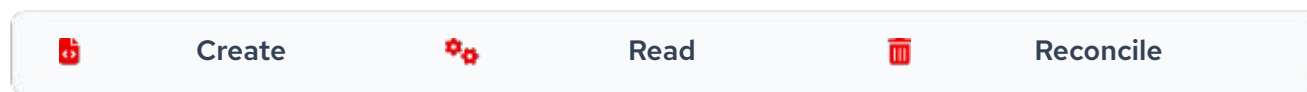




REDHAT-COP / INFRA.SATELLITE_CONFIGURATION

Bringing Satellites Down to Earth: Implementing **Configuration as Code** for Red Hat Satellite

Declarative Day-1 & Day-2 platform engineering with Ansible playbooks, modular filetrees, and GitOps.



Silvio Pérez Torres
Architect

Ivan Aragonés
Senior Consultant

Agenda: **Architecture & Implementation**

Conceptual Foundations

- **Satellite Operations:** Understanding the current model
- **IaC vs CaC:** Infra provisioning vs logical state orchestration.
- **Two-Tier Collection Stack:** Orchestration roles vs certified execution modules.

The Playbook Ecosystem

- **Day 0 (export.yml):** Extracting existing Satellite state into modular .d/ trees.
- **Day 1/2 (configure.yml):** Ingestion via filetree_read & ordered dispatch.
- **Continuous Drift (object_diff.yml):** Drop-diff logic to prune unmanaged assets.
- **Enterprise Promotion:** Base + environment overrides across DEV, TEST, PROD.

Satellite Operations: **Understanding the current model**

The "ClickOps" Reality

Satellite is the backbone of enterprise RHEL lifecycle management, yet standard operations create critical risks:

- **Dispersed Complexity:** Orgs, locations, content views, activation keys, and host groups configured by hand.
- **Tribal Knowledge:** Configurations locked inside individual admins' heads with zero formal documentation.
- **Configuration Drift:** Unnoticed divergence between Dev, QA, Staging, and Production instances.

Operational Impact

The problem is not Satellite's automation capabilities, but making its configuration manageable like software:

- **Audit Blind Spots:** Unable to reliably answer who changed what, when, and under which authorization.
- **Slow Recovery:** Rebuilding instances during Disaster Recovery requires lengthy, error-prone manual runbooks.
- **Fragile Governance:** Lack of automated peer review or pull request gates before applying changes.

IaC or CaC? Defining Platform Boundaries

Infrastructure as Code (IaC)

PROVISIONING

Instantiates the **underlying compute, VM, storage, and networking layers**.

```
# Terraform / OpenStack / AWS scope:
resource "virtual_machine" "satellite_primary" { cpu = 16 "64GB" disk = "1TB" }
```

Provides the bare server running RHEL and the Satellite RPMs, but knows nothing about Katello/Foreman entities.

Configuration as Code (CaC)

DAY-1 / DAY-2 STATE

Declares the **internal logical state** of the platform entities via structured YAML data.

```
# infra.satellite_configuration scope:
satellite_activation_keys:
- name: ak-rhel9-prod
  organization: ACME_Global
  lifecycle_environment: Production
```

An automation engine reconciles existing objects so the live API matches the version-controlled repository description.

The Solution Stack: **Community & Modules**

Orchestration Collection

Community

`infra.satellite_configuration`

Contains the high-level playbook workflows, data ingestion, and dispatch engine:

filetree_create : Day-0 [Export]

filetree_read : ingests .d yamls

dispatch : Dependency runner

filetree_reconcile : Detects drifts

Certified Execution Modules

Red Hat Official

`redhat.satellite`

Official modules communicating with Foreman and Katello REST APIs:

- `redhat.satellite.organization` & `location`
- `redhat.satellite.content_view` & `lifecycle_environment`
- `redhat.satellite.activation_key` & `hostgroup`
- `redhat.satellite.sync_plan`, `product`, `repository`

Guarantees idempotent execution: creates if missing, updates if drifted, skips if intact.

The Playbook Suite: **devel/playbooks/**

The upstream repository standardizes three core playbooks covering the full platform lifecycle:

1. filetree_create.yml

DAY 0

Export Existing State

Queries Satellite API, normalizes objects, formats YAML files, and populates the modular .d/ filetree.

```
- hosts: localhost
  roles:
    - infra.satellite_configuration.filetree_create
```

2. filetree_read.yml

DAY 1 & DAY 2

Apply Desired State

Ingests the filetree (with optional environment overrides) and dispatches configurations in strict dependency order.

```
- hosts: localhost
  roles:
    - infra.satellite_configuration.filetree_read
    - infra.satellite_configuration.dispatch
```

3. filetree_reconcile.yml

RECONCILIATION

Prune Deleted State

Compares live Satellite API state against a list state. Deletes or disables orphaned resources.

```
- hosts: localhost
  roles:
    - infra.satellite_configuration.object_diff
```

Day 0 Migration: **playbooks/export.yml**

How filetree_create converts live Katello/Foreman API entities into Git-ready declarative definitions:



Execution Syntax: `ansible-navigator run playbooks/export.yml -e satellite_export_directory="./config"`

Introspects current production Satellite, parses schema properties, and establishes an immediate versioned baseline.

