HPE Morpheus Software & Terraform

Agenda

HPE Morpheus Software

Virtualization cost increases shattered the status quo

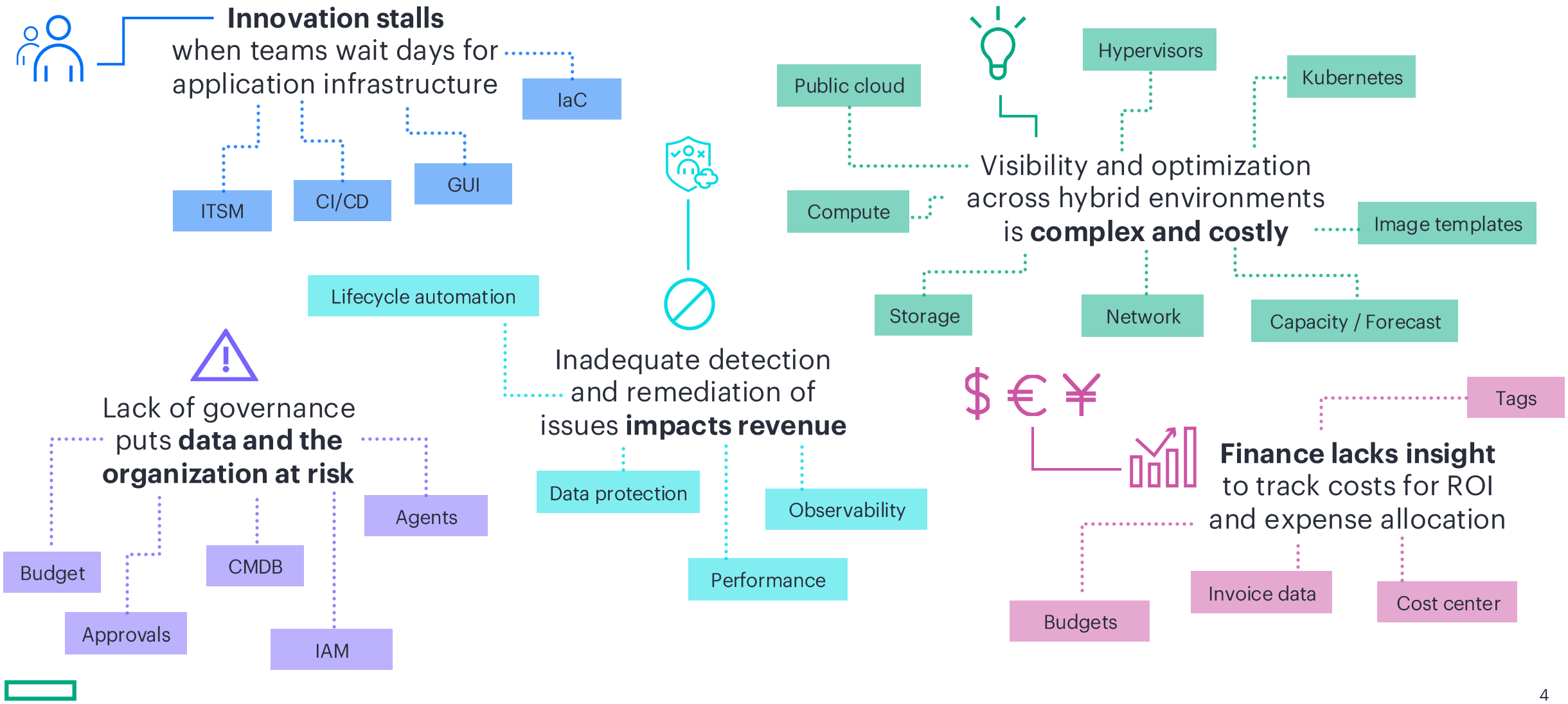
From VM tax and lock-in to hybrid-by-design flexibility and control

	Tomorrow's possibility
Rising license and support costs and forced product bundles	Predictable economics and flexible consumption 
New workloads driving increasing heterogeneity	Freedom to deploy into any mix of hardware, runtime, and cloud 
Tool sprawl and siloed operations	A cloud operating model with end-to-end workload orchestration 
Limited readiness for data-intensive AI	A secure foundation of IT for AI plus built-in intelligence to apply AI for IT 

Hybrid by design

Getting to a hybrid cloud operating model can be challenging

Complexity has scaled disproportionally to IT staff, skills, and resources



The opportunity

Confident hybrid cloud operations across any environment

Manage every workload

through unified hybrid cloud ops

- Any workload
- Any runtime
- Any cloud, private or public
- Any location

90%

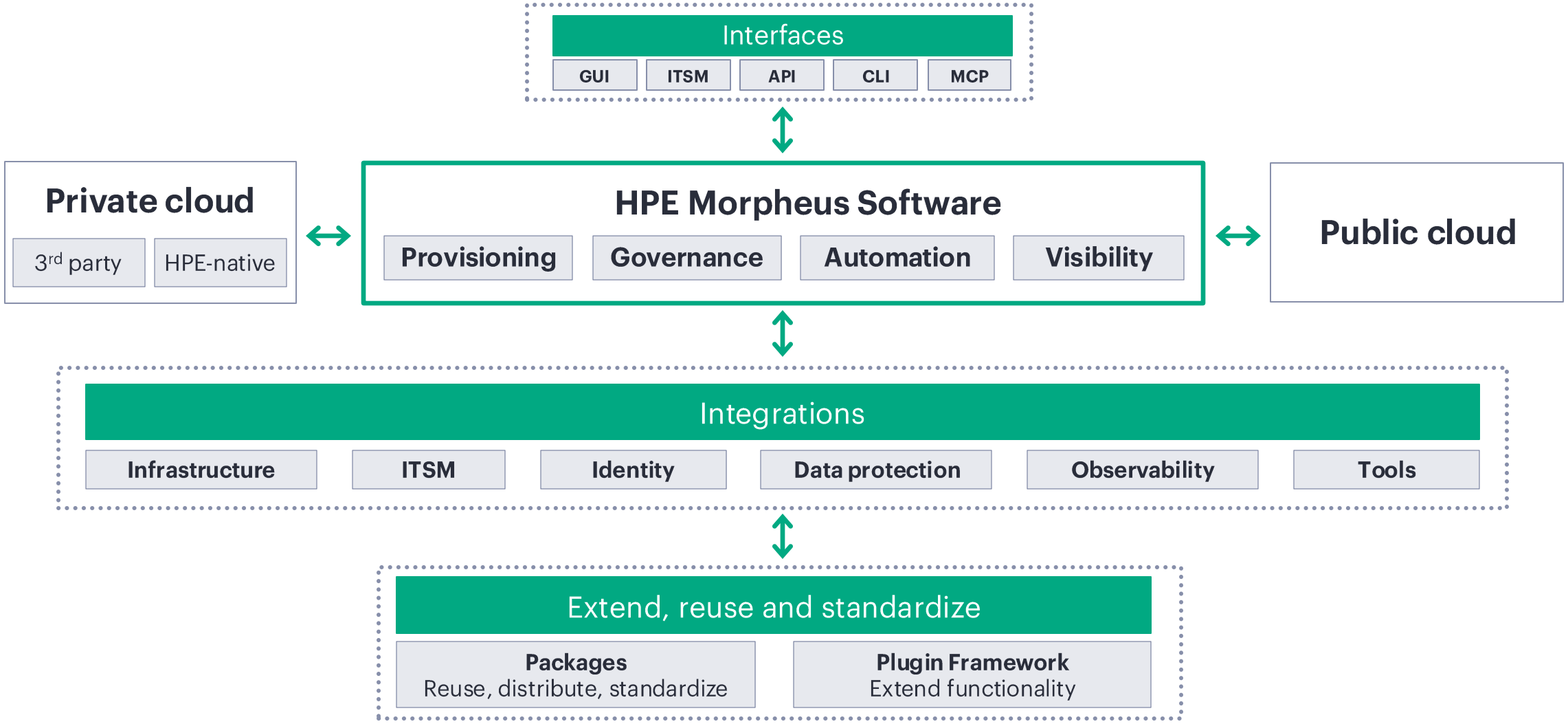
of enterprises will accelerate hybrid cloud through 2027

