Parallel Task Execution

Mco replacement

AUGUST 2018

AUTHOR:
Jose Manuel de Frutos Porras

CERN IT-CM-LCS

SUPERVISORS(S):

Steve Traylen
CERN IT-CM-LCS

Project Specification

The following project consists on evaluating the new open source task runner Puppet-Bolt that executes ad hoc tasks across a set of infrastructure and applications as possible replacement for MCollective. Bolt connects to remote systems via SSH, so it doesn't require the installation of any agent and tasks unlike Mcollective that requires a daemon to be running in each managed machine. Is important to add that Bolt tasks are reusable and shareable and can written in any programming language.

Abstract

Puppet is a great tool for making changes on systems, and ensuring that those changes happen. But Puppet is not intended to make this happen on many systems at the same time. Puppet is intended for eventual compliance over time. Each agent checks in over a period of time, allowing the puppetmaster to process only a few at a time. In very few environments could every puppet client get a catalogue from the puppetmaster and execute it at the same moment in time. Additionally, each puppet client would take differing times to process their catalogues. Configuration Management Infrastructure at CERN is using MCollective as a framework for parallel job execution. It is commonly used to orchestrate change across cluster of server in near real time.

Puppet Inc has recently announced a new Open source task runner that executes ad hoc tasks across a set of infrastructure and applications. It is called Puppet-Bolt and it's particularly suited for troubleshooting or deploying one-off changes, distributing scripts to run across all infrastructure, or automating changes that need to happen in a particular order as part of an application deployment. Moreover, Bolt is driven through a command line interface and connects to remote systems via SSH, so it doesn't require the installation of any agent software. The tasks that Bolt runs are reusable and shareable via the Puppet Forge and can be written in any scripting or programming language.

In this project we will evaluate Puppet-Bolt as possible replacement for MCollective which is lacking real orchestration features and requires a daemon to be running in each managed machine.

Contents

List of Figures

List of Tables

Listings

1. Puppet-Bolt evaluation as a replacement for Mcollective

1.1 Mcollective

MCollective provides a framework for parallel job execution. It is commonly used to orchestrate change across cluster of server in near real time. MCollective is an adjunct tool in your tookbox that cooperates and enhances the capabilities configuration management like Puppet. Whereas these tools analyse and act to ensure complete configuration consistency, MCollective orchestrates specific and often singular actions across system significantly faster. MCollective was designed from the ground up to achieve true parallel execution with consistent and repeatable results.

We stand out the following characteristics of the architecture of Mcollective:

- **Masterless:** MCollective avoids the use of a centralized master for command and control, thus avoiding centralized resource problems. It also doesn't reach out to the clients in an ordered loop, thus avoiding drift between each of the systems.
- **Publish-Subscribe Middleware:** MCollective uses Publish-Subscribe Middleware to transport requests to servers. Any node you want to control by MCollective is an MCollective server running mcollectived. Any system which has the mco command installed is an MCollective client which can issue commands.
- **Deamon Running in each server:** On each server mcollectived registers with the middleware broker and remains in a listening or IDLE state. Whenever a client sends a request to the middleware, each server receives and evaluates the request immediately and independently. If mcollectived has an agent installed capable of process the request, it executes it immediately and sends back the results. In this model you can have a command execute on tens, hundreds, or thousands of hosts at exactly the same time.

While MCollective plays very well with configuration management systems, it works above and outside of them.

1.1.1 Mcollective functionalities

We highlight the following functionalities:

- Interact with clusters of servers, whether in small groups or very large deployments.
- Break free from identifying devices through complex host-naming conventions, and instead use a rich set of metadata provided by each machine — from Puppet, Facter, or other sources — to address them.
- Use simple command-line tools to call remote agents.
- Write custom reports about your infrastructure.

- Use agent plugins to manage packages, services, and other common components created by the community.
- Write RPC style agents, clients, and web UIs in Ruby.
- Leverage rich authentication and authorization models in middleware systems as a first line of control.
- Include fine-grained authentication using SSL or RSA, authorization, and request auditing. For more information, see the Security Overview.
- Re-use middleware features for clustering, routing, and network isolation to realize secure and scalable configurations.

1.1.2 Installing

It is best to use Puppet to deploy and maintain MCollective. You will be constantly tweaking the MCollective configuration and adding new plugins. Every change will need to be synchronized across servers, yet many servers will also have customized settings.

