turf.build

REFERENCE ARCHITECTURE · V0.10 · NOVEMBER 2026

Agentic IaC for Platform Teams

Infrastructure as code with Turf™ on kagent 1.0. Ask in chat, approve the plan, and your cluster provisions it: cloud resources, clusters, anything with a Terraform® provider.



Runs on kagent 1.0 and Kubernetes®.

HOW IT FITS TOGETHER

The cluster is the control plane

kagent 1.0 runs an agent in your cluster. Turf™, an infrastructure-as-code toolset built on OpenTofu®, gives that agent Terraform® plan and apply as MCP tools. Ask in the kagent chat for cloud infrastructure, and it comes back as a plan for you to approve.

The agent plans. A person approves. Only the engine touches the cloud.

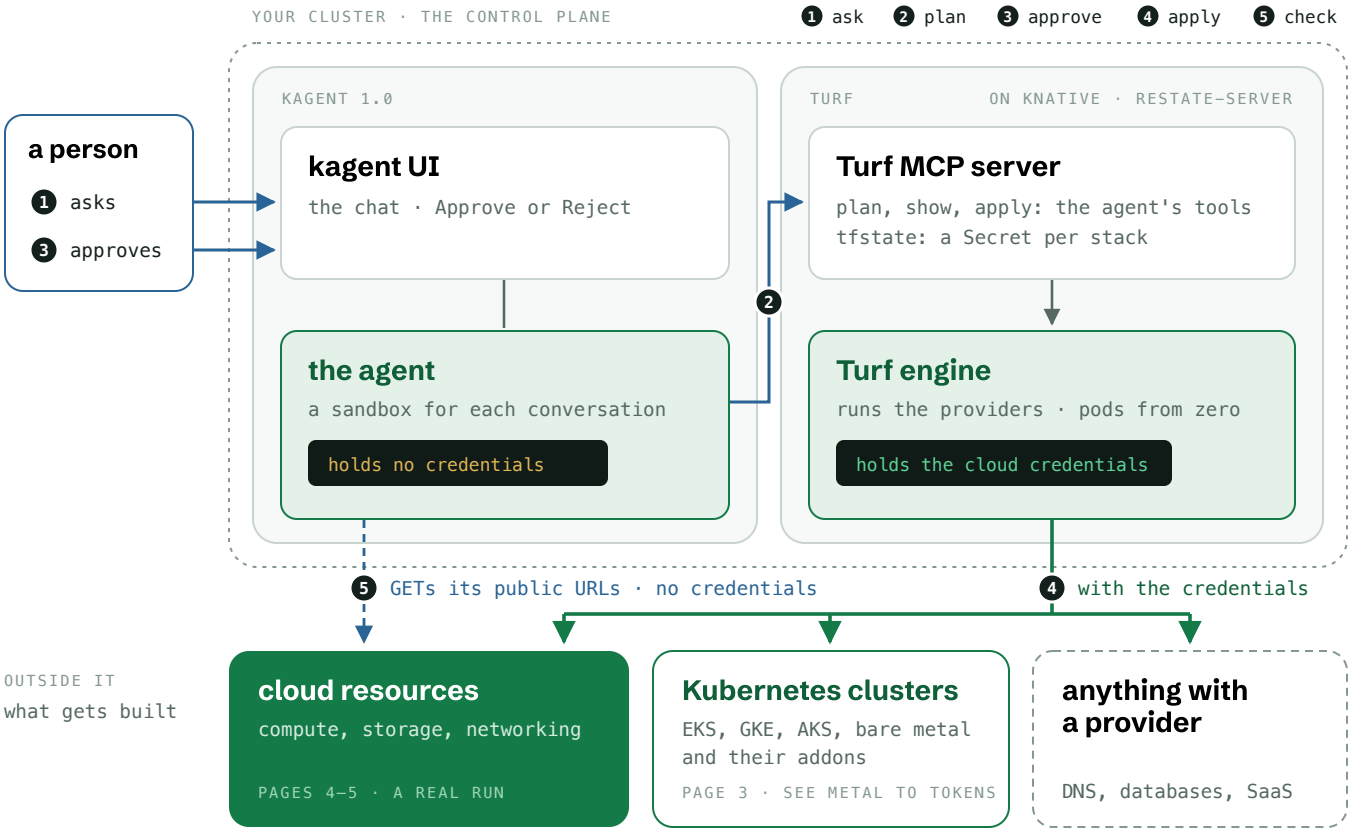


Figure 1. What runs where. The cloud credentials stay with Turf's engine. The agent's sandbox holds none: it can propose a plan, which waits for a person, and afterwards fetch a public URL to check the result.

```
kind: AgentTemplate                    # the agent: manifests/kagent/agent.yaml in turf-kagent, trimmed
systemPrompt: |
  You are turf, an infrastructure-as-code agent in a Kubernetes cluster.
  The cluster is the control plane: what you build with it is outside it.

  A catalog of Terraform configurations is available to you, read-only. You
  operate what it holds; you do not author Terraform.

  Use Turf's `plan` and `apply` tools to create, change and destroy what a
  configuration declares. A person approves every `apply` before it runs.

  Turf's engine holds the cloud credentials. You hold none and need none.

  Never report a success you did not observe. …
tools:
  - mcp: {server: {name: turf}, tools: [init, open, up, destroy, plan, show, …]}
  - mcp: {server: {name: turf}, tools: [apply, taint, untaint, delete], requireApproval: true}
```

WHAT IT IS FOR

One management cluster, a whole fleet

One management cluster makes a fleet of workload clusters and keeps watch over it. The layout is familiar from Cluster API; Turf builds it with Terraform modules and Terraform providers instead. Each cluster is a stack made from a Terraform module: ask for one in the chat and approve its plans. With drift detection on, Turf watches it from then on.

