Containerization & Deployment Decision Framework

A Systematic Assessment Guide for
Modern Application Deployment

Version 1.0 | February 2026

Tier 1: Fitness	Tier 2: Constraints	Tier 3: Strategy

About This Framework

This framework provides a structured, repeatable method for making deployment decisions about applications. Rather than relying on intuition or checklists, it walks you through a three-tier evaluation that progressively narrows the decision space: first assessing whether an application is technically suitable for containerization, then checking what real-world constraints exist, and finally selecting an execution strategy.

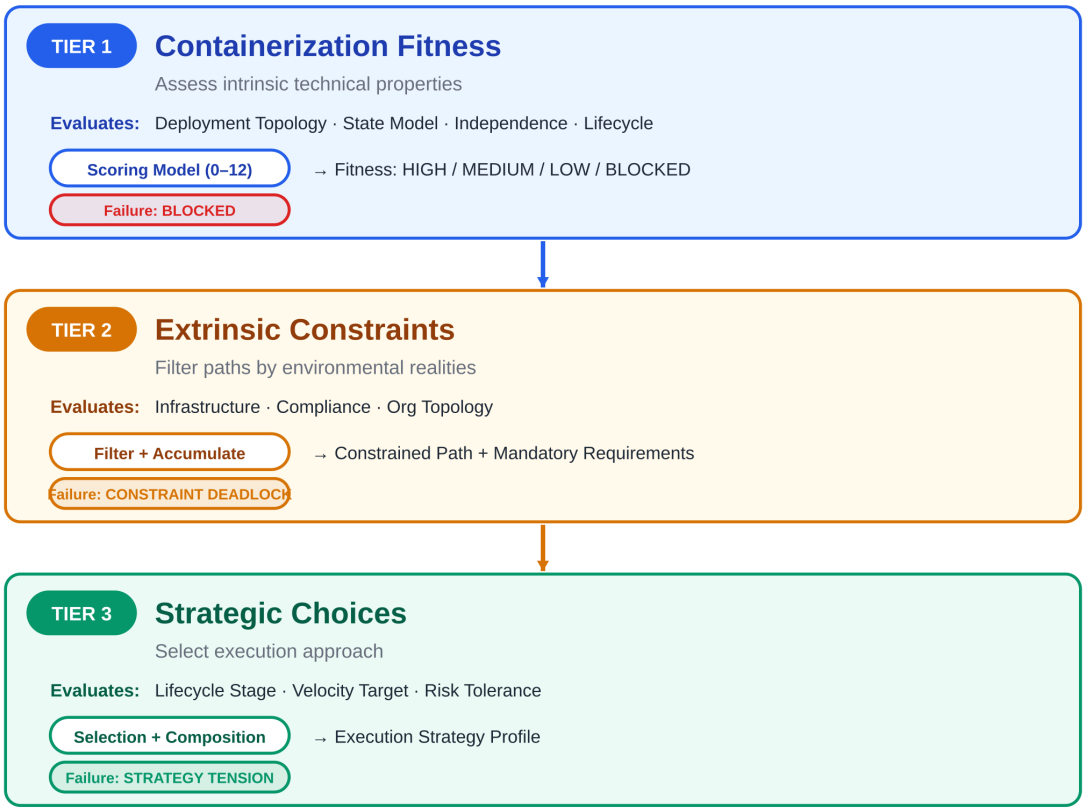
Who is this for? Engineers, architects, and technical leads who need to make or justify deployment decisions. You should be familiar with containers and cloud concepts, but deep platform engineering expertise is not required.

How to use it: Work through the tiers sequentially. Tier 1 discovers your application's fitness (you can't choose this). Tier 2 checks environmental realities (you work within these). Tier 3 is where you make genuine choices about execution strategy.

> Core principle: Each tier has a distinct nature. Tier 1 is discovery (what IS the application). Tier 2 is reality-checking (what are we ALLOWED to do). Tier 3 is decision-making (what do we CHOOSE). Skipping tiers or reversing the order leads to suboptimal decisions.

Framework Overview

The framework operates as a sequential pipeline. Each tier takes input from the previous tier and progressively refines the deployment decision.