As the installation documentation on the Puppet Labs website says, "[MCollective] is the textbook example for why you need config management". Indeed, Mcollective is characterized by:

- It has multiple components that run on many different machines.
- It has pieces of global configuration that must be set harmoniously, everywhere.
- Most of its settings are identical for machines in a given role (e.g. every server), but some of its settings have per-system differences. This is easy to manage with a template, and incredibly frustrating to manage by hand.
- Its configuration will change over time, and the changes affect many many systems at once.

In summary, its configuration requirements are strict, and configuration drift will cause it to stop working. To be able to successfully meet the requirements when configuring Mcolective, you must use a configuration manager like Puppet.

Next we show quickly how to install Mcollective, in your cluster you probably want to tune the installation. If that is your purpose we recommend to see [7, p. 119-122] and [5]. At the time of preparing the evaluation we have installed this way Mcollective in our cluster.

First, we have to install the Mcollective module from Forge [5].

```
puppet module install puppet-mcollective
```

Listing 1.1: Install puppet Mcollective module

Then, If you wanted to install a machine that provided ActiveMQ as middleware and had the server and client installed as well, you would use:

```
manifests/mco.pp:
node default {
  include activemq
    class { '::mcollective':
    client                => true,
    server                => true,
    middleware_hosts      => ['activemq.example.net'],
  }
}
```

Listing 1.2: Install Mcollective puppet

For most nodes on your network you would use only install mcollective server. You should set mcollective client parameter to true on any hosts from which you wish to submit requests, this may be your management hosts, a bastion host, or could be your laptop or desktop systems in the office.

1.2 Puppet-Bolt

Puppet-Bolt is an open source tool from Puppet that allows you to run code on other servers without needing to install a puppet client on the servers. This is achieve by connecting with remotes systems via SSH or WinRM. Puppet is driven by the command line interface. The pieces of code that Bolt executes (called Task and plan in Bolt argo) are easily reusable and shareable through Puppet Forge, and can be written in any script or programming language.

Unlike Mcollective, Bolt's architecture is Masterless & Agentless because it connects remotely to a device via SSH or WinRM so no agents are needed, due to bolt does not need a publish-subscribe middleware.

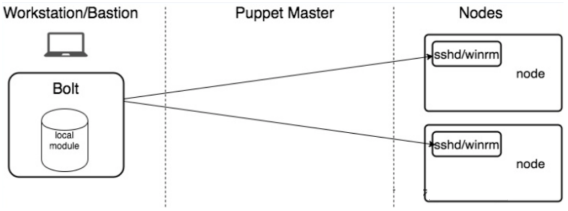


Figure 1.1: Bolt Architecture 1

1.3 Bolt Functionalities

There is a one-to-one correspondence between the functionalities capable of performing Mcolective and Bolt's functionalities.

- Bolt also offer the possibility to interact with clusters of servers and break free from identifying devices through complex host-naming convention, see 1.4.2 and 1.4.3.
- Also uses a simple command-line tool 1.5.
- It gives you the opportunity to write Tasks and Plans in any programming and scripting language. You can share Tasks and Plans and reuse those that are already coded by the community, see 1.5.2, 1.5.4 and 1.5.5.
- Bolt uses SSH as a transport layer, so it includes authentication and encryption provided by SSH.

1.4 Installing Bolt

To use Puppet-Bolt, your system must run Ruby and have a GCC compiler. In versions higher than $>= 0.17.2$ Puppet-Bolt require a Ruby version equal to or greater than 2.3.

We have work on the puppet module that on Forge [2] so that all the pertinent dependencies are resolved among other things automatically when including the module. We have also worked on this module so that it is possible to generate the Bolt configuration file (at least part of it) indicating the configuration values as a parameter in the module. See `https://gitlab.cern.ch/ai/it-puppet-module-bolt` .

1.4.1 Configuring Bolt

Bolt gives the opportunity to create a configuration file to store and automate the command-line flags you use every time you run Bolt. By default, Puppet-Bolt search first the configuration file in `~/.puppetlabs/bolt/`. To set up a default global configuration file for Bolt, create a `~/`.

`puppetlabs/bolt/bolt.yml` file with global options at the top level of the file. Configure transport specific options for each transport. If a configuration option is set in the configuration file and passed with the corresponding command-line flag, the flag takes precedence.

Your Bolt configuration file can contain global and transport options.

