Douglas Worley

Senior Professional Services Engineer
JAMF Software

The first thing I want to get off my chest – I'm not in Sales. I'm not here to sell you on the Casper Suite.

Next month brings my four year anniversary working with JAMF Software, and in that time I've been lucky to work with some rather large deployments using the Casper Suite.

The past 12-18 months have been full of a few exciting announcements about some large scale Apple deployments using the Casper Suite.

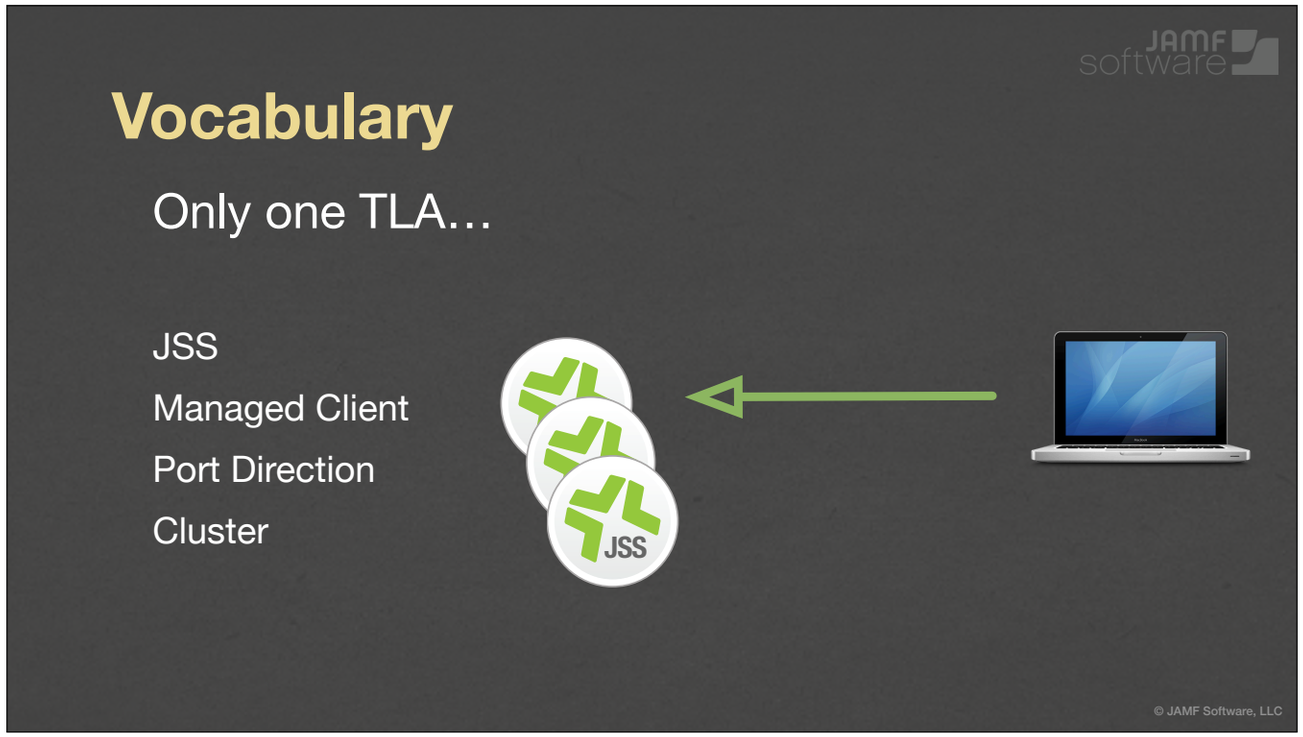
While I cannot speak to any specifics about any particular project or company, my goal today instead is to share some stories and insights into some of the larger deployments we have been party to.

Each deployment is mostly unique, depending on exactly what the requirements for the project are. There are however some constants that guide towards common solutions to common problems.

When scaling out the Casper Suite for a customer I have a number of leading questions to help determine what the server architecture ought to look like.

I can share some of the ways in which we need to think differently when moving away from 500 devices and instead you're facing down 50k devices. The tools simply work differently, and there are things we need to do to adapt.

I'd like to share with you all the questions I ask when designing a scaled Casper Suite infrastructure.

