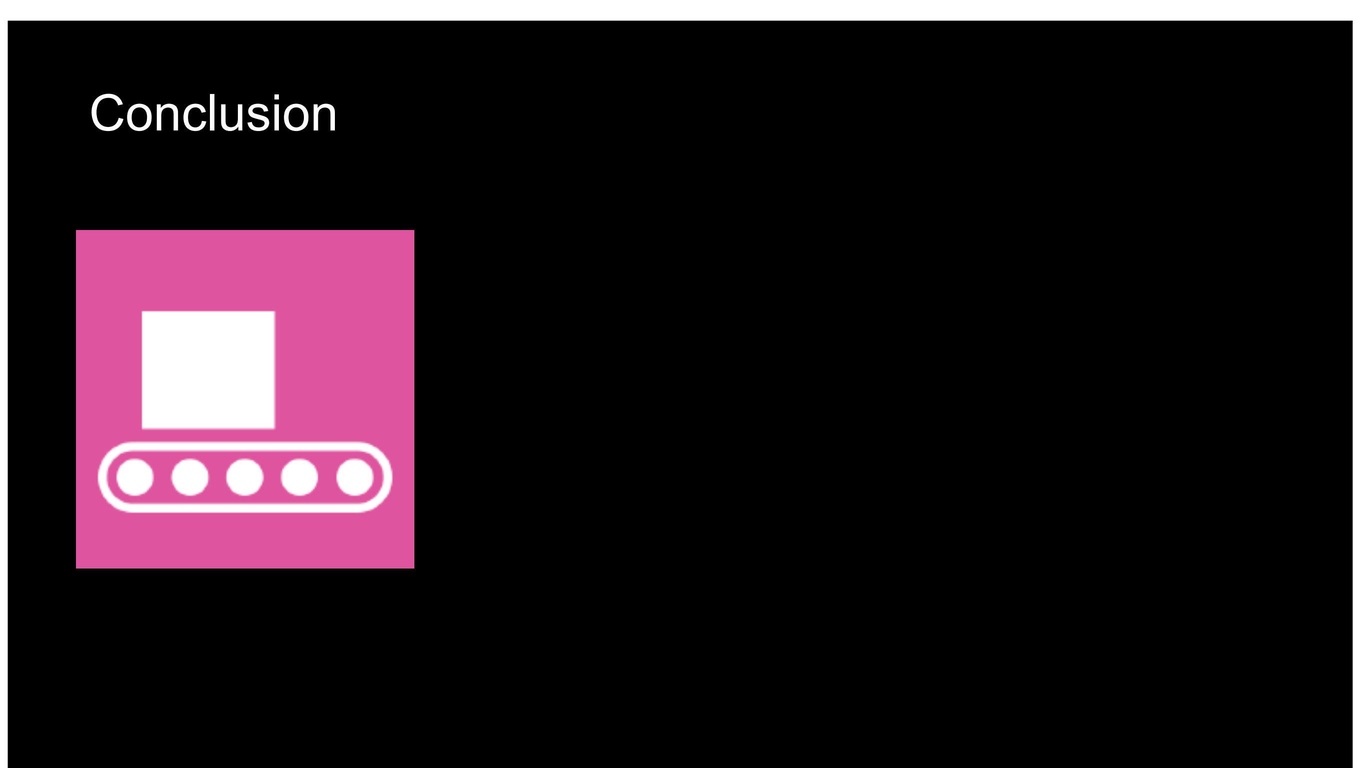
- Preventing unnecessary recipe runs via the wrapper scripts
- Separate recipe-lists and schedules for critical, frequently updated and infrequently updated software
- Rationalised preferences file for -prod and -nodes recipes
- M1 hardware

