turf.build

REFERENCE ARCHITECTURE · V1.1 · OCTOBER 2026

Metal to Tokens

A reference architecture for managed GPU cloud infrastructure, as code.



For the teams who bring up, certify, sell and service managed GPU clusters.

Goal: full orchestration and remediation

A neocloud doesn't sell GPUs by the hour. It sells a managed cluster: qualified before handover, current on drivers and firmware, the same on every node, and back in service soon after a fault.

That is what the market grades. SemiAnalysis's ClusterMAX 3.0 rates GPU clouds on the operating discipline above the hardware, and it sets the bar in two sentences: “Handling a failure takes two things: Identify it happened. Fix it properly.”

Today that discipline is spread across tools. Terraform® for the cloud API. Ansible® for the hosts and devices. Helm® for the runtime. Scripts for burn-in. A runbook in a wiki, and a person on call to stitch them together.

> **Turf™ is cluster orchestration software for neocloud providers, built on OpenTofu™.**

One convergence loop brings up the whole managed cluster, from the metal to the tokens, NVCRE certification included. Every change, from a person or an agent, arrives as a change plan that is checked and approved before it runs. Turf deploys the stack and then services it, across a fleet of clusters.

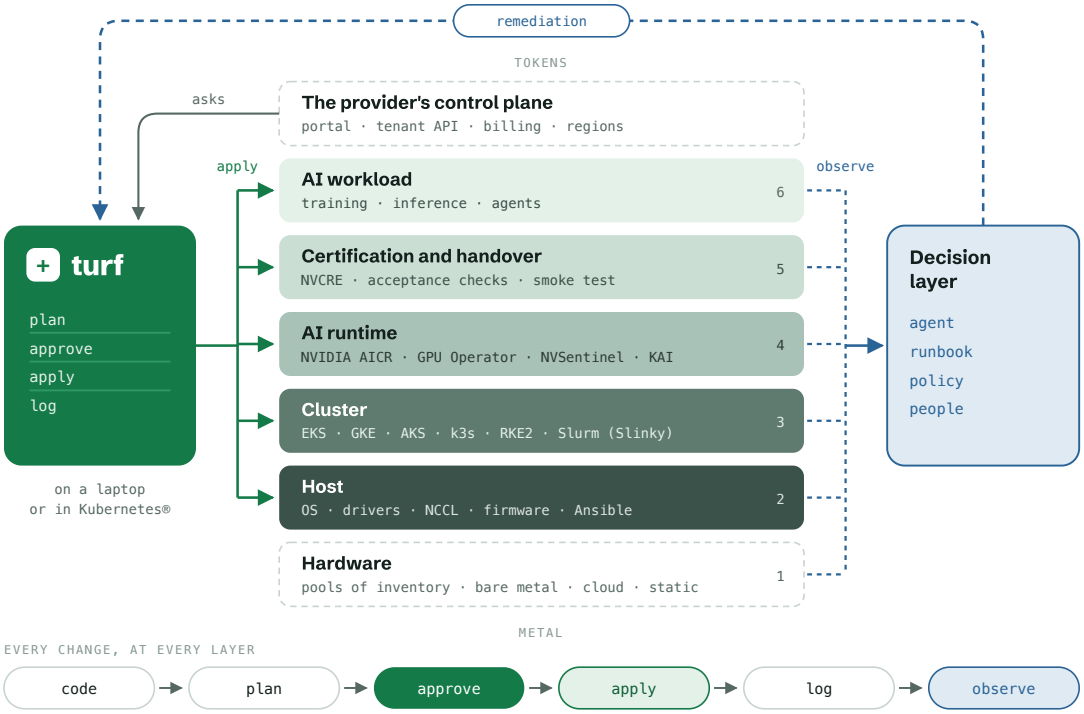
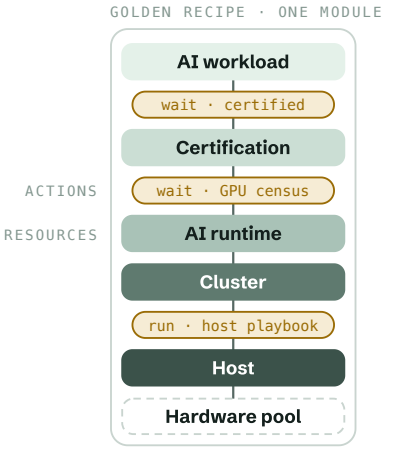


Figure 1. Metal to tokens. Every layer is code, brought up by one convergence loop. Turf runs on a laptop or in a Kubernetes® cluster. The decision layer (the provider's agent, runbook and policy) approves each change plan and oversees remediation. The dashed layers are the provider's own: hardware, offered as pools of inventory on day 0, and the control plane that asks Turf for clusters.

At a glance

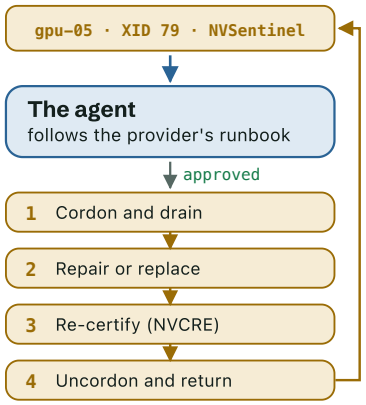
Turf builds and operates managed clusters in three ways.



A golden recipe

The managed cluster is one body of Terraform code: the provider's whole stack, from the hardware pool to the tenant's agent. Terraform resources are the desired state at every layer. Terraform actions are the steps between them: run the host playbook, wait for the GPU census. One change plan covers all of it.