HPE Morpheus Software

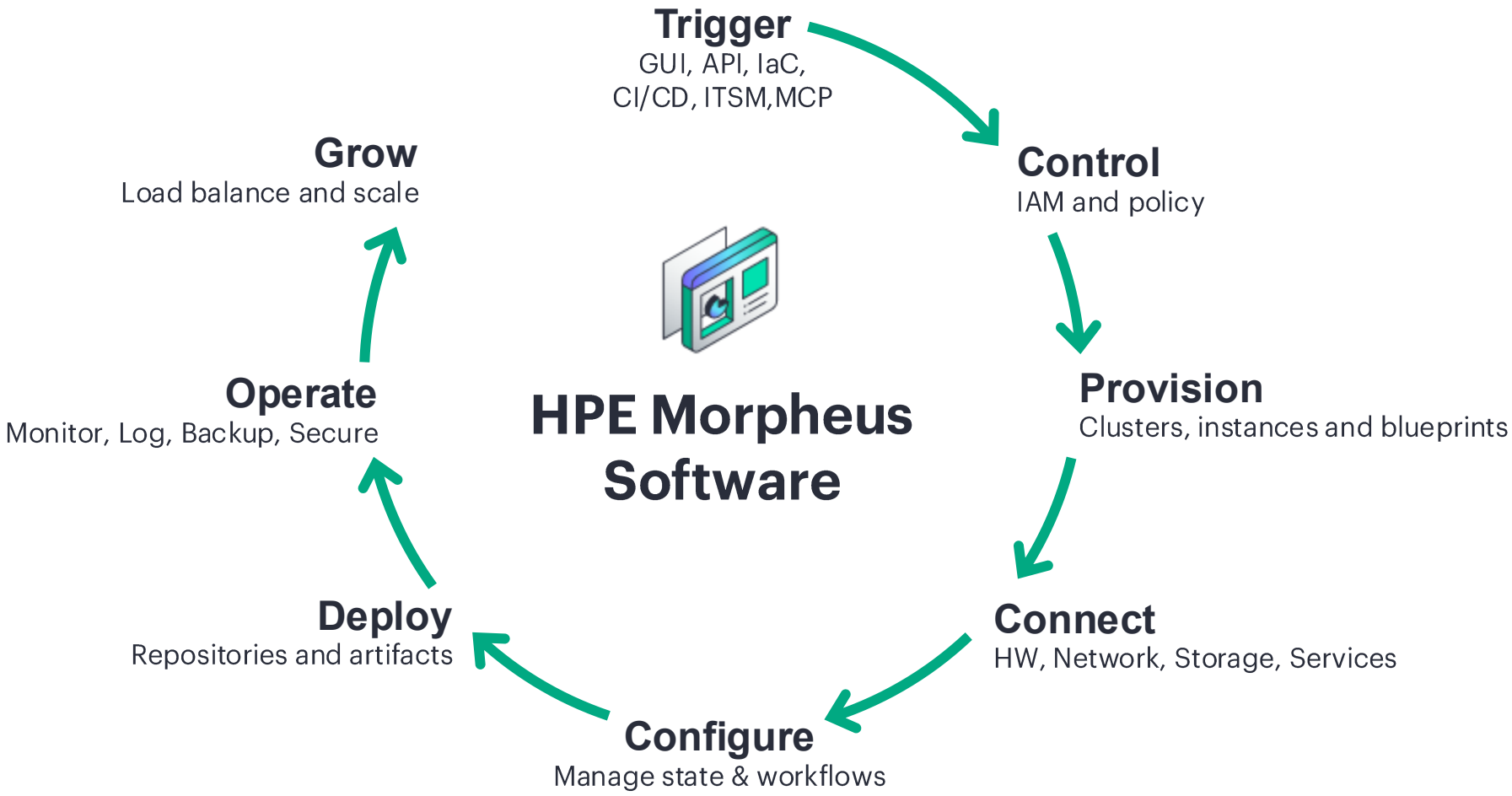
Architecture

HPE Morpheus Software logical architecture

Built to deliver and manage application workloads

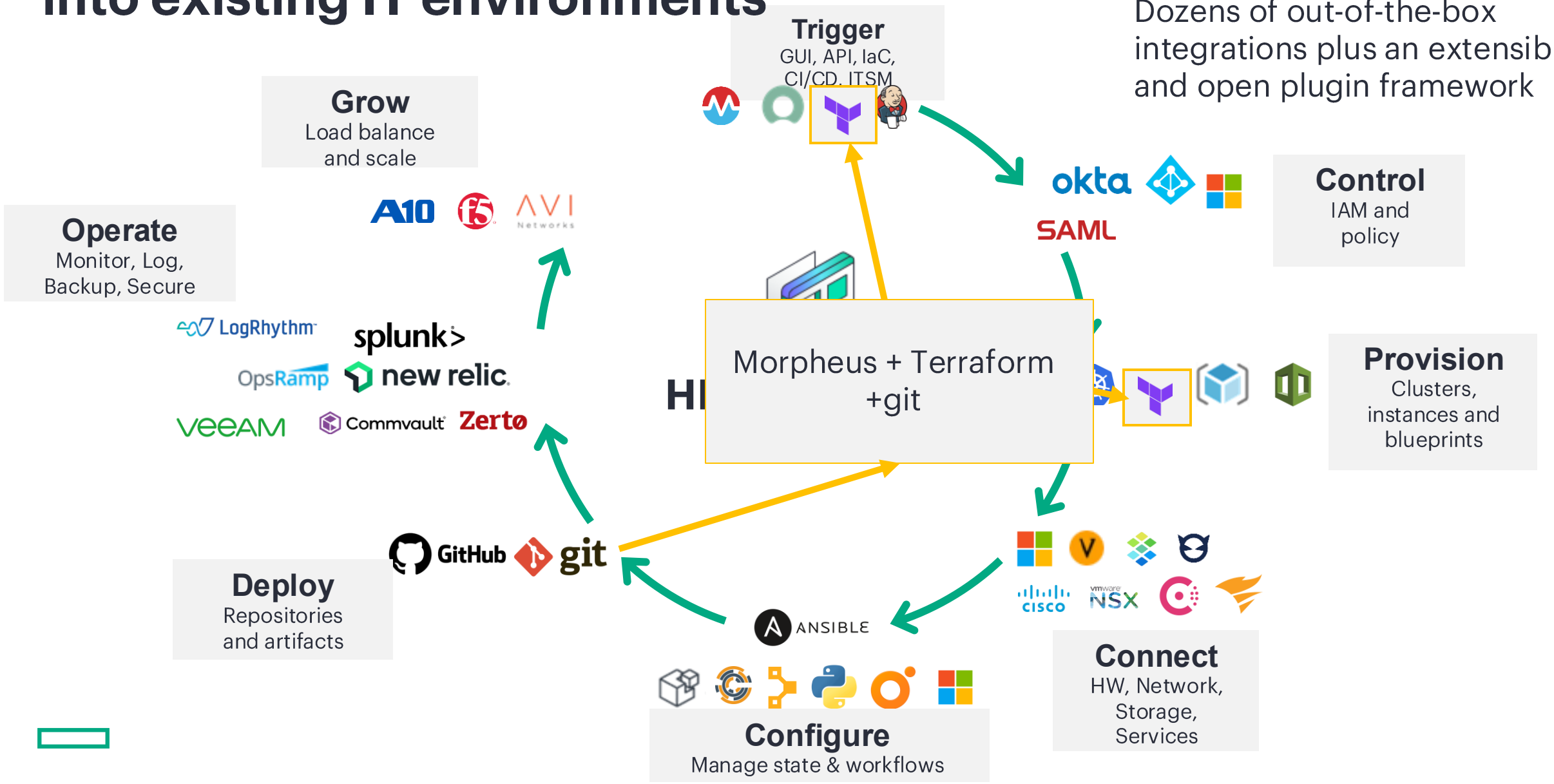


HPE Morpheus Software abstracts underlying tools and processes from a workload-centric point of view



