Paper 814-2017

Platform à la carte: An assembly line to create SAS® Enterprise BI Server instances with Ansible

SAS® Global Forum 2017

Javor Evstatiev, EVS; Andrey Turlov, Allianz Managed Operation & Services SE

ABSTRACT

Installation and configuration of a SAS Enterprise BI platform in the requirements of the today's world depends on knowledge on a wide variety of subjects. Security requirements are growing, the number of involved components is increasing, time to delivery should be shorter and the quality improved. The expectations of the customers are based on cloud experiences where automated deployments with ready to use applications are state of the art. This paper describes an approach to address the challenges of deploying SAS 9.4 on Linux to meet today's customer expectations.

INTRODUCTION

In the reality of 2017 there is a need to provide the option to create and tailor platforms in accordance with the rapidly changing requirements of developers and business users. This process not only includes the creation of new environments on demand, but also the ability to scale processes in order to be able to host applications of any size and complexity.

A lot has happened since the initial introduction of the SAS 9 Platform. Administrators can now easily record and replay the installation and configuration of a SAS environment. The risk of human error is reduced and the result is reproducible. Hotfixes can be applied during installation, TLS configuration comes out of the box as does middle tier clustering… we have come a long way. But, is setting up a new SAS environment for your users a quick task? Well, there is more to it.

Before we can run the SAS® Deployment Wizard, the servers must be provisioned. Storage must be set up, user accounts created, firewall rules implemented and verified, certificates issued and dependencies cared for. Some important hotfixes may have been released that need to be downloaded into the SAS Depot. After the installation and configuration, site-specific content like Access Control Templates or user groups will be typically loaded and customizations performed. At the end of the process, the new system must be verified.

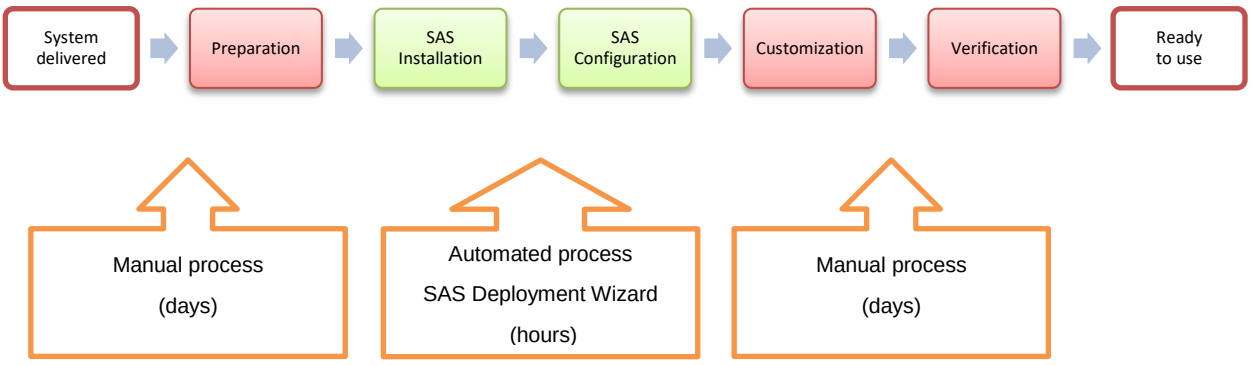


Figure 1: SAS Deployment Process

As you can see, the end-to-end process is still manually driven to a large extent and performed by different units, so the results may vary. This is especially troublesome if multiple stages with the same configuration must be deployed (Development, Integration, User Acceptance Test, Production) and multiple separate environments set up. Time to delivery is typically measured in days. How can this situation be improved?

There are multiple topics to address:

- Complexity
- Consistency
- Compliance
- Quality
- Reproducibility
- Configuration Management
- Integration in digital processes
- Change

After the system has been put in use, the much longer operations phase begins, again with its own challenges: These include maintenance releases, configuration changes and scaling out, to name but a few.

