Journal of Physics: Conference Series **664** (2015) 052026 doi:10.1088/1742-6596/664/5/052026

Integrating Puppet and Gitolite to provide a novel solution for scalable system management at the MPPMU Tier2 centre

C Delle Fratte[1], J A Kennedy[2], S Kluth[1] and L Mazzaferro[1]
[1]Max Planck Institut für Physik, Föhringer Ring 6, 80805 München
[2] Rechenzentrum Garching, Giessenbachstraße 2, 85748 Garching

Abstract. In a grid computing infrastructure tasks such as continuous upgrades, services installations and software deployments are part of an admins daily work. In such an environment tools to help with the management, provisioning and monitoring of the deployed systems and services have become crucial.

As experiments such as the LHC increase in scale, the computing infrastructure also becomes larger and more complex. Moreover, today's admins increasingly work within teams that share responsibilities and tasks. Such a scaled up situation requires tools that not only simplify the workload on administrators but also enable them to work seamlessly in teams.

In this paper will be presented our experience from managing the Max Planck Institute Tier2 using Puppet and Gitolite in a cooperative way to help the system administrator in their daily work.

In addition to describing the Puppet-Gitolite system, best practices and customizations will also be shown.

1. Introduction

The computing centers supporting research are becoming day by day more complex and bigger with many different services to deploy, configure and maintain.

Moreover the experiments and researchers communities often require a dedicated computing environment which, as a consequence, need specific competences from the sysadmin. Therefore the sysadmin's job is becoming very specialized and in a middle or large centers is not uncommon to find several administrators working in the same physical place but lacking of the instruments to share their knowledge and experiences. Such situation can be summarized in the figure 1.

In the last few years many tools have been developed to support the system administrator's daily work. They are able to take care of the provisioning, deployment, monitoring and management of either full nodes and single services. These solutions increase the reliability, availability of the services, providing also a certain degree of automation, and increasing the reproducibility of the operations.

The goal of the solution presented in this document is to provide a system where different system administrators can cooperate without interfering each other, using one centralized service for provisioning, monitoring and automation.

This solution is based on the integration between **Puppet**[1], one of the most diffused provisioning tool, supported by **Foreman** dashboard[2] and **Gitolite**, a service used to store git.

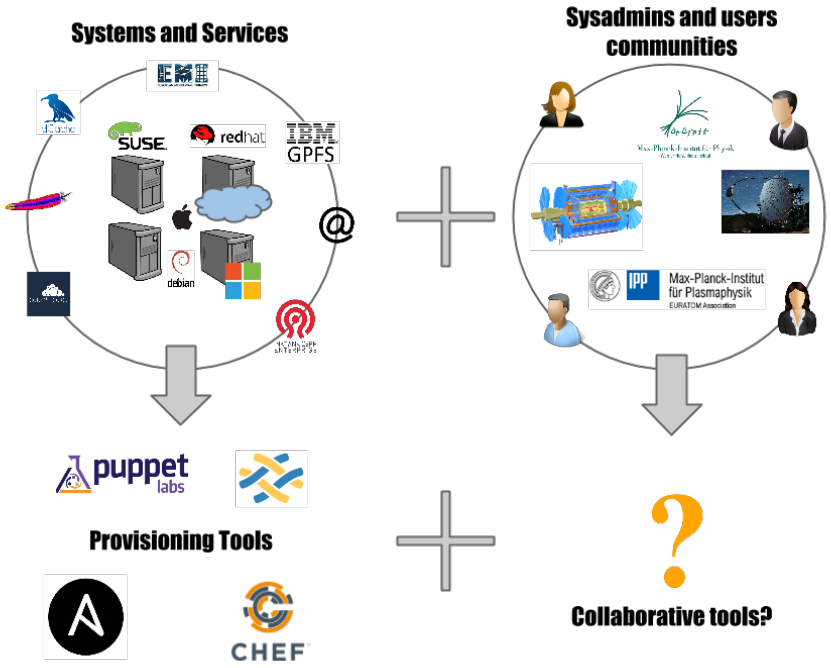

Figure 1. A schematic view of the complexity of the work environment in a modern computing center. While many tools for automation and provisioning are present, how to deal with a complex an heterogeneous organization is still an open issue.

2. The Ingredients

In this section will be briefly described the each tool used for the integration and its role in the infrastructure.

