SpaceOps Conferences
28 May - 1 June 2018, 2018, Marseille, France
2018 SpaceOps Conference

10.2514/6.2018-2337

Ground Control Segment automated deployment and configuration with ANSIBLE and GIT

Rémi PIEPLU[1]
Centre National d'Etudes Spatiales, Toulouse, 31400, France, remi.pieplu@cnes.fr

In this paper, we present our method to simplify the deployment of a ground control segment which relies on several dozens of machines divided into several validation platforms. We need to automatically instantiate those platforms from a common and centralized configuration. For this purpose, we will describe how we setup a unified configuration management by an Infrastructure as Code approach. We will expose the benefits of this innovative method.

The deployment of a ground control segment is made up of several stages: virtual machine creation, operating system installation, computer security and installation of space oriented software. Usually, this deployment is managed by complex operational documentation and homemade scripts adapted to specific needs. A reinstallation can take days and requires a lot of systems engineers. Consequently, a reinstallation is considered as a major risk and therefore rarely done.

Cloud oriented approaches intensively use automation to deploy important infrastructures. Methods and tools from these approaches could be interesting for our use case. Infrastructure as Code is one of these methods. It is a way of managing and provisioning computer data centers through machine-readable definition files. It involves the same types of processes used during development to deploy infrastructures and components. In our case, we use a version control system named GIT to track modifications. To achieve the deployment, we rely on a configuration management software named Ansible.

GIT is the most popular decentralized version control system for tracking changes in files and coordinating work inside the team.

Ansible is an open source configuration management software recently bought by Redhat that enables us to deploy and manage a large set of interconnected machines (physical or virtual). It is based on the popular protocol Secure Shell (SSH) and the largely deployed interpreted language Python. Moreover, it does not require a specific agent installed on each client machine. The definition files describing the different actions managed by Ansible are written in a way that is close to natural language and easily readable. Their organization simplifies the reusability of code. The deployment is idempotent. It means that it can be applied several times without changing the result beyond the initial application. That allows us, when a modification has to be made on the ground control segment, to modify definition files instead of working on a specific server and to reapply the whole configuration via Ansible.

[1] Ground Engineer, 18 avenue Edouard Belin 31400 Toulouse

This way of thinking necessarily involves changes in working habits of the operational team. In this paper, we will describe how we handle these changes in our team with specific training session and specific guide. We will also present the gain in efficiency and consistency.

Furthermore, we highlight three main benefits. The first one is a technical benefit as we are able to deploy a ground control segment for a specific test in less than two hours. The second one is a benefit in term of quality as the whole configuration of the ground control segment is versioned and it can be audited by the Quality Assurance Manager. The last one is a reusability benefits as the definition files could easily be reused for further missions

I. Background

The ground control segment is part of a project of an earth observation constellation. The ground control segment must be operational by the end of 2018. The integration and deployment phase will continue until the launch phase. In this part, we will describe the different elements that drive our choice of an automated deployment.

A. Context of the project

1. Team organization

Integration is managed internally at CNES. The team is composed of "pure" system engineers specialized in virtualization, storage, network and system and by operational engineers with knowledge on how to operate a ground control segment and how to orchestrate the operational chronology.

Technical choices are made by the operational team. Concretely, the team that made the integration and the deployment of the control center will operate it in the LEOP phase and during the routine phase. It insures that technical choices made during integration phase will be adapted to the operational phase.

Modifications of the ground control segment are discussed in short iterations inside the team. We want to quickly implement new solutions, refactor our infrastructure code or implement new needs coming from onboard operational engineer or mission operational engineer.

2. Schedule

The schedule is very tight to make the ground control segment operational. Every week, a set of validation platforms must be ready, each for performing onboard test procedures. Major system tests happen every month. This test rate represents an important pressure on the team to setup correctly a platform without errors.

3. Validation platforms

In order to fit within the schedule, we need to parallelize the tests on the ground control segment. So we need at least ten validations platforms and one operational platform. These platforms must be identical in terms of functionality, deployed in the same way and with the same performance. It is clearly not possible to do it manually.