- Global configuration options.
- SSH transport configuration options.
- WinRM transport configuration options.
- PCP transport configuration options.
- Local transport configuration options.
- Log file configuration options.

For more information on the Puppet-Bolt configuration parameters, see [4].

Note: It is possible to change the directory where the configuration file is searched.

1. Using flag –configfile CONFIG_PATH
2. Changing the default_paths function in config.rb

1.4.2 Inventory Config

In Bolt, you can use an inventory file to store information about your nodes. For example, you can organize your nodes into groups or set up connection information for nodes or node groups.

The inventory file is a yaml file stored by default at inventory.yaml inside the Boltdir. At the top level it contains an array of nodes and groups. Each node can have a config, facts, vars, and features specific to that node. Each group can have an array of nodes and a config hash. node. Each group can have an array of nodes, an array of child groups, and can set default config, vars, and features for the entire group. For more information see [4].

1.4.3 Connecting Bolt to PuppetDB

It is possible to connect Puppet-Bolt with PuppetDB, for this it authenticates with PuppetDB through an SSL client certificate or a PE RBAC token.

To configure the Bolt PuppetDB client we just need to add a puppetdb section to the bolt.yml configuration file, see [4].

1.5 Running Puppet-Bolt

Bolt can be used to execute arbitrary commands, scripts, tasks, plans and even to upload files to our remote machines. To know more about running commands, scripts, tasks and plan with Puppet-Bolt see [4] and [11].

1.5.1 Running arbitrary commands

Bolt by default uses SSH for transport. If you can connect to systems remotely, you can use Bolt to run shell commands. It reuses your existing SSH configuration for authentication.

The syntax to run a command against a remote Linux node is the following one:

```
bolt command run <command> —nodes <nodes>
```

Listing 1.3: bolt run command

1.5.2 Running scripts

To run a script across Linux nodes you must include a shebang (#!) line specifying the interpreter, then we use the command:

```
bolt script run <script-name> <script options>
```

Listing 1.4: bolt run script

When you run a script with Bolt, the script is transferred into a temporary directory on the remote system, run on that system, and then deleted. You can run scripts in any language as long as the appropriate interpreter is installed on the remote system. This includes Bash, PowerShell, or Python.

1.5.3 Uploading files

Bolt can be also used to copy files to remote nodes:

```
bolt file upload <SOURCE> <DESTINATION> --nodes <NODE NAME>,<NODE NAME>
```

Listing 1.5: bolt upload file

1.5.4 Running tasks

Tasks are similar to scripts, they can be written in any programming language that runs on your target nodes. However tasks are kept in modules and can have metadata (Task metadata describes the task, validates input, and controls how the task runner executes the task). This allows you to reuse and share them more easily. They can be uploaded and downloaded as Puppet Forge modules, run from GitHub or used locally to organize the commands that are used regularly.

To view the list of tasks that are installed in the current module path:

```
bolt task show
```

Listing 1.6: bolt show tasks

A to view the documentation of a task:

```
bolt task show <TASK NAME>
```

Listing 1.7: bolt task documentation

To run a available task:

```
bolt task run mymodule::task parameter=value --modulepath ./path/to/modules --nodes
    node_name
```

Listing 1.8: bolt task run

1.5.5 Running Plans

Plans are sets of tasks that can be combined with other logic. This allows you to do more complex Task operations, such as running multiple Tasks with one command, computing values for the input for a Task, or running certain Tasks based on results of another Task. You write plans in the Puppet language. And like Tasks, Plans are packaged in modules and can be shared on the Forge.

To discover the list of plans that are installed on the current module path:

```
bolt plan show
```

Listing 1.9: bolt show plans

And as for Task, to view the parameters and other details for a plan:

```
bolt plan show <PLAN NAME>
```

Listing 1.10: bolt show plans details

To run a available plan:

```
bolt plan run mymodule::myplan parameter=value --modulepath ./path/to/modules --nodes
    node_name
```

Listing 1.11: bolt run plan

1.6 Performance Evaluation

As we have said before Mcollective uses publish/subscribe middleware to transport requests between clients and servers. The publish and subscribe operations are done through persistent connections to a middleware broker. That is, Mcollective uses messaging, the connections are all ready in place, messages are signed with SSH key for authentication.

However, Bolt uses SSH as a 'transport layer', which starts a new connection for every Task. SSH authentication overhead is the reason for bad performance.