The Filetree Pattern: **satellite_organization.d/**

Modular Directory Pattern

Directory Organization (.d/)

```
satellite-configuration/  
├── satellite_organizations.d/  
│   └── corporate_orgs.yml  
├── satellite_locations.d/  
│   └── regional_datacenters.yml  
├── satellite_lifecycle_environments.d/  
│   └── lifecycles.yml  
├── satellite_content_views.d/  
│   ├── rhel8_soe.yml  
│   └── rhel9_soe.yml  
├── satellite_activation_keys.d/  
│   ├── prod_keys.yml  
│   └── dev_keys.yml  
└── satellite_hostgroups.d/  
    └── base_infrastructure.yml
```

Why the .d Pattern?

- **No Monolithic Files:** Avoids 5,000-line unstructured YAML files.
- **Granular Pull Requests:** Engineers modify single files (e.g., rhel9_soe.yml) without causing git merge conflicts.
- **Role Decoupling:** Each .d/ directory maps cleanly to its corresponding Ansible variable dictionary.
- **Dynamic Discovery:** Any new *.yml file added to a .d folder is automatically picked up by `filetree_read`.

Applying Desired State: **Satellite Config as Code**

Step 1: filetree_read

Loads, validates, and concatenates all .d/ files into memory:

```
# Inside playbooks/configure.yml:
- name: Load configuration filetree
  ansible.builtin.include_role:
    name: infra.satellite_configuration.filetree_read
  vars:
    satellite_configuration_directory: "{{ playbook_dir }}/../config"
    satellite_configuration_overrides_directory: "{{ env_dir |
default(omit) }}"
```

Creates unified in-memory dictionaries: satellite_organizations, satellite_content_views, etc.

Step 2: dispatch

Orchestrates roles in strict entity dependency order:

```
# Guaranteed Execution Hierarchy:
1. settings & organizations & locations
2. products & repositories & sync_plans
3. lifecycle_environments & content_views
4. activation_keys & hostgroups
5. subnets, domains & compute_resources
```

Guarantees parent items exist before children attempt binding.

Multi-Environment: Base & Overrides Pattern

Repository Inheritance Hierarchy

```
satellite-casc/  
├── common/ # Base definitions (DRY)  
│   ├── satellite_organizations.d/  
│   ├── satellite_content_views.d/  
│   └── satellite_hostgroups.d/  
└── environments/  
    ├── dev/ # Development overrides  
    │   └── satellite_settings.d/  
    ├── test/ # Staging overrides  
    │   └── satellite_subnets.d/  
    └── prod/ # Production overrides & Vault  
        ├── satellite_subnets.d/  
        └── vault_secrets.yml
```

Deep Dictionary Merging

- **Base Directory:** Stores common SOE content views, base organizations, and generic hostgroups.
- **Overrides Directory:** Points filetree_read to environment-specific folders to replace or add values.
- **Name-Based Key Matching:** If an object has the same name, environment properties override base properties.
- **Promotion Strategy:** Promotes tested YAML definitions across environments without duplicate code.

Playbook Execution: Navigator & EEs

> CLI Execution Commands

Using ansible-navigator with containerized Execution Environments:

```
# 1. Export configuration
ansible-navigator run playbooks/export.yml \
  -e satellite_export_directory="./config" \
  --mode stdout
# 2. Apply configuration
ansible-navigator run playbooks/configure.yml \
  -e satellite_configuration_directory="./config/common" \
  -e satellite_configuration_overrides_directory="./config/environments/prod" \
  --mode stdout
```

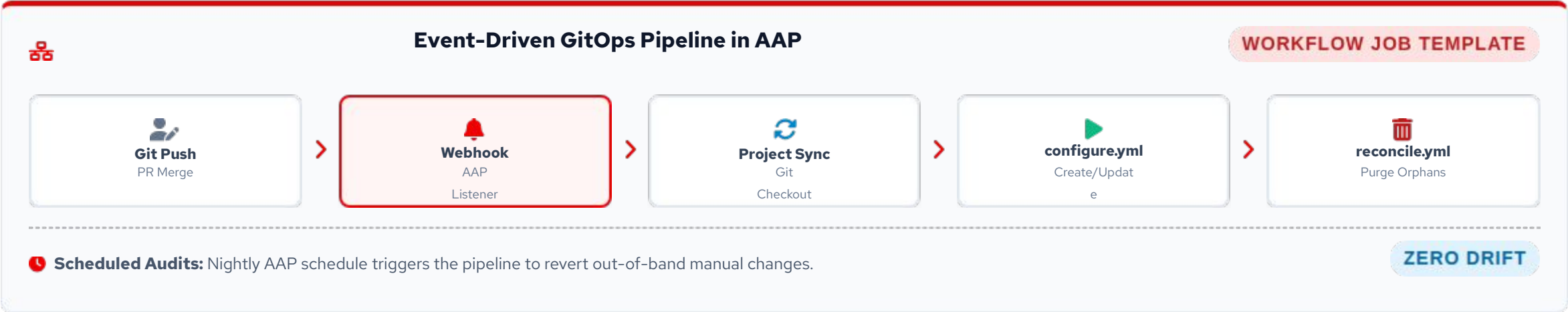
Technical Requirements

- **Ansible Core:** >= 2.16
- **Satellite Version:** Red Hat Satellite 6.16 - 6.18
- **Python Libraries:** PyYAML, requests, apypie (for Katello/Foreman API binding).

• Collections:

```
collections:
  - name: infra.satellite_configuration
  - name: redhat.satellite
```

Continuous Delivery: Reconciliation in AAP Controller / AWX



Let's test it!



red.ht/collection-infra-satellite-configuration



red.ht/demo-satellite-filetree-migration

Thank You

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.



linkedin.com/company/red-hat



youtube.com/user/RedHatVideos



facebook.com/redhatinc



twitter.com/RedHat