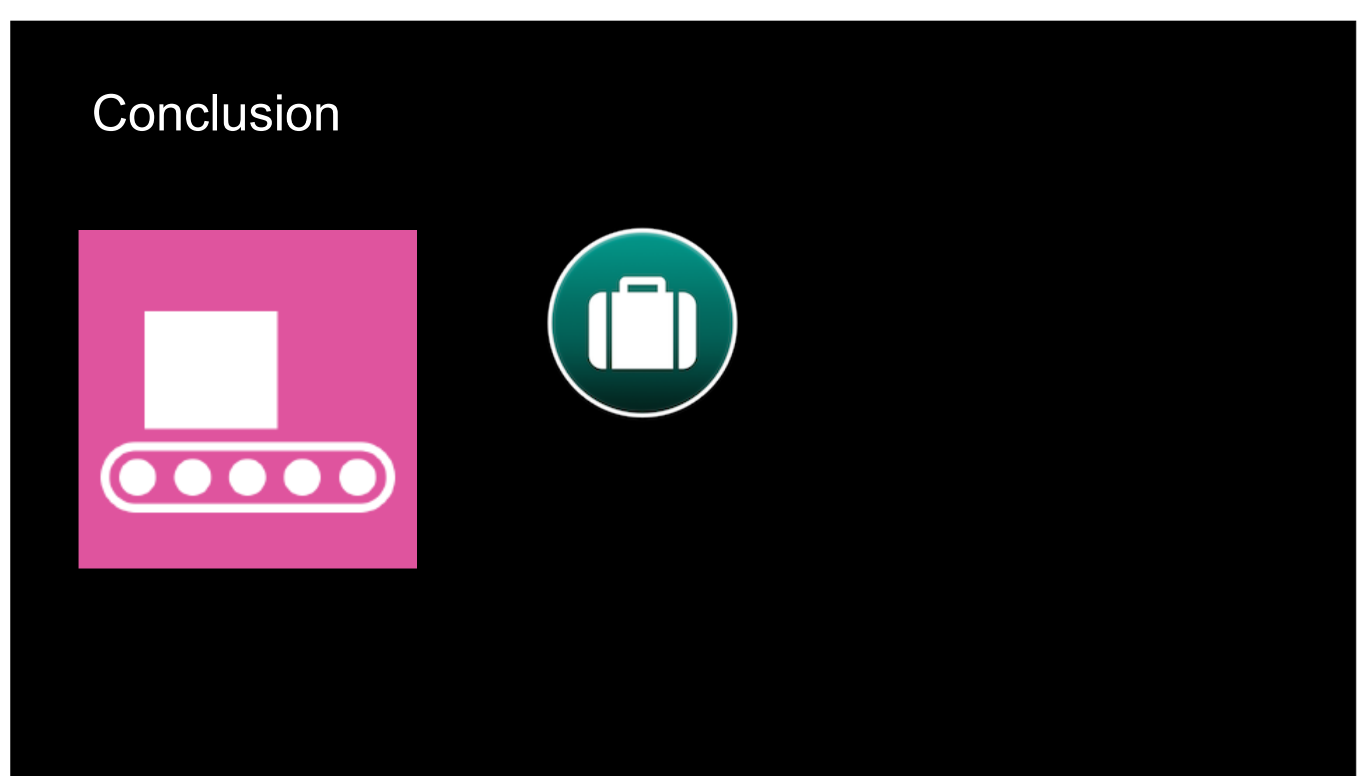


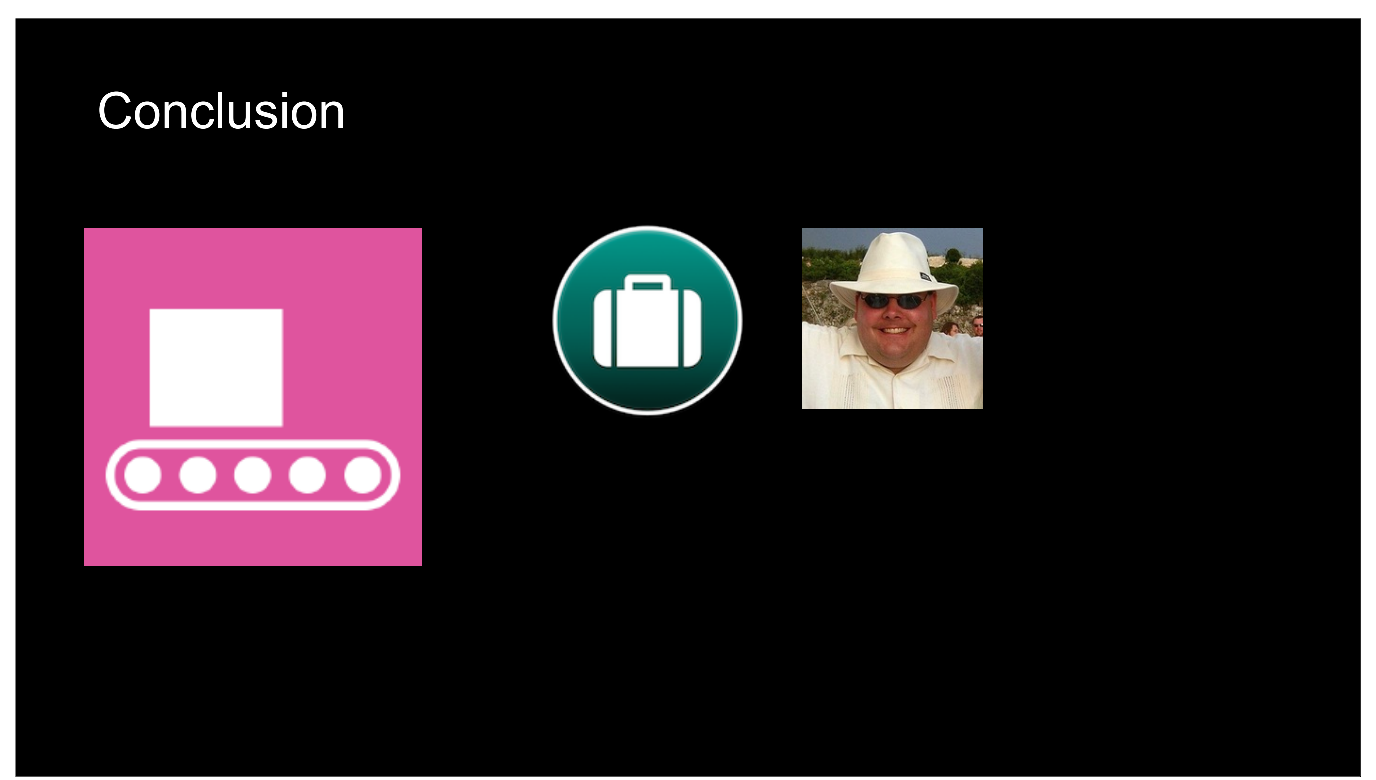