1.7 Why to change

We highlight the following reasons why replace Mcollective by Puppet-Bolt,

- Puppet-Bolt do not requires a demon to be running in each managed machine. For this reason, installation, configuration and maintenance is more simple.
- The possibility that Bolt offers to write Tasks and Plans in any language and the ability to reuse and share them.
- MCollective will no longer be shipped in Puppet Agent version 6 which is currently due around Fall 2018.

2. Looking for better performances

2.1 Bolt source Code

Bolt is open source, that means we can inspect and modify the code at our whim. After a first inspection of the bolt module, we distinguish the following folders.

```
.
|-- bolt-modules
|   `-- boltlib
|-- exe
|   |-- bolt
|   `-- bolt-inventory-pdb
|-- lib
|   |-- bolt
|   |-- bolt_ext
|   `-- bolt.rb
|-- modules
|   |-- aggregate
|   `-- canary
`-- vendored
    |-- facter
    |-- hiera
    |-- puppet
    `-- require_vendored.rb
```

Figure 2.1: File structure of Puppet-Bolt 0.17.2

- **bolt-modules:** Bolt uses an internal version of Puppet that supports Tasks and Plans, so you do not need to install Puppet. In this directory we can find puppet datatypes
  - result
  - resultset
  - target
  - targetspec

  They can be used in Tasks and Plans.

  We also can find utility functions used in Puppet-Bolt Plans:
  - fail_plan
  - file_upload
  - get_targets
  - run_command
  - run_plan
  - run_script
  - run_task

  - set_var
  - vars

- **exe:** We can find the bolt's executables. bolt is the main executable. It only instance the CLI class that is in lib that allows to parse the input arguments. the CLI class calls the logic that establishes SSH connection with the remote hosts and executes the commands, scripts, Tasks, Plans, etc.
- **lib:** In this folder we can find the main ruby files and modules that allow connection with remote hosts, create files, directories and execute commands, script, Tasks and Plans in remote machines.When bolt is executed the first file of /lib called is cli.rb .
- **modules:** Core functionality such as the aggregate and canary plans included in modules.
- **vendored.**

2.2 Improving Bolt performances

2.2.1 Techniques to improve performance

By observing which techniques are used to improve the performance of Ansible, we can highlight the following techniques to obtain better performance from Bolt, see [1, p. 161-171].

- **SSH Multiplexing and ControlPersist:** When you use SSH multiplexing, then multiple SSH sessions to the same host will share the same TCP connection, so the TCP connection negotiation only happens the first time.
- **Pipelining:** Ansible supports an optimization called pipelining, where it will execute the Python script by piping it to the SSH session instead of copying it. This saves time because it tells Ansible to use one SSH session instead of two.
- **Parallelism:** We will connect to the hosts in parallel to execute the Tasks.
- **Python Mitogen:** Mitogen is a Python library for writing distributed self-replicating programs. An extension to Ansible is included that implements connections over Mitogen, replacing embedded shell invocations with pure-Python equivalents invoked via highly efficient remote procedure calls to persistent interpreters tunnelled over SSH.

Puppet-Bolt already uses the techniques of Parallelism and Pipelining. Indeed, the number of threads to use when executing on remote nodes is set as default to 100. (Can be changed in the configuration file see 1.4.1, section "Global configuration option"). To be completely correct, Bolt does not use the Pipelining technique because it copies the file to the remote host. However, unlike Ansible, it does not create a new connection to copy the file and it is avoiding creating a new connection where the technique of Pipelining in Ansible results in most cases effective.

