

Trusted configurations for UMD deployments

Bruce Becker, EGI Operations

✉ bruce.becker@egi.eu

🐦 [@brusisceddu](https://twitter.com/brusisceddu)

🗨️ [brucellino](#)

🆔 0000-0002-6607-7145

Outline

- Ansible in the UMD path to production
- A style guide for EGI
 - What makes our roles 'ours' ?
 - How can we trust each others' work ?
- Tests and Infrastructure Specifications
- Collaboration and re-usability

Ansible in the UMD path to production

- We used to have a unique* configuration management tool – YAIM
- YAIM solved at least two problems :
 - Configuration Management : single place in which to express the desired configuration state
 - Deployment : executable means to achieve the state
- UMD products slowly dropped YAIM, favouring one or other tool for configuration management :
 - Puppet
 - Ansible

Site configuration management – too many options ?

- Sites are now freer to choose how to configure and deploy middleware
- The choice of tool comes down to *local* expertise and historical preferences
- Support for debugging configuration issues however becomes harder – community tends to split into ‘if you use this tool, change this variable... oh, sorry you use other tool, can’t help.’
- Given that there is no objective measure by which one tool is better than another, can we find a way to support each other?
- *IE*, does a site admin really have to understand the internals of Puppet or Ansible to configure the middleware at their site ?

Ansible or Puppet – does it matter ?

- It is difficult to say why Ansible or Puppet have the following they have in certain environments.
- There are design and ecosystem considerations which suit different scenarios better in each case
- Both can be used to achieve continuous, correct deployment
- So : does it matter to the site admin whether a product expresses a preference for either ? I hazard that it should not.
- UMD deployment should be a **conservative force** :
 - End states should not depend on the path taken to get there

Many tools : Why ? Pair programming

- First off : Configuration code *is code. Treat it as such.*
- Pair programming allows collaboration and quality code review.
- If both tools should achieve the same state for a given middleware product, we should be able to review each others' work.
- But they are different languages and paradigms? How can we review each others' work ?
 - Focus on patterns instead of specific implementation
 - Collaborate on the *objective measures of quality* - ie the final result

Why ? Cross-validated deployments

- Why are there 4 experiments looking for the Higgs ?
- There are always biases and assumptions in deployment and configuration scenarios – these make their way into the code for deployment.
- They implicitly exclude certain use cases or scenarios
- Cross-validating deployments with different tools tends to surface these assumptions and force us to confront them.
- A good goal would be to achieve consistent deployments from a given state, regardless of the means to achieve it.

Why ? : A healthy ecosystem

- Reliance on a single tool and tribal knowledge around it is not a good sign
- Healthy 'inter-breeding' of ideas from slightly different ways of doing things will probably lead to better health of the UMD ecosystem and whatever proceeds it.
- We can teach *patterns* and *skills* rather than tools – these are useful to industry (better employability) but also makes it easier for us as a community to attract talent

EGI in the Ansible universe : the EGI Style Guide

- Ansible is simple *but powerful* IT automation – really tempting to just solve problems and be done with it.
- However, this same power leads to massive divergence in the way in which problems are solved, making it difficult to trust that other peoples' work will work for you.
- e.g. you find a role for configuring CREAM :
 - Will it respect my local setup ?
 - Does it do the network configuration ?
 - Who maintains this ?
 - **Is it even correct ?**

Objective measures

- Step 1 : A Style Guide
 - 🔗 [EGI-Foundation/ansible-style-guide](https://github.com/EGI-Foundation/ansible-style-guide)
- Expresses opinions on :
 - Documenting roles
 - Ansible syntax in roles
 - Testing role scenarios, testing tools
 - Role release and publication
 - Collaborating with code
- Read more : brucellino.github.io/blog/Ansible-Style-Guide
- WIP : egi-foundation.github.io/ansible-style-guide/

Ansible Style Guide rôle Skeleton

- When creating new roles, one typically uses `ansible-galaxy init <role name>`
- The default has several important bits 'missing' which are necessary for engendering re-use and trust :
 - Issue and PR templates, contributing guide, links to EGI support structures
 - Relevant platforms which EGI supports in `meta.yml`
 - Properly-generated `.travis.yml`
 - Proper webhooks on `build-passing` to `galaxy.ansible.com`
- `ansible-galaxy init --role-skeleton=ansible-style-guide/egi-galaxy-skeleton high-performance-grid-cloud`

Objective measures

- Step 2 : A compliance profile
 -  EGI-Foundation/ansible-fashion-police
- We implement controls (using Inspec) for:
 - Automated testing
 - GitHub repository configuration
 - Role Metadata
 - Role Skeleton