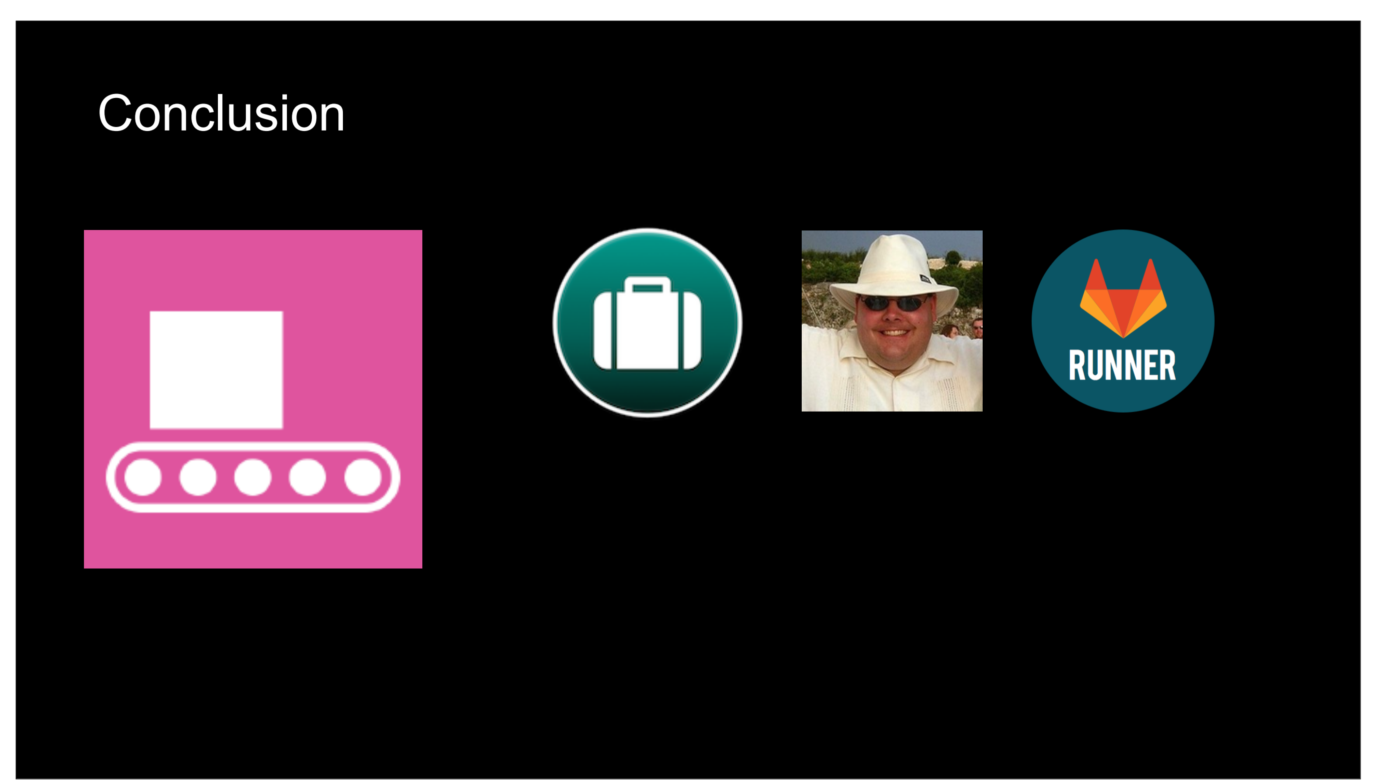
To conclude.