The idea of a toolset to control the entire process/workflow (not only during the provisioning phase but also along the entire life cycle of the system) comes to mind.

SAS AUTOMATIC PROVISIONING WITH ANSIBLE

ORCHESTRATION

Across large organizations multiple provisioning tools will probably be already in use. Such tools will typically distribute binary packages on selected machines. To do so, they often rely on some agent on the target machine. Leveraging them for SAS deployment soon becomes complicated, especially in the post-configuration steps where multiple tasks must be executed on separate tiers in dependency of each other.

Small teams will often write some shell scripts to perform the preparation and customization tasks. But maintaining and executing these sets of scripts under different technical users and on different sets of machines can be a tedious exercise; furthermore, refactoring and reusing them is not straightforward. There is a need for a tool that can act as the “glue layer”: Automating across services and applications no matter where they are.

A tool that has been very successful as a foundation for DevOps is Ansible. Citing the Ansible website:

“Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.” [1]

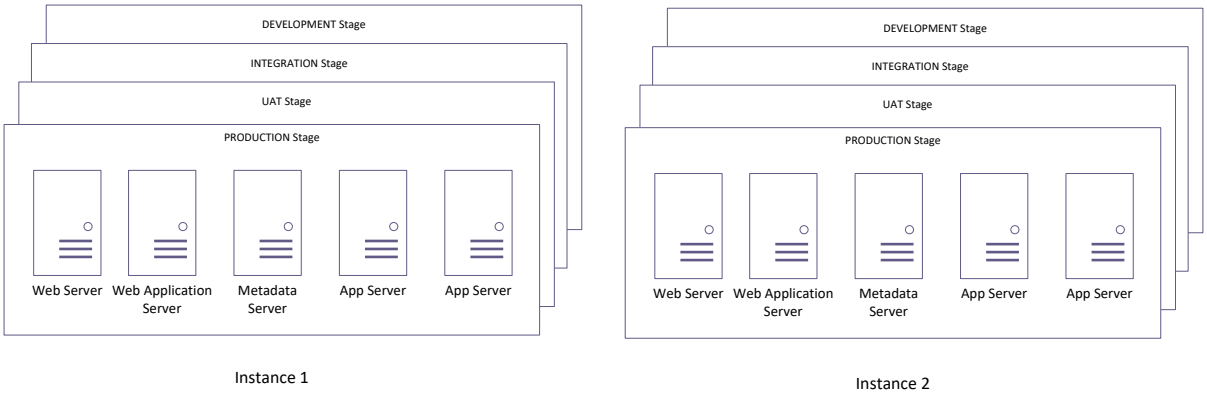
The core software is licensed under the GPL that guarantees it is free to use. The parent company was acquired by RedHat in 2015, which shows the importance of the tool for the industry.

Ansible allows for the creation of small and reusable components (tasks and roles) that reduce redundancies. It does not need a client, just ssh access. In contrast to shell scripts that are usually not safe to be re-run, Ansible allows idempotent execution of tasks without a lot of extra coding. Ansible features the concept of "Facts", i.e. system and environment information gathered before running tasks. Ansible uses these facts to check the state and to identify if it needs to change anything in order to achieve the desired outcome. This makes it safe to run Ansible repeatedly.

THE USE CASE

Let us look at a specific scenario. A 5 tiered SAS Enterprise BI Server must be installed in 4 stages (Development, Integration, User Acceptance Test, Production). Two independent instances will be created. The tiers are: Web, Web Application Server, Metadata Server, Application Server, SAS Web Infrastructure Platform Data Server. Physically, two servers are used as App Servers, 1 Webserver, 1 Webapp + SAS Web Infrastructure Platform Data Server, 1 Meta. The total number of servers is therefore 40 (5 servers x 4 stages x 2 instances).

[1] Ansible homepage (https://ansible.com). Retrieved January 14, 2017



All prerequisite processes for the installation and configuration (create users in central directories, certificates and similar) are complete. The SAS Depot is downloaded, a plan file is available and the response files are recorded. For the sake of simplicity we will ignore the client setup and exclusively focus on the server components.

