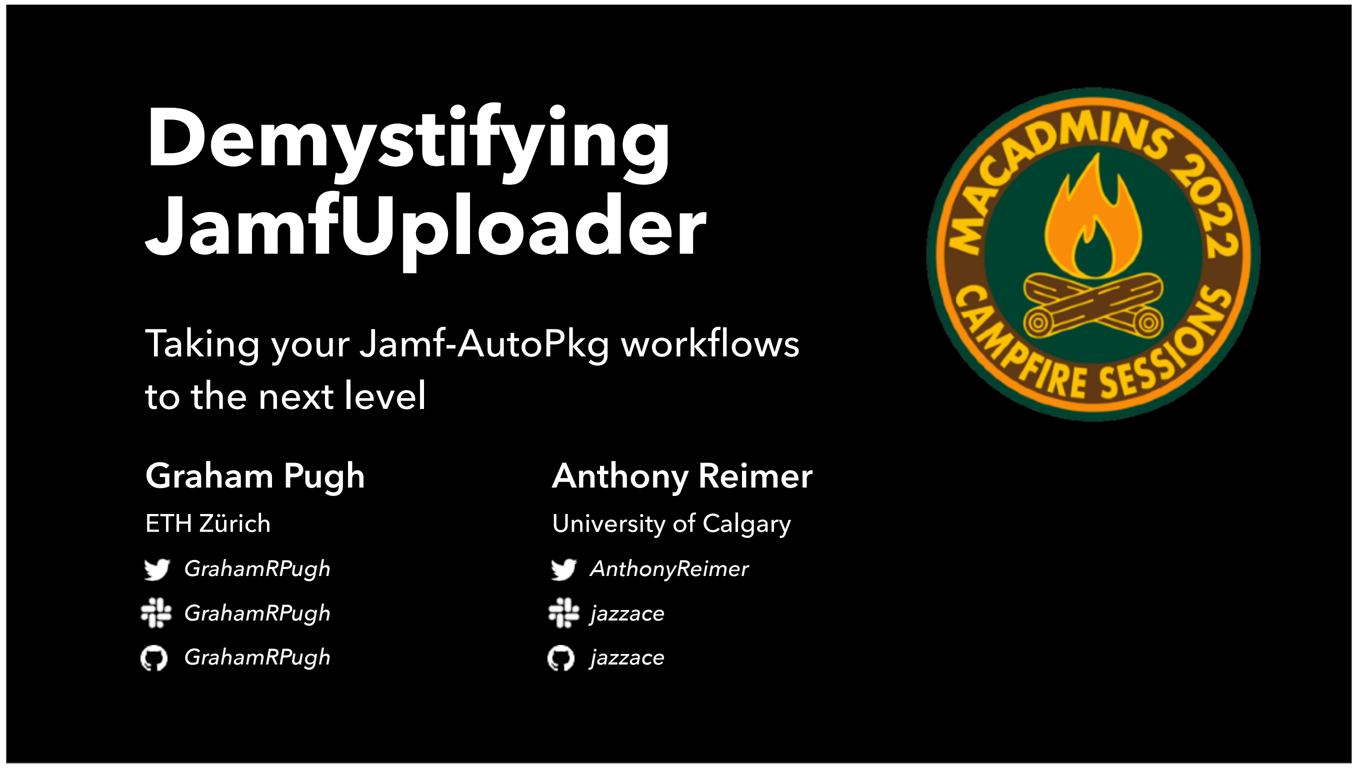


Demystifying JamfUploader

Taking your Jamf-AutoPkg workflows to the next level

Graham Pugh
ETH Zürich
GrahamRPugh
GrahamRPugh
GrahamRPugh

Anthony Reimer
University of Calgary
AnthonyReimer
jazzace
jazzace

MACADMINS 2022
CAMPFIRE SESSIONS

I'm Anthony Reimer. As I am coming to you today from Calgary, Alberta, Canada, I would like to acknowledge the traditional territories of the people of the Treaty 7 region in Southern Alberta, which includes the Blackfoot Confederacy, as well as the Tsuut'ina First Nation, and the Stoney Nakoda. The City of Calgary is also home to Métis Nation of Alberta, Region 3. I am currently the Head Technician of the Integrated Arts Media Labs at the University of Calgary. MacAdmins was the first conference that took a chance on me as a presenter in 2012, so I am both grateful and thrilled to present at this Conference once again.…
And I'm Graham Pugh, and I'm a Senior Client Engineer at ETH Zürich, the Swiss Federal Technical University. I just want to say how much of an honour it is to be presenting for the MacAdmins at Penn State conference. 8 years ago last week, as a relatively new Mac admin looking for ideas, I flew from the UK to Washington DC, drove up to State College, Pennsylvania, and attended the first day of my first ever Mac Admin conference, which was a workshop called Mac Admin Fundamentals, co-led by Anthony. I learned so much in that workshop and the subsequent presentations, including hearing about AutoPkg for the first time, and it's not an exaggeration to say that the conference was transformational in my career. So I want to give my appreciation to this conference and to Anthony today!

Introduction

This presentation is aimed at Jamf Pro administrators, like Anthony and myself - either at people who are looking to start automating their package uploads, or people who are already using AutoPkg for uploading packages to Jamf with JSSImporter.

- What is JamfUploader?
- Starting from Scratch
- Converting from JSSImporter
- Wheel of Topics

Our agenda today is:
First, a very brief overview of what JamfUploader is.
[Anthony] Second, I will talk about how I started my JamfUploader automation journey.
[Graham] Next, I'll talk from the perspective of an existing JSSImporter user looking to convert to JamfUploader.
And finally, because we have too much to cover, we are going to allow you to choose from a series of FAQs that you would like us to try and clarify.

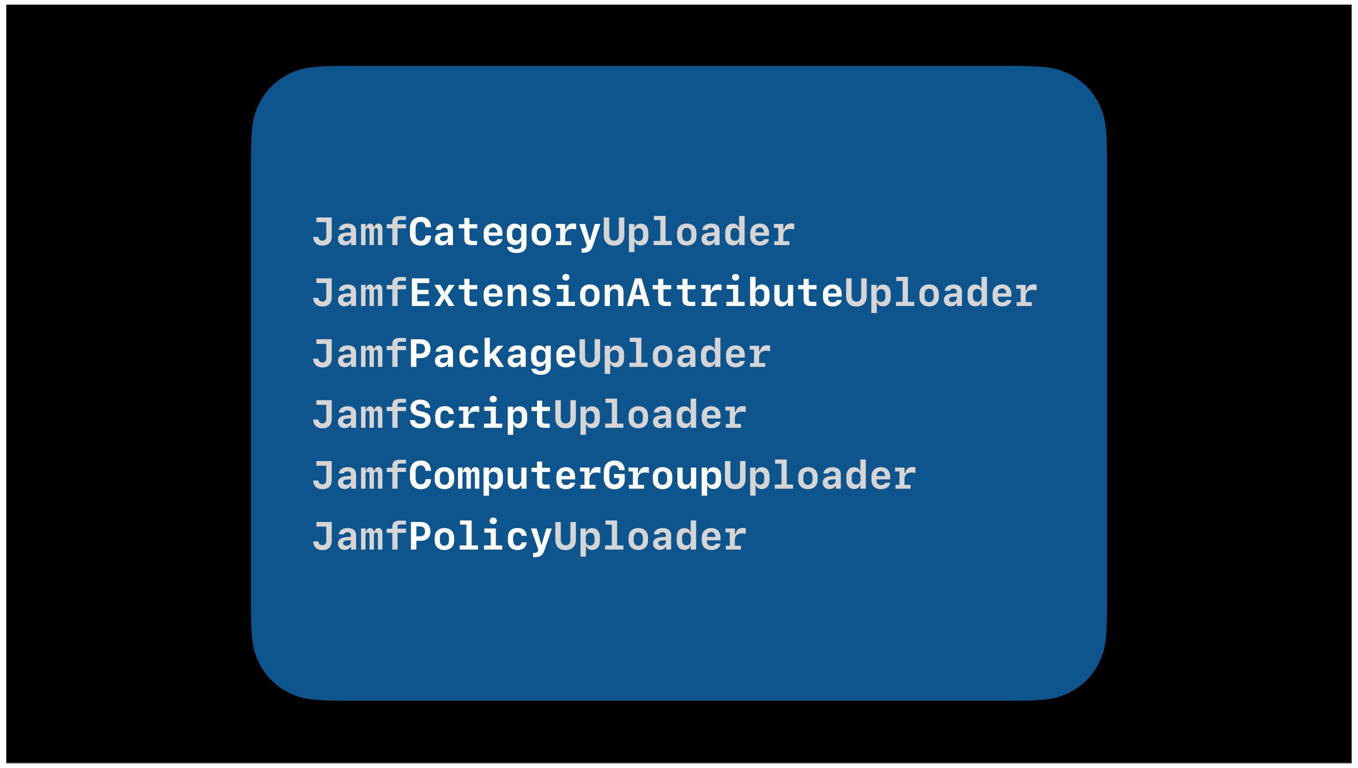
What is JamfUploader?

So, what is JamfUploader?

Making package uploading and deployment easier with JamfUploader

grahamrpugh.com/2021/10/21/jnuc-presentation-jamfuploader-session.html

Anthony and I introduced JamfUploader at a presentation back in October of last year at the JNUC conference. For those of you that haven't seen that, I'll very briefly explain what it is.

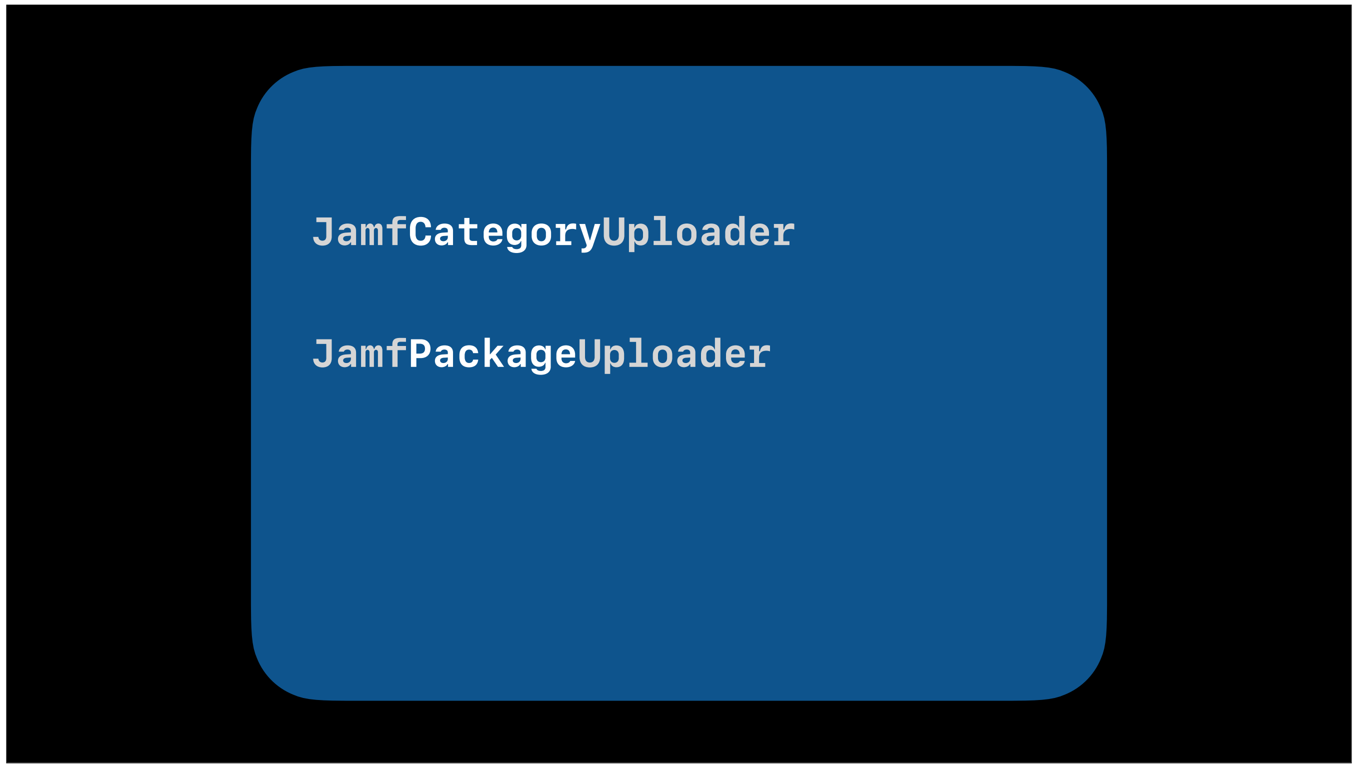


```
JamfCategoryUploader
JamfExtensionAttributeUploader
JamfPackageUploader
JamfScriptUploader
JamfComputerGroupUploader
JamfPolicyUploader
```

In short, JamfUploader consists of a set of AutoPkg processors for uploading things to Jamf Pro, using the Jamf Pro APIs.
The idea behind these processors is similar to the core AutoPkg processors, in that each processor is designed to do a single task.
You only add the processors you need to perform the complete set of tasks required.
If you've used JSSImporter, you'll know that it is a single processor that does a bunch of different API requests in one.
The six JamfUploader processors shown here can be added to an AutoPkg recipe in combination to produce exactly the same result as a JSS recipe - that means, an uploaded package, a new policy and a smart group or groups, plus any required scripts and extension attributes.

JamfPackageUploader

The advantage of JamfUploader over JSSImporter is that its individual processors provide greater flexibility and visibility of what the recipe actually does. For example, if you only want to upload a package, then you only need to add the JamfPackageUploader processor to your recipe. You can assign a category to the package if that category already exists in Jamf.

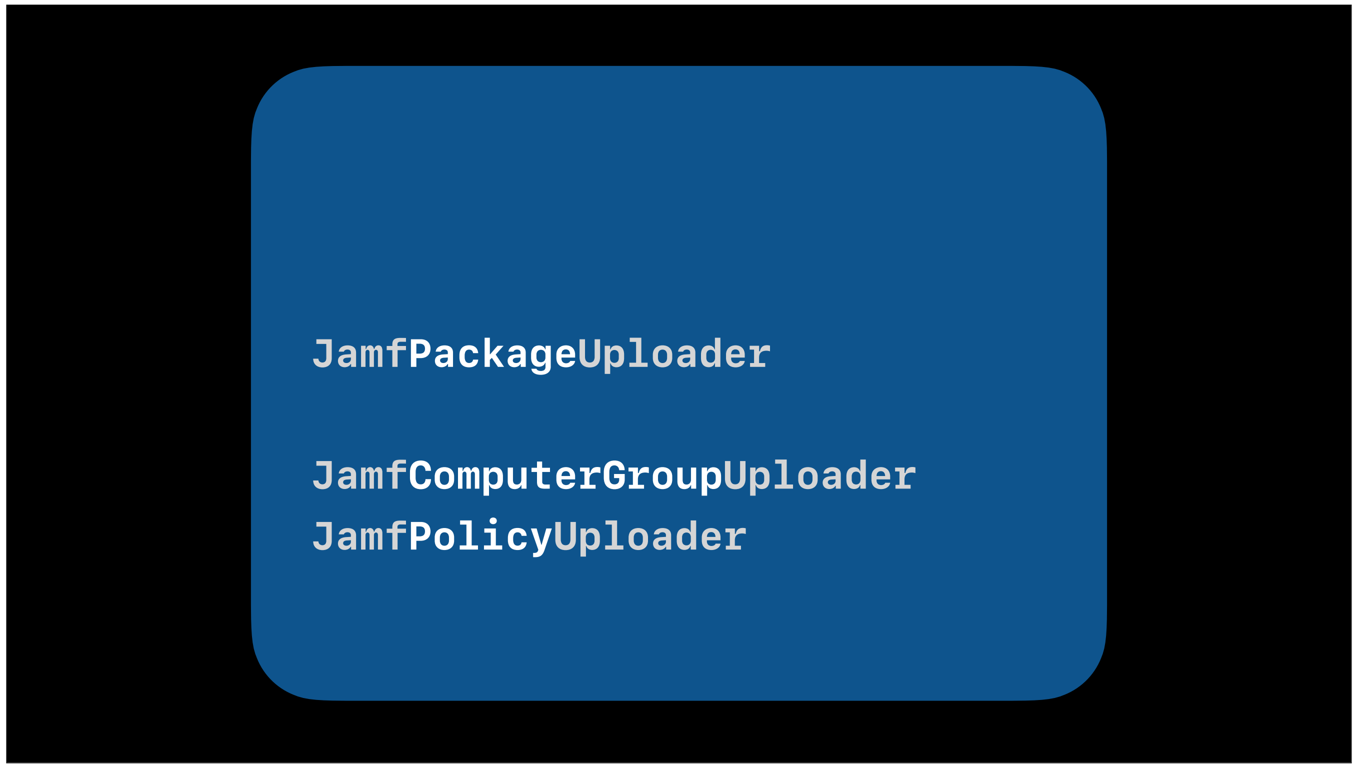


```
JamfCategoryUploader

JamfPackageUploader
```

If you want the recipe to make sure the category exists, creating it if it's not there, then you would also include the JamfCategoryUploader processor in your AutoPkg recipe.

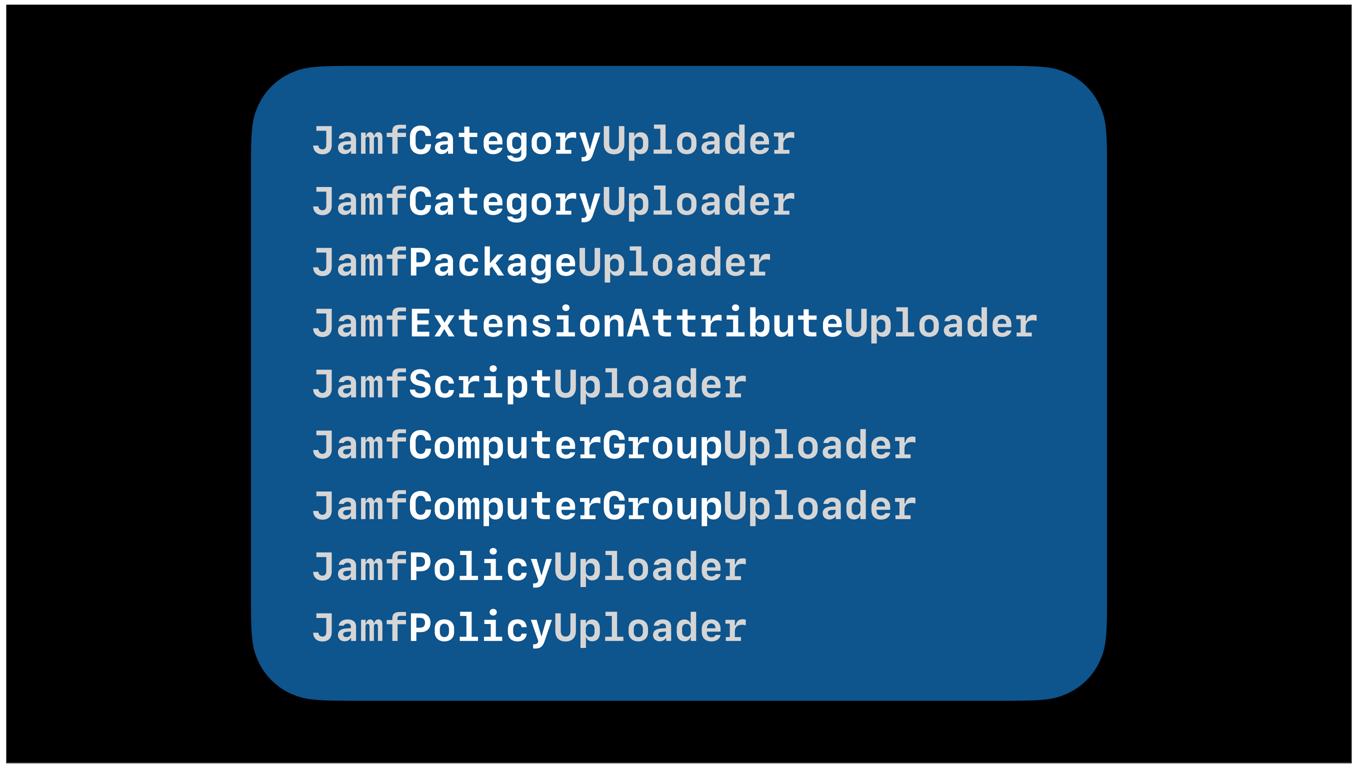
```
JamfPackageUploader

JamfPolicyUploader
```

If you also want to create a policy when a package is uploaded, you add the JamfPolicyUploader processor.



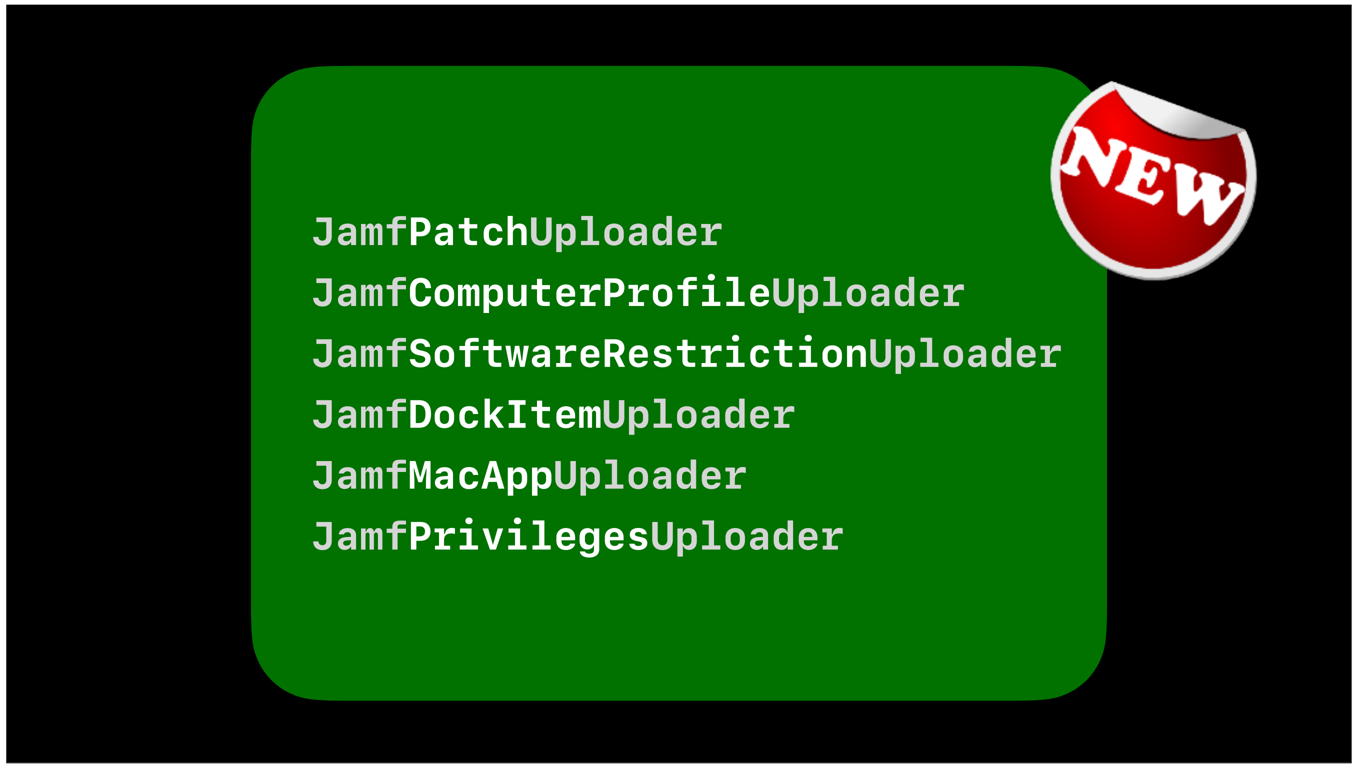
```
JamfPackageUploader

JamfComputerGroupUploader
JamfPolicyUploader
```

If you want to scope the policy to a computer group, then you add the JamfComputerGroupUploader processor.



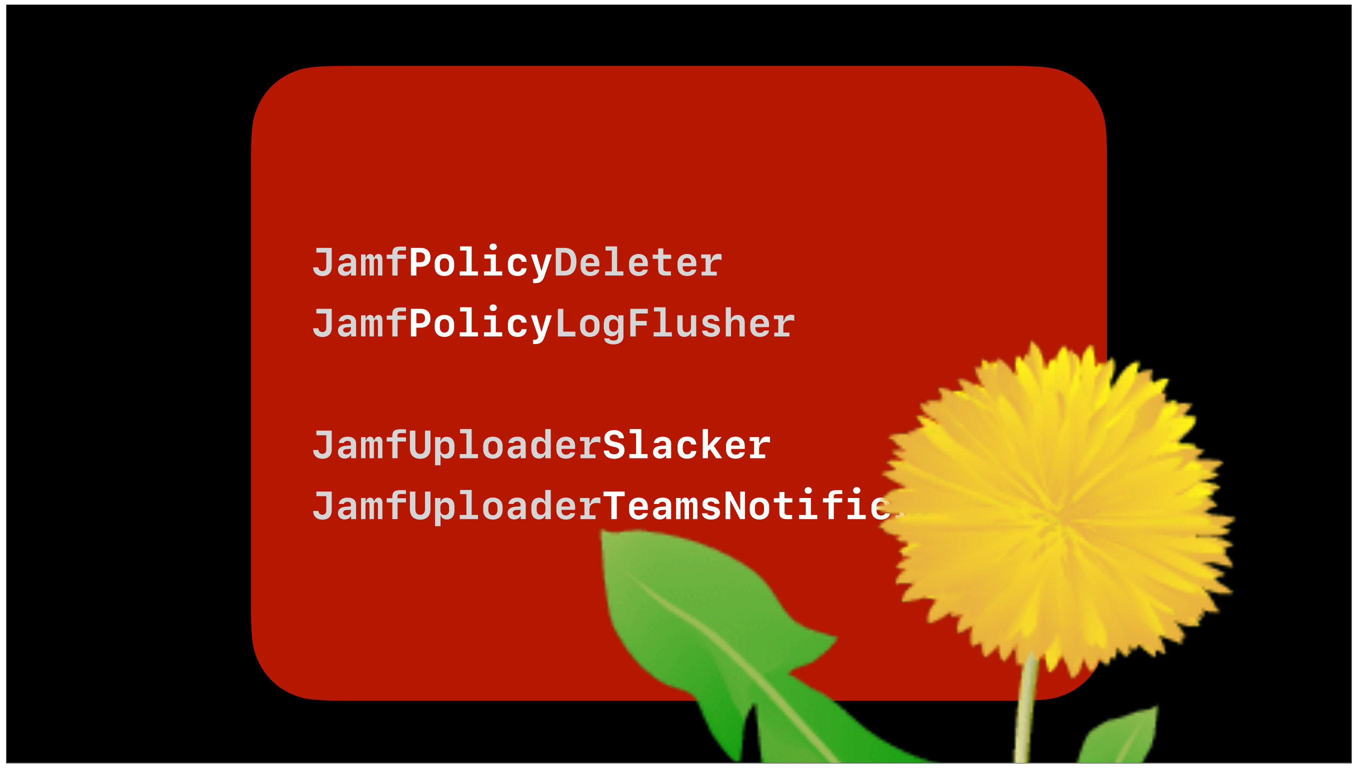
```
JamfCategoryUploader
JamfCategoryUploader
JamfPackageUploader
JamfExtensionAttributeUploader
JamfScriptUploader
JamfComputerGroupUploader
JamfComputerGroupUploader
JamfPolicyUploader
JamfPolicyUploader
```

Your recipe can have as many or few processors as it needs to create or update whatever objects you need in your Jamf Pro server. Each of the processors can appear more than once, say if more than one category or policy needs to be created or updated.



```
JamfPatchUploader
JamfComputerProfileUploader
JamfSoftwareRestrictionUploader
JamfDockItemUploader
JamfMacAppUploader
JamfPrivilegesUploader
```

Since our JNUC presentation last year, more processors have been added to the suite, allowing more types of Jamf objects to be created. JamfUploader is no longer confined to that package-group-policy workflow. I use recipes to create script-based policies, to configure Mac App Store apps with consistent settings, to upload config profiles, and even to maintain the correct CRUD privileges for all our users.

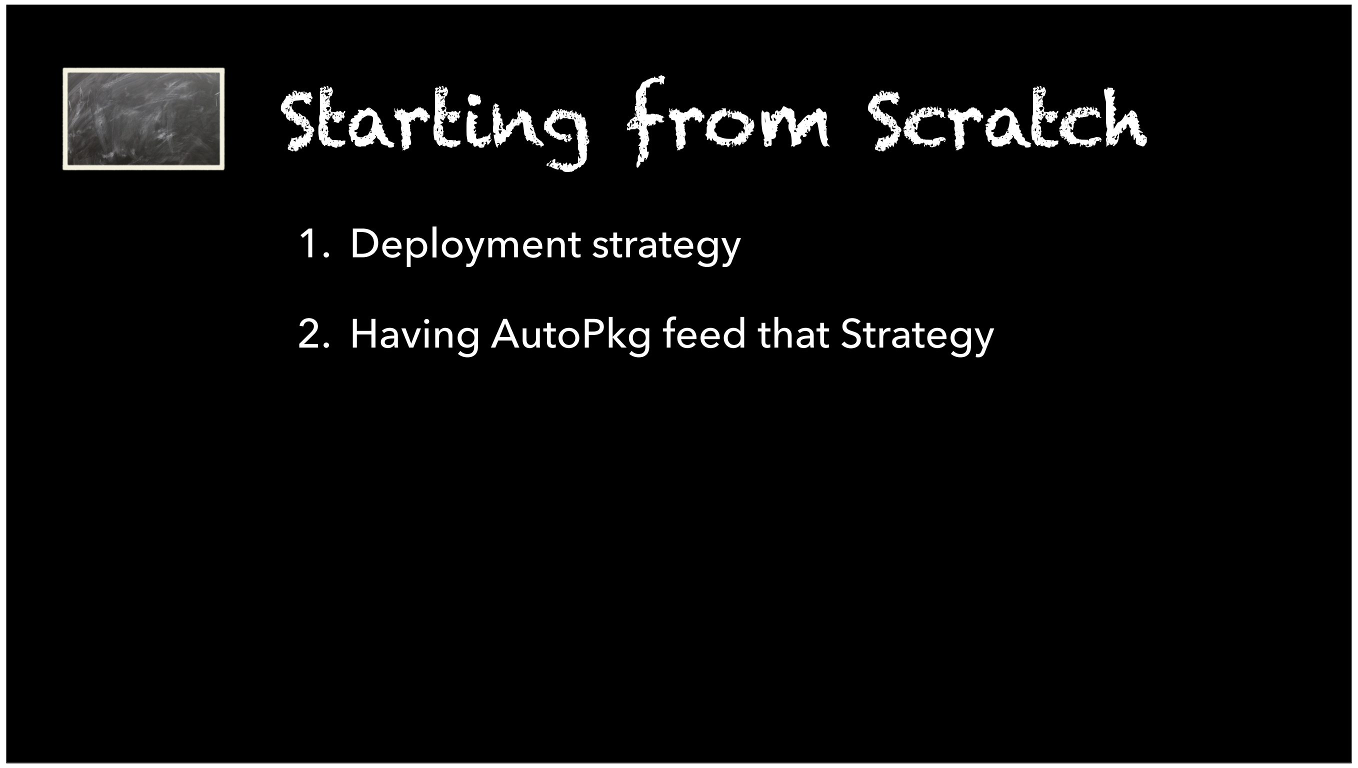


```
JamfPolicyDeleter
JamfPolicyLogFlusher

JamfUploaderSlacker
JamfUploaderTeamsNotifie
```

There are also some other processors that are part of the suite but are not for uploading anything. We have a processor for deleting a policy, which I use to delete a testing policy when the package is moved to production. There is a processor for flushing a policy, which is good if you use "Once per computer" policies. And we have processors for sending notifications of the results of the other processors to Slack and Teams.

But before I get too far into the weeds, let's go back to the beginning, and look at one practical way to begin using the processors. Anthony, over to you.

[Anthony:] Thanks, Graham. I am a long-time AutoPkg user but most of my experience was focussed on the benefits of automatically finding and fetching software installers, plus making custom installer packages in a programmatic way. We only adopted Jamf Pro at our institution in a serious way when Apple's guidance on MDM solutions changed from "It's nice to have" to "Select your MDM Vendor." My AutoPkg workflows had never fed a management system before, unless you considered DeployStudio and ARD to be management systems. So I was starting from scratch when it came to integrating AutoPkg and Jamf Pro.