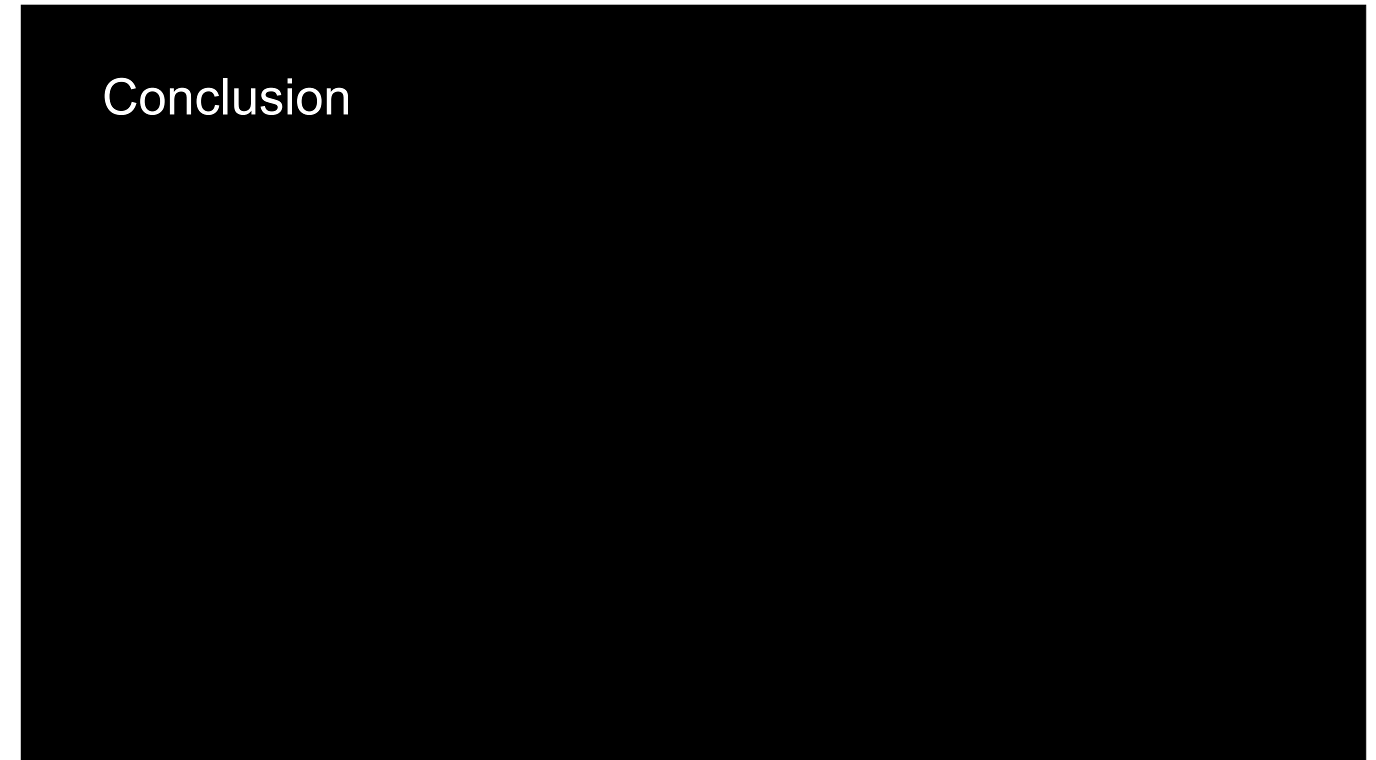
🍎 AutoPkg's core processors do a great job of packaging up software.
It's straightforward to set up and to schedule, either with AutoPkgr, a script and launchdaemon solution such as Rich Trouton's AutoPkg-conductor, or a CI/CD tool like GitLab, GitHub or Jenkins.
So, in my opinion there's no better tool to use for preparing software for deployment.