4. Computer security requirements

Computer security requirements are high on this project. These requirements must be applied on all the validation platforms without exception. Audits are planned during the integration of the ground control segment to verify the implementation of the security requirements. So the computer security shall be treated with a particular attention. It must be part of the deployment process.

B. Architecture

The objective here is to give a high level overview of the architecture. The detailed description of this architecture is not the purpose of this article.

1. Hardware architecture

Our hardware infrastructure is based on a VMware Cluster managed by a VCenter. It allows us to distribute the creation of virtual machines across different hypervisors. The VCenter provides also an API to programmatically create, delete, and modify Virtual Machines.

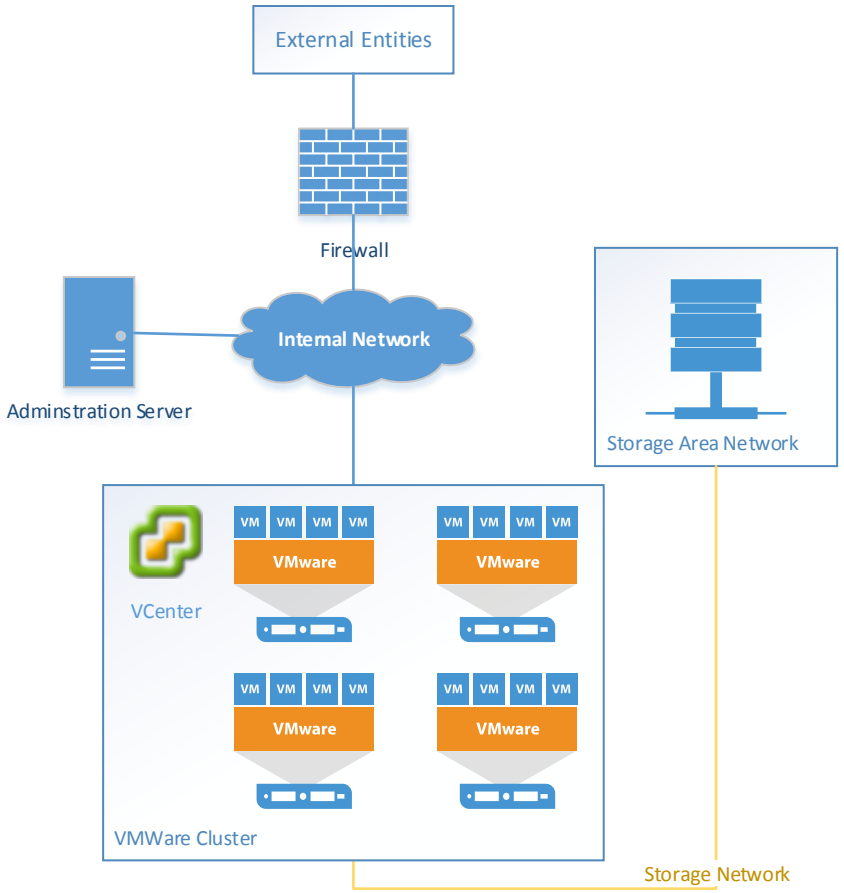

Figure 1. High level hardware architecture

2. Software Architecture

The Common Services Platform provides authentication services and monitoring services to each Validation platform. The authentication system acts as a domain controller. Each host must be enrolled inside this domain. Each host must be registered in the monitoring system.

