I’m Neil and I work in the End User Compute team which is part of IT Services at UEL. Like lots of folks working with Apple stuff in enterprise IT, I don’t have an IT background - I actually used to be a sound engineer…

<C> I’m also an admin on the Slack, where you may know me as this person.

I’d like to give a shout out to the London Apple Admins community - if you’re in town come along to our next meet up which will be at Jigsaw on the 12th of April!

If you want to know more about what we do, check out our website and come and say hello to one of our herders - Ben Toms, Darren Wallace, Graham Gilbert, Steve Quirke or me.

We’re always on the look out for future venues and speakers.

The University of East London

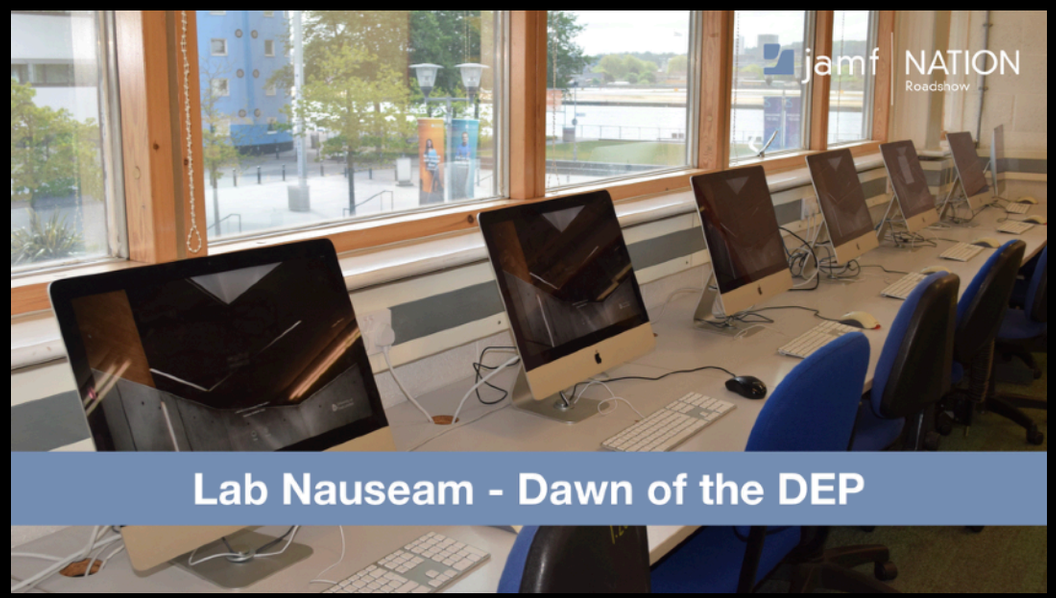
- Over 19,000 students
- 3 campuses
- 470 Macs, 200 in labs
- 2015 - 2018, Jamf Pro
- April 2018, Jamf Cloud

UEL has more than 19,000 students, spread over our Docklands, Stratford and University Square Stratford campuses.

My team manages around 5,000 PCs, with Macs making up about 10% of the estate. On both platforms, we're responsible for 2 key sorts of experience - a 1-1 model for staff, and a shared "Lab" model for students.

We've been managing our Macs with Jamf Pro since 2015 and last year, we migrated to Jamf Cloud.

https://soundmacguy.wordpress.com/2018/05/17/lab-nauseam-dawn-of-the-dep/



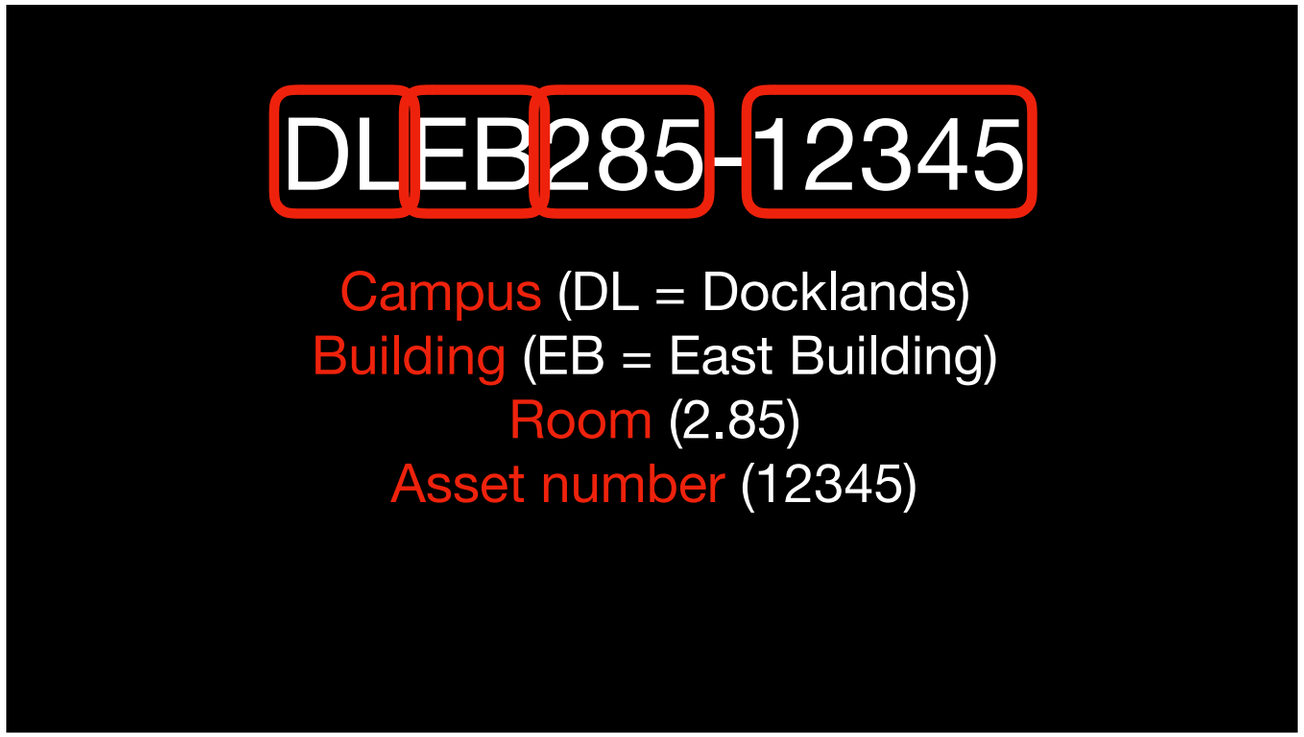
Last May, I spoke at the Jamf Nation Roadshow in London.
I talked about the fun I had re-engineering our Mac provisioning process; how we moved away from the old modular imaging workflows.
And how we got to grips with using Apple's modern methods, like the device enrolment programme, Apple School Manager and MDM.
If you check out the link, you can read all about it.

“Provisioning”

Our goal is to get our Macs deployed, configured and managed so they’re ready to give the experience we want to our students and staff. I’m lazy so I want to automate that as much as I can.
We use Jamf, so some of this talk will be specific to that. But for your environments, if you know what you want to achieve early on, you can definitely leverage these concepts in other management tools.

The Hostname matters to us

To manage a Mac the way we want to, we need to know a bit more information about it. For us, one piece of information is its hostname. We don’t just want to know it - we want to set it, because the naming convention we use defines how we’ll manage our Macs.

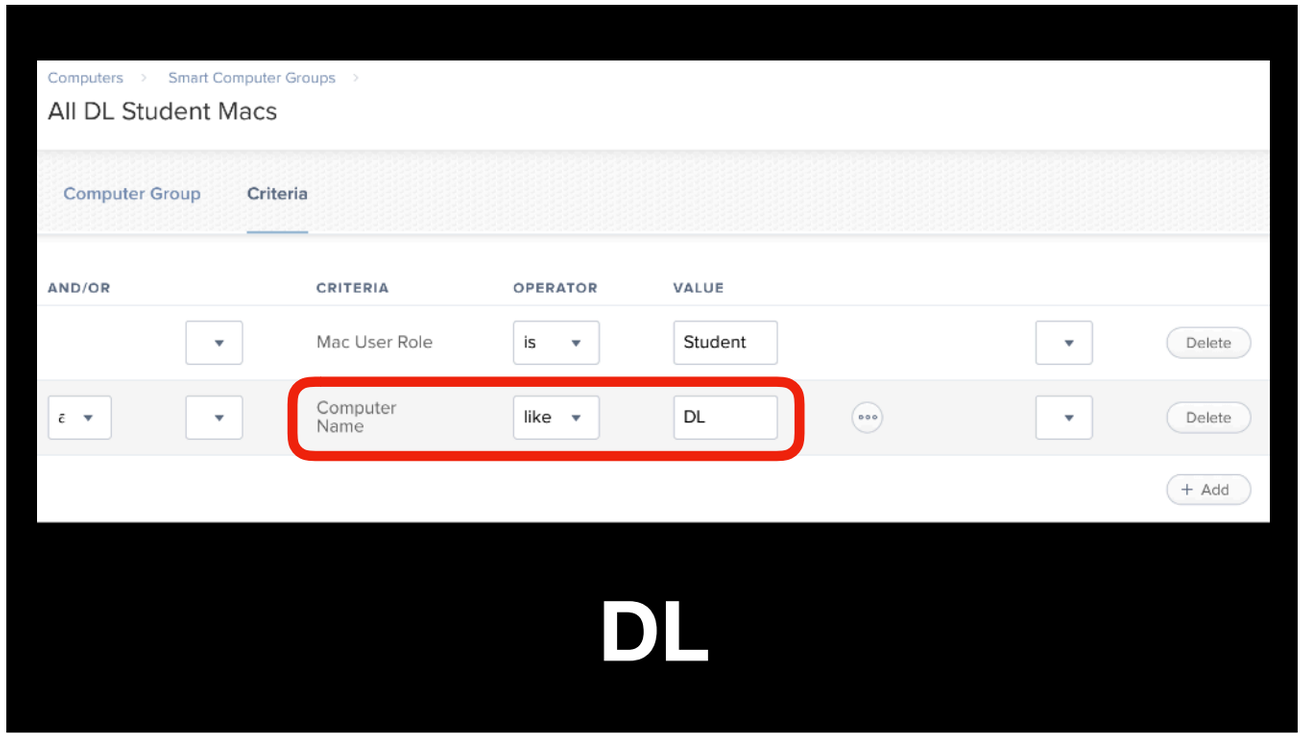


This is one of our typical hostnames. Let’s pick it apart. Our naming convention tells us the <C> Campus (Docklands), <C> Building (East Building), <C>, Room number <C> and our own internal Asset number.

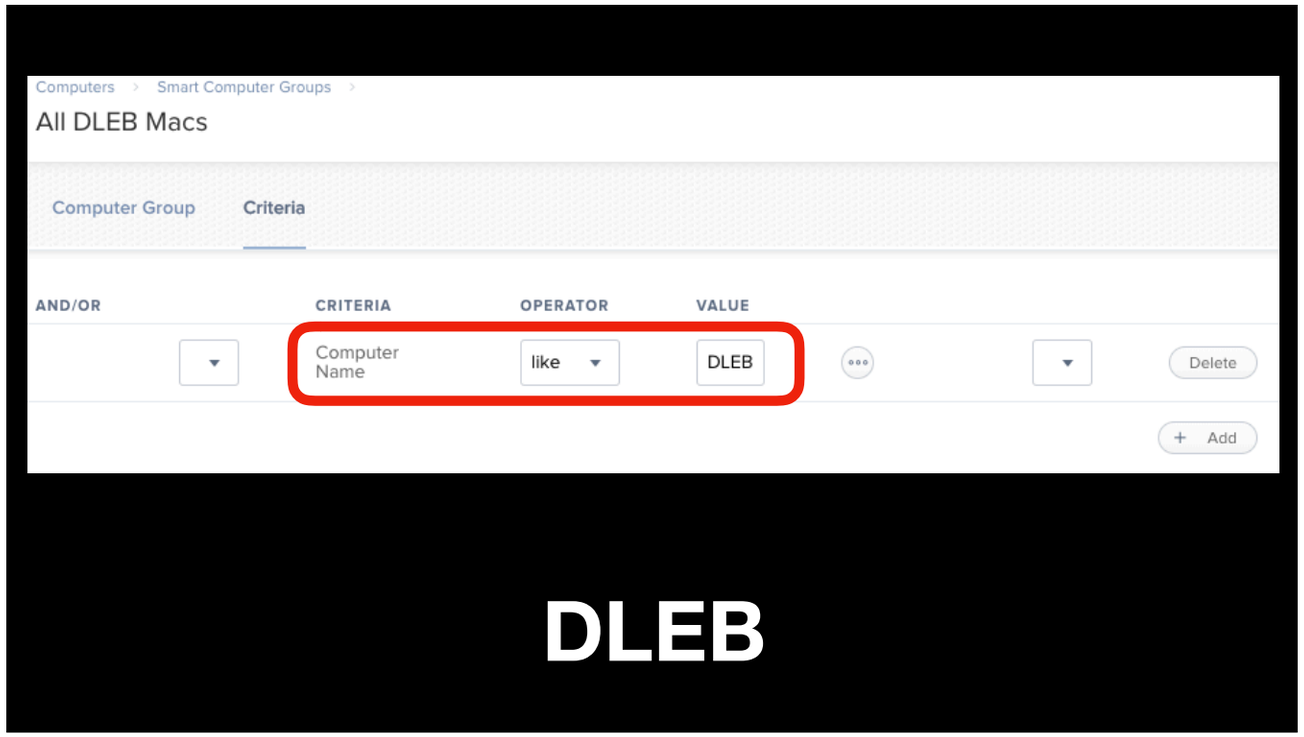
The role matters too…

Staff, or Student?

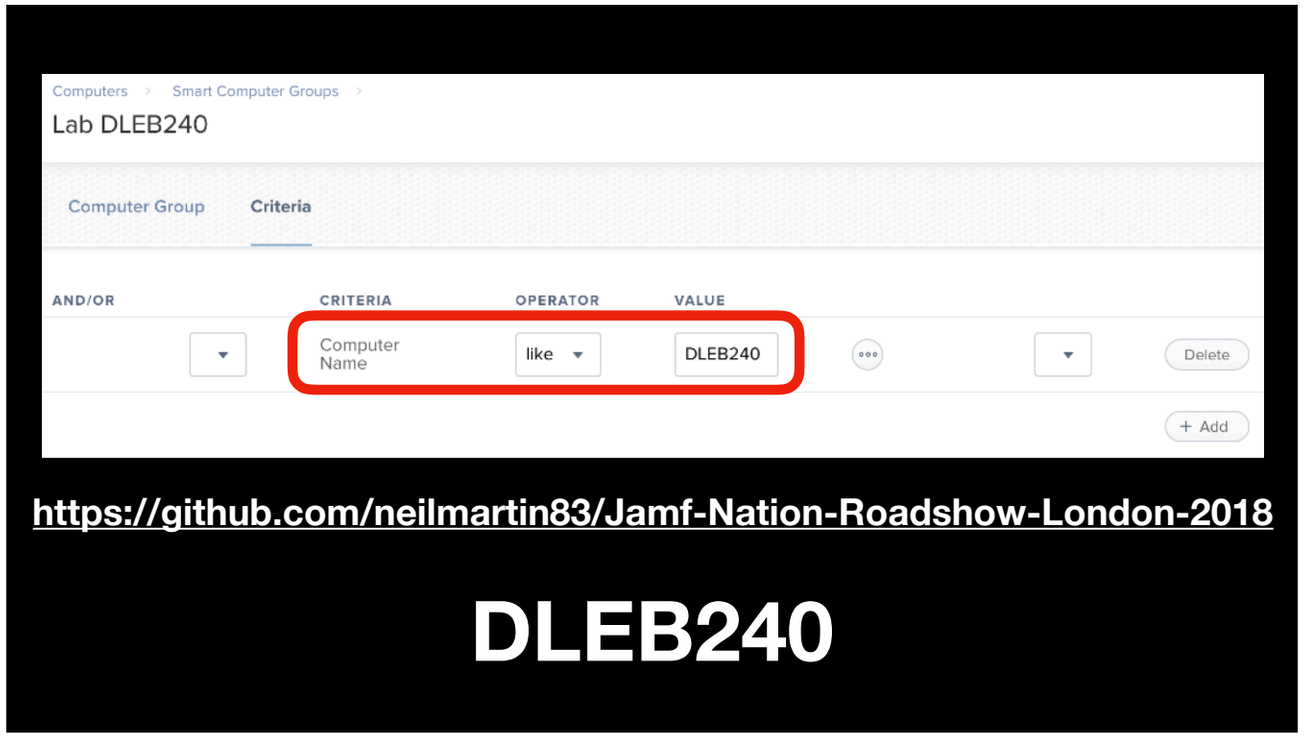
Along with the name, we need to know its role… Will the Mac sit in a member of staff's office and be used in a 1-1 setting? Is it going to be shared by lots of students in a lab? The experience we deliver is going to be different for each. So with Jamf, we'll make use of something called an extension attribute - that's the Jamf way of saying an extra piece of data about a computer we can define and collect. I would imagine other management tools probably have something similar.



With those two bits of information, we can create smart groups in Jamf, like this one, which will automatically populate with all the student facing Macs at our Docklands campus.



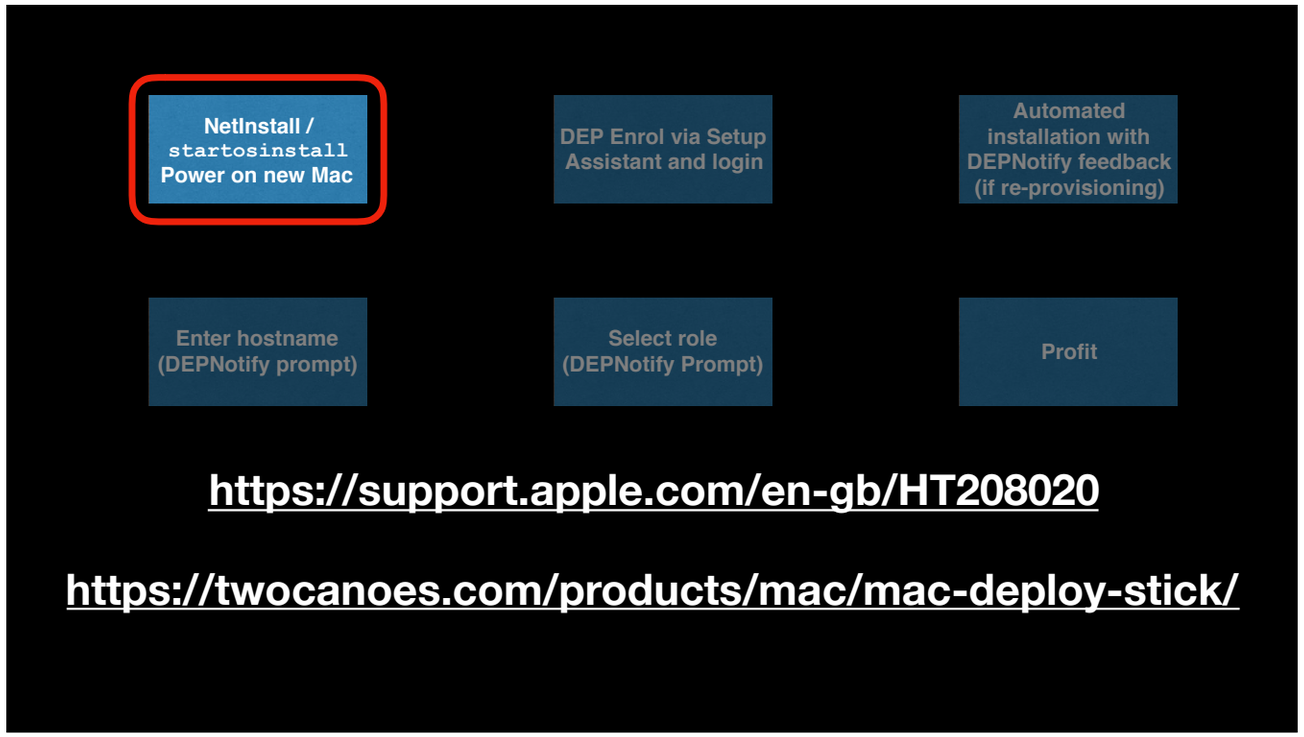
For all the Macs in the East Building at Docklands…



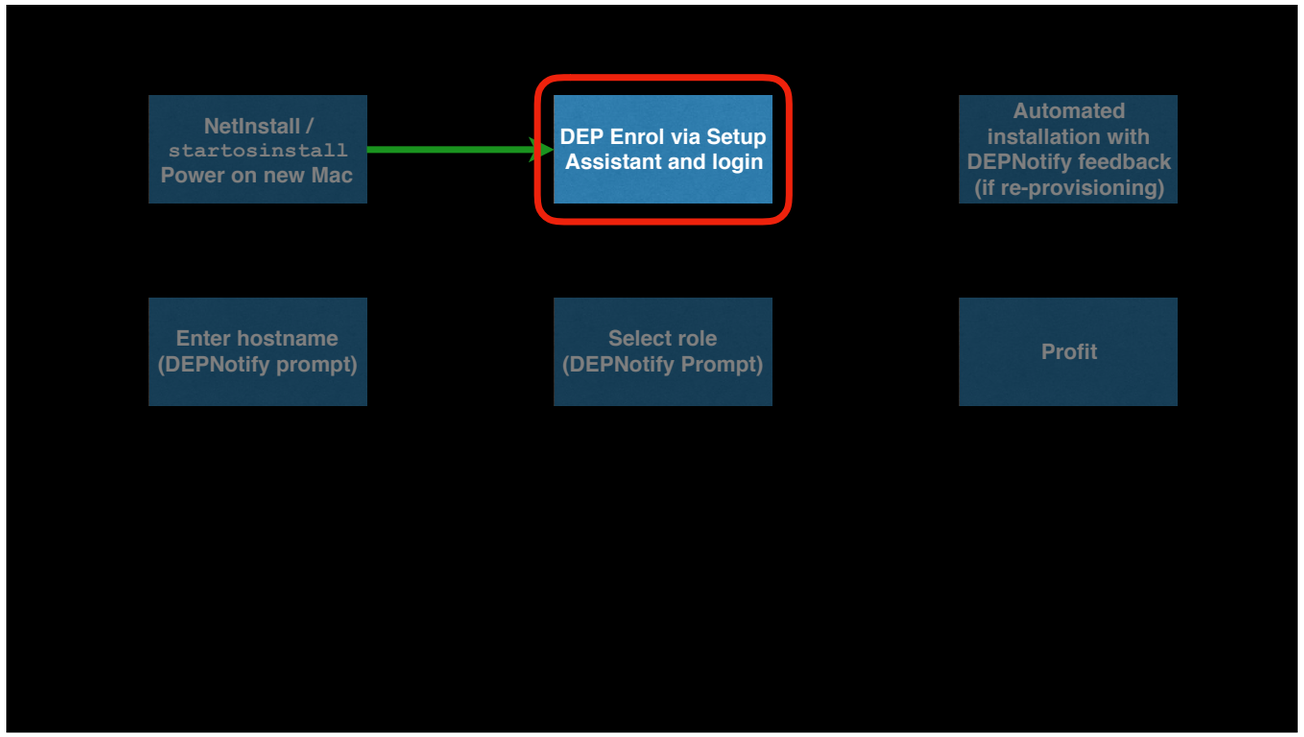
Or for a specific lab by its room number.

This kind of framework gives you a lot of flexibility when it comes to defining the software and settings you want to deploy to your computers, depending on who they're used by and where they are, or combinations thereof.

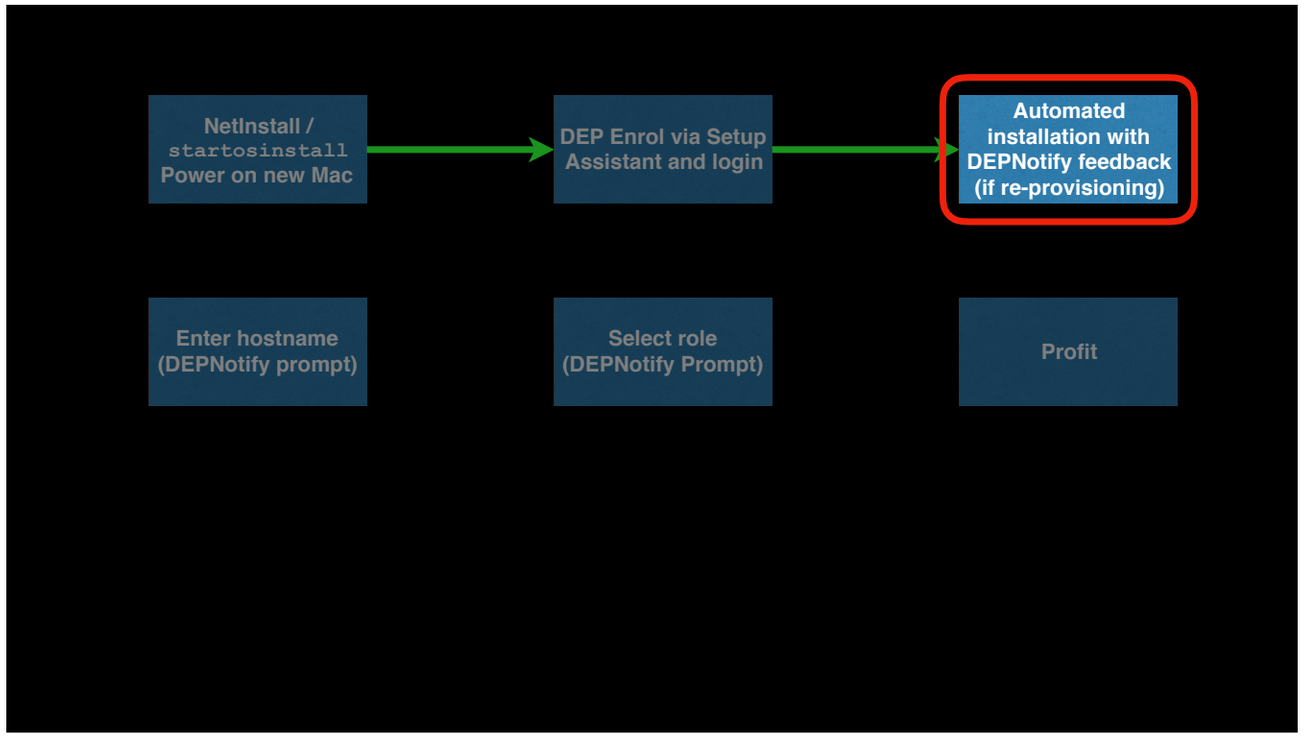
That's beyond the scope of this session but if it's something you're interested in, please do have a look at this link where I've gone into detail with an example of how you'd deploy applications using this model.