Conclusion


Conclusion


Conclusion


Conclusion
RUNNER



And if you are using AutoPkg, and especially if you have already got used to 🍎 writing recipes and perhaps 🍎 custom processors for solving individual software packaging issues, then it's worth thinking about the parts of your workflow that are not yet automated.

Perhaps writing your own AutoPkg processors would a good solution, so that you can integrate more steps into a single workflow.

Conclusion

```
1   Description: Downloads the current version of FoldingAtHom
2   Identifier: com.github.grahampugh.recipes.download.Folding
3   MinimumVersion: "2.3"
4
5   Input:
6     NAME: Folding@Home
7
8   Process:
9     - Processor: URLTextSearcher
10      Arguments:
11        re_pattern: (?P<url>https:\/\/download\.foldingathom
          *fah-installer_(?P<version>.*?)_x86_64\.mpkg\.zip)
12        url: https://download.foldingathome.org
13        user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 1
          13.1 Safari/605.1.15
14
15    - Processor: URLDownloader
16      Arguments:
17        url: "%url%"
18
19    - Processor: EndOfCheckPhase
20
```

Conclusion

```yaml
1   Description: Downloads the current version of FoldingAtHom
2   Identifier: com.github.grahampugh.recipes.download.Folding
3   MinimumVersion: "2.3"
4
5   Input:
6     NAME: Folding@Home
7
8   Process:
9     - Processor: URLTextSearcher
10      Arguments:
11        re_pattern: (?P<url>https:\/\/download\.foldingathom
          *fah-installer_(?P<version>.*?)_x86_64\.mpkg\.zip)
12        url: https://download.foldingathome.org
13        user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 1
          13.1 Safari/605.1.15
14
15    - Processor: URLDownloader
16      Arguments:
17        url: "%url%"
18
19    - Processor: EndOfCheckPhase
20
```