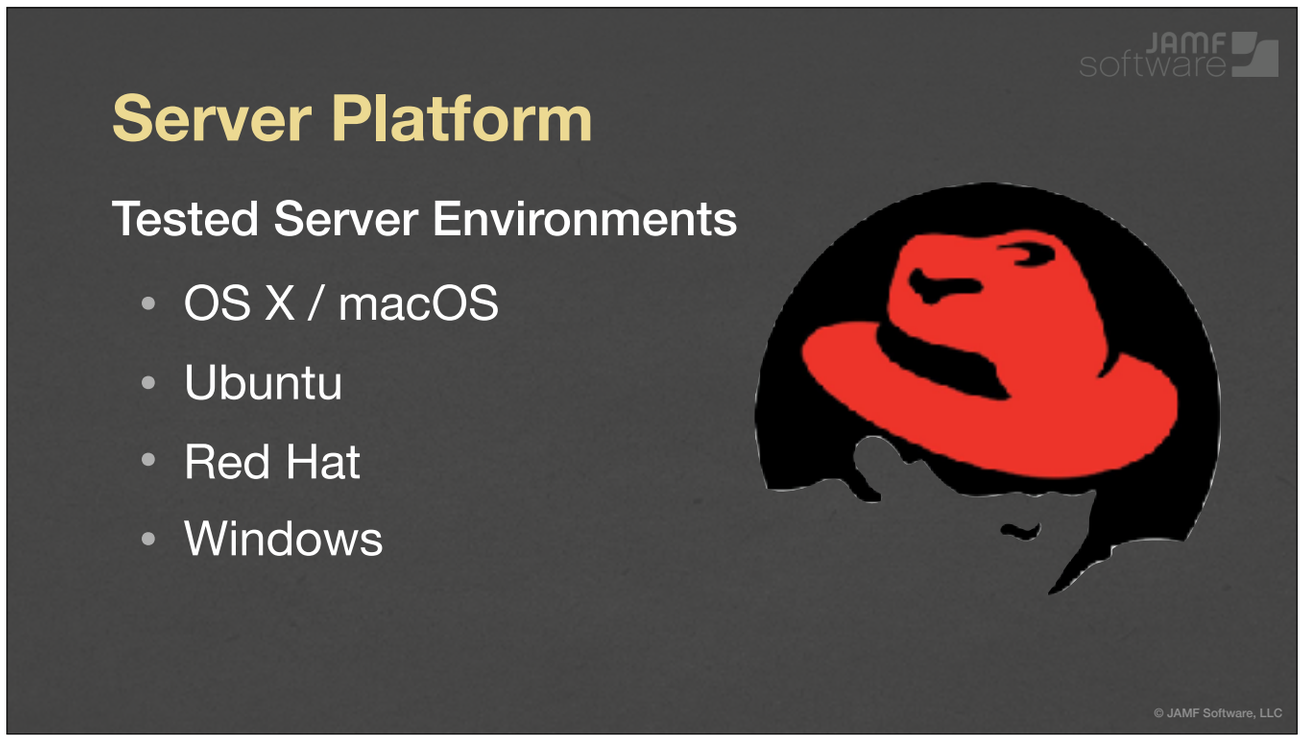


Before we go too far, let’s take a moment and make sure we are using the same terminology.

The Casper Suite from JAMF Software is the industry leader in management software for the Apple platform. Our product hinges on what we call the “JAMF Software Server” or JSS for short.

When someone refers to the JSS, it could potentially mean a couple different things.
1) We could mean a single web application, running on a single server,
2) Or we could mean the entire set of services, the entire Casper Suite which spans across multiple servers to provide a scaled environment for many many thousands of services.

When we talk about a “Managed Client” that refers to a macOS or iOS device that has:
1) Local configuration file which accompanies a certificate pairing that tells the device where to find the JSS, with a secure path to communicate. On macOS this is a plist combined with the jamf.keychain, while on iOS both components are combined in the MDM profile.
2) A corresponding inventory record in the JSS to match the devices configuration file. There is some accompanying security behind these inventory records and configuration files, but those are handled



We can install the JSS on any platform you want to support.