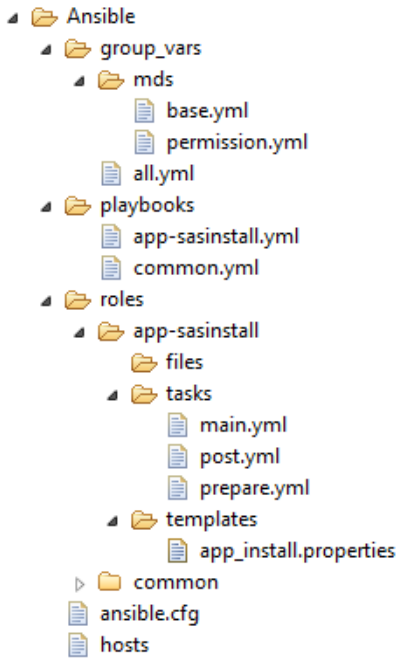
AUTOMATIC PROVISIONING

The installation and configuration of Ansible goes beyond the scope of this paper and is therefore not covered. For a good introduction into these topics, please refer to the links at the end of the document.

Before making a deep dive into one of the specific topics, let us outline the automatic provisioning process in general.

Setup of the Ansible project

The overall Ansible project structure is as follows:



Ansible can be configured to simultaneously work on multiple tiers of the SAS infrastructure. This is realized by grouping SAS installation and configuration information in Ansible's inventory files. The definition of SAS tiers is executed in the "hosts" file:

```
[mds]
mds.example.com
[wip]
wip.example.com
[app]
app1.example.com
app2.example.com
[web]
web.example.com
[webapp]
webapp.example.com
```

mds (Metadata Server), wip (SAS Web Infrastructure Platform Data Server), app (List of all App Servers), web (Web Server) and webapp (WebApp Server) are group names, which are used to classify variables and tasks that are sourced and executed at different stages of the SAS provisioning process.

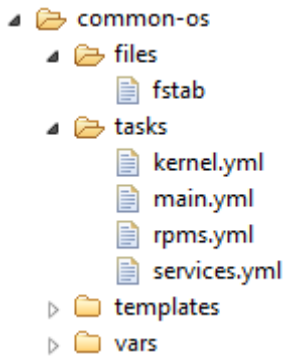
Specific variables (e.g. metadata server port, webserver certificates) for each group are defined in the group_vars folder, which contains the all.yml file for common variables and a subfolder for each group. Variables are sourced and are made available for Ansible tasks depending on the selected group in a step of a playbook (common variables in the all.yml file are always sourced).

Playbooks are used to orchestrate the provisioning process as a sequence of synchronous or asynchronous steps executed on different sets of hosts in a specific order. Playbooks are expressed in YAML format and contain a description of a certain configuration or a process. To load variables automatically, tasks and handlers playbook steps can be organized in a defined folder structure called a role. Roles are a means of automation around included tasks (see above) that allow to express playbook steps as reusable units which can be executed against one or several groups.

For example, the “common operation system” role is used in the common playbook as

```
- name: Common installation
  remote_user: root
  hosts: app, mds, webapp, wip, web
  roles:
  - role: common-os
```