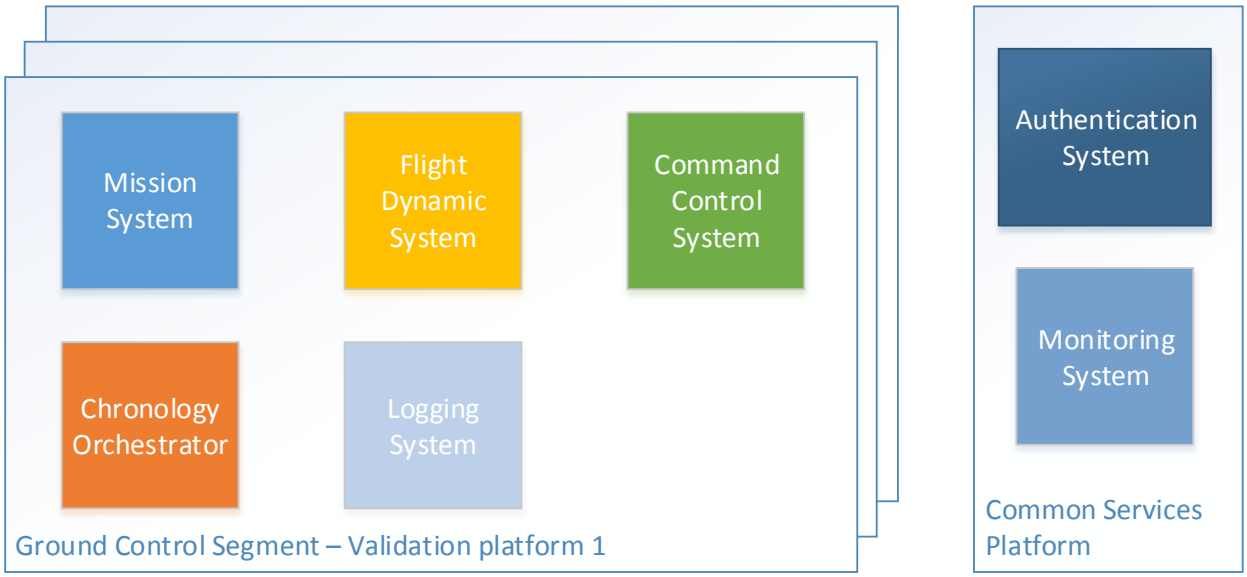

Figure 2. High level software architecture

A validation platform is an auto sufficient platform which means it provides all the functionalities of the ground control segment. For example, one validation platform can be used during a Flight dynamic test while we are testing connection to station with the command control system on another validation platform. In the table below, we describe briefly their functionalities and the number of virtual machines required for running each system.

System	**Description**	**Number of Virtual machines required**
Mission System	Manage the mission of the satellite and collect requests from end users	2
Flight Dynamic System	Assure the performance, stability, and control of the satellite	2
Command Control System	Monitor and control the satellite	3
Chronology orchestrator System	Orchestrate the activity in the ground control segment following the operational chronology	1
Logging System	Collect operational logs and store them in a proper way	1
Total		9

Table 1. Number of Virtual Machines per platform

In total we have 90 Virtual Machines (10 platforms with 9 Virtual Machines) to create, install and configure in parallel.

C. Requirements

We need a way to instantiate on demand from scratch a ground control center for a designated validation platform. Our requirements are the following:

- It shall be fast, reliable, repeatable with predictable outcomes. For example, two weeks to install a test environment is not acceptable due to the tight schedule.
- It shall interact with our virtualization infrastructure.
- It shall permit to deploy Linux Operating System with a specific security policy.
- It shall permit to install easily any type of software, even legacy software.
- It shall produce consistent environments. Deployment in validation will use the same process as production.
- It shall deploy all operational data (Satellite Database, Onboard procedure, Synoptics to display telemetry, …) in a secure way.

The first idea should be to write complex operational documentation and homemade scripts adapted to specific needs. But this method we will not cover all uses cases and we will have difficulties to maintain it.

To resolve this problem, Cloud approaches are very interesting. Massive deployments of virtual machines with complex application runtime are a common practice. Configuration management tools are the key to maintain consistency with an architecture composed of hundred Virtual Machines.

We can also make a list of what characterizes the operational state of the ground control.

- The hardware configuration of each Virtual Machine
- The configuration and the hardening of the operating system
- The installed version of each operational system part of the ground control segment
- The configuration of each operational system
- The version of each operational data used by the ground control segment

Changes on these elements must be tracked and kept in a Version Control System.

II. Automation is the solution

A. What is Ansible?

Ansible is an open source configuration management software recently bought by Redhat that enables us to deploy and manage a large set of interconnected machines (physical or virtual).

1. State-driven resource Model

Ansible features a state-driven resource model that describes the desired state of computer systems and services, not the paths to get them to this state. No matter what state a system is in, Ansible understands how to transform it to the desired state. This allows reliable and repeatable IT infrastructure configuration, avoiding the potential failures from scripting and script-based solutions that describe explicit and often irreversible actions rather than the end goal.