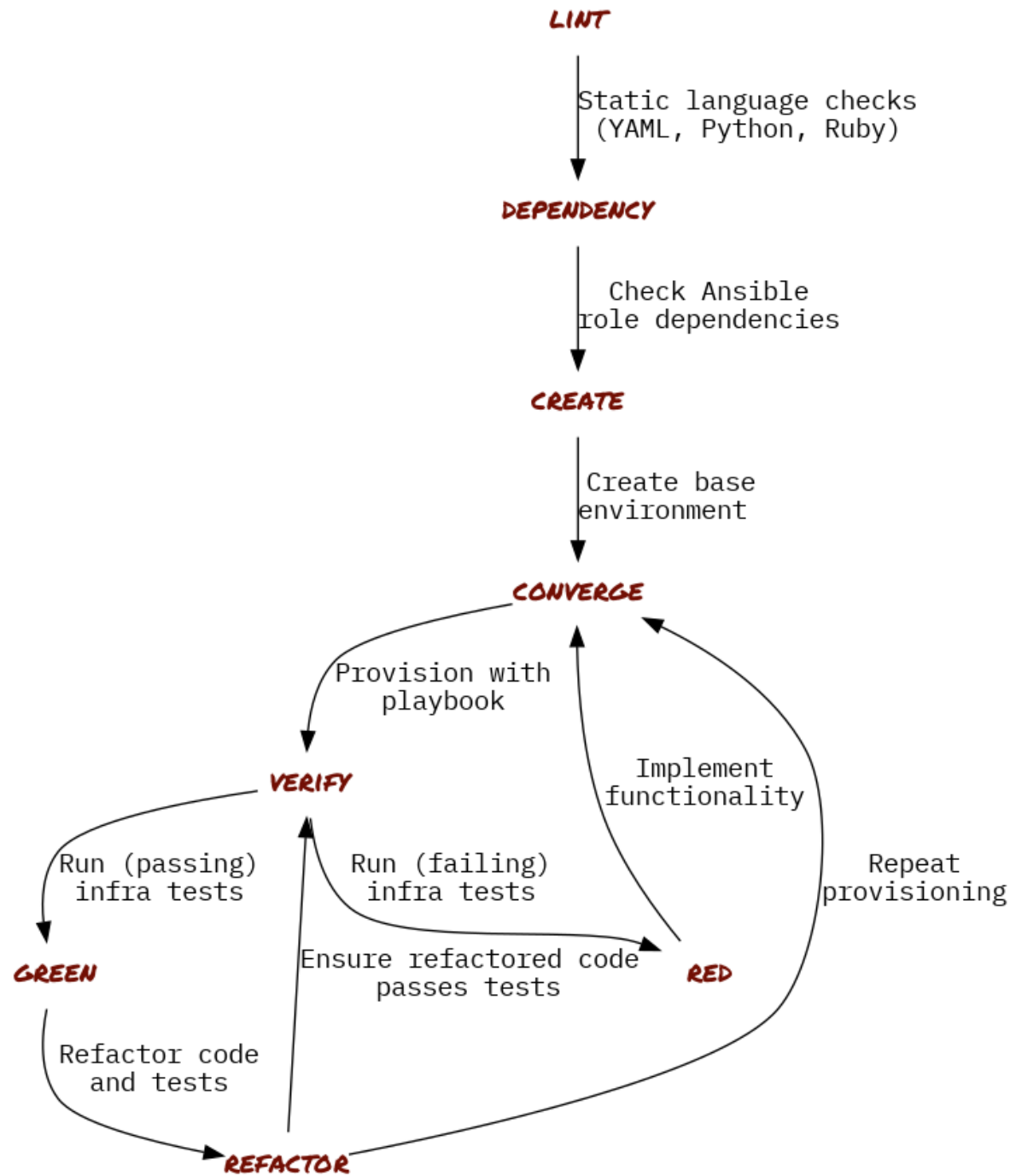
One role, many scenarios

- The underlying platform is changing – clouds, vms, DMZs, containers, *etc*
- The configuration tool should not enforce a particular execution environment, but should express the middleware product appropriately in the respective environment
- We need to mock and test various production environments

Can we apply traditional TDD to Infrastructure ?