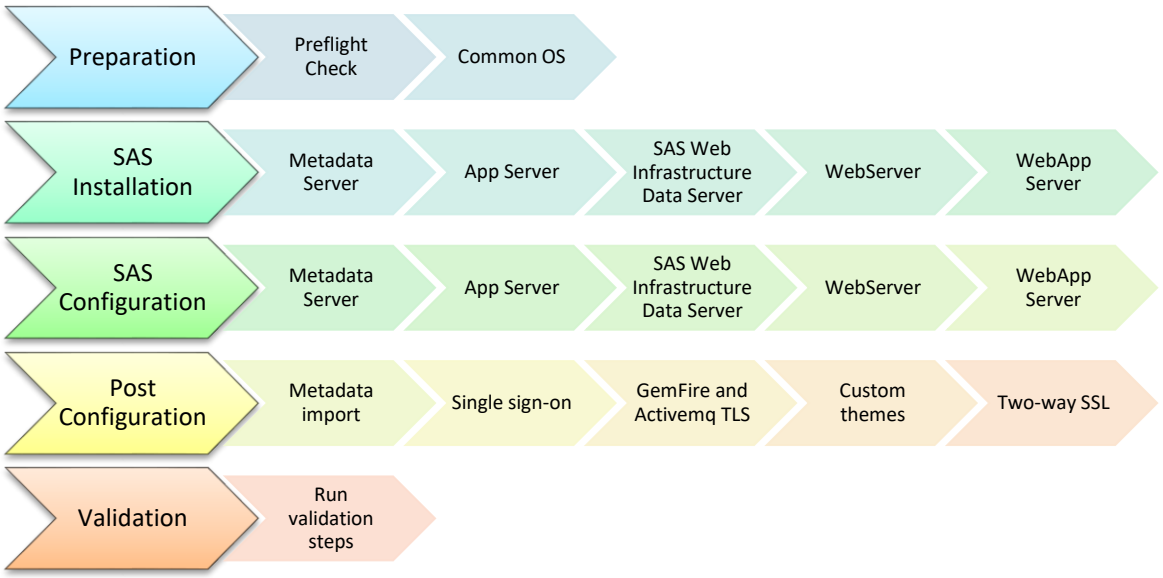
It features the following folder structure:



The “files” folder contains files that can be used by a role without changes (be copied, executed etc.). In contrast, files from the “templates“ folder are expressed in Jinja2 template language and are transformed first before being used. “vars” contains role specific variables and “tasks” contains role-specific tasks.

SAS Provisioning Process

The provisioning process is organized as a flow of playbooks executed against different combinations of servers. To be able to control the status of the whole process, each playbook delivers a status report at the end of the execution.



Playbooks should have no built-in dependencies between them. Playbooks must clean up after execution (clean up temporary files) and leave the system in a known state.

Preflight-Check Playbook

To ensure a successful installation process, the target hosts must first be checked for suitability. Initial checks are performed within this playbook. The list of possible checks can include:

- Available RAM, CPUs and disk space. Each SAS tier has a different resource requirement.
- RPMs check: Are all required RPMs installed?
- ulimits check, e.g.:

```
- name: ulimits
  shell: ulimit -a|egrep "files|proc"
  register: ulimits
- debug: var=ulimits.stdout.split('\n')
```

- Kernel version check.
- DNS aliases.
- IP configuration, interfaces and similar.

Common OS Playbook

This playbook is executed for each group and is responsible for the execution of common system operational tasks for all components, e.g.:

- Installation of system packages (RPMs):

```
- name: install common RPMs
  yum: name={{item}} state=installed
  with_items: #list of rpms can be defined in group_vars, or listed
  - screen
  - libstdc++
  - vim
  - …
```

- Enable or disable certain services.
- Create logical volume groups.
- Create filesystems:

```
- name: create filesystems for metadata server
  filesystem: fstype={{item.fstype}} dev={{item.device}}
  with_items:
   -{name: "sasmeta", device: "/dev/applvg/lvsasmeta", mountpoint:
   "/var/opt/data/MetadataRepositories", fstype: "ext4"}
```

Note: The order of filesystems defines the order of entries in the fstab. This can be very important to prevent pile mounts.

- Mount created filesystems:

```
- name: mount and entries in fstab MDS
  mount: name={{item.mountpoint}} src={{item.device}} state=mounted
  fstype={{item.fstype}}
  with_items:
        -……
```

SAS Installation Playbook

This playbook is executed for each tier and is responsible for the installation of SAS binaries. A response file is recorded for each tier and is parameterized with variables from group_vars.