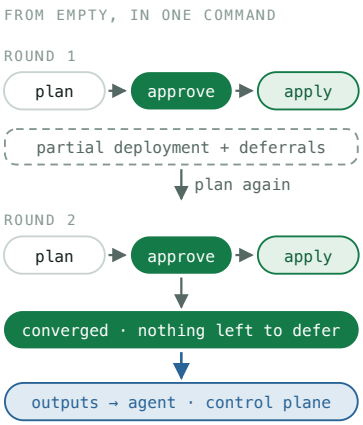
→ Chapters 7 and 12



A day-2 runbook

The health checks find the fault. What happens next is the provider's remediation runbook, and the agent follows it. Each step is a change plan, approved by policy or a person and carried out by the engine, with the agent overseeing.

→ Chapters 10 and 11



A convergence loop

Turf plans what it can know, applies it, and plans again. Every round's change plan is checked and approved before it runs. Then the engine reports the outcome, and the outputs, to the agent and the provider's control plane.

→ Chapter 8

This paper walks the cluster stack three times. **Part I** zooms in, one layer at a time. **Part II** zooms out to the convergence loop. **Part III** zooms out again, to day-2 across the data center.

PART I
ZOOM IN

01

TOKENS

HARDWARE

METAL

DAY 0 · THE PROVIDER'S

Hardware: pools of inventory

Day 0 is the provider's. The servers are racked and cabled, the fabric is up, and every machine's BMC answers. What the layers above need from the hardware is a pool of machines to draw from.

A provider offers that pool in one of three ways.

- **Bare metal as a service.** MAAS, Tinkerbell, Metal3 or Ironic enrolls each machine, commissions it over its BMC, and deploys an operating system on demand.
- **Infrastructure as a service.** A cloud API hands out GPU instances.
- **A static list.** Hosts that are already installed, named in an inventory.

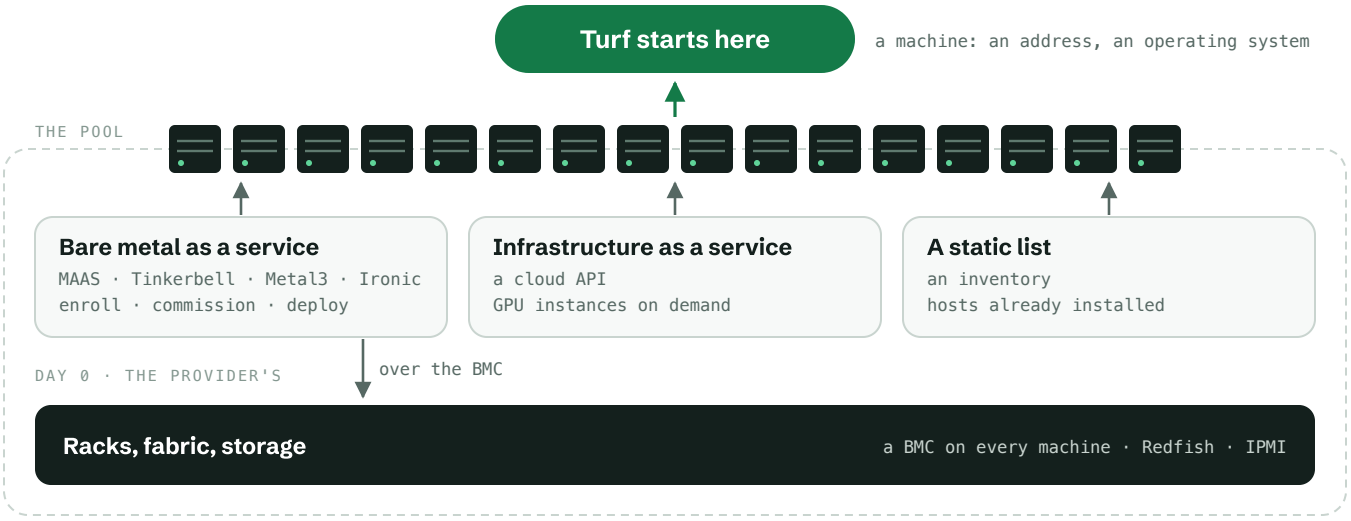


Figure 2. Pools of inventory. Hardware reaches the stack as a pool of machines, offered by a bare-metal service, a cloud API or a static list.

Redfish is the standard API on a modern BMC: power, boot order, BIOS settings, firmware. The bare-metal services speak it. Dell's Terraform provider drives it directly, and the Ansible Redfish modules reach the rest, so a BIOS setting or a firmware update is code too.

Turf starts where the pool hands over a machine: an address to reach it, and an operating system to configure. The layers above don't care which pool it came from.

The fabric and the storage are day 0 as well. What changes for each tenant after that is code like the rest, through the Terraform providers that vendors publish (Cisco, Juniper, Palo Alto Networks, VAST, NetApp) and through Ansible where there is none (Arista AVD, NVIDIA NVUE).

DAY 1 · BRING-UP

The host

The faults that sink a cluster's grade are mostly host configuration. PCIe ACS left on. GPUDirect RDMA not enabled. An old kernel, a stale driver, a missing line in `nccl.conf`. Tooling that isn't installed.

Terraform providers stop at the machine's API. Past it, most operators configure the host with Ansible, on the side. **Turf brings the playbook inside.** A machine from the pool becomes the inventory, and the playbook runs as a Terraform action as soon as the host is deployed, before a cluster is built on it. A changed variable runs it again under the same approval. Hosts that Ansible discovers for itself work the same way.

The acceptance check for this layer can be SemiAnalysis's own audit. Run `cmax audit` as the last play and stop on a failure.

```
terraform {
  required_providers {
    ansible = {
      source  = "ansible/ansible"
      version = "1.5.0"
    }
  }
}

action "ansible_playbook_run" "gpu_node" {
  config {
    playbooks   = ["${path.module}/gpu-node.yml"]
    inventories = [data.ansible_inventory.gpu.json]
  }
}
```

```
# gpu-node.yml
- name: Configure a GPU node
  hosts: gpu_nodes
  become: true
  tasks:
    - name: Enable GPUDirect RDMA
      community.general.modprobe:
        name: nvidia_peermem
        persistent: present

    - name: Point NCCL at the InfiniBand adapters
      ansible.builtin.lineinfile:
        path: /etc/nccl.conf
        line: NCCL_IB_HCA=mlx5

    - name: Audit the node
      ansible.builtin.command: cmax audit
      changed_when: false
```

The Ansible provider, from the public registry, and the host playbook as its action. Illustrative, after turf-examples/use-cases/datacenter/ansible-k3s.