With hooks to rapidly integrate into existing IT environments

Dozens of out-of-the-box integrations plus an extensible and open plugin framework

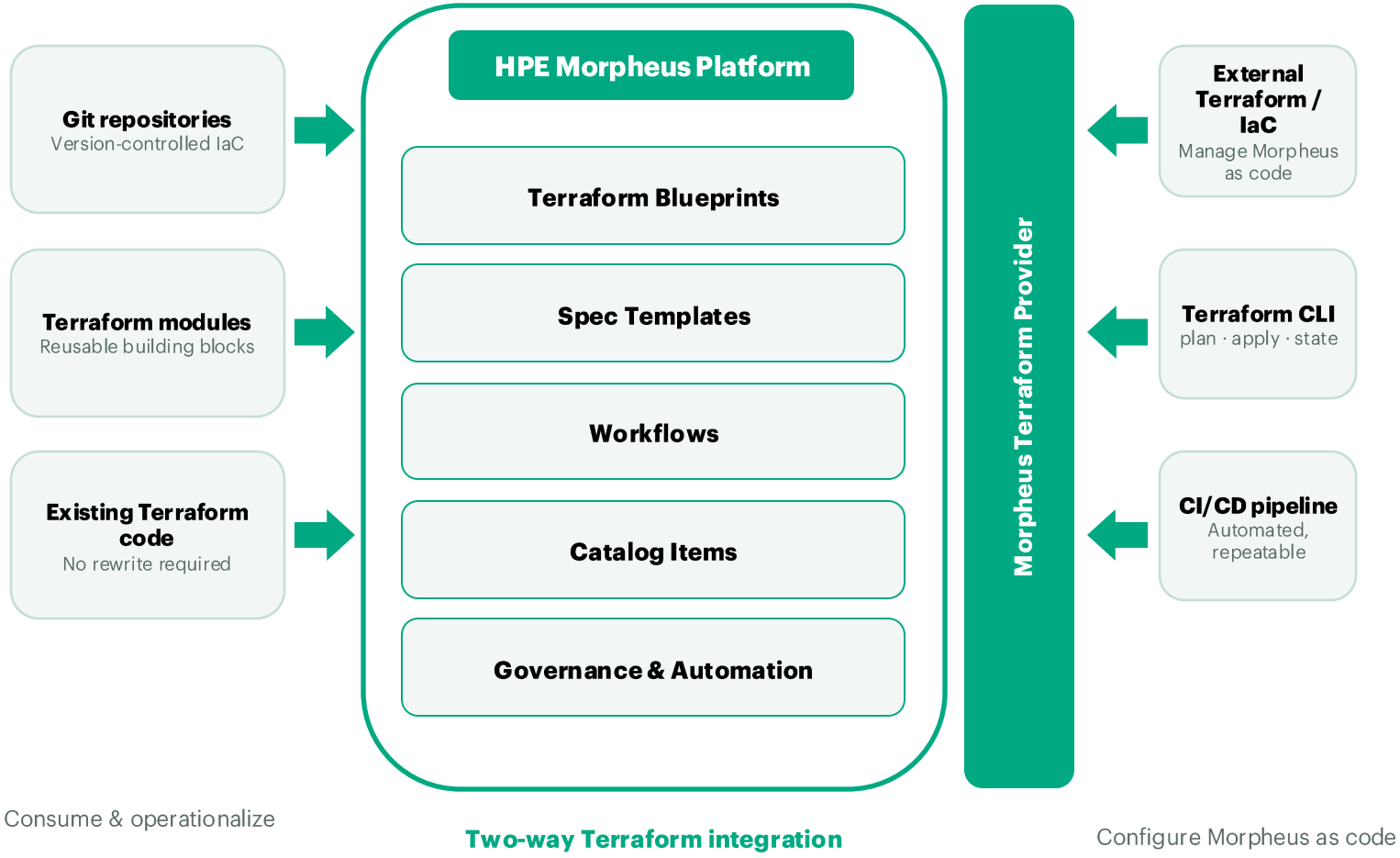


Morpheus + Terraform: Better Together

Reuse and govern existing Terraform assets—and manage Morpheus as code

Amplify your Terraform investment

- Reuse existing Terraform code, modules, and Git repos—no rewriting.
- Turn Terraform assets into governed self-service blueprints and catalog items.
- Apply RBAC, approvals, policies, cost controls, secrets, and lifecycle automation.
- Visualize deployments, monitor drift, and manage multi-cloud infrastructure in one platform.
- Use the Morpheus Terraform Provider to manage Morpheus configuration as code.

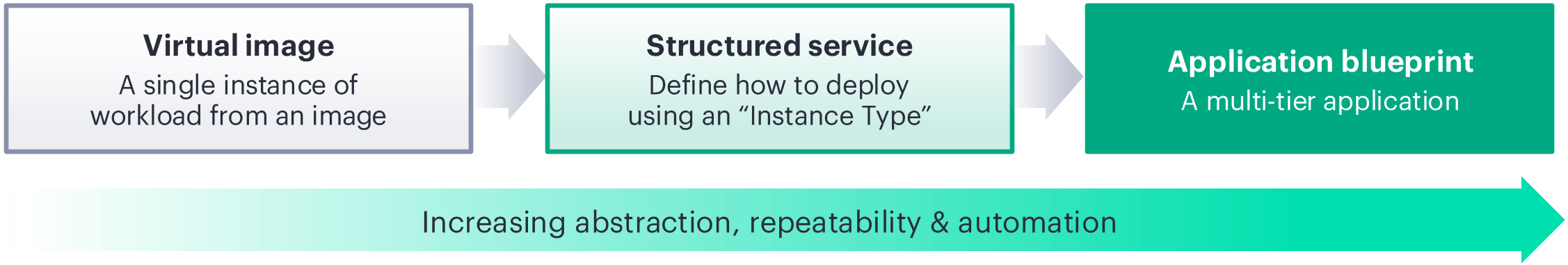


HPE Morpheus Software

Provisioning

Provisioning workloads: From simple to structured

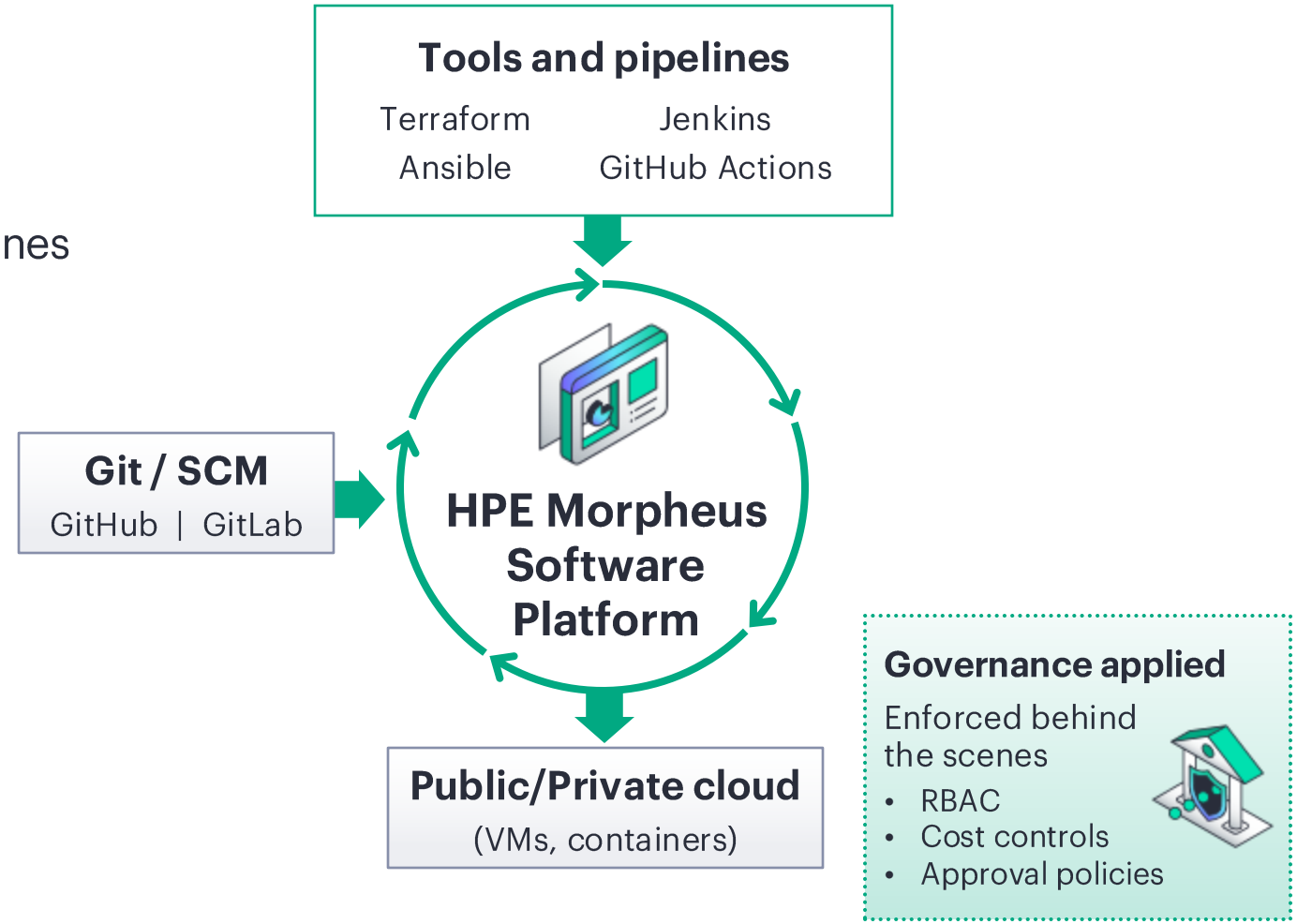
- Provisioning = How workloads are deployed onto infrastructure (public / private clouds)
- Multiple approaches exist, from simple VM or container workload creation to full application stacks