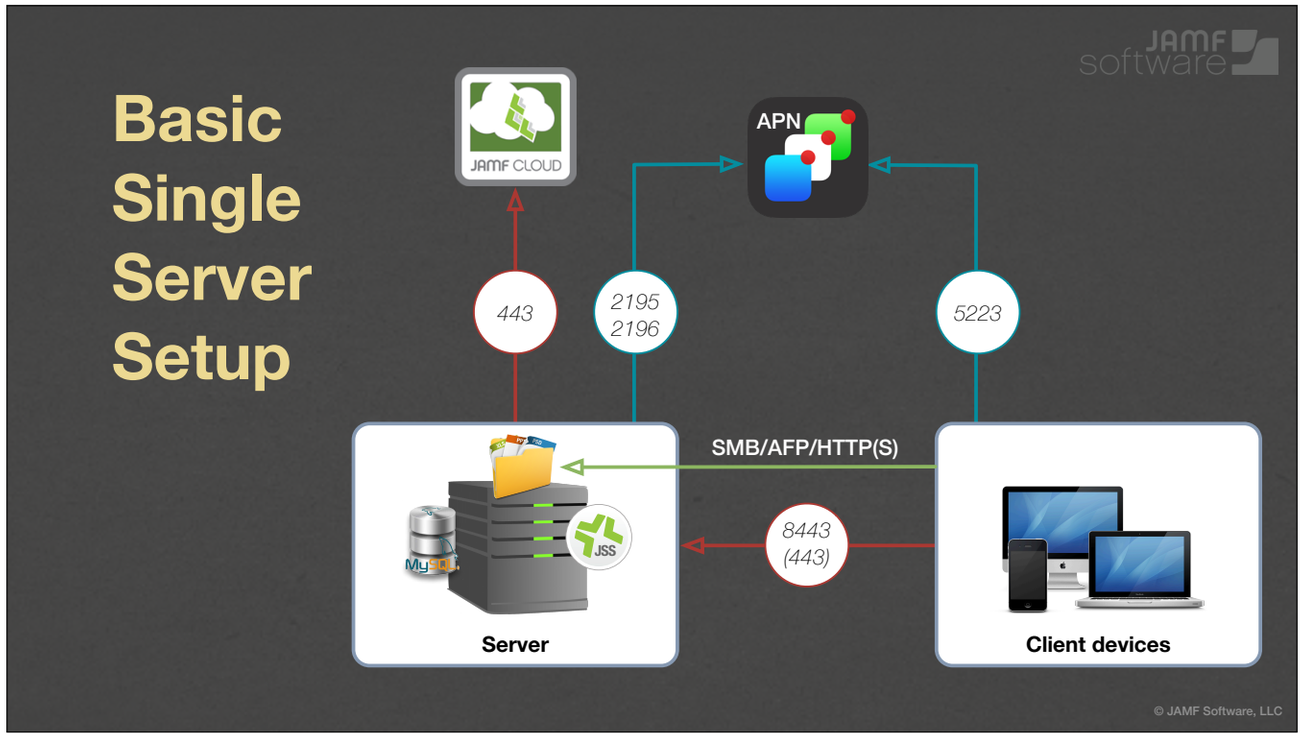
Mac, Linux, Windows.
It's funny. I've worked in Mac IT for 12 years and I have never touched so many Windows servers until I worked for JAMF.

Some of these OS platforms have better inherent security or support, some are cheaper, some are easier for various kinds of nerds to support. It's all the same to us, if we test against them then we commit to supporting it.

http://docs.jamfsoftware.com/9.96/casper-suite/administrator-guide/Requirements.html

Tested operating systems include:

- OS X v10.7
- OS X v10.8
- OS X v10.9



The JSS is comprised of a few different components:
1) MySQL database (5.5.x or 5.6.x)
2) Java JDK and JCE (1.7 or 1.8)
3) Apache Tomcat web server (7 or 8)
4) JSS Web Application
5) Package Distribution server, via AFP/SMB/HTTP(S)

Part of every sale is a JumpStart, which is an onsite service engagement where we normally install all components of the JSS on one server.

This is to say, that unless otherwise specified, an all-in-one server is the assumed deployment for all new customers.

Each of these services can be broken out onto different, discrete servers to spread out the load, and that is where this whole talk is pointing towards.