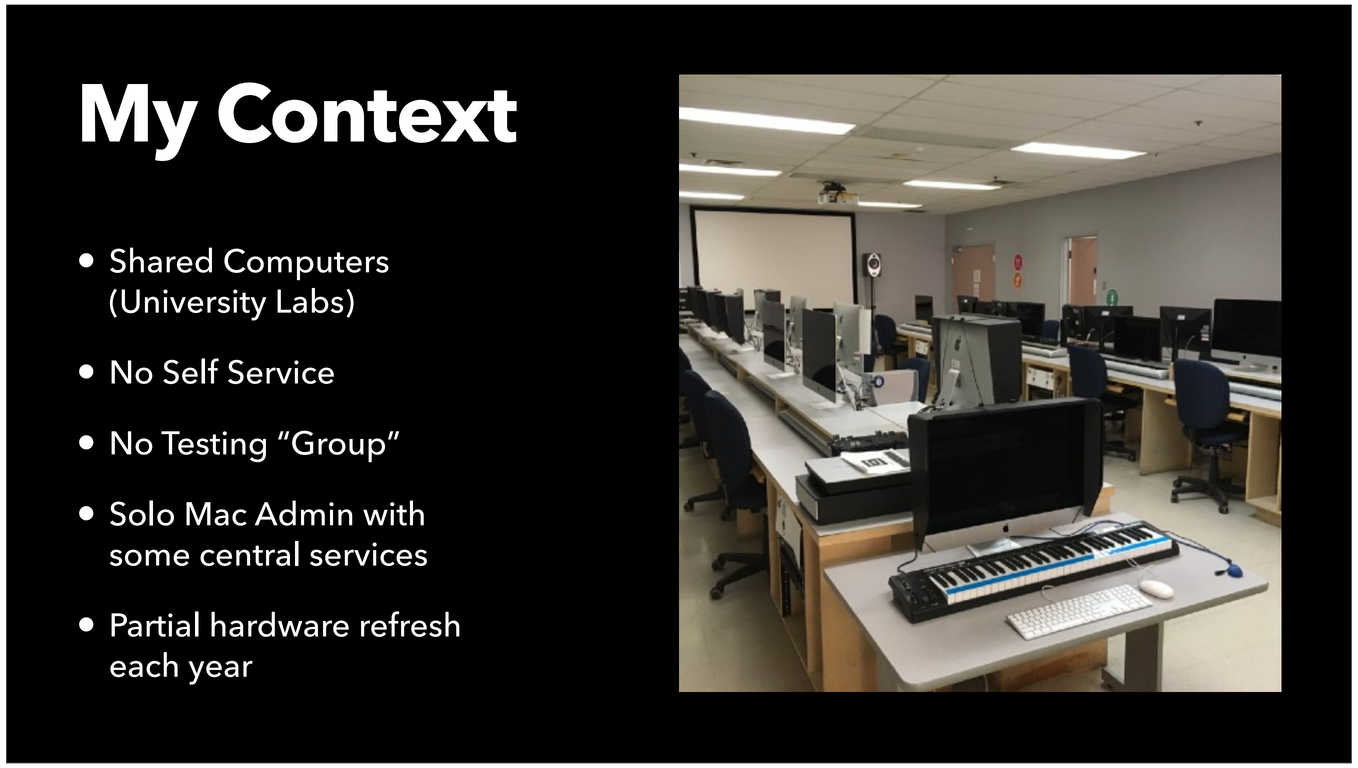


Starting from Scratch

1. Deployment strategy
2. Having AutoPkg feed that Strategy

I'm going to break this part of the talk into two sections:

1. How do I want to deploy and update my software using Jamf Pro? This is a good exercise to do whether you are using AutoPkg or not, and whether you are using Jamf Pro or any other management system.
2. How do I get AutoPkg to do as much of that work as possible?

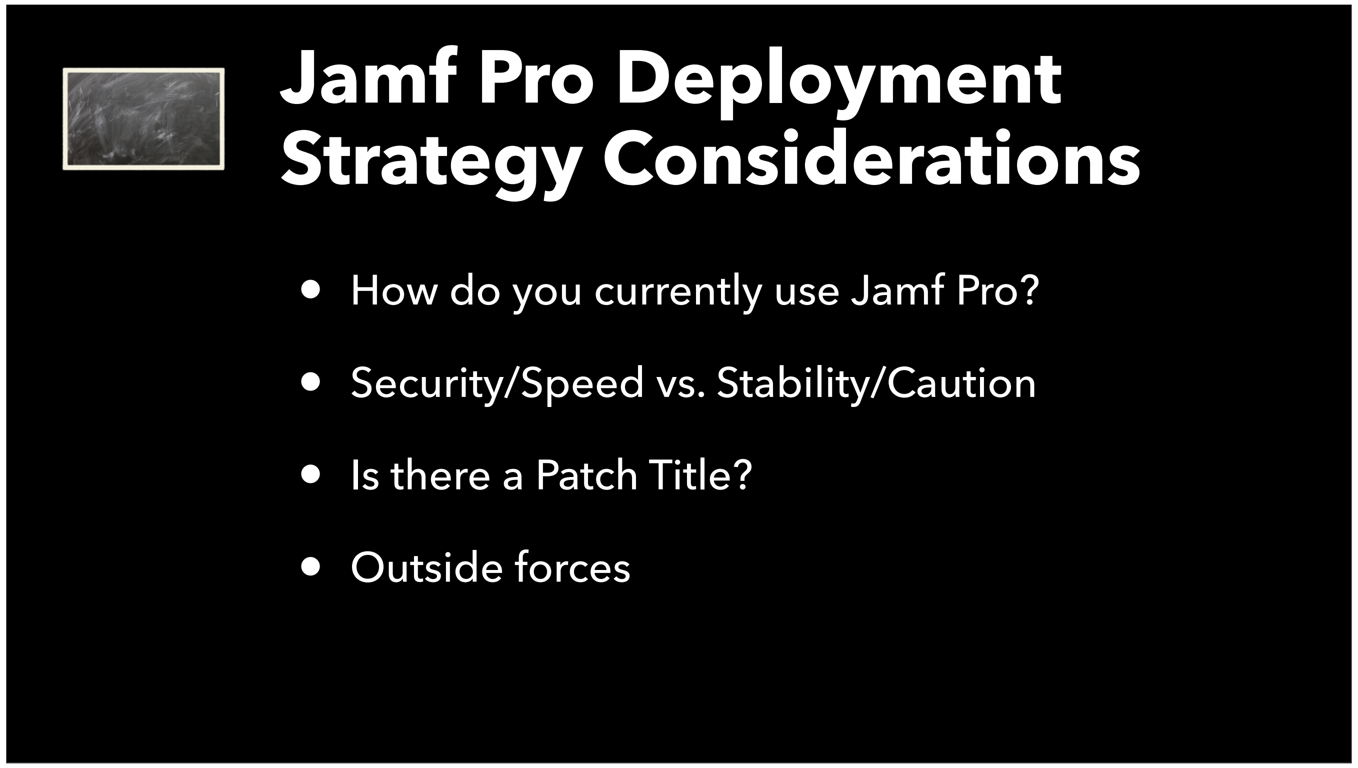


My Context

- Shared Computers (University Labs)
- No Self Service
- No Testing "Group"
- Solo Mac Admin with some central services
- Partial hardware refresh each year

All of this is influenced by my context. I manage a few dozen Macs in my computer labs. Everything has to be installed by me or the management system; Self Service is not a thing. There are also no testing groups in computer labs—essentially, *I* am the testing group. As well, my labs are a site within Jamf Pro. While some of the installer packages I upload via AutoPkg may be used by others, I write all my own policies with the exception of things like Antivirus, which Central IT controls. And finally, my hardware tends to get replaced in small groups, not full labs. This means that I have to be able to enrol and initially deploy a new Mac at any time. Supporting that workflow also gives me the flexibility to nuke and pave as needed.

I mention all of this because my context is not your context, so you should take all of this into consideration when considering how you wish to manage the computers under your direction.



Jamf Pro Deployment Strategy Considerations

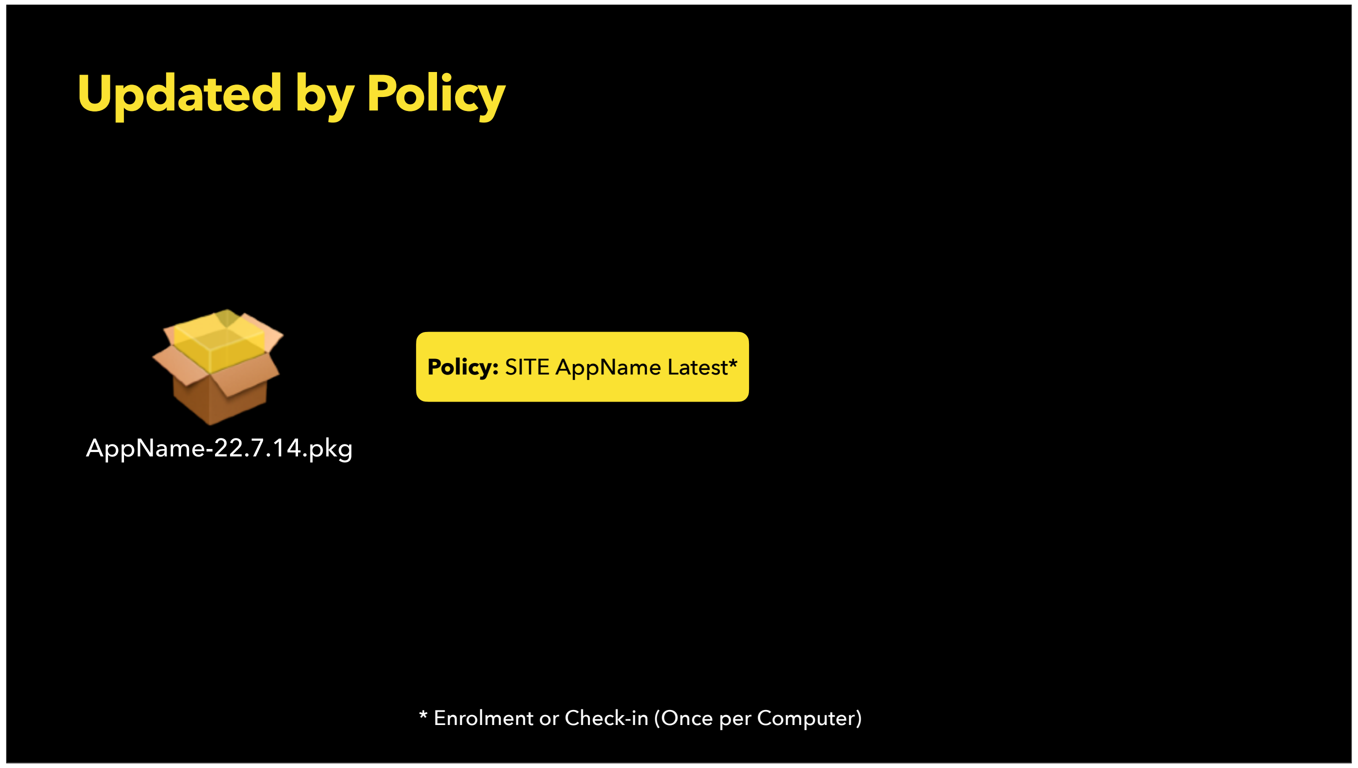
- How do you currently use Jamf Pro?
- Security/Speed vs. Stability/Caution
- Is there a Patch Title?
- Outside forces

I started out using Jamf Pro from the web interface to create policies and upload configuration profiles, and I used Jamf Admin to upload installer packages and scripts. So the strategic goal for me was to automate what I was already doing with policies and packages. There are certainly other strategies, but this is where I started.

The next consideration in developing an overall deployment strategy was related to how and how often I want to push out updates. Which titles do I want to patch very rapidly? An example might be web browsers, where every second patch seems to fix a security vulnerability and users would have an alternative browser to use if an update did break something unexpectedly. I also have an aggressive policy on macOS updates, and since it is a Lab, I can do it overnight or whenever I see a computer not in use. In contrast, which titles do I want to hold off patching unless things are broken or I can find a break in the schedule? Which items are somewhere in the middle? For example, we don't upgrade major versions of software during a 4-month academic term, but we might push updates if they are available. Also, the bigger your payload, the more work you have if your process is controlled manually, so if you are like me and have 50 titles to push and manage, that may lead to a more conservative update posture unless you can get some help through automation.

Patch Management in Jamf Pro got a whole lot better when Jamf purchased Mondada. So when I was devising my deployment strategy, I wanted to use Patch where I could because it supports blocker apps *à la* Munki and the ability to specify a particular version in case you need to downgrade. Of course, Patch Management only updates an app that is already on the computer, so you also have to create a policy to do the initial install when using it.

Finally, you may have change management processes that you have to follow or other external factors that influence what you can or cannot do. In my case, since virtually all my work is scoped to my site of a few dozen computers, my main restriction is getting software approved by Legal and IT before I can deploy it. Otherwise, the mess I make is my own and it doesn't affect others.



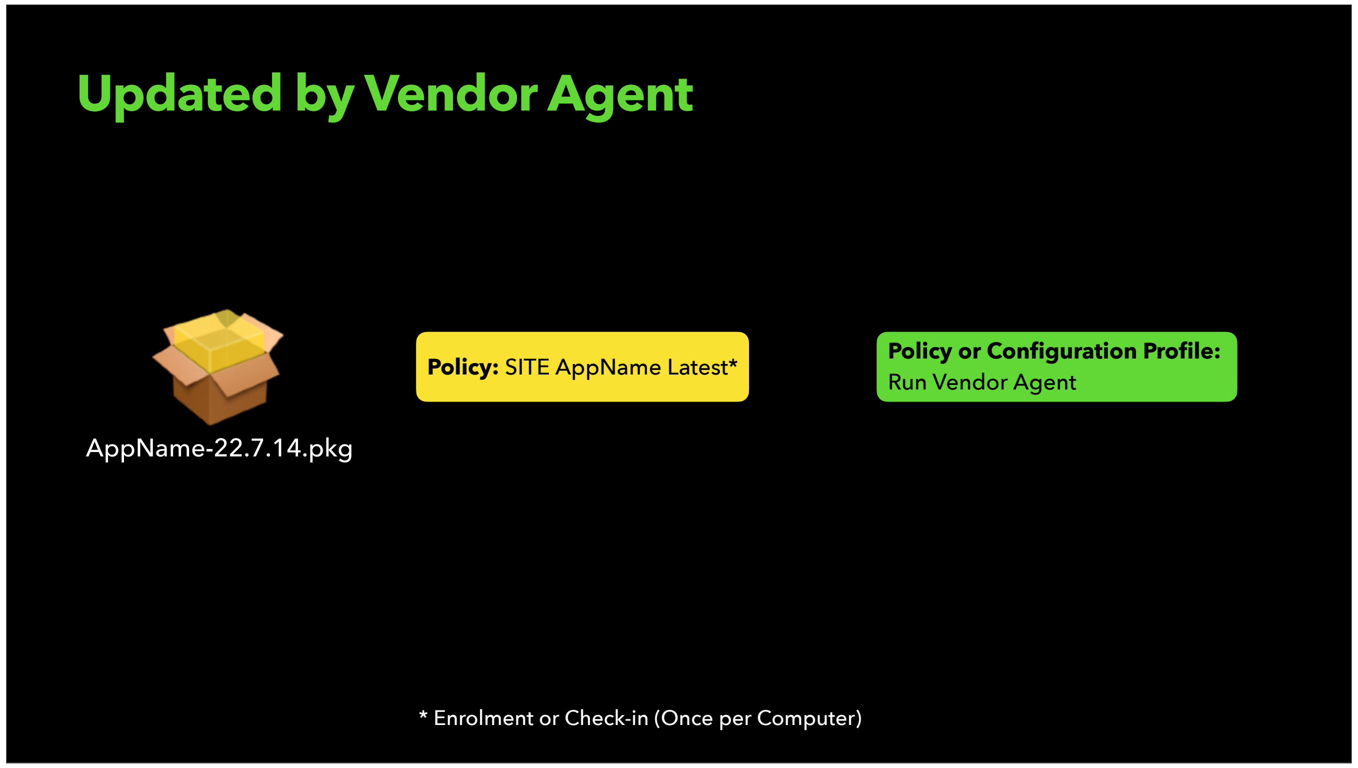
Updated by Policy

AppName-22.7.14.pkg

Policy: SITE AppName Latest*

* Enrolment or Check-in (Once per Computer)

Here's a summary of what my Jamf Pro workflow looks like:
For most of my titles, I have a Jamf Pro Policy that runs Once per Computer on Enrolment or Check-in that installs the software. When a new version is released, I replace the pkg in the policy and manually flush the policy when I want to push the update. >



Updated by Vendor Agent

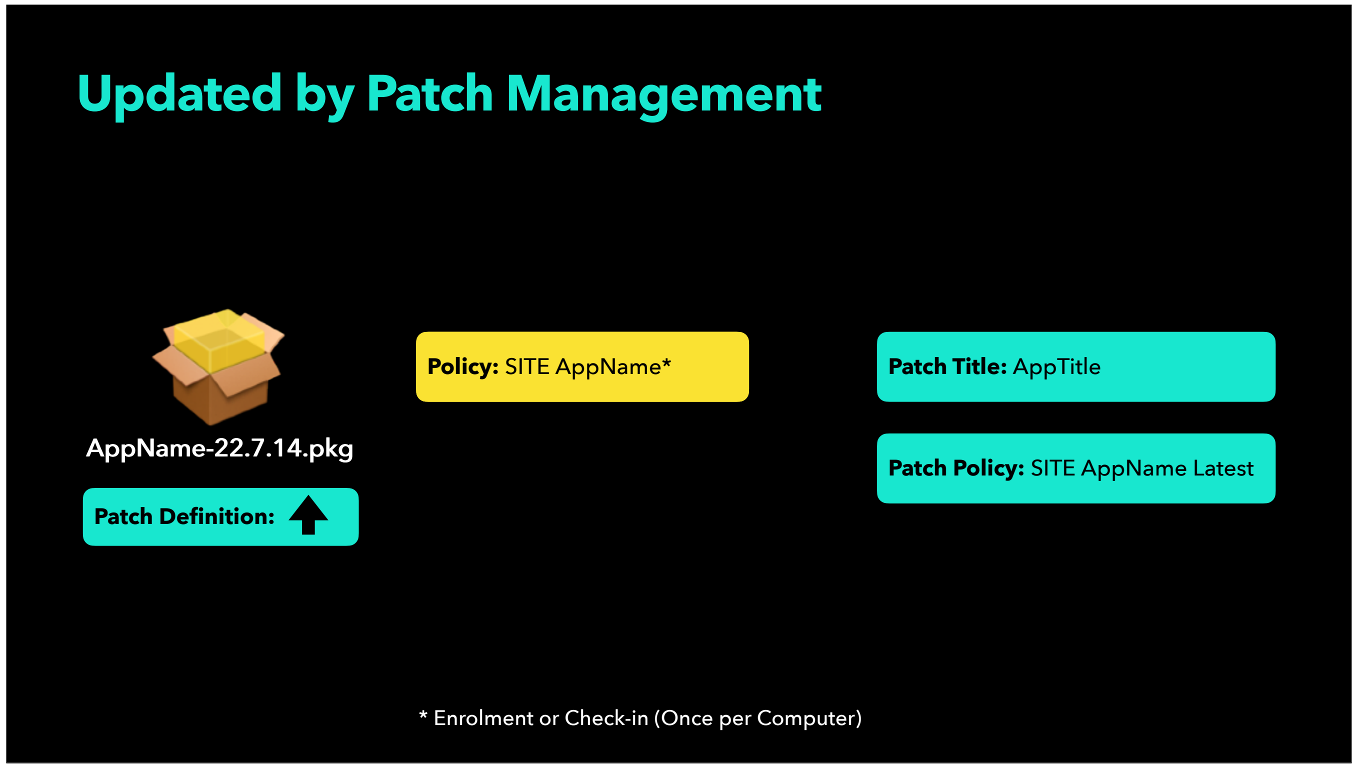
AppName-22.7.14.pkg

Policy: SITE AppName Latest*

Policy or Configuration Profile: Run Vendor Agent

* Enrolment or Check-in (Once per Computer)

For my Adobe Creative Cloud and Microsoft Office apps, I have a normal policy install the major version of the app and then I trigger the vendor's update mechanism. In the case of Adobe Remote Update Manager (RUM), I run a one-liner once per week that checks for updates of the currently deployed major version. In the case of Microsoft AutoUpdate, I have a configuration policy turn that update mechanism on. This vendor update trigger I only need to setup once, so I still do this manually. >



Updated by Patch Management

AppName-22.7.14.pkg

Patch Definition: ⬆

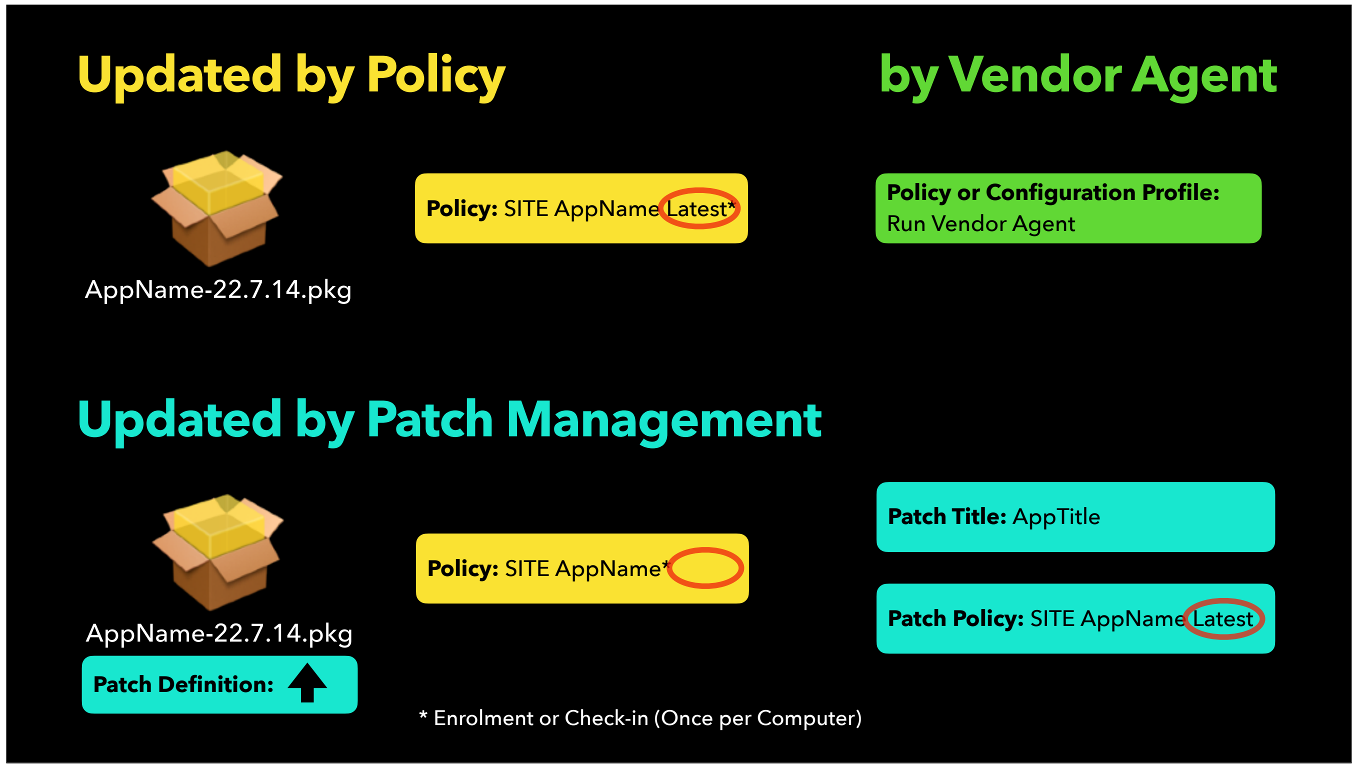
Policy: SITE AppName*

Patch Title: AppTitle

Patch Policy: SITE AppName Latest

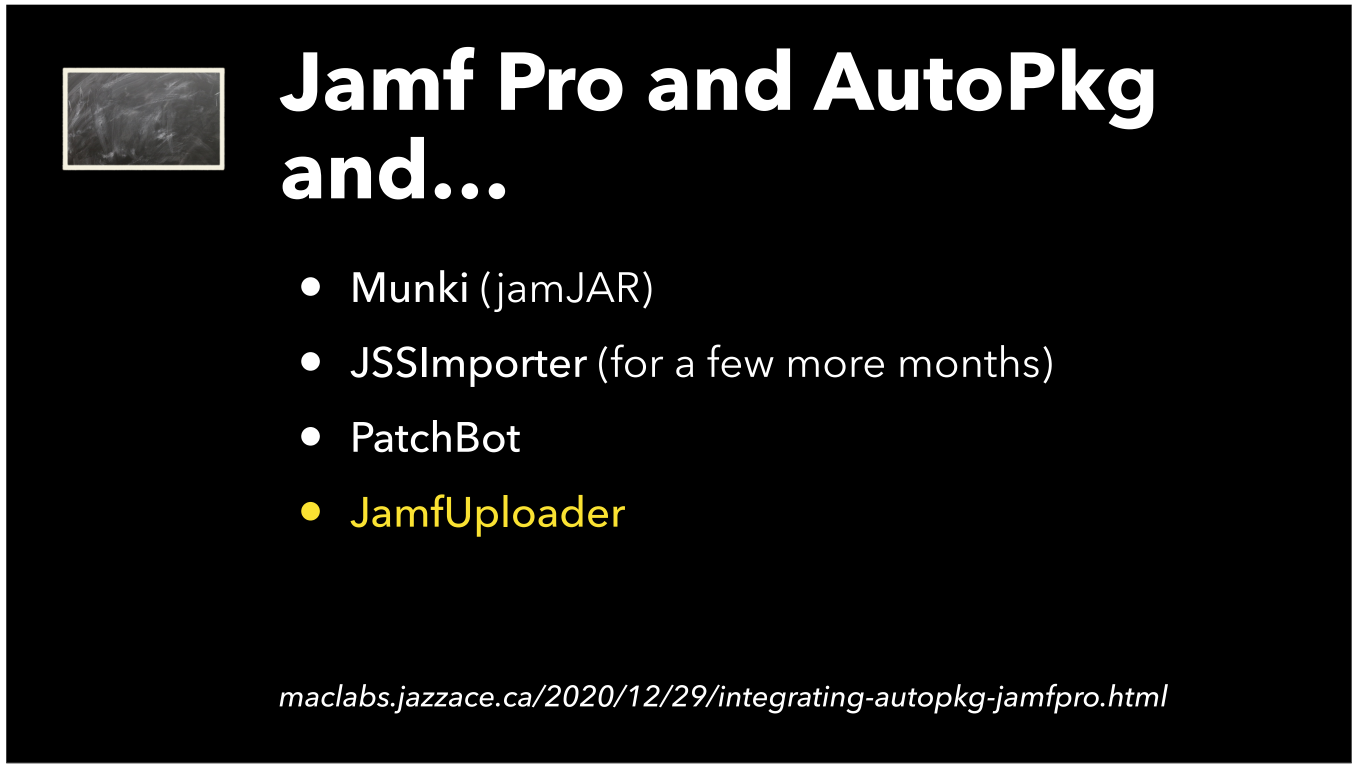
* Enrolment or Check-in (Once per Computer)

For titles I have setup using Patch Management, I setup the same regular Policy for initial deployment but I then setup a Patch Management Software Title, linking the latest pkg to the correct version in the Definition tab, and then add a Patch Policy that knows about that definition. When a new version is released, I once again link the latest pkg to the correct version in the Definition tab and update the Patch Policy to push out the new pkg to those computers in scope that have the software title.


Updated by Policy
by Vendor Agent
AppName-22.7.14.pkg
Policy: SITE AppName Latest*
Policy or Configuration Profile: Run Vendor Agent
Updated by Patch Management
AppName-22.7.14.pkg
Patch Definition:
Policy: SITE AppName*
Patch Title: AppTitle
Patch Policy: SITE AppName Latest
* Enrolment or Check-in (Once per Computer)

So that's the Jamf Pro deployment workflow I am looking for AutoPkg to help me automate.

Just a small aside: I have adopted a convention where I add the word "Latest" to the end of the policy name that controls updating of the title. This way, I can tell at a glance whether a particular policy is linked to a Patch Title or not. This is also consistent with other methodologies that append the version number to the policy name to indicate it is just for this version. I use "Latest" to indicate that the policy — whether regular or patch — will be updated. If "Latest" doesn't appear, it only applies to an initial install.

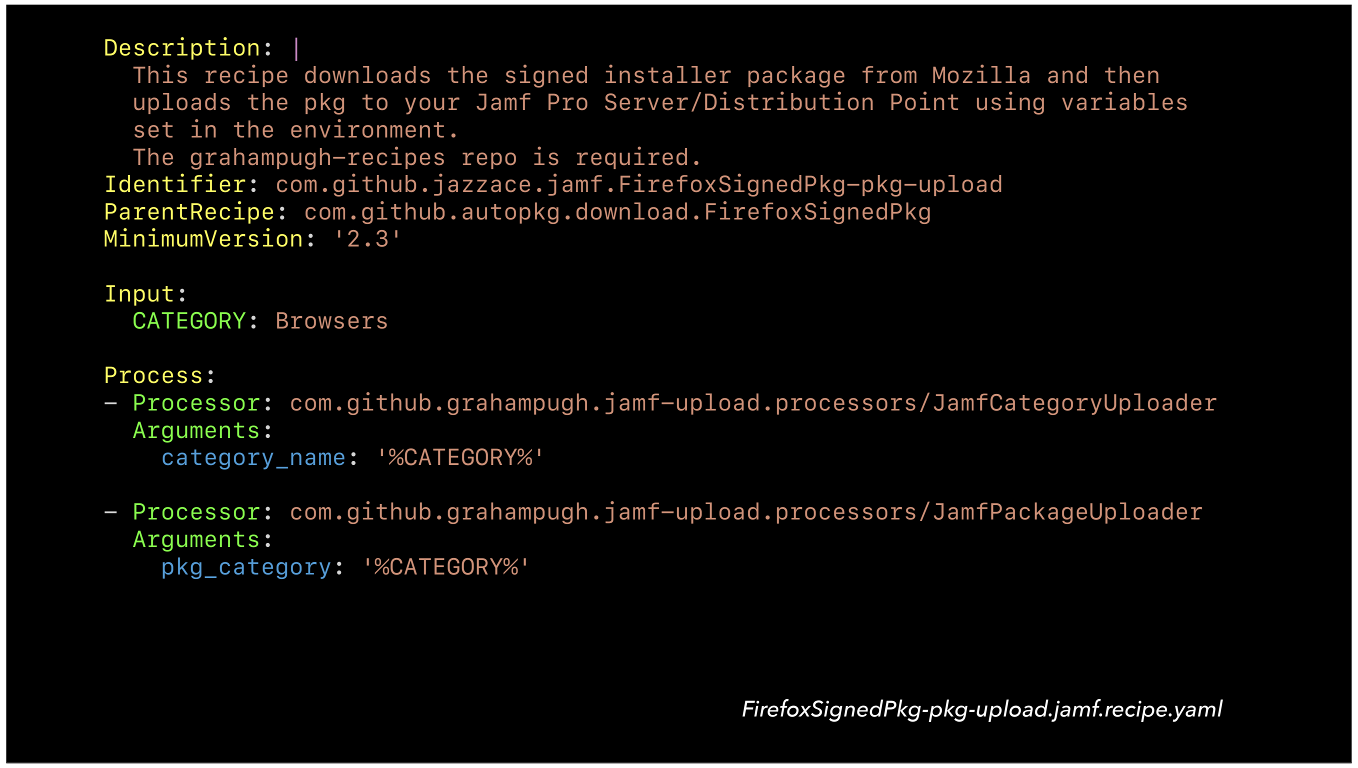


Jamf Pro and AutoPkg and…

- **Munki** (jamJAR)
- **JSSImporter** (for a few more months)
- PatchBot
- JamfUploader

maclabs.jazzace.ca/2020/12/29/integrating-autopkg-jamfpro.html

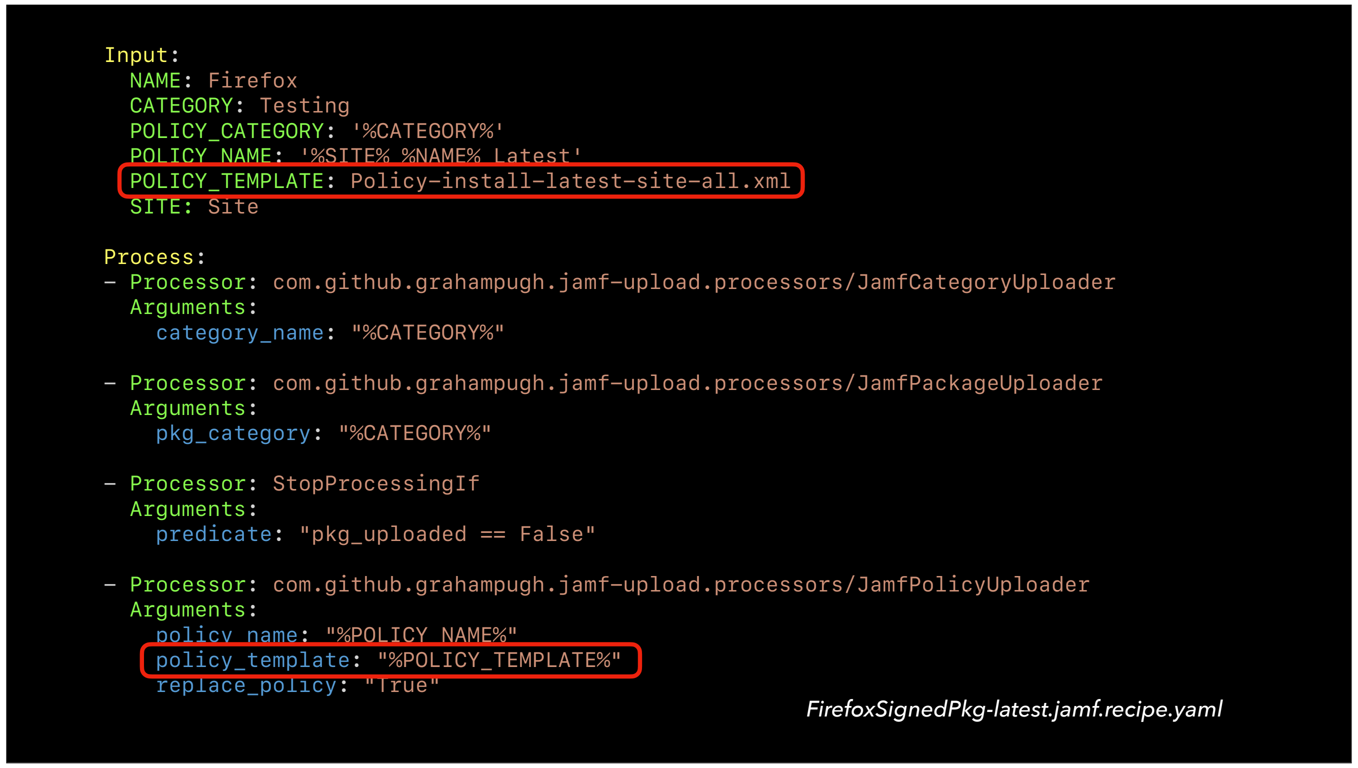
There are a lot of ways you can integrate AutoPkg with Jamf Pro depending on your needs. I wrote a blog post about this in Dec. 2020 if you want more detail on the considerations I made prior to deciding on JamfUploader. I found that JamfUploader was the most AutoPkg-like in design and you could use as little or as much as you wanted, just as Graham described earlier. It also had no additional dependencies like the rest of these tools have. And since I had a lot of experience writing recipes, this would be easy for me to adopt. So how did I get started?



```yaml
Description: |
  This recipe downloads the signed installer package from Mozilla and then
  uploads the pkg to your Jamf Pro Server/Distribution Point using variables
  set in the environment.
  The grahampugh-recipes repo is required.
Identifier: com.github.jazzace.jamf.FirefoxSignedPkg-pkg-upload
ParentRecipe: com.github.autopkg.download.FirefoxSignedPkg
MinimumVersion: '2.3'

Input:
  CATEGORY: Browsers

Process:
- Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader
  Arguments:
    category_name: '%CATEGORY%'

- Processor: com.github.grahampugh.jamf-upload.processors/JamfPackageUploader
  Arguments:
    pkg_category: '%CATEGORY%'
```