2. Secure and Agentless architecture

Ansible relies on the most secure remote access system available as its default transport layer: OpenSSH.

Moreover, Ansible does not require any remote agents. Ansible delivers all modules to remote systems and executes tasks, as needed, to enact the desired configuration. These modules run with user-supplied credentials, including support for sudo and even Kerberos and clean up after themselves when complete. Ansible does not require root login privileges, specific SSH keys, or dedicated users and respects the security model of the system under management.

As a result, Ansible has a very low attack surface area and is quite easy to deploy into new environments.

3. Language

Ansible configurations are simple data descriptions of your infrastructure (both human-readable and machine-parsable). Everyone in our team will be able to understand the meaning of each configuration task. New team members will be able to quickly dive in.

4. Ansible concepts

Ansible is based on several concepts to deploy a configuration on specific virtual machines. Here is a non-exhaustive table of Ansible concepts.

Concepts	Description	Example

Inventory	It specifies the environment Ansible operates in. This is text files in an INI format containing the list of the hosts or groups of hosts. For example, an inventory is defined for each validation platform or operational platform.	`[Webserver]`
		`server1.example.com`
		`server2.example.com`
		
		`[Database]`
		`db1.example.com`
		`db2.example.com`

Module	A module is a small program written in Python that executes an action on the remote machine. There are a lot of modules provided by Ansible. It is possible to write custom ones.	Module *service* enables to change the state of a init.d service. Module *copy* enables to copy a file from the administration server to the remote machine. Module *vmware_guest* enables to create Virtual machines on a VMWare Infrastructure.

Task	A task is a parametrized call to a module.	`- name : Start HTTP server`
		`service:`
		`name: "{{ httpd_service_name }}"`
		`state: started`
		This task insures that the service is started.

Variable	Variables can be defined in different locations of the Ansible configuration (inventory, Playbook, Role, …) Variables are the key to parametrize a deployment between different inventories.	The software version should be defined per inventory. That allows having different version of the software installed depending on the inventory (production or validation). The value of an installation path should be a variable defined directly in the task or in the role. It may not change depending on the inventory.
Role	A role is a collection of tasks and is focused on one very specific goal. A role can be reused in different playbooks.	For example, a role to configure a web server or a specific application.
Playbook	It's a YAML file that defines sequences of tasks to	`- hosts : WebServer`

	be executed on a group of hosts previously defined in an inventory. It contains roles, tasks, modules and variables.	`tasks:`
		`  - name : Start HTTP server`
		`  service:`
		`  name: "{{ httpd_service_name }}"`
		`  state: started`
		
		`- hosts : Database`
		`  roles:`
		`  - role: InstallMariaDB`

Table 2. List of Ansible concepts

To launch the execution of a playbook on a set of hosts is straightforward. Here is an example to deploy a ground control segment on an operational platform:

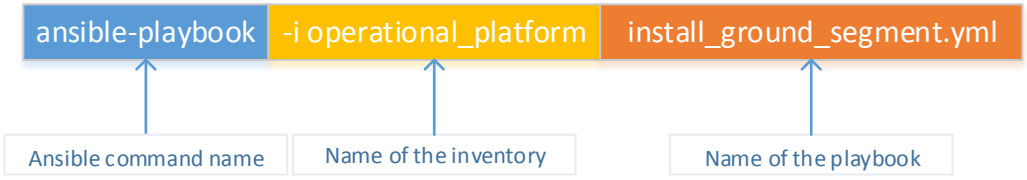

Figure 3. ansible-playbook command

During the execution of an Ansible playbook, the Ansible management server will connect via SSH on each host defined in the inventory and listed in the playbook and remotely execute each task defined in the playbook. Tasks are executed in parallel. With this mechanism, we can quickly apply a modification on a set of hosts.