Workflows

Scaling depends on how much work being done

versus

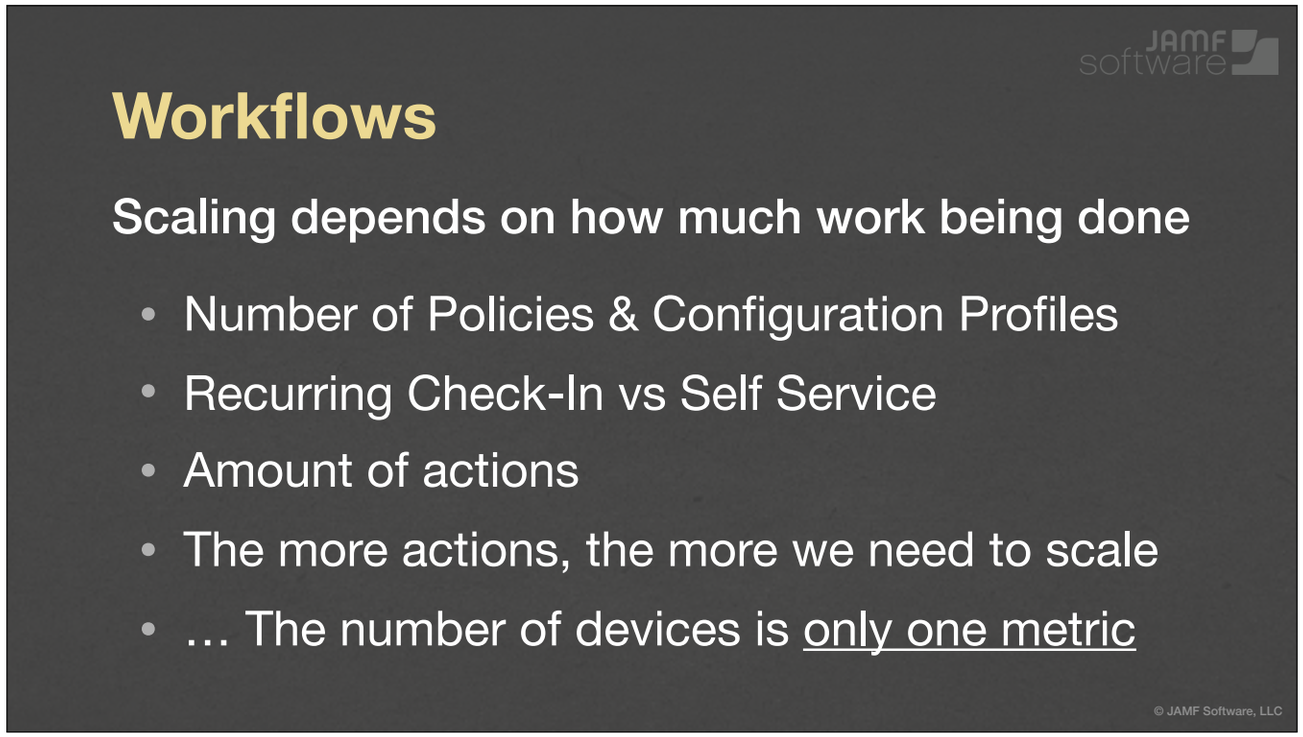
But before we can really determine how many servers and in what configuration, we need to take a step back and talk about how much work is really being done. The workflows that your Casper Suite will be performing are critical to consider when designing a scaled JSS.

If your JSS has a small number of policies, the Recurring Check-In is every60, most policies of which are via Self Service, and each policy is only doing a single action and isn't updating inventory, then we can fit more managed devices on each server.

Instead, if your JSS has a large number of policies, many policies have Inventory updates, APNS has User-Level profiles and/or certificate enrollment payloads, the Recurring Check-In is every15 or every5… you can imagine that the extra computation will require extra resources

That's all good, but we all need to know numbers to work from. This is a rough rubric, lots of other factors come in to determine the exact server configuration:

- 5-7k macs per tomcat node
- 10-12k ios devices per tomcat node.