FirefoxSignedPkg-pkg-upload.jamf.recipe.yaml

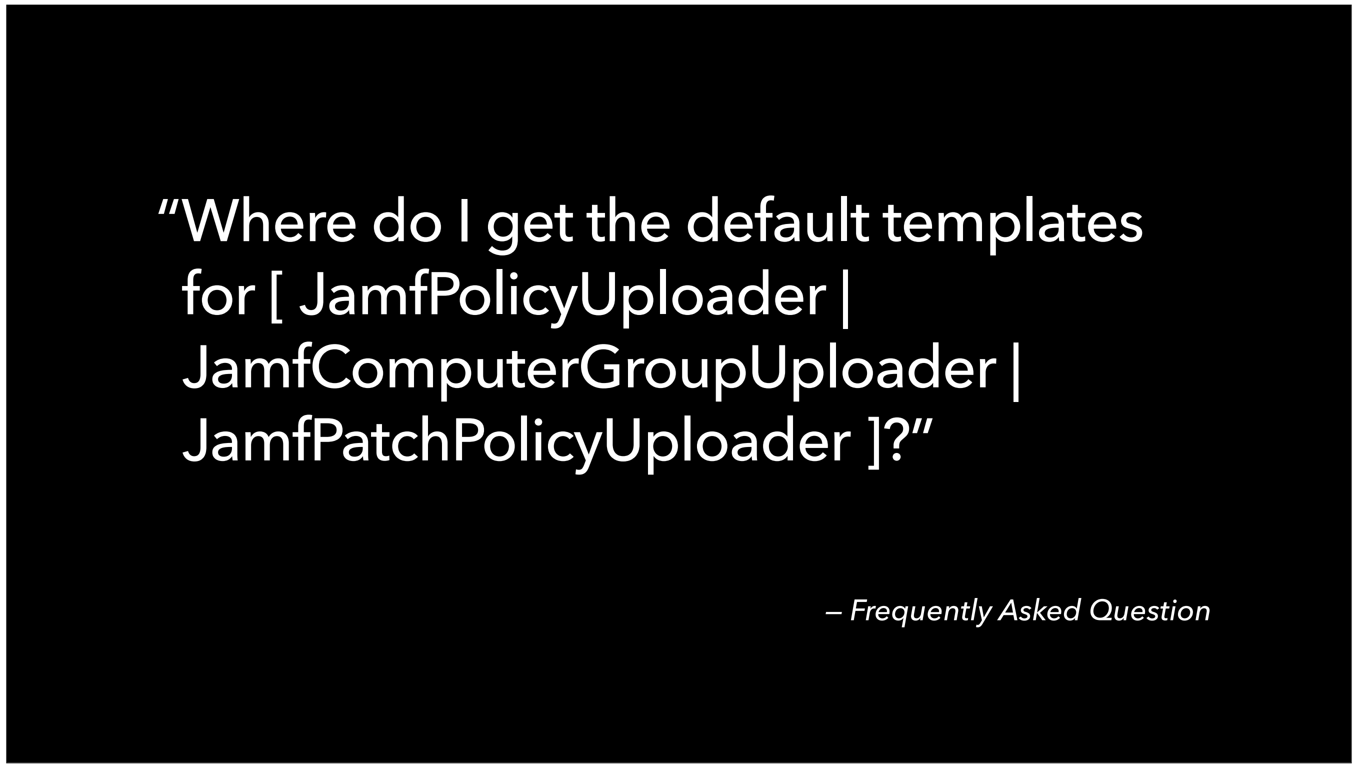
I started by writing recipes that simply got the latest pkg installer into Jamf Pro. As Graham described earlier, you can create a very simple recipe using one or two JamfUploader processors. Here’s what a simple -pkg-upload recipe looks like, with JamfCategoryUploader and JamfPackageUploader. The Category processor is optional if the category you want already exists on the server. In recipes in my repo, I do not use the JamfCategoryUploader processor. Note that I’m showing you this recipe in YAML format for ease of presentation, but you can also write your .jamf recipes in plist format, which I do.



```
Input:
  NAME: Firefox
  CATEGORY: Testing
  POLICY_CATEGORY: '%CATEGORY%'
  POLICY_NAME: '%SITE% %NAME% Latest'
  POLICY_TEMPLATE: Policy-install-latest-site-all.xml
  SITE: Site

Process:
- Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader
  Arguments:
    category_name: "%CATEGORY%"

- Processor: com.github.grahampugh.jamf-upload.processors/JamfPackageUploader
  Arguments:
    pkg_category: "%CATEGORY%"

- Processor: StopProcessingIf
  Arguments:
    predicate: "pkg_uploaded == False"

- Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader
  Arguments:
    policy_name: "%POLICY_NAME%"
    policy_template: "%POLICY_TEMPLATE%"
    replace_policy: "True"
```

FirefoxSignedPkg-latest.jamf.recipe.yaml

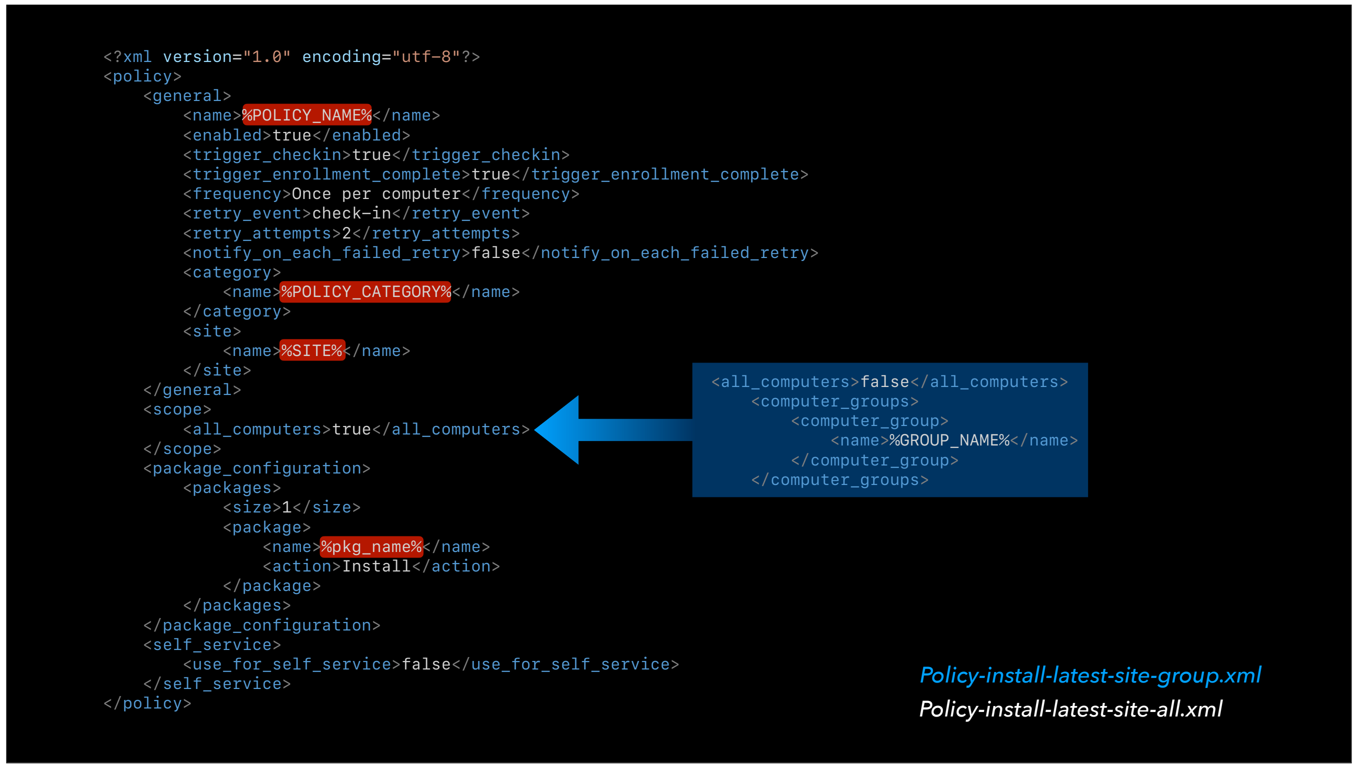
Next, I wanted to automatically create a policy that used the particular package I just uploaded. So I added the JamfPolicyUploader processor. As a safety measure, most such recipes add a StopProcessingIf processor just before changing or adding the Package to avoid messing with an existing policy. This is particularly important if you follow this with a step that flushes the policy using JamfPolicyLogFlusher, which would trigger an install based on whatever triggers you had in the policy. The JamfPolicyUploader processor is a little more complicated because it requires that you have a Policy Template. I describe this in some detail in our JNUC 2021 presentation including how you can use the Jamf Classic API to help model a template, but the short version is that you will want a few templates for the different scopes you have.



"Where do I get the default templates
for [JamfPolicyUploader |
JamfComputerGroupUploader |
JamfPatchPolicyUploader]?"

– Frequently Asked Question

The need for templates is true of a few JamfUploader processors. These templates consist of XML that gets sent to the Jamf Pro Server via the Classic API, but in true AutoPkg fashion, you use variables from the recipe to customize what you send to the server. There are models available in Graham's repo or my repo that you can copy, or you can model your own using the Classic API. There is no one template that will work for everyone, so there really is no default that most people can just *use.* Let's look at a possible Policy template for the recipe I showed you a moment ago.



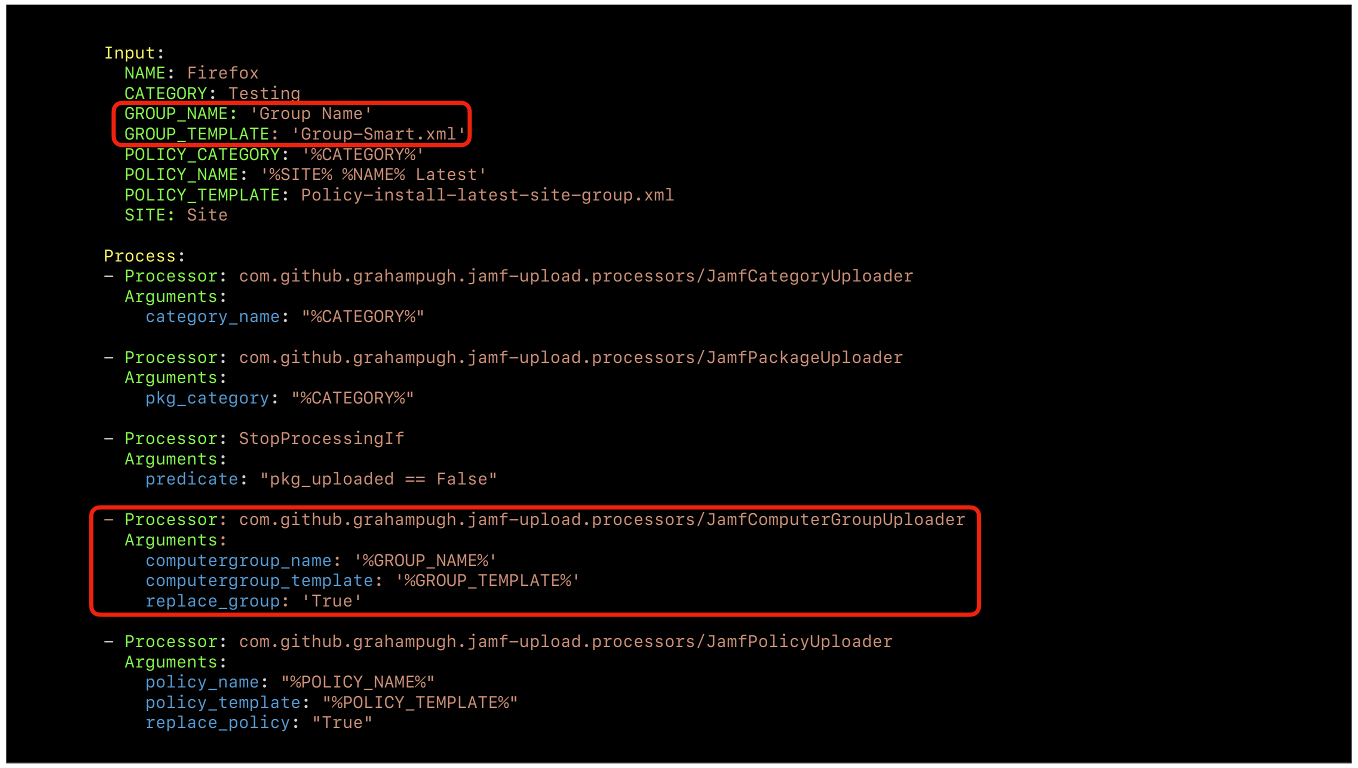
```
<?xml version="1.0" encoding="utf-8"?>
<policy>
    <general>
        <name>%POLICY_NAME%</name>
        <enabled>true</enabled>
        <trigger_checkin>true</trigger_checkin>
        <trigger_enrollment_complete>true</trigger_enrollment_complete>
        <frequency>Once per computer</frequency>
        <retry_event>check-in</retry_event>
        <retry_attempts>2</retry_attempts>
        <notify_on_each_failed_retry>false</notify_on_each_failed_retry>
        <category>
            <name>%POLICY_CATEGORY%</name>
        </category>
        <site>
            <name>%SITE%</name>
        </site>
    </general>
    <scope>
        <all_computers>true</all_computers>
    </scope>
    <package_configuration>
        <packages>
            <size>1</size>
            <package>
                <name>%pkg_name%</name>
                <action>Install</action>
            </package>
        </packages>
    </package_configuration>
    <self_service>
        <use_for_self_service>false</use_for_self_service>
    </self_service>
</policy>
```

```
<all_computers>false</all_computers>
    <computer_groups>
        <computer_group>
            <name>%GROUP_NAME%</name>
        </computer_group>
    </computer_groups>
```

Policy-install-latest-site-group.xml

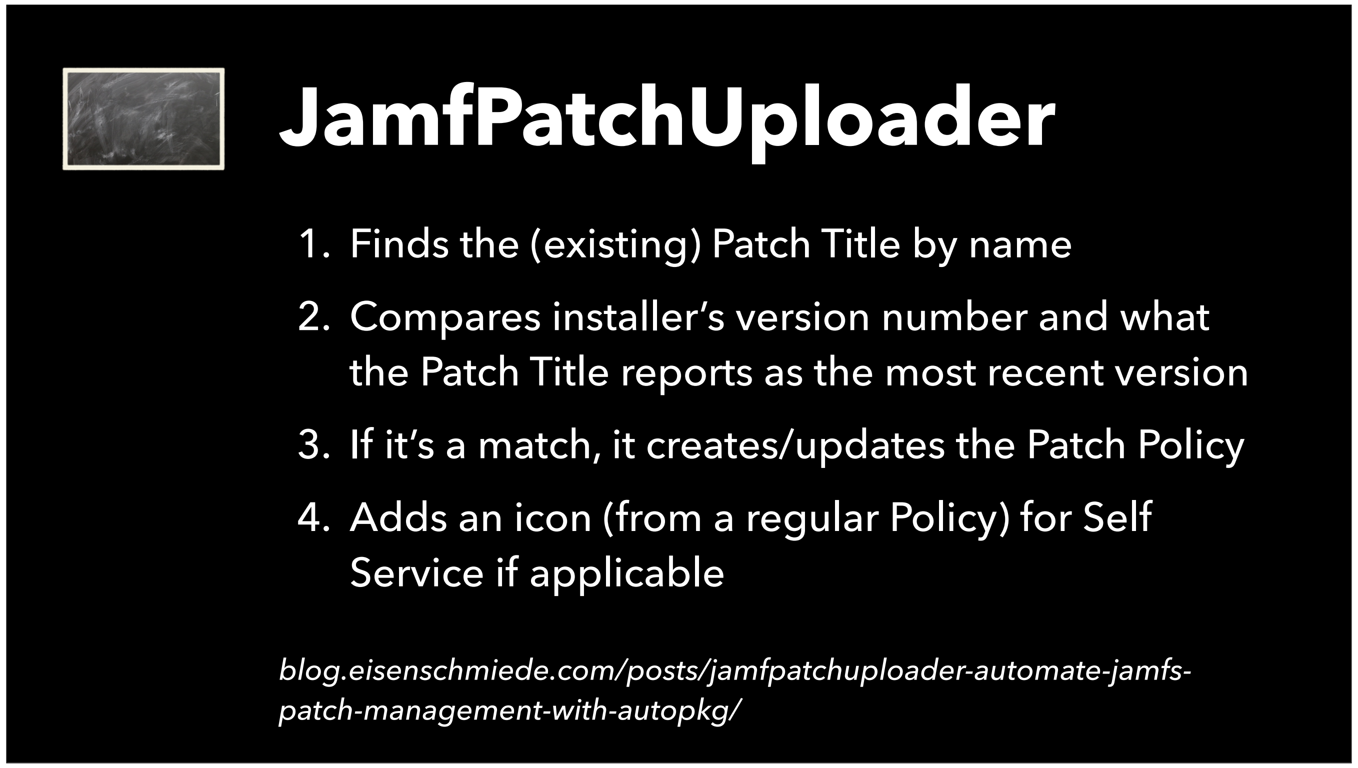
Policy-install-latest-site-all.xml

This template falls in line with the strategy I am trying to automate. [read through] The POLICY_NAME, POLICY_CATEGORY, and SITE are all supplied as input variables in the recipe and the pkg_name is generated via earlier processors. For those who have worked with AutoPkg recipes before, this is why there are variables in the Input dictionary that don't appear in the Process section of the recipe: they are needed for the template, and not directly in a processor. The most common variation I have on this template is to scope a Policy to a particular Group. I just make the substitution shown and save it to a different template. Note that the GROUP_NAME would need to be added to the Input variables in order to use this template.



```
Input:
  NAME: Firefox
  CATEGORY: Testing
  GROUP_NAME: 'Group Name'
  GROUP_TEMPLATE: 'Group-Smart.xml'
  POLICY_CATEGORY: '%CATEGORY%'
  POLICY_NAME: '%SITE% %NAME% Latest'
  POLICY_TEMPLATE: Policy-install-latest-site-group.xml
  SITE: Site

Process:
- Processor: com.github.grahampugh.jamf-upload.processors/JamfCategoryUploader
  Arguments:
    category_name: "%CATEGORY%"

- Processor: com.github.grahampugh.jamf-upload.processors/JamfPackageUploader
  Arguments:
    pkg_category: "%CATEGORY%"

- Processor: StopProcessingIf
  Arguments:
    predicate: "pkg_uploaded == False"

- Processor: com.github.grahampugh.jamf-upload.processors/JamfComputerGroupUploader
  Arguments:
    computergroup_name: '%GROUP_NAME%'
    computergroup_template: '%GROUP_TEMPLATE%'
    replace_group: 'True'

- Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader
  Arguments:
    policy_name: "%POLICY_NAME%"
    policy_template: "%POLICY_TEMPLATE%"
    replace_policy: "True"
```

So here is the same recipe but now scoped to a group. The changes are highlighted. And just as we can omit the JamfCategoryUploader processor if we always want to use an existing Category, we can omit the JamfComputerGroupUploader processor and the GROUP_TEMPLATE input key if we are willing to use an existing Smart or Static Group as the scope of our policy. For simplicity's sake, that is what I do, so my most common .jamf recipes only have three processor steps: JamfPackageUploader, StopProcessingIf, and JamfPolicyUploader. But what about when I want to use Patch Management?



JamfPatchUploader

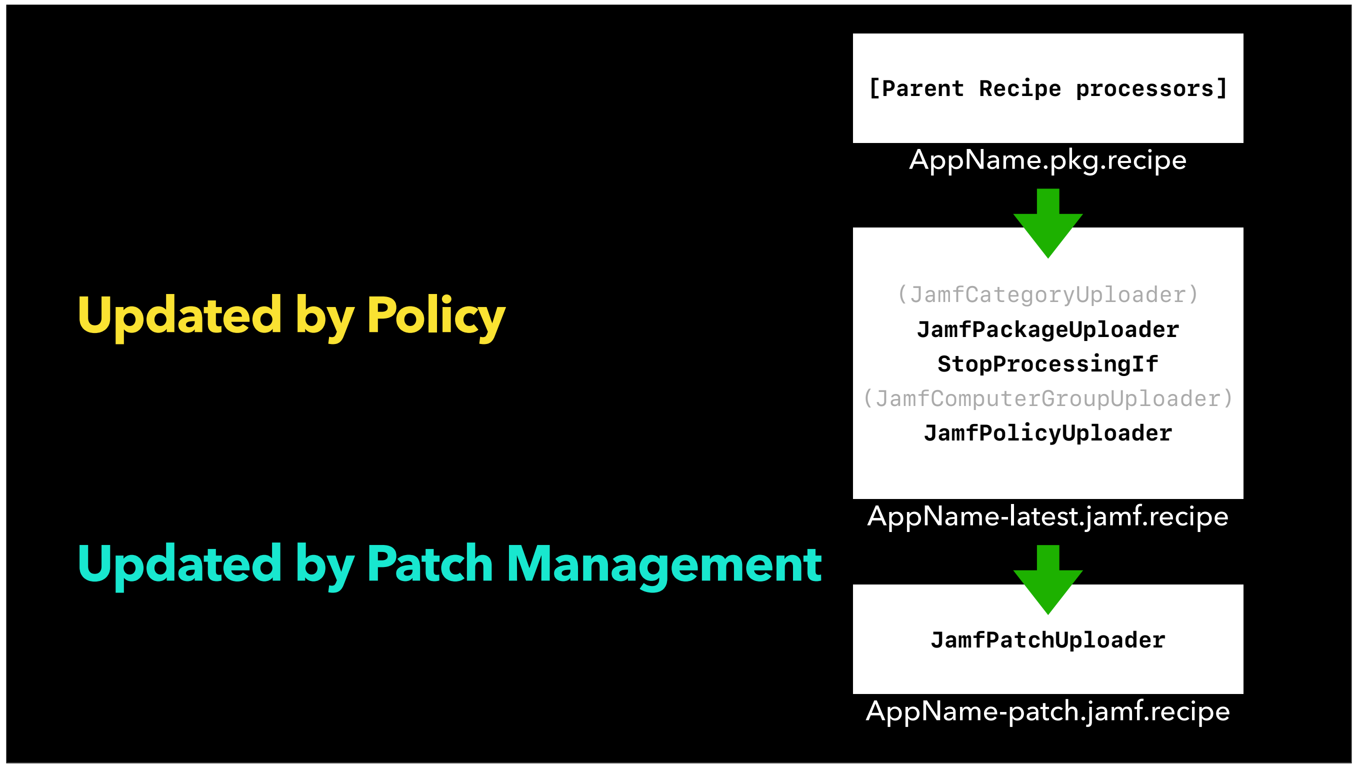
1. Finds the (existing) Patch Title by name
2. Compares installer’s version number and what the Patch Title reports as the most recent version
3. If it’s a match, it creates/updates the Patch Policy
4. Adds an icon (from a regular Policy) for Self Service if applicable

blog.eisenschmiede.com/posts/jamfpatchuploader-automate-jamfs-patch-management-with-autopkg/

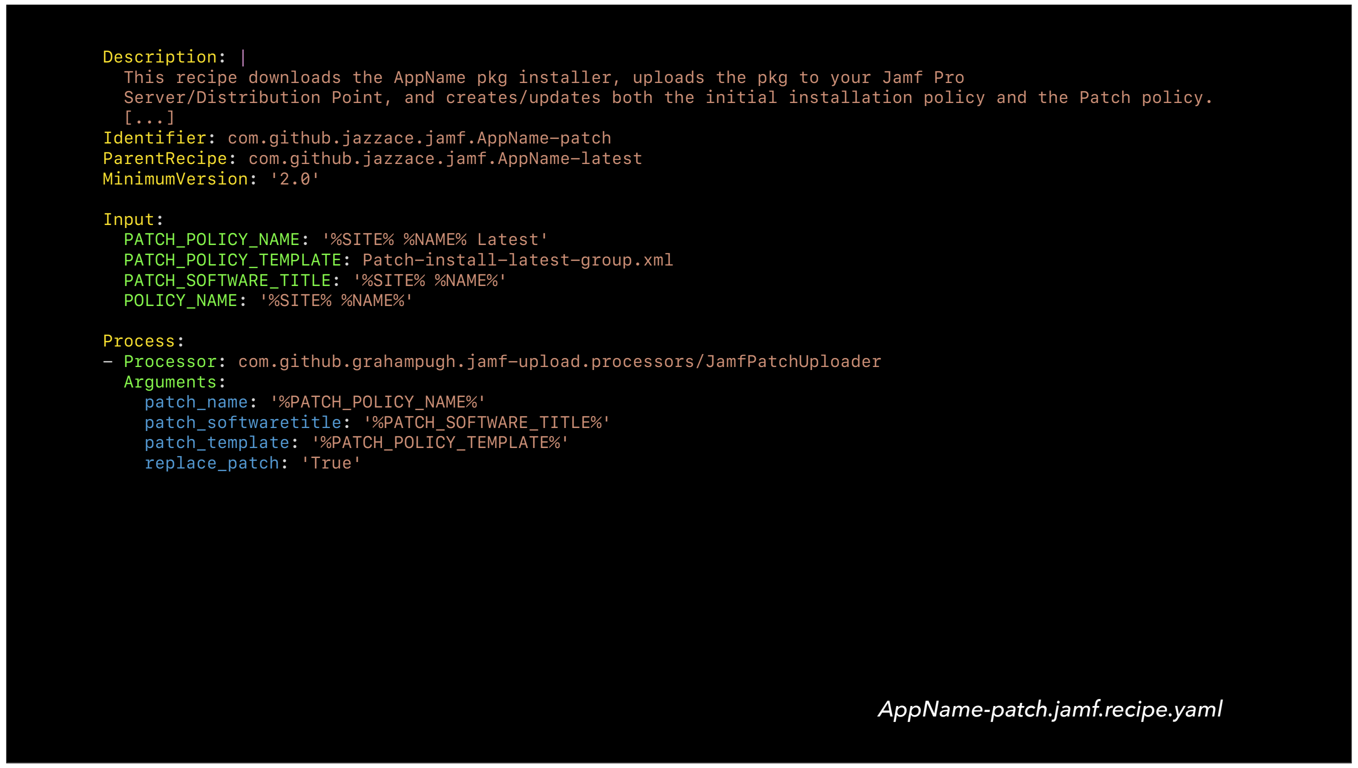
Marcel Keßler contributed JamfPatchUploader to this project, which tries to take care of all the manual steps you would do to update a Patch Title when a new version comes in, as I've shown on screen. Using this processor has saved me a lot of time, but there are some common issues you should be aware of that correspond to each of the steps noted on screen:

1. JamfPatchUploader can’t select the patch title definition from Jamf's catalogue. So you must manually go into Patch Management and create the title before running the recipe. You will need to feed the processor the exact name of the Patch Title using an Input variable. Luckily, you only have to do this once per title.
2. In some rare cases, the version that your pkg recipe uses and puts in the version variable does not match the form of the Patch Title. You may need to use a shared processor like VersionSplitter or FindAndReplace from Elliot Jordan to massage the version string to match what Patch Management provides.
3. Just like with regular Policies, you’ll need a Patch Policy template for JamfPatchUploader to do its magic.
4. You can't reference an icon directly via the API, so you have to specify a regular policy that uses that icon. That could be a Self Service policy from earlier in the recipe chain. I don’t use Self Service, so I don’t need an icon, but you need to include the icon fields in your template.

Marcel wrote a detailed blog post about the processor this past February that deals with many more technical details and use cases. It’s a great reference, particularly if your workflow is different than mine for Patch.