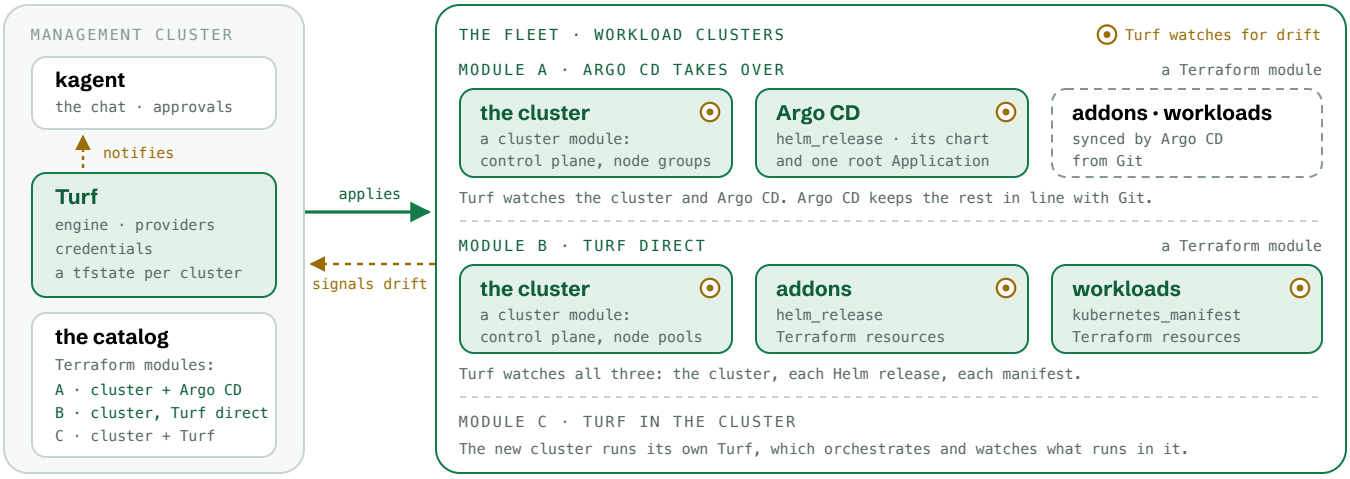


Figure 2. A fleet from three Terraform modules. When something drifts, a watching agent gets a notice and proposes a plan, approved like any other. *Metal to Tokens*, a companion paper, builds one such module, its golden recipe, from hardware to a served model.

```
# module A · Argo CD takes over
terraform {
  required_providers {
    aws  = { source = "hashicorp/aws" }
    helm = { source = "hashicorp/helm" }
  }
}
module "cluster" {
  source = "terraform-aws-modules/eks/aws"
}
# Argo CD itself
resource "helm_release" "argocd" {
  chart = "argo-cd" …
}
# a root Application: Git from here
resource "helm_release" "root" {
  chart = "argocd-apps" …
}
```

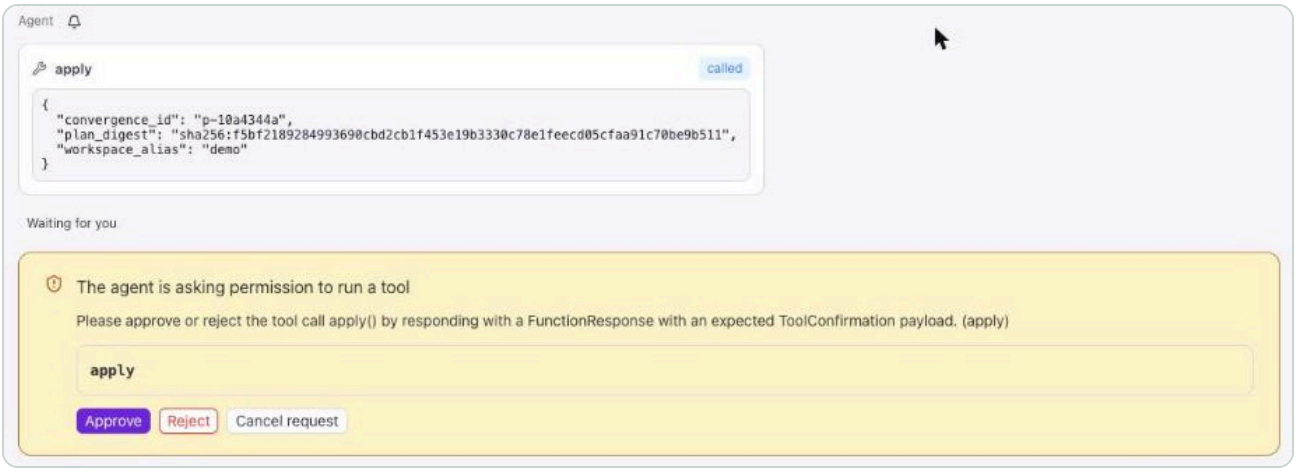
```
# module B · Turf direct
terraform {
  required_providers {
    azurerm    = { source = "hashicorp/azurerm" }
    helm       = { source = "hashicorp/helm" }
    kubernetes = { source = "hashicorp/kubernetes" }
  }
}
module "cluster" {
  source = "Azure/aks/azurerm"
}
# the addons
resource "helm_release" "addons" {
  for_each = var.addons …
}
# the workloads
resource "kubernetes_manifest" "workloads" {
  manifest = …
}
```

A CHANGE, START TO FINISH · A REAL RUN

Ask, read the plan, approve

The run printed here is small: two AWS Lambda functions, each with a public URL. It starts with one sentence in the kagent chat.

```
THE CHAT, UP TO THE APPROVAL REQUEST · ONE KAGENT SESSION · 2026-11-16 UTC · TRIMMED
18:26:15   you    “Deploy a Lambda fleet as a stack called demo.”
18:26:25   agent  “Workspace demo is open and its state is empty. I'm starting a convergence with the stack's variables.”
18:26:25 ✓ up({ var: { fleet_size: 2, greeting: "Hello from the cluster.", name_prefix: "turf-kagent-demo" }, … })
18:27:01 ✓ plan   → awaiting_approval · add 9, change 0, destroy 0
18:27:02   agent  “Planned: 9 to add. I'm showing the plan, and the approval request comes next.”
18:27:03 ✓ show   → the plan, as JSON, in the chat
18:27:05 ◷ apply({ convergence_id: "p-10a4344a", plan_digest: "sha256:f5bf…", … })   kagent asks · approved 18:27:58
```