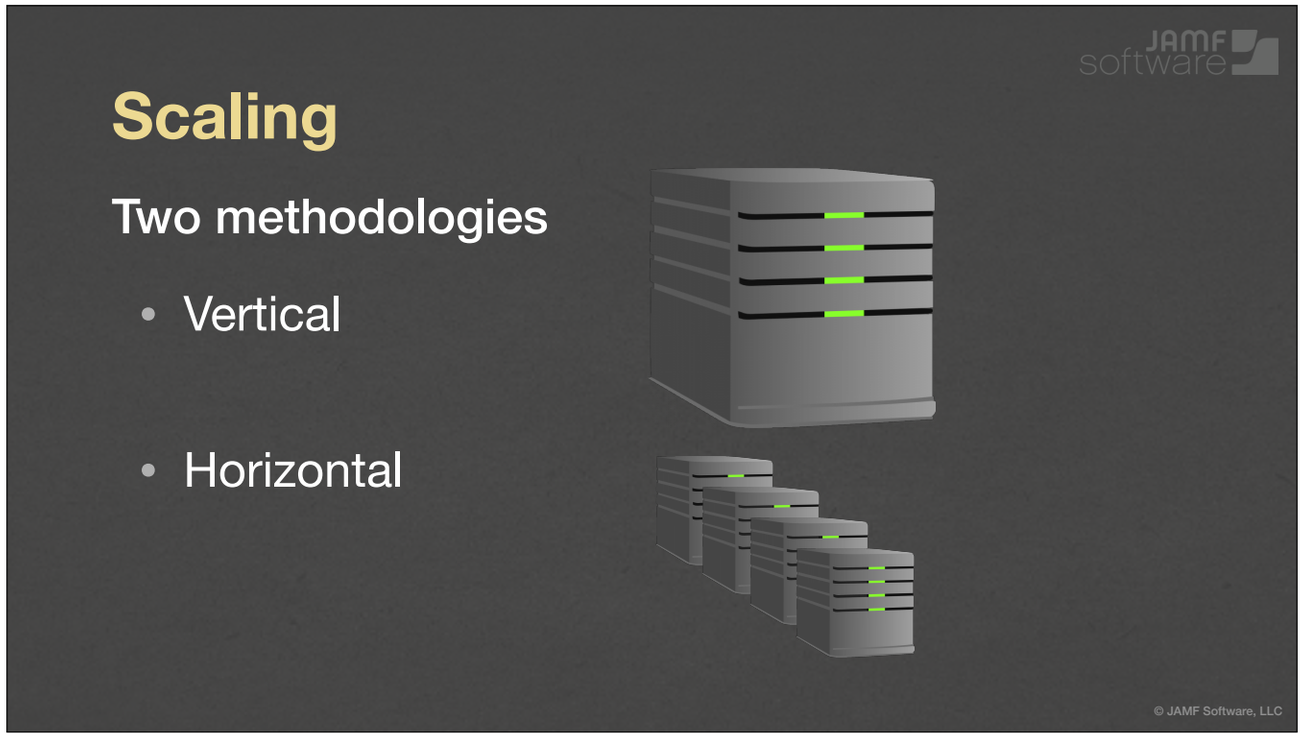


But before we can really determine how many servers and in what configuration, we need to take a step back and talk about how much work is really being done. The workflows that your Casper Suite will be performing are critical to consider when designing a scaled JSS.

If your JSS has a small number of policies, the Recurring Check-In is every60, most policies of which are via Self Service, and each policy is only doing a single action and isn't updating inventory, then we can fit more managed devices on each server.

Instead, if your JSS has a large number of policies, many policies have Inventory updates, APNS has User-Level profiles and/or certificate enrollment payloads, the Recurring Check-In is every15 or every5… you can imagine that the extra computation will require extra resources

That's all good, but we all need to know numbers to work from. This is a rough rubric, lots of other factors come in to determine the exact server configuration:
•5-7k macs per tomcat node
•10-12k ios devices per tomcat node.



This is a bit of foreshadowing, but there will come a point where we need to add more resources to handle a large number of devices, we need to consider how to scale the servers.