DAY 1 · BRING-UP

The cluster

Kubernetes, Slurm, or Slurm on Kubernetes. Managed Kubernetes from the hyperscalers and the GPU clouds (EKS, GKE, AKS), or bare metal with k3s or RKE2.

On a cloud, the cluster and its GPU node groups come from a Terraform module. Below is an EKS cluster with a pool of eight-GPU H100 nodes, tainted so only GPU work lands there.

On bare metal the cluster comes from the same ecosystems. Terraform modules such as `xunleii/k3s/module` and Ansible playbooks such as k3s-ansible and Kubespray build Kubernetes on the hosts from chapter 2. Slurm runs on Kubernetes through Slinky, with NVIDIA's KAI Scheduler for queues and quotas.

```
module "eks" {
  source  = "terraform-aws-modules/eks/aws"
  version = "~> 21.0"

  name               = "gpu-west-1"
  kubernetes_version = "1.35"
  vpc_id             = module.vpc.vpc_id
  subnet_ids         = module.vpc.private_subnets

  eks_managed_node_groups = {
    h100 = {
      ami_type       = "AL2023_x86_64_NVIDIA"
      instance_types = ["p5.48xlarge"]
      min_size       = 2
      max_size       = 16
      desired_size   = 8
      labels         = { "nvidia.com/gpu.present" = "true" }
      taints = {
        gpu = { key = "nvidia.com/gpu", value = "true", effect = "NO_SCHEDULE" }
      }
    }
  }
}
```

A GPU cluster on EKS: eight p5.48xlarge nodes to start, room for sixteen. Written against terraform-aws-modules/eks/aws v21.

DAY 1 · BRING-UP

The AI runtime

A GPU cluster's runtime is dozens of components on their own release cycles: kernels, drivers, container runtimes, operators, frameworks. A combination that works for one service, GPU and Kubernetes release can fail on another, and which ones work has lived in validation systems, deploy scripts and runbooks.

NVIDIA AI Cluster Runtime (AICR) answers with version-locked, validated recipes. A recipe is chosen by service, GPU, operating system, workload intent and platform. It pins the components that work together and carries signed validation evidence from the hardware it was tested on. AICR v1.0 ships recipes for EKS, GKE, AKS, OKE and LKE, OpenShift, RKE2, k0s and bare metal, A100 to VR200.

A recipe describes the desired configuration. It does not orchestrate a cluster. That half is Turf's. The AICR renders the recipe as a Terraform module for Turf to deploy: planned, approved and recorded. An upgrade is a new module version. Drift from the recipe is a reviewable diff.

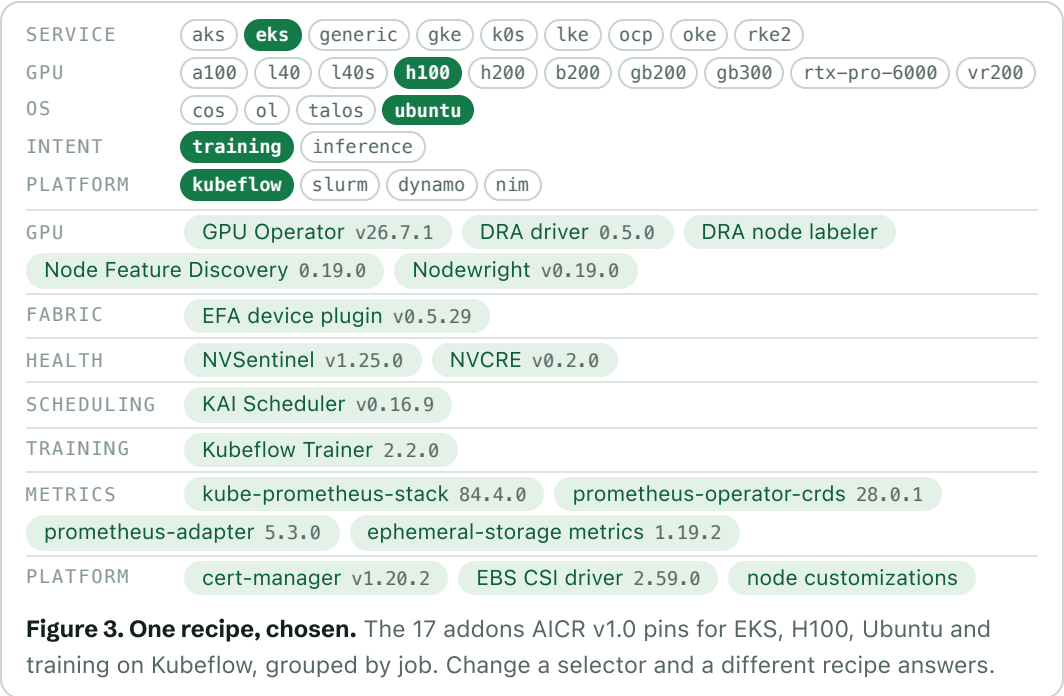


Figure 3. One recipe, chosen. The 17 addons AICR v1.0 pins for EKS, H100, Ubuntu and training on Kubeflow, grouped by job. Change a selector and a different recipe answers.

```
module "eks" { … }                              # the cluster, chapter 3
module "runtime" {
  source     = "registry.example.com/platform/aicr-runtime/eks"
  version    = "1.0.0"                          # h100 · ubuntu · training · kubeflow
  depends_on = [module.eks]
}
module "certification" { … }                    # chapter 5, once the runtime is up
```

The provider's golden recipe, abridged: cluster, the rendered recipe as one module call, certification. Rendered once with `aicr bundle --deployer terraform`. Illustrative.