The kagent chat, 18:27 UTC: the agent's `apply` call, with the plan's digest, and the approval request.

Figure 3. From the ask to the approval request. Fifty seconds, and a plan of nine resources to create waits for a person.

- **You approve that exact plan.** The agent's `apply` carries the plan's digest, and kagent asks because that tool is bound with `requireApproval: true`.
- **The plan is yours to read first.** `show` puts it in the chat, above the call, and the `plan` result links to it in Turf's console.
- **A rejection applies nothing.** In another session that day a teardown was rejected, and the agent answered: “You rejected the destroy, saying the team is still using the fleet, so nothing was applied.”

HOW YOU KNOW IT WORKED

The agent checks what it built

A turf action is a step of the apply that the agent carries out. This one asks it to fetch each function's URL and compare what it answers.




```
THE RUN, FROM THE APPROVAL · THE SAME SESSION · 2026-11-16 UTC · TRIMMED
18:27:05 ◷ apply({ convergence_id: "p-10a4344a", plan_digest: "sha256:f5bf…", … })   asked · approved 18:27:58
18:28:42 ✓ apply  → awaiting_actions · action.turf_action.verify
18:28:44 ✓ bash   wget -qO- -S https://d3jf7lxepmc4mjpthsksan232u0bzcoj.lambda-url.us-west-2.on.aws/
           → HTTP/1.1 200 OK · {"message":"Hello from the cluster.","function":"turf-kagent-demo-1"}
18:28:44 ✓ bash   wget -qO- -S https://lwwdk5tp4iecy7ba5omuz65xum0ogbjd.lambda-url.us-west-2.on.aws/
           → HTTP/1.1 200 OK · {"message":"Hello from the cluster.","function":"turf-kagent-demo-0"}
18:28:48 ◷ apply({ action_results: { …verify…: { progress: […], succeeded: true } }, … })   asked · approved 18:29:09
18:29:11 ✓ apply  → converged · applied 9, invoked 1, deferred 0
```

The `demo` Lambda fleet is deployed and converged: 2 functions, each with a public URL, and both URLs answered correctly.

Plan: 9 to add, 0 to change, 0 to destroy.

- add `aws_iam_role.lambda_exec`: the shared execution role `turf-kagent-demo-exec`
- add `aws_iam_role_policy_attachment.lambda_policy`: attaches `AWSLambdaBasicExecutionRole`
- add `aws_lambda_function.hello[0]` and `[1]`: functions `turf-kagent-demo-0` and `turf-kagent-demo-1`
- add `aws_lambda_function_url.public[0]` and `[1]`: a public URL for each function
- add `aws_lambda_permission.public[0]` and `[1]`: allow public invocation of each URL
- add `terraform_data.rollout`: the release marker that triggers the `verify` action

Verification: I fetched each URL once, and both returned HTTP 200 with the message "Hello from the cluster."

Function	URL
`turf-kagent-demo-0`	https://lwwdk5tp4iecy7ba5omuz65xum0ogbjd.lambda-url.us-west-2.on.aws/
`turf-kagent-demo-1`	https://d3jf7lxepmc4mjpthsksan232u0bzcoj.lambda-url.us-west-2.on.aws/

The kagent chat: the agent's report at 18:29 UTC, in part

Figure 4. From the approval to the end. Just under three minutes from the ask to the end, 75 seconds of them waiting for a person: kagent asks twice, because the agent returns the check's result in a second `apply` call. The sandbox reaches the function URLs through one egress policy; it has no route to the AWS API, and no credentials for it.