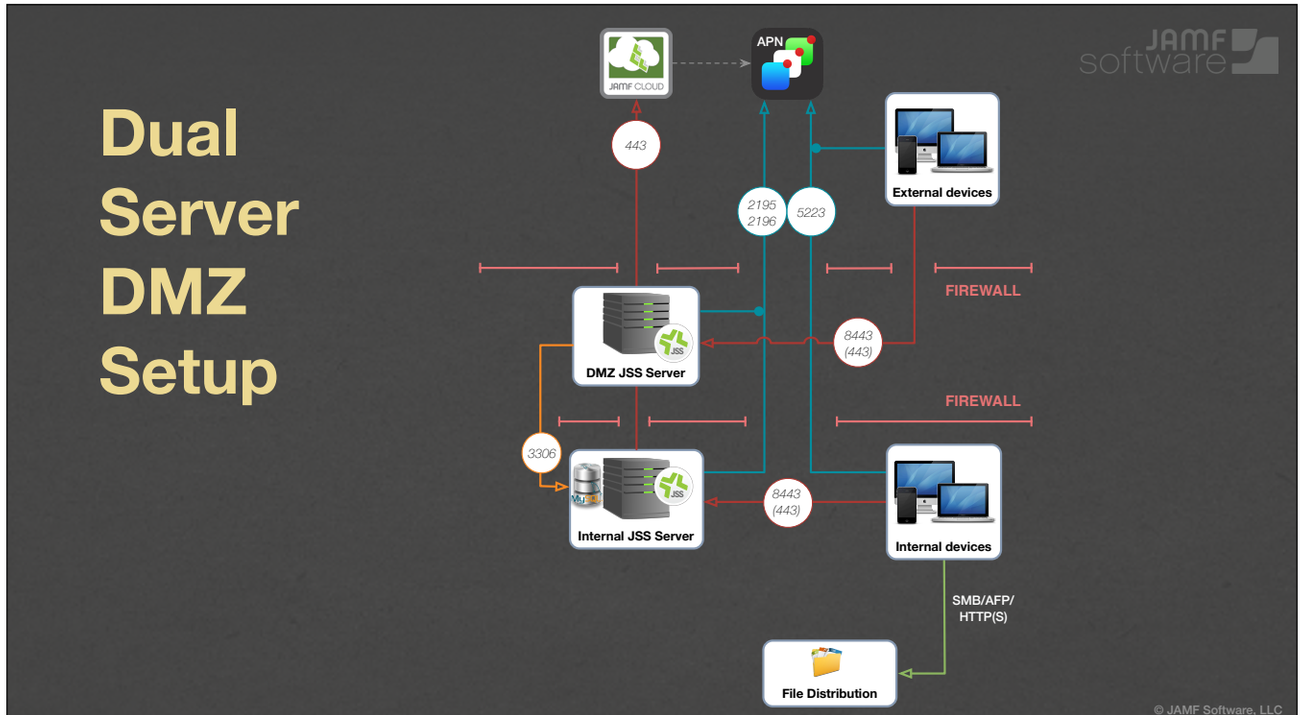
We can think about scaling in two ways: Scaling Vertically and Scaling Horizontally. We can do either, or both depending on a few factors.

Scaling Vertically refers to adding more resources to the server. Often we install the JSS on virtual machines, which are very easy to “upgrade” in hardware, which leads to scaling a JSS vertically by adding more CPU, RAM, etc to the core server. Once we do that, we can add more resources to Tomcat and MySQL to let them be more performant.

Scaling Horizontally refers to adding more “stock” servers into the cluster.

There are Pros and Cons to each methodology, and you might choose to use one or the other or both depending on your specific environment.

Dual Server DMZ Setup



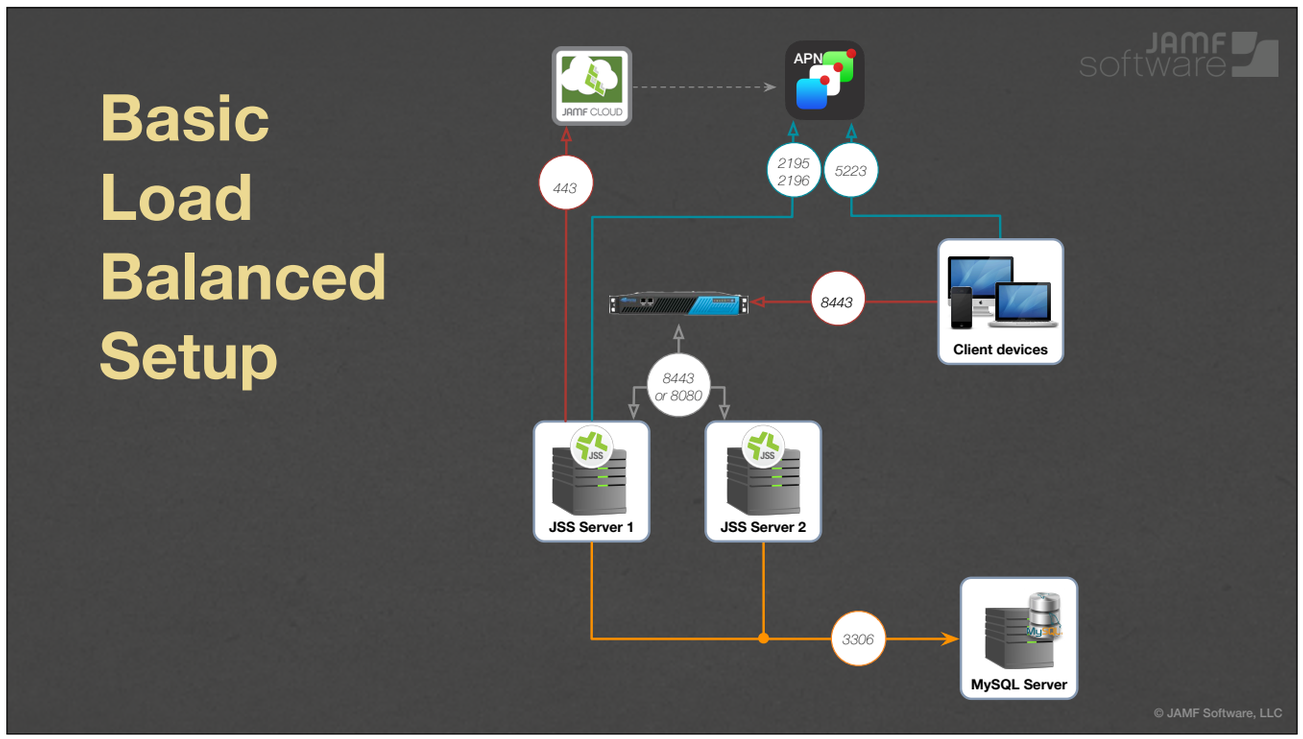
The first step we can provide for external access is to put a JSS web server in the DMZ. At this point a split-horizon DNS would tell devices on both sides of the corporate network how to find a JSS, each of which share the same database.