Updated by Policy
Updated by Patch Management
[Parent Recipe processors]
AppName.pkg.recipe
(JamfCategoryUploader)
JamfPackageUploader
StopProcessingIf
(JamfComputerGroupUploader)
JamfPolicyUploader
AppName-latest.jamf.recipe
JamfPatchUploader
AppName-patch.jamf.recipe

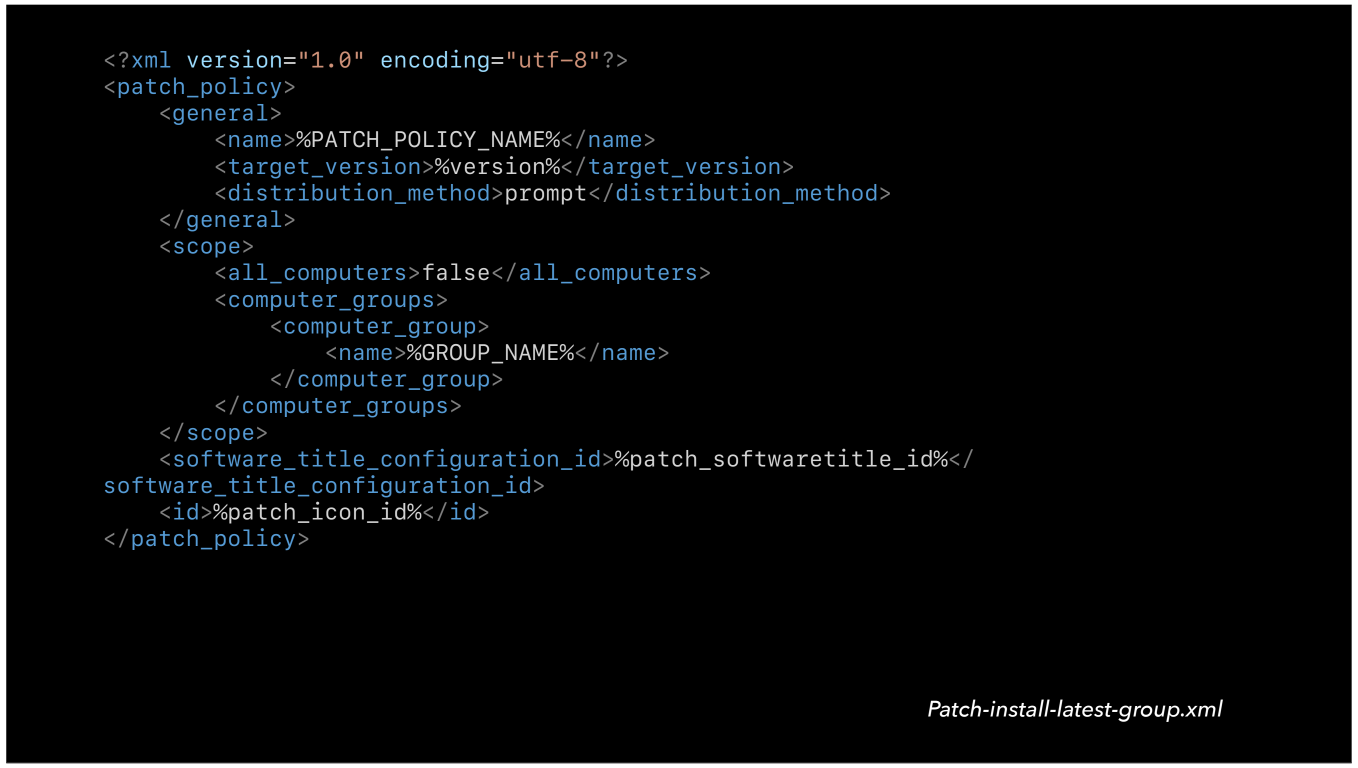
As I mentioned earlier, Patch only works on titles that are already installed. So I still want to have a child recipe that uploads the package and creates or updates the policy associated with it. For titles that I want to use with Patch, I have an additional child recipe that creates or updates the Patch Policy of an existing title. Even though these are both “.jamf” recipes, you can distinguish between the two recipe types by the label just before “.jamf”. This is a convention Graham started when he originally released JamfUploader. If the recipe didn’t match the old .jss recipe functionality, then he added a qualifying label within the filename. The basic format is AppName-label.jamf. For example, for a recipe that simply uploaded a package with its category, Graham labelled it with “-pkg-upload.” I label my package + policy recipes with “-latest” since they update packages and policies with the latest version. I label my Patch-related child recipes with “-patch”. If I were to add a JamfPolicyLogFlusher step to a -latest recipe, I would put that step in a child recipe with a “-flush” label. Feel free to adopt what other people are using if your usage is a match, but it is more important that you use an internally consistent nomenclature, particularly if you choose to share your recipes publicly.



```
Description: |
  This recipe downloads the AppName pkg installer, uploads the pkg to your Jamf Pro
  Server/Distribution Point, and creates/updates both the initial installation policy and the Patch policy.
  [...]
Identifier: com.github.jazzace.jamf.AppName-patch
ParentRecipe: com.github.jazzace.jamf.AppName-latest
MinimumVersion: '2.0'

Input:
  PATCH_POLICY_NAME: '%SITE% %NAME% Latest'
  PATCH_POLICY_TEMPLATE: Patch-install-latest-group.xml
  PATCH_SOFTWARE_TITLE: '%SITE% %NAME%'
  POLICY_NAME: '%SITE% %NAME%'

Process:
- Processor: com.github.grahampugh.jamf-upload.processors/JamfPatchUploader
  Arguments:
    patch_name: '%PATCH_POLICY_NAME%'
    patch_softwaretitle: '%PATCH_SOFTWARE_TITLE%'
    patch_template: '%PATCH_POLICY_TEMPLATE%'
    replace_patch: 'True'
```

AppName-patch.jamf.recipe.yaml

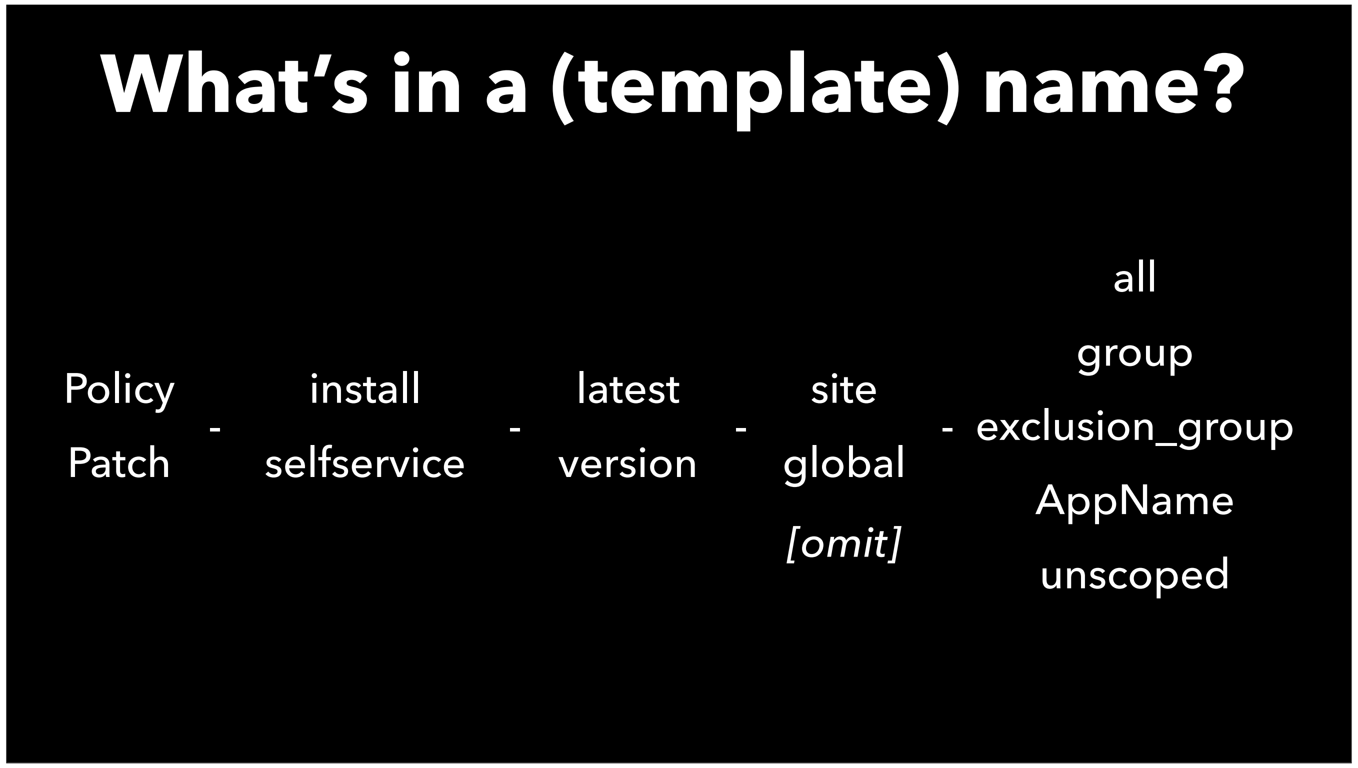
So my -patch recipes are very simple! The recipe that establishes the policy — what I call -latest — is the parent. I add one processor step and provide the Input variables needed for the processor to do its work. In all my cases, the scoping for the regular policy is the same as for the Patch Policy, so I can just use the existing value for the GROUP_NAME variable if needed. If you wanted something different, you could add one or more Input variables for use in your patch template.



```
<?xml version="1.0" encoding="utf-8"?>
<patch_policy>
    <general>
        <name>%PATCH_POLICY_NAME%</name>
        <target_version>%version%</target_version>
        <distribution_method>prompt</distribution_method>
    </general>
    <scope>
        <all_computers>false</all_computers>
        <computer_groups>
            <computer_group>
                <name>%GROUP_NAME%</name>
            </computer_group>
        </computer_groups>
    </scope>
    <software_title_configuration_id>%patch_softwaretitle_id%</
software_title_configuration_id>
    <id>%patch_icon_id%</id>
</patch_policy>
```

Patch-install-latest-group.xml

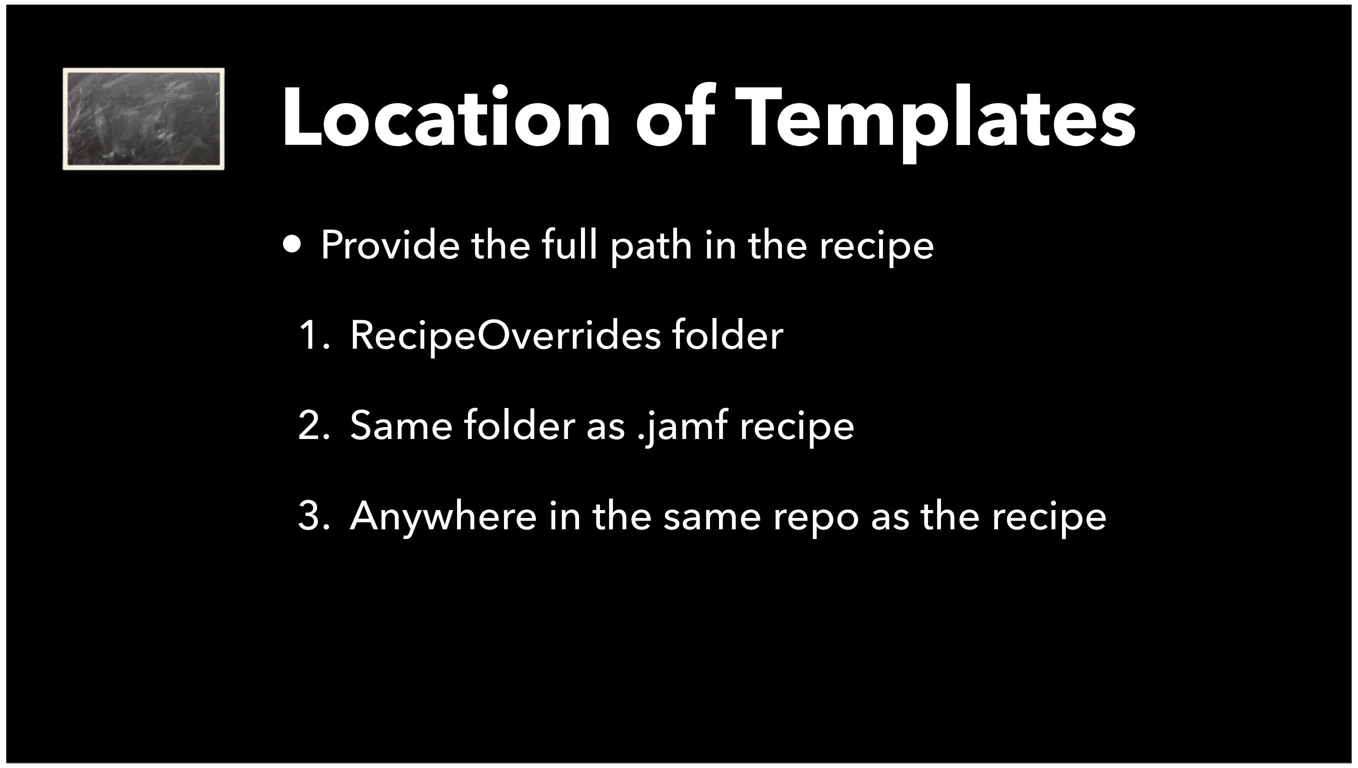
Patch Policy templates are much simpler than regular policy templates. The only thing that makes it more or less complicated in my case is the scoping. This is my template for scoping to a particular group. Graham has two sample patch templates in his repo, including one for Self Service, that shows all the fields available in Patch Policies. Those are a good reference if you want something more elaborate than what I have here.



What’s in a (template) name?

								all
								group
Policy	-	install	-	latest	-	site	-	exclusion_group
Patch		selfservice		version		global		AppName
						[omit]		unscoped

I mentioned my naming convention for recipes. This is my naming convention for Policy and Patch Policy templates. [Go through it.] If I created Groups with templates, I would follow the same pattern. As you can see, when someone asks about the default template for .jamf recipes, there is no one good answer because the permutations are significant. The flexibility of JamfUploader means that you can get exactly what you want in an automated fashion.

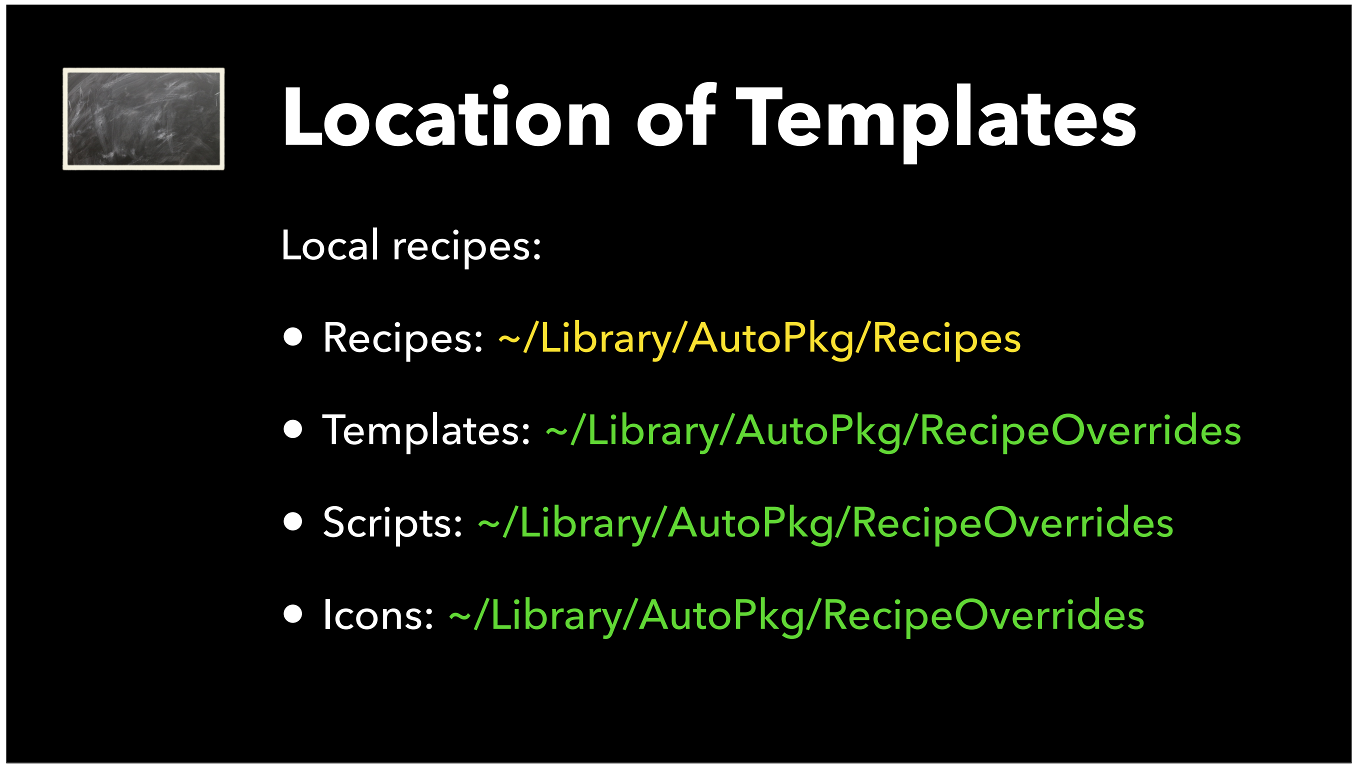


Location of Templates

- Provide the full path in the recipe

1. RecipeOverrides folder
2. Same folder as .jamf recipe
3. Anywhere in the same repo as the recipe

Finally, another common question is about where the templates should be stored so that the JamfUploader processors will find them.
If you provide a full path to a template, it will use that path. If you supply only the filename, then there is a hierarchy of locations where JamfUploader will look. Top of the list is the RecipeOverrides folder within the AutoPkg working directory; next is in the same folder as the .jamf recipe you overrode; and finally, JamfUploader looks anywhere in the same repo as the recipe being run, so that you can put all your common templates in a common folder.

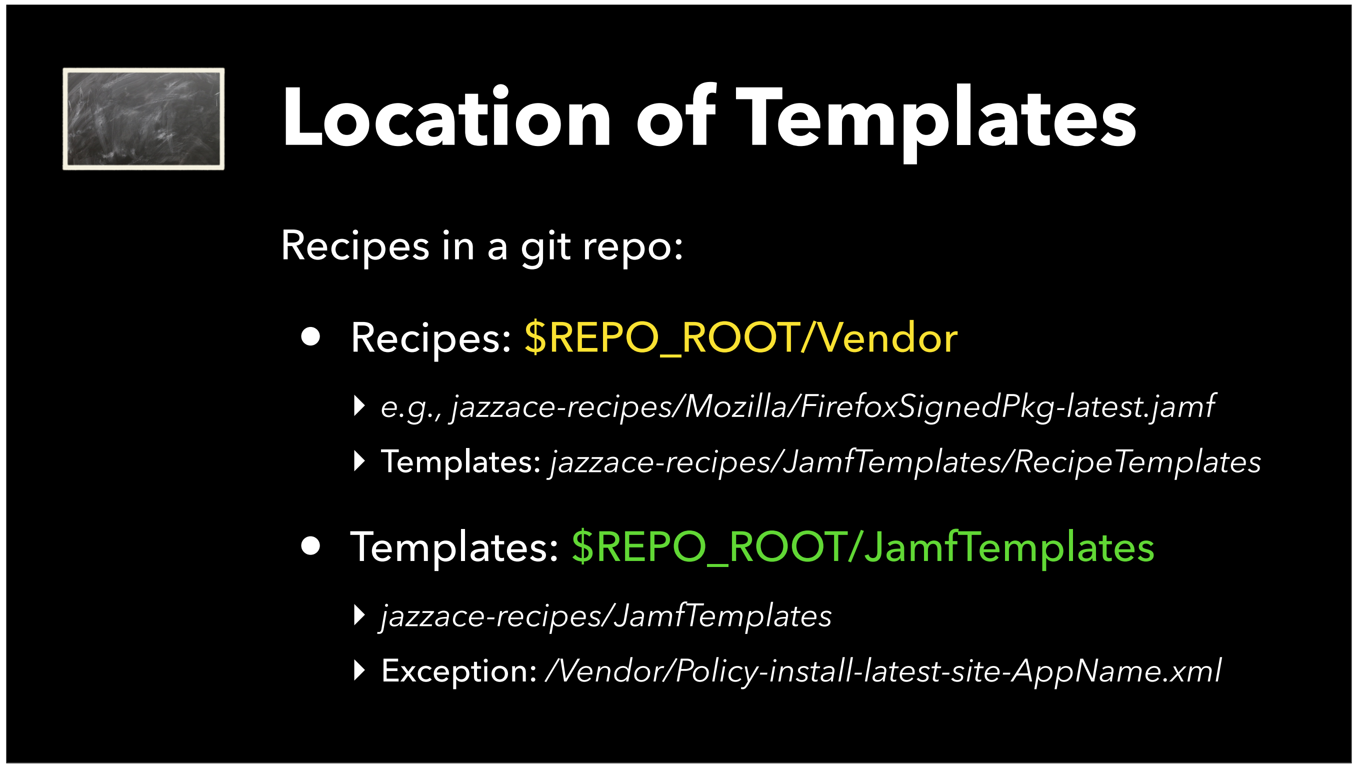


Location of Templates

Local recipes:

- Recipes: ~/Library/AutoPkg/Recipes
- Templates: ~/Library/AutoPkg/RecipeOverrides
- Scripts: ~/Library/AutoPkg/RecipeOverrides
- Icons: ~/Library/AutoPkg/RecipeOverrides

My recommendation if you are just starting out is to put the recipes you are currently writing into the Recipes folder within ~/Library/AutoPkg. If that folder doesn’t exist, create it. Then, the templates, and any scripts and icons can go in RecipeOverrides folder along with the recipe overrides themselves. They can run from those locations permanently if you don't want or need to make them public. This is how I have worked for most of the past year.



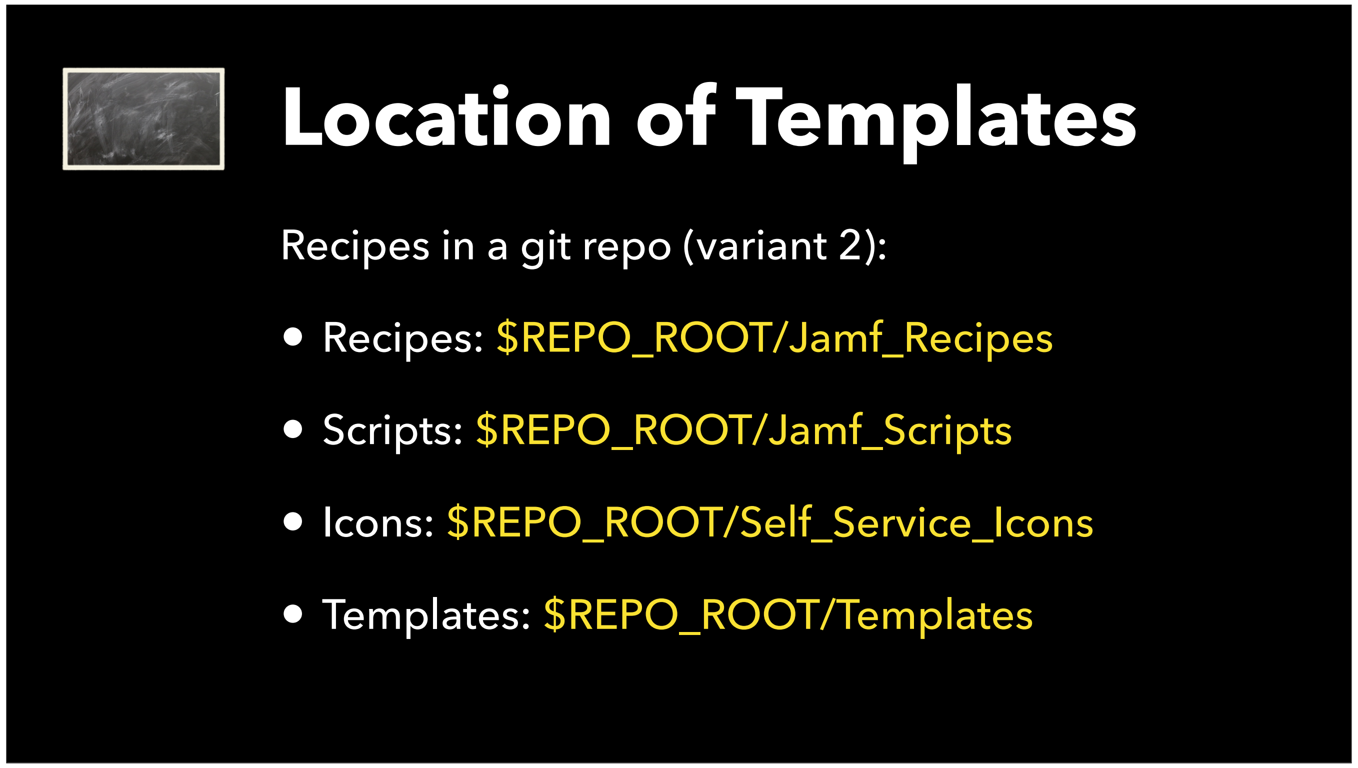
Location of Templates

Recipes in a git repo:

- Recipes: $REPO_ROOT/Vendor
  - *e.g., jazzace-recipes/Mozilla/FirefoxSignedPkg-latest.jamf*
  - Templates: *jazzace-recipes/JamfTemplates/RecipeTemplates*
- Templates: $REPO_ROOT/JamfTemplates
  - *jazzace-recipes/JamfTemplates*
  - Exception: */Vendor/Policy-install-latest-site-AppName.xml*

I am now ready to share my recipes and templates, so I have been moving the recipes and templates to my jazzace-recipes repo in the AutoPkg project. I put my jamf recipes inside a folder named after the vendor of the application, right beside my download and pkg recipes where applicable. I write most of my recipes using a starter template, which I have also shared in my repo.
My Policy and Patch Policy templates are in a folder named JamfTemplates. The only time I don't put them there is if I have a template that is specific to an app, usually due to unusual scoping or policy attributes. Those go with the app.
Note that once you move your local recipes to a git repo, you should remove the policy templates, icons and scripts from the RecipeOverrides folder, and any recipes from the local Recipes folder, since AutoPkg will see those copies before the copies you now have in your GitHub repo.

I know Graham has *other* ideas to share about how you might organize your JamfUploader-related resources, much of which is similar to how the old .jss recipes were organized, so this is a good time to hand it over to him.



Location of Templates

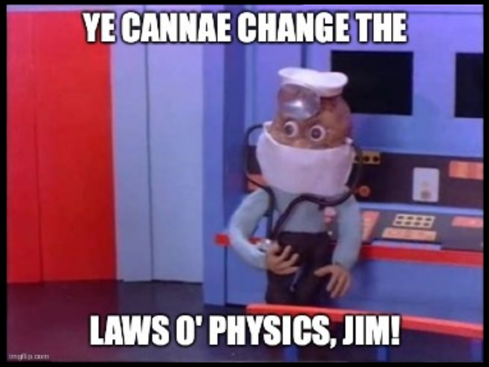
Recipes in a git repo (variant 2):

- Recipes: $REPO_ROOT/Jamf_Recipes
- Scripts: $REPO_ROOT/Jamf_Scripts
- Icons: $REPO_ROOT/Self_Service_Icons
- Templates: $REPO_ROOT/Templates

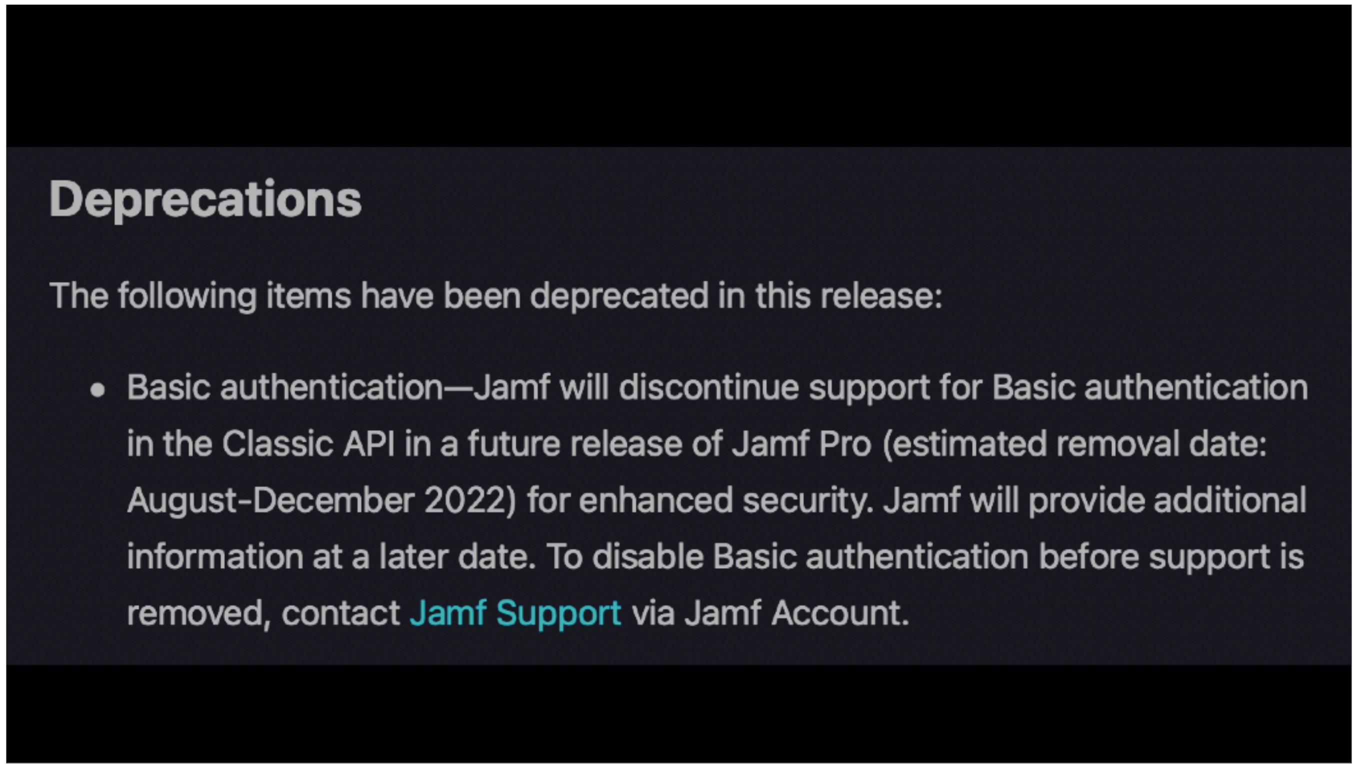
[Graham:] Thanks Anthony. Another option for organising the files in your git repo is by type - for example, different folders for each recipe type, and separate folders for scripts, icons and templates. This is how my repo is organised.

The organisation is up to you. The only rule here is that AutoPkg will only look into a top-level folder for recipes, so don't put recipes in folders within folders. JamfUploader will look deeper into folders for templates, icons and scripts though.

Converting from JSSImporter



Alright, so Anthony has covered how Jamf recipes can be constructed from scratch, and how you might organise your files.
But like many of you watching, I already had a bunch of JSS recipes, and you might be wondering - why change?
So I want to talk about why - and how - you should go about converting to Jamf recipes.
I have been maintaining the JSSImporter project for the past few years, after taking it off Shea Craig's hands after he stopped working for a company that used Jamf.
A year or two ago, I really struggled to make JSSImporter compatible with python 3 - in fact I was saved only by some great detective work by Elliot Jordan. It became clear to me that I was not clever enough to properly understand the underlying framework, and my calls for collaborators were not successful - this left our organisation vulnerable - we depended on JSSImporter for all Mac packaging, and if some future change to Jamf Pro or the macOS system broke the code in a way I couldn't fix, we could be in trouble at short notice.
With no obvious alternative to JSSImporter that integrates with AutoPkg, I decided that it was time to bite the bullet and write a replacement that better suited our organisation, and this project turned into the JamfUploader processors.



Deprecations

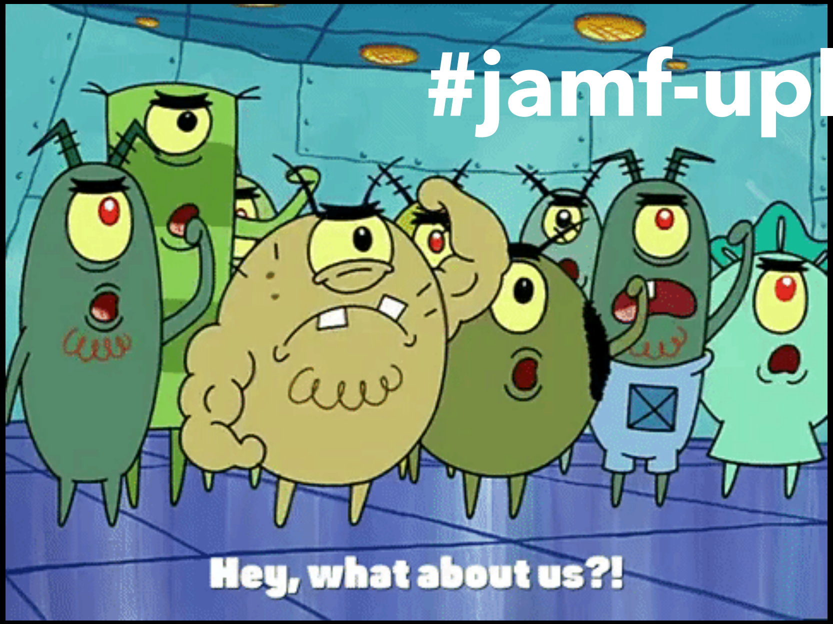
The following items have been deprecated in this release:

- Basic authentication—Jamf will discontinue support for Basic authentication in the Classic API in a future release of Jamf Pro (estimated removal date: August-December 2022) for enhanced security. Jamf will provide additional information at a later date. To disable Basic authentication before support is removed, contact Jamf Support via Jamf Account.

