Quali | Torque

Writing Torque-Ready Ansible Playbooks

A Playbook Author’s Guide to Torque Integration

Writing Torque-Ready Ansible Playbooks

A Playbook Author's Guide to Torque Integration

Quali Torque Documentation

Table of Contents

1. How Torque Runs Your Playbook

Understanding what Torque does behind the scenes helps you write playbooks that work seamlessly. When a Torque environment launches an Ansible grain, four things happen in order:

Step 1: Inventory Generation

Torque reads the inventory-file section from the blueprint YAML and generates a standard Ansible inventory file. Your playbook never needs to ship with its own inventory. The blueprint author controls which hosts and groups exist, including all host_vars and group_vars.

Step 2: Inputs Become Extra-Vars

Torque collects the grain's inputs section, writes them into a JSON file at /var/run/ansible/inputs/inputs.json, and passes that file to ansible-playbook via --extra-vars. Every key you define in the grain inputs becomes a top-level variable in your playbook, as if the user had passed -e on the command line.

Step 3: Playbook Execution

Torque runs: ansible-playbook <your-playbook.yaml> --extra-vars @/var/run/ansible/inputs/inputs.json -i <generated-inventory>, plus any command-arguments defined in the blueprint.

Step 4: Output Collection

After execution, Torque reads any outputs exported by the torque.collections.export_torque_outputs module and makes them available to downstream grains and blueprint outputs.

> **Key Insight:** Your playbook does not need to know it is running inside Torque. If you write it to accept variables via extra-vars and work with a standard inventory, it will work both locally and in Torque without modification.

1.1 Practical Implications for Playbook Authors

- Do not ship inventory files with your playbook. Torque generates them.
- Do not hardcode values that should change per environment. Use variables with defaults.
- If your playbook creates resources other grains need to reference, export outputs.

- If your playbook creates resources that should be cleaned up, write a teardown playbook.

2. Rule 1: Use Variables, Not Hardcoded Values

Every value that might change between environments should be a variable. Torque injects values through two channels: extra-vars (from the grain inputs section) and inventory vars (from the inventory-file section). Your playbook should reference variables, never literal values.

2.1 Where Variables Come From in Torque

Source	Ansible Precedence	Best For
Grain inputs (extra-vars)	Highest: overrides everything	Names, sizes, regions, feature flags
inventory-file vars	Group/host var level	Connection info: credentials, endpoints, ansible_user
Playbook vars section	Play-level defaults	Sensible fallbacks for standalone testing

2.2 Writing Flexible Variables

Use the Jinja2 default filter to create a fallback chain. This lets your playbook work standalone (with defaults) and in Torque (with injected values):

```
vars:
  app_name: "{{ application_name | default('my-app') }}"
  db_engine: "{{ database_engine | default('postgres') }}"
  region: "{{ deploy_region | default('us-east-1') }}"
  instance_size: "{{ size | default('small') }}"
  env_name: "{{ environment | default('dev') }}"
```

2.3 What to Avoid

```
# BAD: hardcoded credential
api_key: sk-abc123def456

# BAD: hardcoded resource name
server_name: prod-web-01

# BAD: hardcoded connection info
db_host: 10.0.1.50

# GOOD: variable with default
server_name: "{{ vm_name | default('test-server') }}"
db_host: "{{ database_host | default('localhost') }}"
```

> **Rule:** If you can imagine someone wanting a different value in staging vs. production, it must be a variable with a default.

3. Rule 2: Let Torque Own the Inventory

This is the most important shift for Torque integration. In traditional Ansible, you maintain a static inventory file. In Torque, the inventory is generated dynamically from the blueprint. Your playbook should never depend on a specific inventory file existing in your repository.

3.1 How the Blueprint Creates Your Inventory

The blueprint author defines the inventory-file section, which Torque converts into a standard Ansible inventory at runtime:

```
# Blueprint YAML (written by the blueprint author):
inventory-file:
  web_servers:
    hosts:
      web1:
        ansible_host: '{{ .grains.provision_vm.outputs.vm_ip }}'
      web2:
        ansible_host: '{{ .inputs.static_server_ip }}'
    vars:
      ansible_user: '{{ .inputs.ssh_user }}'
      ansible_become: true
      http_port: '{{ .inputs.http_port }}'

# Generated Ansible inventory (what your playbook sees):
[web_servers]
web1 ansible_host=10.0.1.25
web2 ansible_host=10.0.1.30

[web_servers:vars]
ansible_user=ubuntu
ansible_become=true
http_port=8080
```

Your playbook code does not change. It references the same group names and variables it always has.

