17th Int. Conf. on Acc. and Large Exp. Physics Control Systems ICALEPCS2019, New York, NY, USA JACoW Publishing
ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEAPP04

ICS INFRASTRUCTURE DEPLOYMENT OVERVIEW AT ESS

B. Bertrand[*], S. Armanet, J. Christensson, A. Curri, A. Harrisson, R. Mudingay, ESS, Lund, Sweden

Abstract

The ICS Control Infrastructure group at the European Spallation Source (ESS) is responsible for deploying many different services. We treat Infrastructure as code to deploy everything in a repeatable, reproducible and reliable way. We use three main tools to achieve that: Ansible (an IT automation tool), AWX (a GUI for Ansible) and CSEntry (a custom in-house developed web application used as Configuration Management Database). CSEntry (Control System Entry) is used to register any device with an IP address (network switch, physical machines, virtual machines). It allows us to use it as a dynamic inventory for Ansible. DHCP and DNS are automatically updated as soon as a new host is registered in CSEntry. This is done by triggering a task that calls an Ansible playbook via AWX API. Virtual machines can be created directly from CSEntry with one click, again by calling another Ansible playbook via AWX API. This playbook uses proxmox (our virtualization platform) API for the VM creation. By using Ansible groups, different proxmox clusters can be managed from the same CSEntry web application. Those tools give us an easy and flexible solution to deploy software in a reproducible way.

INTRODUCTION

The Integrated Control System Division (ICS) is an organisational unit responsible for the control systems within the European Spallation Source (ESS) facility, including control systems for accelerator, target, neutron scattering systems and conventional facilities. Within ICS, the Control System Infrastructure group is in charge to design, implement and operate the IT infrastructure needed to reliably run the Experimental Physics Industrial Control System (EPICS) eco-system. As such we have a large number of networks, physical devices and virtual machines to administer. To make this task manageable by a limited team, we try to put a lot of automation in place and treat infrastructure as code to make deployment repeatable, reproducible and reliable. We rely upon an internal GitLab [1] server to store all our code and JFrog Artifactory [2] to host binary artifacts. We do extensive use of GitLab's integrated CI/CD pipelines for continuous integration. Our deployment workflow is built on Ansible [3] (an IT automation tool), AWX [4] (a GUI for Ansible) and CSEntry [5] (a custom in-house developed web application).

ANSIBLE

Ansible is an open-source configuration management, orchestration and application deployment tool. Its main goals are simplicity and ease-of-use. It was developed by Michael DeHaan and acquired by Red Hat in 2015.

[*] benjamin.bertrand@esss.se

Ansible Concepts

Ansible is **agentless**. Only Python and OpenSSH are required on the **managed nodes**, the servers or network devices you want to manage. The list of nodes that can be accessed is defined in an **inventory**, which can be in different format (YAML or ini). The inventory is also used to organize the hosts in different groups as shown in Fig. 1.

```
[lbservers]
loadbalancer1

[webservers]
web1
web2
```

Figure 1: Ansible inventory.

An Ansible **playbook** is a **YAML** file describing an ordered lists of **tasks** that are mapped to a group of hosts, as illustrated in Fig. 2.

```
- hosts: all
  become: true
  tasks:
    - name: install chrony
      package:
        name: chrony
        state: present

    - name: ensure chrony is running and enabled
      systemd:
        name: chronyd
        state: started
        enabled: true
```

Figure 2: Ansible playbook.

Each task performs an action using a **module**, the unit of code Ansible executes. Each module has a particular use. In Fig. 2 example, the *package* module installs chrony using the Operating System package manager (yum on CentOS) and the *systemd* module starts and enables the chronyd daemon. Those tasks are performed on the *all* group, a default group that refers to all the hosts defined in the inventory. The name of the task provides a human readable description that is displayed by Ansible when running it. As you can see, playbooks are quite easy to read even for people not familiar with Ansible.