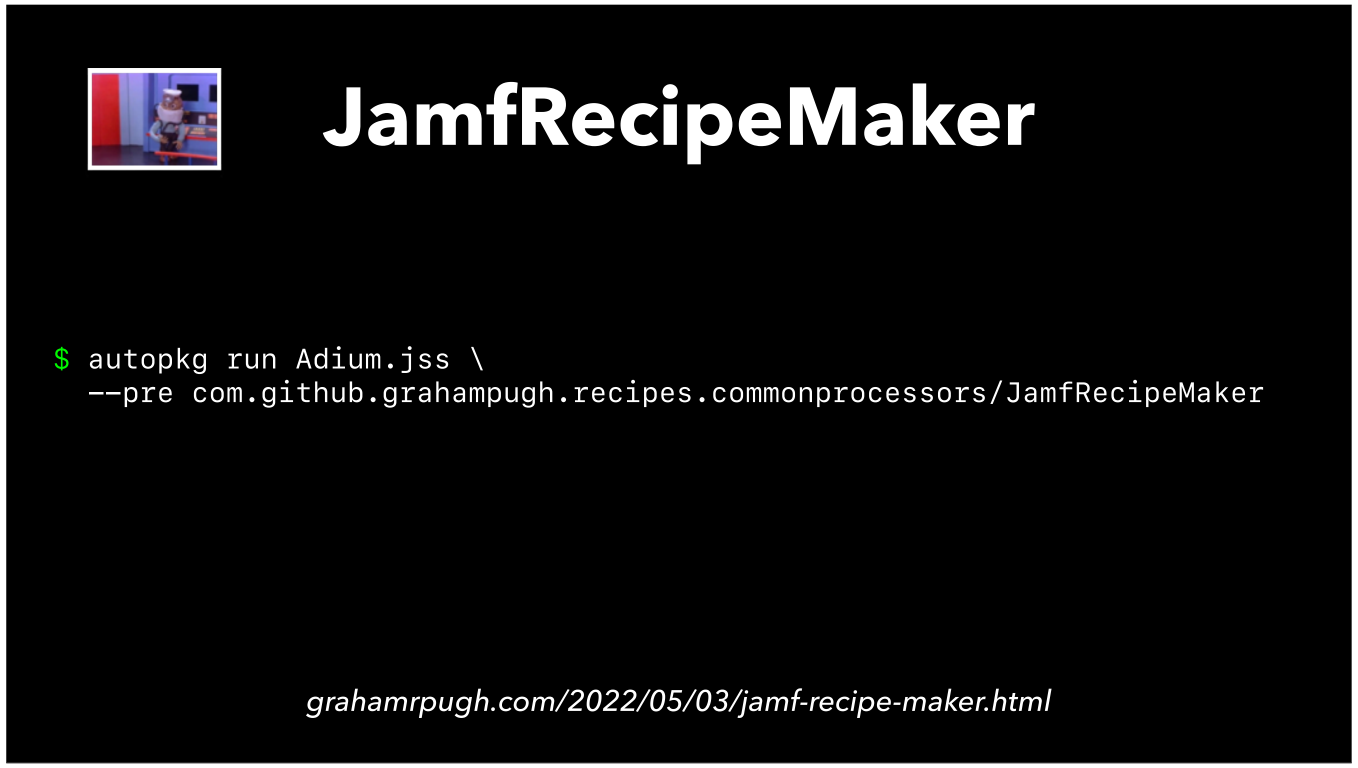
And then, in December of last year, Jamf stated that basic authentication would be discontinued between August and December of 2022.
It was easy for me to change the code for JamfUploader to accommodate the new Bearer Token authentication method, because that's my simple code. But I was not able to figure it out with JSSImporter's complicated underlying python framework. This was the breaking change that I feared.

I decided to concentrate my time on converting all the remaining existing in-house JSS recipes to Jamf recipes, so that my organisation would be ready for Bearer Token auth before August 2022, and not require JSSImporter any more.
I achieved this a few weeks ago, and we no longer use JSSImporter at all.

#jamf-upload



For those of you still using JSSImporter, it's definitely time to plan your migration to another solution. There are a few tools out there to help you, and I've been trying to contribute to that transition both in terms of some new tools, and advice in the MacAdmins Slack, along with others in the #jamf-upload channel including Anthony. I'd like to introduce one tool now, called JamfRecipeMaker.

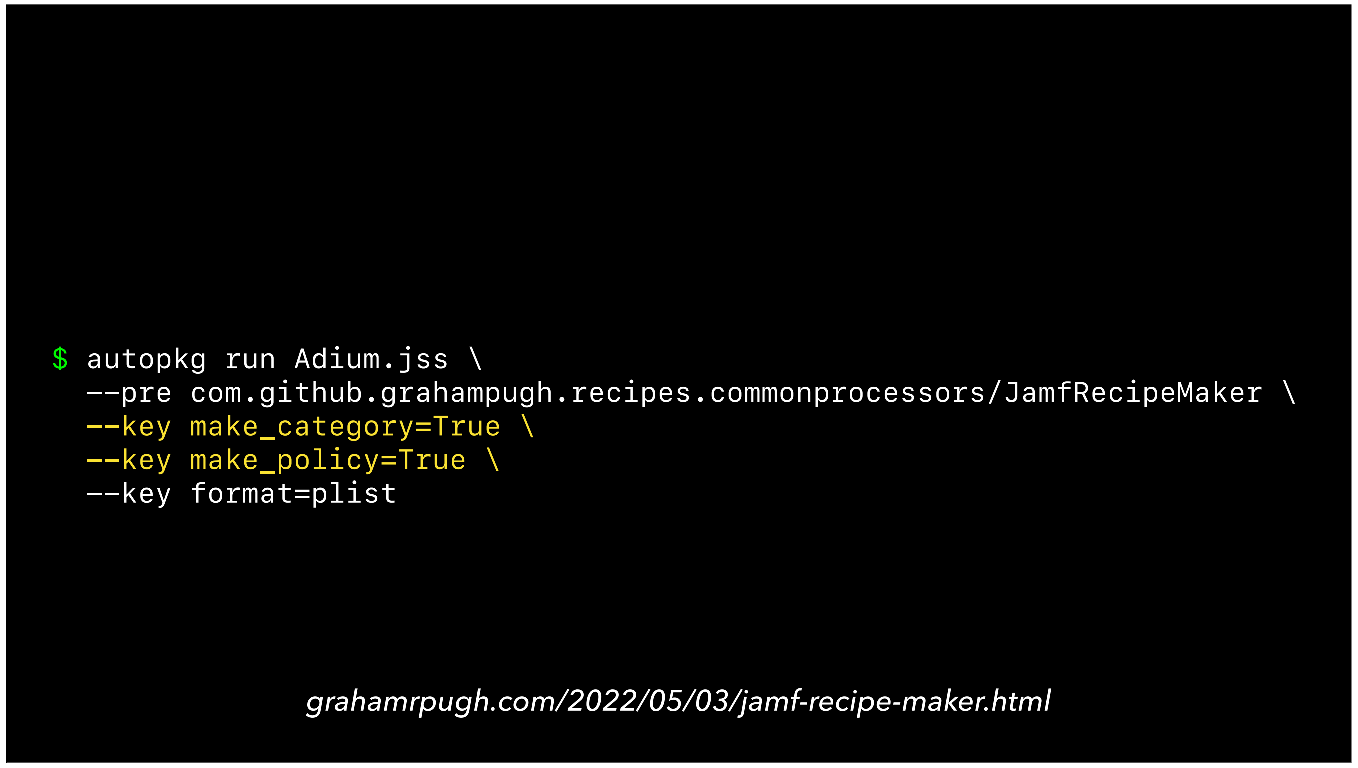


JamfRecipeMaker

```
$ autopkg run Adium.jss \
  --pre com.github.grahampugh.recipes.commonprocessors/JamfRecipeMaker
```

grahamrpugh.com/2022/05/03/jamf-recipe-maker.html

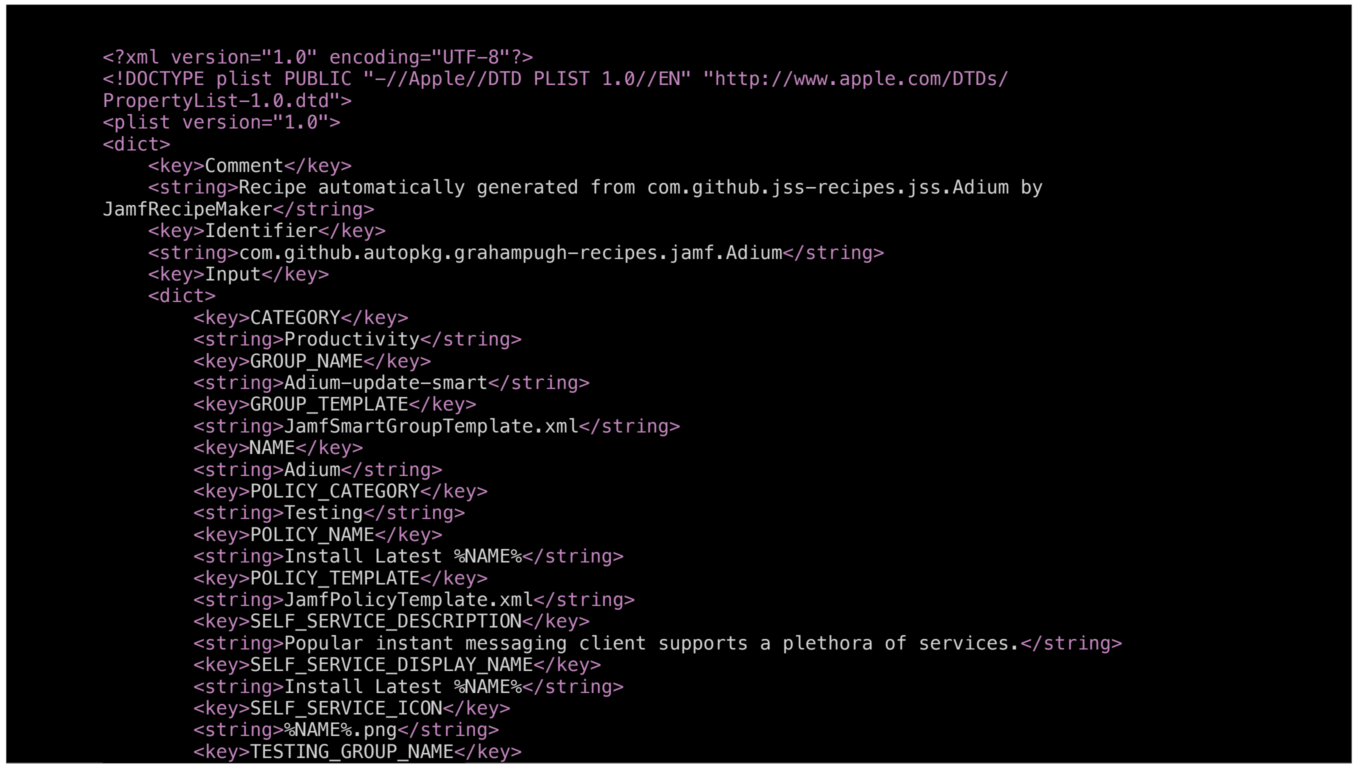
JSSRecipeMaker is a tool for taking the inputs from a JSS or PKG recipe and using them to construct an equivalent Jamf recipe. It is an AutoPkg processor that you can add to your autopkg run command as a pre-processor step. The default options result in the processor creating a recipe that only uploads the package.
By default it writes the recipes in yaml format, but if you prefer the traditional plist format, you can add the format key to output to plist.



```
$ autopkg run Adium.jss \
  --pre com.github.grahampugh.recipes.commonprocessors/JamfRecipeMaker \
  --key make_category=True \
  --key make_policy=True \
  --key format=plist
```

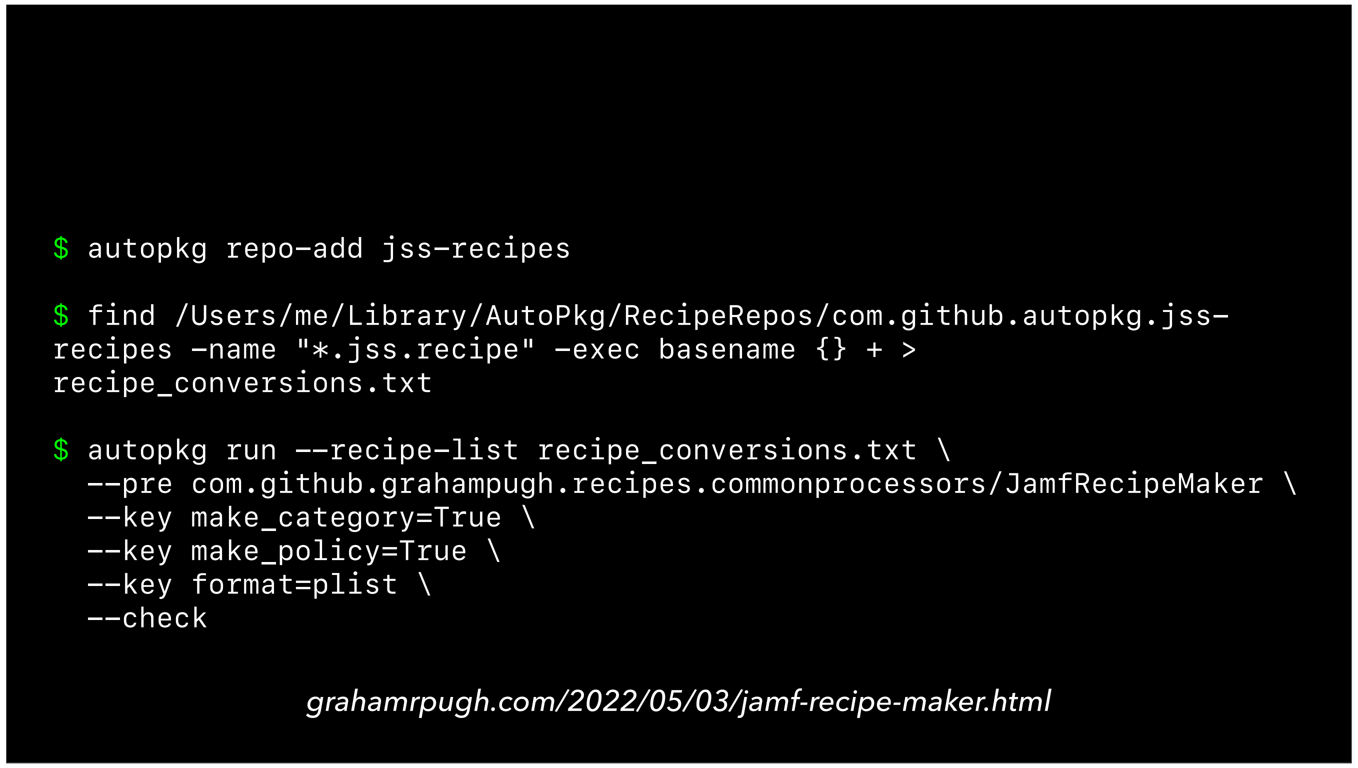
grahamrpugh.com/2022/05/03/jamf-recipe-maker.html

If you add the keys make_policy = True and make_category = True, then it will spit out a full recipe that also creates the policy, smart group and category - all the same things as the existing JSS recipe.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Comment</key>
    <string>Recipe automatically generated from com.github.jss-recipes.jss.Adium by
JamfRecipeMaker</string>
    <key>Identifier</key>
    <string>com.github.autopkg.grahampugh-recipes.jamf.Adium</string>
    <key>Input</key>
    <dict>
        <key>CATEGORY</key>
        <string>Productivity</string>
        <key>GROUP_NAME</key>
        <string>Adium-update-smart</string>
        <key>GROUP_TEMPLATE</key>
        <string>JamfSmartGroupTemplate.xml</string>
        <key>NAME</key>
        <string>Adium</string>
        <key>POLICY_CATEGORY</key>
        <string>Testing</string>
        <key>POLICY_NAME</key>
        <string>Install Latest %NAME%</string>
        <key>POLICY_TEMPLATE</key>
        <string>JamfPolicyTemplate.xml</string>
        <key>SELF_SERVICE_DESCRIPTION</key>
        <string>Popular instant messaging client supports a plethora of services.</string>
        <key>SELF_SERVICE_DISPLAY_NAME</key>
        <string>Install Latest %NAME%</string>
        <key>SELF_SERVICE_ICON</key>
        <string>%NAME%.png</string>
        <key>TESTING_GROUP_NAME</key>
```

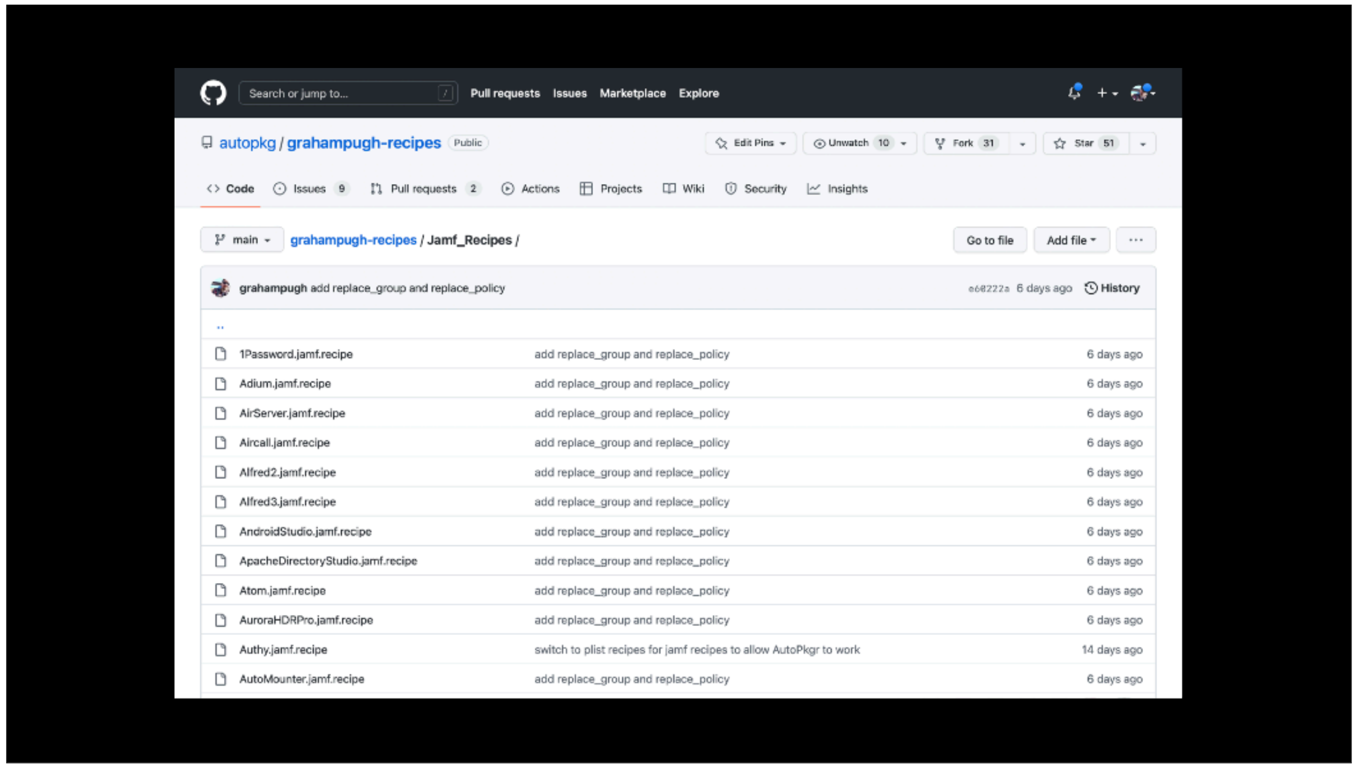
This is the recipe that it spits out. It creates the category, package, smart group and policy. Just be aware that if the JSS recipe had any extension attributes or scripts in it, these won't be migrated - you would have to add those to the recipe manually. But it's a good start...



```
$ autopkg repo-add jss-recipes

$ find /Users/me/Library/AutoPkg/RecipeRepos/com.github.autopkg.jss-
recipes -name "*.jss.recipe" -exec basename {} + >
recipe_conversions.txt

$ autopkg run --recipe-list recipe_conversions.txt \
  --pre com.github.grahampugh.recipes.commonprocessors/JamfRecipeMaker \
  --key make_category=True \
  --key make_policy=True \
  --key format=plist \
  --check
```

grahamrpugh.com/2022/05/03/jamf-recipe-maker.html

Since AutoPkg can run entire recipe lists in one run, with just these 3 steps, I was able to generate a recipe list of all the 100 or so recipes in the jss-recipes repo, and convert them all to jamf recipes in just a few minutes. Perhaps you could do the same with your own repo of JSS recipes.



autopkg / grahampugh-recipes Public

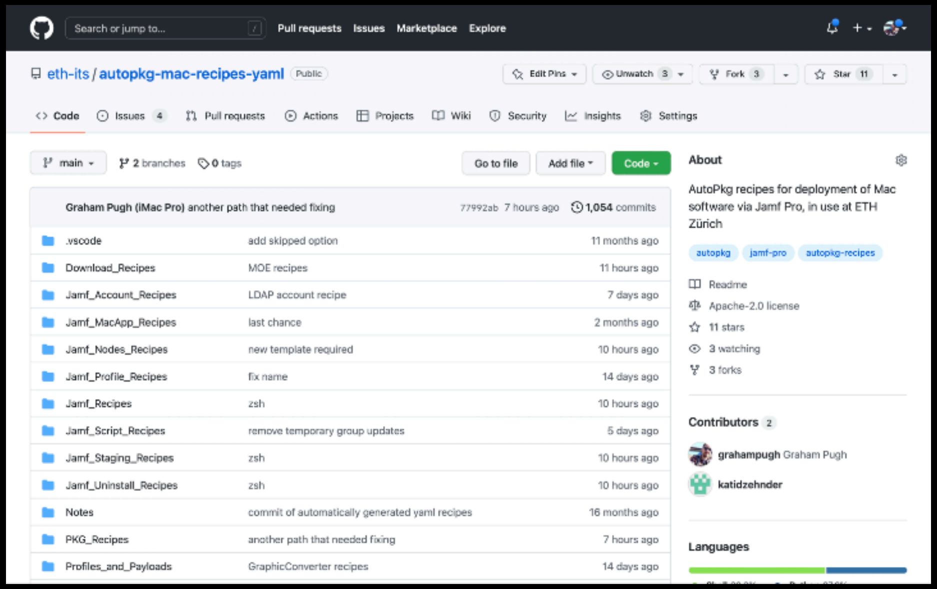
grahampugh-recipes / Jamf_Recipes /

grahampugh add replace_group and replace_policy e68222a 6 days ago

..		
1Password.jamf.recipe	add replace_group and replace_policy	6 days ago
Adium.jamf.recipe	add replace_group and replace_policy	6 days ago
AirServer.jamf.recipe	add replace_group and replace_policy	6 days ago
Aircall.jamf.recipe	add replace_group and replace_policy	6 days ago
Alfred2.jamf.recipe	add replace_group and replace_policy	6 days ago
Alfred3.jamf.recipe	add replace_group and replace_policy	6 days ago
AndroidStudio.jamf.recipe	add replace_group and replace_policy	6 days ago
ApacheDirectoryStudio.jamf.recipe	add replace_group and replace_policy	6 days ago
Atom.jamf.recipe	add replace_group and replace_policy	6 days ago
AuroraHDRPro.jamf.recipe	add replace_group and replace_policy	6 days ago
Authy.jamf.recipe	switch to plist recipes for jamf recipes to allow AutoPkgr to work	14 days ago
AutoMounter.jamf.recipe	add replace_group and replace_policy	6 days ago

I've uploaded all these recipes to my repo. So any of you who are currently using the jss recipes in the jss-recipes repo can therefore now start to use either the equivalent pkg-upload jamf recipes, or the full jamf recipes, that are in my repo.

If they don't quite fit your workflow, they should still be a good reference for writing your own jamf recipes.



eth-its / autopkg-mac-recipes-yaml Public

2 branches 0 tags

Graham Pugh (iMac Pro) another path that needed fixing 77992ab 7 hours ago 1,054 commits

.vscode	add skipped option	11 months ago
Download_Recipes	MOE recipes	11 hours ago
Jamf_Account_Recipes	LDAP account recipe	7 days ago
Jamf_MacApp_Recipes	last chance	2 months ago
Jamf_Nodes_Recipes	new template required	10 hours ago
Jamf_Profile_Recipes	fix name	14 days ago
Jamf_Recipes	zsh	10 hours ago
Jamf_Script_Recipes	remove temporary group updates	5 days ago
Jamf_Staging_Recipes	zsh	10 hours ago
Jamf_Uninstall_Recipes	zsh	10 hours ago
Notes	commit of automatically generated yaml recipes	16 months ago
PKG_Recipes	another path that needed fixing	7 hours ago
Profiles_and_Payloads	GraphicConverter recipes	14 days ago

About

AutoPkg recipes for deployment of Mac software via Jamf Pro, in use at ETH Zürich

autopkg jamf-pro autopkg-recipes

Readme
Apache-2.0 license
11 stars
3 watching
3 forks

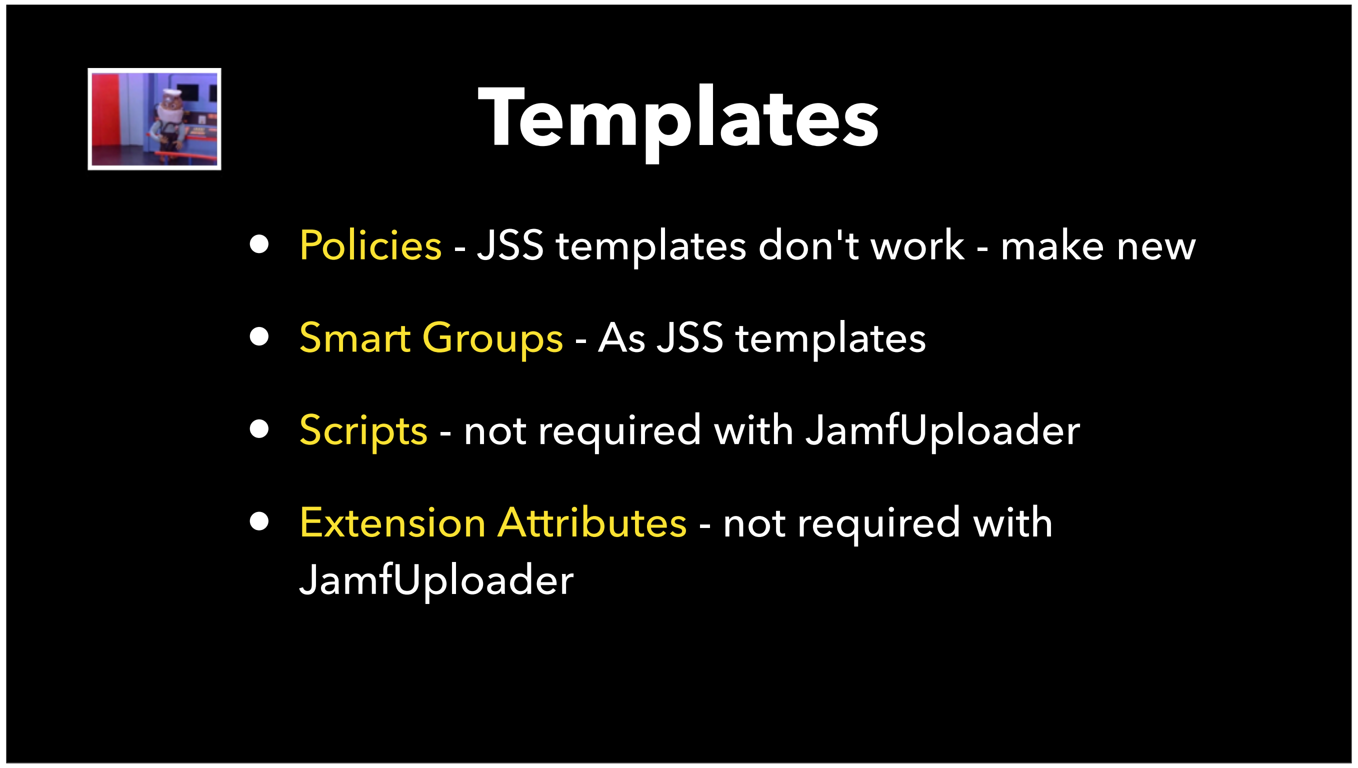
Contributors 2

grahampugh Graham Pugh
katidzehnder

Languages

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

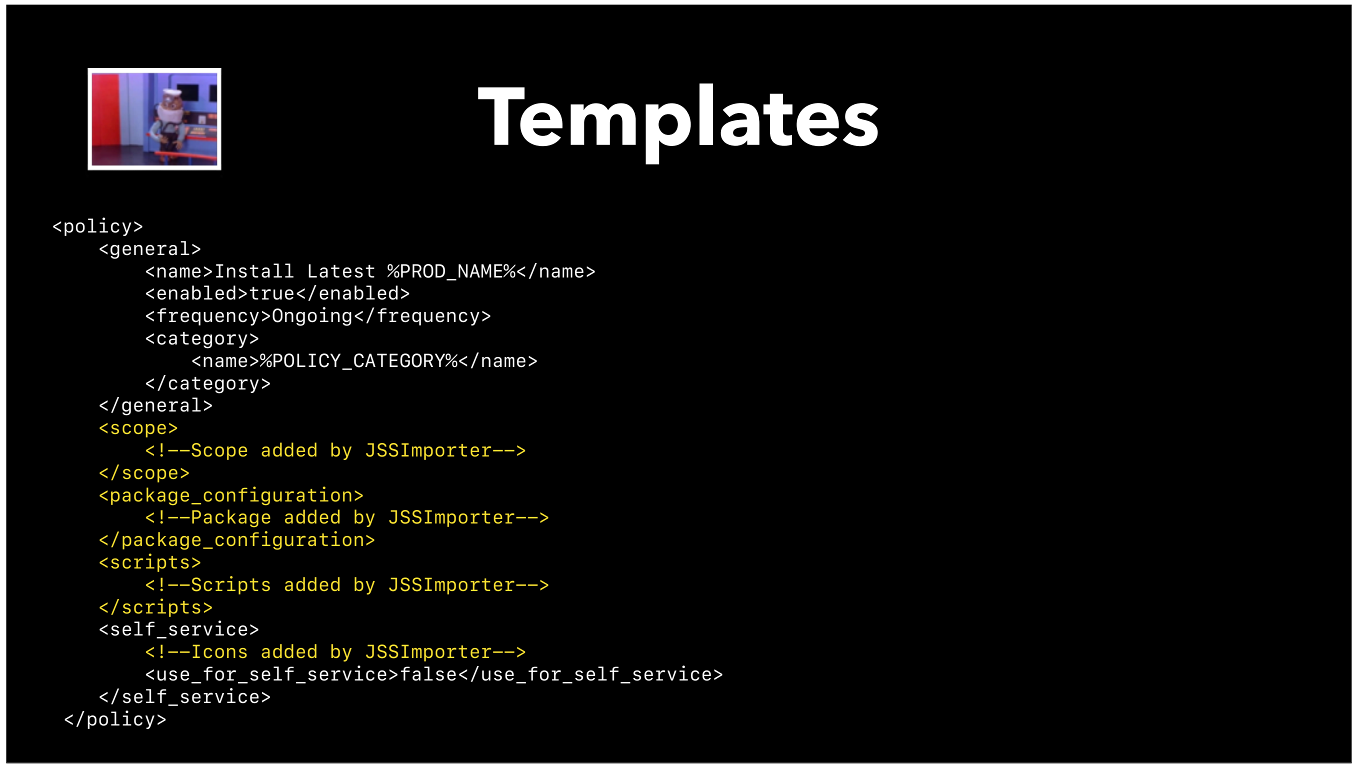
And finally, because my organisation has a specific set of needs that never did fit that JSSImporter standard, we've always maintained our own recipes. Switching to Jamf recipes was an opportunity to make our own recipes available in a public GitHub repo (including my shameful git commit comments!). We have hundreds of recipes openly available for those who want to see more examples of how to write more exotic jamf recipes, in the link at the bottom of the screen. Any secrets needed to run these recipes are simply added to the RecipeOverrides, which we keep in a private repo.



Templates

- Policies - JSS templates don't work - make new
- Smart Groups - As JSS templates
- Scripts - not required with JamfUploader
- Extension Attributes - not required with JamfUploader

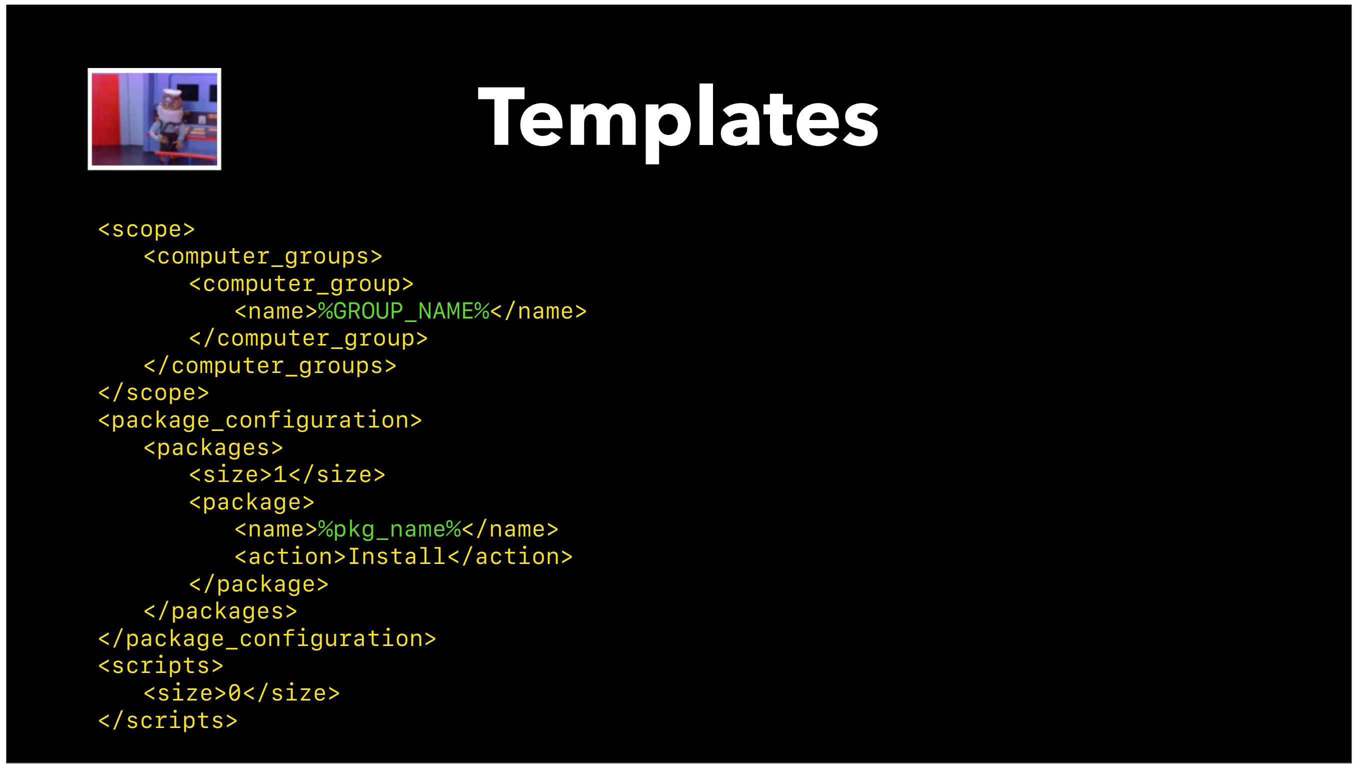
I want to quickly mention templates for Jamf recipes, in the context of converting from JSS recipes.
Smart Group Templates are basically the same, but Policy Templates are not the same.
The really good news, scripts, and the scripts inside extension attributes don't need a template at all with JamfUploader. And, you can use variable substitution in scripts and EAs.



Templates