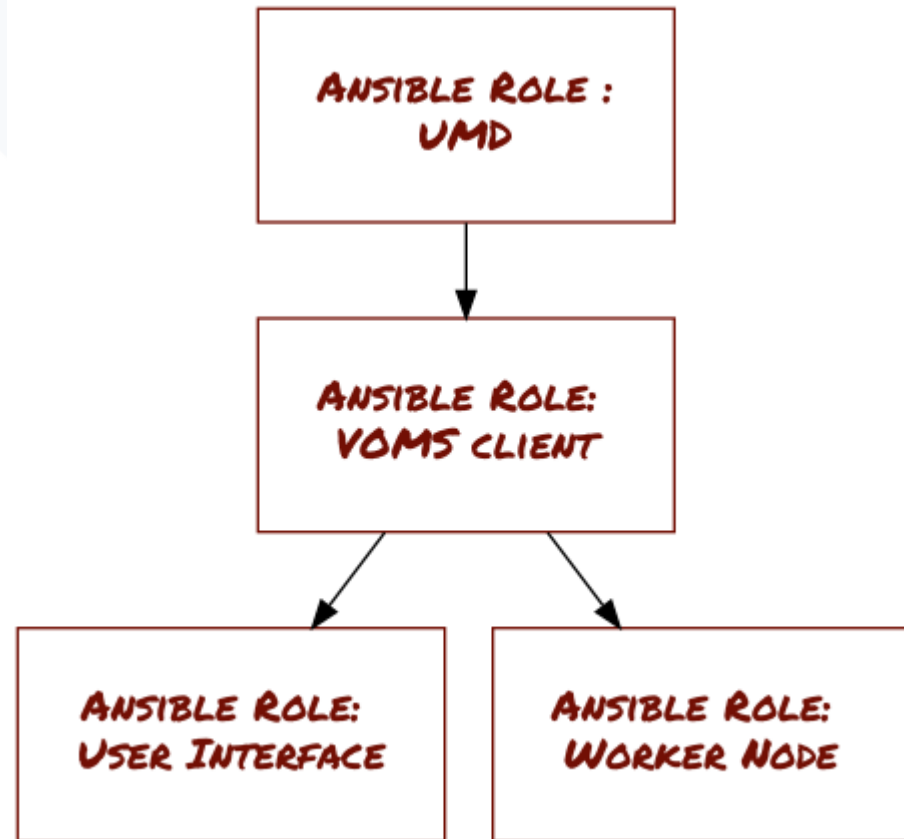
- Molecule provides a general-purpose mock and testing framework for Ansible roles
- Allows developer to define many deployment scenarios and test against them :
- Easiest is to test in Docker, but can test against OpenStack or bare-metal scenarios, from given starting points

TDD for Ansible roles



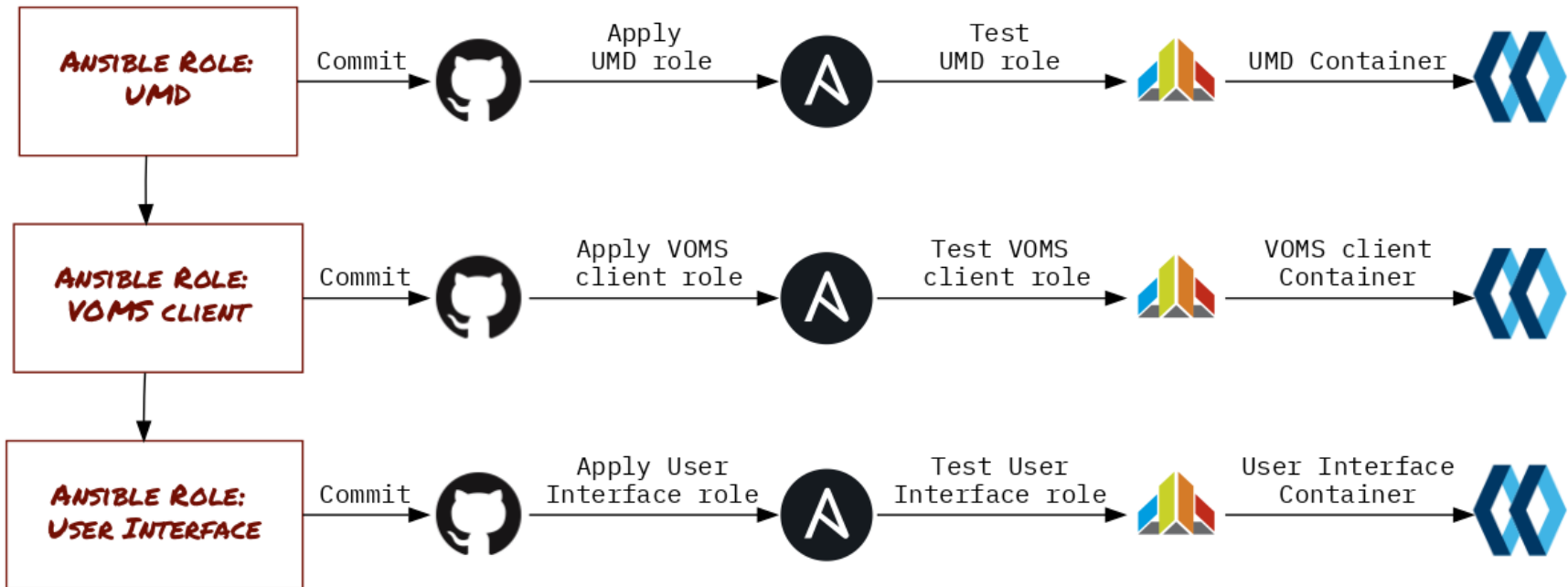
Infrastructure Models and Specifications

- Remember : “UMD deployment should be a **conservative force**”
- We should be able to model deployments independently of the tool used...