- Metadata Server installation.
  meta_install.properties are copied from the templates folder of the role and rendered based on metadata-specific variables defined in the group_vars/mds folder, e.g.:

```
# Specify SAS Home
# Specify the location where SAS software will be installed on this
machine.
#SAS_HOME=<full path>
SAS_HOME={{sas_home_mds}}
```

  The installer is executed as follows:

```
- name: Run installer
  shell: ./setup.sh -quiet -responsefile /tmp/meta_install.properties
  chdir=/var/opt/depot
```

- SAS APP installation. Executed on each APP in the cluster.
- SAS Web Infrastructure Platform Data Server installation.
- Web Server installation.
- WebApp Server installation.

SAS Configuration Playbook

This playbook is executed for each tier and is responsible for the configuration of the SAS environment. The configuration of each tier starts with the installation of the certificates.

- Metadata Server configuration.
  Certificate installation response file (add_ca.properties):

```
MANAGE_TASK=certframe_add
certframe.add.file=/tmp/certificates/
```

  Certificate installation task:

```
- name: install certs
```

```
  shell: ./sasdm.sh -quiet -responsefile /tmp/add_ca.properties
  chdir=/opt/sas/SASDeploymentManager/9.4
```

Configuration task:

```
- name: Run configure
  shell: ./setup.sh -quiet -responsefile /tmp/meta_config.properties
  chdir=/var/opt/depot
```

- SAS APP configuration.
- SAS Web Infrastructure Platform Data Server configuration.
- Web Server configuration.
- WebApp Server configuration.

Post Configuration Playbook

Post-configuration tasks allow tailoring the system to the individual requirements of each separate installation. The playbook contains tasks that are usually executed manually after configuration by the SAS Deployment Wizard, for example the import of site-specific metadata, setting up single sign-on, GemFire and ActiveMQ with TLS or custom themes.

One of the tasks that deserves more detailed attention is the configuration of a two-way SSL between Web Server and Web Application Server. In two-way SSL authentication, the SSL client application verifies the identity of the SSL server application, and then the SSL server application verifies the identity of the SSL-client application. As the SAS Deployment Wizard currently allows configuration of two-way SSL only between the Web Server and the Internet browser, two-way SSL between the Web Server and the Web Application Server must be configured manually. The following steps must be executed:

1. Generate self-signed server certificate for webapp server using openssl.
2. Create Java keystore and copy to webapp server:

```
- name: run keytool
  command: ./keytool -importkeystore -deststorepass {{deststorepass}}
  -destkeypass {{destkeypass}}
  -destkeystore /tmp/ssl/keystores/webapp.jks
  -srckeystore /tmp/ssl/keys/webapp.p12 -srcstoretype PKCS12
  -srcstorepass {{srcstorepass}} -alias {{webapp_alias}} -noprompt
  args:
      chdir: "/opt/sas/SASPrivateJavaRuntimeEnvironment/9.4/jre/bin"
```

3. Add ssl-connectors with generated keystore to server.xml configuration of each Web Application Server.

The connector includes, among others, the path to the keystore file generated in step 2.

4. Import server certificates to trust store to be able to configure auto.publish to ssl:

```
- name: import certificates to truststore
  command: ./keytool -importcert -keystore
/opt/sas/SASPrivateJavaRuntimeEnvironment/9.4/jre/lib/security/cacert
s -storepass "{{castore_pass}}" -alias webapp.example.com -file
/tmp/ssl/certs/webapp.example.com.crt -noprompt
  args:
      chdir: "/opt/sas/SASPrivateJavaRuntimeEnvironment/9.4/jre/bin"
```

5. Modify Java options in setenv.sh file for each Web Application Server instance to configure ssl.
6. Copy self-signed certificate chain generated in step 1 to Web Server.
7. Switch balancer in the httpd.conf of the webserver to use https.
8. Create a client certificate (web.example.com.pem) for a Web server.
9. Add SSLProxy settings to httpd-ssl.conf of the Web Server:

```
- name: add proxy settings
  blockinfile:
    dest: "SAS_WEB_CONF/Lev1/Web/WebServer/conf/extra/httpd-ssl.conf"
    insertbefore: "</VirtualHost>"
    backup: yes
    marker: "#### SSL Proxy {mark} ####"
    block: |
```

```
        SSLProxyEngine on
        SSLProxyVerify require
        SSLProxyVerifyDepth 10
        SSLProxyCACertificateFile "ssl/chain.pem"
        SSLProxyMachineCertificateFile "ssl/web.example.com.pem"
```

Note: Using "blockinfile" marker with {mark} variable allows multiple execution of the task.