PART I
ZOOM IN

05

TOKENS

CERTIFICATION

METAL

DAY 1 · BRING-UP

Certification and handover

NVCRE, NVIDIA's Cluster Readiness Engine, "certifies GPU clusters before production workloads run on them". It runs categories such as NCCL all-reduce, DCGM level 4 diagnostics and a short training run, and reports a verdict. "Quarantine is left to your platform."

Turf is that platform step. The certification is a Terraform resource like any other, with checks on either side of it.

- **Before:** a GPU census waits until every expected node is Ready, schedulable and advertising its GPUs.
- **After:** Turf waits for the certification to finish. A failed certification is marked to run again, and nothing downstream runs on a cluster that failed.
- **Then:** a smoke test, the verdict as outputs for the decision layer, and handover.

The certification is tied to what it certified: the nodes, the pools, the recipe and the spec. Change any of them and it runs again. That is acceptance testing at bring-up, and continuous testing on every change after it.

```
resource "kubernetes_manifest" "certification" {
  manifest = {
    apiVersion = "nvcre.nvidia.com/v1alpha1"
    kind       = "Certification"
    spec       = local.spec
  }
  lifecycle {
    action_trigger {
      events  = [before_create]
      actions = [action.kubewait_condition.gpu_census]
    }
    action_trigger {
      events     = [after_create]
      actions    = [action.kubewait_condition.certification_complete]
      on_failure = taint
    }
  }
}
```

Adapted from turf-examples/use-cases/datacenter/aicr-eks-certification

```
TOOL CALL · TURF MCP
✓ outputs · gpu-west-1
certification = {
  verdict = "Succeeded"
  nodes   = 8
  "communication/nccl-all-reduce" = "Succeeded"
  "diagnostics/dcgm-level4"       = "Succeeded"
  "training/nemotron5-8b"         = "Succeeded"
}
smoke_test = "Complete"
```

THE PROVIDER'S AGENT — DECISION LAYER

Certification passed. `gpu-west-1` is ready to hand over.

✓ NCCL all-reduce	8 of 8	✓ DCGM level 4	8 of 8
✓ Training run	8 of 8	✓ Smoke test	complete

Next in the runbook is handover. I added the tenant's access to the configuration. Approve the change plan?

Approve | Review the change plan

Illustrative. The stack's outputs carry the verdict, and the provider's agent reads them through Turf's MCP server.

PART I
ZOOM IN

06

TOKENS
AI WORKLOAD
METAL

DAY 1 · BRING-UP

The AI workload

At the top of the stack are the tokens: training jobs, inference endpoints, agents. They are why the cluster exists, and they are code like the rest.

An agentic workload is a few Terraform resources. A model served on the cluster's GPUs, here an NVIDIA NIM run by the NIM Operator from AICR's inference recipe, turns GPU hours into tokens. A tenant's agent spends them: a kagent Agent with a system message, its tools and that model.

Turf waits until the endpoint is ready before it creates the agent. Training has the same shape: a Kubeflow TrainJob is one resource.

```
resource "kubernetes_manifest" "model" {
  manifest = {
    apiVersion = "apps.nvidia.com/v1alpha1"
    kind       = "NIMService"
    metadata   = { name = "llama-3-2-1b", namespace = "inference" }
    spec = {
      image = {
        repository = "nvcr.io/nim/meta/llama-3.2-1b-instruct"
        tag        = "1.8.3"
      }
      authSecret = "ngc-api-secret"
      resources  = { limits = { "nvidia.com/gpu" = "1" } }
      expose     = { service = { type = "ClusterIP", port = 8000 } }
    }
  }

  lifecycle {
    action_trigger {
      events     = [after_create]
      actions    = [action.kubewait_condition.model_ready]
      on_failure = taint
    }
  }
}

resource "kubernetes_manifest" "agent" {
  depends_on = [kubernetes_manifest.model]
  manifest = {
    apiVersion = "kagent.dev/v1alpha2"
    kind       = "Agent"
    metadata   = { name = "triage", namespace = "kagent" }
    spec = {
      type = "Declarative"
      declarative = {
        modelConfig   = "llama-nim"
        systemMessage = "You triage failing jobs on this tenant's cluster."
        tools         = local.triage_tools
      }
    }
  }
}
```

A NIM endpoint on one GPU, and a tenant's agent that calls it. Illustrative, written against NIM Operator 3.1 and kagent v0.10.

PART II
ZOOM OUT

07

THE CLUSTER AS CODE

The unified planner

Zoom out, and Part I is one body of code.

A Terraform plan carries two kinds of change. **Resources** are desired state: a node group, a host configuration, a Helm release, an inference service. **Actions** are the steps in between: run a playbook, wait for a census, drain a node. Both are written in the Terraform language and reviewed the same way.

Every change plan is approved before it runs. Policy, written as code or instruction, approves routine changes on its own: the version floor, the tenancy rules, the change window. A person approves what the provider says a person approves.

Every effect leaves a record, and the record is the audit trail.

```
Turf will perform the following actions:

  # module.runtime.module.gpu_operator.helm_release.this will be updated in-place
  ~ resource "helm_release" "this" {
        id      = "gpu-operator"
        name    = "gpu-operator"
      ~ version = "v26.7.0" -> "v26.7.1"
        # (24 unchanged attributes hidden)
    }

  # module.certification.kubernetes_manifest.certification will be replaced due to changes in replace_triggered_by
  ∓ resource "kubernetes_manifest" "certification" {
      ~ object = { ... } -> (known after apply)
    }

    # Actions to be invoked before this change in order:
    # module.certification.action.kubewait_condition.gpu_census

    # Actions to be invoked after this change in order:
    # module.certification.action.kubewait_condition.certification_complete

Plan: 1 to update, 1 to replace. Actions: 2 to invoke.
```

A recipe upgrade to AICR v1.0. The GPU Operator moves to v26.7.1, and the certification is replaced, with the census before it and the wait after. Abridged and illustrative, after turf-examples/…/aicr-eks-certification.