Ansible provides a large number of modules to work with files, database, package managers and even network devices. There is probably a module for the action you want to perform. If not, you can always fallback to the *command* module or even write your own. All Ansible modules are designed to be **idempotent**, meaning that the second time a task is run, nothing should change. Some care should of course be taken to make a playbook idempotent, like when using the *command* module. Idempotence is an important property to make a system maintainable. A playbook can be split in **roles**. A role is a pre-packaged unit of work that can be reused and shared via Ansible Galaxy or git repositories. A role defines tasks but doesn't map them to any hosts.

Ansible At ESS

There are different ways to organize Ansible roles and playbooks. At ESS we use mainly roles and our playbooks only include roles by default, as demonstrated in Fig. 3.

```
---
- hosts: csentry
  become: true
  roles:
    - role: ics-ans-role-csentry
```

Figure 3: ESS Ansible playbook.

Each role and playbook is stored in a separate git repository on our GitLab server. It makes roles easy to reuse and enables us to pin their version using git tag. By convention, the required roles for a playbook are stored in the *roles/requirements.yml* file as shown in Fig. 4.

```
---
- src: git+https://gitlab.esss.lu.se/ics-ansible-galaxy/ics-ans-role-repository.git
  version: v0.11.0
- src: git+https://gitlab.esss.lu.se/ics-ansible-galaxy/ics-ans-role-docker.git
  version: v1.2.1
- src: git+https://gitlab.esss.lu.se/ics-ansible-galaxy/ics-ans-role-traefik.git
  version: v0.8.1
- src: git+https://gitlab.esss.lu.se/ics-ansible-galaxy/ics-ans-role-csentry.git
  version: v0.11.0
- src: git+https://gitlab.esss.lu.se/ics-ansible-galaxy/ics-ans-role-certificate.git
  version: v0.1.4
```

Figure 4: Playbook roles/requirements.yml.

This file is used by the *ansible-galaxy* command to download the roles locally before to run the playbook.

Each role is tested using Molecule [6], a framework designed to aid in the development and testing of Ansible roles. Molecule makes it easy to run a playbook in a Docker container or Vagrant box for local testing. It verifies the role syntax, takes care of setting the needed instances to run it, ensures it is idempotent, checks Ansible best practices by running ansible-lint [7] and even run some tests on the deployed instance with Testinfra [8]. Testinfra is a framework that helps to write unit tests in Python to test the actual state of a server configured by a management tool as demonstrated in Fig. 5.

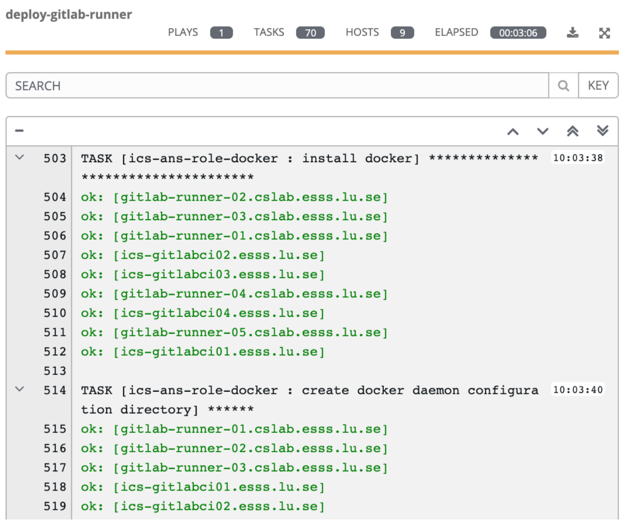
```
def test_docker_running_and_enabled(host):
    docker = host.service("docker")
    assert docker.is_running
    assert docker.is_enabled


def test_docker_run(host):
    with host.sudo():
        cmd = host.command("docker run hello-world")
    assert cmd.rc == 0
    assert "Hello from Docker!" in cmd.stdout
```

Figure 5: Testinfra tests.

This can help ensure that the role did what you expected. Molecule encourages an approach that results in consistently developed roles that are well-written, easily understood and maintained.