Trigger provisioning using integrated tools for automation

Use existing tools while adding governance, security, and control

- Use existing tools without change
- Centralize governance and access control
- Secure variables and state management
- Automate provisioning through APIs and pipelines
- Sync infrastructure code from Git repositories

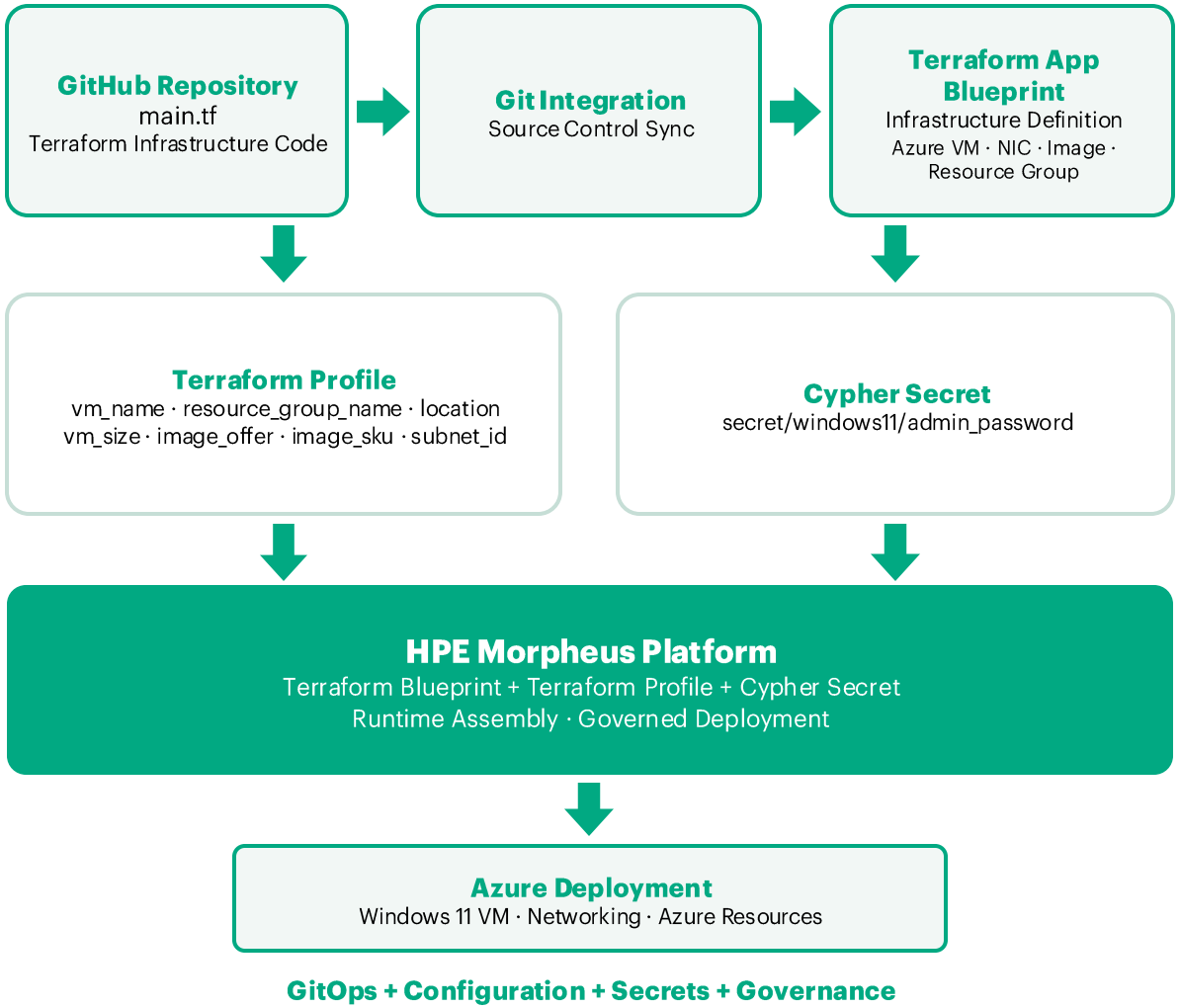


Morpheus + Terraform: Infrastructure, Configuration & Secrets

GitOps-driven Terraform with source control, configuration, and secrets

GitOps-Driven Terraform with Morpheus

- Terraform code is maintained in GitHub
- Morpheus consumes infrastructure directly from source control
- Terraform Profiles provide environment-specific configuration
- Cypher securely stores sensitive credentials
- Morpheus assembles code, configuration, and secrets at deployment time
- Enables governance, reuse, and self-service Infrastructure as Code



DEMO

Morpheus + Terraform + git Source Code Management (SCM)

IAC at HPE

HPE Morpheus Terraform Provider

IaC at HPE

Customer demand

Customers increasingly want to provision infrastructure as code rather than through a UI.

This applies to almost all of our products.

Products in scope

Morpheus – distinct from Morpheus Blueprints (which can be driven from the provider)

OpsRamp – configuration.

Morpheus Central

Two toolsets, one team

Terraform/OpenTofu and Ansible are the two major IaC toolsets.

Zod**IaC** owns Hybrid Cloud IaC support and development: the "unified" **hpe** Terraform/OpenTofu provider, migration to it, and Ansible as an alternative to Terraform (Morpheus first).

One unified provider

hpe_**morpheus**_* for Morpheus, hpe_**opsramp**_* for OpsRamp — one provider, a namespace per product.

Supported and shipping

HPE support channels and JIRA. Monthly releases to the HashiCorp and OpenTofu registries.