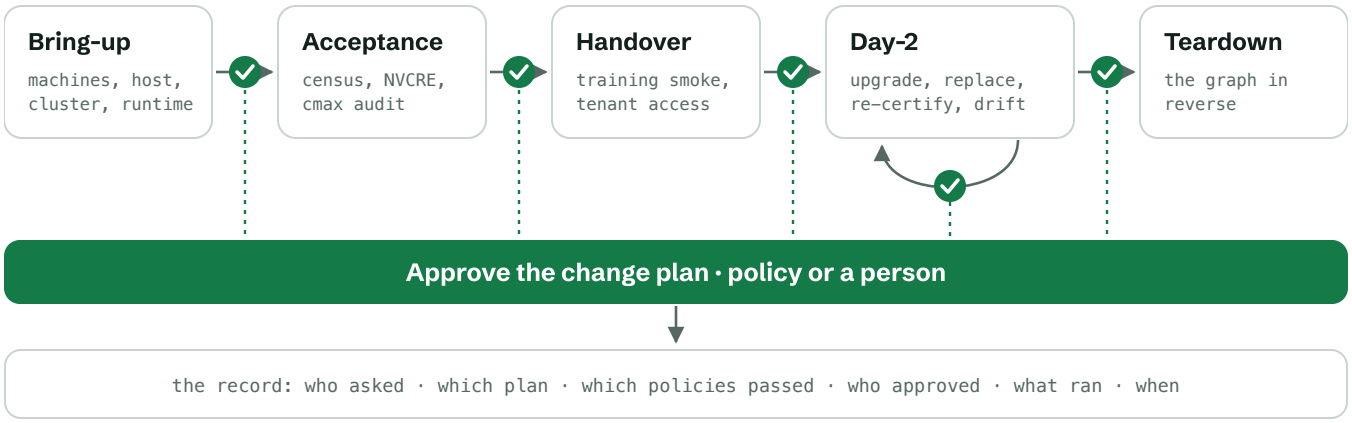


Figure 4. The managed-cluster lifecycle. Every transition is a change plan, approved and recorded.

PART II
ZOOM OUT

08

THE CLUSTER AS CODE

The convergence loop

Turf runs one loop: plan, approve, apply, and plan again.

Each round plans everything it can know. What it can't know yet is deferred to the next round: a host's address before the host exists, a cluster's endpoint before the cluster, an addon's types before the addon. Every round's change plan is approved like any other. **A round with nothing left to defer ends the run**, converged, from empty, in one command.

The Turf console shows the loop as it runs. Runs sit on the left, newest on top. Each round's milestones run across the top, and every resource has its own lane, grouped by module.

