



BEYOND THE GREEN SQUARE

Governing the Thing You Cannot Read: Containerisation and the Programme Manager's Literacy Problem

When the orchestration layer became the programme's critical path, and the people accountable for it could no longer describe what it did

Governing the Thing You Cannot Read: Containerisation and the Programme Manager's Literacy Problem

Programme · No. 189

Originally written August 2016 | Re-edited and re-rendered for the WhereBeyond Edition, 2025

"The green square is not a lie he is telling. It is a lie he is unable to detect."

1. When the map stops matching the ground

The status report was green and the demonstration worked. A service that had run for years on a pair of virtual machines was now running in containers, and on the screen it looked identical — the same responses, the same latency, quicker to deploy. The programme manager reported the milestone as met. Then someone from the risk function asked a plain question: if one of those containers dies at two in the morning, what happens to the data it was holding? The room went quiet. The programme manager looked to the engineers; the engineers looked at the floor. The honest answer, which took another week to surface, was that nobody accountable for the programme had asked where the state lived. The migration had moved a stateful workload onto an infrastructure built to treat every running instance as disposable, and the rollback plan still assumed the old world of virtual-machine snapshots that no longer existed.

That gap — between what was reported and what was true — was not a failure of engineering. The engineers knew precisely what they had built. It was a failure of governance, and more precisely a failure of the person whose job was to govern. He could not read the thing he was accountable for.

I want to argue something that still sounds impolite in a steering committee: in the container era, technical literacy has become a delivery competence for programme managers, not an optional enrichment for the ones who happen to enjoy it. The manager who cannot form an independent picture of the architecture is no longer merely less effective. He is a source of delivery risk in his own right, because the reporting line now runs through a person who cannot tell whether the report is true.

The question is no longer whether a programme manager can build the system. It is whether he can tell, without being told, when the story he is being given does not add up.

2. Why the ground moved

For most of the last two decades a capable programme manager could govern technology work without understanding it in any depth, and the method held up reasonably well. The unit of delivery was a server, or a virtual machine, and it behaved like a thing you could point at. You provisioned it, you patched it, you backed it up, and when it broke you restored it. The abstractions were stable enough that a manager could track them through a plan and a RAG status without ever reading a line of configuration. Governance-by-proxy worked because the underlying model was slow, physical, and legible.

Containers broke that comfortable arrangement, and the orchestration layer broke it further. A container is not a small server; it is a process that assumes it is disposable. Schedule a few hundred of them across a cluster with Kubernetes, or Swarm, or Mesos and Marathon — the field has not settled, and anyone who tells you it has in the summer of 2016 is selling something — and the system stops being a set of machines you can point at. It becomes a control loop that is constantly killing and recreating workloads to match a declared desired state. The things that now carry the risk are invisible on the old map: where state is externalised, how the scheduler behaves under pressure, what happens to in-flight work when a node is drained, whether the desired state held in version control actually matches what is running.

None of this is beyond a programme manager. But none of it is legible from a plan, a milestone, or a green square on a slide. And that is the shift that has quietly changed the job: the risk has migrated into exactly the layer that governance-by-proxy cannot see.

3. "That is what the technical lead is for"

The standard objection to everything I have just said is a good one, and it deserves to be put at its strongest rather than waved away. It runs like this. A programme manager governs outcomes, not implementations. The whole point of a delivery organisation is the division of labour: the architect owns the architecture, the technical lead owns the technical decisions, and the programme manager owns scope, cost, schedule, risk, and the coordination of people. Asking the manager to understand Kubernetes is a category error and, worse, an invitation to meddle in decisions he is not equipped to make. Push this line and you reach a tidy conclusion: technical literacy is not the manager's job, and demanding it just produces a worse architect and a distracted manager.

I have some sympathy with this, because its opposite failure is real. A programme manager who has learned just enough to be dangerous, and who now second-guesses every engineering decision in the language of a conference talk he half-followed, is a genuine menace. But the objection smuggles in an assumption that no longer holds — that the boundary between "outcome" and "implementation" sits where it used to. When state, resilience, and rollback are properties of the orchestration layer rather than of the application, the implementation is the outcome. The question "what happens to the data when a container dies" is not an architectural nicety. It is a question about whether the service loses customer records at two in the morning,

which is about as pure an outcome as a programme has.

The division of labour was never a licence for the manager not to understand. It was a licence not to decide. Those are different things, and the container era has pulled them apart. The manager still should not choose the scheduler. But he can no longer govern a programme whose central risks he is constitutionally unable to perceive, and then take shelter behind the technical lead when the risk arrives.

4. Literacy is not the same as coding

If the demand were that programme managers learn to code, or to administer a cluster, the objection would win, because that demand is both unrealistic and beside the point. Literacy is not authorship. A literate manager does not write the manifests; he reads the shape of the risk. Concretely, in this era, that means being able to do a small number of things without a translator:

- Ask where state lives, and understand why the answer matters — that a stateless service and a stateful one carry entirely different migration risk even when they look identical in a demonstration.
- Tell the difference between "containerised" and "cloud-native": that packaging an application in a container without externalising its state, its configuration, and its logging has moved the code and left every hard problem exactly where it was.
- Read a rollback story critically enough to notice when it still assumes the old world — snapshots, pinned hosts, manual restores — beneath a new architecture that has abandoned all three.
- Recognise when "ninety per cent containerised" is a dangerous number, because the difficult ten per cent — the stateful, the legacy-integrated, the compliance-bound — was always going to carry most of the risk, and reporting by percentage-of-services hides precisely that.

None of these requires writing a line of YAML. All of them require the manager to hold an independent model of the system in his head, good enough to know which questions expose the truth. That is a reading skill, and reading skills are learnable by any capable adult who decides the material is worth the effort.

"A programme manager does not need to build the cluster. He needs to be un-lie-to-able about it."

The reason this matters so much more now than it did five years ago is that the container stack has dramatically widened the distance between "looks done" and "is done." A virtual-machine migration that demonstrated cleanly usually was, more or less, done. A container migration that demonstrates cleanly may be nowhere near done, because the demonstration exercises the happy path and every hard property of the new architecture — failure, scale, state, rollback — lives off the happy path. The manager who cannot read the architecture cannot see that distance, and so he reports

the demonstration. The green square is not a lie he is telling. It is a lie he is unable to detect.

5. The uncomfortable conclusion

I have watched capable, diligent programme managers deliver container programmes that were structurally unsound and never know it, because every instrument they trusted told them things were fine. They were not negligent. They were governing with a map drawn for a landscape that no longer exists, and no one had told them the survey was out of date. The failure, when it came, was always logged as a technical incident, which let everyone avoid the more awkward diagnosis: the governance had been blind for months.

So the conclusion I will defend is narrow but firm. We should stop treating technical literacy as a personal quirk of the more technical programme managers, and start treating its absence as a delivery risk to be managed like any other. That does not mean sending every manager on a Kubernetes course. It means being honest, when we staff a container programme, that a manager who cannot read the architecture will need a named, trusted, independent technical conscience sitting inside his governance — not the delivery team reporting up, but someone whose job is to tell him when the green is not real. And it means, for the managers themselves, accepting that the comfortable era of governing-by-proxy has ended for this class of work, and that the reading is now part of the job.

The programme in the story recovered. The state was externalised, the rollback plan was rewritten around the world as it actually was, and the service went on to run perfectly well. What never fully recovered was the steering committee's faith in the green square — which, on reflection, was the most useful thing the whole episode produced.