

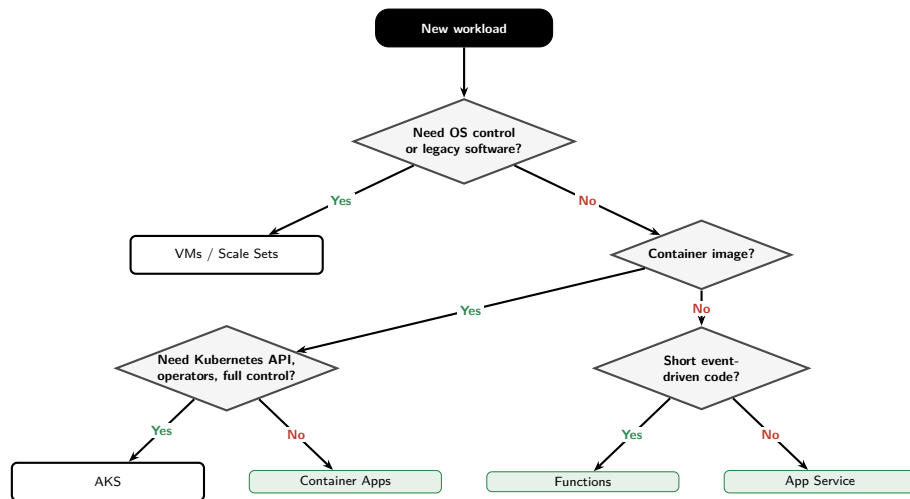


Atlas: Compute and Containers

Azure compute decision tree, Virtual Machines, App Service, Functions, Container Apps and AKS documentation, as of 2026 | Every compute service with purpose, when to choose, look-alikes, AWS and Google Cloud equivalents; VM families; scaling

[Azure](#) [Services](#) [Compute](#) [Containers](#) [AZ-900](#) [AZ-104](#)

● Choosing a compute service



Service	Use it for	Look-alike and difference	AWS	Google Cloud
Virtual Machines	lift-and-shift, custom OS, licences	Dedicated Host: whole server for you	EC2	Compute Engine
VM Scale Sets	many identical VMs, autoscale, zones	Availability set: placement only, no scaling	Auto Scaling groups	Managed instance groups
App Service	web apps and APIs (code or container)	Static Web Apps: static front end + Functions API	Elastic Beanstalk	App Engine
Functions	event-driven snippets, triggers and bindings	Logic Apps: low-code workflows	Lambda	Cloud Run functions
Container Apps	microservices, APIs, jobs; scale to zero	ACI: one container group, no autoscale	App Runner / ECS on Fargate	Cloud Run
Container Instances	quick isolated containers, CI agents, bursts	Container Apps for long-running services	Fargate tasks	Cloud Run jobs
AKS	platform teams needing Kubernetes	Azure Red Hat OpenShift: managed OpenShift	EKS	GKE
Batch	parallel HPC and render jobs	CycleCloud: HPC cluster schedulers (Slurm)	AWS Batch / ParallelCluster	Batch

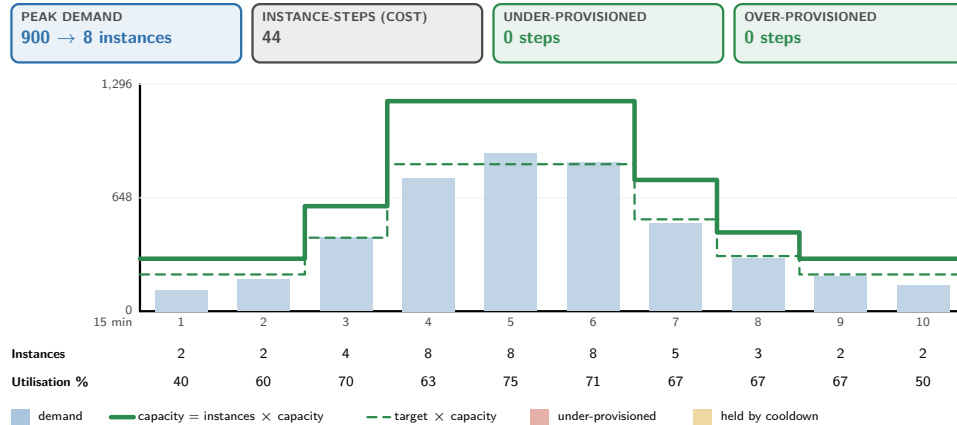
Azure VMware Solution	move vSphere estates as is	VMs: re-platform away from VMware	VMware Cloud on AWS (ended) / EVS	Google Cloud VMware Engine
Azure Spring Apps	Spring Boot apps (retiring, announced; migrate to Container Apps, check date)	Container Apps with Java features	-	-