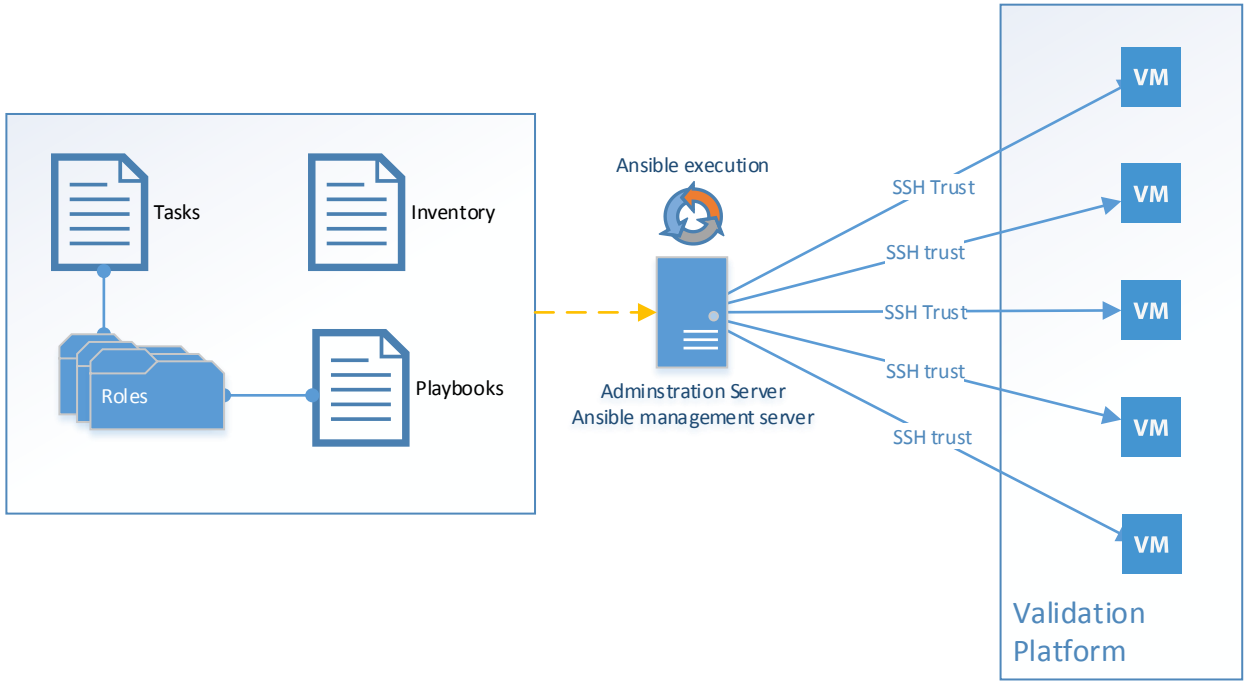

Figure 4. Ansible playbook execution

B. Automatic deployment implementation

In this part, we describe all the steps to deploy a ground control segment from scratch. To start, we define the inventory structure based on our software and hardware architecture. We create inventories as much as we have

validation platforms. Inventory structure is the same between operational and validation platforms. It guarantees consistency of the deployment.

Here is a table with the inventories.

Operational inventory	Validation inventories
[Mission] mission1.operational.example.com mission2.operational.example.com [Fds] fds1.operational.example.com fds2.operational.example.com [CCC] ccc1.operational.example.com ccc2.operational.example.com ccc3.operational.example.com [Logging] log.operational.example.com [Chrono] chrono.operational.example.com	[Mission] mission1.validation0x.example.com mission2.validation0x.example.com [Fds] fds1.validation0x.example.com fds2.validation0x.example.com [CCC] ccc1.validation0x.example.com ccc2.validation0x.example.com ccc3.validation0x.example.com [Logging] log.validation0x.example.com [Chrono] chrono.validation0x.example.com

Table 3. Example of inventories

We decide to implement every step with a role. It is a best practice and the master playbook will be more understandable.

We identify three parts to fully deploy a ground control segment:

- Infrastructure deployment: Infrastructure as Code will be used for this part.
- Space oriented systems deployment
- Operational data deployment

The translation in Ansible Playbook is straightforward:

```
---
#Master playbook install_ground_segment.yml
- include: infrastructure_deployment.yml
- include: application_deployment.yml
- include: operational_data_deployment.yml
```

Table 4. Master Playbook

1. *Infrastructure deployment*

At this step Virtual Machines are not created or have been previously removed. In fact, when we want to reinstall a validation platform we delete all hosts present in this platform. This ensures that we have a clean environment. Tasks defined below are delegated because Virtual Machines are not yet created and we cannot connect to it through SSH. It means that the execution of tasks is delegated to another host.