Where we are

V2.0.0 today. **morpheus** parity since v1.0.0; **hpegl** parity at v2.0.0

hpe v2.0.0 compared with hpegl and morpheus

- One provider now covers what took two — hpe v2.0.0 reaches hpegl parity, and it already replaced the standalone morpheus provider.
- hpe generalises where morpheus specialised: morpheus's 125 resources need only 97 in hpe, so the like-for-like comparison is 97 against 184.
- PCE systems authenticate through GreenLake IAM — no separate Morpheus credential is needed.

hpe

v2.0.0 · the unified provider

184 resources
153 Morpheus · 31 OpsRamp

135 data sources
128 Morpheus · 7 OpsRamp

hpegl

being replaced by hpe v2.0.0

24 resources
14 VMaaS · 7 Metal · 3 CaaS

36 data sources
29 VMaaS · 2 Metal · 5 CaaS

morpheus

already replaced, since v1.0.0

125 resources
→ 97 in hpe, generalised

65 data sources
→ 59 in hpe

WHAT v2.0.0 CHANGES

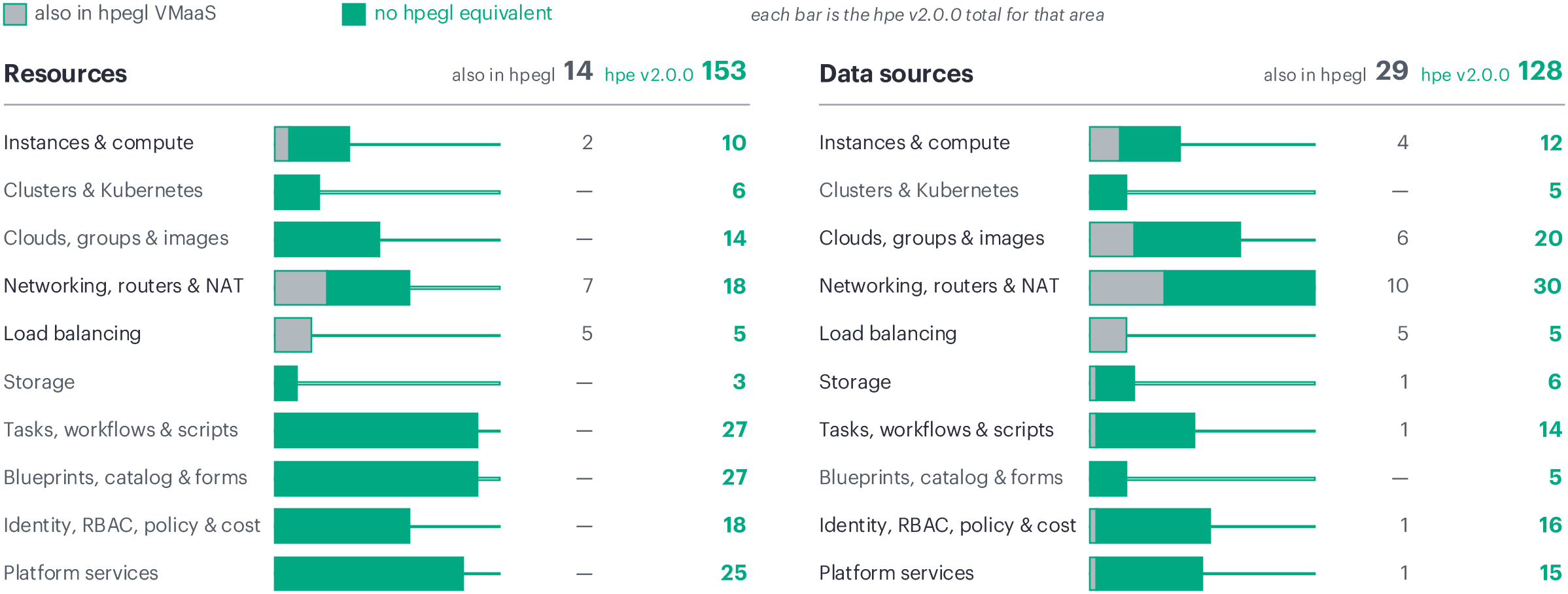
hpegl parity means the hpe provider can take over VMaaS workloads outright, and tfmigrator converts the existing configuration and state. PCE systems authenticate with a GreenLake API client through pce_identity or pce_disconnected_identity.

RELEASES · every month

12 releases so far, v0.1.0 through v2.0.0, and a new version at the end of every calendar month. Published to both the HashiCorp and OpenTofu registries, and supported through the normal HPE channels.

What the providers can manage — hpegl vs hpe

- hpegl VMaaS covers three of these ten areas. hpe v2.0.0 covers all ten — 14 resources become 153, and 29 data sources become 128.
- hpegl manages VMware with NSX-T networking only. hpe v2.0.0 spans VMware, AWS, Azure and HVM/KVM, so even where the counts match the reach is wider.



What the providers can manage — morpheus vs hpe

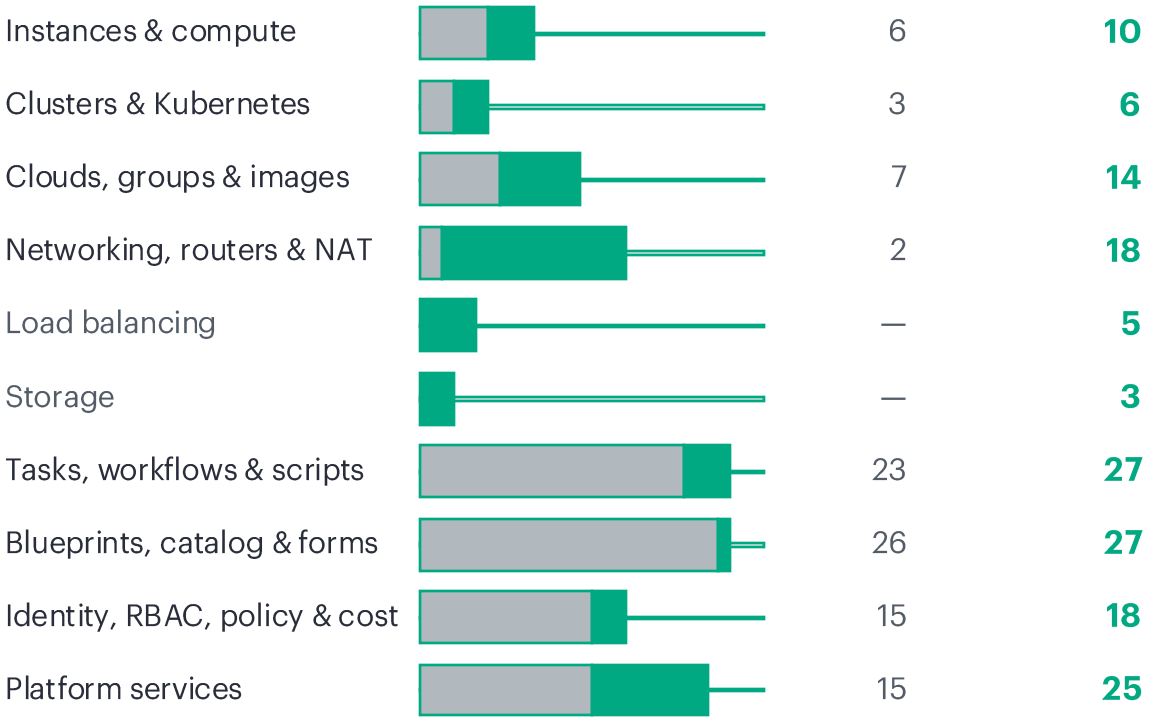
- The morpheus provider reaches all ten areas. 97 of hpe's 153 resources have a morpheus counterpart, and 59 of its 128 data sources.
- hpe generalises where morpheus specialised — 23 policy resources become one hpe_morpheus_policy, four clouds become one, three instances become one — so all 125 morpheus resources land on just 97 hpe ones.

also in the morpheus provider no morpheus equivalent

load balancing, storage, routers & NAT and affinity groups have no morpheus equivalent at all

