Note that this is not a complete list, but an accumulation of details collected through support tickets and discussions with THD and that their priorities may have changed since the time of writing.

* [THD's Issue & Wish List](https://docs.google.com/spreadsheets/d/1SP6MDz_vKAJ81DMpyg8mLTG3yQI2hM_Z24ENdz3-vbM/edit#gid=329757844)

**THD Concerns:**

* THD would like to be more involved with product teams
  + What did Puppet test with?
    - VMware servers 16x6 or what set up?
  + THD wants to be involved in the improvement process and not just get the “fix”, but let them test in between to confirm that this is addressing THD’s issue -- keep them involved in the process

As of Aug 20, 2020, these are THD’s top priorities according to John Thebault

1. **Code Manager,**
2. **Self-Healing,**
3. **Make Upgrade easier,**
4. **Metrics to get indication how env is behaving in real-time. How do I know volume is happening and not hitting a threshold. Some way of looking at the environment to see not exceeding AFTER they made changes! How Many Jruby used, CPU usage? How to get something like that on the Console → very custom specific, he understands.**

**Priority as stated by THD**

1. **Unable to use Code Manager with their 2,500 + modules   
   (PM: Carl and Nick is PO)**  
   → **r10k can execute large #s of modules in less than 30 seconds vs 10 - 15 minutes (*pre July 2020*) using Code Manager.** 
   1. **Please read:** [More details as of July 14, 2020 shared by Patrick see **at bottom of document**](#_hnqtkm2lxyge)
   2. Reid was involved in the past
   3. Supp tickets
      1. [39833 - First time running code manager](https://puppetlabs.zendesk.com/agent/tickets/39833) → Charlie worked with Patrick to reduce time to 1-2 minutes.
      2. [35972 - Code Manager performance issues](https://puppetlabs.zendesk.com/agent/tickets/35972)
      3. [CODEMGMT-1222 Allow Deploying Only a List of Modules Across Environments](https://tickets.puppetlabs.com/browse/CODEMGMT-1222)→ fixed in **PE 2019.4 (fixed per Puppet...need to verify. Patrick is testing)**
   4. Oct 2019: Worked with Carl and Charlie

BJN Recordings  
  
**2019-10-17 Puppet / The Home Depot - Weekly TAM Sync-20191017: Code Manager Issue**

**Description:** Charlie Sharpsteen discussed with Patrick Mayes THD's Code Manager issue.

Streaming recording link: <https://puppetlabsmeetings.webex.com/puppetlabsmeetings/ldr.php?RCID=e7da8ba2c4d3fdc6d55a64d94e0e21b5>

October 17, 2019, 12:09 pm Chicago Time

Duration: 39 min

Recording password: bPMpRPw3

----------------  
  
**2020-01-16 Puppet / The Home Depot - Weekly TAM Sync: THD Demo: POC Virtual Puppet File System**

*Create time: 1/16/20 12:16 pm*

*Duration: 12 minutes*

***Description:*** *Russell Miller, Systems Engineer at THD, created this POC to share a different approach to Code Manager. This POC demonstrates a way to inject modules directly via an API to reduce dependencies on the actual file system itself. The idea is to make Puppet think differently of how to do deployments.*

*Note: This POC is focusing on functionality and not cache/speed optimization.*

*Streaming recording link:* [*https://puppetlabsmeetings.webex.com/puppetlabsmeetings/ldr.php?RCID=da02e8138ea02630f913ed4e0516a422*](https://puppetlabsmeetings.webex.com/puppetlabsmeetings/ldr.php?RCID=da02e8138ea02630f913ed4e0516a422)

1. **Orchestrator Job** is queuing up too many jobs (worked with Charlie and still an issue after his recommendations → *PS Engagement Opportunity? \*Update 08-2020 this is now addressed via services engagement, root cause found and suggested fixes delivered.*
   1. [37030 - Orchestrator job is queuing up too many jobs](https://puppetlabs.zendesk.com/agent/tickets/37030)  
        
      Deployment workflows are moving from MCO to PE Orchestrator

[\* Check here for more details](#_h64gkv3itlkr)

**Patrick: Orchestrator Job is queuing up too many Jobs - still an issue after recommendations from Charlie?**

* + 1. **Patrick: increasing Java memory has been done. It seems Orchestrator is not dying, small issue, bunch of systems that take a long time… need help for tuning in HA environment. Current system is stable.**