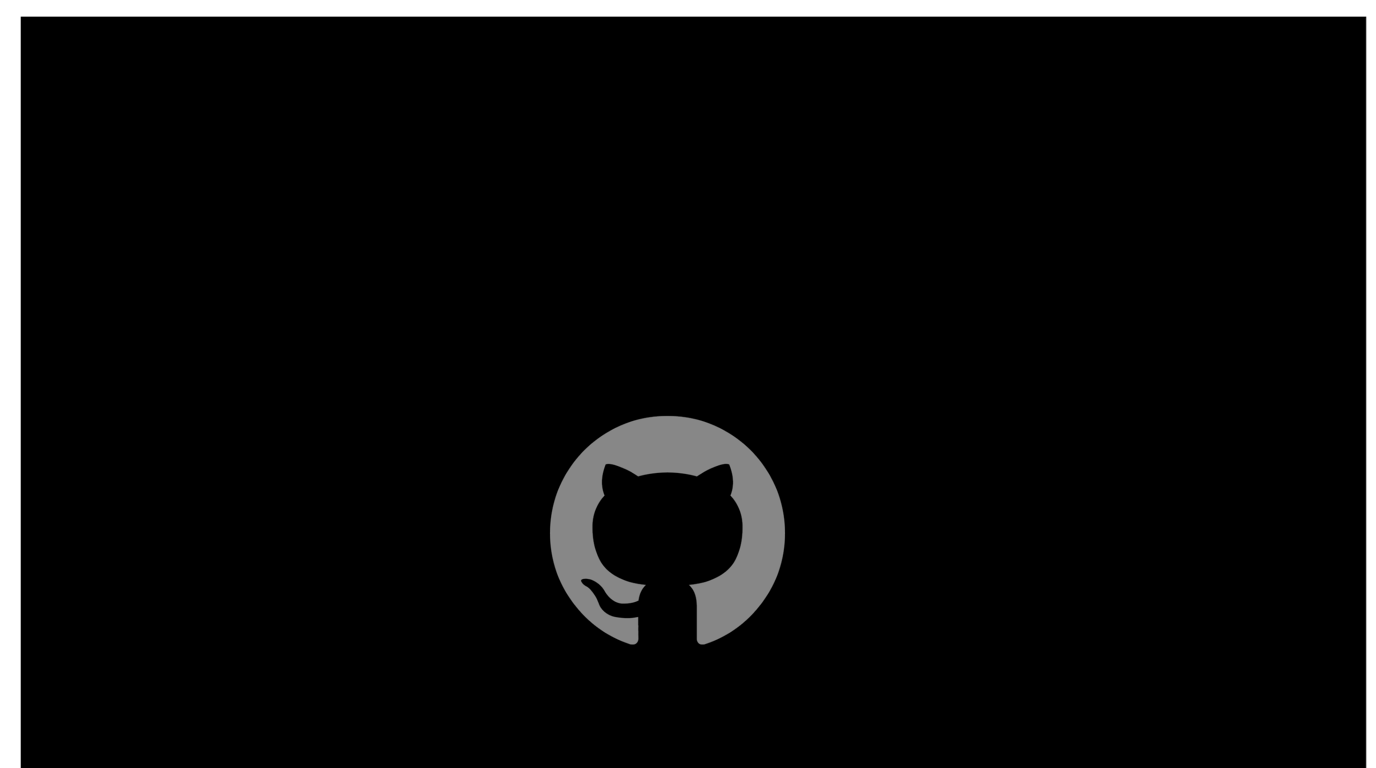
```python
30
31  class LocalRepoUpdateChecker(Processor):
32      """Provides file path to the highest version number.
33
34      input_variables = {
35          "root_path": {
36              "description": "Repo path. Used here for com
37              "required": True,
38          },
39          "found_filenames": {
40              "required": True,
41              "description": ("Output of SubDirectoryList
42          },
43          "RECIPE_CACHE_DIR": {
44              "required": True,
45              "description": ("AutoPkg Cache directory."),
46          },
47      }
48
49      output_variables = {
50          "version": {
```

At ETH, the use of AutoPkg processors has helped consolidate what was a bunch of disparate scripts or manual processes into a single, integrated Mac software deployment solution, with all these processors in daily use.

All of these are publicly available in GitHub, either for direct use, or just for getting some ideas for processors that would work better in your own organisation's workflow.

JSSRecipeReceiptChecker
LastRecipeRunResult
LastRecipeRunChecker
JamfUploadSharepointUpdater
JamfUploadSharepointStageCheck
InternalUpdateChecker
LocalRepoUpdateChecker
SMBMounter
SMBUnmounter

JamfCategoryUploader
JamfExtensionAttributeUploader
JamfPackageUploader
JamfScriptUploader
JamfComputerGroupUploader
JamfPolicyUploader
JamfPolicyDeleter
JamfComputerProfileUploader
JamfUploadSlacker
VersionRegexGenerator