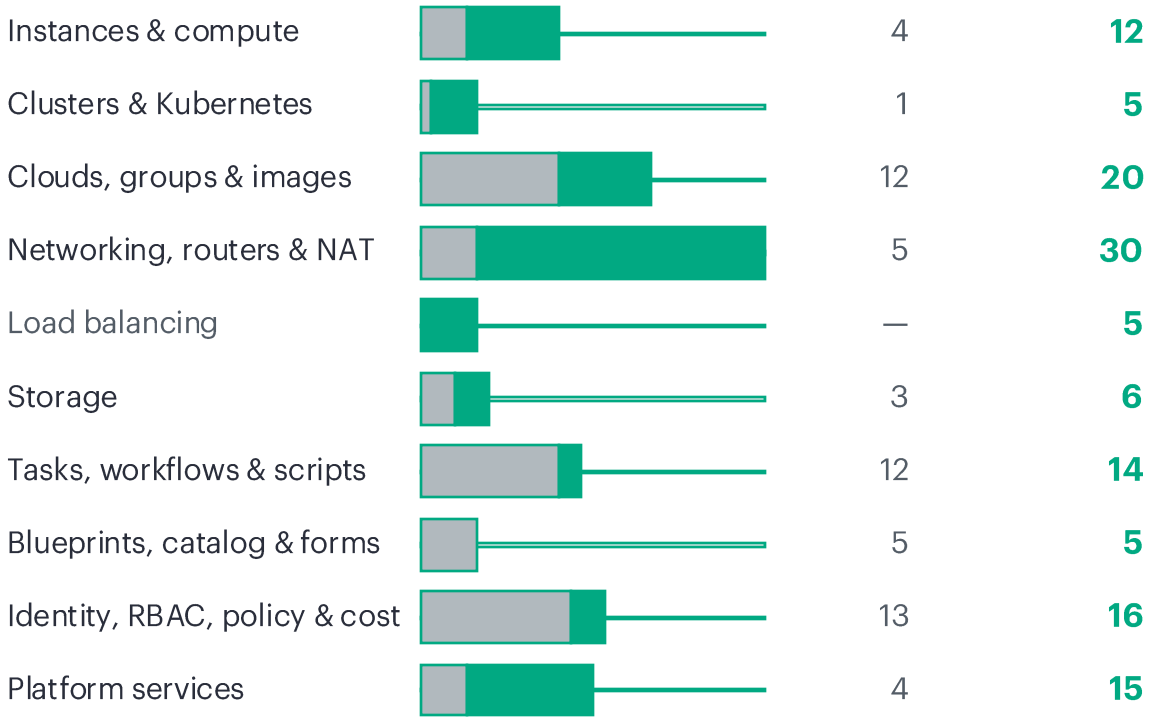
Resources

also in morpheus **97** hpe v2.0.0 **153**



Data sources

also in morpheus **59** hpe v2.0.0 **128**



hpe provider roadmap: 2.0.0 and beyond

ROADMAP

- v2.0.0 is the first release to carry two services: the Morpheus surface grows again, and OpsRamp arrives as a second sub-provider behind the same hpe provider block.
- It is also the release that opens the hpegl path — tfmigrator now migrates hpegl as well as morpheus configurations, so both routes land on the same provider.

resource data source *"+ lookups" — resource with data sources alongside it*

NEW IN v2.0.0

the Morpheus surface

hpe_morpheus_cloud_affinity_group + lookups
affinity and anti-affinity, per-member resources

hpe_morpheus_instance_node
scale an instance out or in, a node at a time

wait_for_ip_address
opt-in wait until the guest reports an address

hpe_morpheus_compute_server(s)
lookups over /api/servers

NEW IN v2.0.0

OpsRamp joins the provider

hpe_opsramp_*
31 resources and 7 data sources, alongside Morpheus

alerting
metric, log and event alerts; correlation policies

integrations
integration apps, configs, events and custom ones

service desk
categories, urgencies, impact and KB articles

estate
clients, sites, device groups and profiles

FUTURE

on the roadmap beyond 2.0.0

other SDN platforms
Juniper SDN, Apstra and Aruba CX beyond NSX-T

multi-tenancy
multi-level tenant-scoped resources and lookups

hpe_morpheus_host + lookups
the bare-metal host itself, and lookups for it

config_bmaas on hpe_morpheus_cloud
create and configure a BMaaS cloud

placement & group edits
parent-host pinning, datastore clusters, in-place edits

MIGRATION · from morpheus or hpegl

tfmigrator converts an existing morpheus or hpegl configuration and its state into hpe form, and the three hpegl provider blocks collapse into one. Both paths are handled by the same binary, shipped with the provider release.

THE 2.0.0 SURFACE · two services, one provider

153 Morpheus resources and 128 data sources, plus 31 OpsRamp resources and 7 data sources — 184 resources and 135 data sources behind a single hpe provider block.

Demo – HVM Affinity Groups

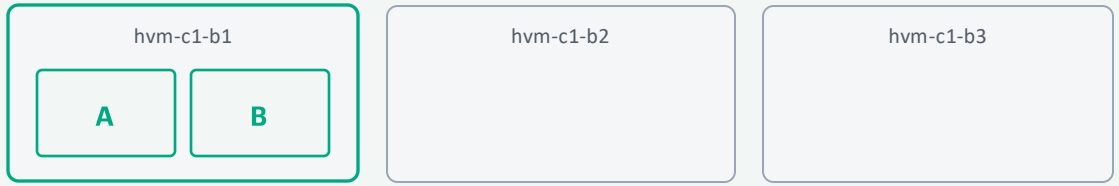
Affinity groups at a glance

- A group is a placement rule. Membership is a resource, so guests can join after the fact or as they are provisioned — and the rule is enforced by the cluster, not by Terraform.

PHASE 1 KEEP_TOGETHER

1. **create the group**
   affinity_type = KEEP_TOGETHER
2. **add guest A**
   membership resource, after it exists
3. **provision guest B into it**
   config_hvm.affinity_group_id

RESULT · both land on one host

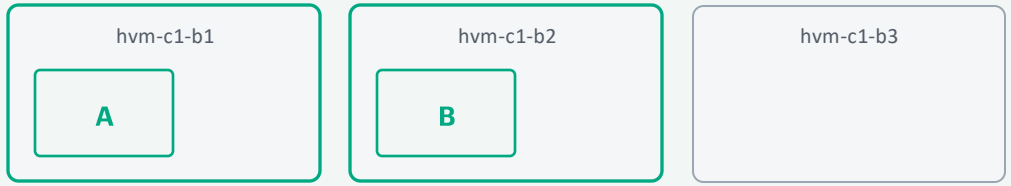


applied as each guest is built — Morpheus places B against A

PHASE 2 KEEP_SEPARATE

1. **deactivate the first group**
   active = false — not deleted
2. **create a KEEP_SEPARATE group**
   affinity_type forces replacement
3. **add both servers to it**
   one membership resource each

RESULT · the cluster moves one, minutes later



not applied at build — the placement loop notices and migrates a guest

Membership is recorded either way. **Only the hypervisor tells you whether anything was actually placed** — the demo reads parent_host_name back from the API.

Terraform provider migration

Migration paths

morpheus → hpe (v1.2.0, end of March 2026).

hpegl → hpe (v2.0.0, end of August 2026).

Why customers need help

Resource, data-source and schema naming changes mean existing HCL has to be modified.

Customers, PM and internal users have asked for help adapting; some customers have thousands of lines of config.

Manual changes are tedious and error prone.

tfmigrator

Generates HCL and the required import blocks from existing config and state, leaving the config and statefile in place.

Handles variables and HCL modules, and is highly configurable per resource pair using YAML.