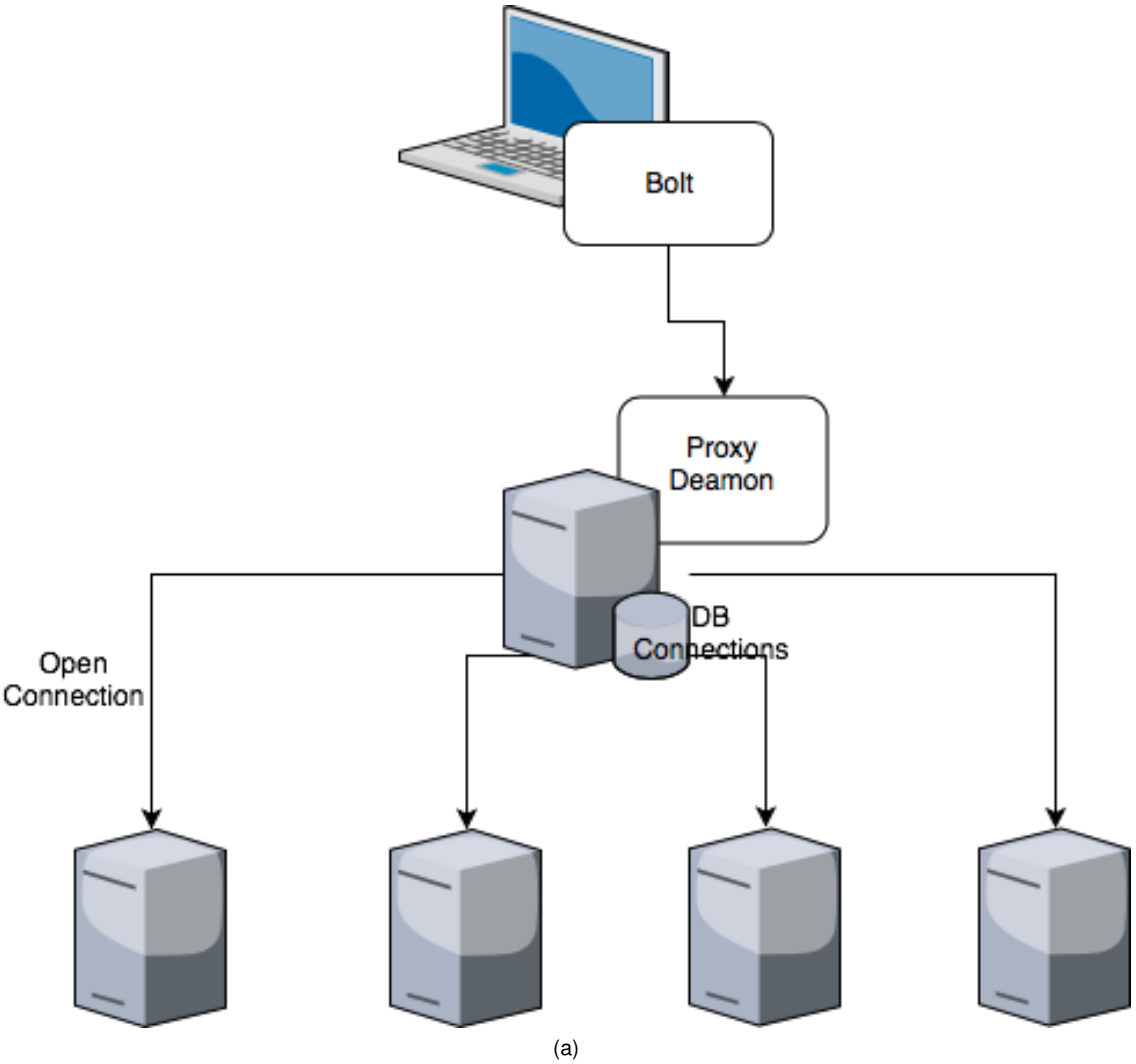
2.2.2 Applying SSH Multiplexing

As we have seen, Puppet-Bolt requires for each Task to start a new SSH connection. Therefore, in most cases it takes more time creating a new SSH connection than running remote code. After executing Bolt the process dies and the ruby object that gave access to the connection is lost. Therefore, when a new bolt command is re-launched, a new connection is created and this again requires the expensive negotiation process. Then to achieve multiplexing we have to launch a background process that keeps the connections alive and who can communicate with the main bolt process (see [9] and [10]).

We have therefore realized this exercise,

we have modified the Bolt code so that it sends the commands that it wants to execute on the remote hosts first to a background process that saves the connections in a hash table. Every time there is a new request, it looks for if it has a connection to the remote host in its database. In case there is already one, it will reuse the connection to execute the action. Once the action has been executed on all remote hosts, it collects all the results and forwards them to the bolt client.

The background process can be executed on the local machine or on an external machine. To more information see `https://gitlab.cern.ch/jdefruto/it-bolt` .