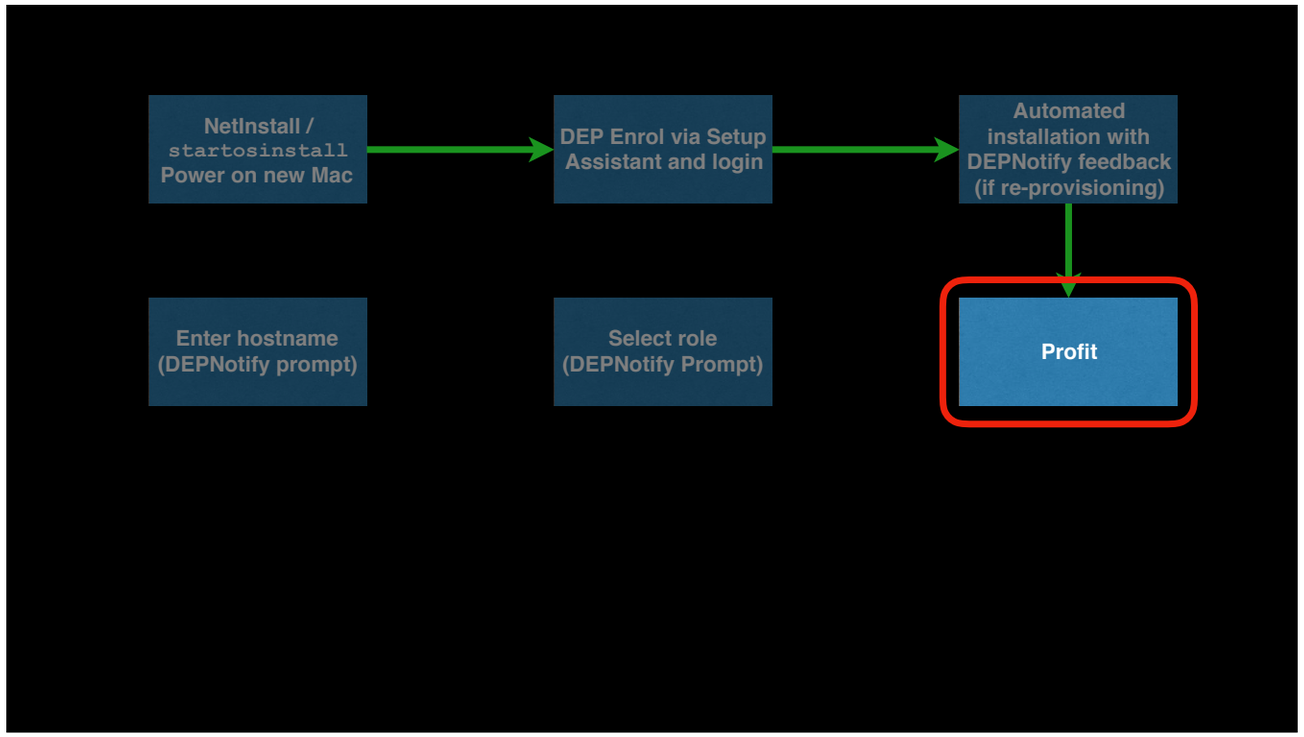
So, back to provisioning. Our previous workflow went something like this…
We begin with a fresh, clean install of macOS.
For a new Mac, that's easy - Apple have done that for you already.
If you're re-provisioning a Mac you're already managing, there are a few ways to achieve this. For our older Macs that don't have the T2 chip which is most of our labs, I still use NetInstall with Graham Gilbert's fantastic tool, Imagr to achieve a "vanilla" erase/install. But, once they go to Mojave. it'll be the end of NetBoot for us.
That's because, if the boot volume on your Mac is using the new APFS filesystem, you can use your management tool to deploy the macOS installer application and remotely run the startosinstall command with the eraseinstall flag.
As an asied, Tim Perfitt has recently done some fantastic work on a product called Mac Deployment Stick - you should definitely check that out too.
And of course, there's Internet Recovery, which I find is okay for one-offs, but it's not scalable when you've got dozens, hundreds or thousands or Macs to churn through, or if internet bandwidth is an issue for you.



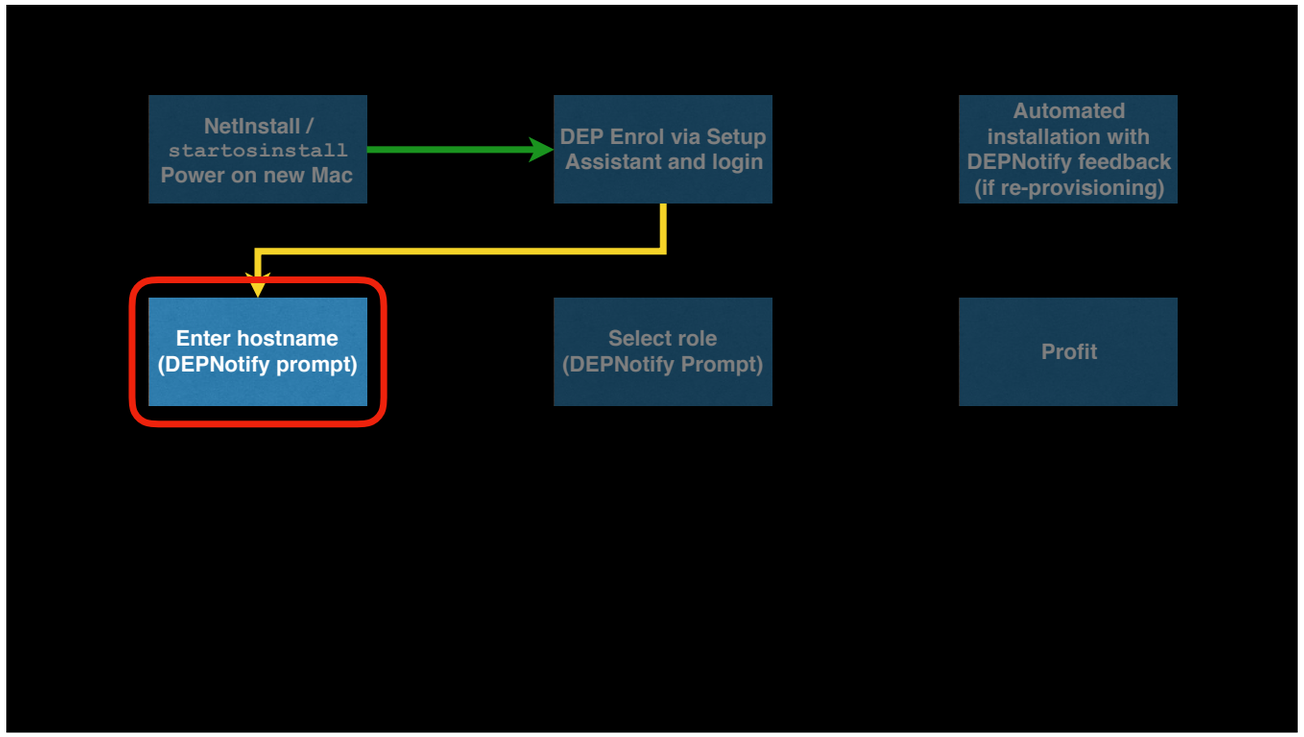
Next up was getting the fresh clean Mac under management - quite a few things happen here.
We use DEP, and assign them to a PreStage in Jamf.
We skip all the steps of the setup assistant we can. We also bound to AD at this point so a technician can log in to continue the process. At this stage, a policy kicks off with the enrolment complete trigger and runs a script to drive the rest of the process.



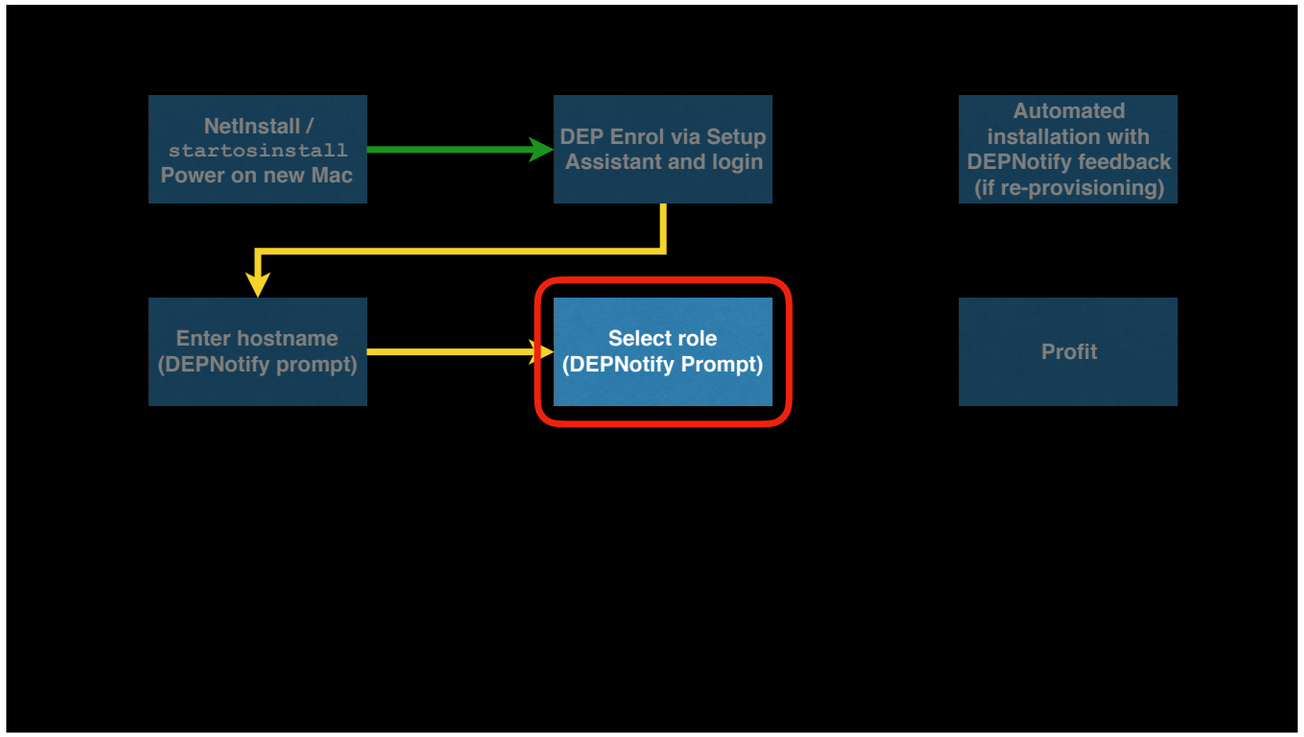
Next up, we drove one of Joel’s other awesome tools; DEPNotify, to get it to tell the provisioner what was happening.
If the Mac had been provisioned before, and had an old record in Jamf - the rest was automated. We'd reuse its old hostname and role and it would get the same software and settings it had before.



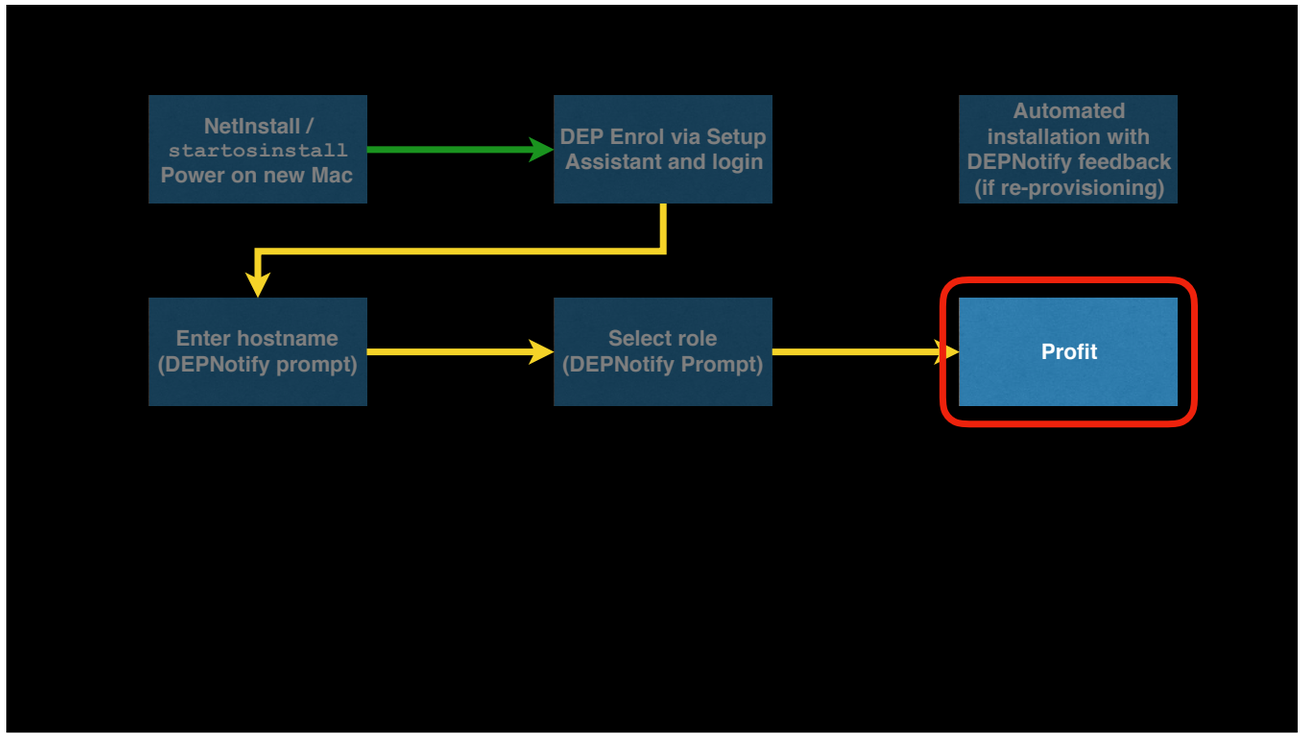
And then, the rest was magic and we'd walk away, job done.



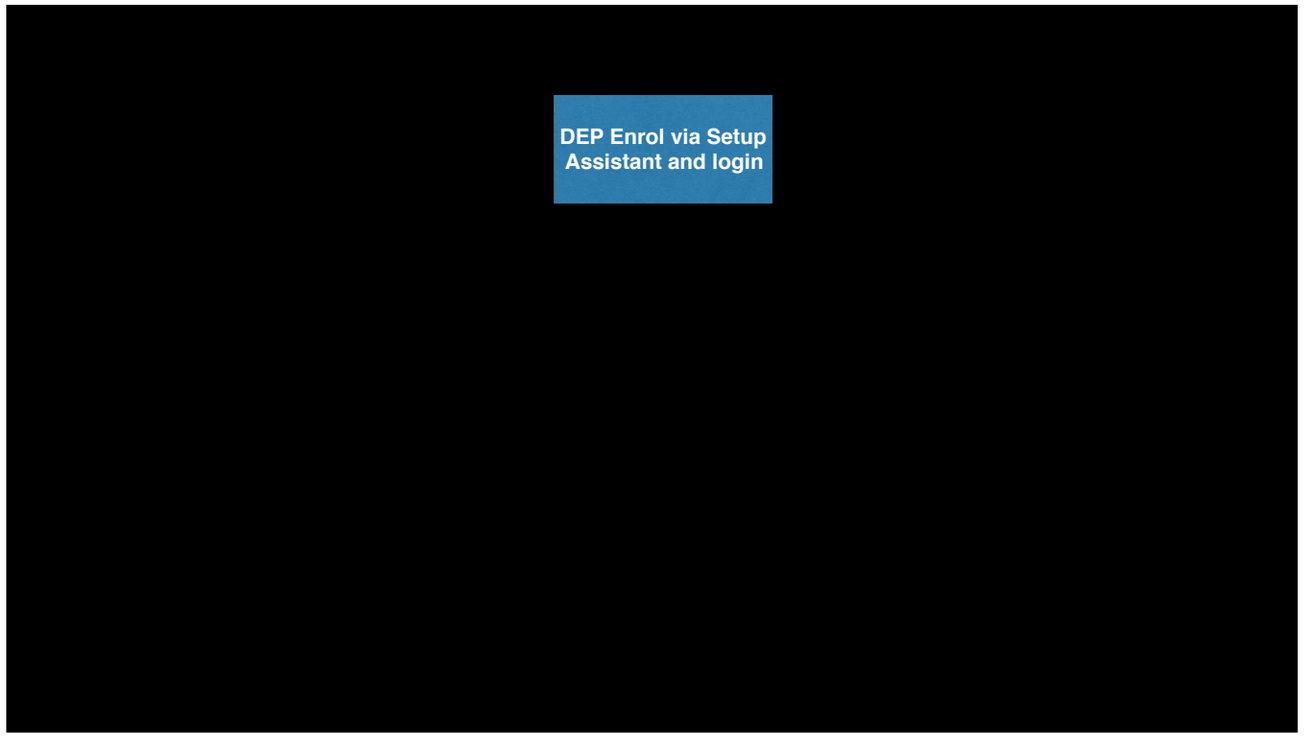
Now, in the case of a new Mac, or one we were re-purposing…
We'd delete its computer record so it was essentially new to Jamf. And in this situation, we would tell DEPNotify to ask our provisioner to input some information. So, the hostname.


NetInstall /
startosinstall
Power on new Mac
DEP Enrol via Setup Assistant and login
Automated installation with DEPNotify feedback (if re-provisioning)
Enter hostname (DEPNotify prompt)
Select role (DEPNotify Prompt)
Profit

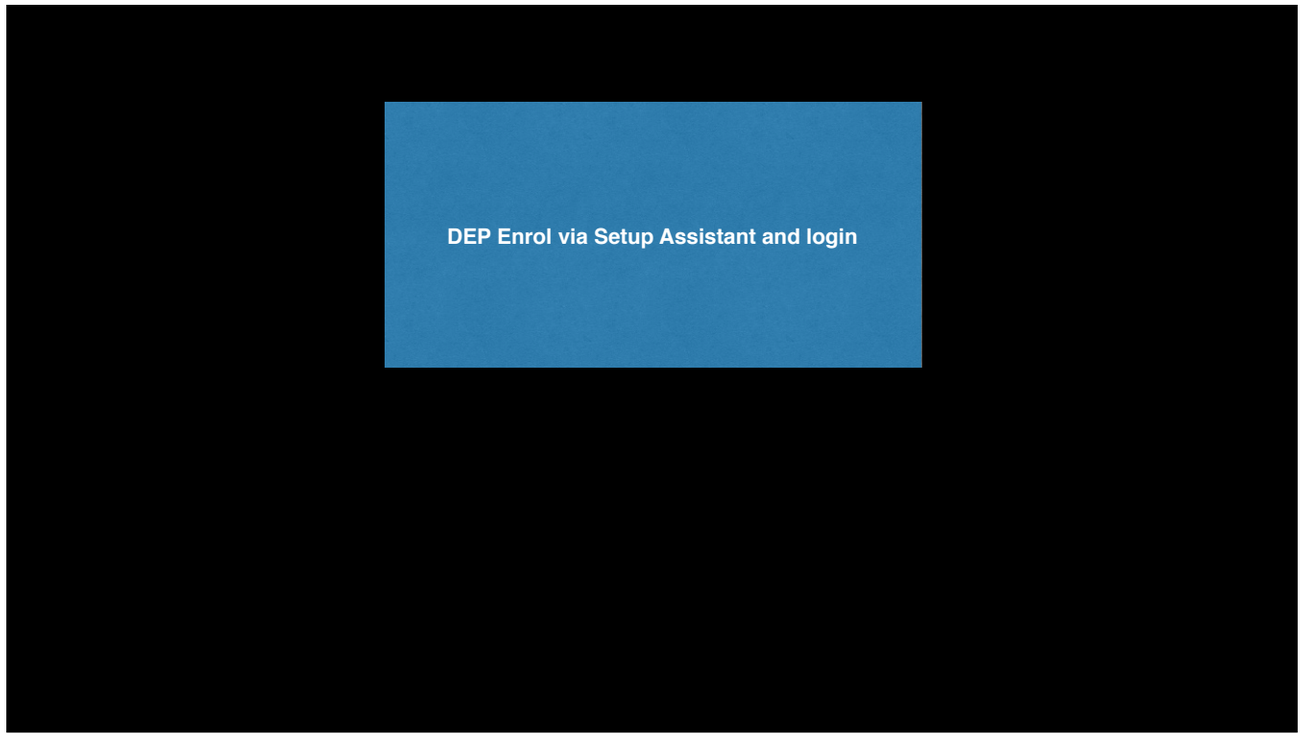
And the role.



And then, the rest was magic…



But - there’s room for improvement.



And when something’s not quite right, it really plays on my mind…

And the conundrum going around my mind was: If you’re just provisioning a computer for someone else to use, why should you need to log into it?

The Setup Assistant isn’t enough

It’s because the Setup Assistant doesn’t do everything we need it to do.
It doesn’t let us provide extra information we need for our management tools to get the right things to it straight away.
It doesn’t tell us what’s happening, which software is being installed and so on…

DEPNotify is awesome!

That’s why, for my environment, DEPNotify really worked to extend the Setup Assistant.
It gave us a way to supply extra information and it provided great feedback about what was happening .
But… it would only run after a user had logged in.

And then there was NoMAD Login…

And this is why I’m here today. Because Joel and his fantastic team merged what DEPNotify does into NoMAD Login!

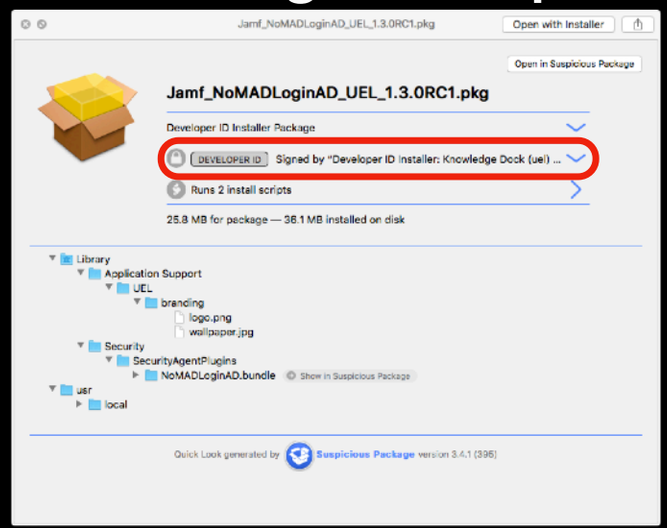
Now, please join me as we look at what happened next…

It’s at this point I have a bit of a confession to make - I’m only using the Notify and User Input parts of NoMAD Login - right now, we’re binding to AD and using mobile accounts, so I’m not yet making use of the awesome stuff it can do with creating local accounts out of your user’s directory account credentials - sorry Joel! On the flip side of that, NoMAD Login is modular - and you have the flexibility to use whichever parts or mechs you want.

Ingredients for login-less provisioning

Now - I want to achieve a provisioning workflow where the provisioner doesn’t need to log in - they’ll just need to go through the Setup Assistant then input the name and role (or not if we’re re-provisioning a Mac against its previous name and role)…

Ingredients for login-less provisioning



We'll need an installer package that contains NoMAD Login. It has to be signed with a developer ID certificate because of how we'll be deploying it with Jamf - other MDMs might need this too. You need access to an Apple developer account to get that certificate.

