

# Classic VPS Scaler Playbook

VPS Scaler is our largest scaler group and defined by using >60% of spend on Compute and <20% of spend on Storage.

SQL

```
WHEN (compute_amt / NULLIF(total_gross_usage, 0) * 100) > 60  
AND (storage_amt / NULLIF(total_gross_usage, 0) * 100) < 20 THEN 'Classic VPS Scaler'
```

Depending on their use case, this scaler archetype may evolve into several others:

- K8s Native Archetype - growth behavior may include users moving to managed clusters to avoid manual hand tuning many identical Droplets
- Data-Centric User - growth behavior may include adding massive block storage for logs/media to scale CPU bound tasks
- High Traffic/Network Heavy - growth behavior may include adding multiple load balancers

## User Signals & Nudges

Pain Point Signals	Trigger	Nudge
<ul style="list-style-type: none"><li>• Single points of failure</li><li>• Resizing</li><li>• Want to consider horizontal scaling</li></ul>	<ul style="list-style-type: none"><li>• 3+ droplets</li><li>• Web traffic signals - look at ports</li><li>• No load balancer</li></ul>	Load balancer
<ul style="list-style-type: none"><li>• Manual resizing</li><li>• Cyclic high usage</li></ul>	<ul style="list-style-type: none"><li>• Has load balancer</li><li>• Cyclic CPU usage</li></ul>	Autoscale Pools
<ul style="list-style-type: none"><li>• Deploying lots of similar virtual machines</li></ul>	<ul style="list-style-type: none"><li>• 3+ VMs</li><li>• Time to spin up is higher</li></ul>	Container Registry
<ul style="list-style-type: none"><li>• Management overhead is getting to be too much</li><li>• Spend is high, efficiency is low</li></ul>	<ul style="list-style-type: none"><li>• Non-homogeneous virtual machines - lots of different sizes</li><li>• 5+ VMs</li></ul>	Kubernetes
<ul style="list-style-type: none"><li>• Management overhead is getting to be too much</li><li>• Spend is high, efficiency is low</li></ul>	<ul style="list-style-type: none"><li>• Homogeneous fleet</li><li>• 5+ VMs</li></ul>	App Platform

# Research w Gemini

**Q:** Imagine you are a cloud native business that falls into the classic VPS scaler bucket. What behaviors might you exhibit when starting to consider if you might scale your infrastructure? The following template outlines different buckets users of digitalocean may fall into:

SQL

```
-- Archetype Assignment
CASE
  WHEN (modern_tools_amt / NULLIF(total_gross_usage, 0) * 100) > 15 THEN 'The AI/Modern App
Builder'

  WHEN (k8s_amt / NULLIF(total_gross_usage, 0) * 100) > 30 THEN 'K8s Native Architect'
  WHEN (compute_amt / NULLIF(total_gross_usage, 0) * 100) > 60 AND (storage_amt /
NULLIF(total_gross_usage, 0) * 100) < 20 THEN 'Classic VPS Scaler'

  WHEN (storage_amt / NULLIF(total_gross_usage, 0) * 100) > 40 THEN 'Data-Centric User'

  WHEN (compute_amt / NULLIF(total_gross_usage, 0) * 100) > 30 AND (storage_amt /
NULLIF(total_gross_usage, 0) * 100) > 30 THEN 'Full-Stack Builder'

  WHEN (networking_amt / NULLIF(total_gross_usage, 0) * 100) > 25 THEN 'High-Traffic/Network
Heavy'

  ELSE 'Generalist / Misc'
END AS archetype_label
```

## Response:

As a **Classic VPS Scaler**, my identity is built on compute. My SQL logic confirms it: over 60% of my spend goes to raw processing power, while my storage footprint remains lean (under 20%). I'm likely running a fleet of virtual machines (VMs), and my growth strategy has been "vertical" for as long as possible.

If I'm starting to consider a major scale-up, my behavior shifts from "maintaining" to "straining." Here is how I'd act as I hit the limits of the VPS model:

### 1. The "Vertical Wall" Panic

I've likely spent months simply resizing droplets (adding more RAM and CPU to the same box). My behavior now changes because I've hit the ceiling of the largest available instance.

- **Behavior:** Frequent monitoring of CPU steal and memory pressure.
- **The Shift:** I start looking at **Load**

**Balancers.** This is the pivotal moment where I realize one giant server is a single point of failure and I must begin horizontal scaling.

## 2. Manual Configuration Fatigue

Since my archetype relies heavily on `compute_amt`, I've probably been hand-configuring my servers or using basic bash scripts.

- **Behavior:** I start complaining about the time it takes to "spin up a new node."
- **The Shift:** I begin exploring **Custom Images (Snapshots)** or basic **Cloud-Init** configurations to automate the "cookie-cutter" deployment of my 5th, 10th, or 20th VPS.

## 3. The Sudden Interest in Managed Databases

As a VPS Scaler, I likely started with my database living on the same VPS as my app (the classic "monolith").

- **Behavior:** Performance degrades because the app and DB are fighting for that  $>60\%$  compute slice.
- **The Shift:** My `storage_amt` or `managed_db_amt` might start to tick upward as I realize that offloading the database is the only way to keep my compute nodes "stateless" and scalable.