(a)

Figure 2.2: Diagram Bolt Multiplexing 1

A more complex and more optimal architecture would be the following:

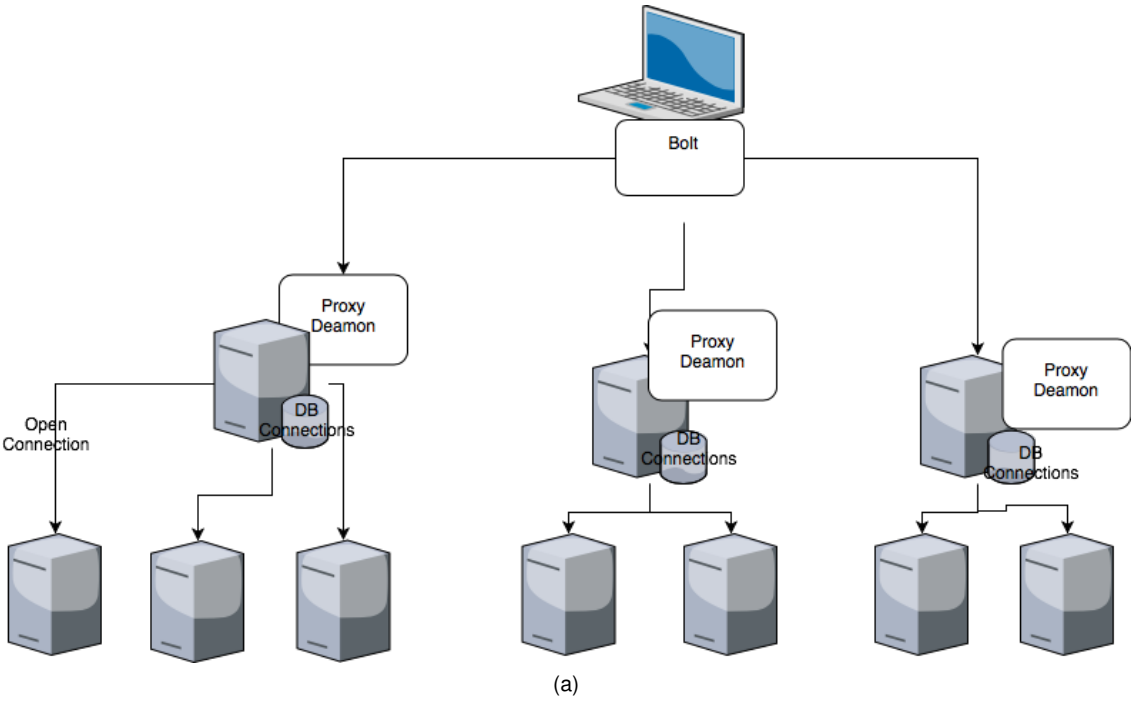


Figure 2.3: Diagram Bolt Multiplexing 2

In other words, each process in the background would have a number of open connections and only the process in the background with an open connection to the remote host would execute the action. However, for routing to be successful, we must implement techniques such as distributed hash table. However this is not a trivial exercise.

With the architecture 2.2a and execution of simple commands, we have obtained the following results

```
[root@playground-jdefruto-01 ~]# netstat -tnpa | grep '137.138.33.140'
[root@playground-jdefruto-01 ~]# time bolt command run 'touch test1.txt' --proxy=playground-jdefruto-01:4913 --nodes playground-jdefruto-03
Started on playground-jdefruto-03...
Finished on playground-jdefruto-03:
Successful on 1 node: playground-jdefruto-03
Ran on 1 node in 0.55 seconds

real    0m0.886s
user    0m0.316s
sys     0m0.062s
[root@playground-jdefruto-01 ~]# time bolt command run 'touch test2.txt' --proxy=playground-jdefruto-01:4913 --nodes playground-jdefruto-03
Started on playground-jdefruto-03...
Finished on playground-jdefruto-03:
Successful on 1 node: playground-jdefruto-03
Ran on 1 node in 0.08 seconds

real    0m0.403s
user    0m0.315s
sys     0m0.053s
[root@playground-jdefruto-01 ~]# netstat -tnpa | grep '137.138.33.140'
tcp        0      0 137.138.33.82:50998     137.138.33.140:22       ESTABLISHED 1694/ruby
[root@playground-jdefruto-01 ~]#
```

(a) multiplexing run command 1

```
[root@playground-jdefruto-01 ~]# time bolt command run 'echo "hello world" > test2.txt' --proxy=playground-jdefruto-01:4913 --nodes playground-jdefruto-03
Started on playground-jdefruto-03...
Finished on playground-jdefruto-03:
Successful on 1 node: playground-jdefruto-03
Ran on 1 node in 0.09 seconds

real    0m0.429s
user    0m0.318s
sys     0m0.068s
```