AWX

Ansible is a command line tool. It doesn't require a single controlling machine. Playbooks can be run from any machine that has the proper credentials and access to the nodes to manage. While this is nice and one of the thing that makes Ansible easy to start with, for production use, having a central server has many benefits. AWX is a web-based user interface for Ansible. It allows us to centralize all our jobs, keep the credentials required to access managed nodes safe, use role based access and log all the events that occurred. AWX also provides a REST API to trigger Ansible jobs remotely. Figure 6 shows a screenshot:



```
TASK [ics-ans-role-docker : install docker] ************** 10:03:38
*********************
ok: [gitlab-runner-02.cslab.esss.lu.se]
ok: [gitlab-runner-03.cslab.esss.lu.se]
ok: [gitlab-runner-01.cslab.esss.lu.se]
ok: [ics-gitlabci02.esss.lu.se]
ok: [ics-gitlabci03.esss.lu.se]
ok: [gitlab-runner-04.cslab.esss.lu.se]
ok: [ics-gitlabci04.esss.lu.se]
ok: [gitlab-runner-05.cslab.esss.lu.se]
ok: [ics-gitlabci01.esss.lu.se]

TASK [ics-ans-role-docker : create docker daemon configura 10:03:40
tion directory] ******
ok: [gitlab-runner-01.cslab.esss.lu.se]
ok: [gitlab-runner-02.cslab.esss.lu.se]
ok: [gitlab-runner-03.cslab.esss.lu.se]
ok: [ics-gitlabci01.esss.lu.se]
ok: [ics-gitlabci02.esss.lu.se]
```

Figure 6: AWX.

Jobs can be triggered with one click from the web interface. Access can be given to non expert users who don't need to know anything about Ansible.

In AWX, a **project** is a collection of Ansible playbooks that are usually placed into a source code management system like Git. A **job template** is the combination of an Ansible playbook (from a project) and the parameters required to launch it: inventory, credentials.. As we store each playbook in its own git repository, a project is basically identical to a playbook for us. A project always points to the master branch of the git repository and is automatically updated before to launch the associated job template. The master branch is thus considered stable. This is because roles are always pinned to a specific version as we saw in Fig. 4. Updating those versions could be a bit cumbersome. This is why this is done by a GitLab bot: galaxy-bot [9]. The bot listens to webhooks events. When a tag is pushed to a role, the *galaxy-bot* creates a merge request to update the role version in each playbook using that role. If the molecule tests pass, the merge is accepted. If not, an e-mail is sent to the author who pushed the tag. The bot also automatically creates a new project in AWX when a new playbook repository is pushed to GitLab. *galaxy-bot* is based on gidgetlab [10], a Python framework to interact with GitLab API and to write GitLab bots: applications that run automation on GitLab, using GitLab WebHooks and API.

CSENTRY

CSEntry, which stands for Control System Entry, is a web application developed using Flask [11], one of the most popular Python web frameworks. It is used to register all devices with an IP address (network switch, physical machines, virtual machines). Figure 7 shows a screenshot:

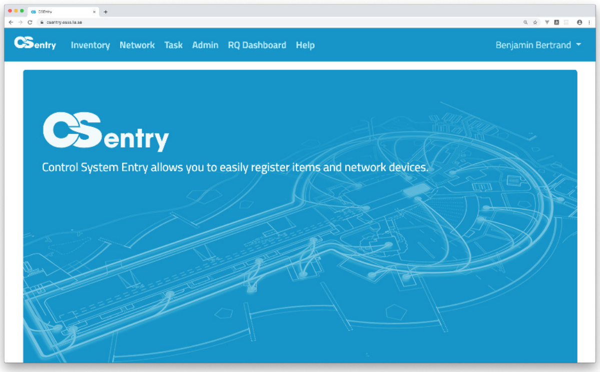

CSentry
Inventory
Network
Task
Admin
RQ Dashboard
Help
Benjamin Bertrand
CSentry
Control System Entry allows you to easily register items and network devices.

