Roblox Workspace vs. Services — Quick Guide

Workspace = the live 3D world

- Put in Workspace only what must be visible/simulated, collide, or be detected by raycasts/pathfinding right now.
- Everything else (scripts, prefabs, configs, shared assets) belongs in the appropriate service.

Put in Workspace

- Geometry & set dressing (Parts/Models, Terrain, doors, platforms, buttons).
- Physical interactables (triggers/transparent Parts, pickups/Tools on the ground, projectiles, ragdolls, constraints/attachments).
- Active NPCs/enemies (spawn the actual live models here; keep prefabs elsewhere).
- World-anchored VFX/SFX (ParticleEmitter, 3D Sound, Beams, BillboardGui/SurfaceGui attached to a Part).
- Anything that must be detected via collisions or raycasts right now.
- CurrentCamera (client-side).

Do NOT put in Workspace

- Server scripts → ServerScriptService.
- 2D UI (menus/HUD) → StarterGui (replicated into PlayerGui).
- Assets/config/prefabs to clone later → ServerStorage (server-only) or ReplicatedStorage (server + clients).
- RemoteEvent/RemoteFunction shared endpoints → ReplicatedStorage.
- ModuleScripts: server-side in ServerScriptService/ServerStorage; shared modules in ReplicatedStorage.

Performance & streaming tip

- With StreamingEnabled, anything under Workspace is streamable. Do not park invisible asset banks in Workspace.
- Keep source assets in *Storage*, clone/parent into Workspace only when needed.

Rule of thumb

- If deleting it from Workspace would immediately change the world visually/physically → it belongs in Workspace.
- If it is a resource, prefab, or script → store it in the dedicated service, not in Workspace.

StreamingEnabled — Essentials

What is StreamingEnabled?

- Roblox client-side content streaming. When enabled on Workspace, the client receives/loads only areas near the player (or ReplicationFocus).
- Benefits: faster joins, lower memory, supports larger worlds and lower-end devices.

Key scripting implications

- On the client, far-away Instances may not exist yet. Use WaitForChild(), write nil-tolerant logic, or listen for arrivals (e.g., via tags/CollectionService).
- The server always has the full scene; design server logic accordingly.

Core settings (Workspace)

- StreamingTargetRadius: desired radius to load around the focus.
- StreamingMinRadius: minimum guaranteed loaded radius.
- StreamingIntegrityMode (formerly StreamingPauseMode): can pause gameplay while the nearby region loads to preserve integrity.

Per-model controls

- Model.StreamingMode: Default/NonAtomic, Atomic (arrives/leaves as a whole), Persistent, PersistentPerPlayer (always present for that player).
- Use Persistent sparingly—overuse negates streaming gains.
- Workspace.PersistentLoaded fires when persistent models have been sent to clients.

Practical workflow

- Place active, nearby gameplay objects under Workspace; keep prefabs and data in *Storage* services.
- For client scripts referencing distant objects, either mark those models Persistent/Atomic or defer logic until they stream in.

Migration checklist (quick)

- Audit client code for direct references to distant Instances; add WaitForChild()/signals.
- Move asset banks out of Workspace into ServerStorage/ReplicatedStorage.
- Tune target/min radii; set integrity mode to fit your UX; only mark truly critical models as Persistent.