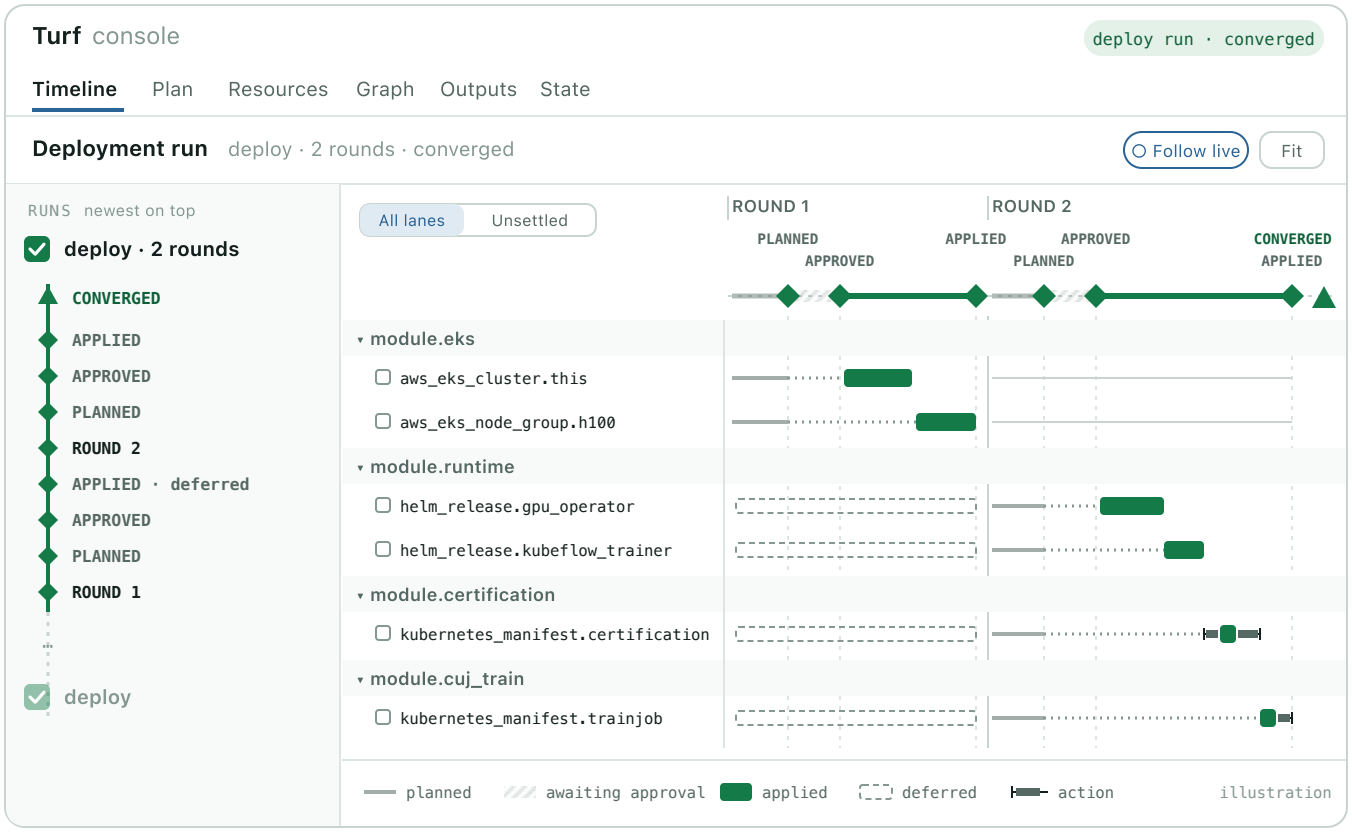


Figure 5. The convergence loop, in the Turf console. Round 1 builds the EKS cluster and defers everything that runs on it. Round 2 plans the deferred work, its change plan is approved, and the run converges.

PART II
ZOOM OUT

09

THE CLUSTER AS CODE

Built on three ecosystems

Nothing in this stack is written from nothing. Terraform providers and modules, Ansible playbooks and Helm charts already cover it, and Turf runs each as it is.

- **Terraform providers and modules.** 7,388 providers and 24,728 modules on the public registry reach clouds, bare-metal services, switch fabrics, storage arrays and Kubernetes. Modules package them into golden recipes.
- **Ansible playbooks.** 4,568 collections on Ansible Galaxy cover what no API reaches: drivers, firmware, kernel settings, switch operating systems, Kubernetes on bare metal. A playbook runs as a Terraform action.
- **Helm charts.** 18,096 charts on Artifact Hub are how the runtime ships: the GPU Operator, NVSentinel, KAI Scheduler, Kubeflow Trainer. A chart installs as a Terraform resource.

That breadth is what a ClusterMAX grade looks at: lifecycle, orchestration, monitoring, reliability and security, plus the configuration side of storage and networking.

	Terraform providers and modules 7,388 providers · 24,728 modules	**Ansible** playbooks 4,568 collections on Galaxy	**Helm** charts 18,096 on Artifact Hub
AI WORKLOAD	Kubernetes manifests	–	NIM Operator Kubeflow Trainer · kagent
CERTIFICATION	kubewait	cmax audit	NVCRE
AI RUNTIME	AICR recipe module	–	GPU Operator NVSentinel KAI Scheduler Prometheus®
CLUSTER	EKS · GKE · AKS · k3s · Talos	k3s-ansible · Kubespray RKE2	Slinky (Slurm)
HOST	Ansible provider	GPU drivers · RDMA NCCL · kernel settings	–
HARDWARE	MAAS · Redfish · cloud GPUs Apstra · NX-OS · PAN-OS · VAST NetApp	Redfish · Arista AVD NVIDIA NVUE	–
IN TURF	as resources and modules	as a Terraform action	as a Terraform resource

Figure 6. Three ecosystems, layer by layer. Examples of what each already carries, named as text. Counts are from the public registries, October 2026.

PART III
ZOOM OUT AGAIN

10

DAY 2

Day-2 runs on the runbook

Day-2 is where a managed cluster earns its grade. Two health-check loops watch it. The passive loop (DCGM, kernel logs, XID errors, NVSentinel) watches every node all the time. The active loop runs NVCRE on idle nodes, on a schedule. Detection belongs to them.

> **“The important part is the mapping from each XID to the action the provider takes.”**
>
> SEMIANALYSIS, CLUSTERMAX 3.0

That mapping is the provider's remediation runbook, and the agent follows it.

1. Cordon and drain the node, as actions.
2. Repair or replace it: a new instance, the host playbook again, a firmware reflash.
3. Re-certify it.
4. Uncordon it and return it to service.

The runbook names the checks along the way: the version floor, `cmax audit`, a policy check on the exported change plan. Approval comes from whoever the runbook says. Policy alone for a routine replacement. A person for a change a tenant will notice.

Each step is an effect with a record. A drain that fails halfway leaves the node marked not done, and the next run starts again from the cordon. No node stays cordoned with nobody remembering why.

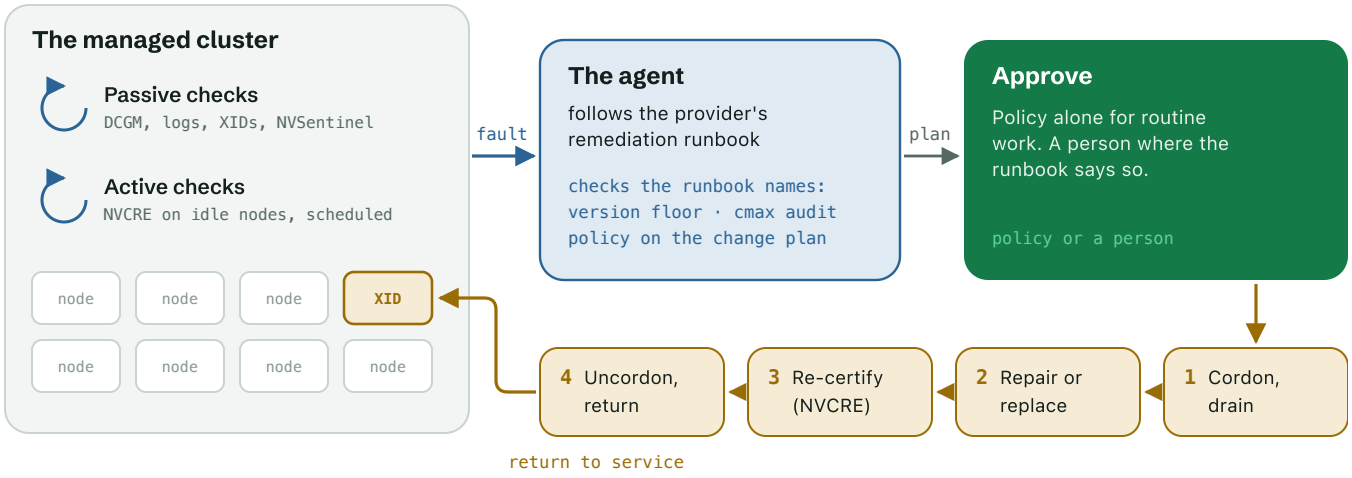


Figure 7. Two loops and one runbook. Detection belongs to the checks. The decision belongs to the provider's runbook. Turf carries it out, one approved, recorded step at a time.