Figure 7: CSEntry.

CSEntry is a typical Flask application. It uses PostgreSQL [12] as database and SQLAlchemy [13] as Object Relational Mapper. Elasticsearch [14] is used for indexing and search. Background jobs are handled by RQ [15] (Redis [16] Queue), a simple Python library for queuing jobs. Long running processes are queued in Redis and processed by workers to avoid blocking the web server. The same Redis instance is also used for some caching. CSEntry also provides a REST API to retrieve information or register devices programmatically.

CSEntry is used as a configuration management database and our Ansible main dynamic inventory. It contains all our networks and hosts with their IP addresses. Ansible variables and groups can be defined like in a standard Ansible inventory. Groups of groups are also supported. Default groups are static and hosts have to be manually assigned to them. The application also has the notion of dynamic groups for networks, network scopes or device types. If a dynamic group of type network is created with the name of an existing network, like utgard-dev, all hosts part of that network are automatically added to this group. This is a powerful way to group hosts and assign variables to them.

Ansible Dynamic Inventory

AWX can use a custom script as dynamic inventory source. The script has to return a defined JSON-encoded dictionary with a list of groups and hosts. This is what the csentry-inventory [17] script does using CSEntry's API. This script makes the CSEntry inventory available and ready to use in AWX. The inventory script automatically populates some variables for every host as the example in Fig. 8 shows. Thanks to those variables, a playbook can access information about any host, and not only the ones targeted by that playbook.

```
ansible_host: 172.30.238.75
csentry_device_type: VirtualMachine
csentry_host_id: 1220
csentry_interfaces:
  - cnames: []
    gateway: 172.30.238.126
    ip: 172.30.238.75
    is_main: true
    mac: '02:42:42:79:b7:db'
    name: archiver-utgard
    netmask: 255.255.255.192
    network:
      domain: cslab.esss.lu.se
      name: Utgrd-EPICS-GW
      vlan_id: 3967
```

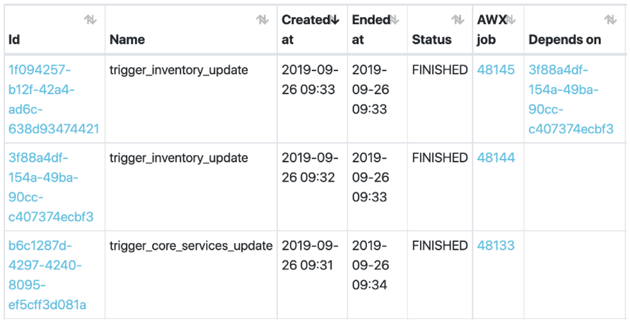
Figure 8: Host inventory variables.

Core Services Update

When registering a host in CSEntry, two tasks are triggered:

- an inventory sync
- an update of the core services

Those background jobs are processed by RQ workers. Commands are sent to AWX using the REST API and the ansible-tower-cli [18] library. The status of each task and a link to the AWX job is kept in the postgres database as shown in Fig. 9.



Id	Name	Created at	Ended at	Status	AWX job	Depends on
1f094257-b12f-42a4-ad6c-638d93474421	trigger_inventory_update	2019-09-26 09:33	2019-09-26 09:33	FINISHED	48145	3f88a4df-154a-49ba-90cc-c407374ecbf3
3f88a4df-154a-49ba-90cc-c407374ecbf3	trigger_inventory_update	2019-09-26 09:32	2019-09-26 09:33	FINISHED	48144	
b6c1287d-4297-4240-8095-ef5cff3d081a	trigger_core_services_update	2019-09-26 09:31	2019-09-26 09:34	FINISHED	48133	

Figure 9: CSEntry tasks.

The update of the core services launches a workflow template (Fig. 10) to update the DHCP, DNS and Radius [19] servers. A workflow template in AWX is a combination of job templates to run several playbooks in parallel or based on some dependency.