## Comparative Archetype Transitions

As I evolve, my spending ratios (from your SQL template) will begin to bleed into other categories:

Current Behavior	Growth Behavior	New Archetype Potential
Hand-tuning 10 identical VPS	Moving to managed clusters	K8s Native Architect
Scaling for CPU-bound tasks	Adding massive block storage for logs/media	Data-Centric User
Adding multiple Load Balancers	Optimizing for global ingress/egress	High-Traffic/Network Heavy

#### 4. The "K8s Curiosity" Phase

Eventually, the "Classic VPS Scaler" becomes a victim of their own success. Managing 50 individual VPS instances is a nightmare.

- **Behavior:** I start looking at the **K8s Native Architect** logic. I notice that my "compute" spend is high, but my "operational efficiency" is low.
- **The Shift:** I'll start a pilot project moving one microservice to a managed Kubernetes provider to see if I can get that same compute power with less manual "babysitting."

**The VPS Scaler's Dilemma:** You love the simplicity of a virtual server, but you're starting to realize that "scaling" means more than just buying a bigger engine—it means building a fleet.

**Gemini Q:** Would you like me to analyze how your SQL logic would need to change to track users who are "migrating" from the VPS Scaler bucket into the K8s Native bucket?

**A:** yes

### Response:

To track a migration from a **Classic VPS Scaler** to a **K8s Native Architect**, we have to look for "DNA shifts" in the billing data. A migration isn't a single event; it's a trend where compute spend becomes more structured and platform-heavy.

## Identifying the "Migration" Signature

When a user moves toward Kubernetes, their `compute_amt` usually remains high, but the *type* of compute changes, and their `k8s_amt` begins to cannibalize the standard VPS spend.

Here is how we can adapt your logic to identify users in this "In-Between" state:

```
SQL
CASE
  -- THE TRANSITIONAL ARCHETYPE
  WHEN (k8s_amt / NULLIF(total_gross_usage, 0) * 100) BETWEEN 5 AND 30 AND (compute_amt /
NULLIF(total_gross_usage, 0) * 100) > 40 THEN 'Scaling VPS / K8s Explorer'

  -- THE COMPLETED MIGRATION
  WHEN (k8s_amt / NULLIF(total_gross_usage, 0) * 100) > 30 THEN 'K8s Native
Architect'

  -- THE ORIGINAL BUCKET
  WHEN (compute_amt / NULLIF(total_gross_usage, 0) * 100) > 60
AND (storage_amt / NULLIF(total_gross_usage, 0) * 100) < 20 THEN 'Classic VPS Scaler'
  ...
END AS archetype_label
```

## Visualization of the Infrastructure Shift

As a business moves from managing individual servers to an orchestrated cluster, the architecture shifts from a "Pets" model (where every server is named and cared for) to a "Cattle" model (where servers are replaceable resources).

## Key Indicators of a Scaling VPS User

If I am this user, you will see specific "telltale" behaviors in my account activity that signal I'm ready to move out of the VPS bucket:

- **Snapshot Frequency:** I'm taking snapshots of my droplets every time I scale, rather than building from scratch.
- **Load Balancer Attachment:** I suddenly have one Load Balancer pointing to 4+ droplets of the same size.
- **API Usage Spikes:** Instead of using the Cloud UI, I'm starting to use the API or Terraform to spin up resources, showing I'm craving automation.
- **The "VPC" Move:** I start configuring private networking (VPC) to allow my growing fleet of VPS nodes to talk to each other securely without hitting the public internet.

## Comparison Table: VPS Scaler vs. K8s Architect

Metric	Classic VPS Scaler	K8s Native Architect
Primary Unit	Individual Droplet (VM)	Pod / Node Pool
Scaling Method	Vertical (Resize)	Horizontal (Autoscaling)
State Management	Local Disk / Shared Files	Managed DBs / Volume Claims
Management Overhead	High (Manual Updates)	Low (Self-healing)

**Gemini:** Would you like me to help you draft a migration plan that moves you from "Individual VPS" to "Managed Kubernetes" while keeping your compute-to-storage ratio stable?

**Response:**

Moving from a "Classic VPS Scaler" to a "K8s Native Architect" on DigitalOcean is a classic evolution for a growing business. Your compute spend (>60%) remains the same, but the way you manage those CPU cycles shifts from manual labor to automated orchestration.

Here is a 2026-ready migration plan to help you leap over those friction points.

## Phase 1: The "Cattle" Assessment (12-Factor Prep)

Before touching Kubernetes (DOKS), you must stop treating your Droplets like individual "Pets."

- Decouple Storage: If your app writes to local `/var/www/uploads`, move those to DigitalOcean Spaces (S3-compatible).
- Managed Databases: Stop running MySQL/Postgres inside your Droplet. Migrate to DigitalOcean Managed Databases. This ensures that when your K8s pods die and restart, your data stays safe and external.
- Containerize: Take your Snapshot-based workflow and convert it into a `Dockerfile`.