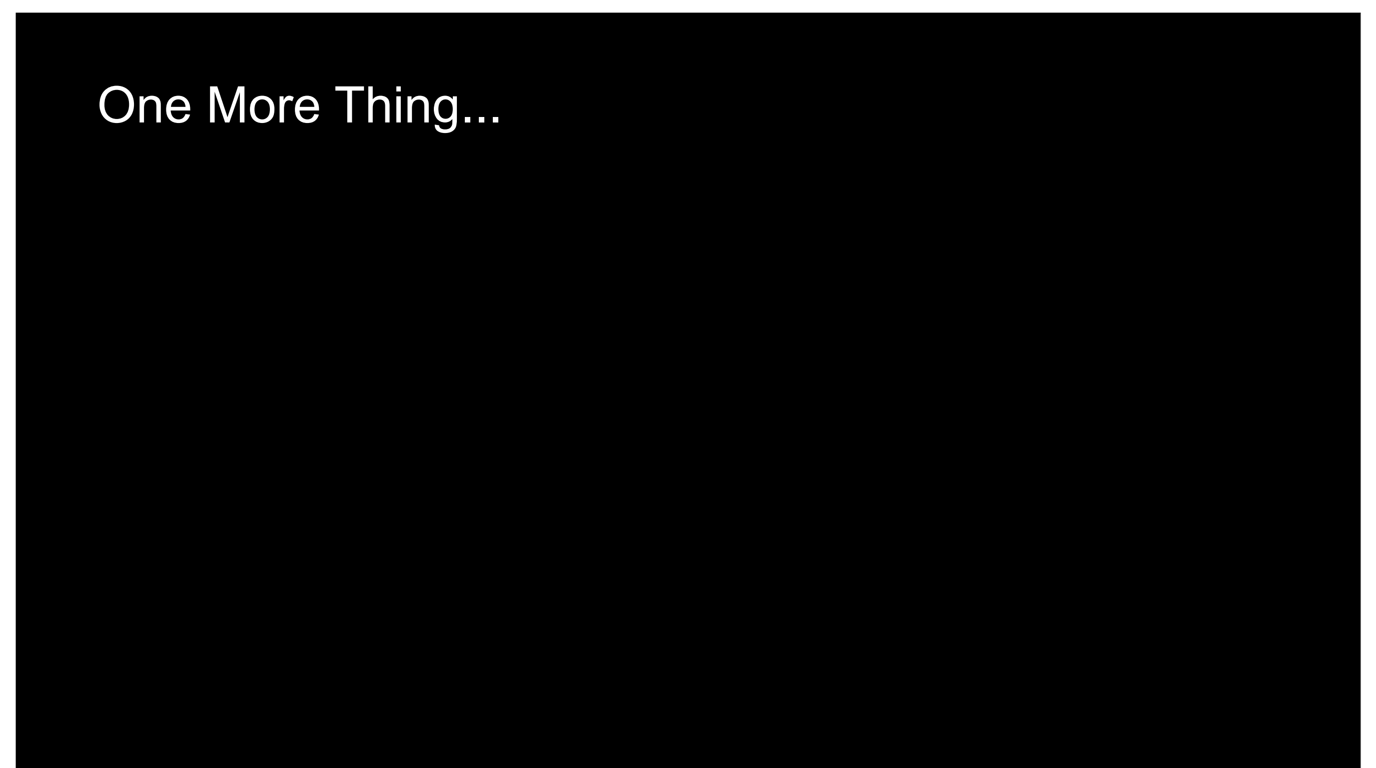


Most of them are available in my recipes repo inside the autopkg organisation, including the JamfUploader processors.
The rest are in the ETH-ITS organisation.