1. **Node Graph Deprecation and functionality**
   1. [39583 - Puppet resources.dot files](https://puppetlabs.zendesk.com/agent/tickets/39583)
   2. [PE-29474 Enhancing resources.dot related features and Node Graph](https://tickets.puppetlabs.com/browse/PE-29474)

Quote from Patrick at THD: “If this is something that Puppetlabs is depreciating I understand but I feel saddened by the fact that instead of improving a tool Puppetlabs has decided to completely get rid of it. I am formally requesting to have some way via the Puppet Console that can be exported to customers that will provide resource dependency in an easily digestible form (potentially from PuppetDB Report).“

1. **True HA (active/active)** for PE infrastructure (like Walmart and Wells Fargo)
   1. Abby is aware of it
2. Finetune Performance of New Production-HA Environment
3. PE in Containerized Environment
   1. [PC-719 PE Containerization Phase 1](https://tickets.puppetlabs.com/browse/PC-719)  → THD asking for this since mid 2018
   2. Larissa Lane presented this on August 9, 2018 (Mike Stahnke, Ethan Brown,David Lutterkot)   
      <https://puppetlabsmeetings.webex.com/puppetlabsmeetings/ldr.php?RCID=08bb79936f1cb03e88aa4146167bb32f>  
      Recording password: QuXV4iyT
   3. THD areas of interest & use cases
      1. Availability 24/7/365 (active/active HA)
      2. Auto ramp up Compilers, PuppetDBs
      3. PE upgrade pains - quickly/easily stand up new containers compared to convoluted architecture
         1. Between 4 - 24 hours + 12 hours of upgrade issues due to their custom architecture, because no HA provided by Puppet. Made PE HA themselves → during upgrade cannot use LB names.
         2. Customized scripts to keep CAs in sync
         3. Versioning of facts not documented - breaking the base facts
         4. Syntax changes w/o warnings until they upgraded, which broke their environment
         5. blue/green path → ideal for THD (before upgrading to PE 2019.2) → good rollback strategy as well
      4. Easily stand up test environments for internal customers
4. Self-Healing in PE  
   Patrick: looking for anything that throws that is a known issue to provide a “self-healing” process.
   1. Offer from THD (Patrick): TAlk to Puppet support/product/application teams how THD can share recurring issues, their “self-healing” list that they handle and manage themselves.
   2. Action Item: Bring someone in from Puppet to see where THD is right now, what errors they see recurring (self healing list), → have everything documented (STATE OF THD so we can share it with Puppet folks → FACTS!)
   3. → THD Willing to sit down and share with our support/dev/product → here is our struggles, here’s the tools we are using,
5. Cloud Native focus
   1. **5/7 Notes: Mike - teams deciding how to manage devices through Cloud platforms… discussions what tools**
   2. **Operating system on demand - Server on Demand - manage it directly. Less controlled environment, but self-service.**
   3. **Established in GCP, Pivotal PCF, and Azure, have large investment.**
   4. **Mike - cloud and management: strong on focus on application teams - led to diversified model.**
   5. **Self installation of required products. Diff platforms come in, diff organizations setting up their own models to use these devices, → under Kevin Mayes, become more standardized.**
   6. **Single tool would be preferred**
   7. **THD Engineering has not lots of experience with Containers, Drew Sears, (IaaS and other stuff) Manager, direct report is Larry Lane (soon Patrick, who is shifting teams)**
   8. **Randall M(endoza) runs virtualization space (VMware)**
   9. **Evan Woods - “build team”, cloud build, cloud management, if you need a one-off solution, or complete cloud stuff**
   10. **Sean Dougherty, Security, under Brian Higgenbacham (Sr. Director)**
6. Patching in PE (Linux)
7. Training Offerings Beyond Practitioner Level
   1. **→ more in-depth:** 
      1. **how MoM works,**
      2. **signing,**
      3. **DB,**
      4. **endpoints, etc. works.**
   2. **Would be nice to have a course cover that so it’s consistent and they know what people will learn.**
   3. **→ good training would be: how to troubleshoot Puppet? Common issues, etc. (couple hours training)**
8. Continued Interest in Litmus (when it meets their needs)
9. Issue/Wish list - shared with product teams
   * **if Product teams want to talk about things, THD is available.**

**Interests**

* Any integration with **OnceOver** they are VERY interested in learning about this!