The check is 32 lines of configuration: `terraform/aws/lambda-fleet/verify.turf.hcl` in `turf-examples`.

SEVERAL SESSIONS AT ONCE

The engine scales from zero, and back

Turf's engine runs as a Knative service. Pods start when a plan or an apply needs them, and go when it is done. restate-server keeps the record of every plan and apply, so nothing is lost with a pod.

- **Three sessions, three stacks.** Three sessions ask for a stack each, a minute apart. One agent definition serves all three, each in its own sandbox, with its own state.
- **Zero pods when idle.** A second pod joins while the work overlaps. About a minute after the last run ends, none are left.

```
kind: RestateDeployment              # turf-engine, run by the Restate operator; deploymentMode: knative
spec: {knative: {tag: v1-4, minScale: 0, maxScale: 2, target: 20}}    # target: concurrent requests per pod
```

manifests/turf/engine.yaml, trimmed.

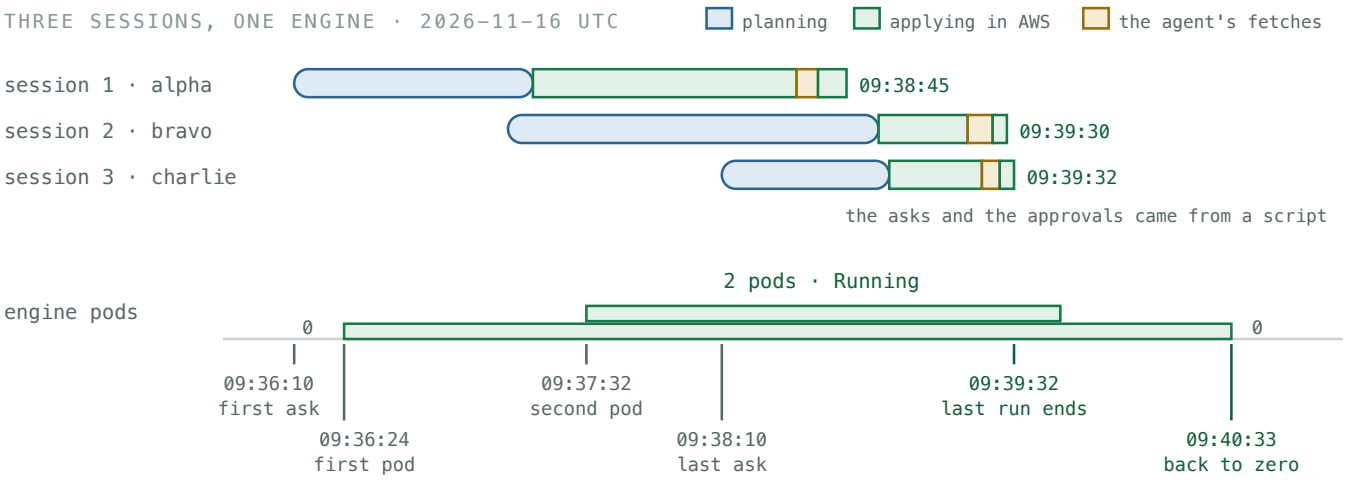


Figure 5. Three sessions, one engine. Three stacks, asked for a minute apart by a script that also answered the approvals, all finished 3 min 22 s after the first ask.