3.2 Design Your Hosts Directive for Flexibility

```
# GOOD: variable host group with a default
- hosts: "{{ target_group | default('web_servers') }}"
  become: true

# GOOD: API-only playbooks that run locally
- hosts: localhost
  connection: local
  gather_facts: false

# GOOD: all hosts in the provided inventory
```

```
- hosts: all
  gather_facts: true

# BAD: hardcoded group name without override
- hosts: production_servers   # cannot be overridden
```

3.3 Document the Inventory Contract

Every playbook directory should include a README that documents what the playbook expects from its inventory:

- Expected host groups (e.g., web_servers, db_servers, localhost)
- Required group/host vars (e.g., ansible_user, ansible_become, db_password)
- Required extra-vars / grain inputs (e.g., app_name, region, instance_size)

> **Think of it this way:** Your playbook is the code. The blueprint's inventory-file is the configuration. Separate them cleanly.

4. Rule 3: Centralize Credentials with YAML Anchors

When your playbook calls an external API (cloud provider, infrastructure controller, REST endpoint), define a single YAML anchor for credentials and reference it in every task. This keeps your playbook DRY and ensures credentials flow from variables, never hardcoded values.

4.1 The Anchor Pattern

```
vars:
  cloud_auth: &cloud_auth
    access_key: "{{ cloud_access_key }}"
    secret_key: "{{ cloud_secret_key }}"
    region: "{{ cloud_region | default('us-east-1') }}"

tasks:
  - name: Create VPC
    cloud_provider.vpc:
      <<: *cloud_auth
      name: "{{ env_name }}-vpc"
      cidr: 10.0.0.0/16
    tags: [network]

  - name: Create Subnet
    cloud_provider.subnet:
      <<: *cloud_auth
      vpc: "{{ env_name }}-vpc"
      cidr: 10.0.1.0/24
    tags: [network]
```

4.2 For SSH-Based Playbooks

When connecting to remote hosts, credentials come from inventory vars. The blueprint author sets ansible_host, ansible_user, and ansible_ssh_private_key_file in the inventory-file section. Your playbook does not need to define connection credentials at all.

```
# The blueprint provides these via inventory-file vars:
# ansible_host, ansible_user, ansible_become, ansible_ssh_private_key_file

# Your playbook simply targets the group:
- hosts: "{{ target_group | default('app_servers') }}"
  become: true
  tasks:
    - name: Install packages
      apt:
        name: "{{ item }}"
        state: present
      loop: "{{ packages_to_install }}"
```

5. Rule 4: Export Outputs with torque.collections

Ansible does not natively support outputs that flow between automation stages. Torque solves this with the torque.collections.export_torque_outputs module. If your playbook creates a resource that downstream grains need to reference (an ID, an IP address, a URL), you must export it.

5.1 Adding an Output Export Task

Add an export task at the end of your playbook. This task MUST run on localhost:

```
  tasks:
    - name: Deploy application
      shell: ./deploy.sh --name {{ app_name }}
      register: deploy_result
      tags: [deploy]

    - name: Export outputs to Torque
      torque.collections.export_torque_outputs:
        outputs:
          app_url: "{{ deploy_result.stdout_lines[-1] }}"
          app_version: "{{ app_version }}"
          deploy_status: "{{ deploy_result.rc == 0 | ternary('success','failed')
}}"
      delegate_to: localhost
      run_once: true
      tags: always
      ignore_errors: true   # Allows standalone testing without Torque
```

5.2 Rules for the Export Task

- **delegate_to: localhost** - the export module must run on the Ansible controller
- **run_once: true** - prevents duplicate exports in multi-host plays
- **tags: always** - ensures the export runs regardless of --tags / --skip-tags filters
- **ignore_errors: true** (optional) - lets the playbook succeed without Torque installed, useful for local testing
- Use snake_case for output names
- Export the minimum needed (IDs, URLs, names), not entire command outputs

5.3 Requirements File

Include torque.collections in your requirements.yaml so Torque auto-installs it:

```
# requirements.yaml (alongside your playbook)
collections:
  - name: torque.collections
```

> **Local Testing:** Install the module locally with: ansible-galaxy collection install torque.collections

6. Rule 5: Tag Tasks for Selective Execution

Tags allow the blueprint author to run subsets of your playbook by passing --tags or --skip-tags through the Torque grain's command-arguments field. Tag every task or logical block of tasks.