2.1. Puppet

Puppet[1] is a modern configuration management system which automates tasks that sysadmins perform often. It works with virtual machines and physical servers and can easily scale up to thousand of nodes.

The simplest infrastructure is made of: a central server acting as `puppet master`, and the client nodes running the `puppet agent` tool. All the communications between the `agents` and the `master` are encrypted using the SSL protocol, for this reason the default `master` deployment provides an internal Certification Authority (CA) able to sign the nodes certificates.

To work properly, the `master` requires complete access to the nodes definitions, encoded in files called `manifests` and written in a simple declarative language developed by **PuppetLabs**. These `manifests` can be then organized in `modules`, used to have a better view of the services to be deployed. Moreover, the code supports the use of variables: the values substitution can be performed by the sysadmin as well as by the `agent` tool. As a consequence the code becomes more general, dynamic and reusable.

The schema on figure 2 describes the complete **Puppet** life-cycle:

1. **Define:** the sysadmin defines the nodes' *states* in `manifests` and `modules`.
2. **Simulate:** the changes are simulated on the nodes before applying them directly on the node/s, a report is produced with the results of the simulation/s. This step is, in any case, optional.

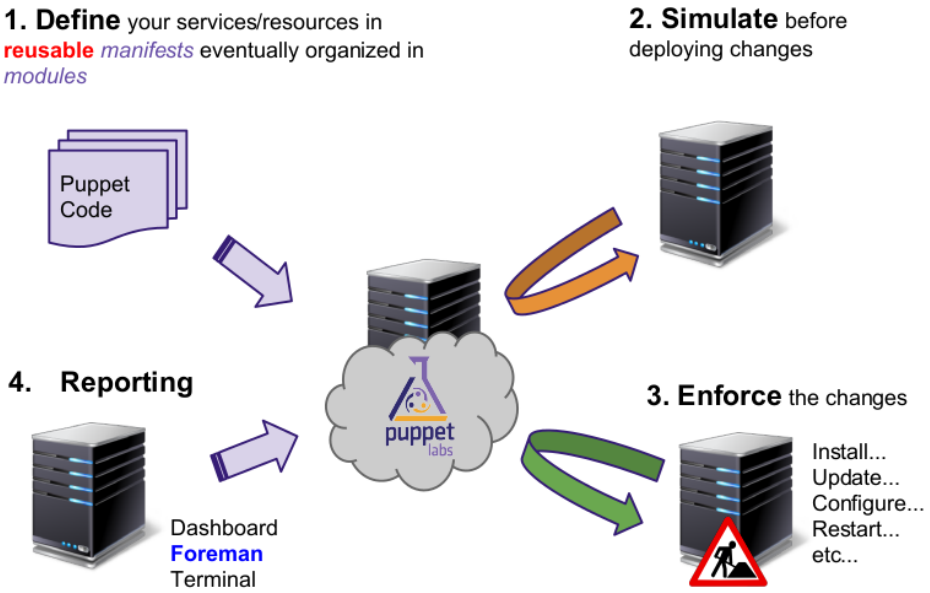

Figure 2. In this figure a simple schema representing the complete Puppet Life Cycle.

3. **Enforce:** the changes, if present, can now be safely applied on the node. The sysadmin can manually run the `agent` tool or it can be configured to work automatically. It works in synergy with the `master` to achieve the desired state.
4. **Report:** a report with a summary of the operations performed is produced and can be collected to be analyzed later. In our case the reports are collected by the **Foreman** interface and can be accessed by the **Foreman dashboard**.

Another key aspect of **Puppet** is the organization of the code in `environments` which are mainly used to keep separated different groups of nodes. Each client can be attached only to one `environment` per session. Inside each environment are saved all the `manifests` and the `modules` to be applied. A typical solution is the creation of three `environments`: *production*, *development* and *testing*. In this case the sysadmin can easily develop the modules and safely test them on the nodes attached to *development* and *testing* environment, not involving the machines in production. Based on the same principle the code coming from the different sysadmins can be isolated in dedicated *environments*. An extensive explanation will be given in section 3.

The *PuppetLabs* products has been selected because of their stability, simplicity to deploy and maintain and the huge community involved in the project which continuously provides `modules`, bug fixes and supports. Finally an enterprise version is also available, which can be considered a better solution when the complexity of the infrastructure increase.

