1 TNTM Agentic Terraform Factory Playbook

Updated: February 27, 2026
Companion article: https://www.talk-nerdy-to-me.com/blog/infra-ai-agent-benchmark-copilot-claude-amazon-q

1.1 Purpose

Build a TNTM-branded system that can:

1. Generate standards-compliant Terraform modules from controlled templates and provider schema checks.
2. Compose IaC stacks from user intent such as “I need 2 web apps and 1 key vault”.
3. Use MCP-enabled tools under explicit policy and approval gates.
4. Keep outputs auditable, reviewable, and production-safe.

1.2 Outcome

A production-minded “agentic IaC factory” where Skills define repeatable build behavior, Agents orchestrate multi-step reasoning, and MCP connects to tools/repositories/docs under governance.

1.3 1) Operating Model

1.3.1 Components

- `Skill A`: Module builder skill (resource-specific module generation)
- `Skill B`: IaC composer skill (intent -> module graph)
- `Agent Orchestrator`: routes work between parser, composer, and module builder
- `MCP Tool Layer`: docs/schema lookup, repo automation, pipeline generation, policy lookup
- `Validation Layer`: fmt, validate, lint, security checks, policy checks
- `Approval Layer`: human gate for risky changes and all infra mutations

1.3.2 Reference topology

```
User intent
  -> intent parser
  -> IaC composer skill
  -> module builder skill
  -> MCP tools
  -> validation gates
  -> human approval (high risk)
  -> PR + evidence
```

1.4 2) Skill Design Blueprint

1.4.1 Skill A: Module builder (`azure-terraform-module-builder`)

Use this skill when someone requests a new module or schema-complete module update.

Mandatory behavior:

- Ask for resource abbreviation before generation.
- Refuse generation if standards profile is missing.
- Pull provider schema before nested block generation.
- Enforce no hardcoded values in example module usage.
- Output complete module repo structure + validation summary.

Starter scaffold:

```
---
name: azure-terraform-module-builder
```

```
description: TNTM skill that generates standards-compliant Azure
        Terraform modules from approved templates and schema lookups.
---

# Trigger
Use when a user requests a new module or asks to update an existing
        module schema.

# Required inputs
- Resource type
- Resource abbreviation
- Standards profile

# Guardrails
- Ask for abbreviation if missing
- Stop if standards profile missing
- Validate generated files before completion
```

1.4.2 Skill B: IaC composer (`iac-composer-from-intent`)

Use this skill when user asks for a solution architecture in natural language.

Mandatory behavior:

- Normalize quantities, dependencies, and environments first.
- Emit machine-readable contract JSON before code generation.
- Map each intent element to approved module source(s).
- Return assumptions explicitly (network model, auth model, env boundaries).

1.5 3) Intent Contract (Required)

```
{
  "request_id": "REQ-2026-02-27-001",
  "environment": ["dev", "staging", "prod"],
  "region": "eastus",
  "intent": {
    "web_app": { "count": 2, "sku": "P1v3", "private_endpoint": true
        },
    "key_vault": { "count": 1, "rbac_enabled": true }
  },
  "constraints": {
    "naming_standard": "TEN_v2",
    "terraform_version": ">=1.5.0",
    "provider_azurerm": ">=4.0.0",
    "no_hardcoded_values": true
  }
}
```

Why this matters:

- Prevents hidden assumptions.

- Makes agent handoffs deterministic.
- Enables repeatable test fixtures for quality checks.

1.6 4) Composition Pattern Example

User request: “I need 2 web apps, 1 key vault, private endpoints, staging and prod.”

Expected composer behavior:

1. Build dependency graph: `app_service_plan -> web_apps`, `key_vault -> web_apps`.
2. Create one reusable module call per deployable unit.
3. Keep all configurable values sourced from variables.
4. Separate state and environment boundaries by design.