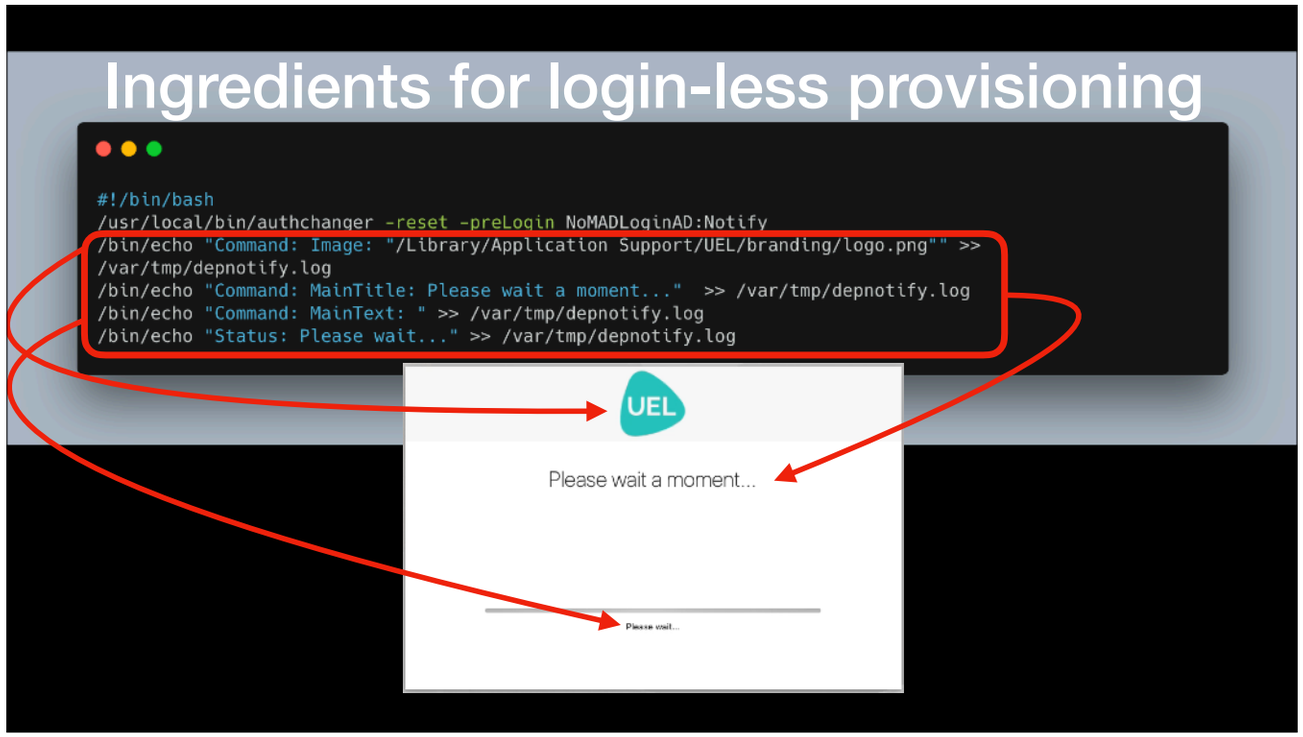
Ingredients for login-less provisioning

I took the package provided on the NoMAD Login gitlab repo and re-packaged it, to include our branding; a logo and wallpaper

Ingredients for login-less provisioning

```
#!/bin/bash
/usr/local/bin/authchanger -reset -preLogin NoMADLoginAD:Notify
/bin/echo "Command: Image: /Library/Application Support/DEL/Branding/logo.png"" >>
/var/tmp/depnotify.log
/bin/echo "Command: MainTitle: Please wait a moment..."  >> /var/tmp/depnotify.log
/bin/echo "Command: MainText: " >> /var/tmp/depnotify.log
/bin/echo "Status: Please wait..." >> /var/tmp/depnotify.log
```

I tweaked the postinstall script. We run authchanger, which a great tool that comes with NoMAD Login that makes changes to the authorisation database safely. We get it to swap out Apple’s login window for just NoMAD Login’s Notify mech.



And echo the commands we needed to make the Notify mech show our logo, along with title and status text asking the provisioner to wait. If you've used DEPNotify - you'll see that those commands are exactly the same - since NoMAD Login's Notify part is DEPNotify.

Ingredients for login-less provisioning

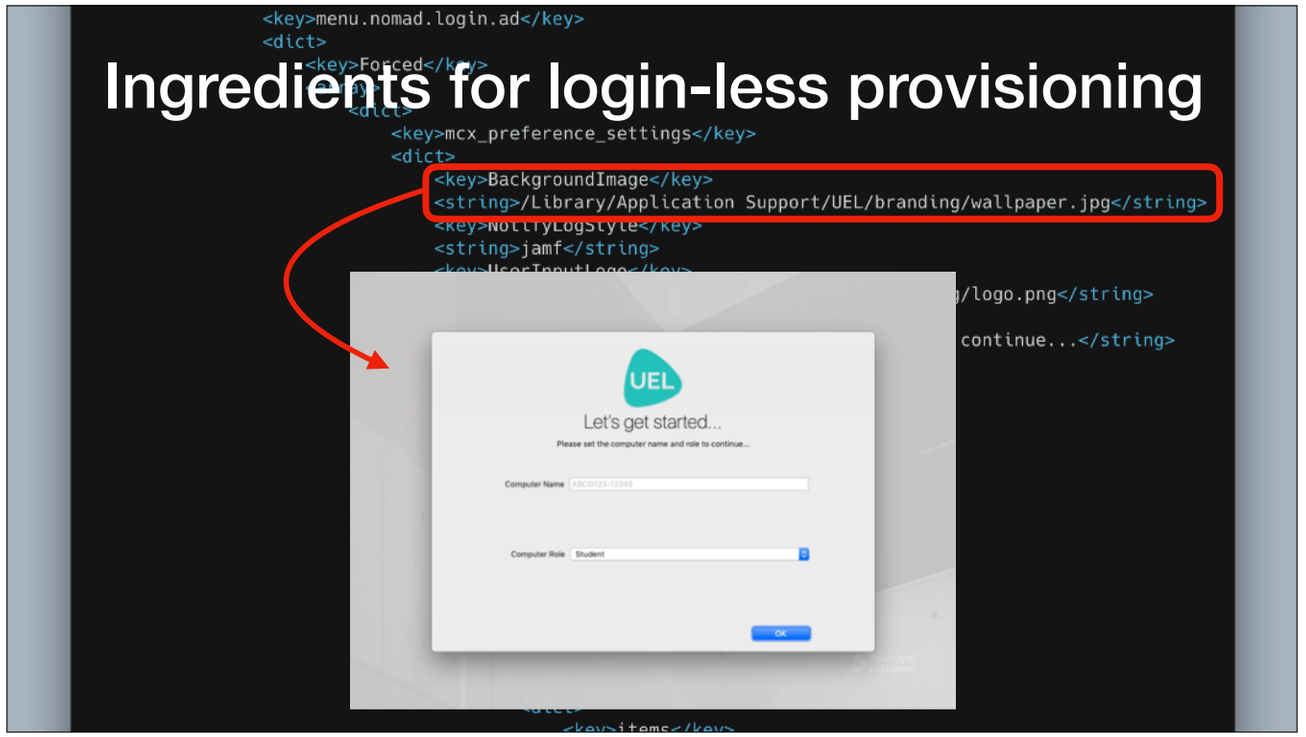
Next up, we need a configuration profile

Ingredients for login-less provisioning

```
<?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>PayloadContent</key>
    <array>
        <dict>
            <key>PayloadContent</key>
            <dict>
                <key>menu.nomad.login.ad</key>
                <dict>
                    <key>Forced</key>
                    <array>
                        <dict>
                            <key>mcx_preference_settings</key>
                            <dict>
                                <key>BackgroundImage</key>
                                <string>/Library/Application Support/UEL/branding/wallpaper.jpg</string>
                                <key>NotifyLogStyle</key>
                                <string>jamf</string>
```

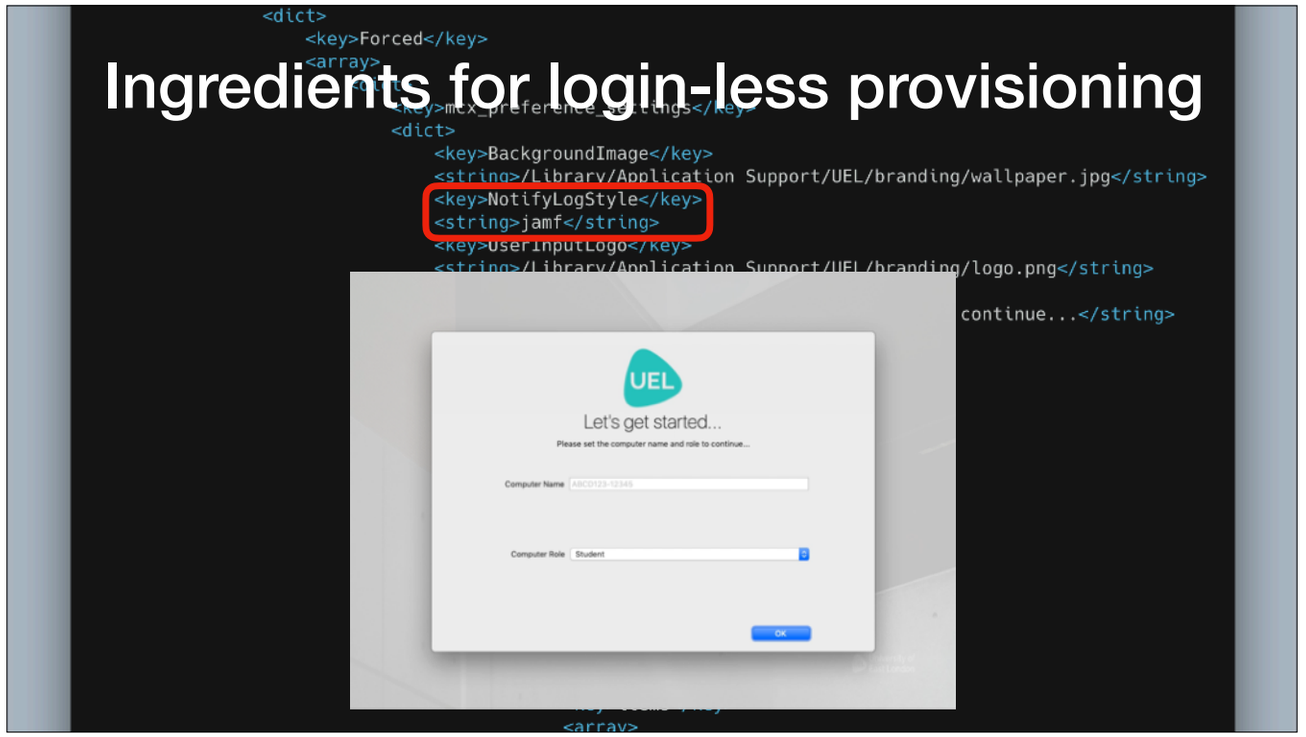
We’re managing the menu.nomad.login.ad domain

Ingredients for login-less provisioning

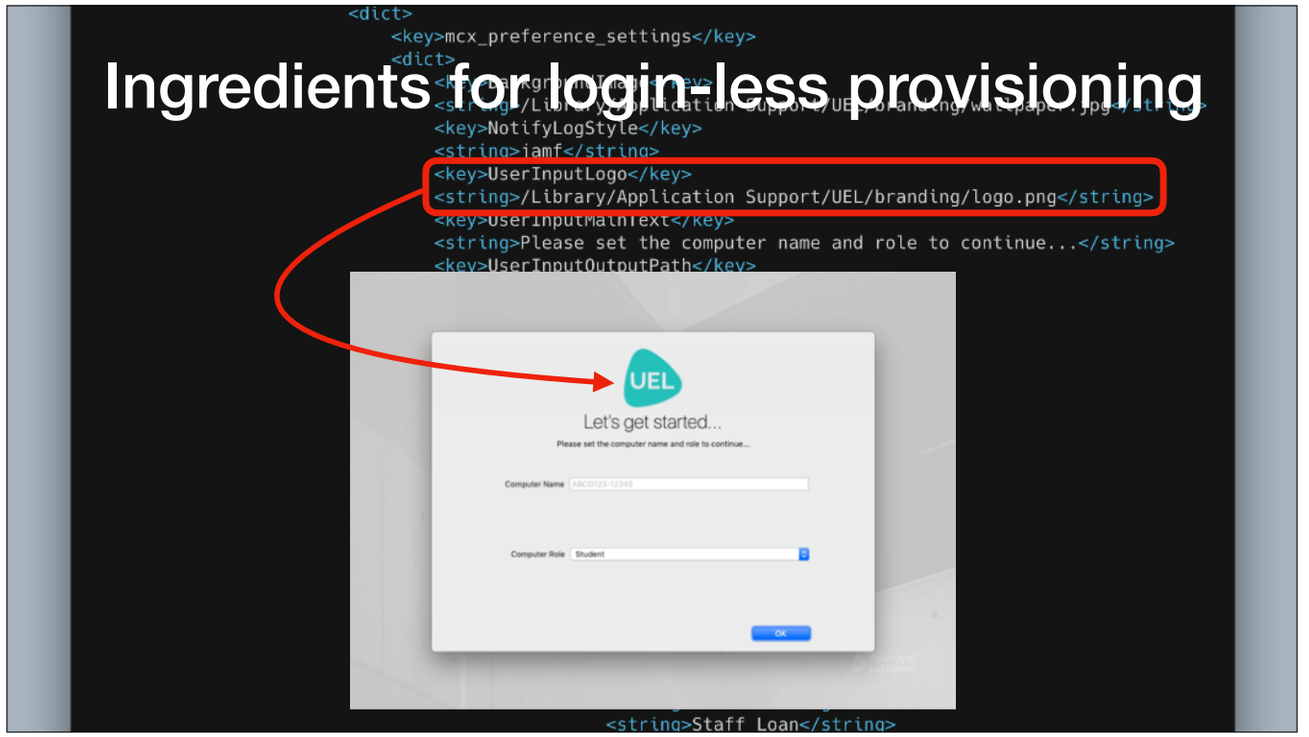


We set NoMAD Login’s background image to the one we installed with our package

Ingredients for login-less provisioning



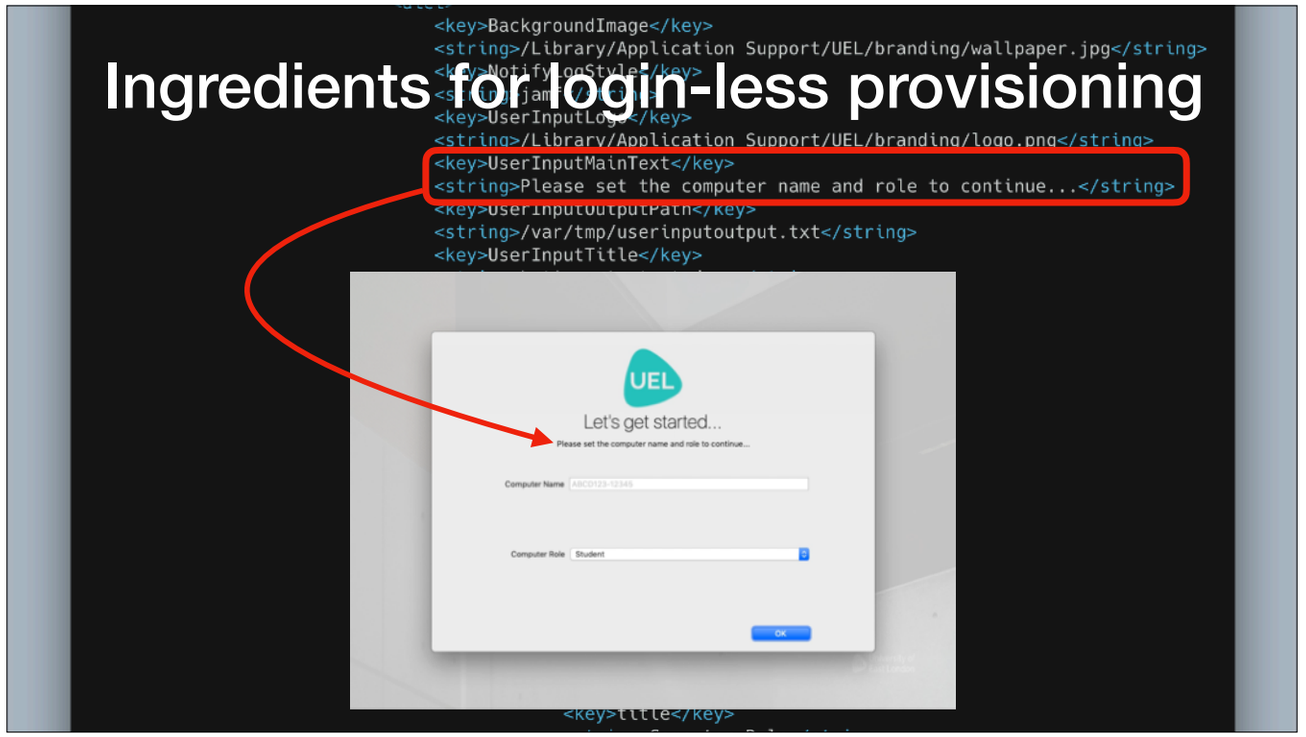
We tell the Notify part of NoMAD login to parse the Jamf log, so that the status text will update to show package installs done by policies. This supports other management tools including munki.



Ingredients for login-less provisioning

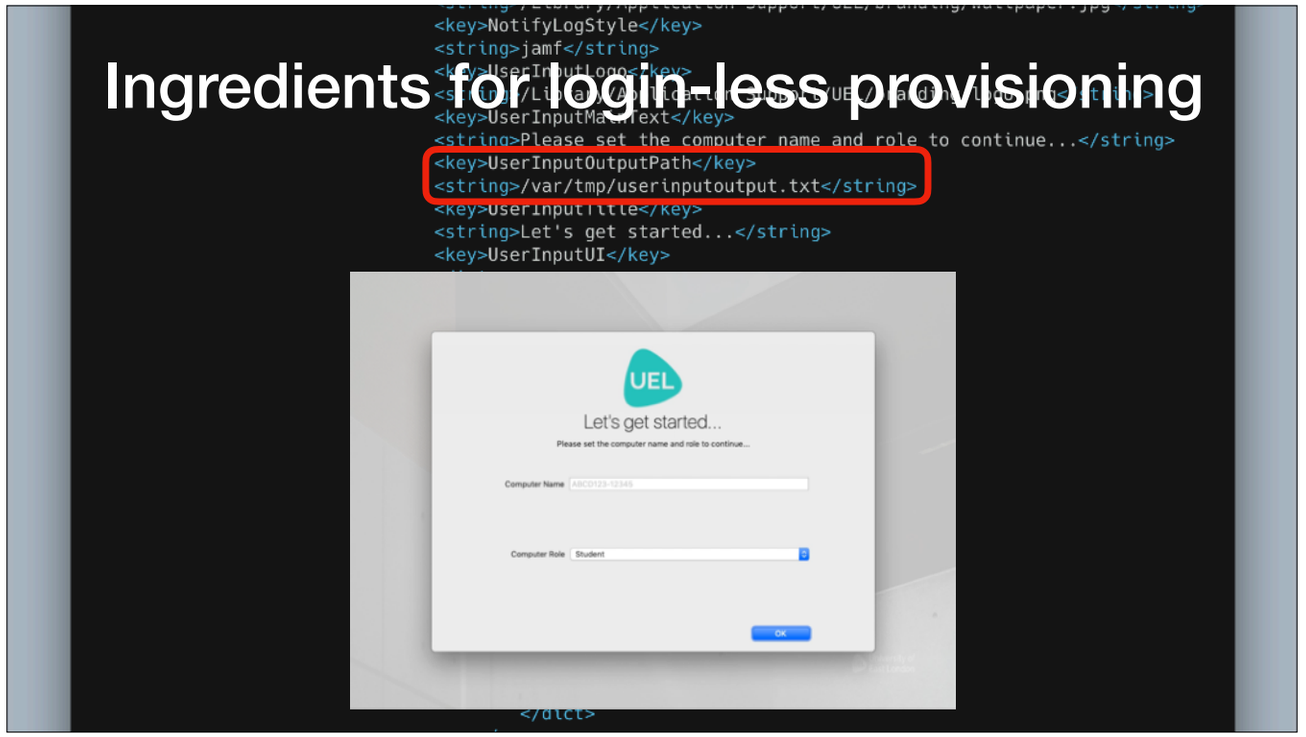
Now we come to the settings we need for NoMAD Login's User Input screen. This is separated out from the Notify screen, as you'll see soon, but we need to set up a few things before we use it. First of all, we'll tell it to use our branding logo.

Ingredients for login-less provisioning



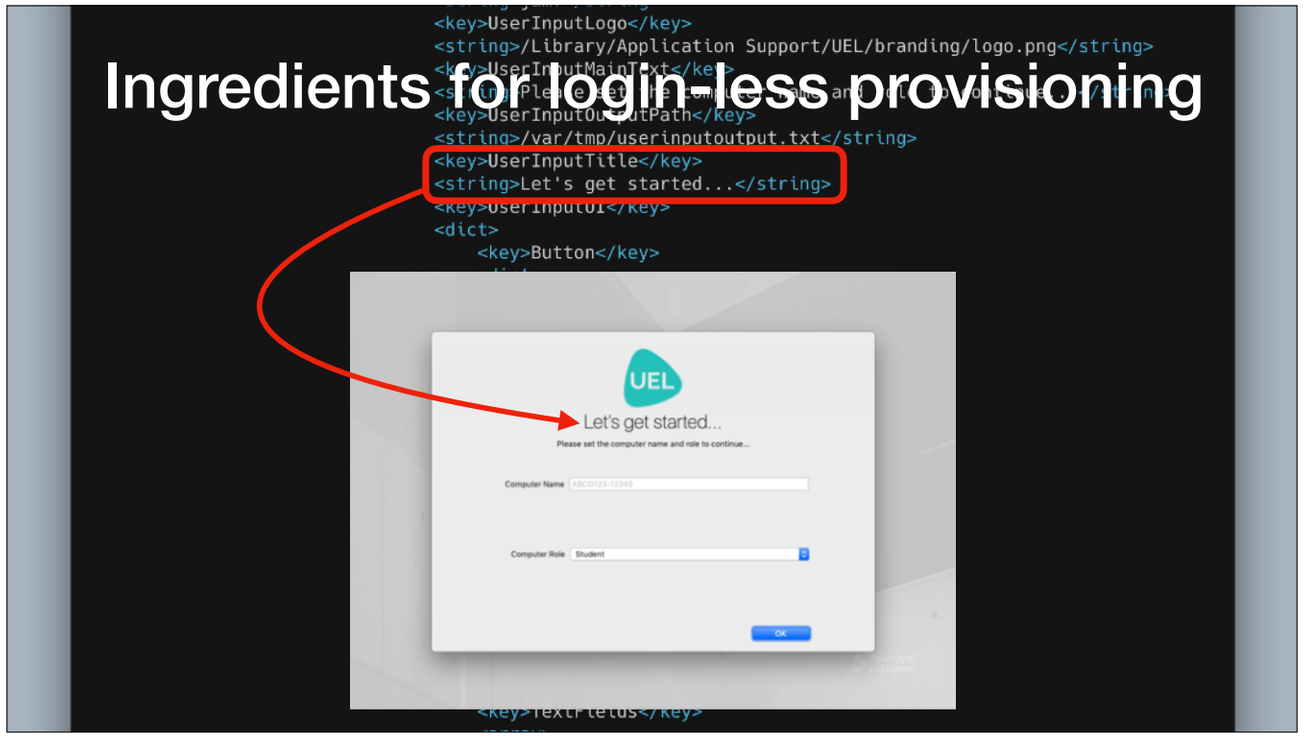
Then a block of main text - in this case we're providing some instructions to the person provisioning the Mac.

Ingredients for login-less provisioning

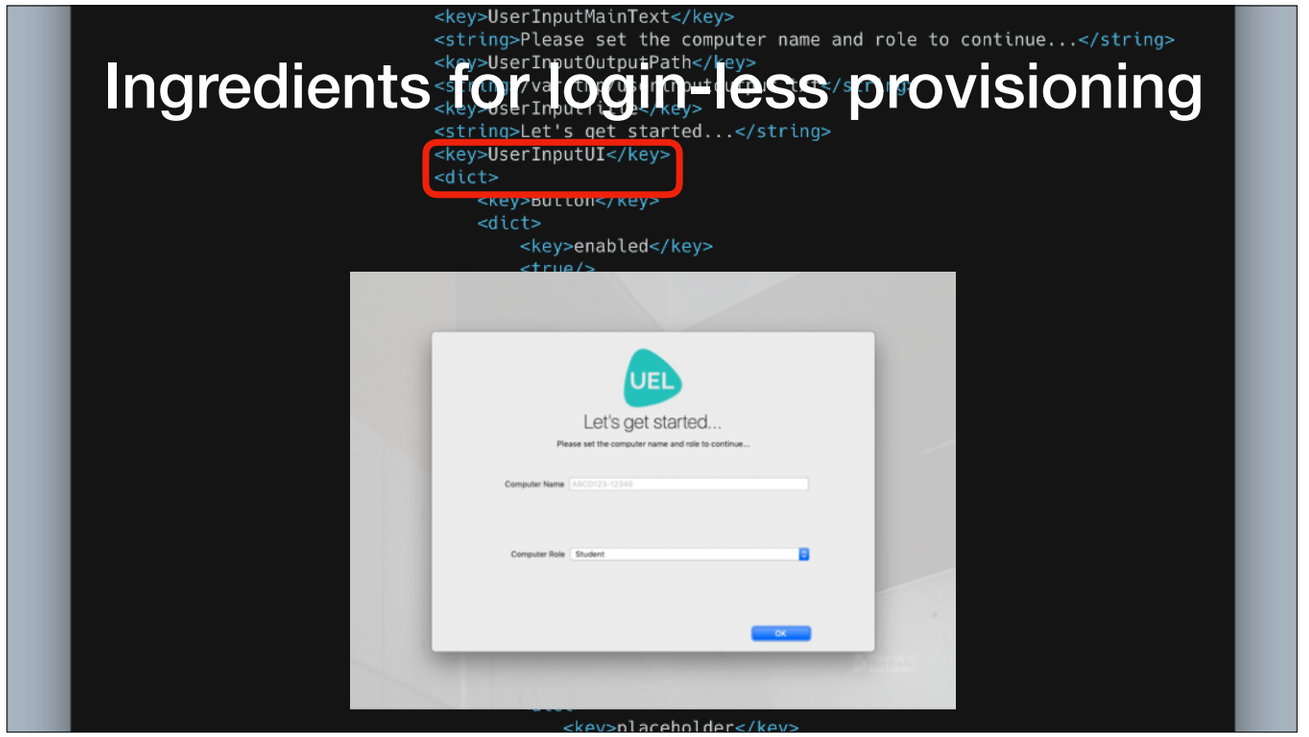


We tell User Input where it’s going to store the information our provisioner provides - this will be an XML property list.