2.2. Foreman

Foreman[2] is an open source management tool used, in our case, on top of **Puppet**.

It takes care of managing users, monitoring nodes status and it acts as an *External Node Classifier* which means it is able, in sinergy with **Puppet**, to register new nodes, define their states and configure **Puppet** `manifests`.

It is worth to note that by default, **Puppet** doesn't provide natively any kind of users management system. Every sysadmin who has the rights to access to the nodes and to write `manifests` can deploy changes on the machines. **Foreman** provides a system for managing the

users permissions, retrieving information on the nodes and defining their state. This assumes a great importance when you want to define isolated work environments for different sysadmins.

2.3. Gitolite

Gitolite[3], is a tool to host different `git`[4] repositories providing and allowing a fine grained access control.

The **Puppet** `manifests` are stored in `git` repositories hosted inside the same server where **Foreman** and the `puppet master` are running. In addition to storing the repositories, **Gitolite** is used as a secure communication channel, based on the well known `ssh` tool, between the users and the `puppet master`, as described later in the section 3.

A web interface, called **gitweb**, has been also installed to enable the users to browse their code.

This strategy permits to keep the different environments, possibly belonging to different sysadmins, isolated and provides all the features of a classic revision control system: commits' log, history of the changes, etc etc.

A key feature present **Gitolite** is the possibility to define actions which are performed when an certain event is triggered. Called `hooks` and can be installed by the **Gitolite** administrator and support different types of languages, including `bash`, `python` and `perl`. The integration will make an extensive use of this feature as explained in section 3.

We chose **Gitolite** because it is mature, easy to install and maintain and most important it is highly customizable.

3. The Integration

On figure 3 it is presented the infrastructure installed at the Rechenzentrum Garching (**RZG**) and the workflow coming from the `manifests` definitions up to the changes deployment on the nodes.

All the services described in the section 2, except for the `puppet agent`, runs on the same host organized in different **Docker** containers following the *micro-services* philosophy. This solution has been crucial during the first pilot phase when it helped to create an easy to test work environment.

Looking at figure 3 and focusing the attention on one of the users, e.g. **WN**s administrator, we can easily understand the entire infrastructure:

1. the sysadmin defines a new *state* writing one or more `manifests`. *PuppetLabs* and the users community provides already `manifests` for the most commons services; they can be easily downloaded and integrated in the local environment. The code must be saved into the a `git` repository representing the **WNs**'s `environment`.
2. Once the `manifests` are ready, the changes can be committed and pushed to **gitolite**. The service takes care to check the user permissions and to store the repository. Via the `gitweb` interface the user can easily review the last changes and browse the commit messages.
3. Two `hooks`, see section 2.3 and figure 4, are here defined. They runs in sequence as soon as the repository as been pushed.
   The first `hook` creates, if missing, a new `environment`, e.g. **WN**, accessible by the `puppet master` and names as the repository. Then it copies the code inside the `environment` just created. Otherwise, it updates the `environment` already present. In this way, **git** and **gitolite** work in synergy as secure communication channel between the sysadmin and the `puppet master`.
   The second `hook` checks for changes on the `manifest` files and if they are present, it runs an update on the **Foreman** service. As a consequence **Foreman** will be always aware of