6.1 Tagging Strategy

- **Broad category tags:** install, configure, deploy, validate, cleanup
- **Specific tags:** nginx, postgres, monitoring, firewall, dns
- **Always tag the output export task with 'always'** so it runs regardless of tag filters

```
tasks:
  - name: Install web server
    apt: name=nginx state=present
    tags: [nginx, install]

  - name: Configure web server
    template: src=nginx.conf.j2 dest=/etc/nginx/nginx.conf
    tags: [nginx, configure]

  - name: Install database
    apt: name=postgresql state=present
    tags: [postgres, install]

  - name: Export outputs
    torque.collections.export_torque_outputs:
      outputs:
        web_url: "http://{{ ansible_host }}:{{ http_port }}"
    delegate_to: localhost
    run_once: true
    tags: always
```

6.2 How Blueprint Authors Use Tags

```
# In the blueprint grain, the author adds:
command-arguments: "--tags install"        # Only install packages
command-arguments: "--skip-tags postgres"   # Everything except DB
command-arguments: "--tags nginx,configure" # Only configure nginx
```

7. Rule 6: Write a Teardown Playbook for on-destroy

When your playbook creates resources (VMs, services, DNS records, cloud objects), provide a companion teardown playbook. The Torque Ansible grain has an on-destroy section that runs a separate playbook when the environment is terminated, ensuring resources are cleaned up automatically.

7.1 Convention

Place the teardown playbook in the same directory as your main playbook, named teardown.yaml:

```
ansible/deploy-app/
  playbook.yaml          # Creates resources
  teardown.yaml          # Removes resources (used by on-destroy)
  requirements.yaml
  README.md
```

7.2 Teardown Playbook Example

```
---
- name: Clean up application resources
  hosts: localhost
  connection: local
  gather_facts: false

  tasks:
    - name: Remove DNS record
      community.general.cloudflare_dns:
        zone: "{{ dns_zone }}"
        record: "{{ app_name }}"
        state: absent
      tags: [dns]

    - name: Deregister service
      uri:
        url: "{{ service_registry }}/{{ app_name }}"
        method: DELETE
      tags: [registry]
```

7.3 How the Blueprint References Teardown

```
grains:
  deploy_app:
    kind: ansible
    spec:
      source:
        store: my-repo
```

```
    path: ansible/deploy-app/playbook.yaml
  # ... inputs, inventory-file, outputs ...

  on-destroy:
    source:
      store: my-repo
      path: ansible/deploy-app/teardown.yaml
    inputs:
      - app_name: '{{ .inputs.app_name }}'
      - dns_zone: '{{ .inputs.dns_zone }}'
    inventory-file:
      localhost:
        hosts:
          127.0.0.1:
            ansible_connection: local
```

Key Point: The teardown playbook receives its own inputs and inventory separately from the main playbook. It must be fully self-contained. Do not assume any state from the deploy phase persists.

8. Rule 7: Structure Your Playbook Directory

Torque auto-discovers playbooks and auto-installs dependencies from requirements.yaml in the playbook's directory.

```
ansible/<grain-name>/
  playbook.yaml           # Main entry point
  teardown.yaml           # on-destroy playbook (if applicable)
  requirements.yaml       # Galaxy dependencies (auto-installed)
  README.md               # Documents host groups, vars, inputs, outputs
  roles/                  # Optional local roles
    <role-name>/
      tasks/main.yaml
      defaults/main.yaml  # Default values for role variables
```