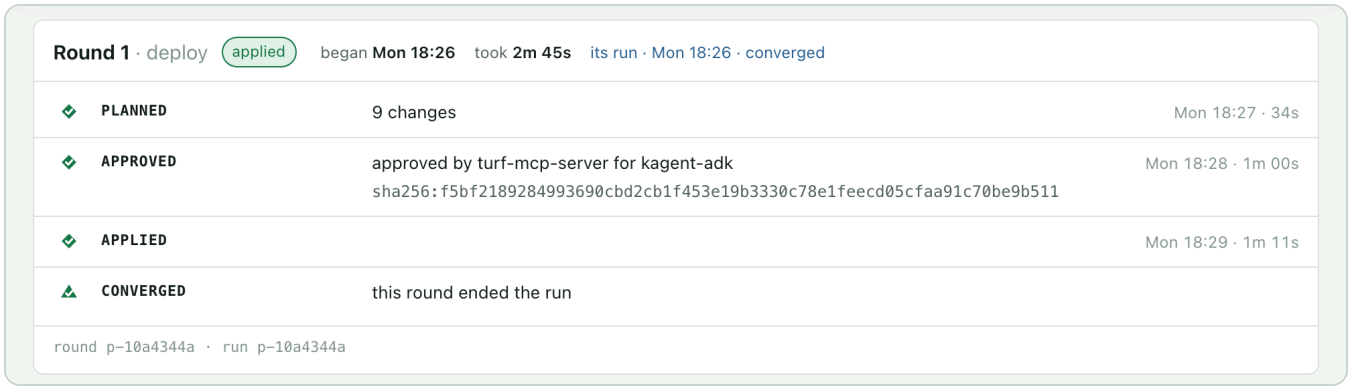


Figure 6. The record of a run. Turf's console, read two days after the run on pages 4 and 5: nine changes planned; the approval a person gave in kagent, recorded under the MCP client with the digest that `apply` named on page 4; applied and converged. Its clock starts at the `up` call, ten seconds after the ask.

Run it

The example in this paper is public. You need a kind cluster with Kubernetes 1.37, and an AWS account.

```
git clone https://github.com/turfbuild/turf-kagent
```

1. Install Turf (below) and the kagent CLI (1.0.0), and set `ANTHROPIC_API_KEY`.
2. Run `turf -C setup/aws up` once, with your own AWS credentials. It makes the engine's key: an IAM user scoped by name to `turf-kagent-*`.
3. Run `./setup/up.sh`: Knative, the Restate operator, kagent, the Secrets, then Turf and the agent.
4. Open the kagent UI, pick the agent `turf`, and ask it for a fleet as a stack called demo. Approve the plan, and the request that follows the agent's check.
5. Destroy the stacks before you delete the cluster: the functions are outside it.

GET TURF

Download it. Run it.

Turf is available now, as a download. No form, no login. Free within the Community limits.

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

Government and larger organizations need a commercial license, and any organization can evaluate Turf in non-production for 30 days: 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 a self-hosted, autonomous infrastructure-as-code tool, built on OpenTofu®. Every change, from a person or an agent, is a plan that is checked, approved and recorded. Turf runs on a laptop, in a cluster, or on a CI runner, and any agent drives it through its MCP server.

Read next: *Metal to Tokens*, the workload cluster from the hardware up: turf.build/reference-architectures

turf.build

Turf™ is a trademark of Turf LLC. Terraform® is a trademark of HashiCorp. Kubernetes® and Helm® are registered trademarks, and Knative and Argo trademarks, of the Linux Foundation. OpenTofu® is a registered trademark of LF Projects, LLC. Amazon Web Services, AWS, AWS Lambda and Amazon EKS are trademarks of Amazon.com, Inc. or its affiliates. GKE is a trademark of Google LLC, and Azure, as in Azure Kubernetes Service (AKS), of the Microsoft group of companies. kagent is a Cloud Native Computing Foundation project. Turf is not affiliated with, endorsed by, or sponsored by HashiCorp, IBM, the Linux Foundation, LF Projects, the OpenTofu project, Amazon, Google, Microsoft, Solo.io or Restate; every mark is used only to describe interoperability. All other marks belong to their owners. Versions: kagent 1.0.0, Turf 1.4.0, Knative Serving 1.24.0, restate-operator 3.2.0, restate-server 1.8.1, Kubernetes 1.37.0.

Every still and transcript in this paper is from runs on 2026-11-16 UTC, on a kind cluster against AWS us-west-2. Agentic IaC for Platform Teams, v0.10, 2026-11-27.