**Infrastructure Drift**

Infrastructure drift happens when what exists (the in-use status of cloud resources) diverges from what has been declared or planned (the infrastructure-as-code single source of truth). Drift occurs by adding manual changes (hot fixing by console/CLI), automated processes update resources (auto-scaling, third-party software), or resources are altered out of IaC workflow (security groups being altered to restore connectivity, for instance).

Drift is a challenge because it ruins reproducibility, breaks change control, makes debugging harder, creates security vulnerabilities (an unstated open port, for instance), and makes disaster recovery unpredictable.

Terraform can prevent and detect drift because it keeps a state file that is a representation of the last-known infrastructure and can diff against real-world declared configuration. terraform plan does a read-only diff of the configuration (and state) against the provider's resources as they exist, revealing what changes would be made; it's the detection step of drift and helps thwart accidental changes by making them explicit. terraform apply takes the plan (or recomputes it) and makes the operations happen in order to sync real resources to match the configuration and, therefore, correct drift. Utilizing both in CI/CD (e.g., terraform plan in PRs and terraform apply after approval) mandates that changes flow through this IaC pipeline instead of editing by hand.

**Best practices to reduce drift include:**

* Implement a single path of change (only changes through Terraform).
* Save state safely (remote backends with locks).
* Use immutable infrastructure patterns (update resources instead of replacing them).
* Execute regular drift detection (scheduled jobs or automated terraform plan in pipelines) to detect out-of-band updates.

Combine Terraform with governance (policies, guardrails) and monitoring so that drift is detected quickly and corrected with minimal service impact.