Infrastructure Models and Specifications

brucellino.github.io/blog/Style-Guide-In-Action












Better use of infrastructure : Ansible Galaxy

☰ My Content

Name ▾

Name ▾ ↓^A/_Z

>	 brucellino	>  1 Owners >  1 Provider Namespaces	+ Add Content ⋮
>	 EGI-Foundation	>  4 Owners >  1 Provider Namespaces	+ Add Content ⋮
>	 sci-gaia	>  3 Owners >  1 Provider Namespaces	+ Add Content ⋮

Better use of infrastructure : Ansible Galaxy

The screenshot displays the Ansible Galaxy interface for the EGI-Foundation organization. At the top, there is a navigation bar with a dropdown menu, the organization name 'EGI-Foundation', and statistics for '4 Owners' and '1 Provider Namespaces'. A '+ Add Content' button is located in the top right corner. Below the navigation bar is a search bar with a 'Name' dropdown and a 'Filter by Name...' input field. The main content area shows a list of roles with the following details:

Role Name	Description	Status	Action
ansible-packstack-role		Failed 5 months ago	Import
ansible_role_ui	Ansible role to deliver Us...	Succeeded 2 months ago	Import
caso		Succeeded 3 months ago	Import
cmd		Succeeded 3 months ago	Import
discourse-sso		Succeeded 3 months ago	Import

Better use of infrastructure : Quay

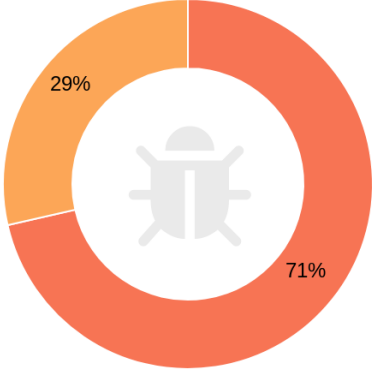
- CI on Travis pushes to Quay on build-passing :
 - artefacts immediately available for re-use in subsequent steps of the pipeline
- Something similar could be done for VMS (push to AppDB)
- Vulnerabilities and obsolete packages immediately visible
 - Can open issues against the repo automatically

DevSecOps – thanks clair

← egi/wn 1b8ada7de317

Quay Security Scanner has detected **7** vulnerabilities.
Patches are available for **7** vulnerabilities.

5 High-level vulnerabilities.
2 Medium-level vulnerabilities.



CVE	SEVERITY ↓	PACKAGE	CURRENT VERSION	FIXED IN VERSION	INTRODUCED IN LAYER
Filter Vulnerabilities					

Collaboration and re-usability

- Putting development into context with a solid foundation and objective measures makes it easier for operations to trust the results thereof.
- Both Dev and Ops can agree on the *final state* of the service *in given scenarios*
- Clear case for following TDD and BDD using relevant tools (TestInfra, Inspec, Cucumber)
- Issues in the final state can be traced back to code if there is an unbroken pipeline between commit and deploy.

DevOps

- For us to achieve DevOps and support many more deployment scenarios -
 - Small sites with few staff, in known scenarios
 - Unmanned deployments
 - Different deployment platforms
- ... we need product teams and infrastructure engineers to collaborate ...
 - Peer review, pull requests, infrastructure specs, documentation
- ... not on the code of the product itself, but the pipeline for delivering that product in a viable state to the production environment
- Close links with the 'lightweight' sites work from CERN and SKA HPC ecosystem sites in Africa.

In summary :

- UMD configurations should be put through the same rigorous testing as UMD products
- Having more than one tool to achieve production states is good, as long as there is a community of practice in EGI around those tools
- A community of practice is expressed in the EGI Ansible Style Guide, along with a compliance profile.
- Allows those wanting to
 - *develop* infrastructure components to do so **smoothly and collaboratively**
 - *Operate* infrastructure components to do so **with confidence**

links

- [Style Guide](#), [Compliance profile](#), [Quay Org](#), [Ansible Galaxy](#)
- [Website – egi-foundation.github.io/ansible-style-guide](http://egi-foundation.github.io/ansible-style-guide)
- **Testing tools**
 - [Molecule : molecule.readthedocs.io](http://molecule.readthedocs.io)
 - [TestInfra : testinfra.readthedocs.io](http://testinfra.readthedocs.io)
 - [Inspec : www.inspec.io](http://www.inspec.io)
 - [Cucumber : cucumber.io](http://cucumber.io)
- **Blogs :**
 - [‘E-Infrastructure Components that are built to last’](#)
 - [‘Style Guide in Action’](#)