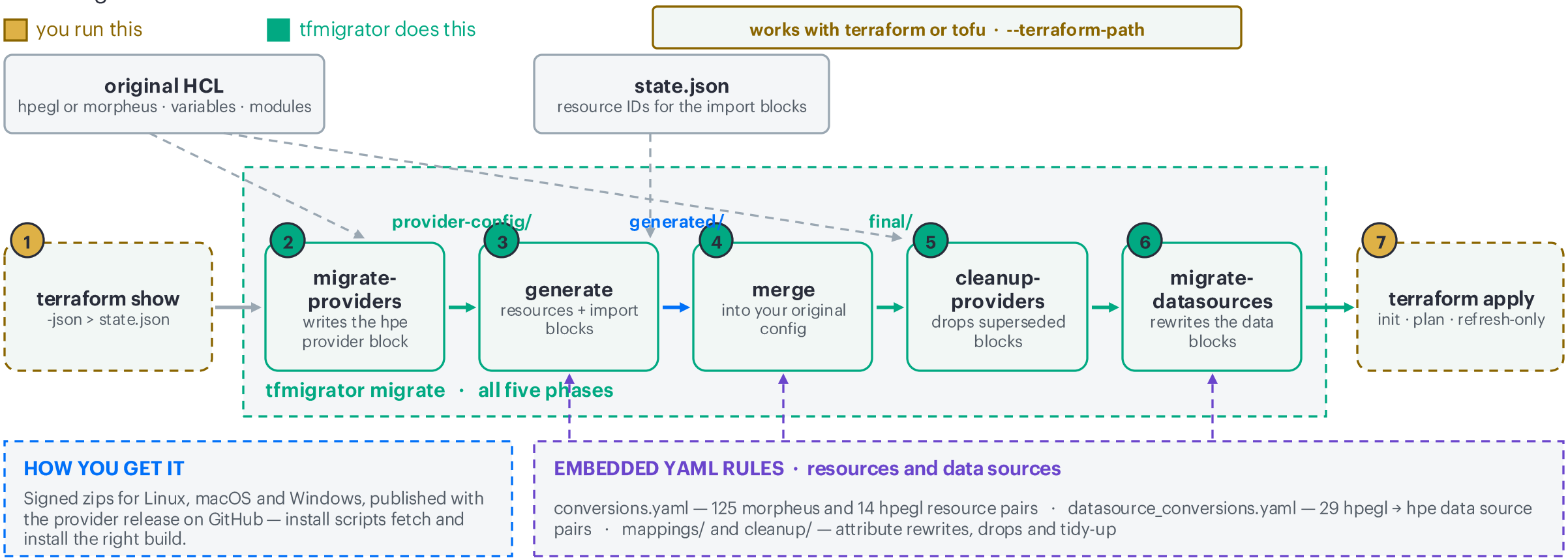
It does not automate the entire process – the remaining steps are documented.

THIS IS NOT A HYPERVISOR MIGRATION. No workload is moved, copied or re-provisioned. tfmigrator transitions control of your existing Morpheus resources to the hpe provider — the infrastructure itself is untouched.

tfmigrator: migrating morpheus and hpegl configs

HPEGL IN 2.0.0

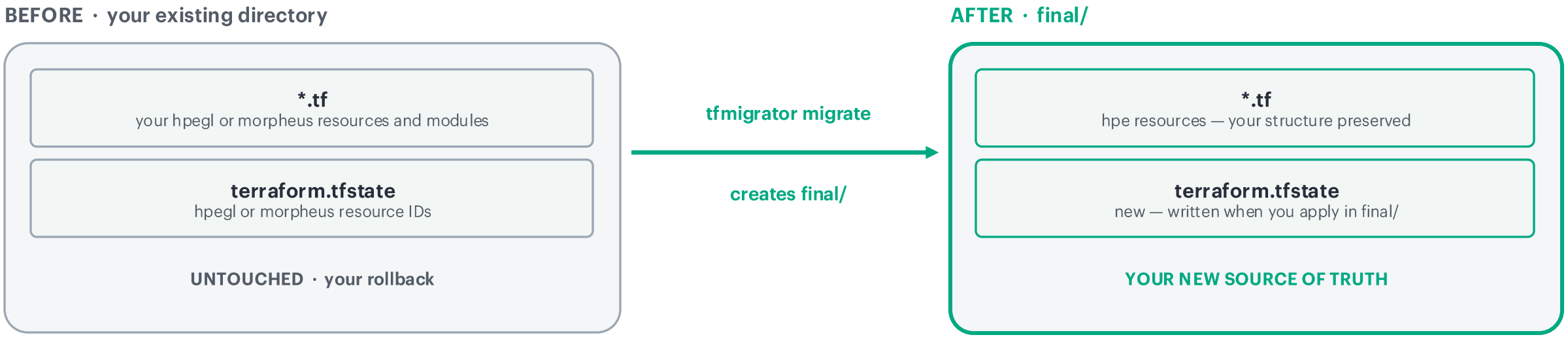
- Three commands: terraform show -json to export your state, tfmigrator migrate to do the work, then terraform apply to adopt the result.
- tfmigrator is a separate binary — nothing in Terraform invokes it. It reads your configuration and state as files, and writes a migrated copy alongside them.



Your variables, expressions, comments and module structure are preserved. cleanup-providers runs either side of the data-source rewrite, so the migrated config no longer installs or authenticates the old providers. Every phase except the prune can also be run on its own.

After migration: a new configuration and a new state file

- Migration produces a new pair: an hpe configuration in final/, and a new state file written by the first apply in that directory.
- Your original hpegl or morpheus configuration and state are never modified — they stay intact as a rollback until you are satisfied with the result.



IN final/ — ADOPT THE IMPORTS, THEN THE TEST THAT MIGRATION IS COMPLETE

```
final $ terraform init && terraform apply
final $ terraform plan
No changes. Your infrastructure matches the configuration.
```

either CLI — terraform or tofu

Once that plan is clean the migration is done: adopt final/ as your working configuration and retire the original config and state at your own pace. Nothing is deleted for you.

tfmigrator: three provider blocks become one

IN 2.0.0

- hpegl needed three blocks: a GreenLake provider, a data source to fetch Morpheus credentials, and a Morpheus provider.
- hpe v2.0.0 replaces all three with one, and tfmigrator picks the right identity block from your existing configuration.

BEFORE · hpegl — all three removed

```
provider "hpegl" {
  vmaas { location, space_name }
  iam_version = "glcs"
}
```

```
data "hpegl_vmaas_morpheus_details"
    "location_1" { }
```

```
provider "morpheus" {
  url      = data...url
  username = data...username
}
```

tfmigrator →

AFTER · hpe v2.0.0

```
provider "hpe" {
  morpheus {
    pce_identity {
      location      = "..."
      space         = "..."
      issuer_url    = "..."
      client_id     = var.<alias>_pce_client_id
      client_secret = var.<alias>_pce_client_secret
    }
  }
}
```

credentials become sensitive variables, never literals

WHICH IDENTITY BLOCK? chosen from iam_version on the hpegl provider, or HPEGL_IAM_VERSION

pce_identity

iam_version = "glcs"

location ← vmaas.location
space ← vmaas.space_name
issuer_url ← iam_service_url

pce_disconnected_identity

iam_version = "glp"

location ← vmaas.location
workspace_id ← vmaas.space_name
broker_url ← vmaas.broker_url
token_issuer_url ← iam_service_url

PCE Disconnected — GreenLake on-premises, isolated

no identity block

vmaas.morpheus_url present

url ← vmaas.morpheus_url
access_token ← vmaas.morpheus_token

direct connect — passthrough, no GreenLake IAM

Demo: hpegl + morpheus → hpe migration

Single instance with a module to create a group

The migration at a glance

- The module carried its own provider chain — three provider blocks to reach one Morpheus. After migration there is one, declared once at the root.

BEFORE · hpegl + morpheus

main.tf

provider "hpegl"

data "hpegl_vmaas_cloud" · layout · plan

resource "hpegl_vmaas_instance"

modules/morpheus-group/main.tf

provider "hpegl"

data "hpegl_vmaas_morpheus_details"

provider "morpheus"

resource "morpheus_group"

3 provider blocks

tfmigrator

AFTER · hpe

main.tf

provider "hpe" { morpheus { pce_identity } }

data "hpe_morpheus_cloud" · layout · plan

resource "hpe_morpheus_instance"

the module has no provider of its own, so it uses the default

modules/morpheus-group/main.tf

resource "hpe_morpheus_group"

no provider block — inherits the root's

1 provider block

Everything is adopted by import, including the resource inside the module. **Nothing is rebuilt** — the running instance is untouched.

Thank you!

Resources:

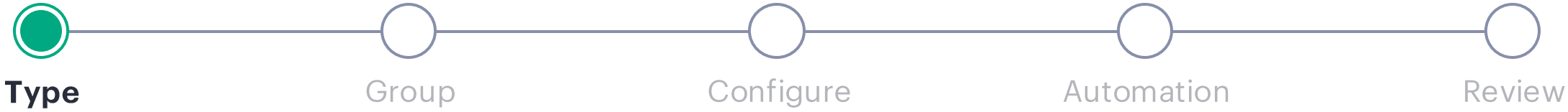
HPE.com – HPE Morpheus Software	https://www.hpe.com/morpheus-software.html
HPE Developer Community Events	https://developer.hpe.com/events/

Backup

More about provisioning

Provisioning workloads: Choosing a Type

What type of **workload** + what **runtime** technology do I want?