## Phase 2: The Pilot Cluster (DOKS Setup)

Instead of one massive 16GB Droplet, you'll start with a small Managed Kubernetes (DOKS) cluster.

### The Migration Pipeline:

1. Build & Push: Use the DigitalOcean Container Registry (DOCR) to store your images. 2. Cluster Provisioning: Use `doctl` or the UI to spin up a 3-node pool.

Tip: Use "Regular" or "Premium Intel" nodes to match the compute performance you had in your VPS Scaler days.

3. The LB Handover: You can actually migrate your existing DigitalOcean Load Balancer ID to your new K8s service. This allows you to keep your public IP address while swapping the backend from Droplets to Pods.

## Phase 3: Solving the "VPS Scaler" Friction

As you migrate, you'll use specific DOKS features to solve the problems we discussed earlier:

### 1. Scaling without "Reboots"

- Horizontal Pod Autoscaler (HPA): Instead of resizing a server, K8s just adds more "Pods" (clones of your app).

- Cluster Autoscaler: If your pods run out of room, DigitalOcean will automatically add a new worker node to the cluster. No more downtime for resizes.

## 2. Zero-Downtime Deployments

- Surge Upgrades: When you update your app or the underlying K8s version, DO spins up "extra" nodes first, moves your traffic, and then kills the old ones.

### Phase 4: Strategy for "Sticky" Data

If you have data that *must* stay on a disk (like a legacy CMS or a specialized cache), use DigitalOcean Block Storage Volumes with the CSI (Container Storage Interface) driver.

- Behavior: When a Pod moves from Node A to Node B, DOKS automatically detaches the volume and re-attaches it to the new node for you.

Comparison Table: Your Spend Transformation

Metric	Before (VPS Scaler)	After (K8s Architect)
Compute Spend	Individual Droplets	DOKS Worker Node Pools
Scaling Logic	Vertical (Manual)	Horizontal (Auto)
Storage Spend	Local Boot Disks	Spaces + Block Storage
Management	SSH + Snapshots	kubectl + GitOps

**Q:** If I'm still a VPS scaler and haven't moved over to k8s yet, when and why might I consider using autoscale pools?

**Response:**

If you're a **Classic VPS Scaler**, you've likely spent your life managing individual Droplets. You might be resistant to Kubernetes because of the "complexity tax," but you've outgrown the manual "Resize" button.

This is exactly where **Droplet Autoscale Pools** come in. They are the "missing link" between a single giant VPS and a full Kubernetes cluster.

## When to use Autoscale Pools

You should consider moving from standalone Droplets to an Autoscale Pool when you recognize these three "trigger" scenarios:

### 1. The "Traffic Rollercoaster"

If your CPU usage looks like a mountain range—spiking during business hours and flatlining at 3 AM—you are currently overpaying.

- **The "Why":** You're likely paying for a 16GB Droplet just to handle 4 hours of peak traffic. With an Autoscale Pool, you can run two small 4GB Droplets at night and automatically burst to eight Droplets during the day.
- **The Result:** You only pay for that<sup>1</sup>>60% compute spend when you actually need it.<sup>2</sup>

### 2. The "Fear of the 2 AM Alert"

If your current scaling strategy is "I'll manually add a server if the site gets slow," you are a single viral post away from a crash.

- **The "Why":** Autoscale Pools monitor your aggregate CPU or RAM.<sup>3</sup>If the pool hits your threshold (e.g., 70% CPU), DigitalOcean spins up a new Droplet, joins it to your Load Balancer, and starts serving traffic **without you waking up.**<sup>4</sup>

### 3. The Need for "Self-Healing"

In a classic VPS setup, if a Droplet's hypervisor has an issue or the OS hangs, your site goes down.

- **The "Why":** Autoscale Pools work with **Load Balancer Health Checks.**<sup>5</sup>If a Droplet

fails a health check, the pool considers it "unhealthy," kills it, and provisions a fresh one from your base image.<sup>6</sup>

## Why choose Pools over Kubernetes?

If you are already comfortable with VPS management, Autoscale Pools give you the benefits of scaling without the learning curve of K8s.

Feature	Classic VPS (Standalone)	Droplet Autoscale Pools	Kubernetes (DOKS)
Learning Curve	Zero	Low (Minutes to set up)	High (Weeks to master)
Scaling Type	Vertical (Manual)	Horizontal (Automatic)	Horizontal (Granular)
Downtime	Requires Reboot	Zero (New nodes add on)	Zero
Management	Manual SSH / Scripts	Snapshot / Cloud-Init	Container Images / YAML

## The "Catch": Your App Must Be Stateless

To use Autoscale Pools, your Droplets must be "clones." You cannot store unique data on their local disks.

- **The Transition:** You must move your database to a **Managed DB** and your file uploads to **Spaces**.
- **The Benefit:** Once you do this, your app is essentially "ready" for Kubernetes whenever you decide to

make that final jump.

**Pro Tip:** When setting up your pool, use **Cloud-Init (User Data)** scripts.<sup>7</sup> This ensures that every time a new Droplet is born, it automatically pulls your latest code from GitHub or a Docker registry, keeping your "clone army" up to date.