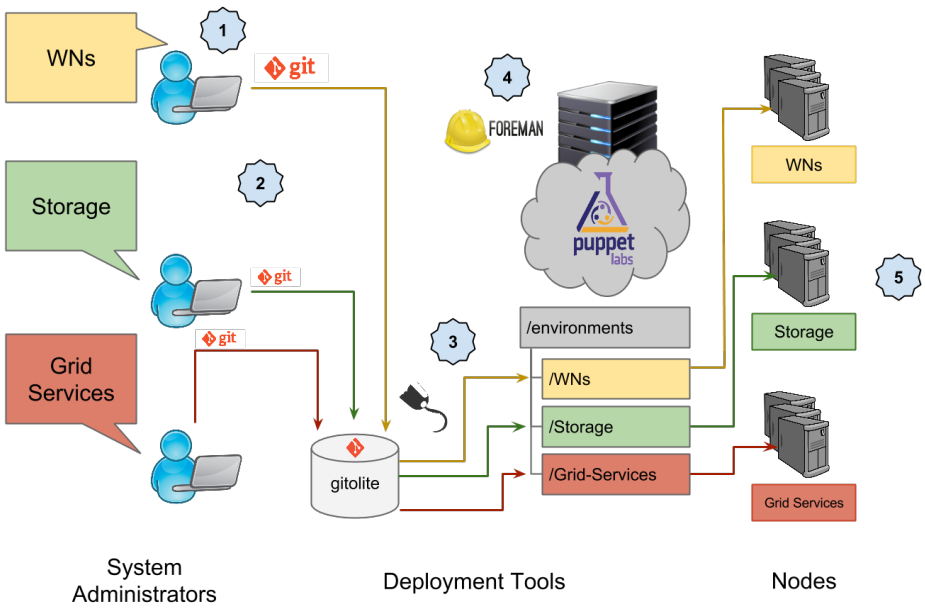

Figure 3. An an complete overview of **Foreman-Puppet-Gitolite** infrastructure at Rechenzentrum Garching.

the last changes present in the code. This `hook` automates an action which can be also performed by the user via the **Foreman** web interface.

4. the sysadmin can now apply the last configurations, setting up the `manifests` variables and attaching the `modules` to the nodes via the **Foreman** interface: these actions defines the use of **Foreman** as **ENC**. Details in figure 5.
   It is worth to note that until this moment nothing is still happened inside the nodes. The **Foreman** interface restricts the sysadmins to their own `environment` and nodes.
5. the nodes attached to the specific `environment`, e.g. **WNs**, can now retrieve the changes. This action can be handled directly by the sysadmin or automatically managed by the `puppet agent` if configured in a such way.

In this configuration different system administrators can work in parallel on different `environments` using only one centralized server without interfering with each other. They can, indeed, write their modules on their workstations, test them before commit.

Moreover the integration of these three system has many other benefits:

- **Puppet** ensures a better scalability of the entire infrastructure also allowing the sysadmin to be more focused on the design of the system.
- **Foreman** and **Gitolite** ensures the code and the nodes isolation for a safe multi-users use of **Puppet** increasing also the security thanks to the lack of direct interactions with the **Puppet** `master`.
- **Gitolite** provides a revision control system useful for users cooperation and simplify the code sharing but having the data safely stored in a local storage.

4. Best Practices

As already extensively explained in the previous sections, the **Puppet** `manifests`, which describes the node's state, are written in specific language defined by **PuppetLabs**. This means

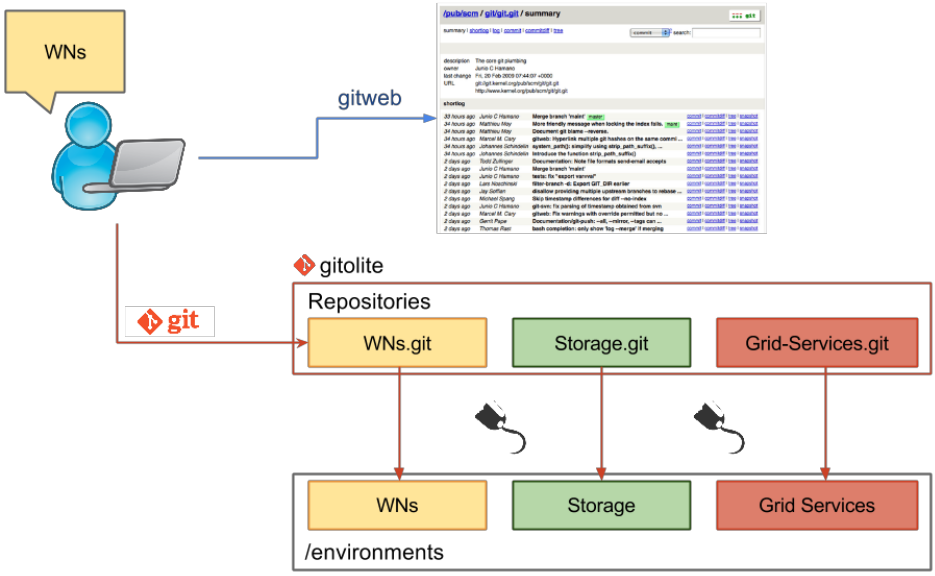

Figure 4. Details about the role played by **Gitolite** and **gitweb**. It is visible as the last changes are automatically pushed from repositories to the **Puppet** `environments`. The sysadmins don't interact directly with the `puppet master` and are restricted to their own repos.