Ingredients for login-less provisioning

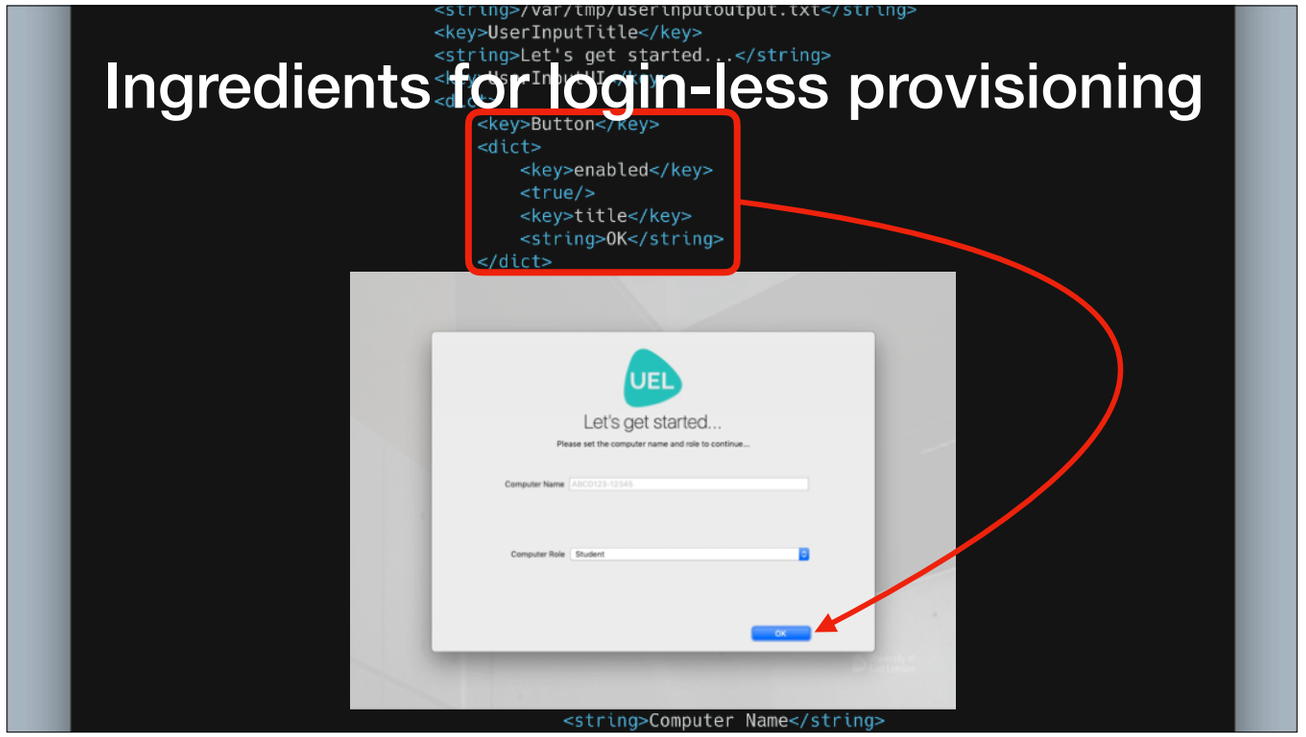


We give the User Input window a nice friendly title



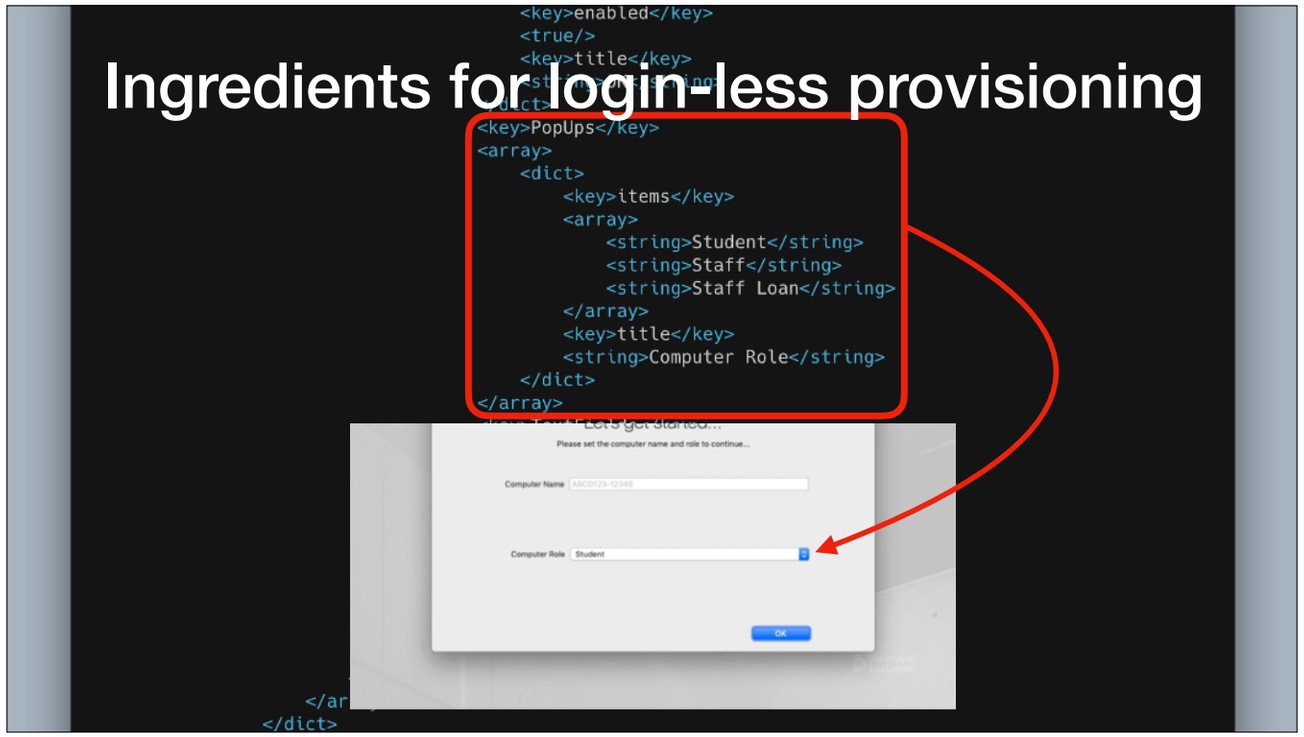
And then we need to define the different user interface elements we'll need. This is all done under the UserInputUI preference but it's fairly complicated because each group of elements is defined as a nested dictionary. I'll make this profile available so you can use it as a template. But also keep an eye on a tool called Profile Creator from Erik Berglund. It has a manifest for NoMAD Login which will be updated to have these new preferences, if it hasn't been already.

Ingredients for login-less provisioning



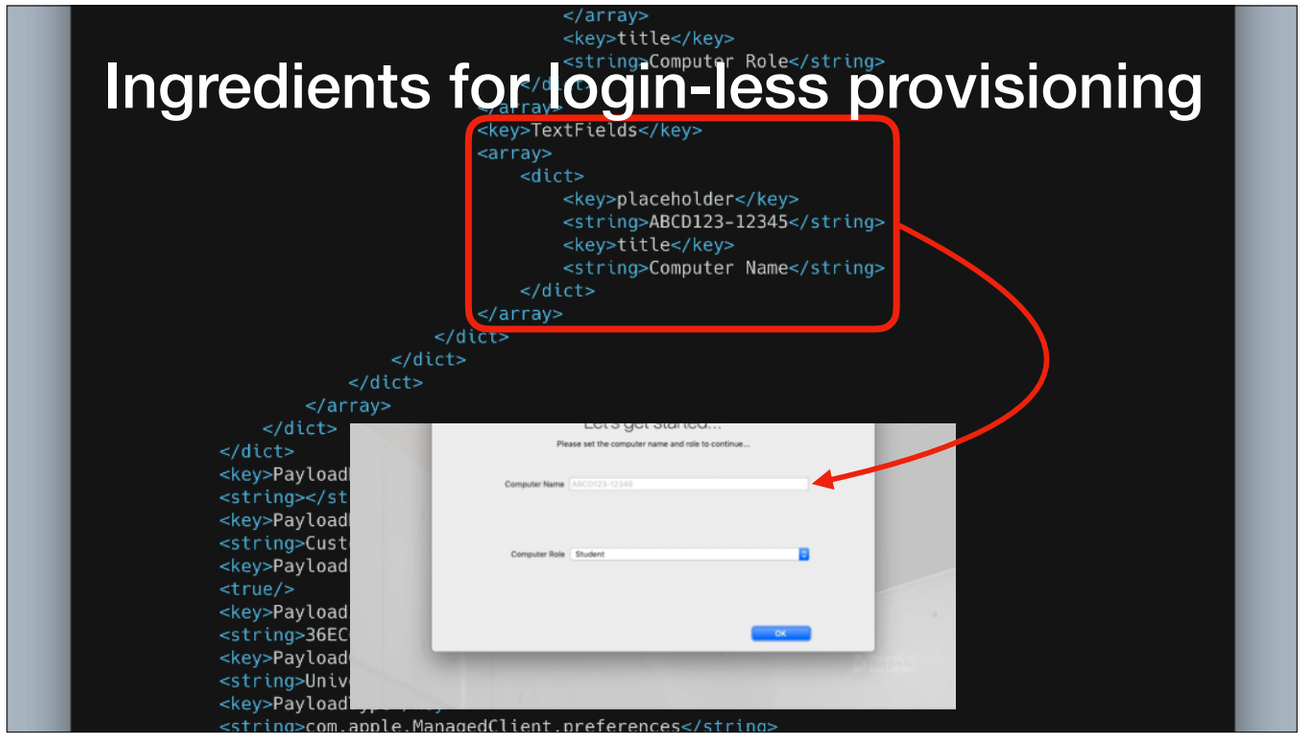
So, we want an OK button to click after we've entered all the information.

Ingredients for login-less provisioning

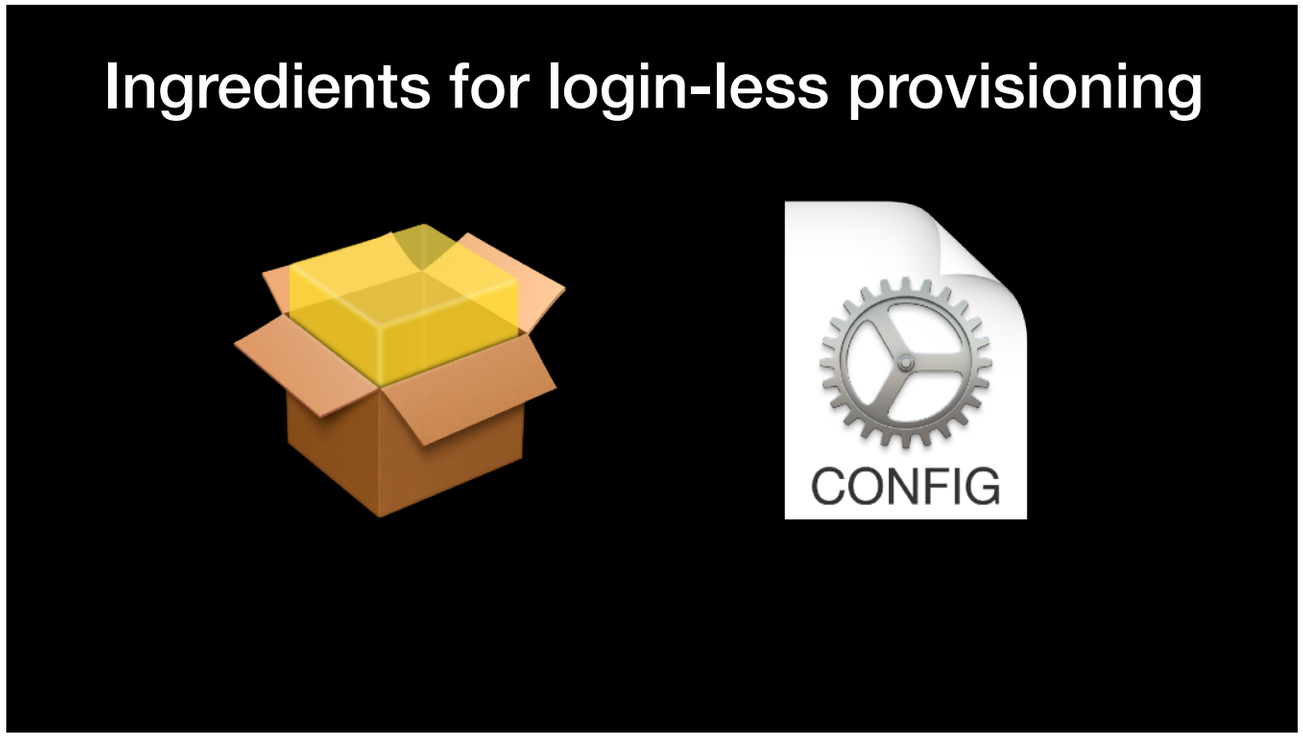


We need one drop-down menu which we'll call Computer Role - it'll have 3 choices; Student, Staff and Staff Loan.

Ingredients for login-less provisioning



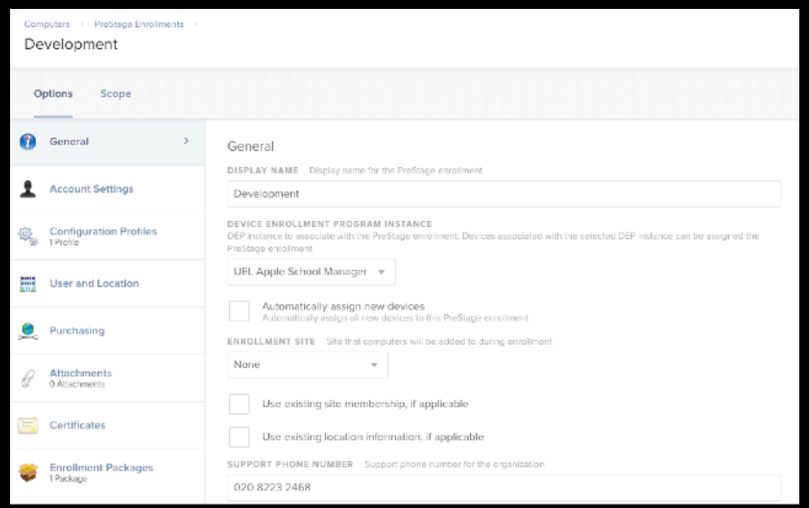
And we need one Text Field, called Computer Name, with some placeholder text just to remind the provisioner of our naming convention.



So we have our signed package with NoMAD Login plus our branding and postinstall script. We have a profile to set all the preferences we need. Let’s get them into our MDM solution - in my case that’s Jamf. So the rest of what I’m going to talk about will focus on Jamf as the management tool. But remember, the concepts can be achieved with other tools.

Ingredients for login-less provisioning

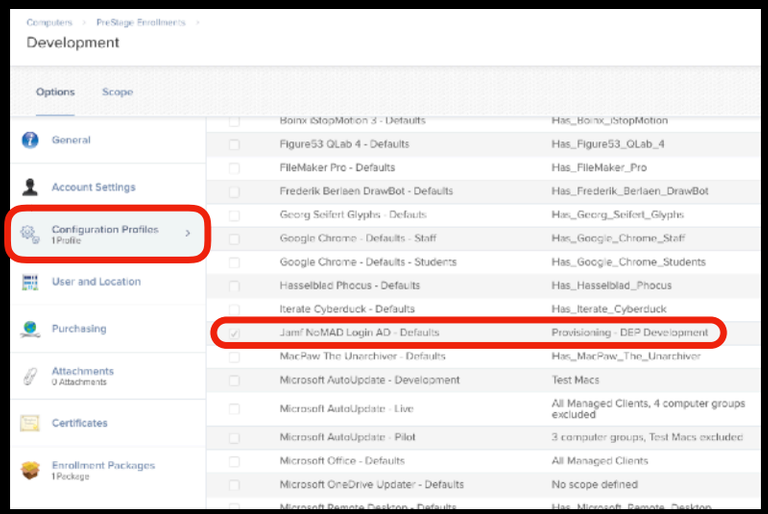
PreStage Enrolment



We need a PreStage Enrolment - to determine what’s going to happen when you hit that DEP Remote Management screen in the Setup Assistant.
And if you’re running the current version of Jamf, you’ll see some new options to add Configuration Profiles and packages so that they’re guaranteed to be installed before the Setup Assistant finishes.

Ingredients for login-less provisioning

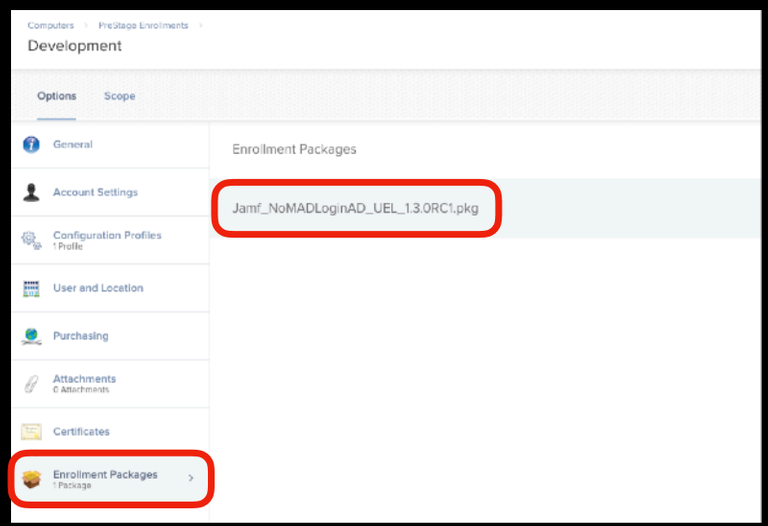
PreStage Enrolment



Add our Configuration Profile.

Ingredients for login-less provisioning

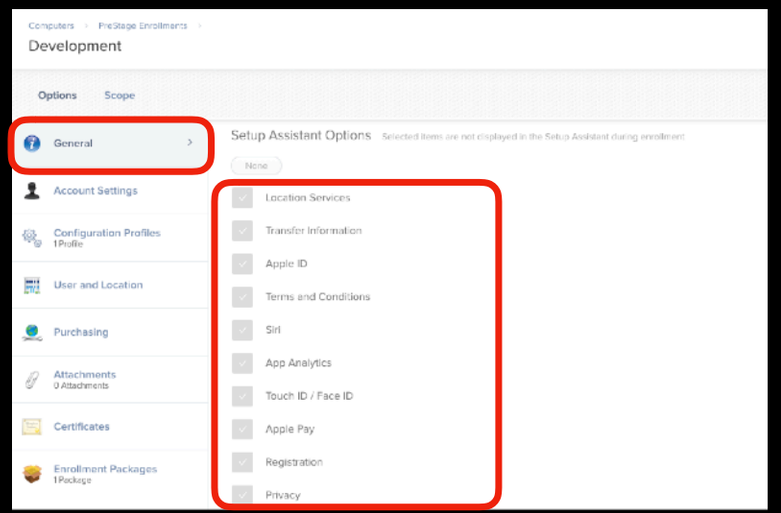
PreStage Enrolment



And our signed package to the Enrolment Packages payload. It’ll be delivered by an MDM command so that’s why it has to be signed - because Apple said so.

Ingredients for login-less provisioning

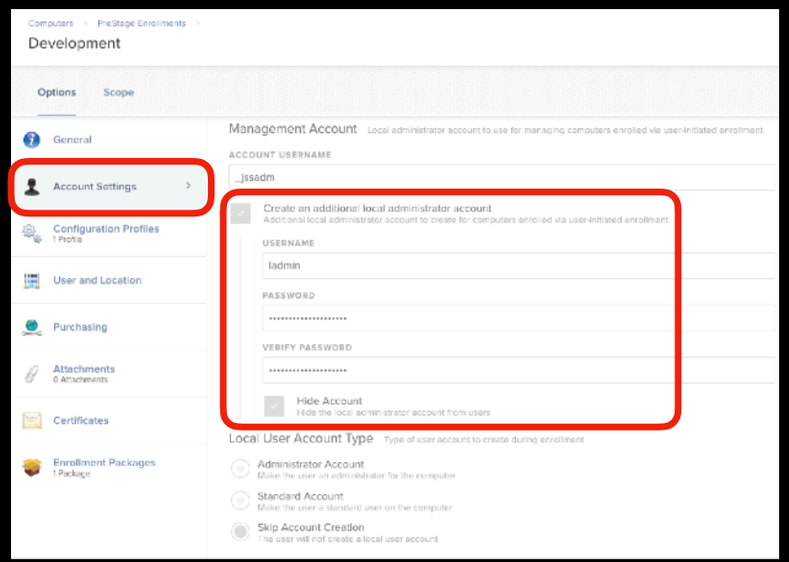
PreStage Enrolment



We want to skip as many parts of the Setup Assistant as we can.

Ingredients for login-less provisioning

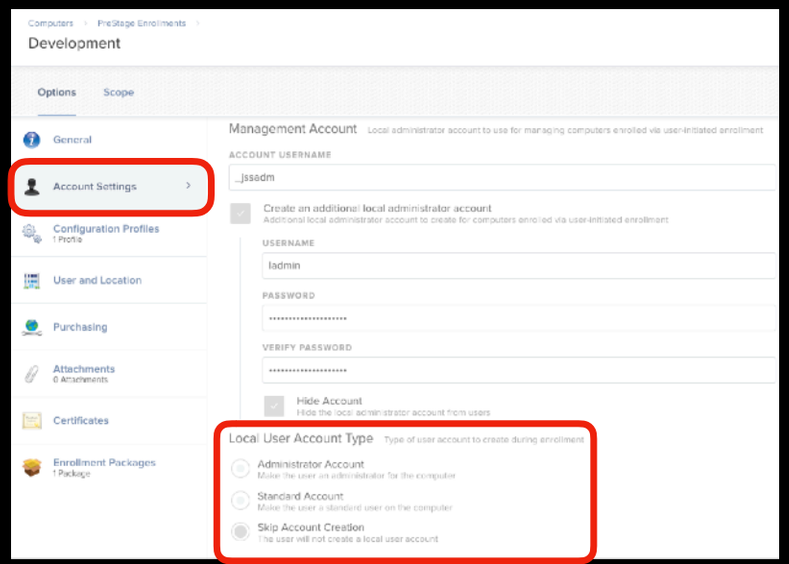
PreStage Enrolment



And have a hidden local administrator

Ingredients for login-less provisioning

PreStage Enrolment

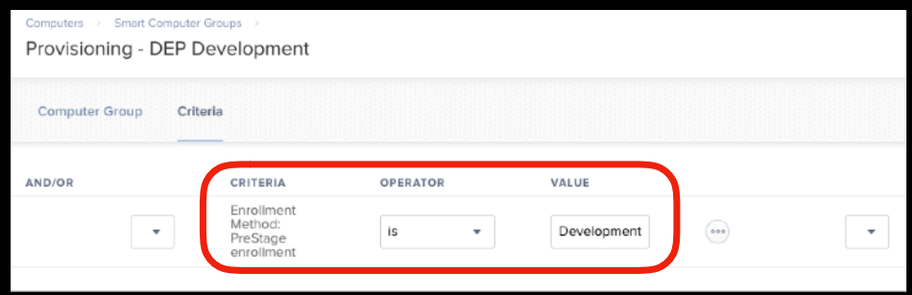


But most importantly, skip the part where the Setup Assistant asks you to create the first local user.

Ingredients for login-less provisioning

PreStage Enrolment

Smart Group



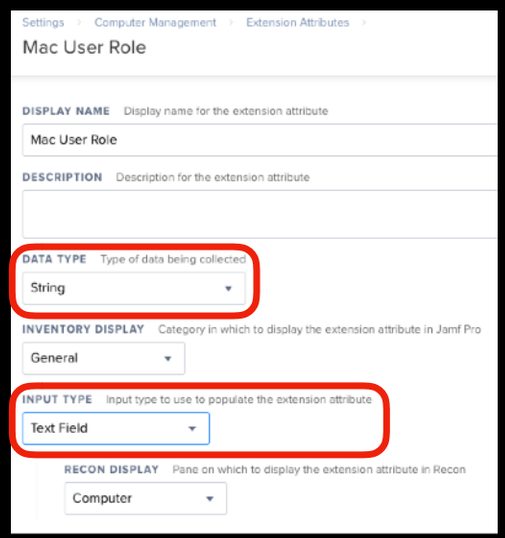
Next up, we'll want a Smart Group for all the Macs we assign to our PreStage Enrolment… Jamf has a criteria called Enrolment Method that we can use, which makes it pretty easy.

Ingredients for login-less provisioning

PreStage Enrolment

Smart Group

Extension Attributes



We need 2 Extension Attributes; one for each Mac’s hostname and the other for its role. These should have a data type of string and an input type of text field.

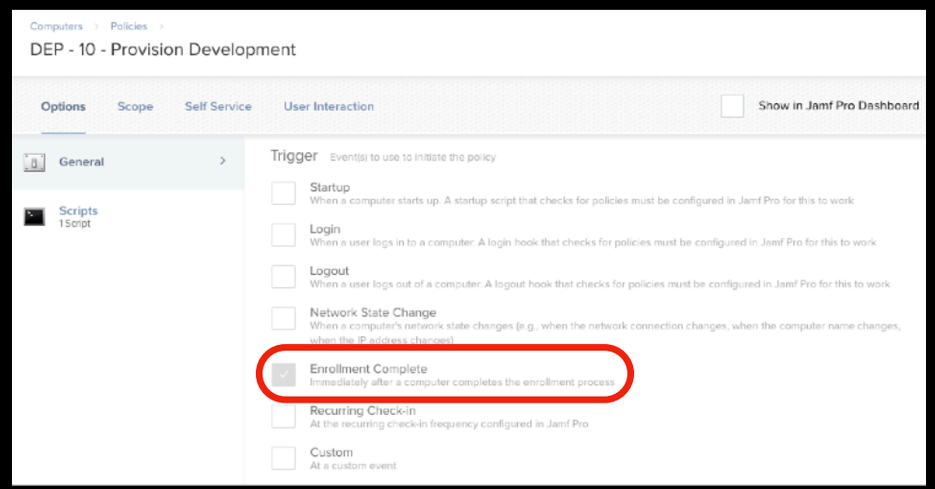
Ingredients for login-less provisioning

PreStage Enrolment

Smart Group

Extension Attributes

Policy



And finally, we want a policy to run as soon as the Mac is enrolled and the Jamf binary is installed.