PART III
ZOOM OUT AGAIN

11

DAY 2

The decision layer

The provider brings its own AI: its model, its skills, its MCP tools, its runbooks and its policies. Turf gives that agent one way to touch infrastructure, and that way runs through a change plan.

ClusterMAX 3.0 doesn't expect “upstart neoclouds with vibecoded datacenter automations” to threaten the leaders any time soon. An agent with a shell can do anything, once, with no record. With Turf, the agent edits the configuration and triggers a replan. The Turf engine prepares a change plan, policy checks it and someone approves it. **Then the engine carries it out, with the agent overseeing.** The agent watches each effect, and on a failure it edits the configuration and triggers a replan. Every effect is recorded.

The same loop covers the rest of day-2.

- **Drift.** Every run re-reads the stack, so a changed setting or a removed addon shows up as a change.
- **Upgrades.** A new version of the recipe's Terraform module, rolled out as a change plan and re-certified on the way.
- **Continuous testing.** The certification on a schedule, and on demand.
- **Retirement.** The graph in reverse: workloads drain before the cluster goes.

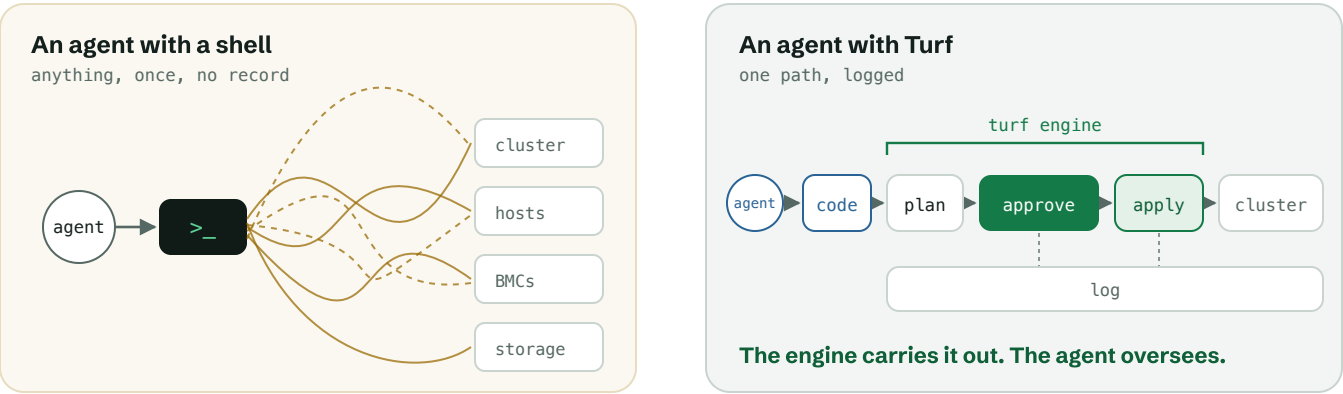


Figure 8. Vibecoded and governed. The same agent, with a shell or with Turf.

```
EVENT
NVSentinel · node gpu-05 · XID 79

TOOL CALLS · TURF MCP
✓ plan      2 to replace · 5 actions
✓ approve   policy · routine replacement
✓ apply     cordon · drain · replace
            re-certify · uncordon
✓ outputs   certification = "Succeeded"
```

THE PROVIDER'S AGENT — DECISION LAYER

gpu-05 is back in service. Every step is in the record.

1. NVSentinel reported XID 79. The runbook says replace it.
2. I tainted it and triggered a replan.
3. Policy approved the change plan as a routine replacement.
4. The engine drained, replaced and re-certified the node.

Illustrative. A day-2 fault, in the agent's words.

PART III
ZOOM OUT AGAIN

12

DAY 2

Thousands of stacks

Zoom out once more. A fleet is many stacks from a few golden recipes.

The fleet runs on GitOps. The golden recipe is a Terraform module in a registry. A tenant's cluster is a call to it with a few variables, and a change to either arrives as a pull request.

The provider's control plane does the asking. Above the fleet sit the portal, the tenant API and billing. A tenant's order becomes a few lines of HCL, merged by pull request. The control plane then calls Turf's MCP server to open the tenant's workspace, plan, and read the outputs back when the run converges. The decision layer approves the change plan like any other.

Turf runs as a service in Kubernetes, on a management cluster in the data center beside the racks it runs. Each stack has its own workspace, state and record, and the operator console shows every run in flight.

```
module "tenant_acme" {
  source  = "registry.example.com/platform/gpu-cluster/kubernetes"
  version = "2.4.0"
  tenant    = "acme"
  recipe    = "h100-inference"
  gpu_nodes = 16
}
```

A tenant's cluster: one call to the golden recipe, with the variables the provider opens. Illustrative.