We see here that the JSS web application (in the DMZ) does not need to live on the same server as does the database. This is the beginning of a scaled deployment.

We also see that there are two JSS web servers talking to the same database. This here is the beginning of a Cluster, which adds some complications.

This sort of scenario works very well for many many deployments, up until there are too many devices to be effectively served by only one JSS.

Basic Load Balanced Setup



There isn't a simple rubric here, but there comes a point where there are simply too many managed devices for one JSS to handle. At this point we need to add more servers to handle the demands of the incoming client communication.

At this point Tomcat/Database/File Distribution are all separated onto distinct services. From here we will mostly ignore package distribution, that's a whole other discussion that could go a number of directions.

We need to make sure that session stickiness and persistence are configured properly so that the managed clients reliably keep contact with the same JSS server.

Inventory Management Concerns

Best practices:

- Spread out Inventory Updates
- More Advanced Searches for queries
- Smart Groups only when needed for policies
- Consider how many criteria are in each Smart Group
  - … is there a simpler way to get there

One of the problems that comes in with large number of devices is database bloat and computational overhead on the Tomcat nodes.

Some of the largest deployments I've seen limit down how many policies run an Inventory Update. By running the update too often it is possible to see certain database tables blow up in size.

For example - check out this shell snippet showing a table growing by 1 megabyte roughly every minute. This was caused by having a policy update inventory on the every15 trigger with 4k Macs.

Other concerns come down to how the information in the database is handled.

Even though the interface in the JSS is almost identical to create an Advanced Search or a Smart Group, the way they are handled in memory are completely different.

An Advanced Search is only rendered when the button in the JSS is clicked, or when it is queried via the API. That is to say, there is very little overhead in having any number of Advanced Searches.

Inventory Management Concerns

```
[16:00:54] root@casper-db01:/var/lib/mysql/jamfsoftware# du -ch applications.*
16K  applications.frm
218M applications.MYD
101M applications.MYI
319M total
[16:01:54] root@casper-db01:/var/lib/mysql/jamfsoftware# du -ch applications.*
16K  applications.frm
218M applications.MYD
102M applications.MYI
320M total
```


One of the problems that comes in with large number of devices is database bloat and computational overhead on the Tomcat nodes.