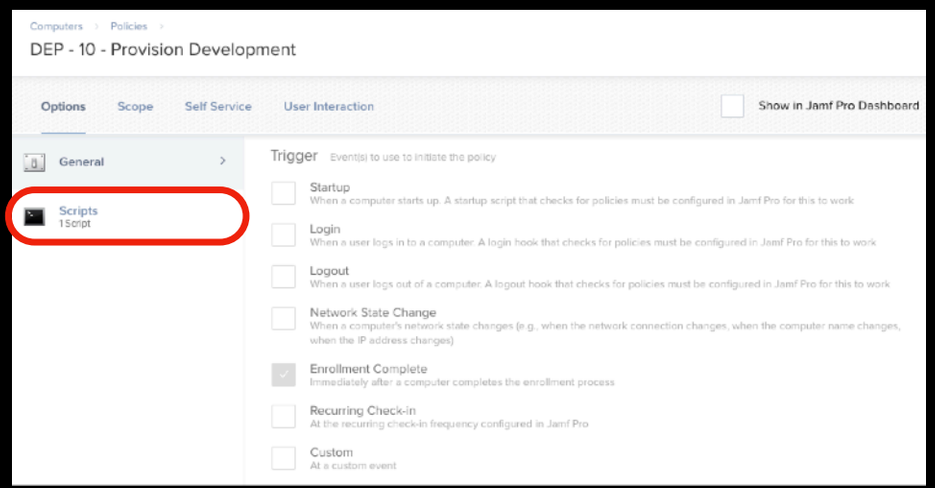
Ingredients for login-less provisioning

PreStage Enrolment

Smart Group

Extension Attributes

Policy



All this policy will do is run one script - which I'll come to in a minute.

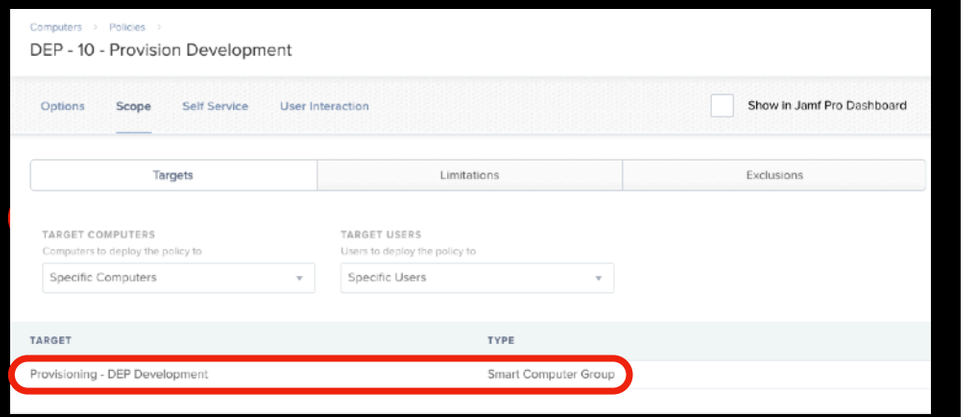
Ingredients for login-less provisioning

PreStage Enrolment

Smart Group

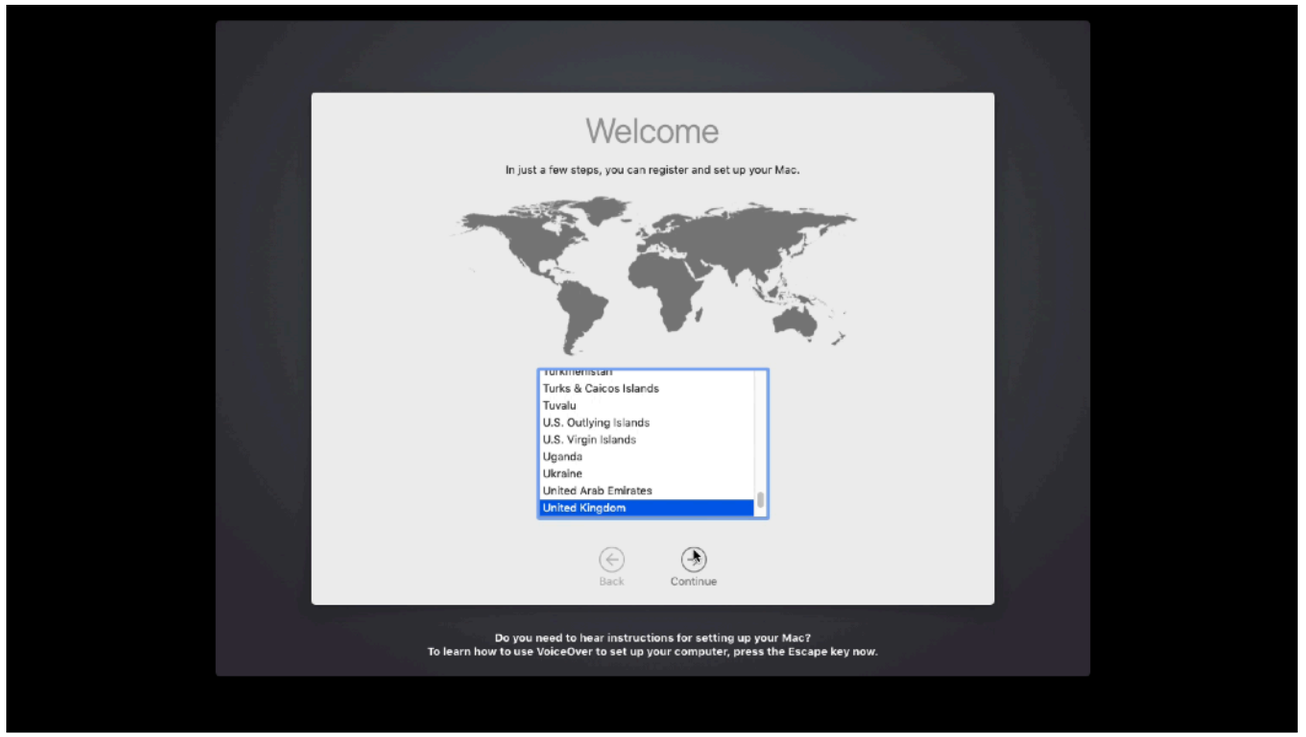
Extension Attributes

Policy



And we scope it so that it’ll run on any Mac in the smart group we created before, so that’s any Mac that’s assigned to our new PreStage Enrolment.

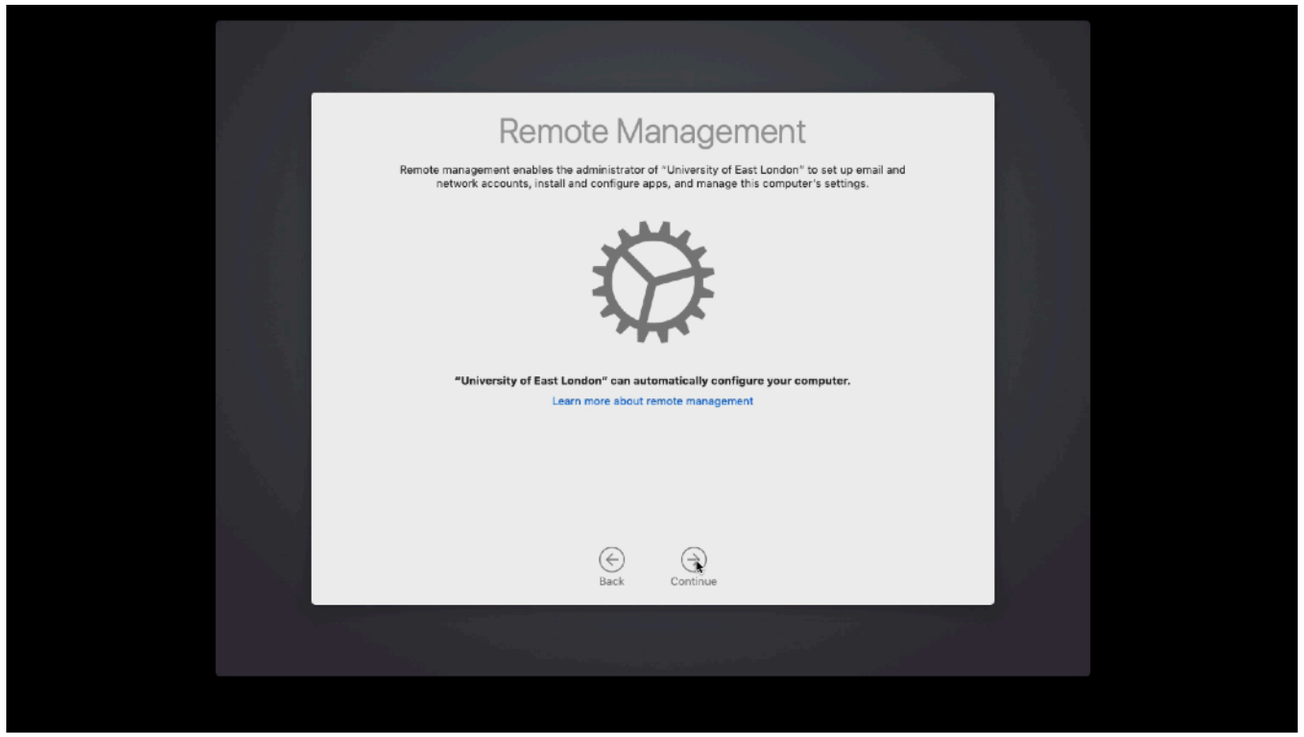
What happens?



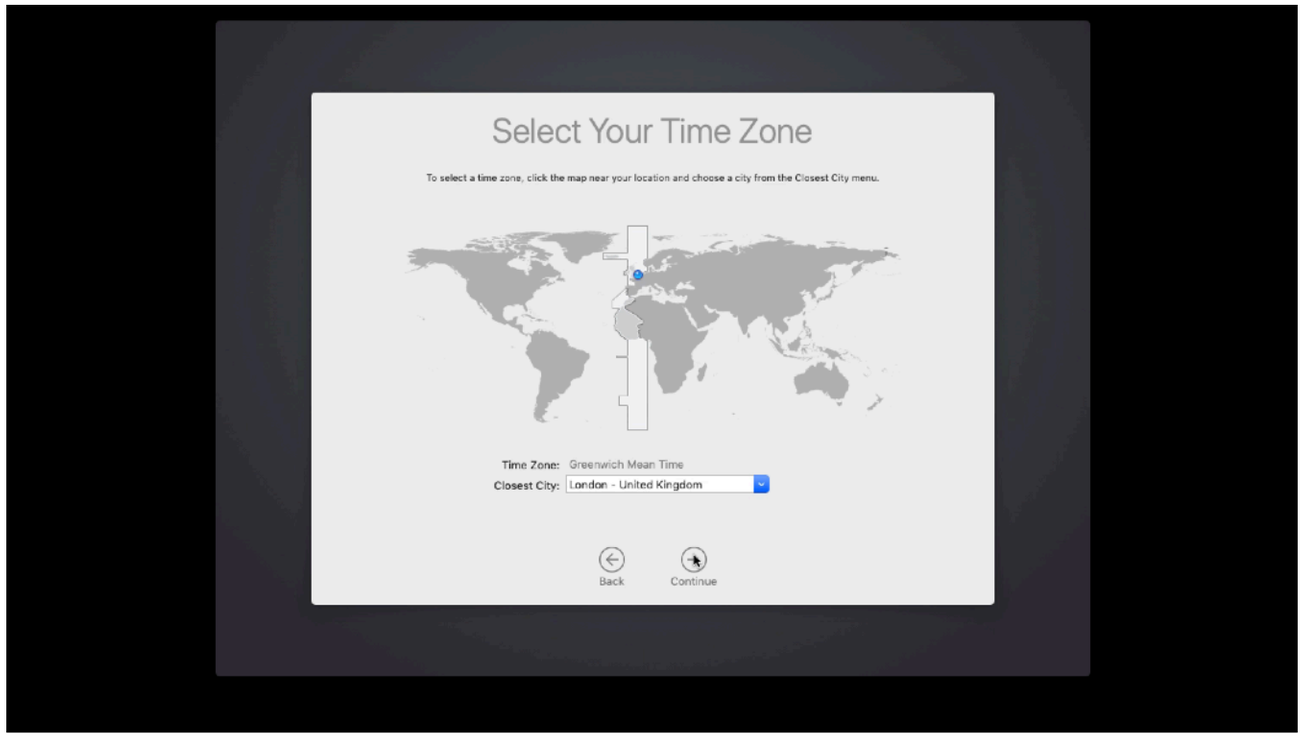
You’ve installed macOS and we’re at the Setup Assistant

Choose your country

And your keyboard layout

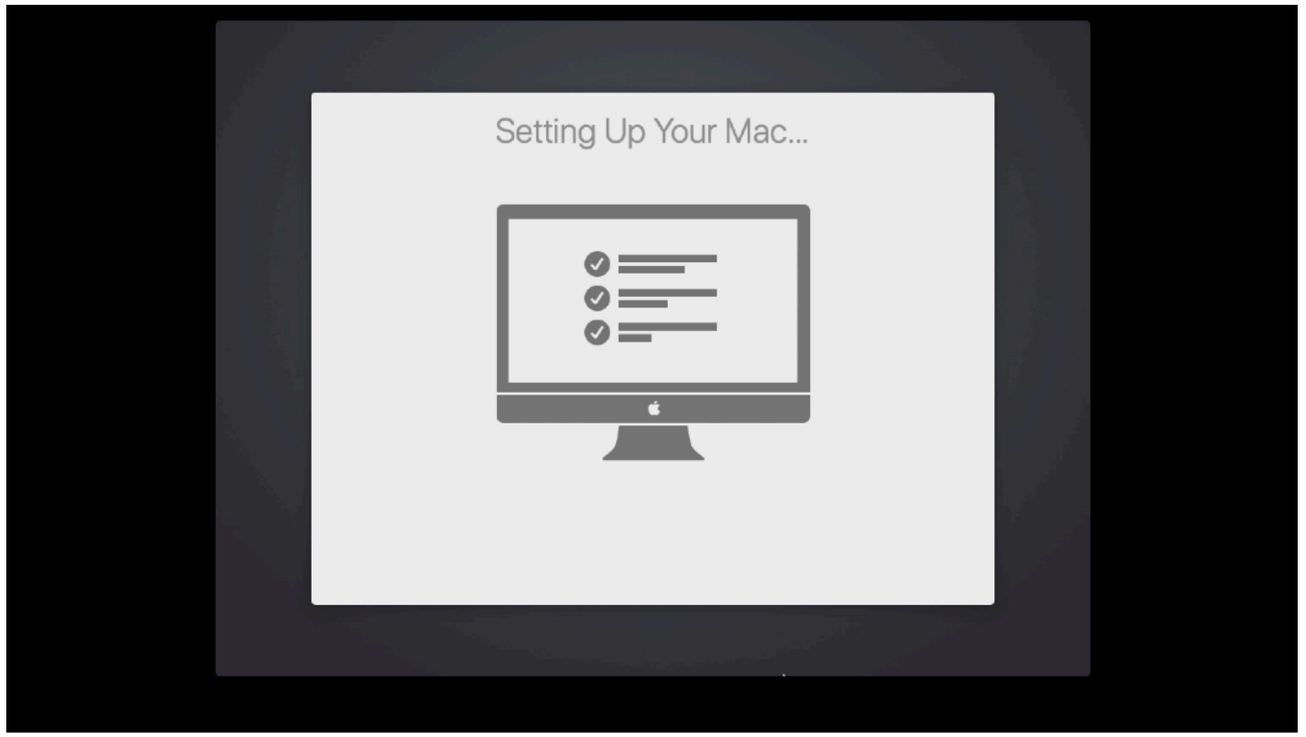


We hit the Remote Management screen because our Mac is assigned to Apple School Manager and our PreStage in Jamf.



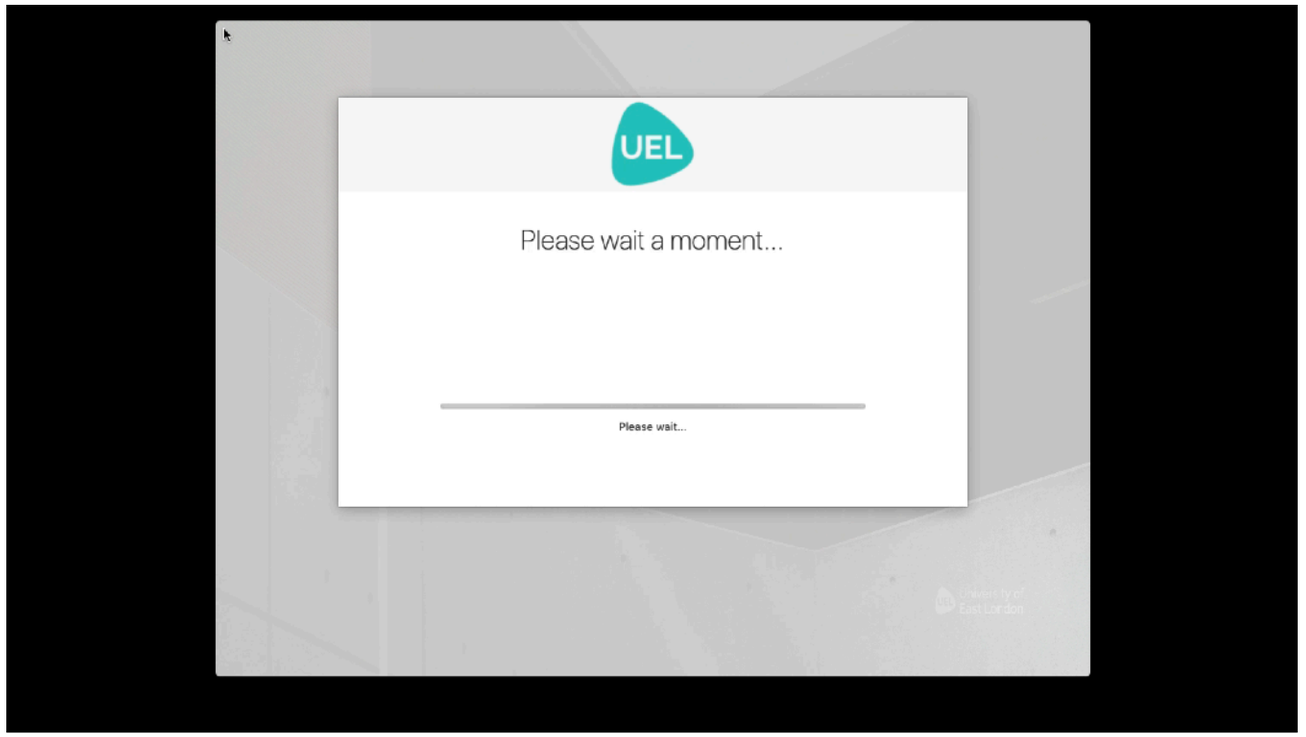
Now at this point, some things start to happen in the background. Our Mac enrolls into the MDM part of Jamf.

Along with the Jamf binary and framework, our package with NoMAD Login and the configuration profile we defined in our Prestage are delivered now.
The important thing to remember is that the Setup Assistant won’t finish until that package and profile are installed.



The Setup Assistant finishes… and remember, our package's postinstall script has done its thing and used authchanger to have the Notify part of NoMAD Login replace Apple's login window…

Drumroll please….

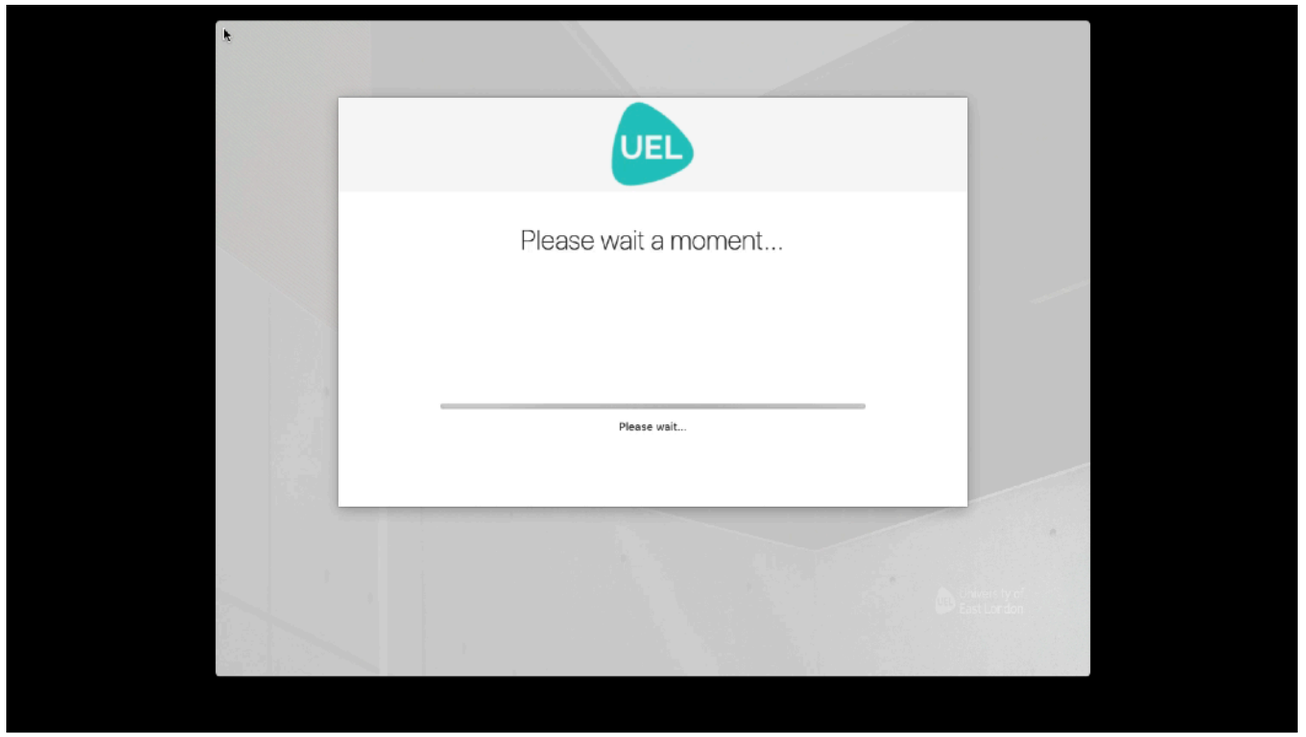


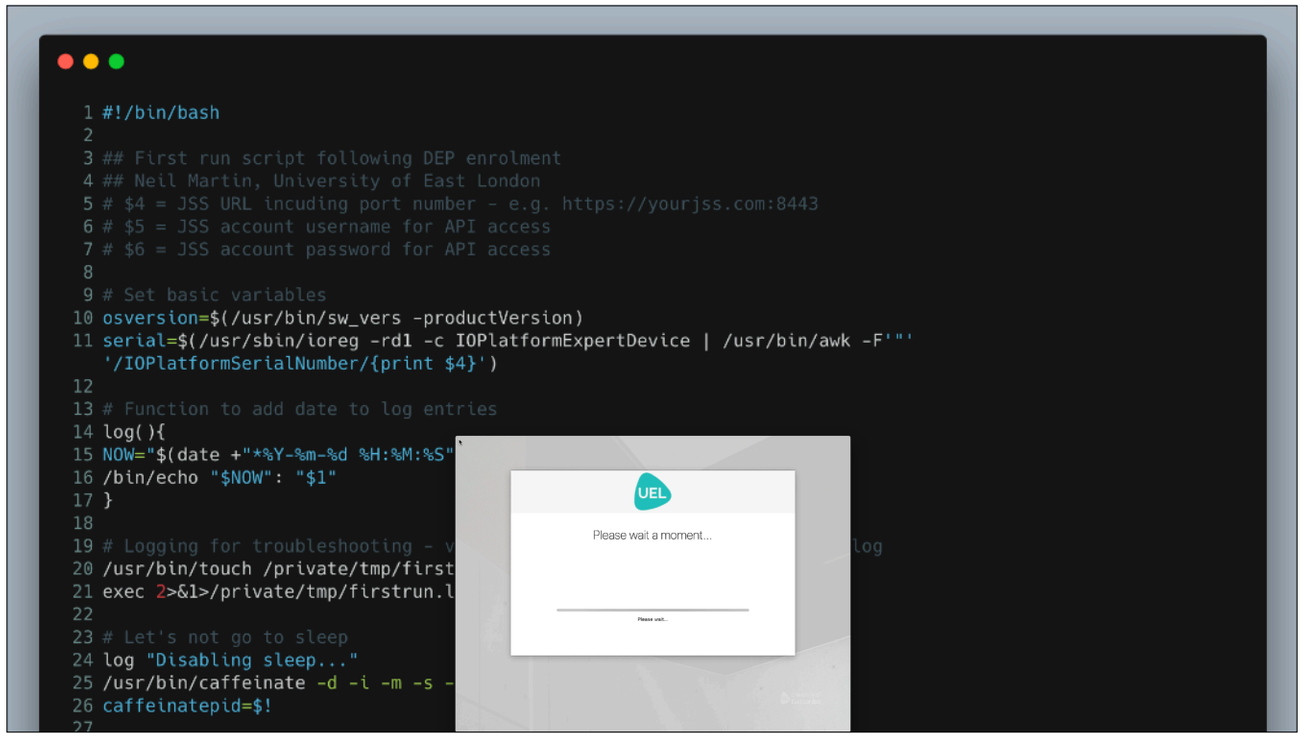
And this is what we see! Now, it’s up to us to drive the journey forward.