Here is the list of tasks:

- Create a DNS entry for each host and enroll it in the domain. After this step, hosts are known from the Single Sign On System provided by the authentication service. This step is required to connect through SSH
- Register Virtual Machines in the monitoring server
- Register Virtual machines in the Operating System network installer with Kickstart/PXE boot
- Create and power up Virtual Machines using the VCenter API
- Wait until the Operating System is installed on Virtual Machines
- Launch the hardening and common component installation (not delegated)

The resulting Ansible playbook is presented below:

```
---
#Playbook Infrastructure Deployment
#First Play
- hosts:
  - Mission
  - Fds
  - CCC
  - Logging
  - Chrono
  # No connection to the hosts via SSH
  #Only delegated actions
  gather_facts: no
  vars:
   #List of servers that will execute tasks by delegation
    authentication_server: "auth.example.com"
    monitoring_server: "monitoring.example.com"
    installation_server: "localhost"
    vcenter_server: "vcenter.example.com"
    timeout_install: 1200
  roles:
    #Enroll the VM in the DNS and in the domain
    - role: domainEnrollment
      delegate_to: "{{ authentication_server }}"

    #Add the VM in monitoring server
    - role: monitoringAdd
      delegate_to: "{{ monitoring_server }}"

    #Create the VM in the VMware Cluster
    - role: vmCreation
      vcenter_hostname: "{{ vcenter_server }}"
      delegate_to: "localhost"

    #Add the MAC address to PXE server and configure the kickstart
    - role: addToPXE
      delegate_to: "installation_server"

    # The VM will boot by default on a specific network card and connect to the installation server
    # to complete the Operating system Installation
    - role: vmPower
      vcenter_hostname: "{{ vcenter_server }}"
      state: "powerOn"
      delegate_to: "localhost"

  post_tasks:
    #This task waits for valid SSH connection.
    # It validates that the host is correctly deployed
    - name: "Wait for the Installation to complete"
      wait_for_connection:
        timeout: "{{ timeout_install }}"

#Second Play
- hosts:
  - Mission
  - Fds
  - CCC
  - Logging
  - Chrono
  # VMs are installed. We will connect through SSH in order to make first configurations
  gather_facts: yes
  roles:
    #Secure the operating System
    - role: securityHardening

# Inclusion of a generic playbook to configure common configuration for Virtual machines
- include: commonConfiguration.yml
```

Table 5. Infrastructure Deployment Playbook

2. Components installation and Operational data deployment

For the components installation and operational data deployment, we use the same playbook type. Unlike in the infrastructure deployment, we need to install different versions of a component or operational data depending on the

validation platform. To achieve this, we define a dictionary structure. The key is the name of the validation platform and the value is a list of software versions or operational data versions. We include this dictionary in each playbook. We can access to values of the dictionary as variables and install a platform with the same playbook but with software versions that differ.

Here is a sample of this dictionary:

```
---
version_configuration:
  operational:
    mission_system_version: "1.0"
    fds_system_version: "3.1.6"
    ccc_system_version: "8.14.2b"
    ....
  validation01:
    mission_system_version: "2.0"
    fds_system_version: "3.1.7"
    ccc_system_version: "8.15"
    ....
  validation0x:
    ....
```

Table 6. Dictionary of versions

With this data structure, we can track any version update on each validation platform. This is where the quality is improved. Quality Assurance Manager can know at any time what is really installed on every existing validation or operational platform.