📖 Virtual machines in one page

Family	Optimised for	Typical use
B	burstable CPU credits	dev/test, small web servers, low average CPU
D	general purpose	most production apps, small databases
E	memory (≈ 8 GB per vCPU)	databases, caches, SAP application servers
F	compute (≈ 2 GB per vCPU)	batch, gaming servers, analytics
L	storage: local NVMe	NoSQL, data warehouses on local disks
M	very large memory (up to TBs)	SAP HANA, large in-memory databases
N (NC, ND, NV)	GPUs	AI training and inference, visualisation
H / HB / HC	HPC with InfiniBand	CFD, weather, simulations

- **What a VM needs** VNet + subnet, NIC, OS disk (managed disk), optional data disks and public IP, NSG; all are separate billable or configurable resources
- **Purchase options** pay-as-you-go, reservations (1 or 3 years), savings plan for compute, Spot (up to ~90 % off, evictable), Azure Hybrid Benefit for Windows Server and SQL licences
- **Availability** single VM 99.9 % (premium disks), availability set 99.95 %, across zones 99.99 %
- **Images** Marketplace, Azure Compute Gallery for your golden images, VM Image Builder to build them

VM Scale Set autoscale through a morning peak

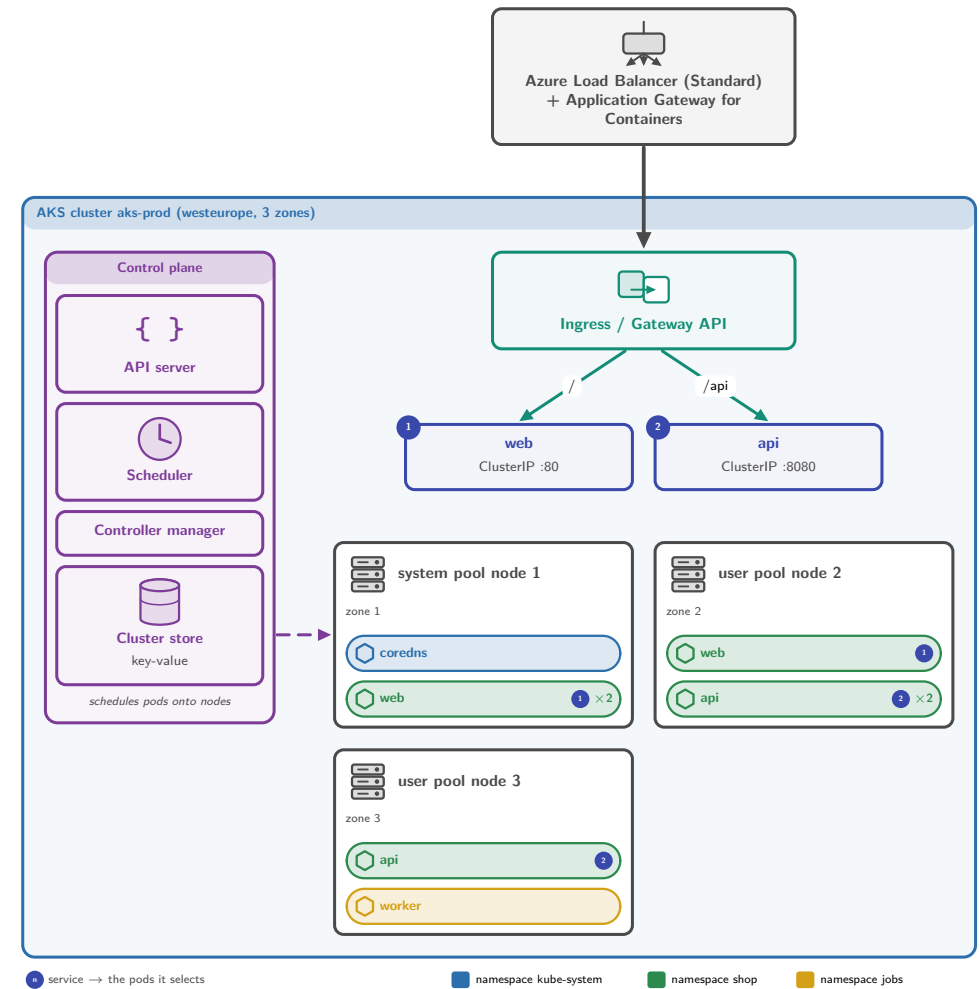


Wanted instances = $\lceil \text{demand} \div (\text{capacity per instance} \times \text{target}) \rceil = \lceil \text{demand} \div (150 \times 0.7) \rceil$, kept between min 2 and max 8;