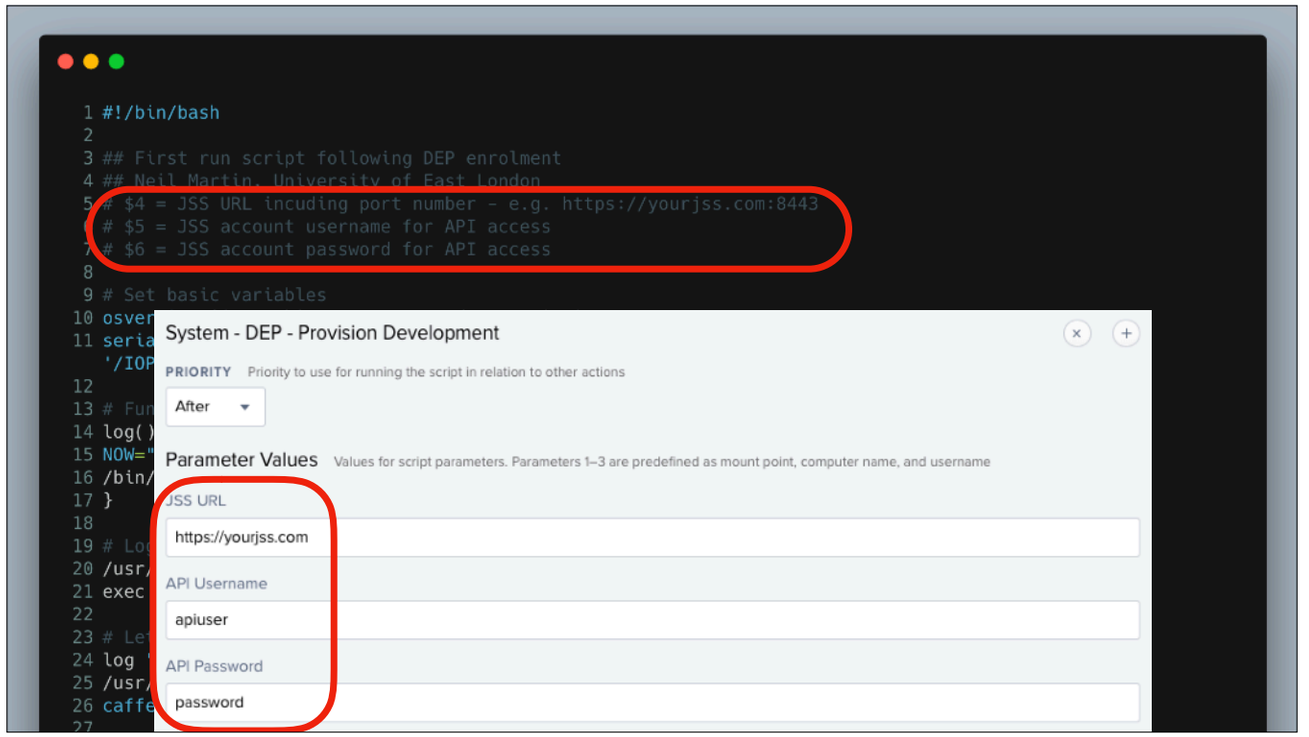
In my case, Jamf will execute a policy with its enrolmentcomplete trigger and that policy will run a script that will do the rest.

I’ll also mention that NoMAD Login has a run script mechanism so it can execute your provisioning script and drive the process instead.

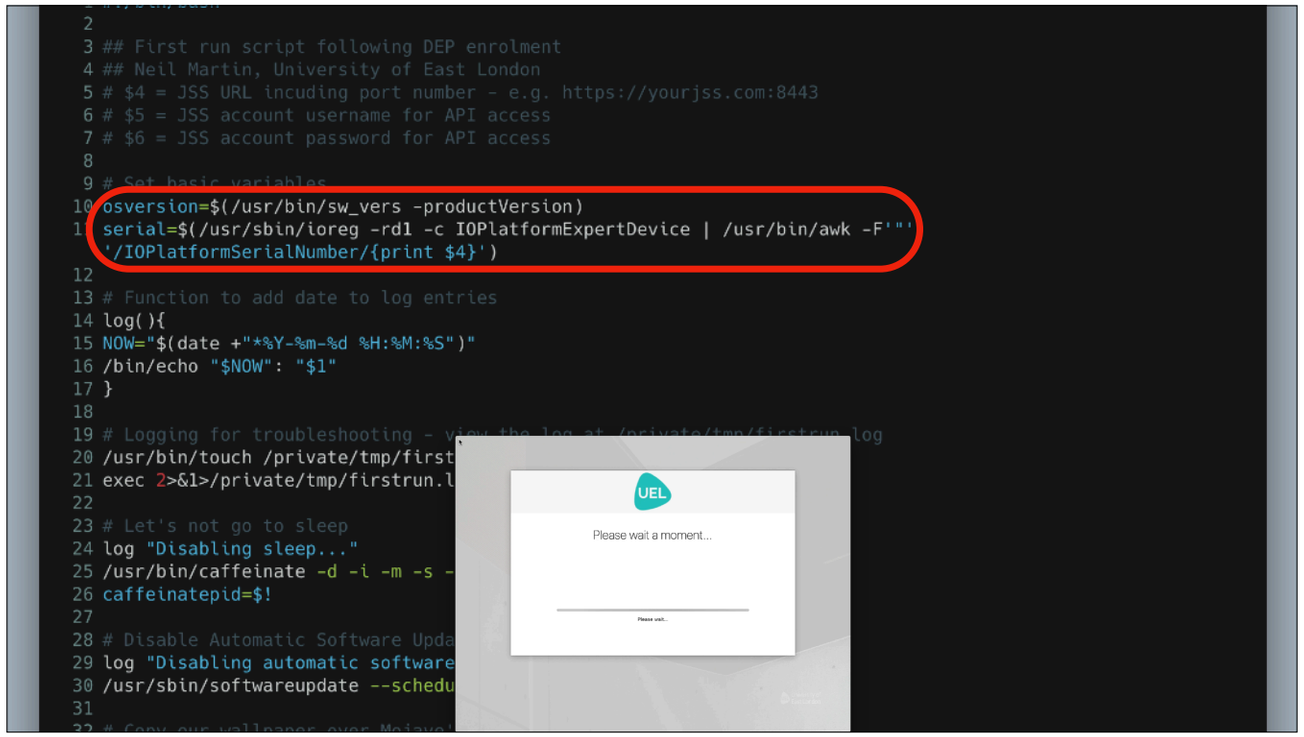
It’s time for some Bash! We all love a bit of Bash, right? Thank you to the wonderful Steve Quirke for this picture!


UEL
Please wait a moment...
Please wait...

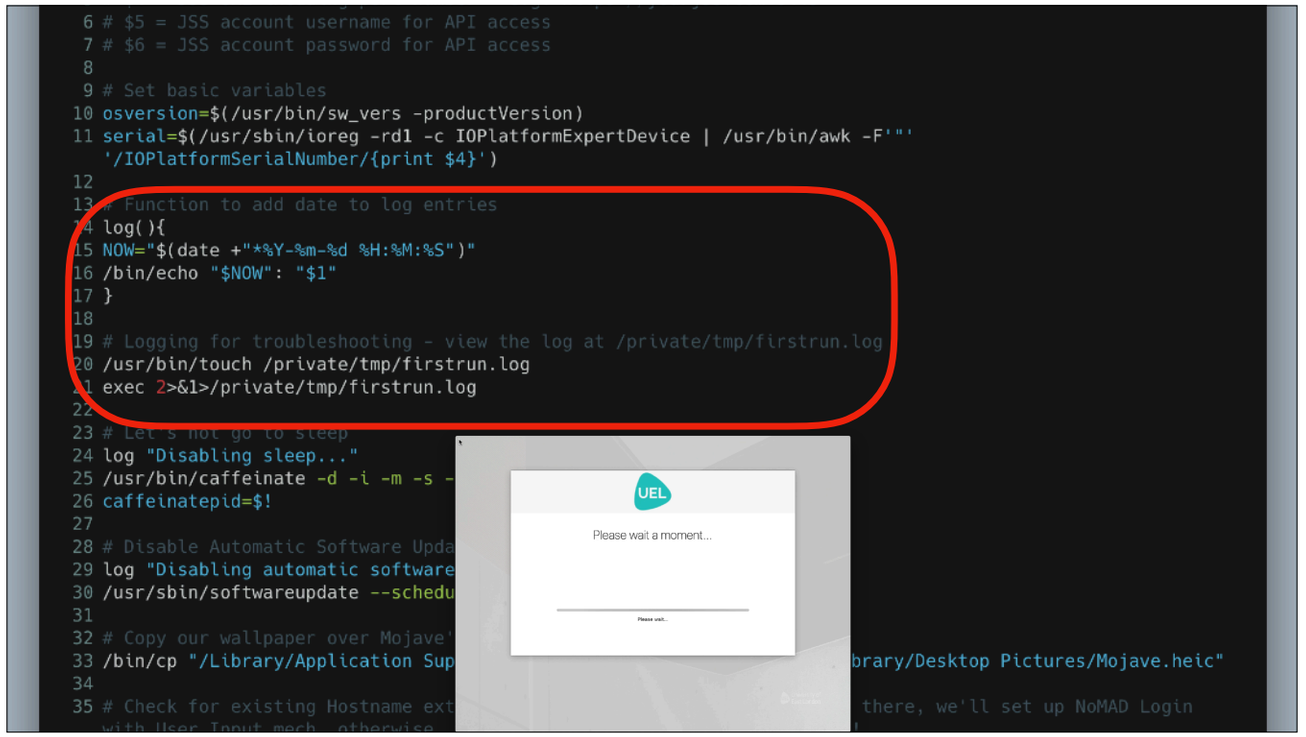

#!/bin/bash
First run script following DEP enrolment
Neil Martin, University of East London
# $4 = JSS URL incuding port number - e.g. https://yourjss.com:8443
# $5 = JSS account username for API access
# $6 = JSS account password for API access
# Set basic variables
osversion=$(/usr/bin/sw_vers -productVersion)
serial=$(/usr/sbin/ioreg -rd1 -c IOPlatformExpertDevice | /usr/bin/awk -F'"' '/IOPlatformSerialNumber/{print $4}')
# Function to add date to log entries
log(){
NOW="$(date +"*%Y-%m-%d %H:%M:%S"
/bin/echo "$NOW": "$1"
}
# Logging for troubleshooting - v
log
/usr/bin/touch /private/tmp/first
exec 2>&1>/private/tmp/firstrun.l
Let's not go to sleep
log "Disabling sleep..."
/usr/bin/caffeinate -d -i -m -s -
caffeinatepid=$!
UEL
Please wait a moment...



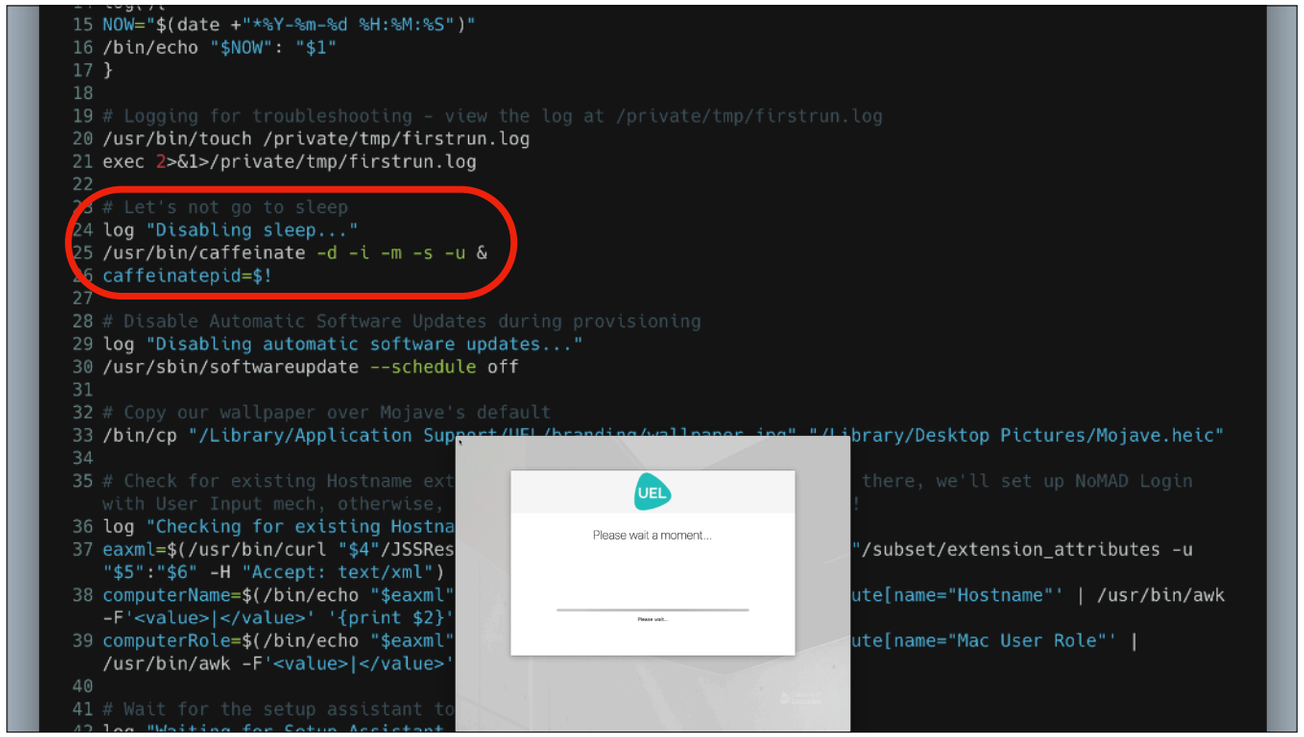
First things first, we will be using a little bit of the Jamf API so the script will need to send credentials to our Jamf Server for the account we're using to access it. Security is important, so, instead of storing these in the script in plain text, we'll add them as parameters in the policy that executes the script itself. Parameter 4 is the JSS URL, 5 is the username of the API account and 6 is its password.



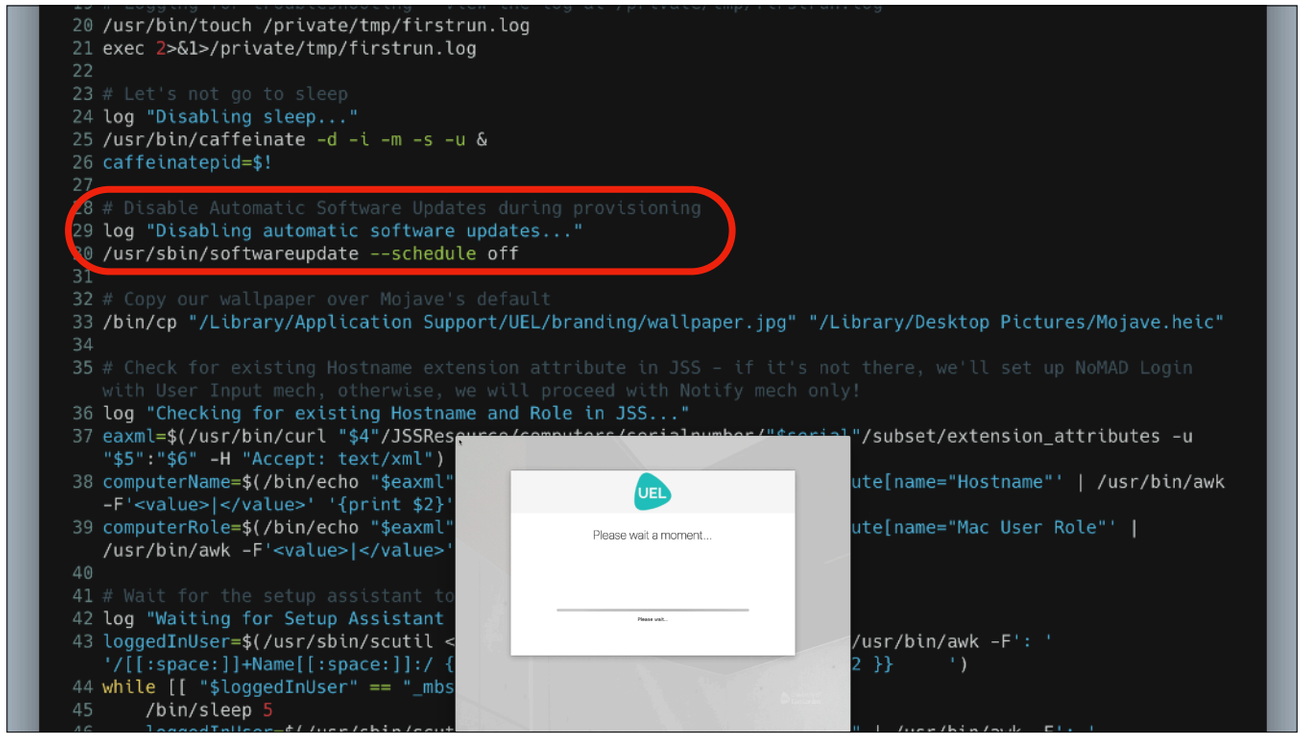
Next up, we'll make a couple of variables. We want the version number of macOS that's installed so we can feed it back through NoMAD Login's notify screen. And we want the Mac's serial number because we'll need it when we make API calls later.



We'll create a function called log - which we'll use to add entries to a log file in case we need to troubleshoot.

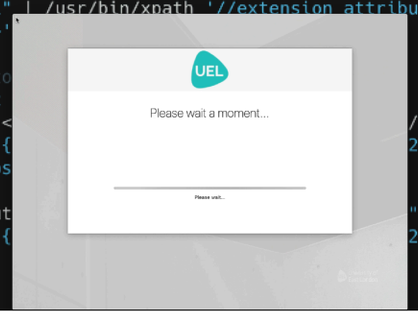


We don’t want the Mac to sleep, so we deal with that.

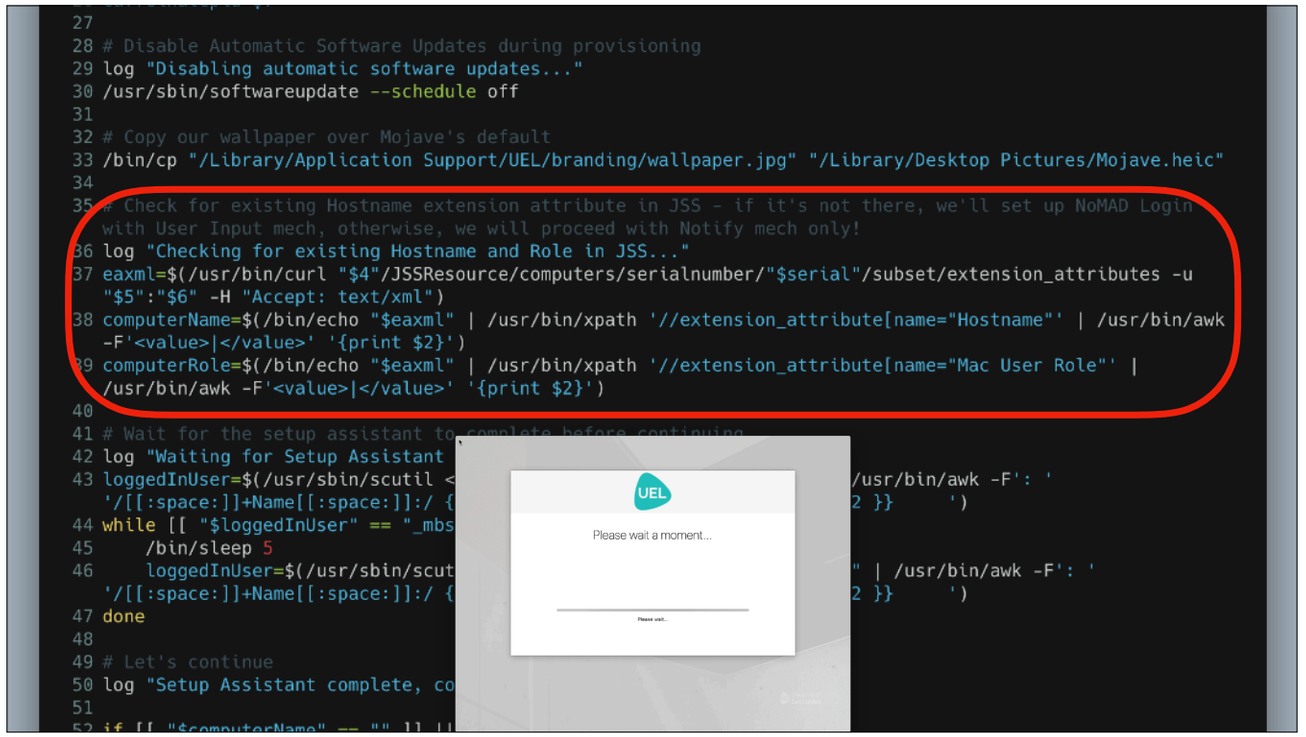


We nip the built-in software update checks in the bud because we manage this with Jamf anyway.

```
24 log "Disabling sleep..."
25 /usr/bin/caffeinate -d -i -m -s -u &
26 caffeinatepid=$!
27
28 # Disable Automatic Software Updates during provisioning
29 log "Disabling automatic software updates..."
30 /usr/sbin/softwareupdate --schedule off
31
32 # Copy our wallpaper over Mojave's default
33 /bin/cp "/Library/Application Support/UEL/branding/wallpaper.jpg" "/Library/Desktop Pictures/Mojave.heic"
34
35 # Check for existing Hostname extension attribute in JSS - if it's not there, we'll set up NoMAD Login
   with User Input mech, otherwise, we will proceed with Notify mech only!
36 log "Checking for existing Hostname and Role in JSS..."
37 eaxml=$(/usr/bin/curl "$4"/JSSResource/computers/serialnumber/"$serial"/subset/extension_attributes -u
   "$5":"$6" -H "Accept: text/xml")
38 computerName=$(/bin/echo "$eaxml" | /usr/bin/xpath '//extension_attribute[name="Hostname"' | /usr/bin/awk
   -F'<value>|</value>' '{print $2}')
39 computerRole=$(/bin/echo "$eaxml" | /usr/bin/xpath '//extension_attribute[name="Mac User Role"' |
   /usr/bin/awk -F'<value>|</value>'
40
41 # Wait for the setup assistant to
42 log "Waiting for Setup Assistant
43 loggedInUser=$(/usr/sbin/scutil <                                    /usr/bin/awk -F': '
   '/[[:space:]]+Name[[:space:]]:/ {                                    2 }}     ')
44 while [[ "$loggedInUser" == "_mbs
45     /bin/sleep 5
46     loggedInUser=$(/usr/sbin/scut                                    " | /usr/bin/awk -F': '
   '/[[:space:]]+Name[[:space:]]:/ {                                    2 }}     ')
47 done
48
49 # Let's continue
```



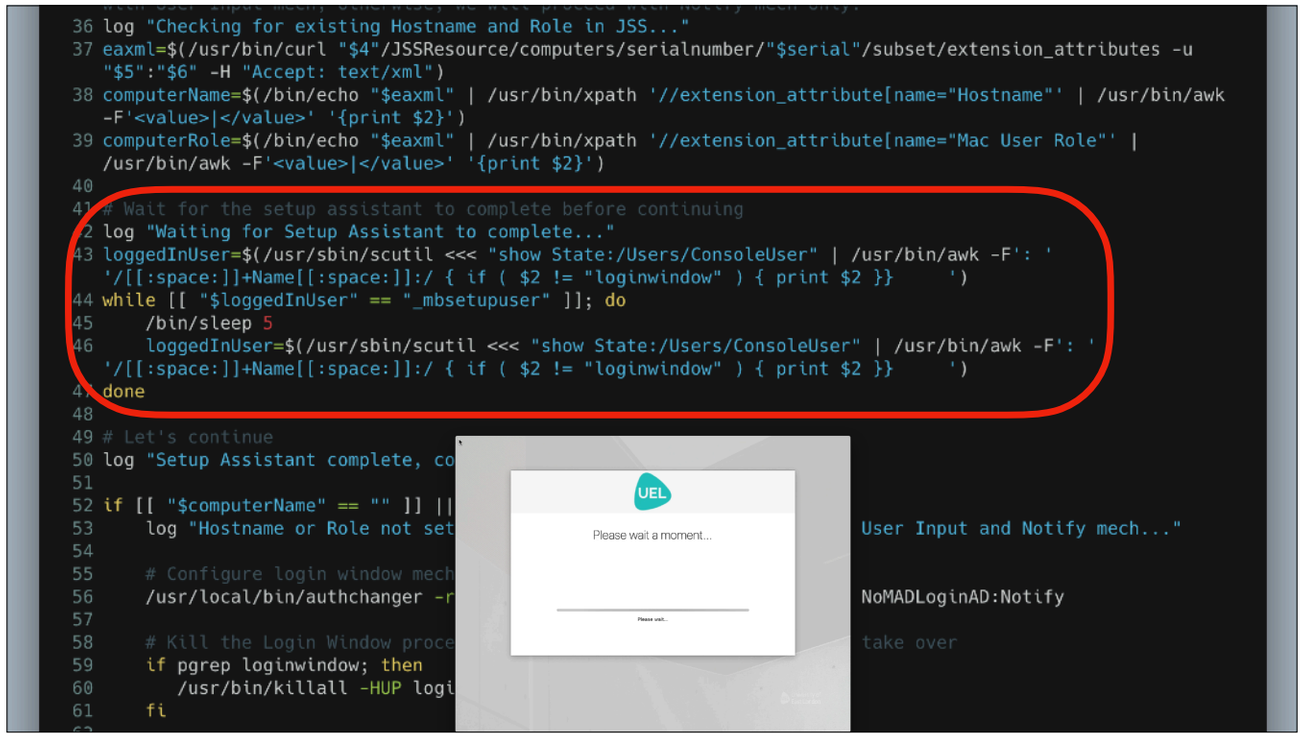
We want the login window to have our branding, so let’s do a quick and dirty copy of our own wallpaper over Mojave’s default one



And let’s do an API read to retrieve all the Extension Attributes from that Mac’s computer record in Jamf, dump it as XML to a variable and extract the value of the Name and Role attributes, feeding those to another pair of variables.