III. Quality process and working habits

A. We are developing "integration"

The software development is strongly linked with a source code management system. Typical workflows are created to manage the lifecycle of the software. Here are some examples:

- Delivery of a new software version
- Hot fixes for the latest stable version
- Implementation of new functionalities

If we are developing, a source code management is essential. We choose GIT. It is the most popular decentralized version control system for tracking changes in files and coordinating work inside a team. Unlike in software development, we are developing "integration". We need to create a workflow that handles these specificities. Here are major specificities that we take into account:

- Freeze the configuration of a validation platform for major system test
- Install new software or upgrade hardware configuration of a Virtual machine

We use the mechanism of branches to cover these specificities. In this figure below, we represent different types of branches that we used:

- The "master" branch is the reference branch where all other branches will eventually be merged. We try to not directly commit on the master and only do merge of specific branch.
- The "major system test" branch is used to freeze the configuration of a validation platform and named with the name of the test. It's created when the test preparation meeting is ended. We reinstall the platform from scratch for the test. We tag at the beginning and at the end of the test in order to identify all hotfixes that were "cherry-picked" (retrieved) from the master branch.
- The "new software integration" branch is named with the name of the software that we integrate. At the end of acceptance phase, we merge the branch to the master branch. This new software will be available for the next major system test.

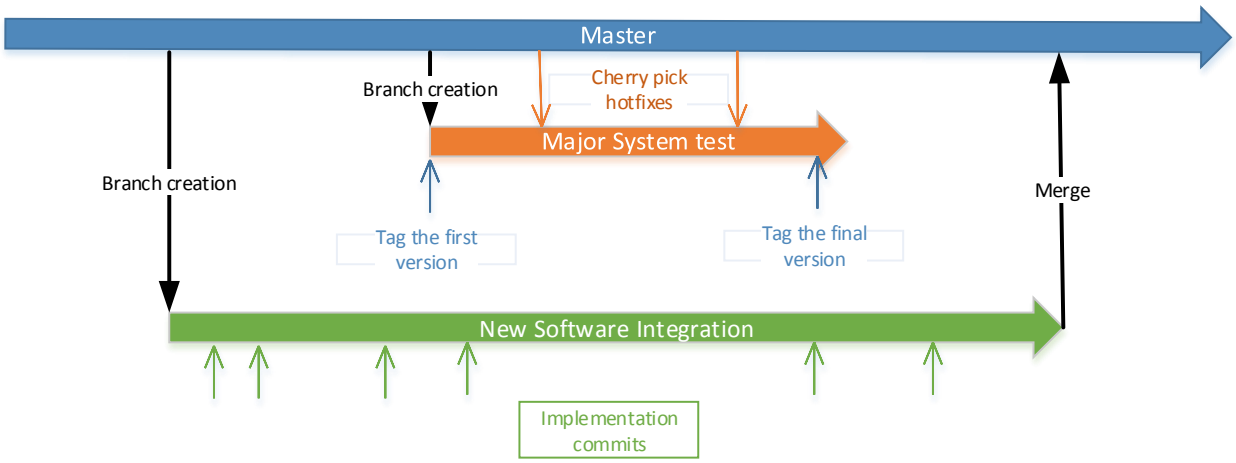

Figure 5. GIT branches

B. Changes in working habits of the operational team

In this part, we will give examples on how the working habits change by using configuration management and source code management.

The team is essentially composed of system engineers or operational engineers. There is no software engineer. The first key to the success of this implementation is to train the team in source code management and configuration management. Dedicated training session of three days for Ansible and a self-education for basic GIT commands are sufficient.

Another change in our working habits is that we never work directly on machines to apply a patch or change a configuration. The new habit is described below:

- Code modifications of the playbooks, roles, variables or inventories to fulfill the configuration change
- Test these modifications deployed via Ansible on a validation platform
- Merge these modifications into master branch
- Deploy via Ansible on the operational platform

The last change in our working habits is that we don't try to reconfigure a validation platform by removing old files, defragmenting disks or verifying the whole configuration. Instead we remove every virtual machine, checkout a specific version of our playbooks stored in GIT and redeploy them via Ansible. We gain in consistency and the team is more confident before a major system test.

IV. Conclusion

With this automated deployment method, we have a reliable, predictable solution to deploy a fully configured and operational ground control segment in less than two hours with only one command line. Only one simple command line means that in the future we will be able to delegate the deployment to the end user with a dedicated user-friendly interface. As a result, the added value of the operational team is not to deploy but to code the deployment.