(b) multiplexing run command 2

```
[root@playground-jdefruto-03 ~]# ls
anaconda-ks.cfg  anaconda-post.log  init.pp  it-puppet-module-bolt  mco_rpc_examples  original-ks.cfg  test1.txt  test2.txt  thanks.sh  typescript
[root@playground-jdefruto-03 ~]# cat test2.txt
hello world
```

(c) multiplexing run command results

Figure 2.4: Example of multiplexing a command

2.2.3 Some words about Python Mitogen.

Mitogen is a Python library for writing distributed self-replicating programs. However the extension of Ansible to make connections over Mitogen does not have today, as much magic as it may seem after a first look. Nevertheless, it aggressively optimizes the creation of SSH connections, something complicated to do it correctly. As documentation says the Mitogen extension for Ansible get its effectiveness by ensuring that:

- One connection is used per target. This is equivalente to an optimized combination of multiplexing and pipelining.
- A single network roundtrip is used to execute a step whose code already exists in RAM on the target.
- Processes are aggressively reused. This avoids the cost of invoking Python and recompiling imports.
- Code is ephemerally cached in RAM.
- Fewer writes to the target filesystem occur.

The promising feature “Connection Delegation” is not yet complete. Connection Delegation should enables Ansible to use one or more intermediary machines to reach a target machine or container, with connections and code uploads deduplicated at each hop in the path. For an Ansible run against many containers on one target host, only one SSH connection to the target need exist, and module code need only be uploaded once on that connection.

Python Mitogen is the perfect example of how to minimize the creation of connections to the maximum. However, it is a library that contains more than 14,000 lines of code and is called by Ansible in a complex way. Therefore, understanding the Mitogen code to reproduce it in Bolt (since Bolt is Ruby code) is not a trivial task. And maybe exaggerated considering that the Bolt '0.17.2' version is less than 5000 lines of code.

3. Conclusions

Puppet-Bolt is a great tool that provides all the features that Mcollective offers and more. However, due to the size of the CERN computer network, the installation and (maybe the software) must be perfected, to achieve the execution of tasks in the entire network in real time. Since you can not run a command or script on a remote host faster, the solution is to reduce connection establishment and data transfer time. After experimenting with the Multiplexing option, we managed to execute the command in half the time. This opens the door to thinking optimistically that with an architecture like the one described in the diagram 2.3a, we will get the same benefits as with Mcollective with the advantages that Puppet-Bolt has.

I would like to add that after having navigated a bit in the files of Puppet-Bolt, Ansible and Python-Mitogen, I think it is realistic to think that we can change the Puppet-Bolt code to achieve much higher execution times. I also think that this exercise can be very interesting and fun as it has been so far.

Thank you very much Steve for all that fun.

Bibliography

[1] HOCHSTEIN, L. *Ansible Up & Running*. O'Reilly, 2015. 8

[2] *IN2P Computing Center* https://forge.puppet.com/ccin2p3/bolt 3

[3] *Marionette. Collective. Official documentation* https://puppet.com/docs/mcollective/current/index.html

[4] *Puppet. Bolt. Official documentation* https://puppet.com/docs/bolt/0.x/bolt.html 4

[5] PUPULI, V. *Mcollective Puppet Module*. https://forge.puppet.com/puppet/mcollective 2

[6] *Python Mitogen. Official documentation* https://mitogen.readthedocs.io/en/stable/

[7] RHETT, J. *Learning Mcollective*. O'Reilly, 2014. 2

[8] RHETT, J. *Learning Puppet 4.*. O'Reilly, 2016.

[9] *Speed Up. Ruby-ssh* https://stackoverflow.com/questions/24708422/speed-up-ruby-ssh-handshake 8

[10] *Store ssh connections*. https://stackoverflow.com/questions/4762522/store-ssh-connections-in-rails 8

[11] *Task hands on lab* https://github.com/puppetlabs/tasks-hands-on-lab 4

A. Interesting/important changes in Puppet-Bolt for the CERN in later versions to '0.17.2'

We highlight the following changes in puppet-bolt functionalities for the versions after the '0.17.2'.

Version	Main Changes
0.21.1	• Support for Kerberos authentication
0.20.6	• PuppetDB CLI configuration files • Escalate privileges
0.20.0	• puppetdb_query added to plan language
0.18.0	• Breaking change: Bolt requires Ruby 2.3 or higher.

Table A.1: Puppet-Bolt new interesting features and Breaking Changes versions $>=$ 0.17.2.