Example skeleton:

```
module "app_service_plan" {
  source =
"git::https://dev.azure.com/yourorg/_git/Azure.AppServicePlan.Terraform
  resource_group_name = var.resource_group_name
  location            = var.location
  project             = var.project
  environment         = var.environment
  ten_regional_code   = var.ten_regional_code
  subscription_code   = var.subscription_code
}

module "web_app_01" {
  source =
"git::https://dev.azure.com/yourorg/_git/Azure.WebApp.Terraform"
  service_plan_id = module.app_service_plan.id
  key_vault_id    = module.key_vault.id
}

module "web_app_02" {
  source =
"git::https://dev.azure.com/yourorg/_git/Azure.WebApp.Terraform"
  service_plan_id = module.app_service_plan.id
  key_vault_id    = module.key_vault.id
}

module "key_vault" {
  source =
"git::https://dev.azure.com/yourorg/_git/Azure.KeyVault.Terraform"
}
```

1.7 5) MCP Integration Pattern

1.7.1 MCP role in this design

- Schema/doc lookup without copy-pasting docs into prompts
- Repository and PR automation
- Optional pipeline generation
- Controlled tool execution in the same conversational flow

1.7.2 Policy skeleton

```yaml
policies:
  - name: read-only-discovery
    match:
      actionTier: [1]
    effect: allow

  - name: code-generation
    match:
      actionTier: [2]
      tools: ["repo.write", "pr.create", "terraform.generate"]
    effect: allow_with_conditions
    requirements:
      - standards_profile_present
      - validation_passed

  - name: infrastructure-mutation
    match:
      actionTier: [3, 4]
    effect: require_human_approval
    requirements:
      - approved_change_request
      - ci_executor_only
```

1.8 6) Validation and Release Gates

Minimum gate before merge:

- `terraform fmt`
- `terraform validate`
- `tflint` (or equivalent)
- `tfsec` / `checkov` (or equivalent)
- policy checks (OPA/Sentinel/custom)

Release checklist:

```
- [ ] naming convention enforced
- [ ] required global variables present
- [ ] no hardcoded env values in examples
```

```
- [ ] dependency graph documented
- [ ] generated assumptions listed
- [ ] high-risk actions require approval
- [ ] all tool actions logged
```

1.9 7) Rollout Plan

1.9.1 Phase 1: Skill foundation

- build module builder skill
- build IaC composer skill
- lock template outputs for deterministic structure

1.9.2 Phase 2: Agent orchestration

- define parser -> composer -> generator handoff contract
- persist normalized intent JSON
- require rationale output on key design choices

1.9.3 Phase 3: MCP tools

- add read-only tools first
- then controlled code-generation tools
- keep infra mutation paths approval-gated

1.9.4 Phase 4: Scale and governance

- track escaped defects and rollback rate
- measure module adoption lead time
- widen scope only when quality stays stable

1.10 8) Zara/TNTM Tone Guidance (for published output)

Use Zara Lab + Blueprint voice:

- clear, direct, technical
- explicit assumptions
- no vague “AI magic” framing
- always end with a concrete next action

1.11 9) Sources and Verification

Verified on February 27, 2026.

- Anthropic MCP launch: https://www.anthropic.com/news/model-context-protocol

- MCP architecture docs: https://modelcontextprotocol.io/docs/learn/architecture
- GitHub coding agent + MCP: https://docs.github.com/en/copilot/how-tos/use-copilot-agents/coding-agent/extending-coding-agent-with-mcp
- Amazon Q Developer MCP usage: https://docs.aws.amazon.com/amazonq/latest/qdeveloper-ug/use-mcp.html
- Terraform module development docs: https://developer.hashicorp.com/terraform/language/modules/develop
- Terraform style guide: https://developer.hashicorp.com/terraform/language/style
- AGENTS.md guidance: https://openai.github.io/openai-agents-js/guides/agents-md/

1.12 10) Implementation Checklist (TNTM)

- Add or update skill folders and `SKILL.md` definitions.
- Add orchestrator prompt templates and intent contracts.
- Add MCP server configs with policy tiers.
- Add CI checks and PR templates for generated outputs.
- Test with at least 5 real requests from recent infra work.