```
<policy>
    <general>
        <name>Install Latest %PROD_NAME%</name>
        <enabled>true</enabled>
        <frequency>Ongoing</frequency>
        <category>
            <name>%POLICY_CATEGORY%</name>
        </category>
    </general>
    <scope>
        <!--Scope added by JSSImporter-->
    </scope>
    <package_configuration>
        <!--Package added by JSSImporter-->
    </package_configuration>
    <scripts>
        <!--Scripts added by JSSImporter-->
    </scripts>
    <self_service>
        <!--Icons added by JSSImporter-->
        <use_for_self_service>false</use_for_self_service>
    </self_service>
 </policy>
```

Let's quickly look at what is different between the policy template of a JSS recipe and a Jamf recipe.
This is a template that works with JSSImporter. The scope, package, script and icons are abstracted out of the template and handled instead directly by JSSImporter.
It was always possible to set these manually in the template, but I rarely saw this done except to scope to all computers, so most of you will have templates like this.
This template does not work with JamfPolicyUploader - you have to provide these sections in their normal form.

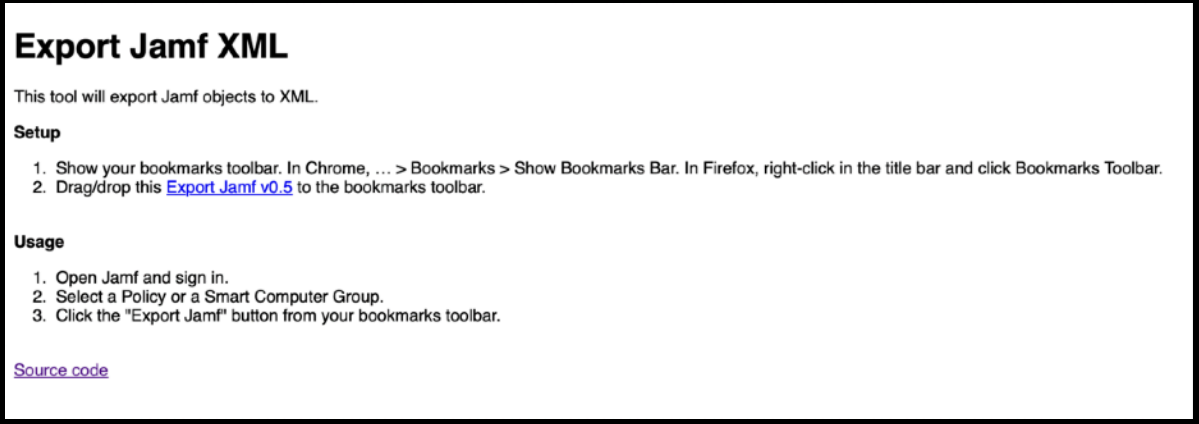


Templates

```
<scope>
   <computer_groups>
      <computer_group>
         <name>%GROUP_NAME%</name>
      </computer_group>
   </computer_groups>
</scope>
<package_configuration>
   <packages>
      <size>1</size>
      <package>
         <name>%pkg_name%</name>
         <action>Install</action>
      </package>
   </packages>
</package_configuration>
<scripts>
   <size>0</size>
</scripts>
```

So that section with the scope, packages and scripts needs to look something like this. This is exactly what it looks like if you export the policy XML from Jamf Pro's api documentation on your instance, except of course you can add variables so that you can re-use the template.
Anthony talked about how to do this in our JNUC 2021 presentation.

Templates



Export Jamf XML

This tool will export Jamf objects to XML.

Setup

1. Show your bookmarks toolbar. In Chrome, … > Bookmarks > Show Bookmarks Bar. In Firefox, right-click in the title bar and click Bookmarks Toolbar.
2. Drag/drop this Export Jamf v0.5 to the bookmarks toolbar.

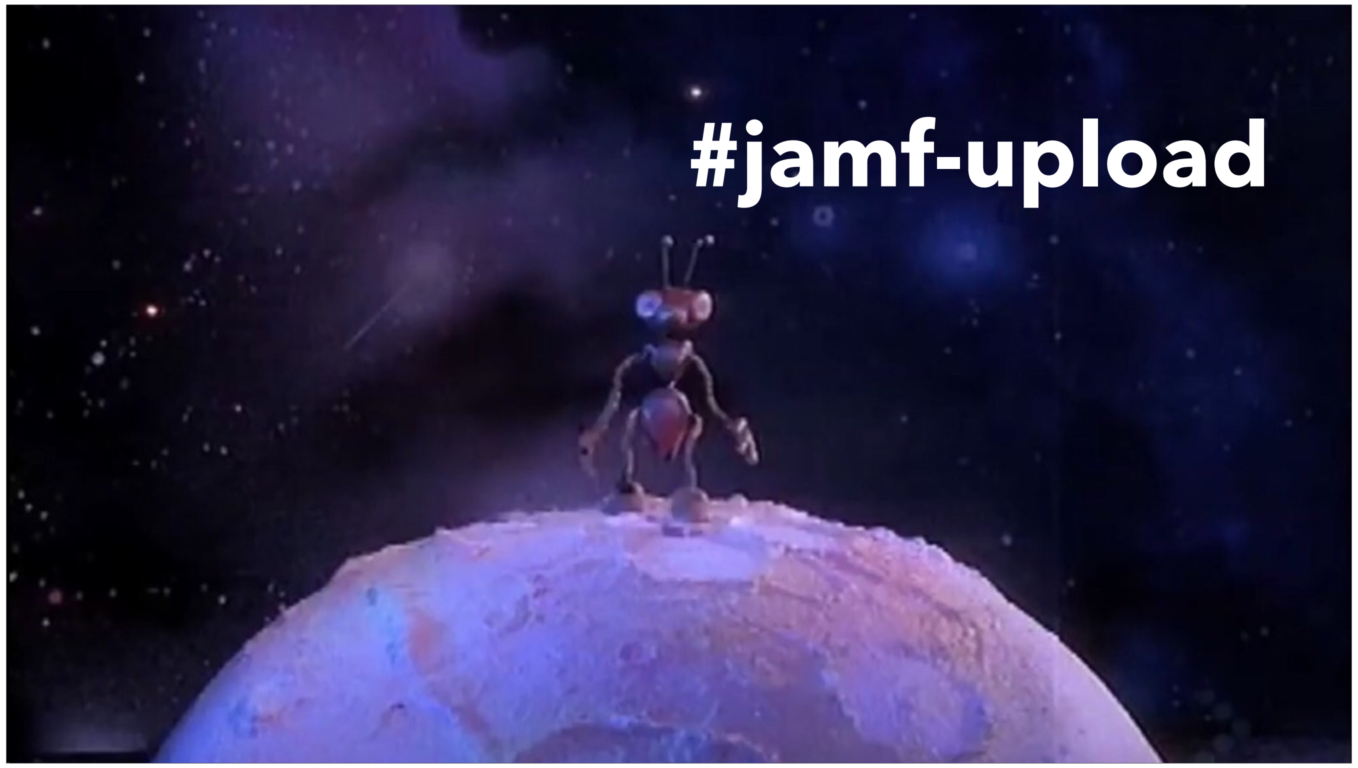
Usage

1. Open Jamf and sign in.
2. Select a Policy or a Smart Computer Group.
3. Click the "Export Jamf" button from your bookmarks toolbar.

Source code

gabrielsroka.github.io/ExportJamfXML.html

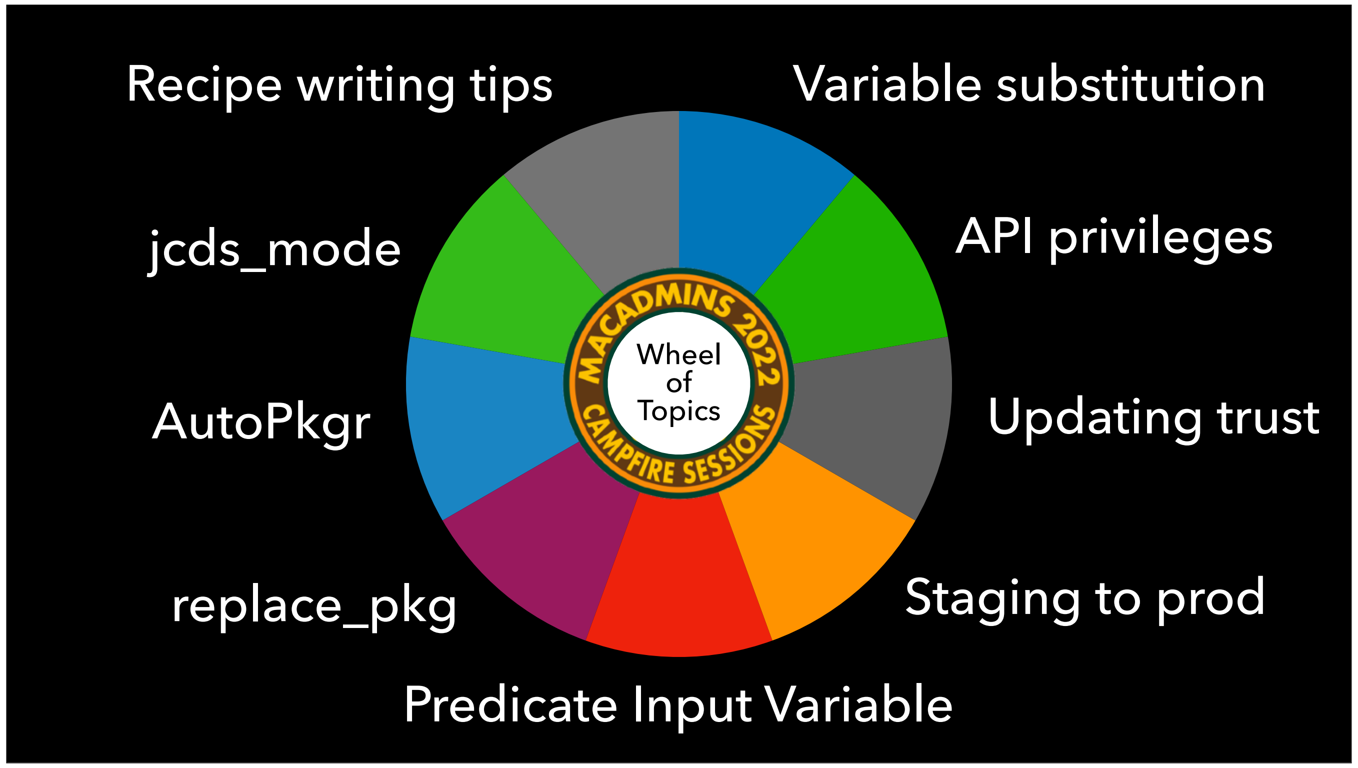
I'd also like to give a shout out to a cool web browser bookmarklet called Export Jamf XML by Gabriel Sroka and Bilal Habib, which is by far the easiest way to grab the XML from a policy or smart group.



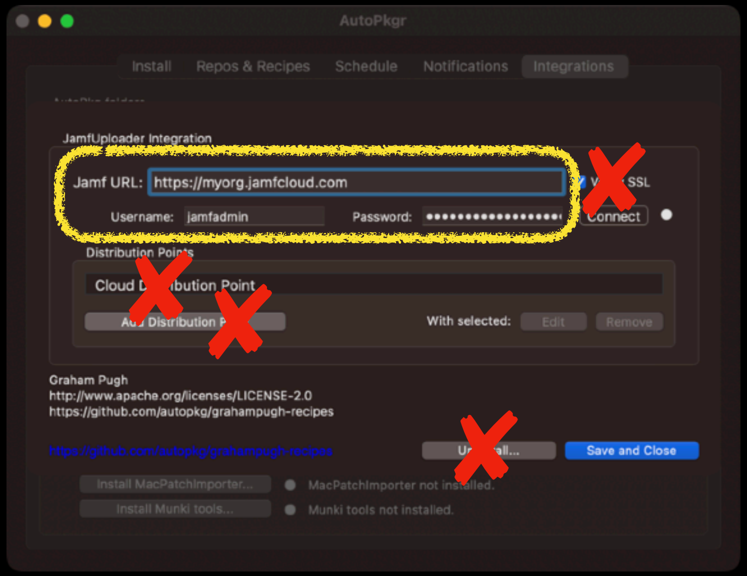
Finally, you are not alone! We are here to help you make the transition to JamfUploader. The best place to come for help is the #jamf-upload channel in the MacAdmins Slack.
And now, back to Anthony.

Over to you…

[Anthony] Over the past year, more and more people have joined the #jamf-upload channel in the MacAdmins Slack to ask about getting started with JamfUploader. And you have had lots and lots of questions. We've tried to answer some in the slides we've already presented, but we want to turn the rest of our presentation time over to you to cover any other topics you are most interested in. So we present to you the…



Wheel… of… Topics!
We probably won't get to all of these, but any slides we have for the topics shown will be included in the slide deck we share at the end. Put your votes in the Q&A or in the #psumac Slack channel. Of course, you are welcome to ask any question you would like via Q&A.



JamfUploader Integration

Jamf URL: https://myorg.jamfcloud.com ✓ V SSL

Username: jamfadmin Password: ••••••••••••••••••

Distribution Points

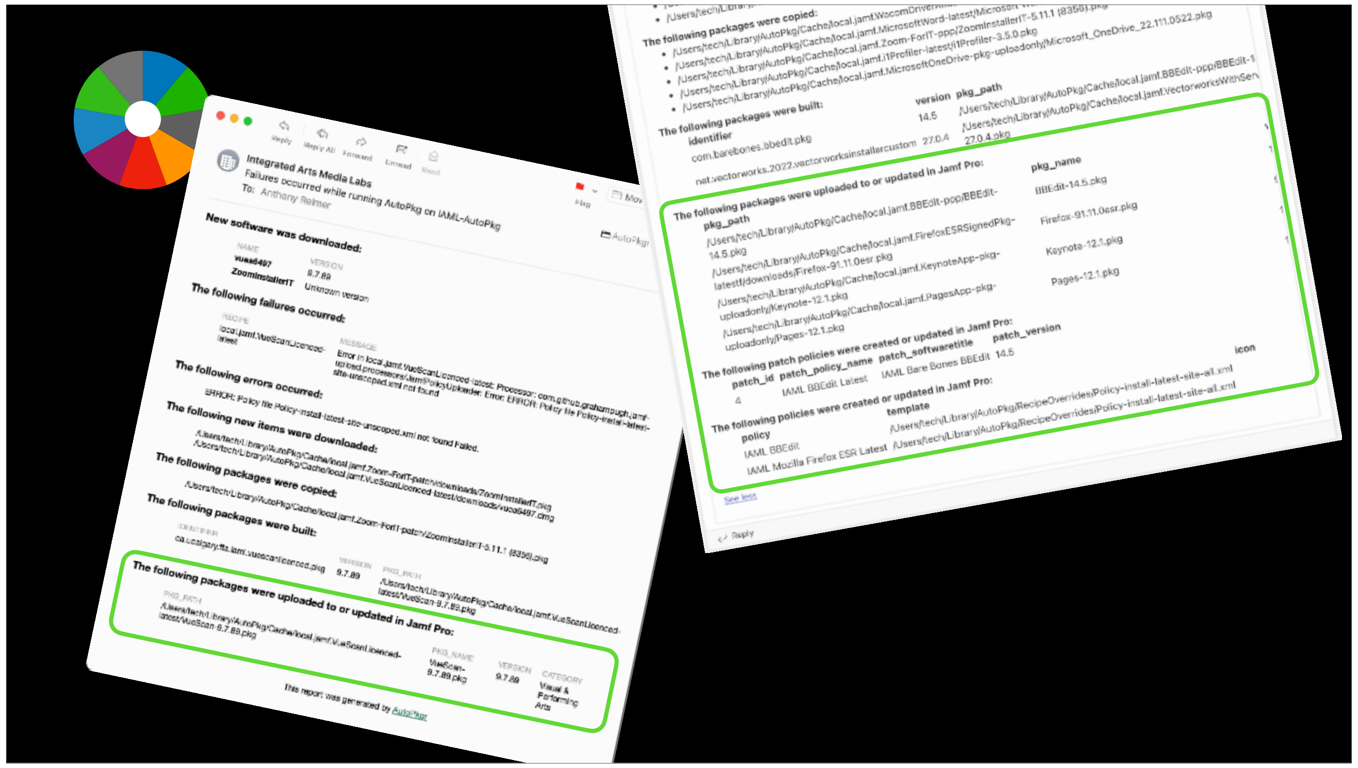
Graham Pugh
http://www.apache.org/licenses/LICENSE-2.0
https://github.com/autopkg/grahampugh-recipes

https://github.com/autopkg/grahampugh-recipes

● MacPatchImporter not installed.
● Munki tools not installed.

github.com/grahampugh/jamf-upload/wiki/JamfUploader-AutoPkg-Processors

[Graham:] AutoPkgr now supports JamfUploader to some extent. Shawn Honsberger has been working on this for the past few months. However, there are a few things still to be fixed. Firstly, yaml-based recipes are not found in AutoPkgr. (You may want to force your overrides to be in Plist format if that is important to you.) Secondly, the configuration options for JamfUploader are not yet correct. These bits are OK, but these are wrong as they are based on the way JSSImporter was configured which is not appropriate here. Until these are fixed, we recommend that you use simple defaults commands to configure the required settings. Our wiki has you covered. [Anthony >]

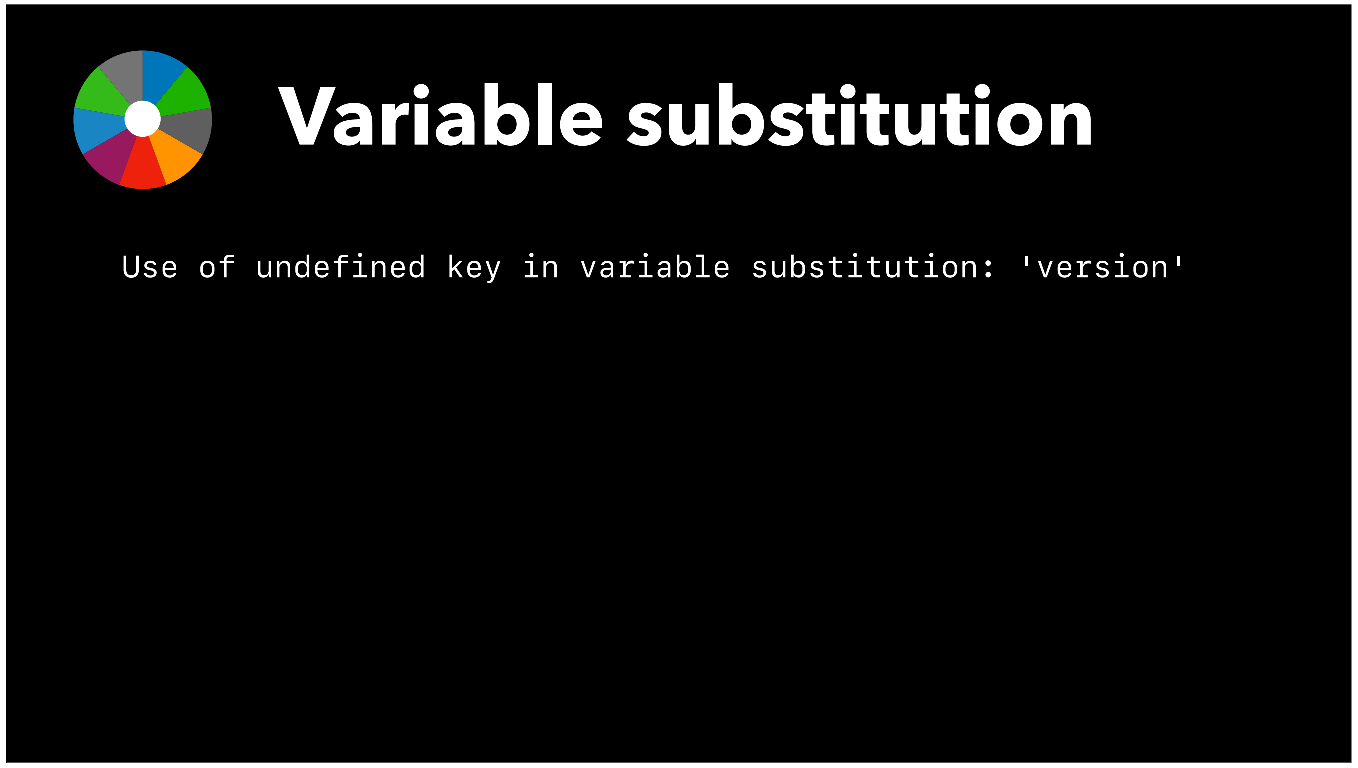


Integrated Arts Media Labs
Failures occurred while running AutoPkg on IAML-AutoPkg
To: Anthony Reimer

```
New software was downloaded:
  NAME            VERSION
  vuea6497        9.7.89
  ZoomInstallerIT Unknown version
The following failures occurred:
  RECIPE                           MESSAGE
  local.jamf.VueScanLicenced-      Error in local.jamf.VueScanLicenced-latest: Processor: com.github.grahampugh.jamf-
  latest                           upload.processors/JamfPolicyUploader: Error: ERROR: Policy file Policy-install-latest-
                                   site-unscoped.xml not found
The following errors occurred:
  ERROR: Policy file Policy-install-latest-site-unscoped.xml not found Failed.
The following new items were downloaded:
  /Users/tech/Library/AutoPkg/Cache/local.jamf.Zoom-ForIT-patch/downloads/ZoomInstallerIT.pkg
  /Users/tech/Library/AutoPkg/Cache/local.jamf.VueScanLicenced-latest/downloads/vuea6497.dmg
The following packages were copied:
  /Users/tech/Library/AutoPkg/Cache/local.jamf.Zoom-ForIT-patch/ZoomInstallerIT-5.11.1 (8356).pkg
The following packages were built:
  IDENTIFIER                           VERSION  PKG_PATH
  ca.ucalgary.ffa.iaml.vuescanlicenced.pkg  9.7.89  /Users/tech/Library/AutoPkg/Cache/local.jamf.VueScanLicenced-
                                                    latest/VueScan-9.7.89.pkg
The following packages were uploaded to or updated in Jamf Pro:
  PKG_PATH                                                   PKG_NAME   VERSION  CATEGORY
  /Users/tech/Library/AutoPkg/Cache/local.jamf.VueScanLicenced-  VueScan-   9.7.89   Visual &
  latest/VueScan-9.7.89.pkg                                  9.7.89.pkg          Performing
                                                                                 Arts
This report was generated by AutoPkgr
```

```
  • /Users/tech/Library/
The following packages were copied:
  • /Users/tech/Library/AutoPkg/Cache/local.jamf.WacomDriver
  • /Users/tech/Library/AutoPkg/Cache/local.jamf.MicrosoftWord-latest/Microsoft
  • /Users/tech/Library/AutoPkg/Cache/local.jamf.Zoom-ForIT-ppp/ZoomInstallerIT-5.11.1 (8356).pkg
  • /Users/tech/Library/AutoPkg/Cache/local.jamf.i1Profiler-latest/i1Profiler-3.5.0.pkg
  • /Users/tech/Library/AutoPkg/Cache/local.jamf.MicrosoftOneDrive-pkg-uploadonly/Microsoft_OneDrive_22.111.0522.pkg
The following packages were built:
    identifier                              version  pkg_path
    com.barebones.bbedit.pkg                14.5     /Users/tech/Library/AutoPkg/Cache/local.jamf.BBEdit-ppp/BBEdit-1
    net.vectorworks.2022.vectorworksinstallercustom  27.0.4  /Users/tech/Library/AutoPkg/Cache/local.jamf.VectorworksWithServ
                                                     27.0.4.pkg
The following packages were uploaded to or updated in Jamf Pro:
    pkg_path                                                          pkg_name
    /Users/tech/Library/AutoPkg/Cache/local.jamf.BBEdit-ppp/BBEdit-   BBEdit-14.5.pkg
    14.5.pkg
    /Users/tech/Library/AutoPkg/Cache/local.jamf.FirefoxESRSignedPkg- Firefox-91.11.0esr.pkg
    latest/downloads/Firefox-91.11.0esr.pkg
    /Users/tech/Library/AutoPkg/Cache/local.jamf.KeynoteApp-pkg-      Keynote-12.1.pkg
    uploadonly/Keynote-12.1.pkg
    /Users/tech/Library/AutoPkg/Cache/local.jamf.PagesApp-pkg-        Pages-12.1.pkg
    uploadonly/Pages-12.1.pkg
The following patch policies were created or updated in Jamf Pro:
    patch_id  patch_policy_name   patch_softwaretitle   patch_version
    4         IAML BBEdit Latest  IAML Bare Bones BBEdit  14.5
The following policies were created or updated in Jamf Pro:
    policy                          template                                                                   icon
    IAML BBEdit                     /Users/tech/Library/AutoPkg/RecipeOverrides/Policy-install-latest-site-all.xml
    IAML Mozilla Firefox ESR Latest /Users/tech/Library/AutoPkg/RecipeOverrides/Policy-install-latest-site-all.xml
See less
Reply
```

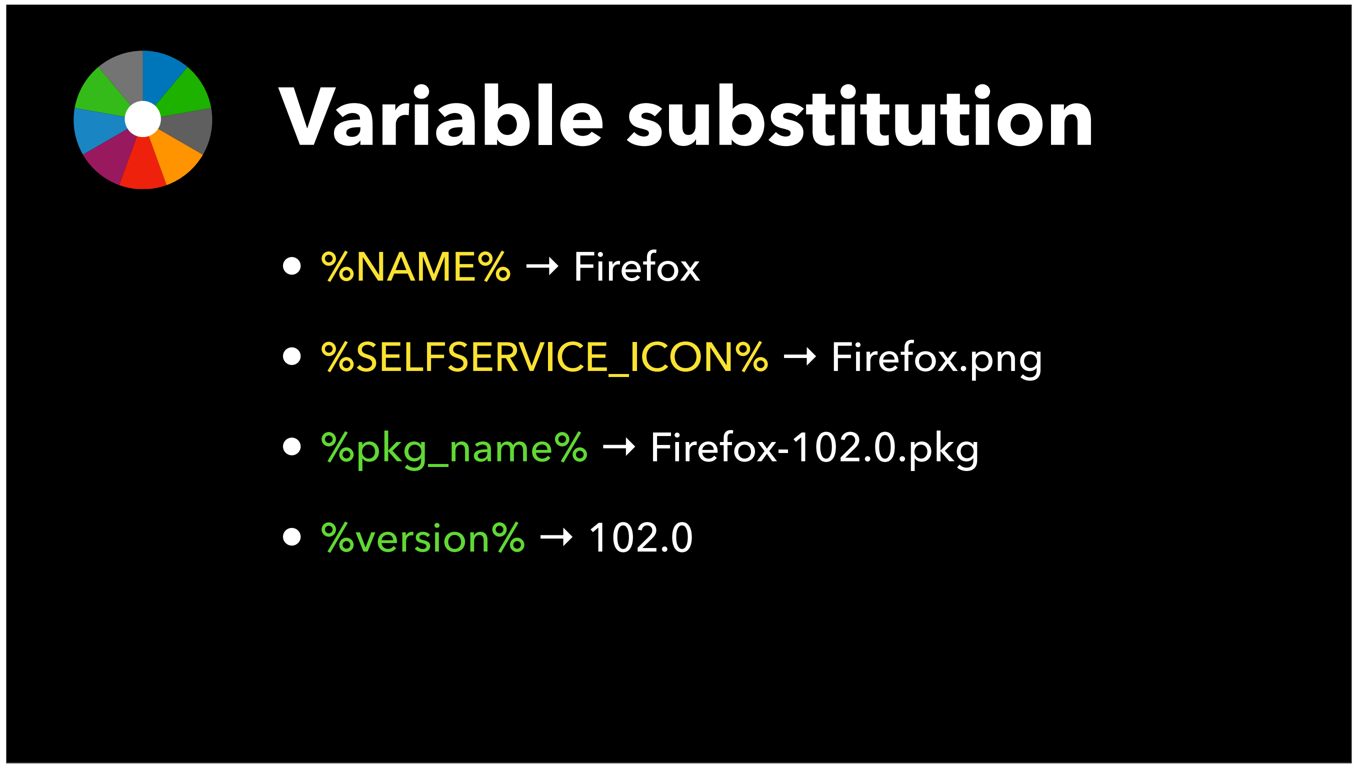
[Anthony:] A nice addition in the latest version is that the reports that AutoPkgr can send via email or Slack or Teams now correctly include the summary information from the current JamfUploader processors. Previously, you could only get that summary information by running the recipes on the command line or using the Teams or Slack processors that Graham mentioned off the top.



Variable substitution