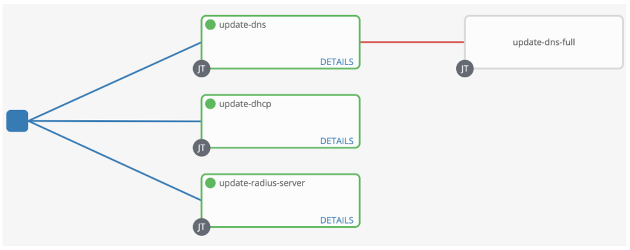


Figure 10: AWX core services update.

For each service (DHCP, DNS and Radius), the full configuration is re-generated by an Ansible playbook using the inventory. We saw earlier in Fig. 8 that all required information (IP, network, vlan id, MAC address, hostname) was populated as host variables by the *csentry-inventory* script. Fig. 10 shows two playbooks to update the DNS. For performance reason, the *update-dns* playbook doesn't check for new subnets. If a machine was just added to a new subnet, this fast job will fail and the *update-dns-full* one will be triggered. This demonstrates how some jobs can be triggered based on the result of another one in a workflow. When *update-dns* is successful, the *update-dns-full* job is not run. The Radius servers are used to automatically assign switches ports to a VLAN id based on the MAC address of the connected device. It simplifies the network switches configuration and enables users plugging a device to different ports and still being connected to the proper VLAN.

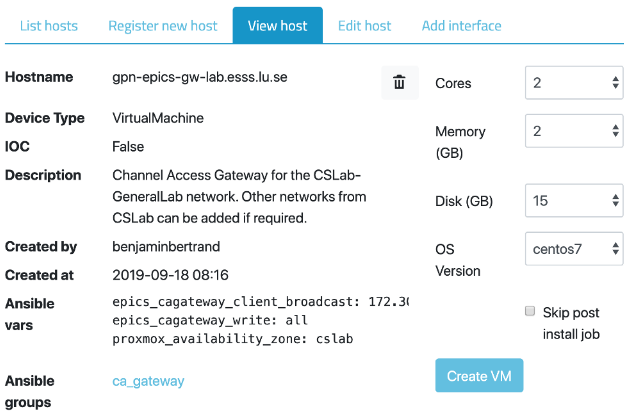
The inventory synchronization is in fact triggered every time a change impacts a group or host in CSEntry's database. This is done using SQLAlchemy event listeners. To avoid unnecessary jobs, there can be only one running task and one waiting in queue. If an inventory update task is already in queue, the change won't trigger a new job. For the core services update, the principle is the same but the trigger is only on interface change.

Virtual Machine Creation

Once a Virtual Machine has been registered, it's possible to create it with one click from CSEntry's view host page represented in Fig. 11.

Clicking on the *Create VM* button will enqueue a job to launch a template on AWX. This playbook uses Proxmox VE [20] REST API to create the Virtual Machine, as represented in Fig. 12.

Proxmox VE is a complete open-source platform for enterprise virtualization. It is based on KVM hypervisor. It can scale out to a large set of clustered nodes, includes a web-based management interface as well as a REST API. We have several clusters for isolation purpose. By default, we target the cluster to deploy to based on the domain of the VM. We can also force a specific cluster by setting the *proxmox_availability_zone* Ansible variable.



Hostname	gpn-epics-gw-lab.esss.lu.se
Device Type	VirtualMachine
IOC	False
Description	Channel Access Gateway for the CSLab-GeneralLab network. Other networks from CSLab can be added if required.
Created by	benjaminbertrand
Created at	2019-09-18 08:16
Ansible vars	epics_cagateway_client_broadcast: 172.3 epics_cagateway_write: all proxmox_availability_zone: cslab
Ansible groups	ca_gateway

Cores	2
Memory (GB)	2
Disk (GB)	15
OS Version	centos7
☐ Skip post install job	

Figure 11: CSEntry view host.