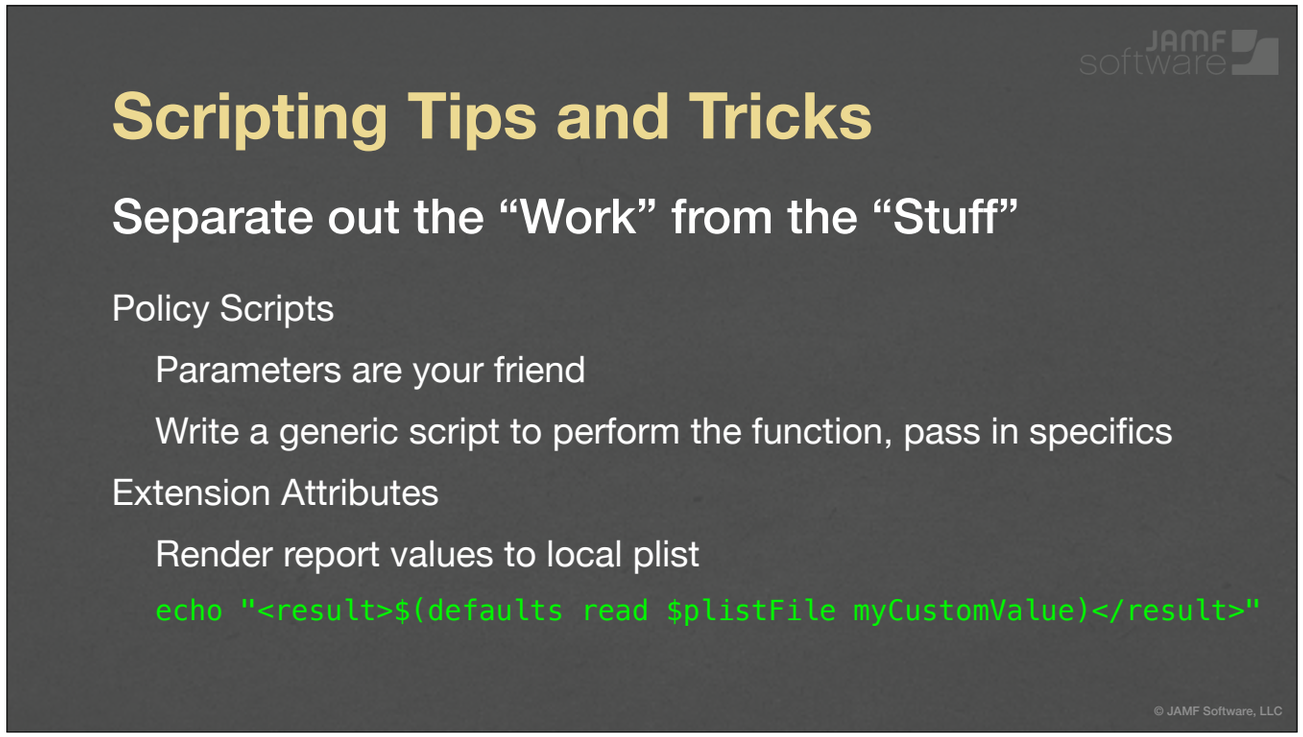
Some of the largest deployments I've seen limit down how many policies run an Inventory Update. By running the update too often it is possible to see certain database tables blow up in size.

For example - check out this shell snippet showing a table growing by 1 megabyte roughly every minute. This was caused by having a policy update inventory on the every15 trigger with 4k Macs.

Other concerns come down to how the information in the database is handled.

Even though the interface in the JSS is almost identical to create an Advanced Search or a Smart Group, the way they are handled in memory are completely different.

An Advanced Search is only rendered when the button in the JSS is clicked, or when it is queried via the API. That is to say, there is very little overhead in having any number of Advanced Searches.



There are often times when I have to write two or three versions of the same script to do subtly different things.

One of the most powerful features of the Casper Suite is the ability to create custom dynamic Computer inventory groupings based on values returned from scripts.

These scripts run every time a computer submits inventory. The more complicated these scripts are, the longer each Recon will take, and the more work each computer has to accomplish.

Deployments of all sizes benefit from thinking about this intelligently, but most of all large scale deployments.

Let’s consider a workflow where we need to determine some complex workflows. We can run scripts to determine those values, but do we need to run them every time a Mac submits inventory?

If the information is not likely to change, it can work out nicely to set everything into a custom plist. That

I'll be presenting on something very very similar at the JAMF Nation User Conference in a couple weeks. I'll be adding more stuff to that presentation, so if you have any questions let me know and I'll probably add it to that talk.

As per tradition, we will be recording and publishing those talks online. I'll follow up with Tycho to get this talk and that talk cross-linked for folks interested in this topic.

Extra talking points:

Inventory collection concerns:
1) Just because there is a checkbox, you don’t necessarily need to click it.
2) MySQL binary logging
3) Fonts
4) Plugins