```
Use of undefined key in variable substitution: 'version'
```

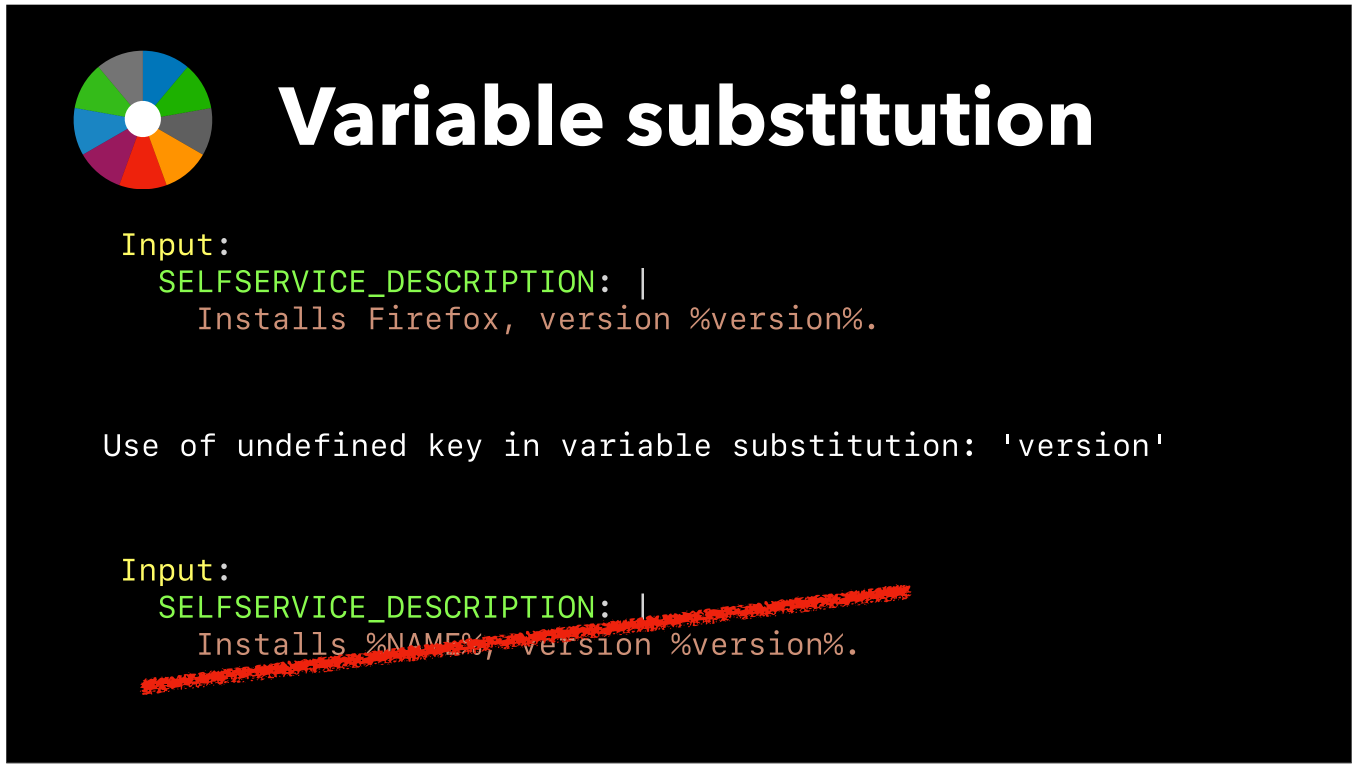
[Graham:] In certain circumstances, a jamf recipe might output the warning "use of undefined key in variable substitution", referring to the version key. This is ok and does not result in an error, but perhaps requires explanation.



Variable substitution

- %NAME% → Firefox
- %SELFSERVICE_ICON% → Firefox.png
- %pkg_name% → Firefox-102.0.pkg
- %version% → 102.0

AutoPkg substitutes values of variables that are indicated by percent signs around a variable name. Variables in the Input keys of all recipes in a chain are substituted at the beginning of the run, and variables in processor steps are substituted at the time of that processor step. This allows the use of values that were outputted in previous processor steps to be used in subsequent steps.



Variable substitution

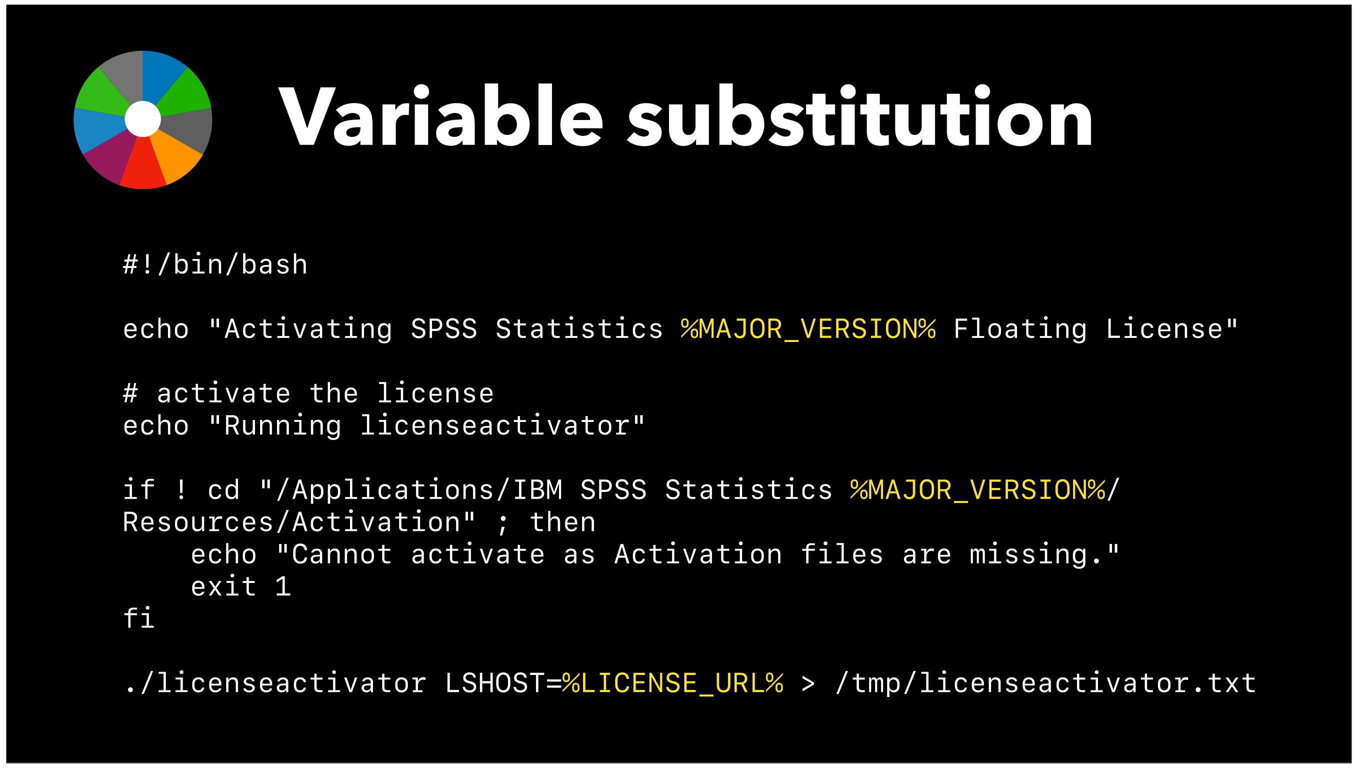
```
Input:
  SELFSERVICE_DESCRIPTION: |
    Installs Firefox, version %version%.
```

```
Use of undefined key in variable substitution: 'version'
```

```
Input:
  SELFSERVICE_DESCRIPTION: |
    ~~Installs %NAME%, version %version%.~~
```

In a normal AutoPkg recipe, that means you cannot include a key like %version% in an input key, because the version has not been generated at the beginning of the recipe.
However, I have included an extra step of variable substitution into the JamfUploader processors so that we can include variables that were generated in the parent recipes in Input steps.

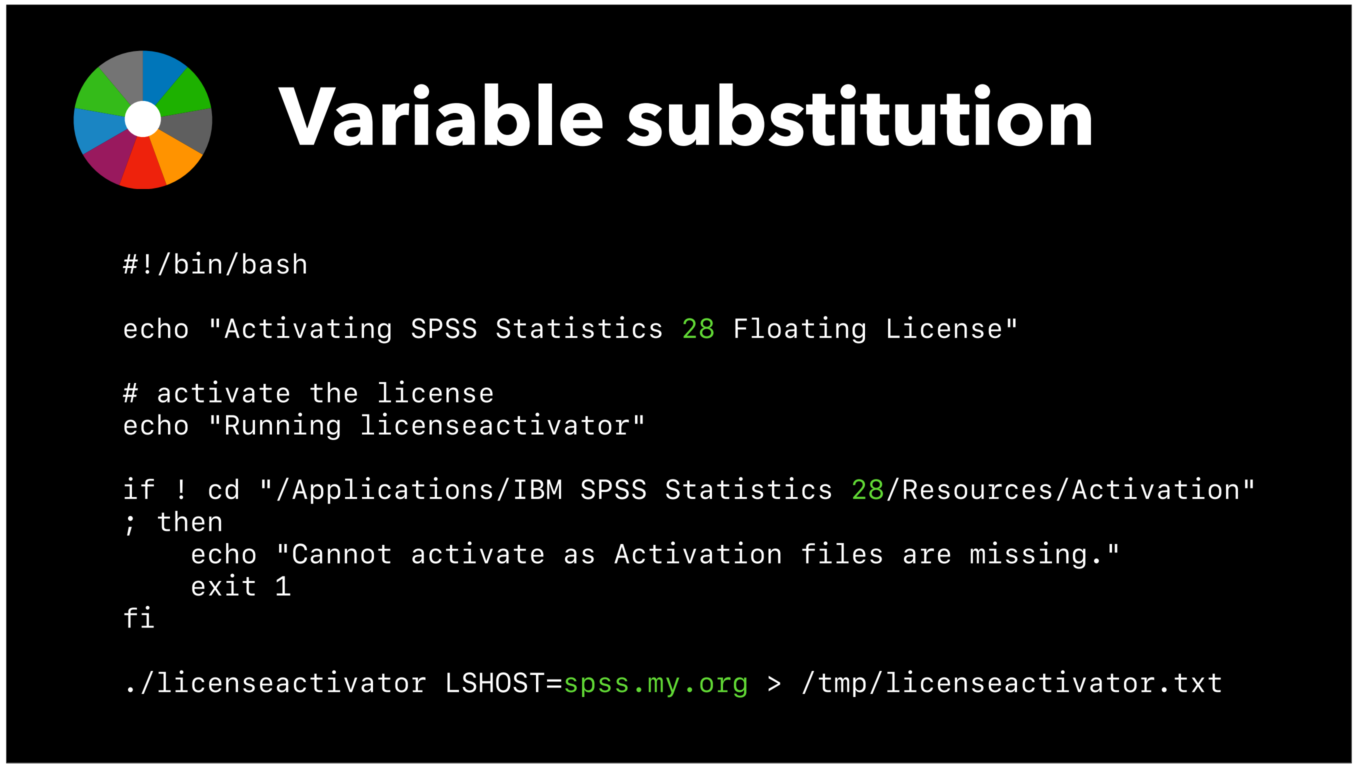
- Unfortunately this does not prevent AutoPkg giving the warnings at the start of the run.
- And just to note a current bug with this - you cannot have both a key from the Inputs and a key generated later in the same Input key, such as NAME and version here.
- This won't work and the recipe will fail. The workaround is to hard-code the value of NAME into the key.



Variable substitution

```
#!/bin/bash

echo "Activating SPSS Statistics %MAJOR_VERSION% Floating License"

# activate the license
echo "Running licenseactivator"

if ! cd "/Applications/IBM SPSS Statistics %MAJOR_VERSION%/
Resources/Activation" ; then
    echo "Cannot activate as Activation files are missing."
    exit 1
fi

./licenseactivator LSHOST=%LICENSE_URL% > /tmp/licenseactivator.txt
```

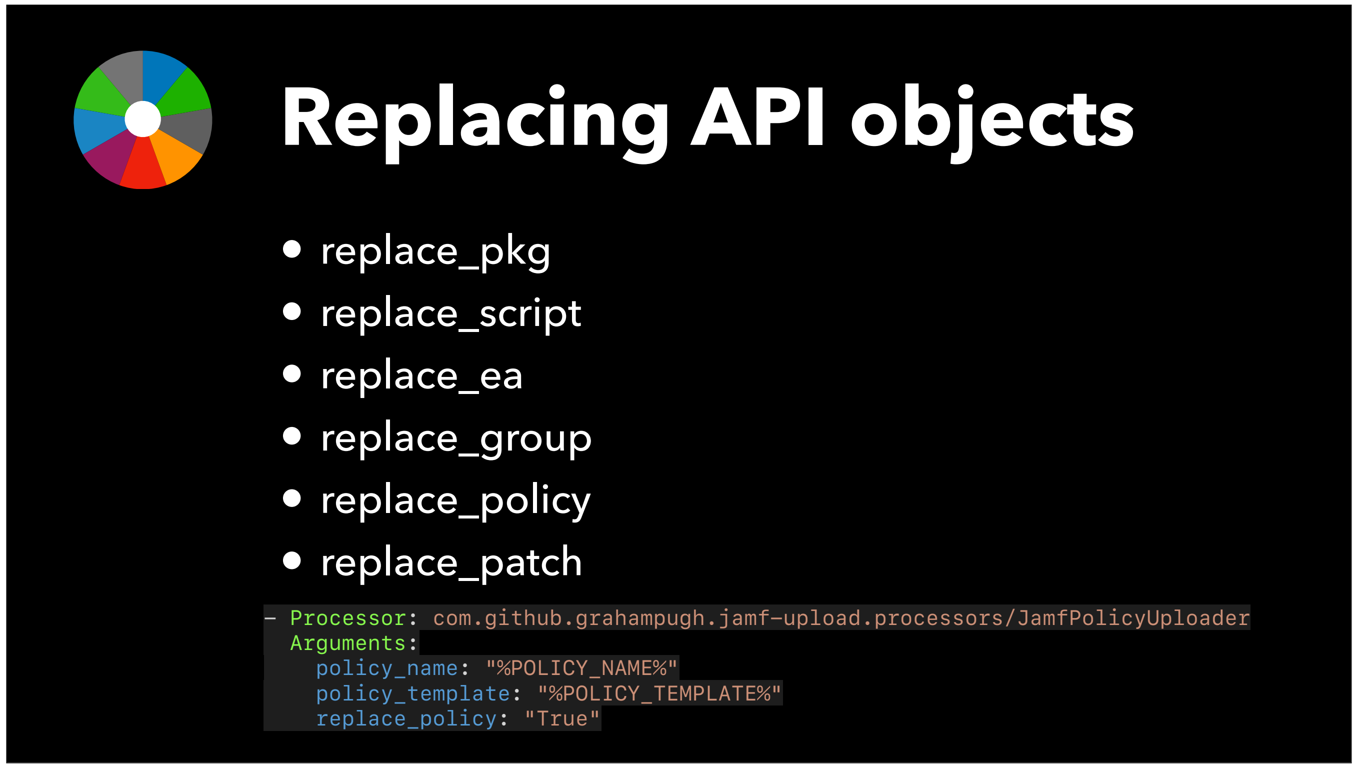
An extra cool feature of JamfUploader processors is that you can add variables from Input keys directly into templates and scripts, including extension attribute scripts. In this script here we are adding the URL of an SPSS license server to a file on the client and substituting the major version of SPSS into the code, so that we can use the same script over multiple releases.



Variable substitution

```
#!/bin/bash

echo "Activating SPSS Statistics 28 Floating License"

# activate the license
echo "Running licenseactivator"

if ! cd "/Applications/IBM SPSS Statistics 28/Resources/Activation"
; then
    echo "Cannot activate as Activation files are missing."
    exit 1
fi

./licenseactivator LSHOST=spss.my.org > /tmp/licenseactivator.txt
```

The variables are substituted during the JamfScriptUploader process, so that the values appear when viewing the uploaded script in the Jamf admin console.



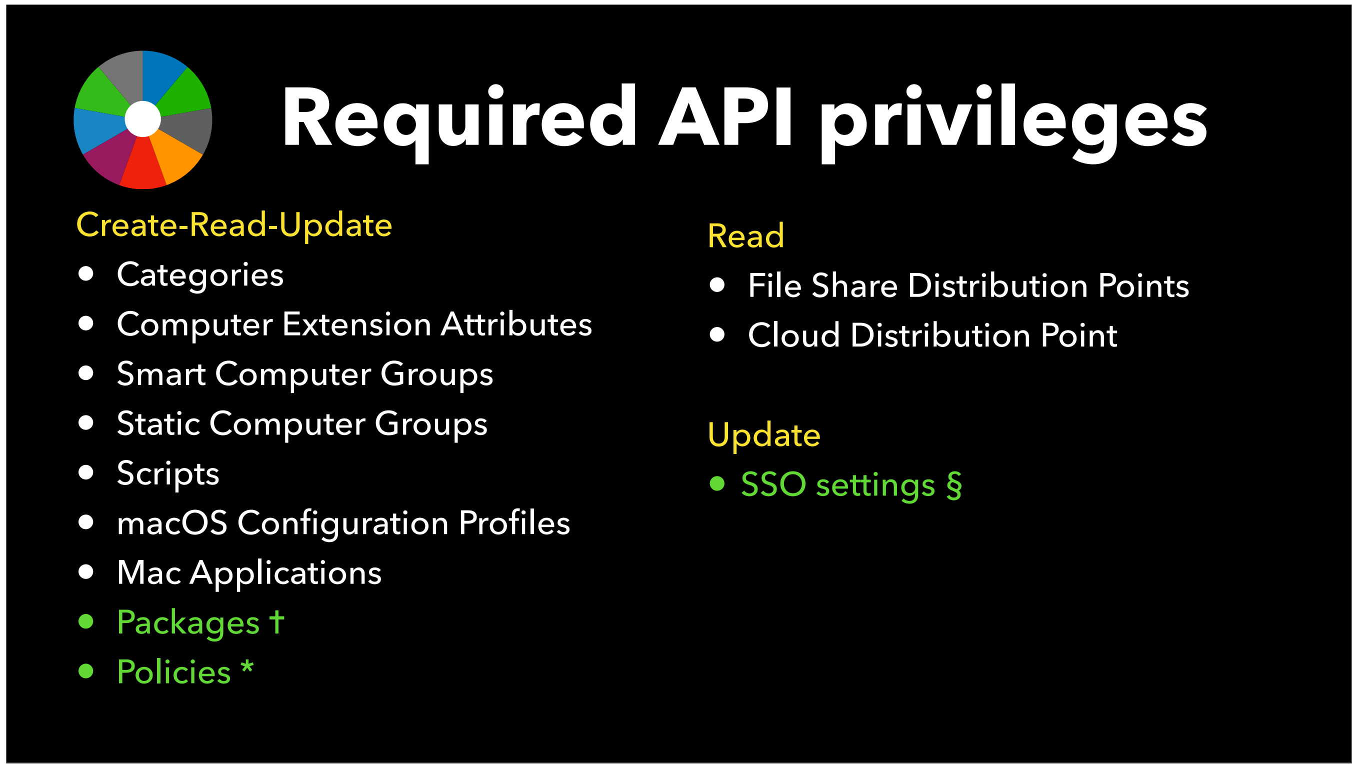
Replacing API objects

- replace_pkg
- replace_script
- replace_ea
- replace_group
- replace_policy
- replace_patch

```
- Processor: com.github.grahampugh.jamf-upload.processors/JamfPolicyUploader
  Arguments:
    policy_name: "%POLICY_NAME%"
    policy_template: "%POLICY_TEMPLATE%"
    replace_policy: "True"
```

[Graham:] JamfUploader processors do not replace existing API objects by default. This is for workflows where you want to use your AutoPkg automation to create any missing object, but do not want to overwrite the object if it already exists. That allows you to provide items to administrators who can then safely edit them afterwards without their changes getting overwritten, for example scoping specific computers to a computer group.
To enforce something to get overwritten, use the "replace" key for that object, e.g. "replace_pkg", "replace_script", "replace_group" etc.

[Anthony:] You saw that in one of the recipe examples I showed you. Based on my methodology, I want to replace the policy each time there is a new package, not just create a new policy, so I set replace_pkg to 'True'. Note that this is the string 'True', not a boolean value. [Mention replace_pkg error message when pkg already exists if time]



Required API privileges

Create-Read-Update

- Categories
- Computer Extension Attributes
- Smart Computer Groups
- Static Computer Groups
- Scripts
- macOS Configuration Profiles
- Mac Applications
- Packages †
- Policies *

Read

- File Share Distribution Points
- Cloud Distribution Point

Update

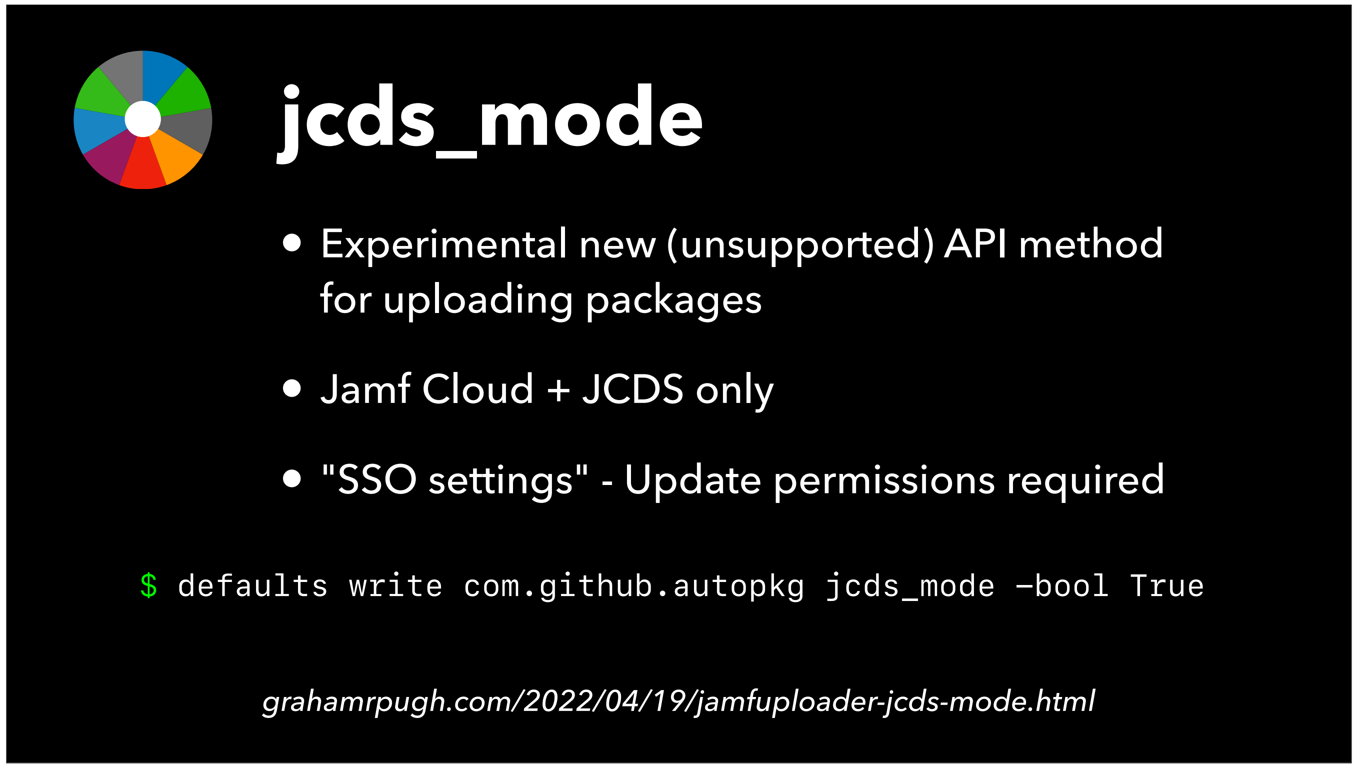
- SSO settings §

[Graham:] To make changes to your Jamf Pro server via the API, you should use an account with the least required privileges for your workflow. Do not use your regular login account or your management account!

 The privileges required depend on which processors you are going to use. In general, you will need Create, Read and Update rights for those objects you are going to upload. For packages you can remove update rights if you are OK with updating the package only without any additional metadata such as category, file hash, notes, etc. For policies, since we have a JamfPolicyDeleter processor, if you wish to make use of this, you will of course need to add Delete privileges for policies.

 You also need to add read privileges for either Fileshare distribution points or cloud distribution point, depending on whether you are using self-hosted or cloud Jamf Pro.

 And just to note that if you are using the jcds_mode, you also need to add Update privileges for SSO settings.



jcds_mode

- Experimental new (unsupported) API method for uploading packages
- Jamf Cloud + JCDS only
- "SSO settings" - Update permissions required

```
$ defaults write com.github.autopkg jcds_mode -bool True
```

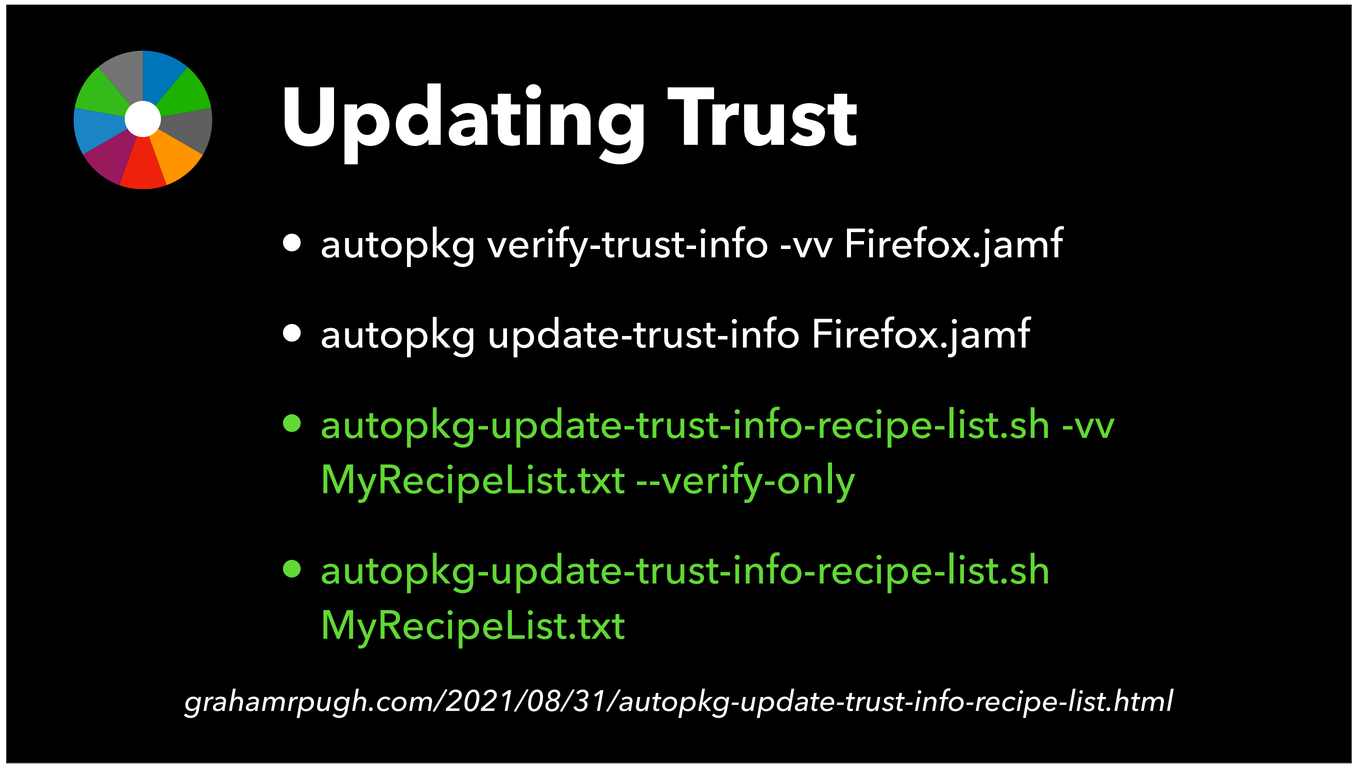
grahamrpugh.com/2022/04/19/jamfuploader-jcds-mode.html

jcds_mode is something I added recently to JamfPackageUploader to try and address reliability issues with uploading packages to Jamf Cloud. There is still no official endpoint for uploading packages, but jcds_mode is a new method which effectively is a reverse-engineering of the GUI upload process.

It only works with Jamf Cloud Distribution Points - not for self-hosted or external AWS cloud distribution points.

Jamf Cloud admins have been reporting higher success rates with uploading packages when using jcds_mode.

To enable it, just set the jcds_mode key to true in your AutoPkg prefs.

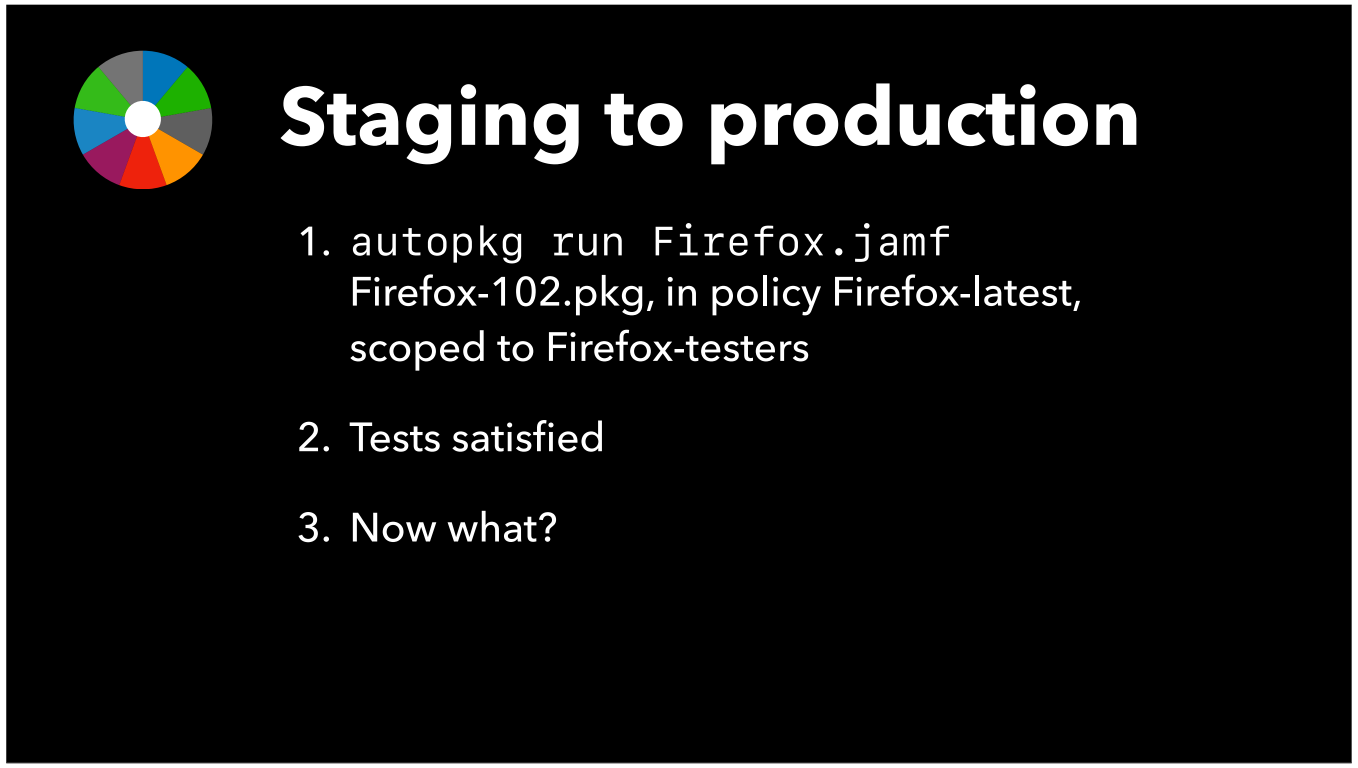


Updating Trust

- autopkg verify-trust-info -vv Firefox.jamf
- autopkg update-trust-info Firefox.jamf
- autopkg-update-trust-info-recipe-list.sh -vv MyRecipeList.txt --verify-only
- autopkg-update-trust-info-recipe-list.sh MyRecipeList.txt

grahamrpugh.com/2021/08/31/autopkg-update-trust-info-recipe-list.html

JamfUploader processors are regular shared processors. When changes are made to shared processors, the recipe trust is broken, and you need to update the trust before you can run the recipe.
Running autopkg with the verify-trust-info verb and level 2 verbosity will show you the changes in whatever element of that recipe has changed.
But if you have tens or hundreds of jamf recipes, updating the trust of each recipe can get tedious.
The verify-trust-info and update-trust-info verbs of autopkg do not have support for recipe lists, so I created a script called autopkg-update-trust-info-recipe-list which adds that functionality.
You can verify all recipes in a recipe list with verbosity set, to see the changes directly in the output, and once happy that all the changes were related to the update of the JamfUploader and no other differences, run the script again to update all the recipe trusts.



Staging to production

1. `autopkg run Firefox.jamf`
   Firefox-102.pkg, in policy Firefox-latest, scoped to Firefox-testers
2. Tests satisfied
3. Now what?

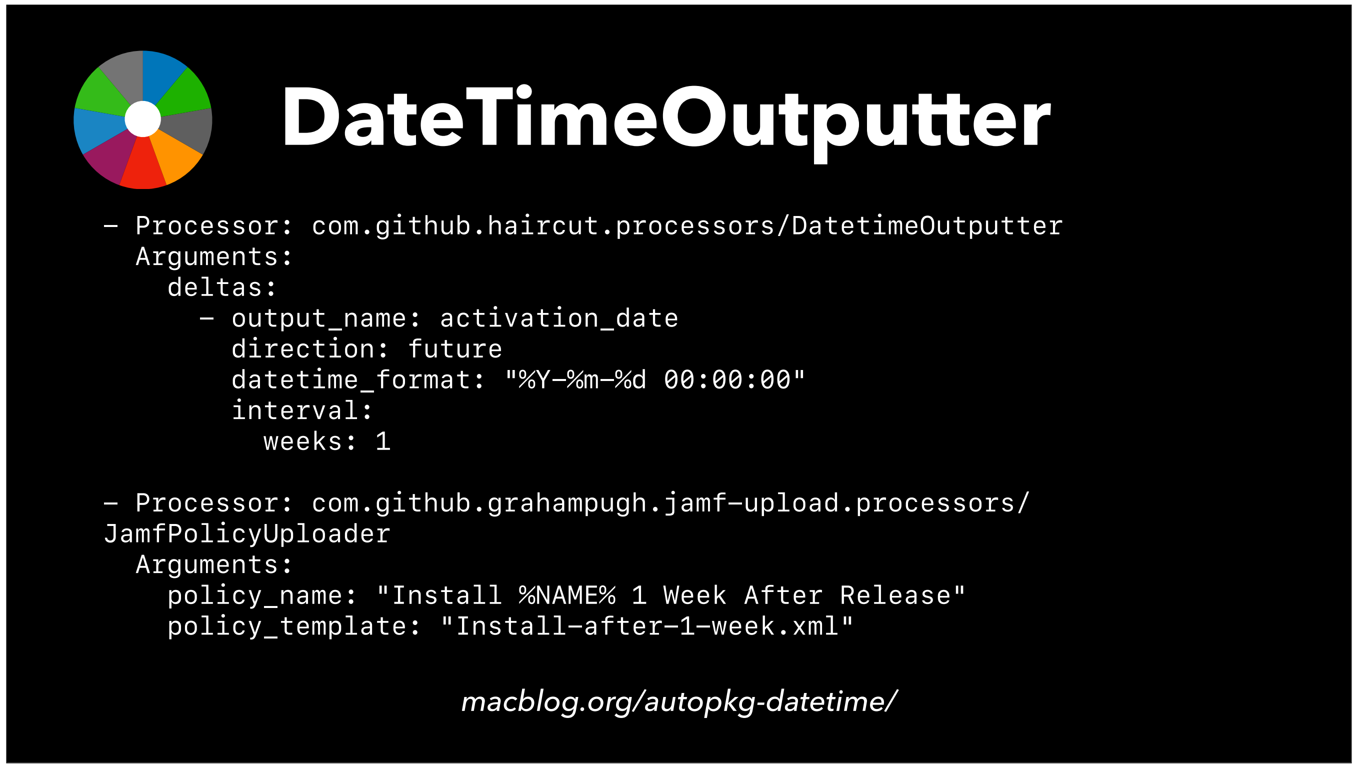
How do you automate staging a package to production?
Normally this involves switching out a package in an existing policy or policies. A normal AutoPkg recipe isn't ideal for this because it will always check for the latest package when it runs, which means you might get a newer package than you wish to stage to production.

pkgctl