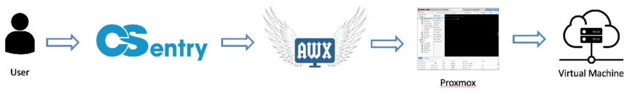
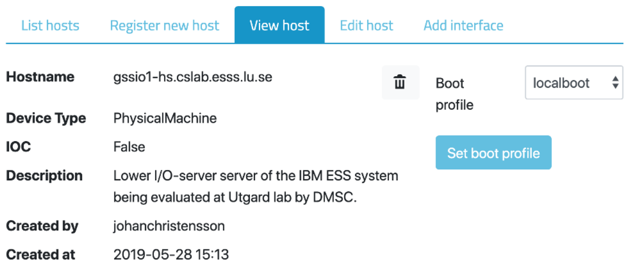


Figure 12: CSEntry VM creation.

Physical Machines

For physical machines, an *autoinstall* server is available to perform network installation via PXE boot. From CSEntry, the boot profile can be selected as shown in Fig. 13.



Hostname	gssio1-hs.cslab.esss.lu.se
Device Type	PhysicalMachine
IOC	False
Description	Lower I/O-server server of the IBM ESS system being evaluated at Utgard lab by DMSC.
Created by	johanchristensson
Created at	2019-05-28 15:13
Boot profile	localboot

Figure 13: Boot profile selection.

Clicking on the *Set boot profile* button launches a task on AWX that updates some links on the autoinstall server to ensure the MAC address of the machine points to the chosen profile. After this step, the machine has to be manually rebooted and set to boot from the network. The installation will start automatically. At the end, the link is set back to local boot to make sure the machine is not re-installed at next reboot.

Application Deployment

For most applications, the deployment is performed directly from AWX. Playbooks usually target a specific group. That group shall first be defined in CSentry and assigned to the required hosts. Once done, the appropriate job template can be launched from AWX web interface. This makes it very easy to deploy an existing application to a new instance. Note that some applications are deployed automatically from GitLab CI using AWX REST API with the tower-cli tool. Thanks to AWX role-based access control, users not part of the infrastructure group can safely be given permissions to deploy their application, without requiring to share any ssh keys.

CONCLUSION

The tools and workflow put in place allow us to deploy our infrastructure in a repeatable, reproducible and reliable way. Ansible and AWX are both easy to use and powerful. Molecule enforces us to follow best practices and write roles in a consistent and maintainable way. Developing our own web application gives us a lot of flexibility. We can give users the means to register and deploy the services they need without creating tickets and waiting for us to handle them. CSEntry is customized to our needs but the principles are quite general and replacing some part of the system shouldn't be too difficult. Note that this solution is for small scale and is of course not intended to replace a system like OpenStack, which is more powerful but also more complex to maintain.

REFERENCES

[1] GitLab, `https://about.gitlab.com`

[2] JFrog Artifactory, `https://jfrog.com/artifactory`

[3] Ansible, `https://docs.ansible.com/ansible/latest/index.html`

[4] AWX, `https://github.com/ansible/awx`

[5] CSEntry, `https://gitlab.esss.lu.se/ics-infrastructure/csentry`

[6] Molecule, `https://molecule.readthedocs.io`

[7] ansible-lint, `https://docs.ansible.com/ansible-lint`

[8] Testinfra, `https://testinfra.readthedocs.io`

[9] galaxy-bot, `https://gitlab.esss.lu.se/ics-infrastructure/galaxy-bot`

[10] gidgetlab, `https://gidgetlab.readthedocs.io`

[11] Flask, `https://flask.palletsprojects.com/`

[12] PostgreSQL, `https://www.postgresql.org`

[13] SQLAlchemy, `https://www.sqlalchemy.org`

[14] Elasticsearch, `https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html`

[15] RQ, `http://python-rq.org`

[16] Redis, `https://redis.io`

[17] csentry-inventory, `https://gitlab.esss.lu.se/ics-infrastructure/csentry-inventory`

[18] Tower CLI, `https://tower-cli.readthedocs.io`

[19] Radius, `https://www.gnu.org/software/radius/`

[20] Proxmox VE, `https://www.proxmox.com/en/proxmox-ve`