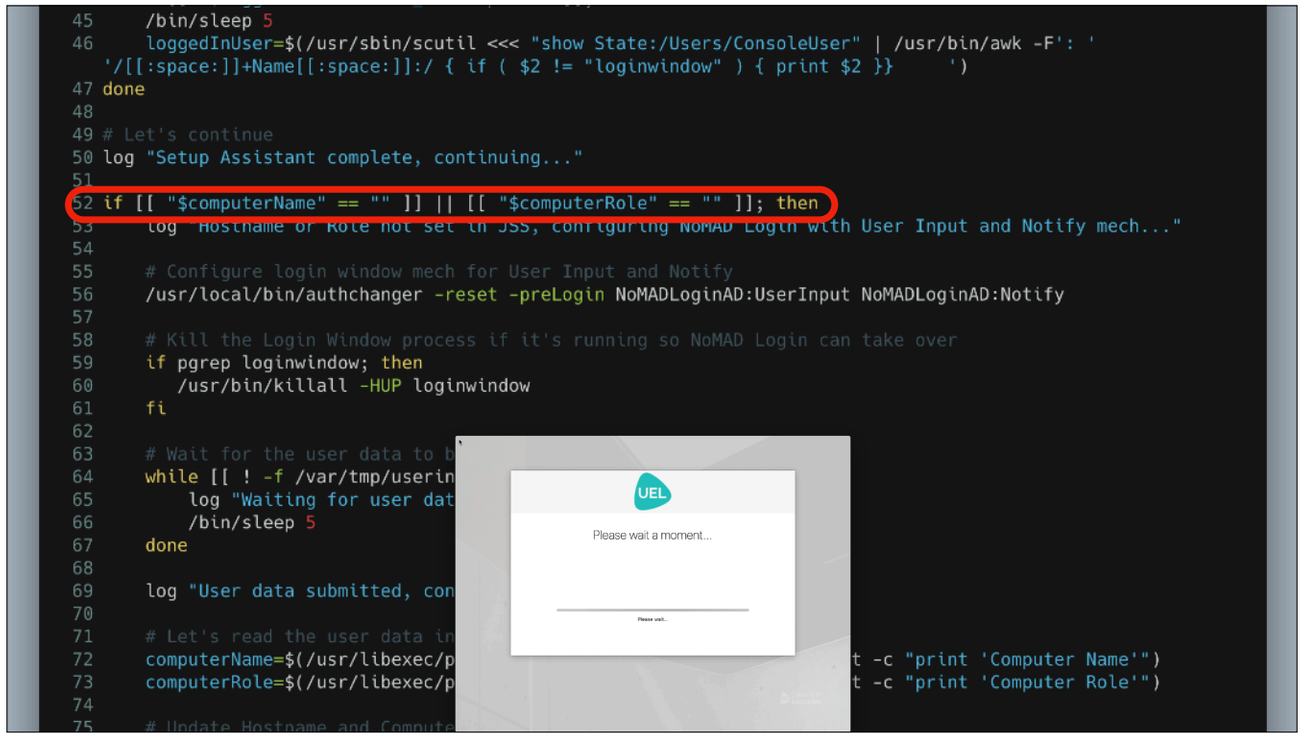
These variables will only be populated if we’re re-provisioning an existing Mac where its old computer record is still in Jamf. In theory it should also work if these attributes have been set using Jamf’s new inventory preload feature - but I haven’t tried that yet.

If it’s a new Mac, or we've deleted its computer record before we started, these variables will be empty.

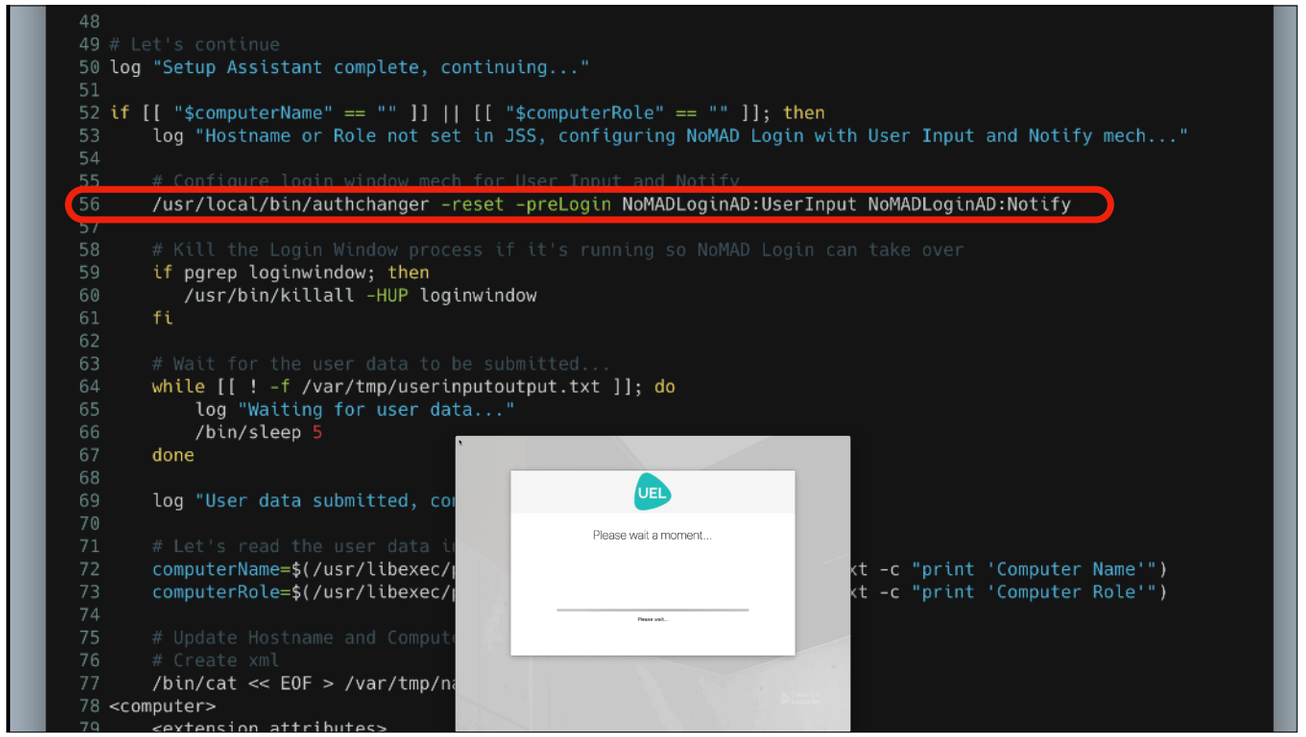


Before we continue, we need to check that the Setup Assistant isn't running. That's because policies that run with Jamf's enrolmentcomplete trigger can actually start before the Setup Assistant finishes, if you don't advance through all the steps quickly enough.

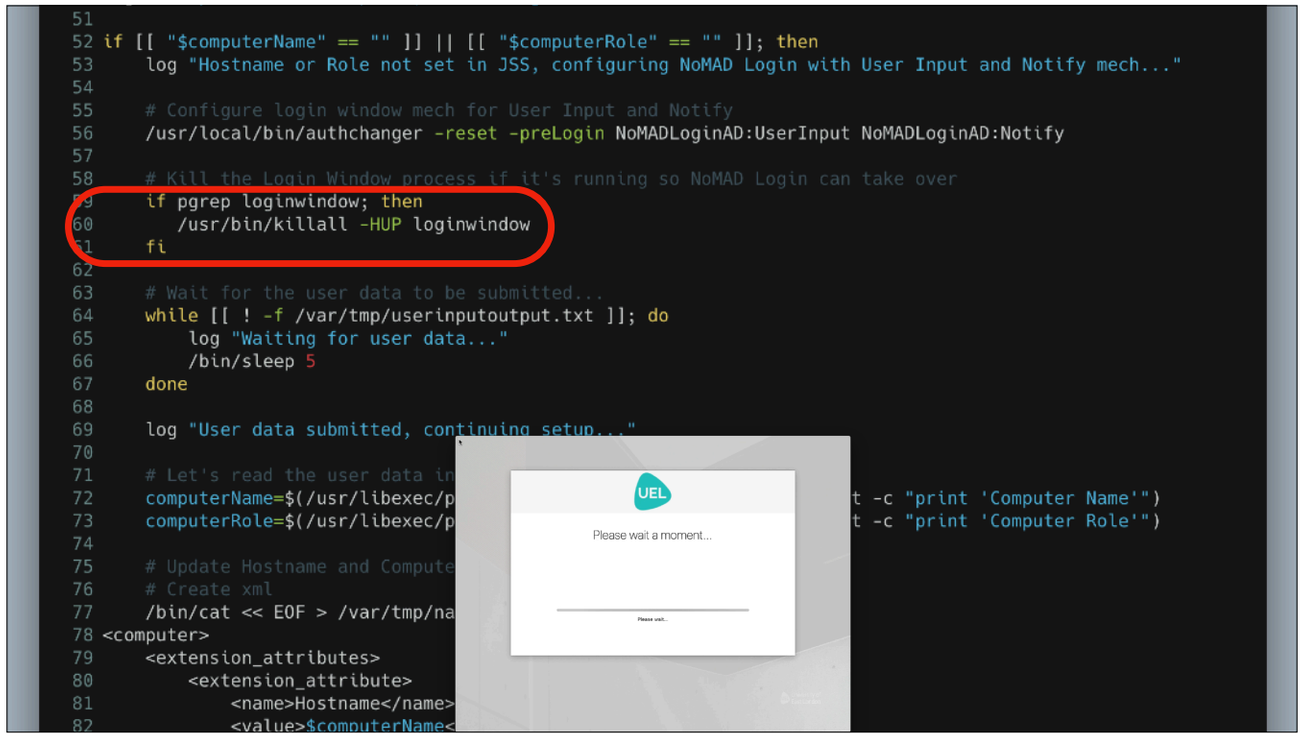
We can keep ourselves trapped in a while loop, checking the logged in username every 5 seconds - which will be _mbsetupuser if the Setup Assistant is running. If the Setup Assistant isn't running, the logged in username will be empty and we'll carry on.



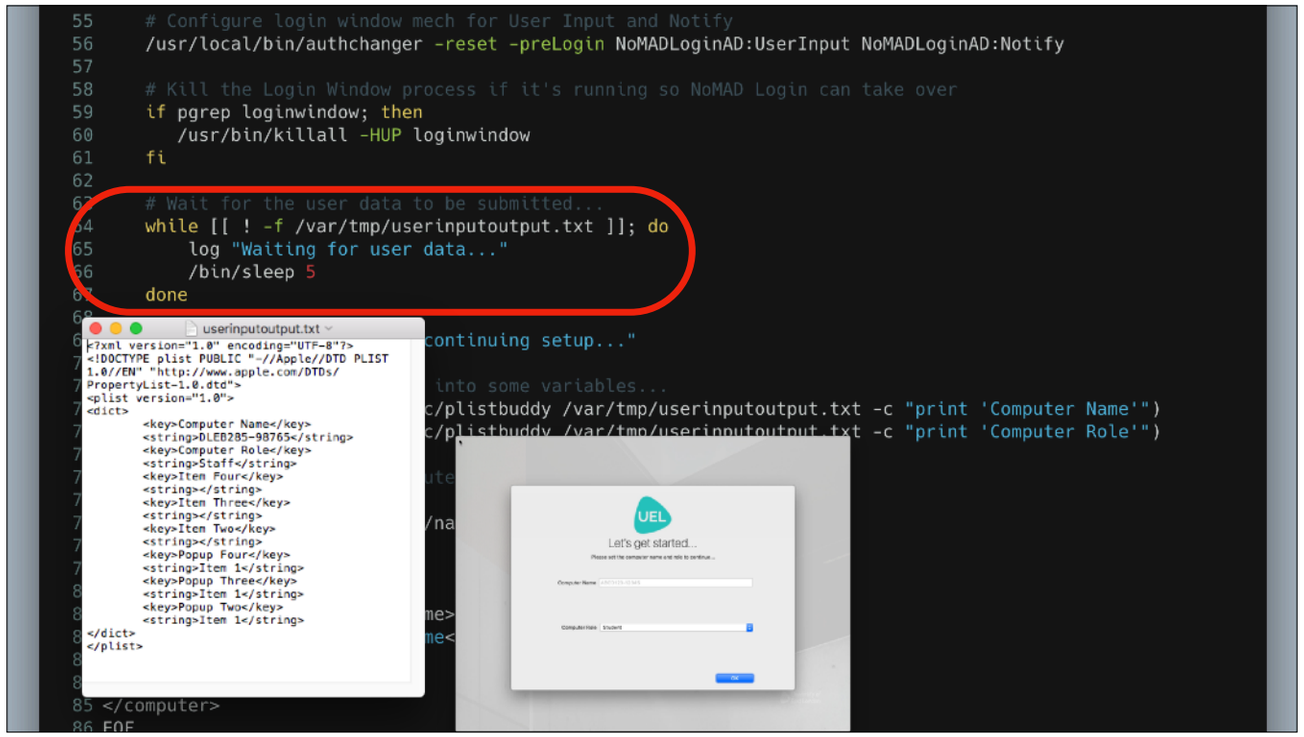
After this, we'll check to see if the Computer Name and User Role values we collected earlier are empty. If they are, we need to get some user input from the person who's provisioning the Mac. Let's assume that's the case.



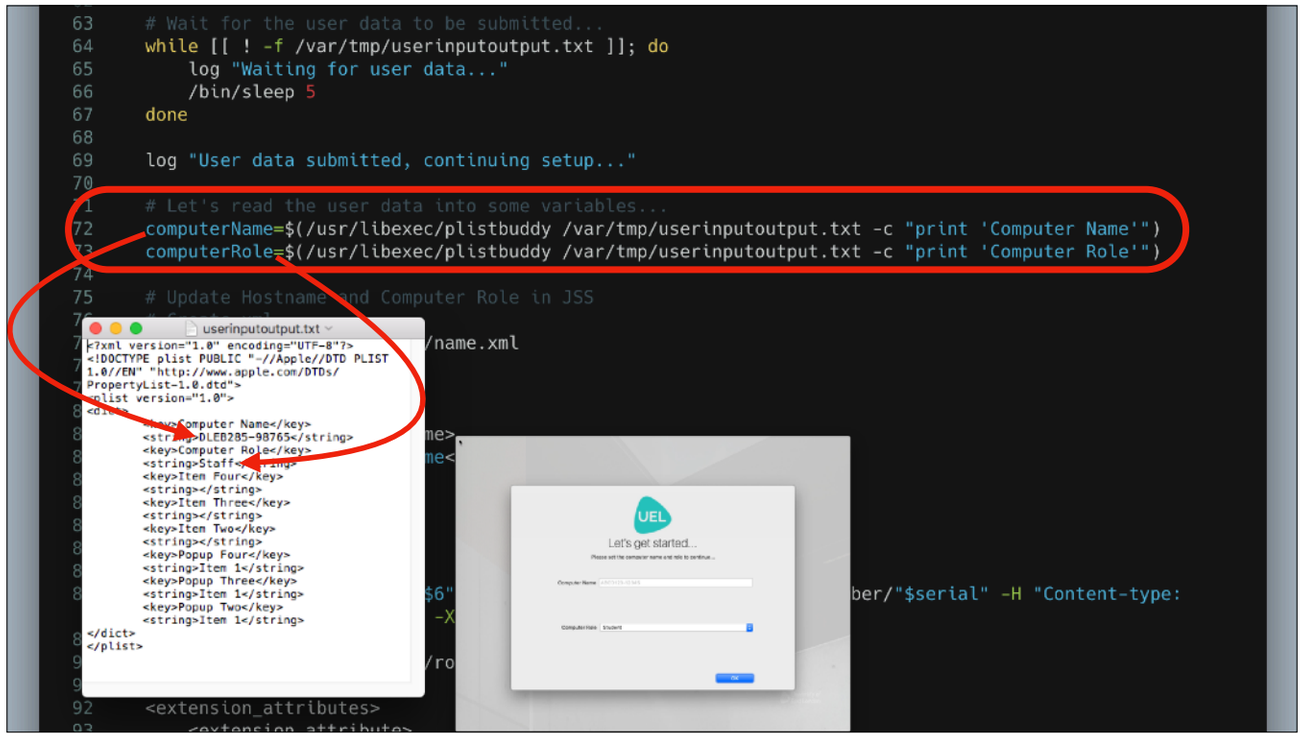
We’ll use authchanger to switch NoMAD Login over to using its User Input mech.



And we'll kill the loginwindow process to make NoMAD Login reload, where it'll show the User Input dialog.

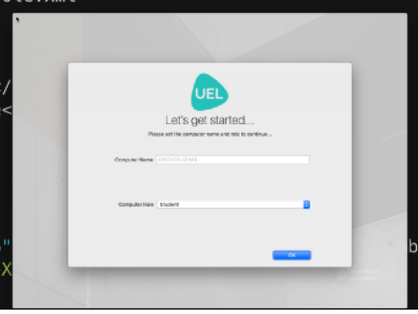


We'll wait until we the Computer Name and Role are submitted, which will be when our output file containing that data is created.

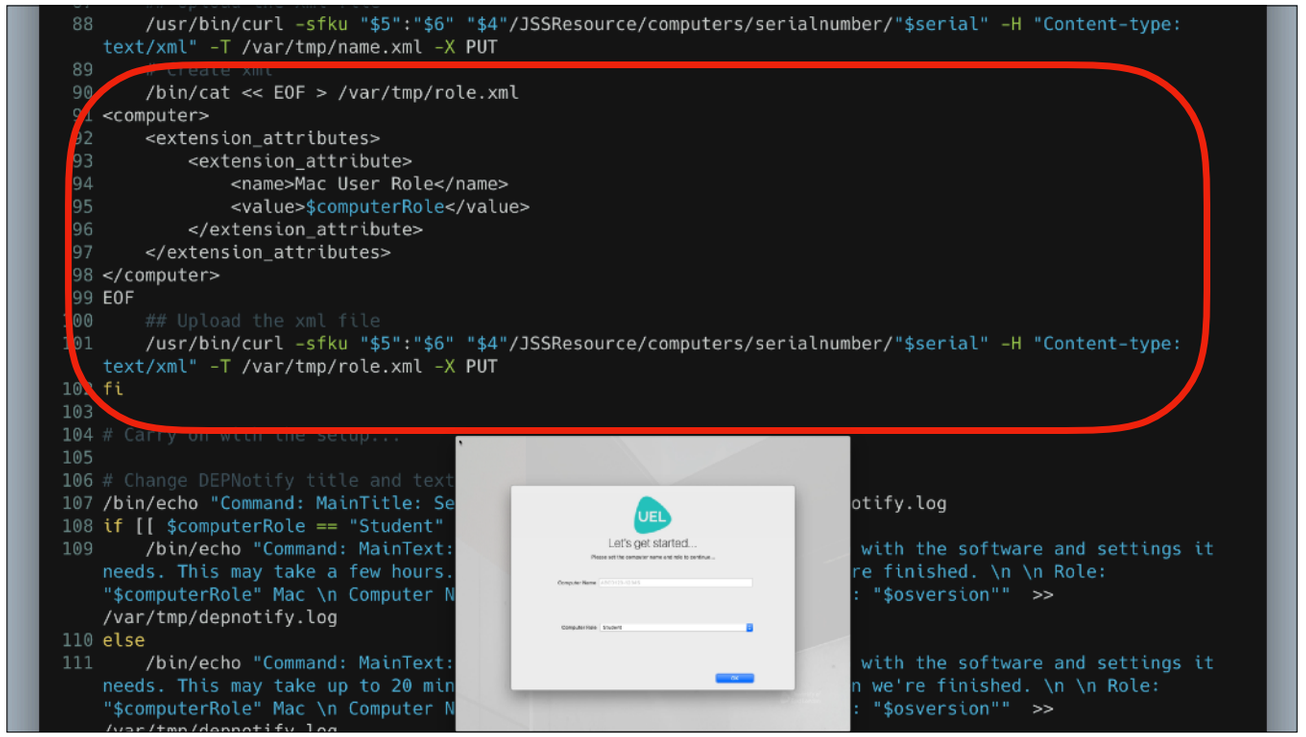


Then we'll read those name and role key values from that file into our computerName and computerRole variables in the script.

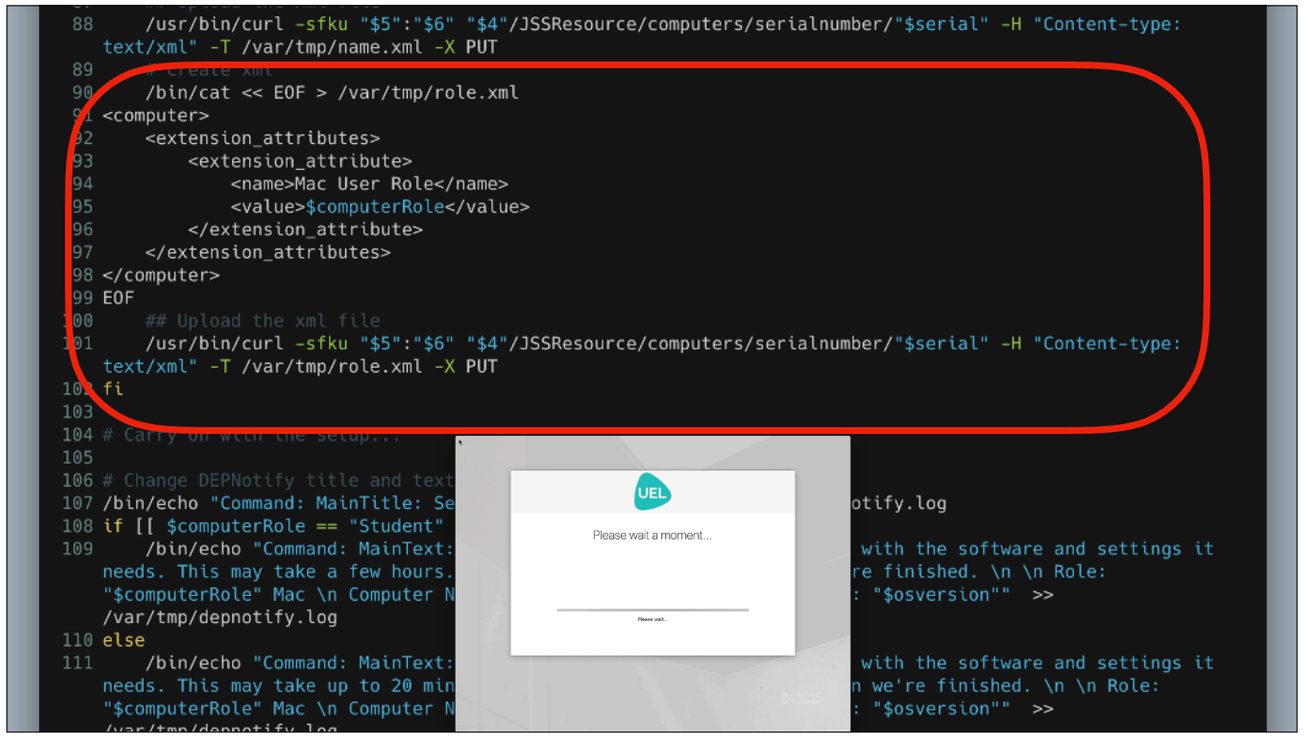
```
    computerRole=$(/usr/libexec/plistbuddy /var/tmp/userInputOutput.txt -c "print "computerRole"")

    # Update Hostname and Computer Role in JSS
    # Create xml
    /bin/cat << EOF > /var/tmp/name.xml
<computer>
    <extension_attributes>
        <extension_attribute>
            <name>Hostname</name>
            <value>$computerName</value>
        </extension_attribute>
    </extension_attributes>
</computer>
EOF
    ## Upload the xml file
    /usr/bin/curl -sfku "$5":"$6" "$4"/JSSResource/computers/serialnumber/"$serial" -H "Content-type:
text/xml" -T /var/tmp/name.xml -X PUT
    # Create xml
    /bin/cat << EOF > /var/tmp/role.xml
<computer>
    <extension_attributes>
        <extension_attribute>
            <name>Mac User Role</
            <value>$computerRole<
        </extension_attribute>
    </extension_attributes>
</computer>
EOF
    ## Upload the xml file
    /usr/bin/curl -sfku "$5":"$6"                                    ber/"$serial" -H "Content-type:
text/xml" -T /var/tmp/role.xml -X
fi
```



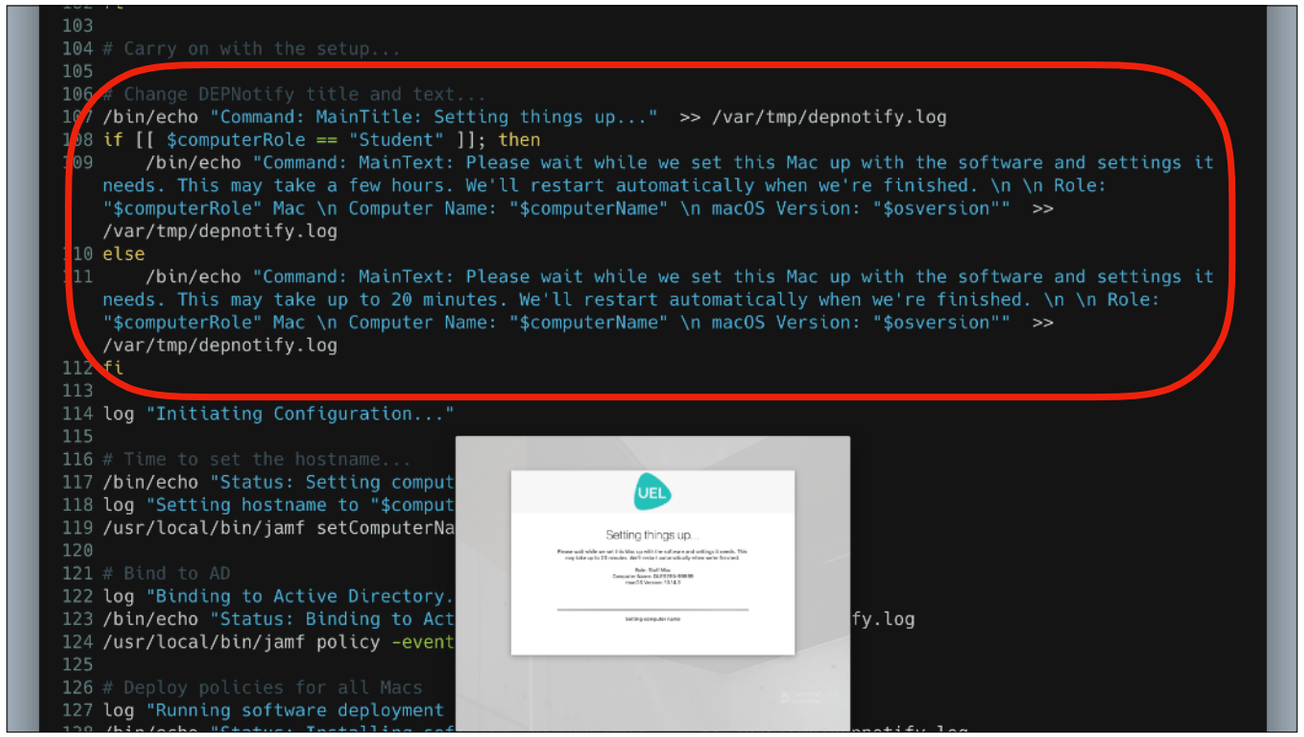
Next, we put the computer name into a chunk of XML and update the computer record’s Extension Attribute for it with the Jamf API.