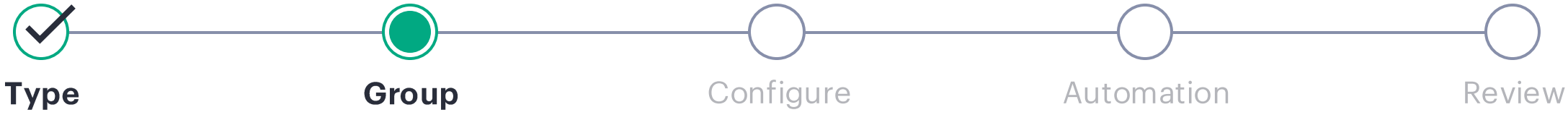
Users choose a **Type** and **Runtime** platform for their workload

- **Hypervisors** (VMs) → VMware, KVM, Hyper-V, Nutanix, etc.
- **Public Cloud** (VMs in cloud) → AWS, Azure, Google, etc.
- **Containers** → Docker, Kubernetes

The Cloud Admin adds infrastructure to HPE Morpheus Software → HPE Morpheus Software exposes it → User can provision to it

Provisioning workloads: Group

Who owns this **workload** and where is it allowed to **run**?



The **Group** unifies deployment and governance decisions

- **Group:** Define ownership and access
- **Cloud:** Select the deployment target
- **Governance, Policy, Cost-tracking:** Tag workload as dev/prod etc.
- **Standardize naming:** Use naming policies to keep consistency

The **Type** abstraction defines capability—**Group** enforces governance and determines exactly where and under what policies the workload will be deployed

Provisioning Workloads: Configure

Translate intent into a concrete, **policy-driven** deployment

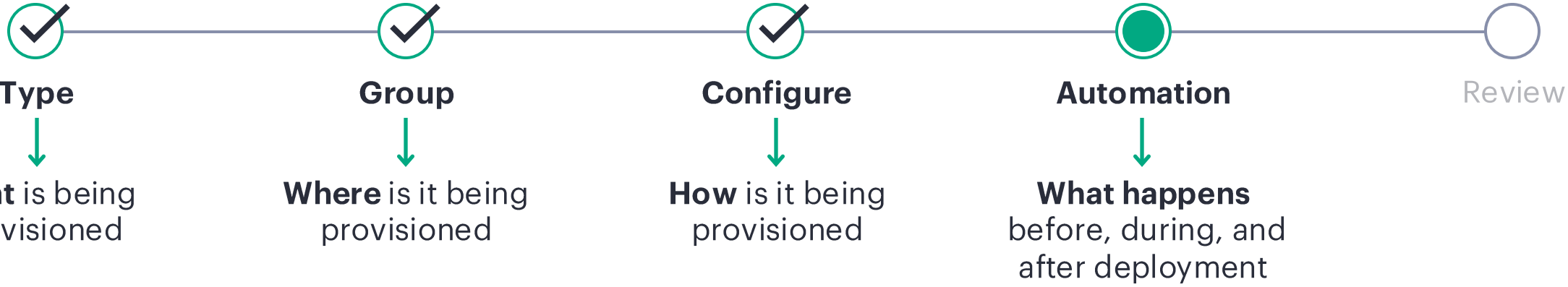
Users define how the workload is **built** and **deployed**

- **Layout** → Structure and platform (KVM, VMware, Kubernetes)
- **Plan** → Sizing and capacity (CPU, memory, cost model)
- **Infrastructure selection** → Resource pools, networks, volumes / datastores
- **Placement** → Host and cluster selection
- **Configuration filtering** → Constrained by Type, Group, and admin policies

The **Configure** step translates user intent into a structured, policy-driven infrastructure definition—abstracting complexity while enforcing standardization, governance, and consistency

Provisioning Workloads: Automate

Automation allows you to standardize and automate workload provisioning



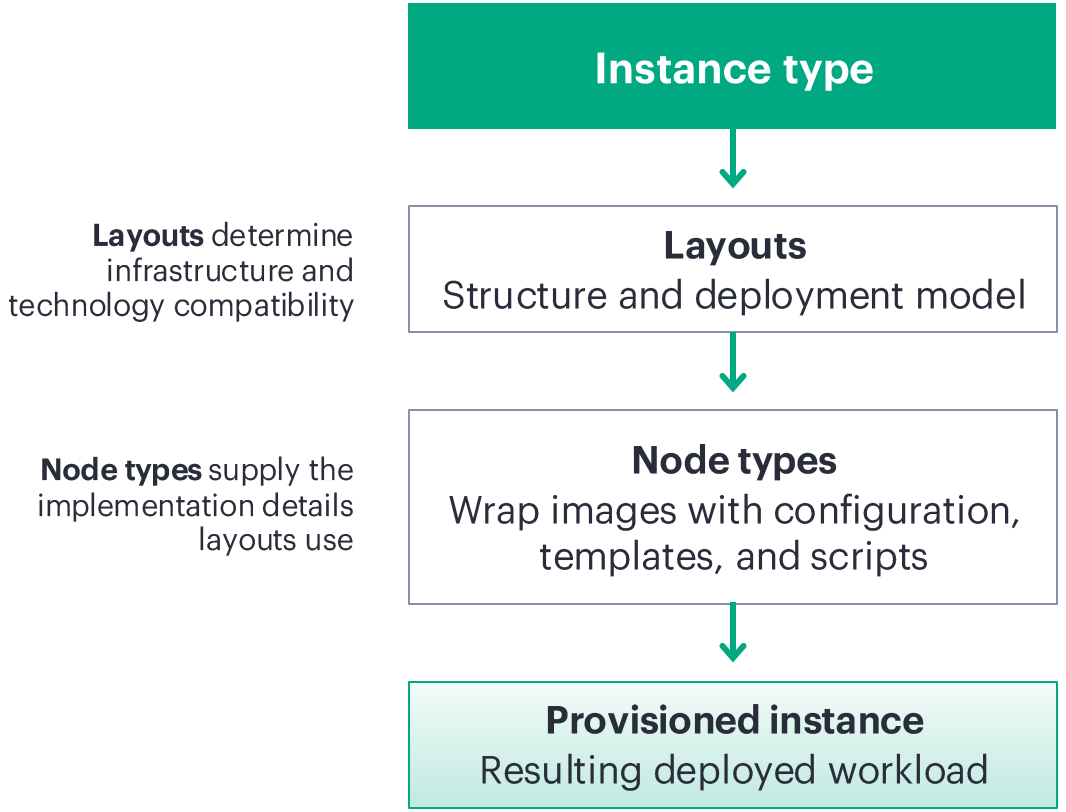
User can turn infrastructure into a repeatable, governed service

- **Workflows:** turning a workload into a fully configured application (provision, post-provision, teardown)
- **Tasks:** Single unit of automation (script, API call, playbook)
- **Optional:** Users can click next through the Automation tab during provisioning

Provisioning and “Instance types”

Reusable provisioning definitions for consistent and governed workload deployment

- **Instance types** define standardized provisioning models
- Organize how workloads are deployed across environments
- Combine **layouts, node types, and configuration** into a single object
- Provide reusable, governed deployment patterns
- Expose consistent options in the provisioning wizard



Virtual image	vs.	Instance type
OS template or disk image; defines **what** is deployed		Full deployment definition; defines **how** it is deployed using layouts and node types

Instance types as services and applications

Reusable service definitions that scale from **single workloads to multi-tier applications**

- Deploy an instance type as a **standalone service**—a single VM, a container, or a multi-node service
- An instance can group multiple VMs or containers into one service
- Combine instance types into complete applications—**Web, Application, and Database tiers**
- **Blueprints** combine instance types into one multi-tier application definition
- Enable consistent, repeatable multi-tier deployments across environments