THD’s Key features for OnceOver:

==========================

* + mock a system that looks as close as possible to system xyz and run the test against that. (same cpu, memory, OS, stock items, so when they run it; potential custom facts)
  + Other developers ask: how do you know that it will work on 8x64, etc.
    - It failed on 2x4 because didn't have enough "power"
  + Core facts of OS, CPU, though is what they look for to match 1:1 to safely and with confidence tell them it works in their tests
* Windows DSC Conversion (low priority)
* **Mike; Patching is of interest. Switching repositories (space walk replaced by artifactory) and feature sets (for Puppet might feed into Patching) → Andrew B.**
* (CIS, but only if it meets their needs)

**Professional Services hours from last September**

* 1. Expiring September 10, 2020
  2. Need to complete SOW before we can actually use them
  3. Randy: **SOW by July done** (adopt PE in Windows env, CD4PE, Patching (Win/Linux - what Kevin was asking long time ago to combine both?)

→ Randy: roughly 5 weeks

**FOLLOW UP - PS Engagement Opp:**Orchestrator Job is queuing up too many Jobs - still an issue after recommendations from Charlie?

* + 1. Patrick: up Java memory has been done. It seems Orchestrator is not dying, small issue, bunch of systems that take a long time… need help for tuning in HA environment. Current system is stable.

>>>> Orchestrator Job is queuing up too many jobs → details from Charlie’s support ticket <<<<<

## **Orchestrator Limits on Concurrency**

Currently, the workflows are posting a series of jobs to the /command/task API endpoint of the Orchestrator:

<https://puppet.com/docs/pe/2017.3/orchestrator_api_commands_endpoint.html#reference-6045>

This endpoint has a task-concurrency limit, with a default of 250 operations "in flight", and operates on a "first come first serve" basis. This means that a long-running job like running puppet agent on 1500 nodes will occupy those 250 slots until all nodes finish. Any job posted to /command/task after the agent runs will have to wait until the number of runs remaining to execute for the puppet agent job drops below 250.

The Orchestrator has another endpoint, /command/deploy that is purpose-built for running the Puppet agent:

<https://puppet.com/docs/pe/2017.3/orchestrator_api_commands_endpoint.html#post-command-deploy>

Jobs posted to this endpoint have a concurrency limit that is seperate from /command/task, this will prevent puppet runs from blocking other tasks and also allows the load on the compiler pool to be controlled. However, the /command/deploy endpoint only offers a fixed set of options and currently does not support passing the --tags flag that mco puppet runonce offered or the --job-id option for puppet agent that your workflow uses to track the reports from Puppet runs.

## **Increasing Task Concurrency**

Our main concern with increasing the concurrency past 250 is that this will allow more puppet agent runs to occur simultaneously which will increase the load on the compilers. Our scale testing in this area has assumed that the concurrency option of /command/deploy will be used to limit the number of simultaneous puppet agent runs based on the number of JRubies available in the compiler pool.

From our discussion today, it sounds like your workflow has been using mco puppet runonce without any concurrency-limiting options such as --batch, --batch\_sleep, or --splay. So, we should be able to increase the Orchestrator concurrency given the compilers are bearing the load of puppet agent runs started en-masse by MCO.

The main resource of concern when placing a peak load on the compiler pool is the number of Jetty worker threads each pe-puppetserver instance has for queuing incoming requests. If these threads become 100% utilized, the services may start rejecting additional incoming requests. Each pe-puppetserver instance runs 200 of these threads, but ~20 end up reserved for various purposes. So, the saturation point is around 180 waiting requests. With 12 compilers in the pool, that would give us a max queue length of 180 \* 12 = 2,160. So, 2,000 might be a reasonable upper limit for increasing the Orchestrator's task-concurrency as that leaves each compiler about a dozen threads to handle non-agent related traffic.

Aside from Jetty threads, the other resource we should make sure we have enough of is RAM for pe-orchestration-services. Each thread added to a Java JVM will allocate 1 MB of native memory for the thread's execution stack. Since this is "native" memory, you don't need to increase the Orchestrator heap size for it, but in order to add 1750 extra threads you will need at least 1750 MB of free RAM on the master. Additional Orchestrator heap may be required if the input or output data of the tasks is large as there could be 2000 instances of this I/O shuffling around in memory instead of 250.