We do the same for the User Role.



Then, it’s time to carry on with the provisioning. The person doing this can walk away now because the rest is automated.
If our Computer Name and User Role extension attributes were already present in the computer record, we would have skipped that user input step and ended up straight here.

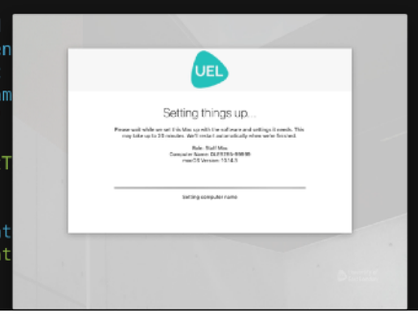


So we carry on, using the same echo commands we would for DEPNotify.

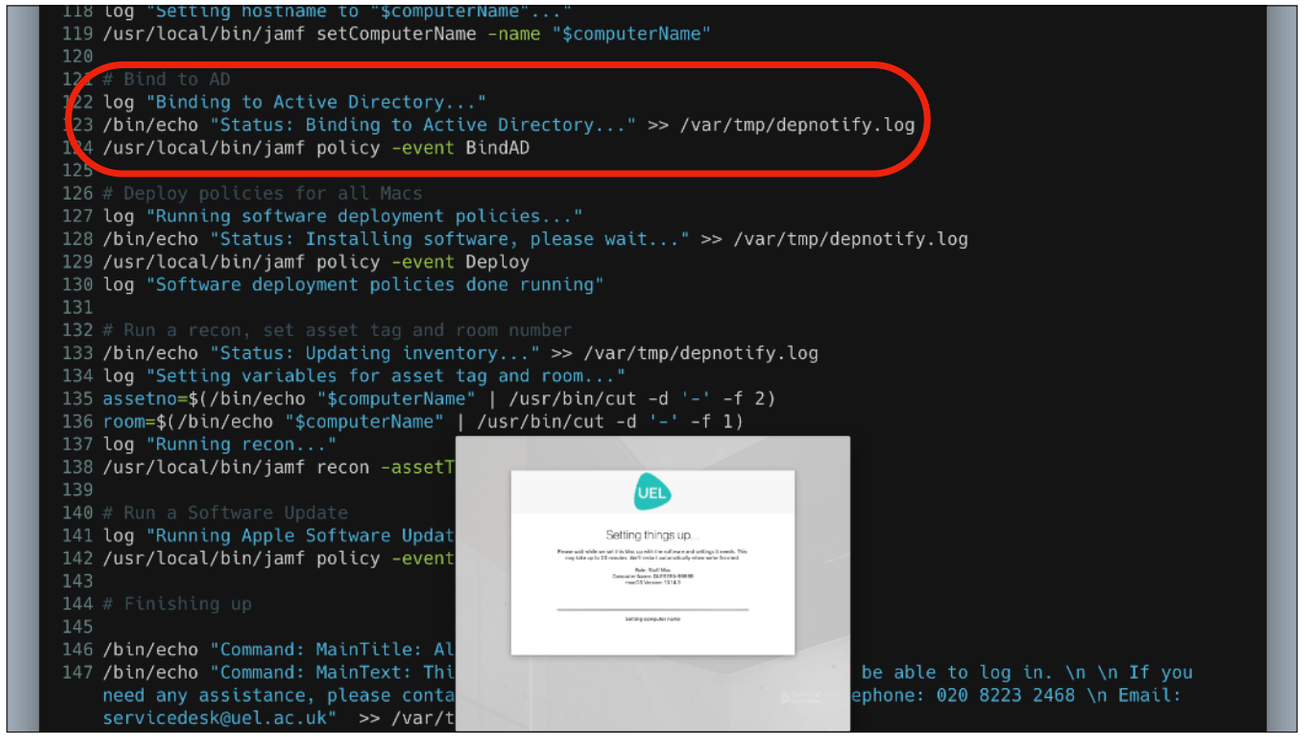
We change title, main text and status to let us know what's happening. And we can make use of our Computer Name, Role and macOS version variables to provide some useful feedback in the text.

One thing I also do here is, if the Mac is student facing, change the main text so people know it's going to take a couple of hours vs 20 minutes for a staff facing Mac.

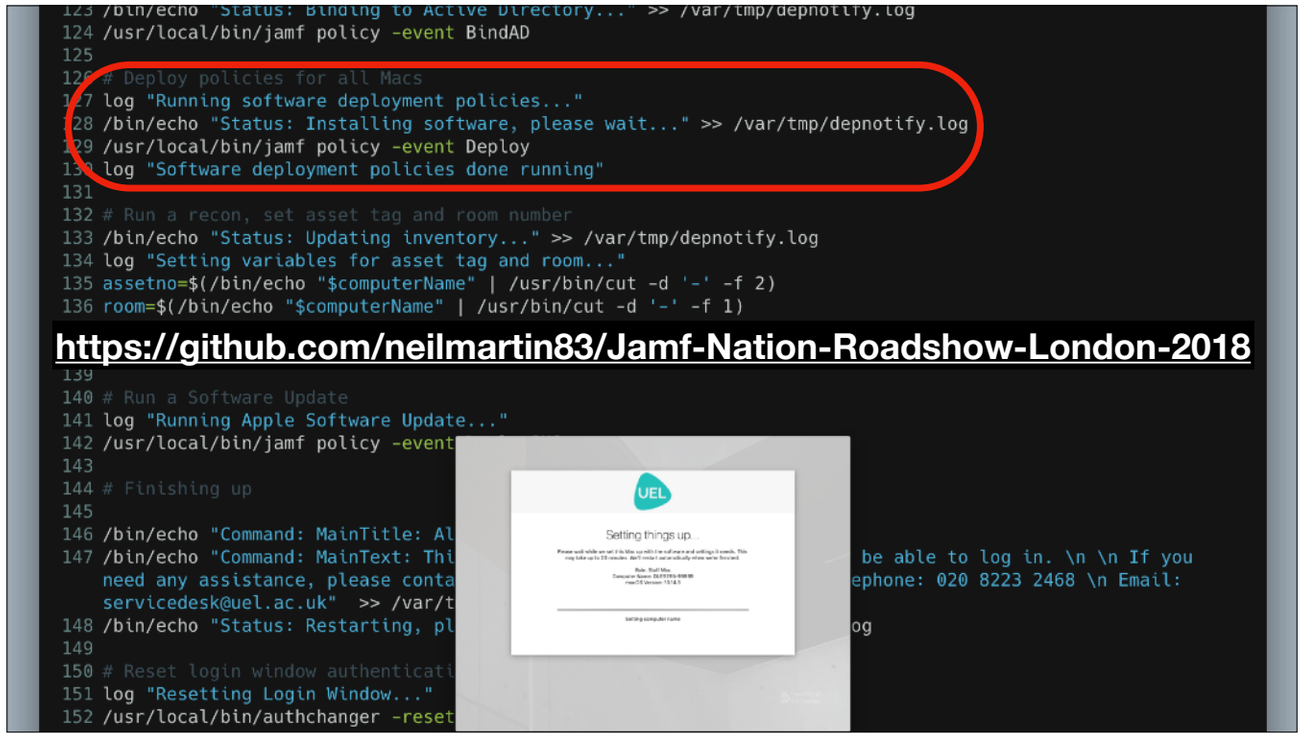
```
113
114 log "Initiating Configuration..."
115
116 # Time to set the hostname...
117 /bin/echo "Status: Setting computer name" >> /var/tmp/depnotify.log
118 log "Setting hostname to "$computerName"..."
119 /usr/local/bin/jamf setComputerName -name "$computerName"
120
121 # Bind to AD
122 log "Binding to Active Directory..."
123 /bin/echo "Status: Binding to Active Directory..." >> /var/tmp/depnotify.log
124 /usr/local/bin/jamf policy -event BindAD
125
126 # Deploy policies for all Macs
127 log "Running software deployment policies..."
128 /bin/echo "Status: Installing software, please wait..." >> /var/tmp/depnotify.log
129 /usr/local/bin/jamf policy -event Deploy
130 log "Software deployment policies done running"
131
132 # Run a recon, set asset tag and
133 /bin/echo "Status: Updating inven
134 log "Setting variables for asset
135 assetno=$(/bin/echo "$computerNam
136 room=$(/bin/echo "$computerName"
137 log "Running recon..."
138 /usr/local/bin/jamf recon -assetT
139
140 # Run a Software Update
141 log "Running Apple Software Updat
142 /usr/local/bin/jamf policy -event
143
144 # Finishing up
```



We’ll set the hostname - I use the Jamf binary because it has a function for it - but you could do it differently if you like.

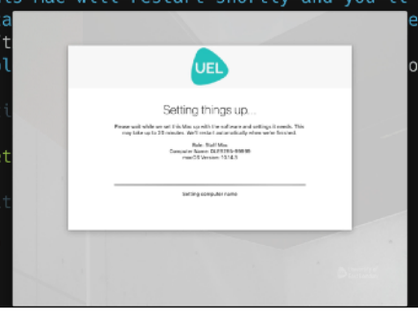


Bind to AD - I have another policy which does this and we fire it off with a custom trigger - those are really useful!

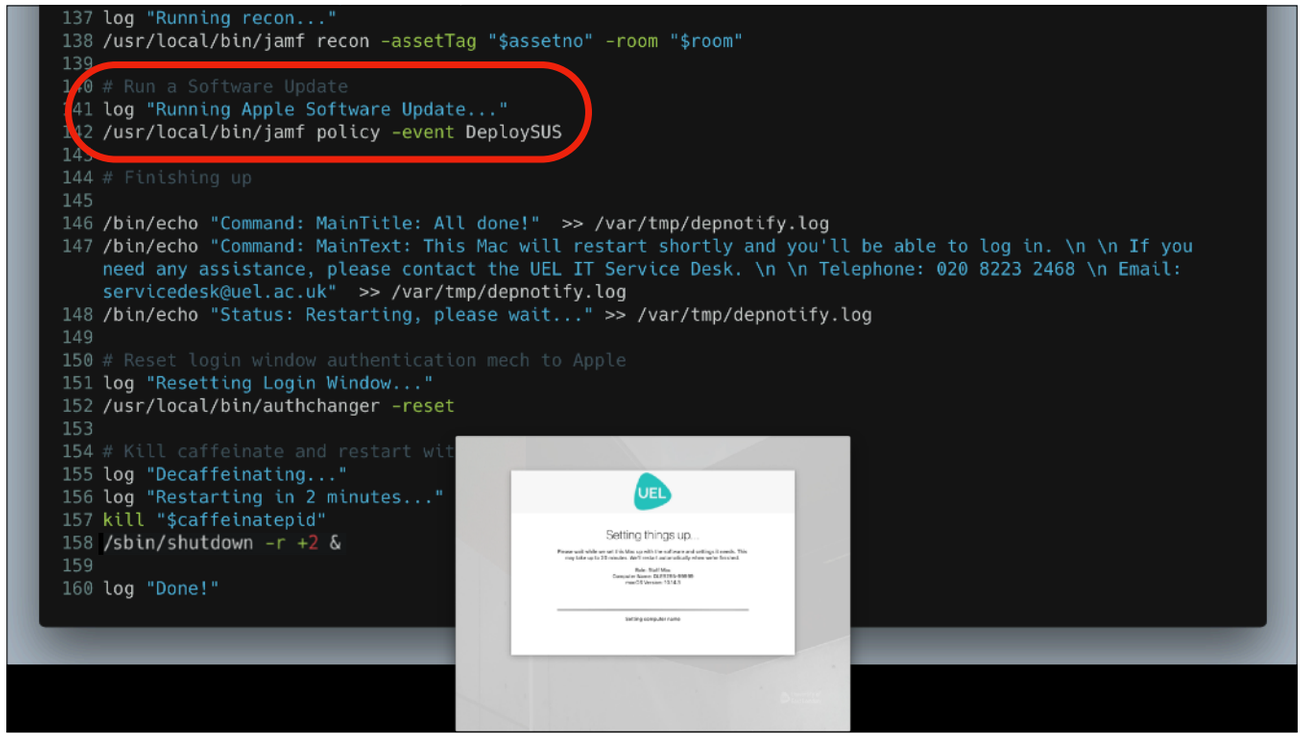


And then, we can run other custom policy triggers. These could install software or do anything else we want.
In my environment, this Mac will be in the right smart groups for its campus, building and lab.
So what I do is have all my software install policies use the same custom trigger - Deploy - but scope them out to the smart groups for the labs or locations we want that software to be.
If you want more detail on that, check out this link.
What's also cool, is that the status text at the bottom of the Notify window will be updated from the Jamf log as various packages install.

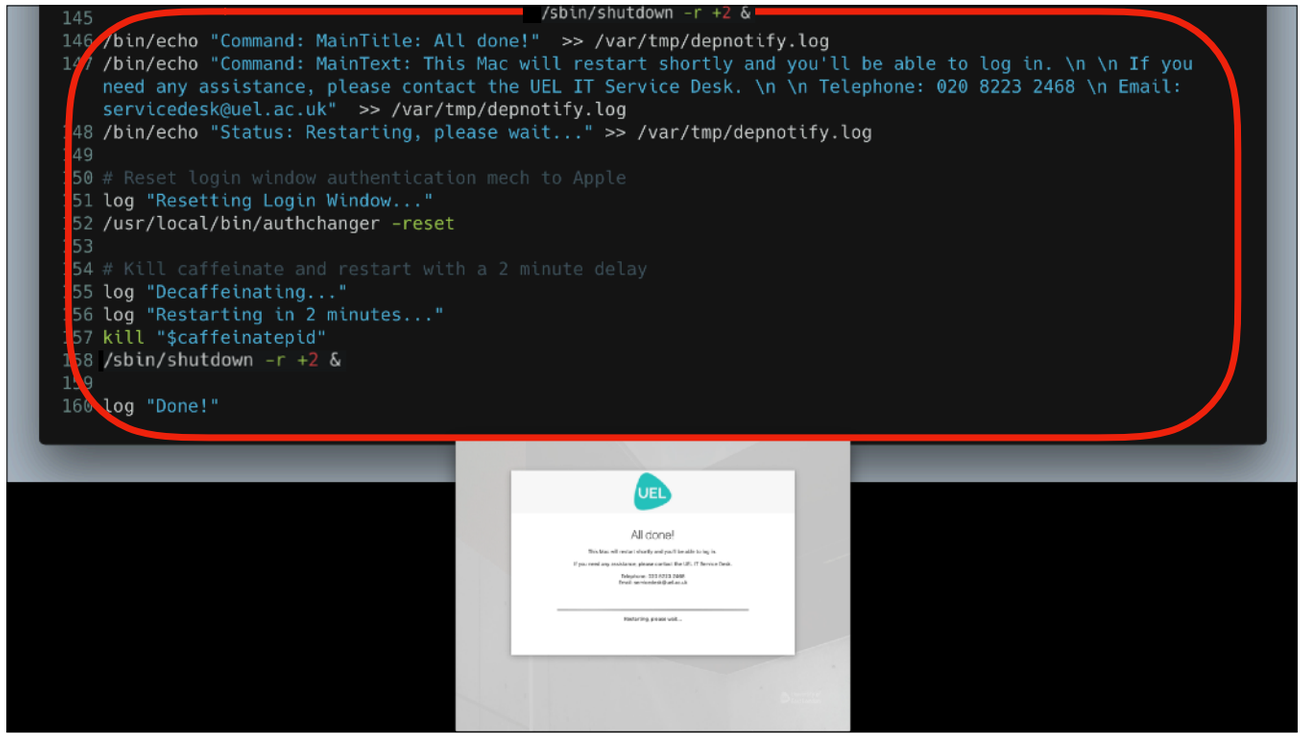
```
/usr/local/bin/jamf policy -event Deploy
log "Software deployment policies done running"

# Run a recon, set asset tag and room number
/bin/echo "Status: Updating inventory..." >> /var/tmp/depnotify.log
log "Setting variables for asset tag and room..."
assetno=$(/bin/echo "$computerName" | /usr/bin/cut -d '-' -f 2)
room=$(/bin/echo "$computerName" | /usr/bin/cut -d '-' -f 1)
log "Running recon..."
/usr/local/bin/jamf recon -assetTag "$assetno" -room "$room"

# Run a Software Update
log "Running Apple Software Update..."
/usr/local/bin/jamf policy -event DeploySUS

# Finishing up

/bin/echo "Command: MainTitle: All done!"  >> /var/tmp/depnotify.log
/bin/echo "Command: MainText: This Mac will restart shortly and you'll be able to log in. \n \n If you
need any assistance, please conta                                    ephone: 020 8223 2468 \n Email:
servicedesk@uel.ac.uk"  >> /var/t
/bin/echo "Status: Restarting, pl                                    og

# Reset login window authenticati
log "Resetting Login Window..."
/usr/local/bin/authchanger -reset

# Kill caffeinate and restart wit
log "Decaffeinating..."
log "Restarting in 2 minutes..."
kill "$caffeinatepid"
/sbin/shutdown -r +2 &
```



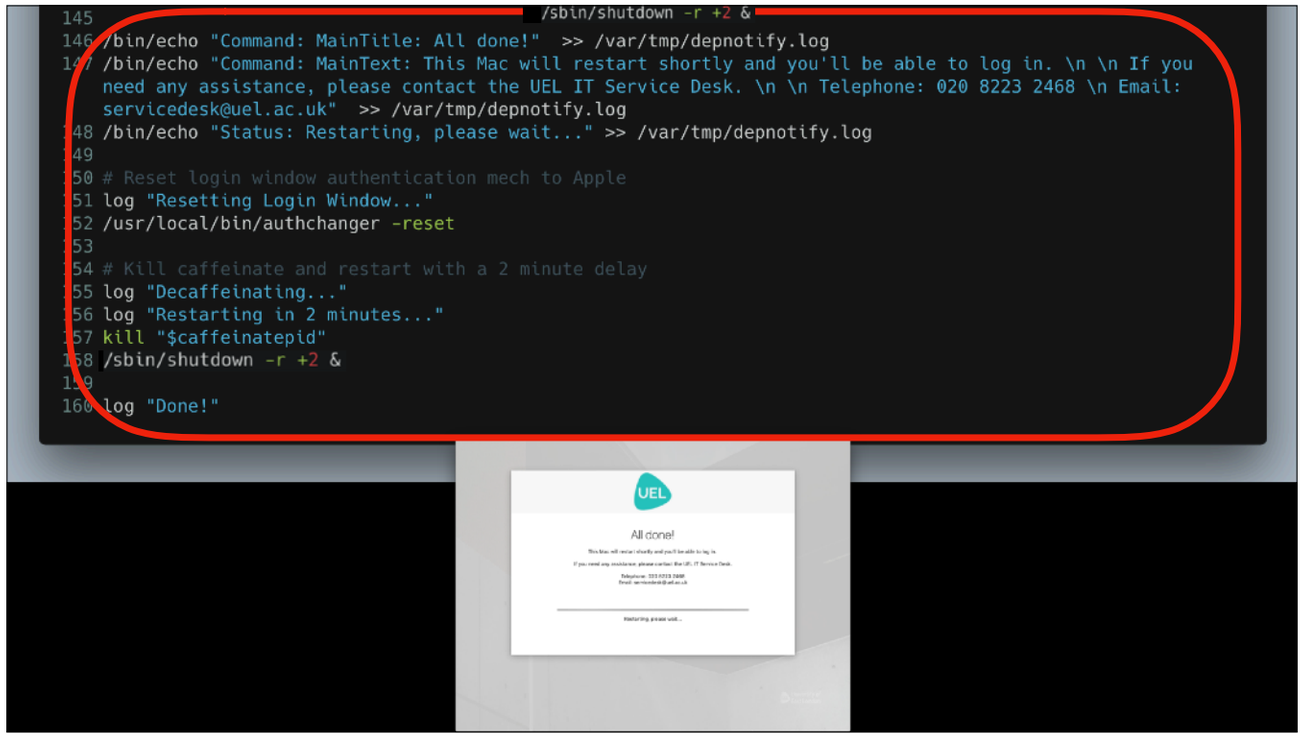
We run an inventory collection, and take the opportunity to use the different parts of the computer name to populate the asset tag and room attributes of the computer record.



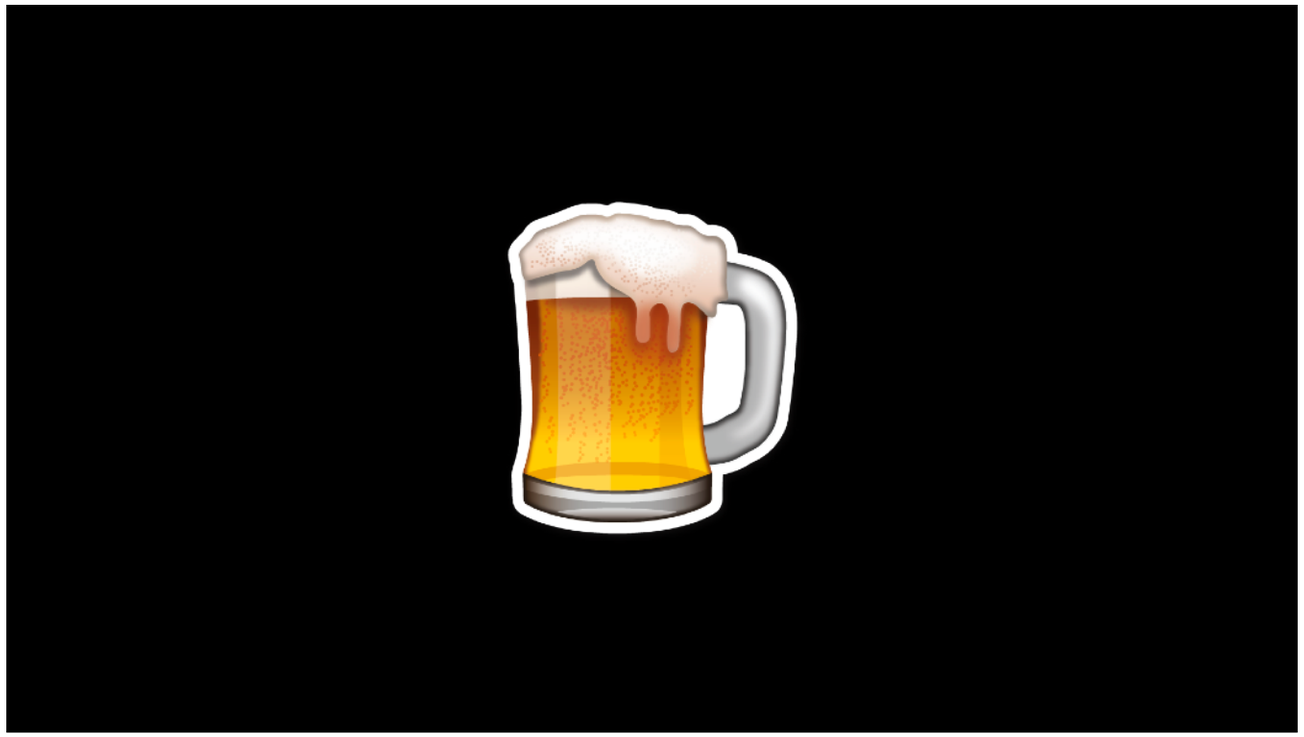
We run a software update to catch any security updates or new versions of things like iTunes and Safari. I do call another policy to do this with a custom trigger because I had issues calling the softwareupdate command directly where the provisioning process would hang.



And we’re finished, so we let people know, run authchanger to reset things back to using Apple’s Login Window then restart with a 2 minute delay - so the policy has time to complete and write its log back to Jamf.



And we’re finished, so we let people know, run authchanger to reset things back to using Apple’s Login Window then restart with a 2 minute delay - so the policy has time to complete and write its log back to Jamf.



Meanwhile, we've all been at the pub… or attending conferences to learn about all the different things we can do to improve our MDM enrolment workflows…

Thanks Joel!

I’d really like to thank Joel and everyone else who’s contributed to these awesome tools for all their work. And especially for making it look like I know what I’m doing!

Thank you!

And thank you!

https://github.com/neilmartin83/MacADUK-2019

And here are some resources for this talk.