github.com/autopkg/grahampugh-recipes

github.com/autopkg/grahampugh-recipes

github.com/eth-its/autopkg-mac-recipes-yaml

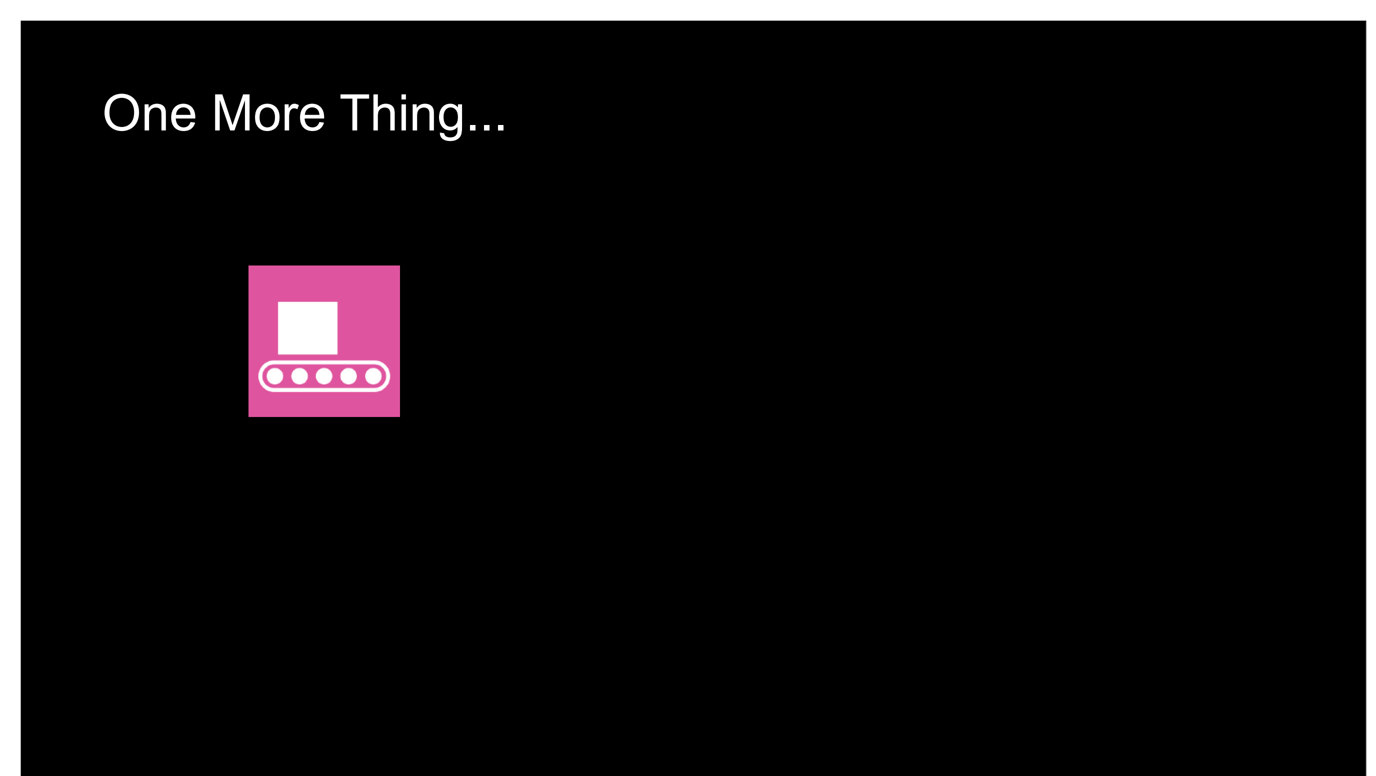


One more thing...

My colleagues in the ETH Client Delivery Windows team, especially Nick Heim, also saw the benefit of the AutoPkg framework for their software deployment needs.

So much so, that Nick decided to fork 🍎AutoPkg for 🍎Windows!

With 30 shared processors, and 🍎working recipes for downloading, packaging and deploying close to 100 software titles, ETH really does use AutoPkg for everything.


One More Thing...

One More Thing...

Windows

github.com/NickETH/autopkg

github.com/NickETH/autopkg

github.com/NickETH/recipes-win

Resources and Questions

https://grahamrpugh.com/2021/10/05/macsysadmin-presentation.html

MacAdmins Slack:
@GrahamRPugh
#macsysadminconf
#jamf-upload
#jss-importer
#eraseinstall
#switzerland

Thank you for listening to how we are using AutoPkg at ETH.

Links to everything I've mentioned today are in a blog post at the link shown here, or just go to graham-r-pugh dot com
And you'll find me in Slack as @GrahamRPugh, of course in the macsysadminconf channel, but also often hanging about in these channels.

ETH zürich

If I've given anyone some ideas about how to take AutoPkg further in their own organisation, then this has been a useful endeavour.
On the other hand, if you have any ideas for us to improve, please get in touch.
If you just think we are mad to go this far with AutoPkg, which could be true, then I'd love to hear about how others are doing this kind of thing in a Jamf-only environment, particularly in complex or multi-tenancy contexts.

Thanks again, and I hope to see many of you at an on-premises conference hopefully in 2022!