Engineer

Ansible

a tool that puts a server into a written-down state

What it is

Ansible is open-source automation software that connects to machines over SSH and brings each one to the configuration described in a set of YAML files: which packages are installed, which users and files exist, which services run and how they are set up. The machine Ansible runs from is the control node; the machines it configures are managed nodes, and nothing has to be installed on them as an agent, because Ansible logs in with the operating system's own credentials and does its work over that connection. The files list the outcome, and Ansible works out on each host whether that outcome is already true.

The pieces have plain names. An inventory lists the managed nodes and sorts them into groups such as `web` or `db`, with variables that apply to a whole group or to one host. A playbook holds one or more plays, and each play maps a group of hosts to a list of tasks. Each task calls a module, a small program that knows how to manage one kind of thing: a package, a file, a service, a user, a firewall rule. The project's own documentation describes playbooks as a repeatable, reusable, simple configuration management and multi-machine deployment system. A role bundles tasks, templates, default variables and handlers under one name, so "a PostgreSQL server" or "a hardened SSH daemon" is written once and applied to any host that needs it.

A short play shows the shape:

```
- name: Web servers serve the site
  hosts: web
  become: true
  tasks:
    - name: nginx is installed
      ansible.builtin.package:
        name: nginx
```

```
        state: present

    - name: nginx runs now and after a reboot
      ansible.builtin.service:
        name: nginx
        state: started
        enabled: true
```

Each task names a state, not an action. Most modules check first whether that state already holds and exit without doing anything if it does, so the second run of this play on a configured host changes nothing and says so. That property, idempotence, is what makes it safe to run the same playbook against the whole fleet every time, instead of tracking which host has had which script applied.

People asking about Ansible often weigh it against three neighbours. Against a shell script, the difference is that a script lists steps and runs them again whether they are needed or not, while a module checks before it acts. Against Terraform or Bicep, the difference is the layer: those tools create cloud resources such as networks, databases and virtual machines, while Ansible configures what runs inside a machine once it exists. Many teams use one of each. Against Puppet and Chef, the difference is the model: those usually run an agent on every machine that pulls its configuration from a central server, while Ansible pushes from wherever it is started and needs only SSH on the far side.

Use the concept in a decision

Put “What is Ansible?” to work in a real decision. We can help relate the concept to the system you are building or buying. The whole setup written down as files, so it can be recreated exactly and never lives only in one person's head.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Wrote 1,000 RBAC rules to secure Linux servers — 1,000 access rules written to close a security gap

Wrote 1,000 RBAC rules to secure Linux servers Get in touch →

Why it matters

A server set up by hand is only as reproducible as the memory of whoever set it up. Six months later nobody can say with confidence why a kernel parameter has its value, which package was pinned and why, or what has to happen first when the machine is rebuilt. Writing the setup down as playbooks turns that memory into files that can be read, reviewed in a pull request and run again on a fresh machine with the same result.

The written form also changes what drift means. When the playbook is the description of the host, running it in check mode against a live machine reports every place where the two disagree. With the `--diff` flag the report shows the lines in each file that would change. A fleet that is converged regularly stays close to its description, and a host that has been edited by hand shows up as a change the next time the playbook runs.

The model has costs. Ansible acts only when it is run, so drift between runs stays until the next one. Tasks run in order, one at a time across all matched hosts, and a large playbook against many machines is slower than a script that does one thing. Logic written in YAML and Jinja2 templates gets hard to follow once a role grows many conditions. And a run that stops partway leaves its hosts partly changed, because Ansible has no transaction to roll back.

What it looks like in practice

Several projects on this site show what a written-down estate looks like over time, and what it takes to keep it honest.

The company's infrastructure as code covers deployed public services and configuration for other components, including backup and mail automation whose activation is a separate step. The recorded implementation comprises 19 playbooks and 34 roles across 12,065 lines of YAML; those figures describe that stage, not the current repository size. A host sits in a tier group whose variables declare which roles it runs, so provisioning with no arguments converges every host to its declared state. Idempotence is treated as a contract: a converged host reports zero changes, and a play that cannot say that is not finished. Four command namespaces keep the promises apart, from a gate that touches no host to verification that changes a host on purpose and returns a verdict. The page is candid about the cost: every change is slower to make than editing a file on the server, and a converge that half-applies is worse than one that refuses.

Access control is another place where a written description pays off. 1,000 RBAC rules were configured through Ansible across developer environments, application instances, PostgreSQL, MS SQL and Linux servers, each rule tied to one role and one system. The recorded result is a documented role-based rule set and a repeatable way to apply and review it. That is what the record supports; it does not quantify security-risk reduction or setup-time savings.

Keeping the dry run truthful took work of its own. Check mode was made to tell the truth across the whole platform after a dry run in September 2026 stopped on an assertion that was wrong about the host and advised setting a flag that would have relaxed a security sandbox. A sweep found six probes deciding on values the host had never given. Probes that read state the run has not touched now run even under check mode; the rest are skipped with a message naming what went unverified, and the rule went into the development guide.

Where it goes wrong

Most Ansible trouble comes from trusting a report more than the mechanism behind it.

A dry run reports a clean result for commands it never ran. Check mode is a simulation. Modules that support it predict their change; the documentation says modules that lack support report nothing and do nothing. The `command` and `shell` modules support it only partly: unless `creates` or `removes` is given, they skip the command and register a result with empty output. Any later `when:` that reads that result decides on a value nobody measured. A probe that only reads the host can be told to run anyway, and one that cannot run should say so:

```
- name: Read the firewall state, even in check mode
  ansible.builtin.command: ufw status
  register: fw
  check_mode: false
  changed_when: false

- name: Say what this dry run did not verify
  ansible.builtin.debug:
    msg: "check mode: the service restart was not exercised"
  when: ansible_check_mode
```

A run stops on its ninth task with the first eight already applied. When a task fails on a host, Ansible drops that host from the rest of the run and leaves whatever it already changed in place. Checks that can run before anything touches a host belong in front of the converge. That is what happened after a secrets change on this site's own estate: a preflight play now validates the vault locally first, running the same task files as the real converge in about a second, so a secret that does not resolve costs a two-second refusal on a laptop instead of a

partly applied change to a live server. The same page notes a trap: facts set during one play outlive it, so a check and a converge chained into one invocation can share a value the check should have re-read.

A host nobody touched reports changes on every run. The `command` and `shell` modules report a change every time they run, because they have no way to know what the command did. A playbook full of them never converges to zero, and real changes hide in the noise. Prefer a module that manages the thing directly, add `creates` or `removes`, or tell Ansible when output means a change with `changed_when`.

A password turns up in a log or in the repository. Variables are plain text unless something encrypts them. Ansible Vault encrypts variable files or single values so they can live in the repository, and `no_log: true` on a task keeps its arguments and results out of the output. Run the playbook once with high verbosity in a test environment and read what it prints before trusting either.

Further reading

- Ansible Community Documentation — the project's own entry point to installation, concepts and every guide.
- Ansible playbooks — plays, tasks, modules, execution order and what idempotence means in practice.
- Validating tasks: check mode and diff mode — what a dry run can and cannot see, and the per-task switches.
- How to build your inventory — hosts, groups and the variables attached to them.
- Protecting sensitive data with Ansible Vault — encrypting files and single values that live beside the playbooks.
- ansible.builtin.command — the module reference, including its partial check-mode support.



Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is Ansible?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/ansible/