8.1 The README Is Critical

Every playbook directory must include a README that documents the playbook's contract:

```
# deploy-app

Deploys an application to target servers.

## Host Groups
- app_servers (or override via 'target_group' variable)

## Required Inventory Vars
- ansible_user: SSH user for target hosts
- ansible_become: true/false

## Extra-Vars (Grain Inputs)
| Variable       | Default     | Description            |
|----------------|-------------|------------------------|
| app_name       | my-app      | Application name       |
| app_version    | latest      | Version to deploy      |
| environment    | dev         | Target environment     |
| http_port      | 8080        | HTTP listen port       |

## Outputs
- app_url: URL of the deployed application
- deploy_status: success or failed

## Teardown
teardown.yaml removes DNS records and deregisters the service.
```

9. Complete Before/After Example

This section shows a typical playbook before and after applying the Torque-ready rules.

9.1 BEFORE: Not Torque-Ready

```
---
- hosts: webservers
  become: true
  tasks:
    - name: Install nginx
      apt: name=nginx state=present

    - name: Deploy config
      template:
        src: nginx.conf.j2
        dest: /etc/nginx/nginx.conf
      vars:
        server_name: app.example.com
        listen_port: 8080

    - name: Register with load balancer
      uri:
        url: https://lb.internal/register
        method: POST
        body: '{"host": "10.0.1.50", "port": 8080}'
```

Problems: Hardcoded host group, server name, port, IP address. No outputs. No teardown. No tags. Cannot be reused across environments.

9.2 AFTER: Torque-Ready

```
---
- name: "Deploy Web Application"
  hosts: "{{ target_group | default('web_servers') }}"
  become: true
  gather_facts: true

  vars:
    app_server_name: "{{ server_name | default('app.example.com') }}"
    app_port: "{{ http_port | default(8080) }}"
    lb_url: "{{ load_balancer_url | default('https://lb.internal') }}"

  tasks:
    - name: Install nginx
      apt: name=nginx state=present
      tags: [nginx, install]

    - name: Deploy config
```

```
    template:
      src: nginx.conf.j2
      dest: /etc/nginx/nginx.conf
    vars:
      server_name: "{{ app_server_name }}"
      listen_port: "{{ app_port }}"
    notify: restart nginx
    tags: [nginx, configure]

  - name: Register with load balancer
    uri:
      url: "{{ lb_url }}/register"
      method: POST
      body_format: json
      body:
        host: "{{ ansible_host }}"
        port: "{{ app_port }}"
    tags: [lb, register]

  - name: Export outputs to Torque
    torque.collections.export_torque_outputs:
      outputs:
        app_url: "http://{{ app_server_name }}:{{ app_port }}"
        registered_host: "{{ ansible_host }}"
    delegate_to: localhost
    run_once: true
    tags: always
    ignore_errors: true

handlers:
  - name: restart nginx
    service: name=nginx state=restarted
```

What Changed: Host group is variable. All names and endpoints are parameterized with defaults. Tasks are tagged. Outputs are exported. This playbook works identically with 'ansible-playbook -i inventory -e http_port=9090' locally and inside a Torque grain.

10. Quick Reference Checklist

Run through this list before committing any new playbook:

#	Check	Details
1	No hardcoded credentials	Passwords, API keys, tokens come from variables (inventory or extra-vars), never literals
2	Credential anchor defined (if API playbook)	Single YAML anchor for auth, all tasks reference via <<: *anchor
3	Host group is variable	hosts: "{{ var | default('GroupName') }}" or hosts: localhost
4	All configurable values are variables	Names, ports, regions, sizes use {{ var | default(value) }}
5	Outputs are exported	torque.collections.export_torque_outputs with delegate_to: localhost, run_once: true, tags: always
6	Tasks are tagged	Broad category tags + specific tags for selective execution
7	requirements.yaml exists	Lists all Galaxy collection dependencies including torque.collections
8	teardown.yaml exists (if applicable)	Companion playbook with state: absent / cleanup logic
9	README.md documents contract	Host groups, required inventory vars, grain inputs, outputs, teardown
10	No static inventory shipped	Playbook works with any inventory provided at runtime
11	gather_facts set appropriately	false for API-only, true when you need ansible_host or system facts
12	connection set appropriately	local for API calls, ssh (default) for remote hosts