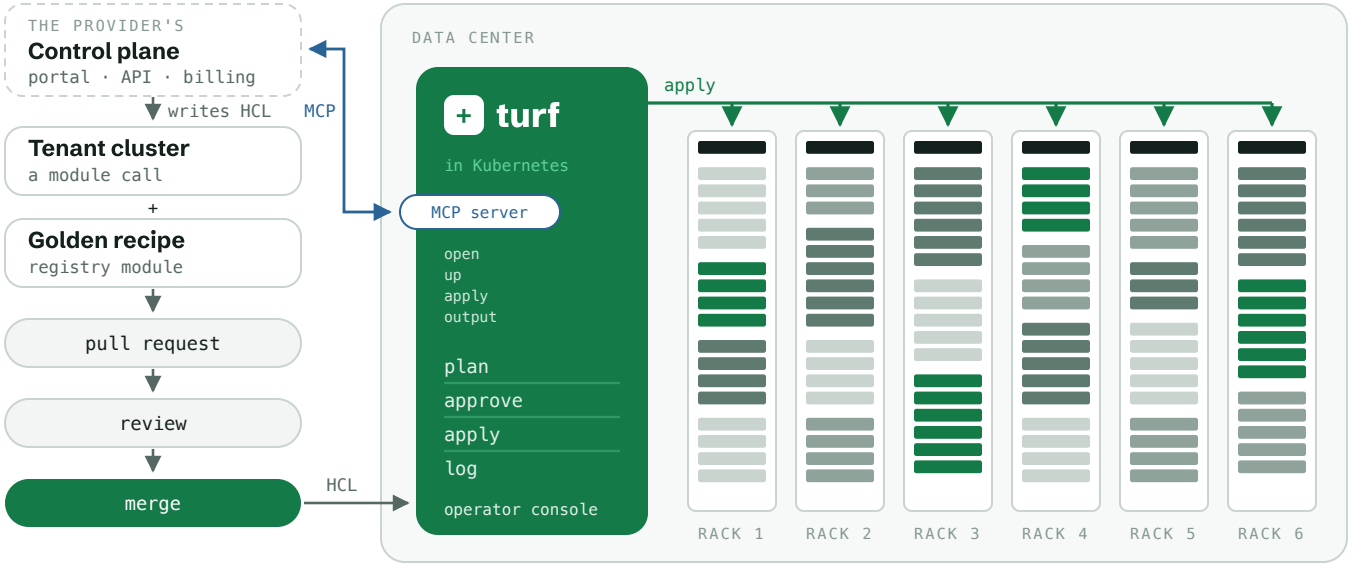


Figure 9. A tenant order, in the data center. Each block is a tenant cluster, and the green ones have a run in flight.

UNDERNEATH

The Turf engine

One engine runs every stack in this paper, from a laptop to the data center.

Laptop or Kubernetes

Runs on a workstation, or as a service in the cluster.

Durable execution with a full record

Every effect is recorded as it happens, and an interrupted run picks up where it stopped.

Built-in OpenTelemetry

Traces and metrics for every change and every effect.

Operator console

Every stack, run and approval in one place.

The Terraform language

Configuration in HCL, with registry modules and providers.

Terraform Actions

Lifecycle triggers and failure handling, the imperative steps between resources.

Policy and governance

Change plans export for the provider's own policy tools to check. Nothing runs until one is approved.

An MCP server and skills

For any agent, and for the provider's own control plane.

Encrypted at rest

State and the engine's store.

Convergence

Plans what it can, defers the rest, and plans again until nothing is left.

Upstream. Turf is one of the organizations contributing to NVIDIA AICR, a project NVIDIA counts at over 100 contributors, almost half from outside NVIDIA. Turf contributed AICR's Terraform deployer, so a recipe renders straight to a Terraform module, its dependency order intact.

`turfbuild/kubewait` is an open-source Terraform provider whose one action waits on Kubernetes conditions, counts and drains. Version 0.1.0 is on the public registry.

GET TURF

Download it. Run it.

Turf is available now, as a download. No form, no login. Free for noncommercial use.

```
$ curl -fsSL https://get.turf.build/install.sh | bash
```

The stacks in this paper are in `github.com/turfbuild/turf-examples`. Commercial use, including a company's internal use, requires a license. Companies evaluate Turf under a free pilot: turf.build/licensing. Turf sends anonymous usage telemetry to Turf LLC by default: the version, the platform, counts of resources, changes and runs, and an anonymous installation id. It never sends resource contents, credentials, state, variable values or names. Turn it off with `turf telemetry off`, or with `TURF_TELEMETRY=off` or `DO_NOT_TRACK=1` in the engine's environment; the full field list and the endpoint are at turf.build/telemetry.

About Turf

Turf™ is cluster orchestration software for neocloud providers, built on OpenTofu™. Every change, from a person or an agent, is a change plan that is checked, approved and recorded. Turf deploys and services complete AI factory stacks across fleets of clusters.

turf.build

Community

Questions, ideas and show and tell: turf.build/discord. Bugs and feature requests: github.com/turfbuild/turf/issues. Release notes by email: turf.build/newsletter; follow Turf on LinkedIn (linkedin.com/company/turfbuild) and X (@turfbuild). Anything else: contact@turf.build. Security reports, privately: security@turf.build.

Turf™ is a trademark of Turf LLC. Terraform® is a registered trademark of HashiCorp, Inc. Ansible® is a registered trademark of Red Hat, Inc. in the United States and other countries. Helm®, Kubernetes® and Prometheus® are registered trademarks, and OpenTofu™ is a trademark, of the Linux Foundation. Turf is not affiliated with, endorsed by, or sponsored by HashiCorp, IBM, Red Hat, the Linux Foundation or the OpenTofu project. NVIDIA, AI Cluster Runtime, NVCRE, NVSentinel, KAI Scheduler and NIM are trademarks or product names of NVIDIA Corporation, referred to here only to describe interoperability. Turf is not affiliated with, endorsed by, or sponsored by NVIDIA. Amazon EKS, Google Kubernetes Engine and Azure Kubernetes Service are trademarks of their respective owners. MAAS is a trademark of Canonical Ltd. Redfish is a trademark of DMTF. ClusterMAX is SemiAnalysis's rating of GPU clouds. Quotations are from ClusterMAX 3.0 (September 2026) and are attributed. Turf has no relationship with SemiAnalysis, and nothing here implies a rating. All other marks belong to their owners.

Metal to Tokens, v1.1, 2026-10-31.