In PE 2018.1 and newer, you can adjust the task-concurrency by setting the following value in Hiera:

puppet\_enterprise::profile::orchestrator::task\_concurrency

However, this setting was not managed in PE 2017.3.4, so you'll need to add a resource to change it:

pe\_hocon\_setting { 'orchestration-services.orchestrator.task-concurrency':

ensure => present,

path => '/etc/puppetlabs/orchestration-services/conf.d/orchestrator.conf',

setting => 'orchestrator.task-concurrency',

value => 2000,

notify => Service['pe-orchestration-services']

}

You may also need to increase the Linux nproc limit to allow the Orchestrator to spawn more threads. This can be done with a file resource:

file { '/etc/security/limits.d/pe-orchestrator.conf':

ensure => file,

content => 'pe-orchestration-services soft nproc 8192',

notify => Service['pe-orchestration-services'],

}

You can check the limit currently imposed on pe-orchestration-services by looking at the "Soft Limit" column of the "Max Processes" row from running:

cat "/proc/$(cat /var/run/puppetlabs/orchestration-services/orchestration-services.pid)/limits"

The RedHat 6 machine I tested on had a default of 1024, which would not be sufficient to to add 1750 additional threads without allowing a higher limit via /etc/security/limits.d/.

## **Monitoring PE Metrics**

I recommend monitoring two sets of metrics to see how the system reacts to the increased task-concurrency:

* The average-requested-jruby metric from pe-puppetserver. This will let us know how close we're getting to that threshold of 180 Jetty worker threads on the compilers.
* The CPU and Java heap usage of pe-orchestration-services. This will let us know if increasing the Orchestrator task-concurrency is overtaxing the CPU on the master or we're running out of heap space to hold task input and output data.

The average-requested-jruby and heap usage numbers can be retrieved with a GET request to the /status API of the services:

* pe-puppetserver: https://<node hostname>:8140/status/v1/services?level=debug
* pe-orchestration-services: https://<node hostname>:8143/status/v1/services?level=debug

The Support and IT teams at Puppet have collaborated on a module that sets up Telegraf to collect this data, store it in InfluxDB, and display the results with Grafana:

<https://forge.puppet.com/puppetlabs/puppet_metrics_dashboard>

Since you folks are already using Telegraf, the configuration for monitoring pe-puppetserver is very straight forward if you would like to add it to your own system. Here's a link to the template for telegraf.conf:

<https://github.com/puppetlabs/puppet_metrics_dashboard/blob/v2.0.1/templates/telegraf.conf.epp#L7-L20>

The puppet\_metrics\_dashboard module currently does not monitor pe-orchestration-services, but the configuration would be the same as pe-puppetserver with the difference of using port 8143 instead of 8140.

CPU usage can be collected via your method of choice.

===============

### 

### **July 2020: Update on Code Manager Pain as stated by Patrick Mayes**

[...] call last week with Patrick and below are some of my notes from that call. Patrick's call [...] is based upon his call [working session] with Charlie Sharpsteen:

A common theme in these conversations is that Patrick/THD are expressing concern that they are having the same conversation with different people from Puppet but they are not seeing the desired progress.

"Orchestrator not working"

"Code Manager is not ready for THD yet."

**Item 1**

While he can see that there has been a major improvement, the time required to execute an update is still too long. What used to take several minutes now takes 2 minutes. By comparison **R10k takes 5-10 Seconds. This is the range they need this to be**.

The reason...they are deploying between 10 and 20 million times a year. Multiply by 110 second difference and it starts to back up after a while. **They will lose the support of their Customers as this time protracts out**

**Item 2**

He mentioned that Code Manager shows "everything" that was done and it is very difficult to find and validate if a single item was completed.

Item 1 and 2 are on a single support ticket

**Item 3**

Code Manager requires the use of Puppet Console. THD needs clarification around why this is a problem.

Item 3 is on a separate ticket

Items 4 and 5 are not on tickets

**Item 4**

Documentation was poor...it took 2 weeks to figure things out.

**Item 5**

More Clarity around testing requested.

Documentation regarding different environment types (Red Hat 7 vs Ubuntu)

**Goals**

To get ServiceNow integration. they see CM speed as a barrier to using SNOW integration

Use Code Manager ...it will eliminate a ton of code.

To upgrade from 2017.3 to 2019.21 it took 4 engineers 1 year to re-write the workarounds required

Today they have to rely upon exit codes to figure out what happened. Code Manager could tell them faster.

Patrick is remarkably upbeat and hopeful that we can work through this. We see challenges ahead because Development priorities are focused elsewhere.