 utilisation = $\text{demand} \div (\text{instances} \times \text{capacity})$. Scale-out happens at once, scale-in waits 1 step after the last change.

EXAM TIP

Autoscale rules in VM Scale Sets and App Service: scale out on a metric above a threshold (e.g. CPU > 70 % over 10 min), scale in below a lower threshold (e.g. < 30 %) with a cooldown; always set min, max and default instance counts. The simulation shows the cost of a low max: the steps flagged under-provisioned would be throttled.

Containers: ACI, Container Apps, AKS

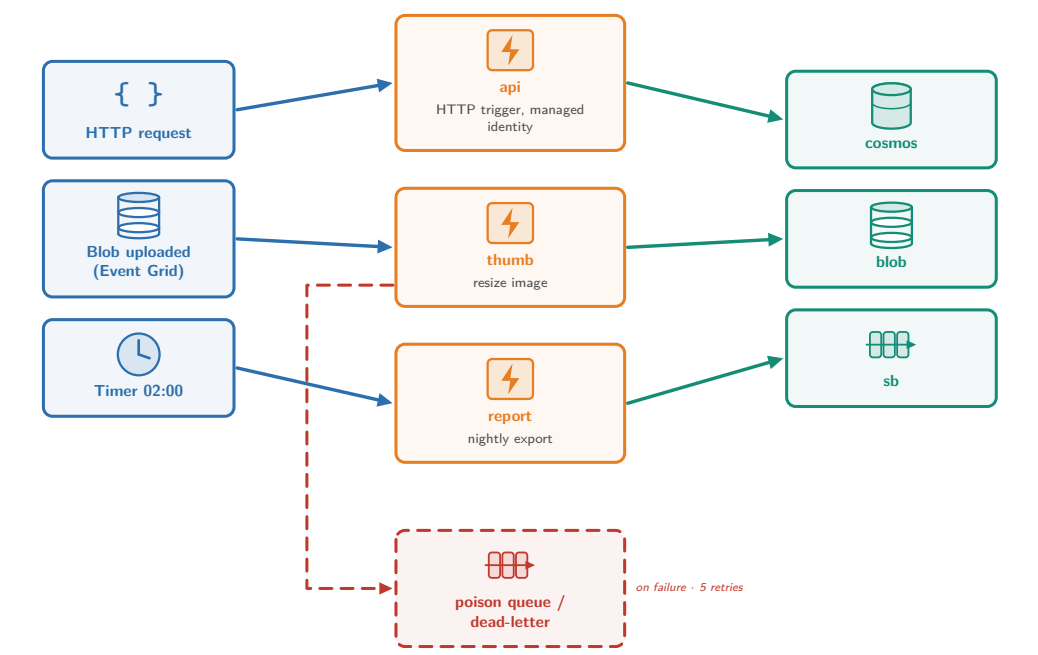


	Container Instances	Container Apps	AKS
You manage	nothing but the container group	apps, revisions, scale rules	node pools, upgrades, add-ons, workloads
Scaling	manual (fixed size)	KEDA rules, to zero	HPA, KEDA, cluster autoscaler, node autoprovisioning
Networking	public IP or VNet	environment in your VNet, built-in ingress	Azure CNI (overlay), any ingress, network policies
Pricing	per second of vCPU and memory	per second when active (consumption) or dedicated workload profiles	VMs of node pools (control plane free tier or paid Standard / Premium SLA)
Pick when	short jobs, CI agents, simple sidecars	most microservices and APIs	you need the Kubernetes API, operators, service mesh, GPU pools at scale

