Who Owns What: A Responsibility Map For The Enterprise Delivery Tier

Most enterprises have funded the agent layer properly and left the tier underneath it unowned. The delivery tier gets assembled during a project by parties who are each correct about the limits of their own responsibility, and nobody signs for the whole. That held up while the traffic was human and forgiving, but it stops holding now that most of what arrives is automated.

This map in this document is built for one conversation: take it into your next platform review, work down the layers, and score each line item. When something is scored as unowned, you have found the gap that agentic traffic will surface first.

How to use the map

Score it honestly.
Work down the nine layers and mark each one owned, partly owned, or unowned. Partly owned counts as unowned when it is under pressure.

Look at the pattern, not the total.
Unowned layers cluster around the seams between parties, which is exactly where agent traffic concentrates.

Then make a decision, not a plan.
Every unowned layer needs either internal resourcing, with headcount, budget, mandate and instrumentation, or a third party who signs for it.
Both are legitimate. Ambiguity is not.

LAYER OF THE DELIVERY TIER	WHO EVERYONE ASSUMES OWNS IT	WHAT THAT PARTY IS ACTUALLY CONTRACTED FOR	THE GAP THAT IS LEFT	SCORE & NOTES
CMS and authoring	The CMS vendor	Availability and support of the CMS itself, to the terms of its SLA	Nothing beyond the CMS boundary. The vendor's SLA stops where your architecture starts	
Content and delivery APIs	The CMS vendor	The API endpoints as published, at documented limits	Behavior under sustained programmatic call volume, and what happens when agents exceed those limits	
Front-end rendering and caching	The systems integrator, or nobody	Building it to specification and reaching go-live	Running it afterwards. Cache strategy, revalidation and render performance drift with no owner	
Build and deployment pipelines	The internal platform team	Keeping them working, usually alongside a full delivery workload	Pipeline maintenance, security gates and release governance as the estate grows	
Cloud environments and tenancy	The cloud provider	The infrastructure underneath, per the shared responsibility model	Everything above the hypervisor. The model is explicit that the workload is yours	
Capacity and performance under load	Assumed to be self-managing	Nothing. Autoscaling is a configuration, not an owner	Deciding headroom, setting limits and testing against agent traffic patterns rather than human ones	

What ownership actually requires

A name on an org chart is not ownership. For each layer in the table, ownership needs all four of these at once. Most organizations have two.

	A NAMED OWNER	A BUDGET LINE	AN SLA SOMEBODY HAS SIGNED	OBSERVABILITY THE OWNER CAN ACT ON
WHAT IT IS	A single person or team recorded as accountable for the layer, by name, somewhere a colleague could find it without asking.	Funding to maintain the layer that doesn't come out of the same pool as feature delivery.	A commitment with a target and a consequence attached, held internally or by a third party.	Visibility deep enough, and current enough, that the owner can see a problem and do something about it while it's still happening.
WHY IT MATTERS	Shared accountability behaves like no accountability the moment something starts degrading. Escalation replaces action, and the incident lasts as long as the discussion about whose job it is.	When the two compete, features win, because their value is visible and maintenance only becomes visible once it's absent. The layer then degrades slowly enough that nobody is ever forced to make a decision about it.	Accountability without visibility only decides in advance who gets blamed. Agent-driven failures are also quiet, so the reporting that caught human-era problems won't surface them at all.	Accountability without visibility only decides in advance who gets blamed. Agent-driven failures are also quiet, so the reporting that caught human-era problems won't surface them at all.
HOW TO TEST IT	Ask three people in three different functions who owns the layer. Three answers, or a job title instead of a name, means it isn't owned.	Without a consequence you have a preference, and preferences are the first thing to go when the team is under pressure. They also can't be enforced after the fact, which is when you'd want them.	Ask what happens if the target is missed. If nobody can describe the consequence, it's an expectation.	Pick one failure mode in this layer and ask how you would find out. Anything that relies on a customer complaint, or a report produced the next morning, is a gap.

So this is what to look for when somebody tells you a layer is owned. All four, not two. An owner with no budget can’t act. A budget with no visibility gets spent in the wrong place. An SLA nobody measures is a piece of paper.

What to insist on, whoever ends up holding it

Whether you resource the tier internally or contract it out, the requirements are the same. It should run inside your own cloud tenant, within your governed boundary. That gives the residency question a clear answer, and it keeps control from moving offshore along with the accountability. Observability should be deep enough to act on in the moment. Security posture and deployment discipline should be part of the architecture rather than a review cycle wrapped around it. And there should be a name and a signed SLA attached, with consequences that are real.

That list is what we built Dataweavers to do, so we are not a neutral party on it. It holds regardless of who runs the tier for you, and it is a fair test to put to us or to anyone else.

Bring us the layer nobody wants to own

If you have worked through the map and the gaps are clear, we are happy to talk through what closing them looks like in practice, including the parts you should keep in-house. We run the delivery tier inside our customers' own Azure tenants, with observability, security posture, deployment discipline and operational ownership as the product itself

hello@dataweavers.com | Fix Less. Build More.