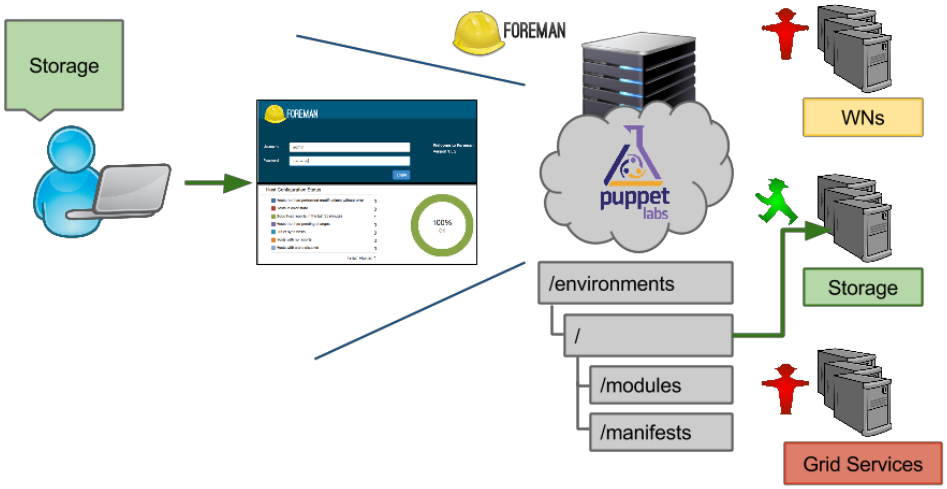

Figure 5. This figure shows as the **Foreman** interface used as *External Node Classifier* restricts the users to define and deploy changes only to their own nodes.

that the new users need to learn it in order to use properly this tool. From our experience this can be an hard task if not handled in the right way.

Here some recommendations:

- **start with easy tasks**, migrating or defining complex infrastructures from scratch can be hard. **Puppet** allows the user to manage partially the node, without interfering with the service deployed in different ways.
- **Search inside *Puppet forge***[6], the official repositories site, before to start to write code. The most common services have their own **Puppet** `modules` already present. They can be downloaded and installed for free. Furthermore **Gitolite** allows the sysadmins to safely share their code, in this way the same `manifests` can be reused by other administrators inside the same structure saving time and resources.
- **Test the `manifests` and `modules` before applying them**. This action can be performed also directly on the nodes with a running a *simulation*, as described in section 2.1. The best thing to do is to recreate a test environment which is a copy of the production one. A tool which helps to achieve this goal is **Vagrant**[7]: it provides a single interface to the most common hypervisors for VMs deployment and provisioning. It supports different tools for provisioning, including **Puppet**. As a consequence, the sysadmin is able to run, also on his workstation, a test session of the **Puppet** code, using VMs similar to the production nodes.

5. Conclusions and next steps

By combining **Puppet, Foreman** and **Gitolite** we have managed to provide a solution which enables different sysadmins to use one centralized `puppet master` without interfering each other and increasing and simplifying the cooperation.

The full infrastructure is running at Rechenzentrum Garching since more than six months, successfully supporting 3 sysadmins which are managing 3 completely different projects:

- the local storage based on **dCache**.
- the ATLAS tier2 grid services and worker nodes.
- the **OwnCloud** local installation.

The total number of nodes connected is about 150.
The next steps will mainly involve two different areas:

- Firstly ensure the scalability of the system when more users and nodes are added. This is especially related to the `puppet master` tool. Big improvements are expected from the next **Puppet** release, when a big software refactoring has be announced. Scaling problems are not expected to become from **Foreman** and **Gitolite**.
- a test environment based on **Jenkins**[8] and **Vagrant** is under study to provide a system to automatically test the **Puppet** code before putting it in production. This should be integrate in the **Gitolite** service.

References

[1] Puppet Home Page, `https://puppetlabs.com/`
[2] Foreman Home Page, `http://theforeman.org/`
[3] Gitolite Home Page, `http://gitolite.com/gitolite/index.html`
[4] Git Home Page, `http://git-scm.com/`
[5] Docker Home Page, `https://www.docker.com/`
[6] Puppet Forge Home Page, `https://forge.puppetlabs.com/`
[7] Vagrant Home Page, `https://www.vagrantup.com/`
[8] Jenkins Home Page, `https://jenkins-ci.org/`