Same container, three ways

```
az container create -g rg -n hello --image mcr.microsoft.com/azuredocs/aci-helloworld --ports 80
↪ --ip-address Public
az containerapp up -n hello -g rg --image mcr.microsoft.com/k8se/quickstart:latest --ingress external
↪ --target-port 80
az aks create -g rg -n aks1 --node-count 3 --zones 1 2 3 --enable-managed-identity --generate-ssh-keys
az aks get-credentials -g rg -n aks1 && kubectl create deployment hello --image=nginx
```

⚡ Serverless: Functions and friends



Functions plan	Scale	Cold start	Choose when
Flex Consumption (recommended serverless plan, 2026)	per-function, to zero, high max instances	yes (always-ready instances optional)	event-driven apps needing VNet integration and fast scale
Consumption (classic)	to zero	yes	legacy apps; Linux classic consumption is being retired (check date)
Premium (Elastic Premium)	pre-warmed instances	no	constant traffic, long runs, VNet, larger instances
Dedicated (App Service plan)	plan autoscale	no (Always On)	you already pay for an App Service plan

Function compute in GB-seconds

FORMULA $G = E \times t \times m$

WHERE

- G = compute consumed per month (GB-s)
- E = executions per month (executions)
- t = average execution time (s)
- m = memory allocated per execution (GB)

WORKED

```
E = 3000000
t = 0.5
m = 0.5
G = 3 000 000 × 0.5 × 0.5
G = 750 000 GB-s per month
```

RESULT
G = 750 000 GB-s per month

READ IT
750,000 GB-s: on the classic Consumption plan the first 400,000 GB-s and 1 million executions per month are free (as of 2026, check); Flex Consumption bills on-demand GB-s and always-ready instances differently.

MISTAKES

- Shaving average duration halves G: optimise code before buying Premium
- Memory is rounded up per plan rules
- Outbound calls that wait still count as execution time

🚩 Recap and self-test

Statement	True	False
1. Container Apps can scale to zero replicas → KEDA-based scale rules on HTTP, queues or events, min replicas 0.	☒	☐
2. AKS charges for the control plane on every tier → The Free tier has no control-plane charge; Standard and Premium add an uptime SLA and LTS options.	☐	☒
3. A Spot VM can be evicted at any time → 30 seconds notice through Scheduled Events; use for interruptible work.	☒	☐
4. Each App Service app is billed separately on a shared plan → You pay for the App Service plan's instances; apps share them.	☐	☒
5. Availability sets add autoscaling → They only spread VMs over fault and update domains; Scale Sets scale.	☐	☒

Key takeaways

01

Climb the managed ladder

Functions / Container Apps / App Service before AKS and VMs

02

VM families by letter

B burst, D general, E memory, F compute, N GPU, M huge memory

03

Scale Sets scale, availability sets place

04

Container Apps for most microservices

AKS when you need Kubernetes itself

05

Functions plans

Flex Consumption, Premium, Dedicated

Q1 Azure service for a Spring Boot or Node API in containers, scale to zero, no cluster?

A Azure Container Apps.

Q2 AWS EKS and Google GKE equivalent?

A Azure Kubernetes Service (AKS).

Q3 VM family for SAP HANA?

A M-series (memory optimised, very large memory).

Q4 Run a container for 5 minutes as a CI task, simplest service?

A Azure Container Instances.

Q5 Which App Service feature gives zero-downtime releases?

A Deployment slots with swap.

Q6 Which Functions plan avoids cold starts with pre-warmed instances?

A Premium (Elastic Premium); Flex Consumption can also keep always-ready instances.

Q7 Notice before a Spot VM eviction?

A 30 seconds (Scheduled Events).

Q8 Static front end + serverless API in one service?

A Azure Static Web Apps (with managed Functions APIs).