```
Analyzing Jamf data
█
```

github.com/univ-of-utah-marriott-library-apple/jctl/wiki/pkgctl

One solution is to use a different API tool to switch out the package in your production policies, such as pkgctl, part of the jctl tools from the good folks at the University of Utah Marriott Library. Depending on your workflow, you may also need to change the criteria in relevant smart groups.

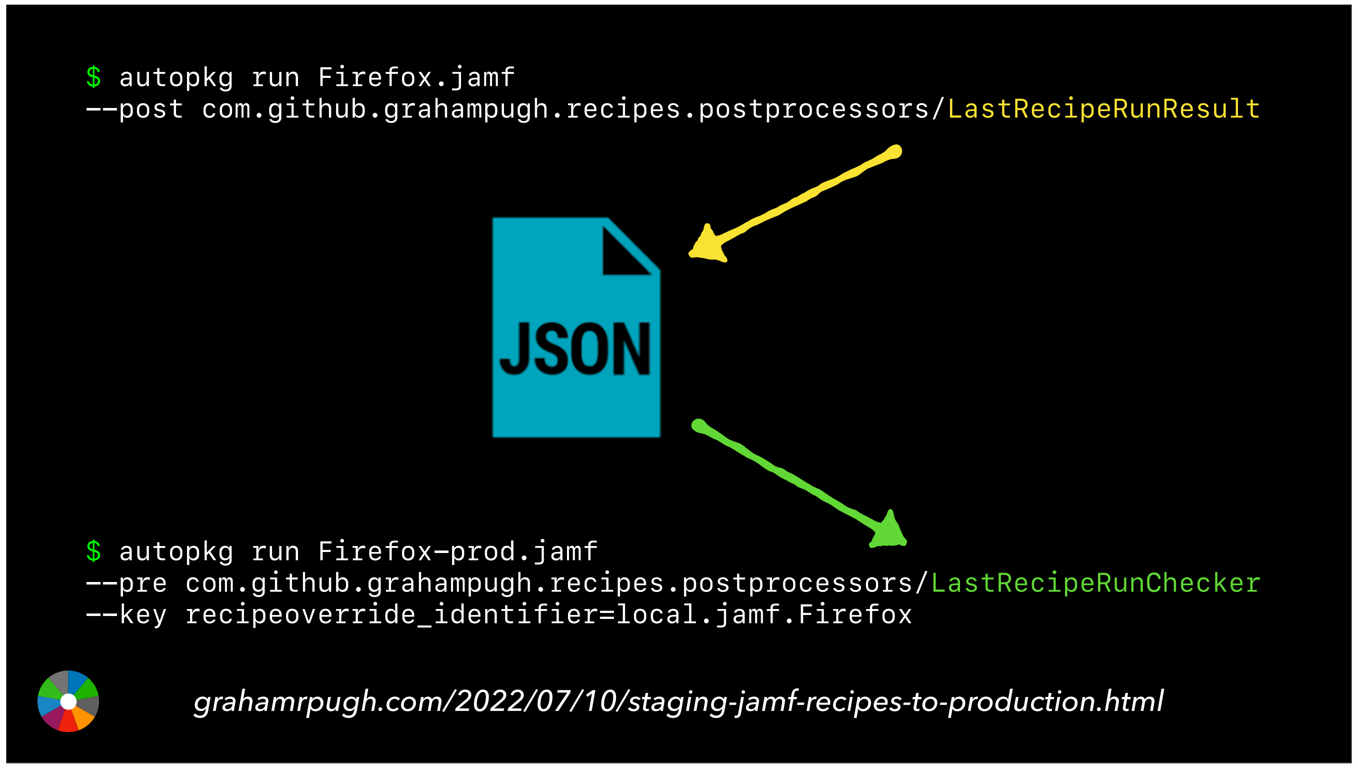


DateTimeOutputter

```
- Processor: com.github.haircut.processors/DatetimeOutputter
  Arguments:
    deltas:
      - output_name: activation_date
        direction: future
        datetime_format: "%Y-%m-%d 00:00:00"
        interval:
          weeks: 1

- Processor: com.github.grahampugh.jamf-upload.processors/
JamfPolicyUploader
  Arguments:
    policy_name: "Install %NAME% 1 Week After Release"
    policy_template: "Install-after-1-week.xml"
```

macblog.org/autopkg-datetime/

If you intend to automatically release new packages after a delay of some time, another option is to create the "production" policy at the same time as the testing policy, but add a future activation date. Matthew Warren's DateTimeOutputter processor can be used to help with this workflow - more information on his blog.

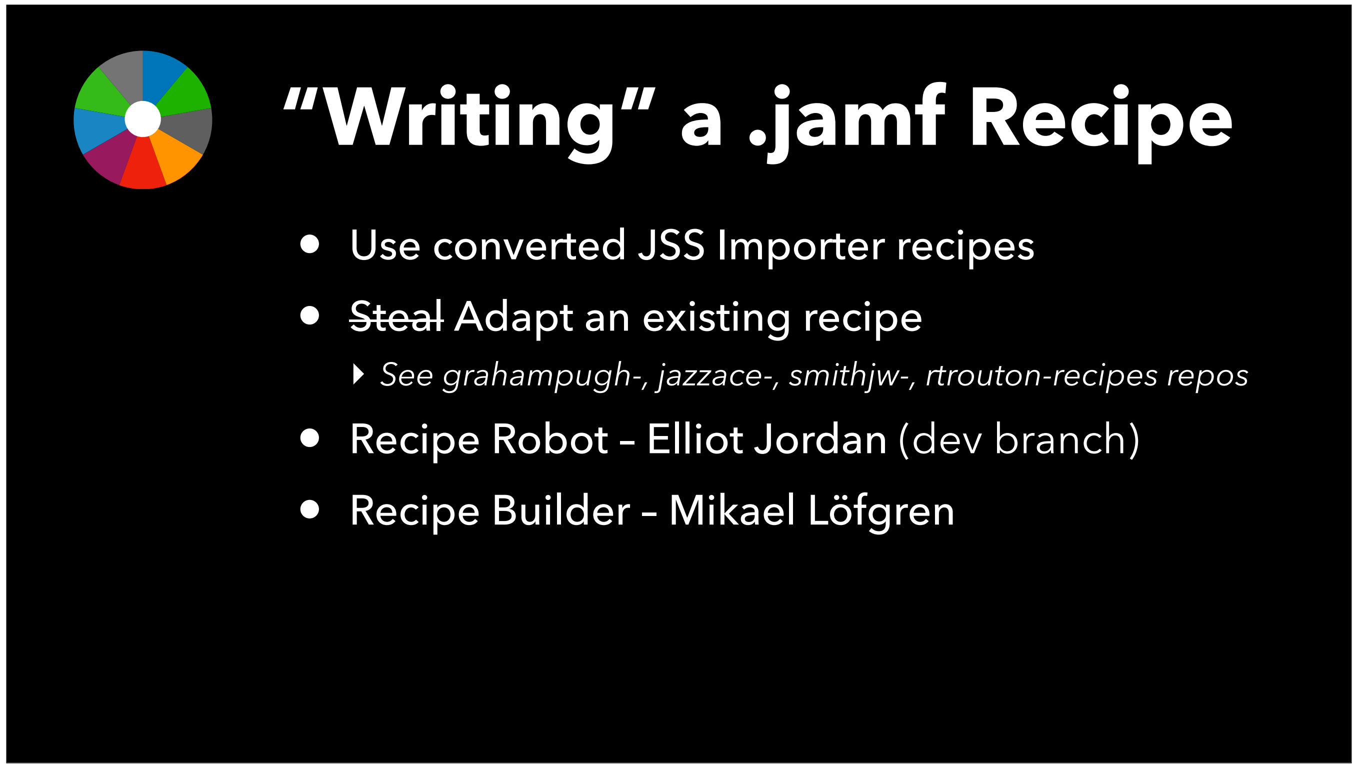


```
$ autopkg run Firefox.jamf
--post com.github.grahampugh.recipes.postprocessors/LastRecipeRunResult
```

```
$ autopkg run Firefox-prod.jamf
--pre com.github.grahampugh.recipes.postprocessors/LastRecipeRunChecker
--key recipeoverride_identifier=local.jamf.Firefox
```

grahamrpugh.com/2022/07/10/staging-jamf-recipes-to-production.html

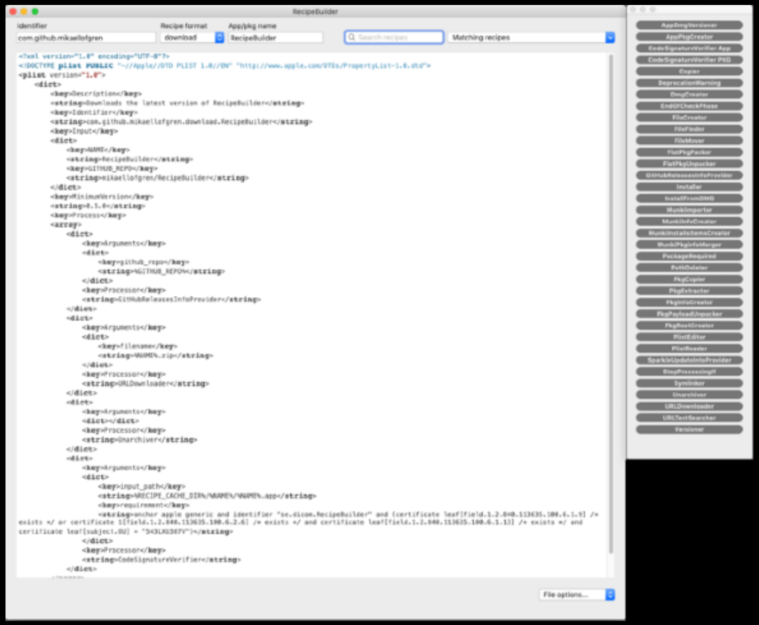
At my organisation, we stage a new package to production once a administrator in our org has tested that it works. This relies on a set of in-house processors that write relevant information about the latest imported package to a json file in the recipe cache. This file is read by a separate jamf recipe that runs when the package is ready to stage, grabbing the package name and version so that it can be injected into the production policies and smart groups.



“Writing” a .jamf Recipe

- Use converted JSS Importer recipes
- ~~Steal~~ Adapt an existing recipe
  - *See grahampugh-, jazzace-, smithjw-, rtrouton-recipes repos*
- Recipe Robot – Elliot Jordan (dev branch)
- Recipe Builder – Mikael Löfgren

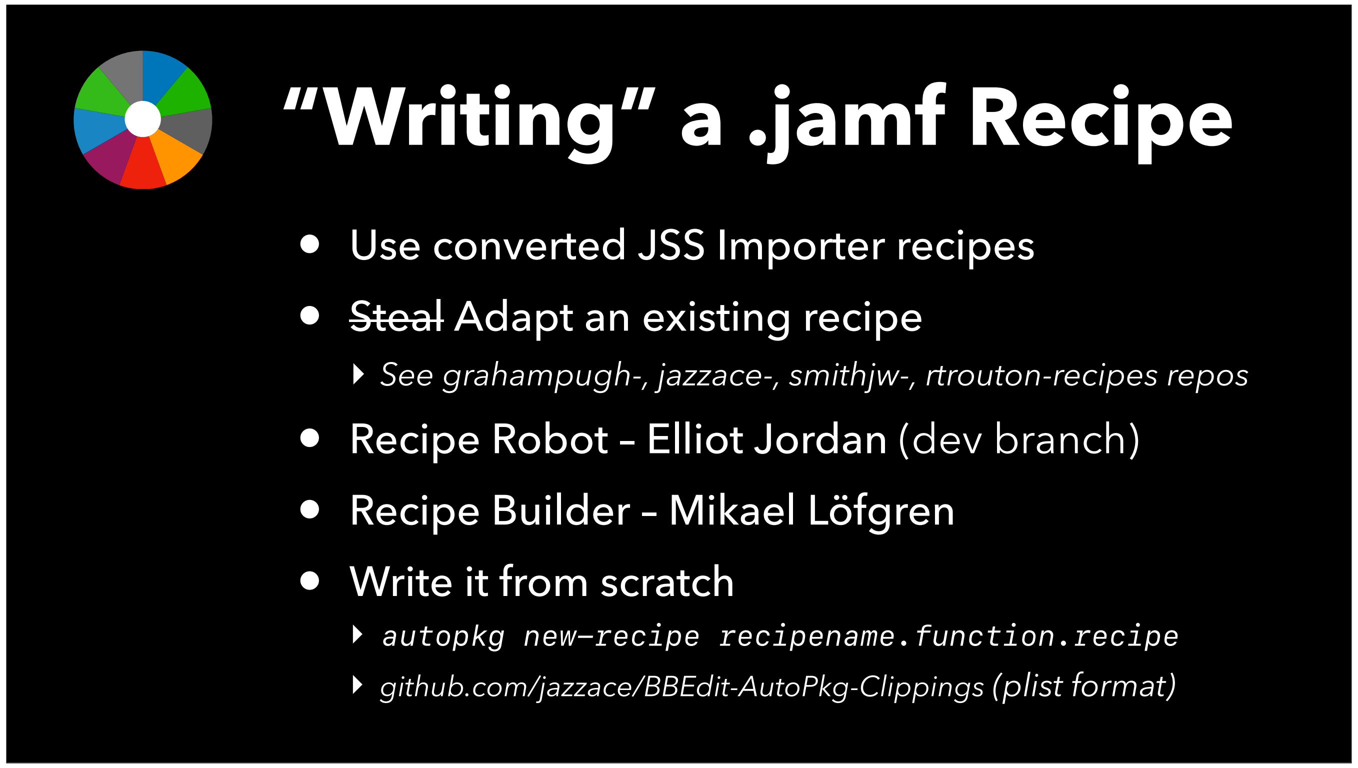
[Anthony:] To get the optimal experience from JamfUploader, you will probably want to write your own recipes. For some of you, this may be a new experience. Here are some ways you can go about doing it. [go though bullets] Here’s what Recipe Builder looks like…



Identifier: com.github.mikaellofgren
Recipe format: download
App/pkg name: RecipeBuilder
Search recipes
Matching recipes

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>Description</key>
        <string>Downloads the latest version of RecipeBuilder</string>
        <key>Identifier</key>
        <string>com.github.mikaellofgren.download.RecipeBuilder</string>
        <key>Input</key>
        <dict>
            <key>NAME</key>
            <string>RecipeBuilder</string>
            <key>GITHUB_REPO</key>
            <string>mikaellofgren/RecipeBuilder</string>
        </dict>
        <key>MinimumVersion</key>
        <string>0.5.0</string>
        <key>Process</key>
        <array>
            <dict>
                <key>Arguments</key>
                <dict>
                    <key>github_repo</key>
                    <string>%GITHUB_REPO%</string>
                </dict>
                <key>Processor</key>
                <string>GitHubReleasesInfoProvider</string>
            </dict>
            <dict>
                <key>Arguments</key>
                <dict>
                    <key>filename</key>
                    <string>%NAME%.zip</string>
                </dict>
                <key>Processor</key>
                <string>URLDownloader</string>
            </dict>
            <dict>
                <key>Arguments</key>
                <dict></dict>
                <key>Processor</key>
                <string>Unarchiver</string>
            </dict>
            <dict>
                <key>Arguments</key>
                <dict>
                    <key>input_path</key>
                    <string>%RECIPE_CACHE_DIR%/%NAME%/%NAME%.app</string>
                    <key>requirement</key>
                    <string>anchor apple generic and identifier "se.dicom.RecipeBuilder" and (certificate leaf[field.1.2.840.113635.100.6.1.9] /*
exists */ or certificate 1[field.1.2.840.113635.100.6.2.6] /* exists */ and certificate leaf[field.1.2.840.113635.100.6.1.13] /* exists */ and
certificate leaf[subject.OU] = "543LX8387Y")</string>
                </dict>
                <key>Processor</key>
                <string>CodeSignatureVerifier</string>
            </dict>
```

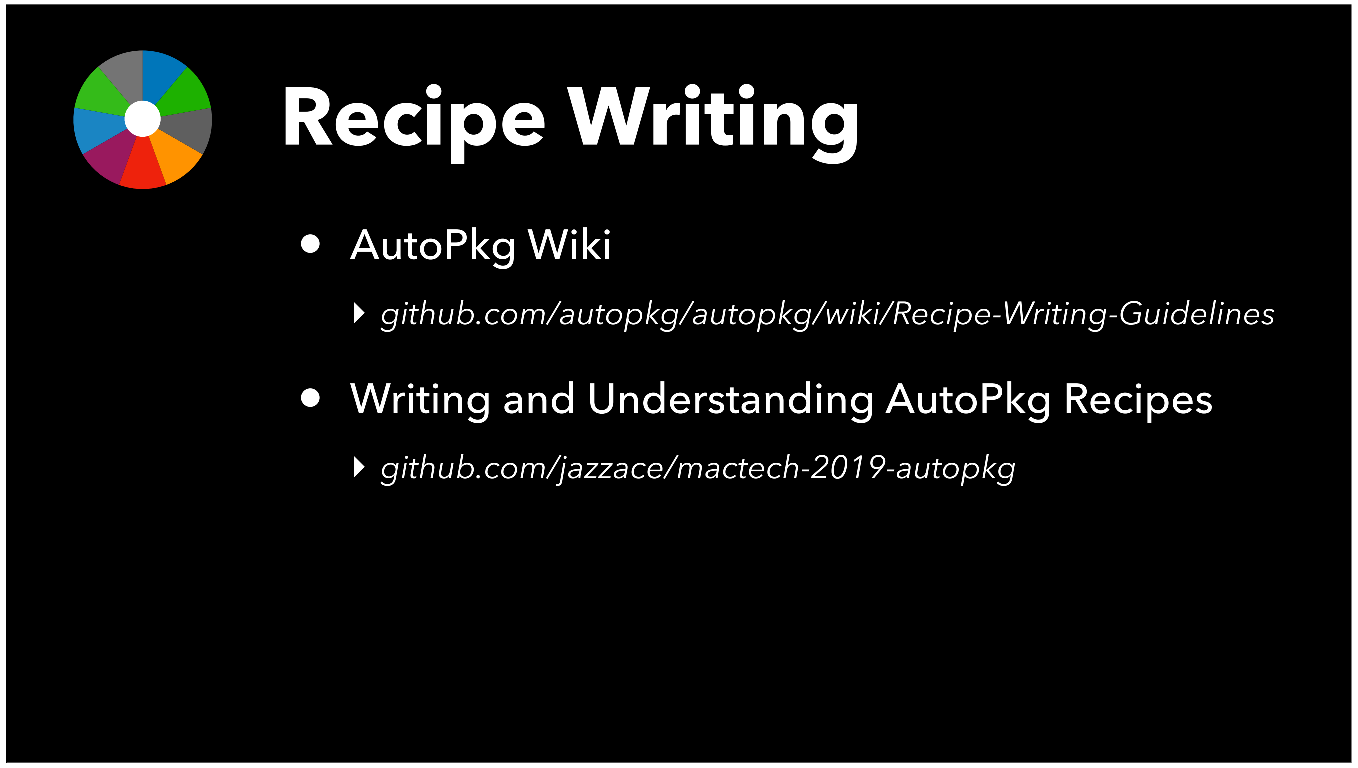
github.com/mikaellofgren/RecipeBuilder



"Writing" a .jamf Recipe

- Use converted JSS Importer recipes
- ~~Steal~~ Adapt an existing recipe
  - *See grahampugh-, jazzace-, smithjw-, rtrouton-recipes repos*
- Recipe Robot – Elliot Jordan (dev branch)
- Recipe Builder – Mikael Löfgren
- Write it from scratch
  - `autopkg new-recipe recipename.function.recipe`
  - *github.com/jazzace/BBEdit-AutoPkg-Clippings (plist format)*

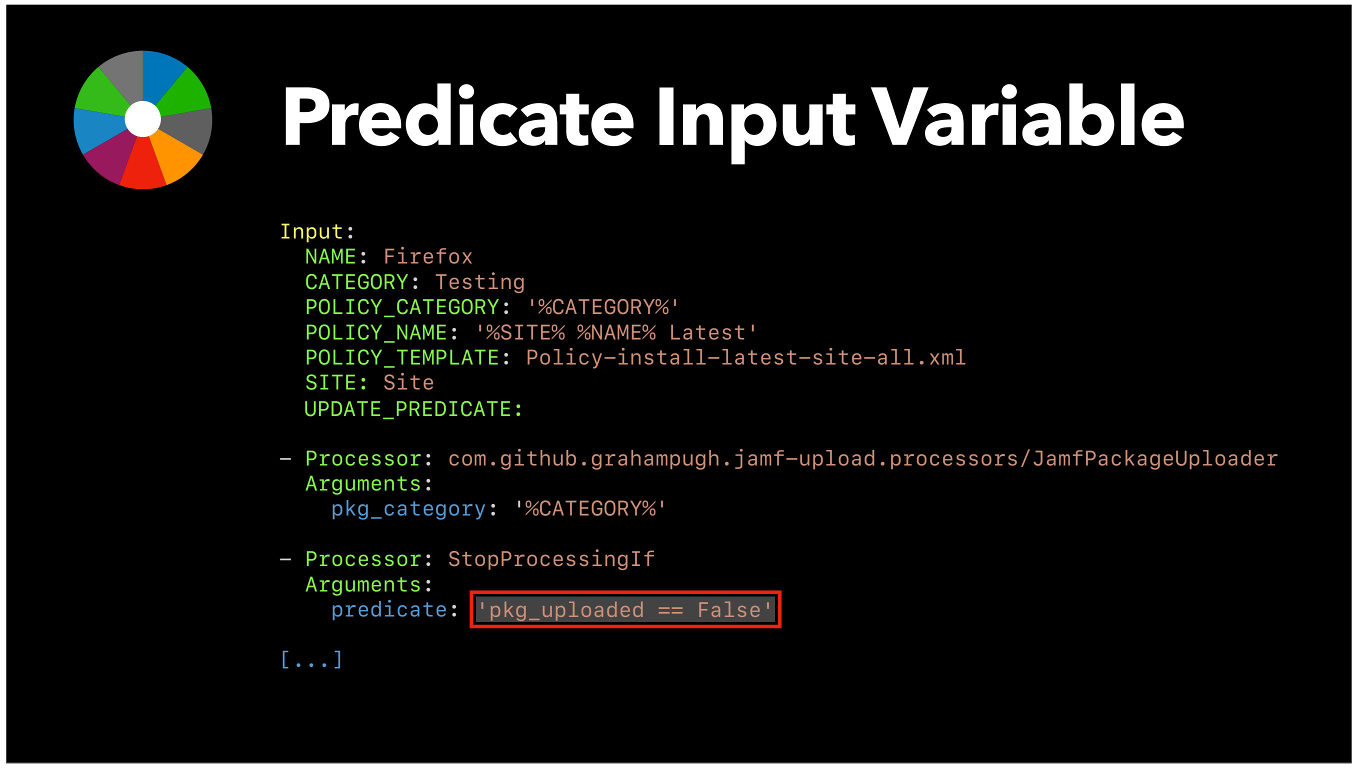
Finally, you can just write it from scratch if you really want, which is not too bad if you are just doing one of the Patch child recipes I showed earlier. When I am writing or modifying recipes, I have a constantly-evolving set of text clippings for BBEdit that insert the XML for a particular processor and its common arguments. I've made my current set available in my personal GitHub repo, both to share with others and because this allows me to keep my text clipping in sync between the different Macs I use for editing recipes.



Recipe Writing

- AutoPkg Wiki
  - *github.com/autopkg/autopkg/wiki/Recipe-Writing-Guidelines*
- Writing and Understanding AutoPkg Recipes
  - *github.com/jazzace/mactech-2019-autopkg*

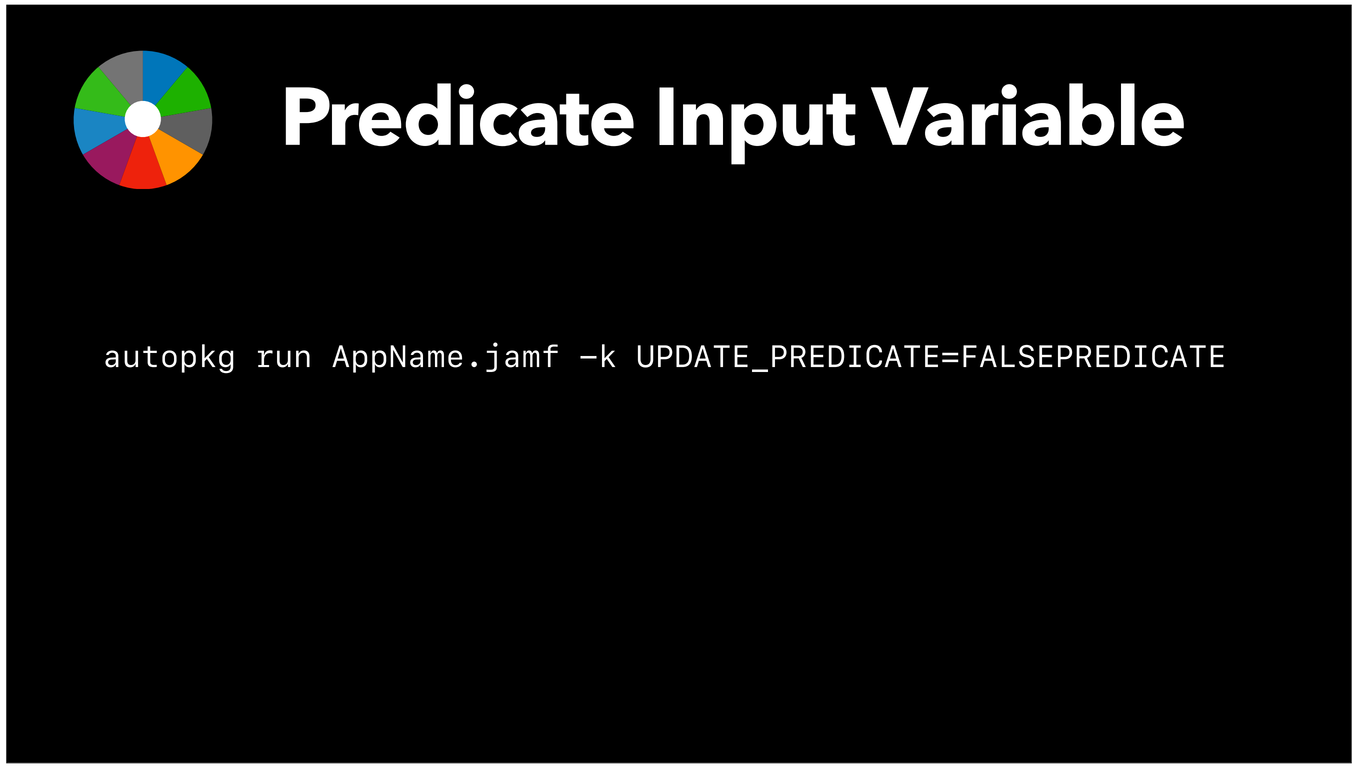
If you want more information on *general* AutoPkg recipe writing, check out the AutoPkg wiki, particularly the page I've linked in the slide. If you prefer to learn by watching videos, you can always check out the presentation I did at the 2019 Mac Tech Conference on that topic.



Predicate Input Variable

```
Input:
  NAME: Firefox
  CATEGORY: Testing
  POLICY_CATEGORY: '%CATEGORY%'
  POLICY_NAME: '%SITE% %NAME% Latest'
  POLICY_TEMPLATE: Policy-install-latest-site-all.xml
  SITE: Site
  UPDATE_PREDICATE:

- Processor: com.github.grahampugh.jamf-upload.processors/JamfPackageUploader
  Arguments:
    pkg_category: '%CATEGORY%'

- Processor: StopProcessingIf
  Arguments:
    predicate: 'pkg_uploaded == False'

[...]
```

[Anthony] You'll notice in all the recipes I currently have in my repo, I hard-code the condition on which I want processing to stop. In Graham's recipes, he makes the predicate an Input variable. Why would he do that?



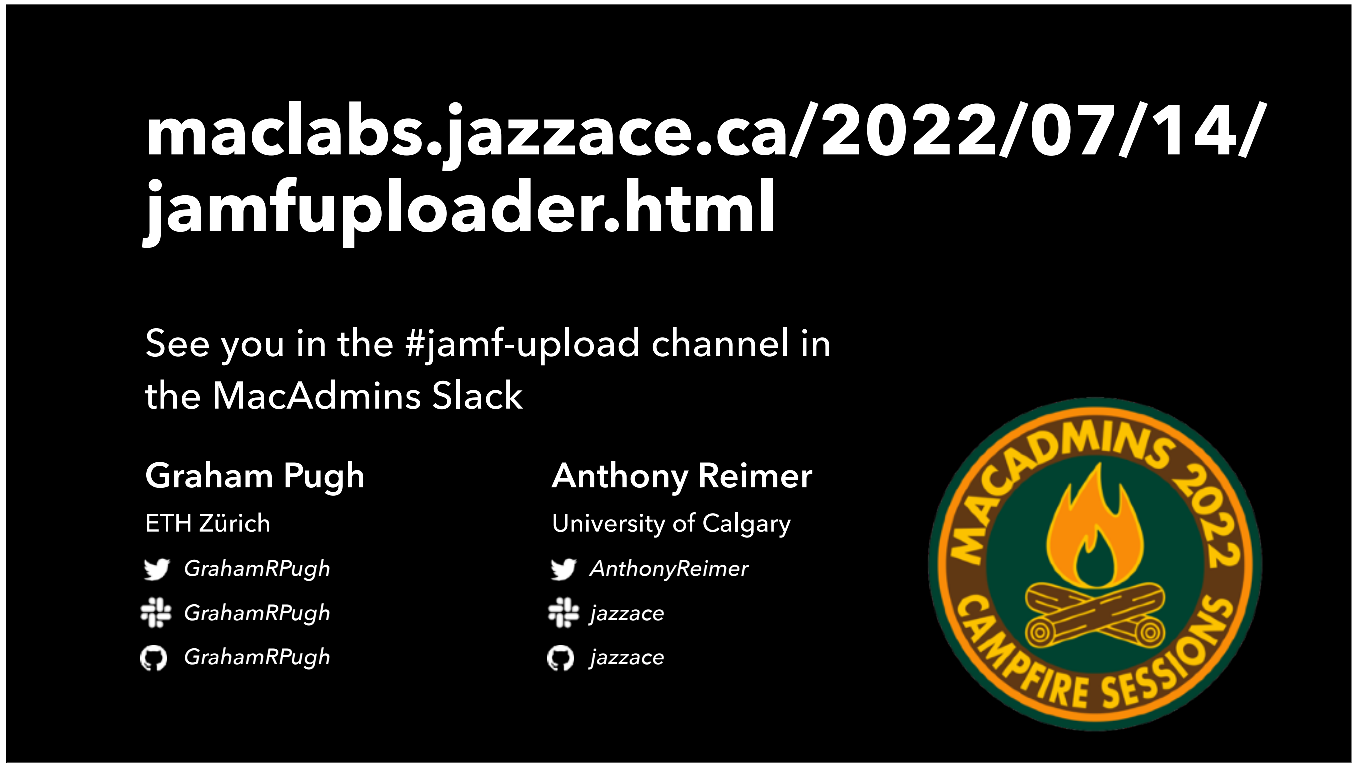
Predicate Input Variable

```
autopkg run AppName.jamf -k UPDATE_PREDICATE=FALSEPREDICATE
```

So he can do *this*. FALSEPREDICATE and TRUEPREDICATE are essentially reserved words that you can use to cause a predicate to resolve to False or True respectively. So when might we want to do this?

- When a new pkg uploads successfully but there is a failure in the policy or patch policy. We can re-run the recipe like this and tell it to ignore the fact that the pkg is the same as the last run so that the policy is built. That instance where AutoPkg fetches a new version before Jamf updates the Patch definition is one of those cases. I had that happen to me yesterday with BBEdit 14.5.1. I will have to manually make the change in the GUI because I hardcode my predicate.
- [Graham] yes, I recommend making the predicate overridable. During testing of new recipes I use this all the time. I also have a StopProcessingIf predicate in my recipes that stage packages to production, and if I want to force push something to production that hasn't had a full test report, I just override the staging predicate key value.

Camper Question!



maclabs.jazzace.ca/2022/07/14/jamfuploader.html

See you in the #jamf-upload channel in the MacAdmins Slack

Graham Pugh
ETH Zürich
GrahamRPugh
GrahamRPugh
GrahamRPugh

Anthony Reimer
University of Calgary
AnthonyReimer
jazzace
jazzace

MACADMINS 2022
CAMPFIRE SESSIONS

You can grab a copy of the slides and notes from today's talk from [my | Anthony's] blog and you can join us in the #jamf-upload channel of the MacAdmins Slack.