Validation Playbook

This playbook performs checks on the installed platform:

- Look for errors in deployment logs.
- Verify that all services are up and running, e.g. check if the Connect Spawner is up and running:

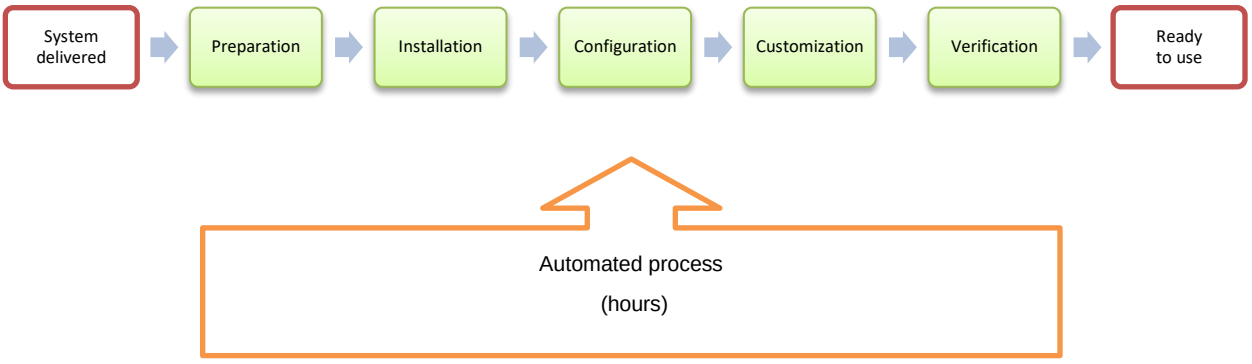
```
- name: check if connect spawner is running
  wait_for: port={{connect_spawner_port}} timeout=15
```

- Validate SAS services (WorkSpace Server, StoreProcess Server etc.)
- Login and run some test SAS code.

CONCLUSION

To recapitulate, the following has been achieved by the automated provisioning process based on Ansible:

- An assembly line to create as many environments as required has been set up.
- The process from bare server delivery to operational SAS Platform is down to hours instead of days.
- Streamlined processes increase the quality of deployments and reduce human error. The results are standardized and the infrastructure is more stable.
- The sources (code, configuration files etc.) and documented results of the installation are available to reproduce the whole process in case of emergency or on regulatory request.



Where do we go from here? The ability to deploy the platform faster and with predictable result opens a wide field of opportunities:

- Continuous integration use-cases can be implemented.
- Platform upgrades can be simplified.
- Many repeatable operations tasks can be "ansibilized". This increases the efficiency of the operations team.
- Easier introduction of DevOps to the SAS world is possible.

REFERENCES

Ansible

http://ansible.com

http://docs.ansible.com/ansible/playbooks.html

SAS Two-Way SSL:

http://support.sas.com/resources/papers/proceedings14/SAS054-2014.pdf

http://support.sas.com/documentation/cdl/en/bimtag/68217/HTML/default/viewer.htm#n0k64m6cvfcvy1n181b3cwn659kx.htm

http://support.sas.com/documentation/cdl/en/bimtag/68217/HTML/default/viewer.htm#n0nakjyj6hlqmvn11p9p04l25j9n.htm

ACKNOWLEDGMENTS

We thank Jan Bigalke for critically reviewing the manuscript.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Javor Evstatiev, EVS

javor.evstatiev@evs-data.com

Andrey Turlov, Allianz Managed Operations & Services SE

andrey.turlov@allianz.de