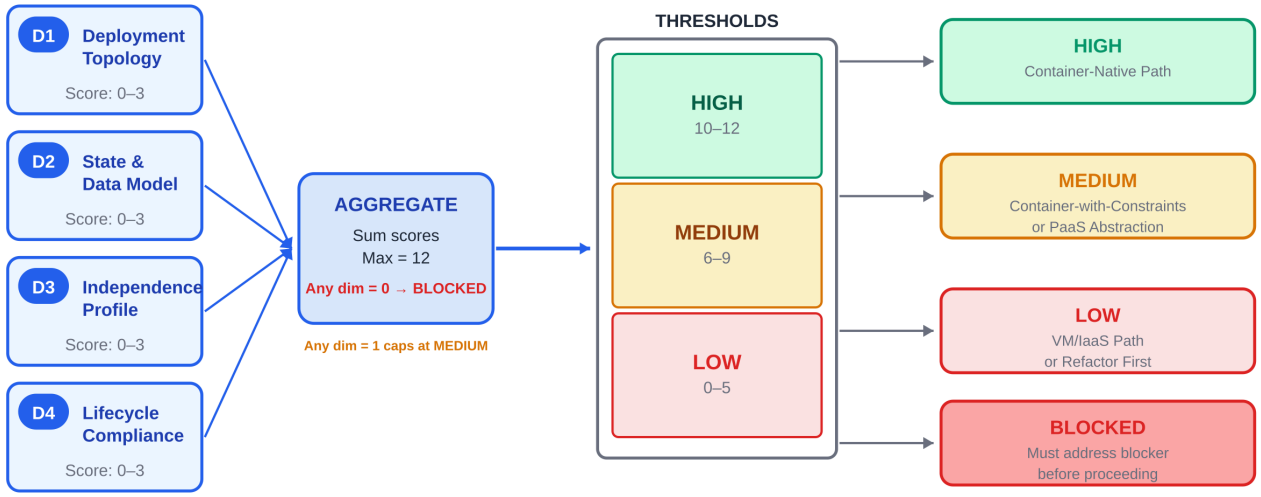


Structural Comparison Across Tiers

Aspect	Tier 1: Fitness	Tier 2: Constraints	Tier 3: Strategy
Model	Scoring (0–12)	Filter + Accumulate	Select + Compose
Nature	Discovery	Reality check	Human decision
Agency	None — fitness is intrinsic	Minimal — constraints are external	High — genuine choice
Failure Mode	**BLOCKED** (hard)	**DEADLOCK** (hard)	**TENSION** (soft)
Output	Fitness Classification + Path Candidates	Constrained Path + Requirements	Execution Strategy Profile

TIER 1: Containerization Fitness Assessment

Tier 1 evaluates four intrinsic technical dimensions of the application, scoring each 0–3. The combined score (max 12) determines fitness classification and which deployment paths are candidates.



Dimension Reference

	Dimension	What It Assesses	Key Question	Score 0 (Blocker)
D1	Deployment Unit Topology	Architecture type, process model, build/release boundaries	What constitutes a single deployable unit?	Distributed Monolith (services coupled at deployment)
D2	State & Data Model	State management, data ownership, transaction boundaries	Where does state live and who owns it?	Distributed State Coupling (shared mutable state)
D3	Independence Profile	Coupling, communication patterns, failure isolation	Can components live, fail, scale independently?	Hidden Coupling (undocumented dependencies)
D4	Lifecycle Compliance	Startup/shutdown, config model, health signaling	Does it respect container orchestration contracts?	Incompatible Lifecycle (e.g. >2min start, no signal handling)

Scoring Reference: All Dimensions

Score	D1: Topology	D2: State	D3: Independence	D4: Lifecycle
3	True Microservices or Modular Monolith	Cloud-Native Stateless (externalized state)	Fully Independent (fail/scale/deploy alone)	Cloud-Native (<30s start, SIGTERM, env config, health)
2	Traditional Monolith (single unit, self-contained)	Managed Statefulness (volumes, external DBs)	Bounded or Coordinated dependence (explicit)	Compliant with gaps (30s–2min start, partial signals)

1	—	Embedded State (local files, in-memory caches)	—	Legacy Lifecycle (slow start, no graceful shutdown)
0	**BLOCKER:** Distributed Monolith	**BLOCKER:** Distributed State Coupling	**BLOCKER:** Hidden Coupling	**BLOCKER:** Incompatible Lifecycle

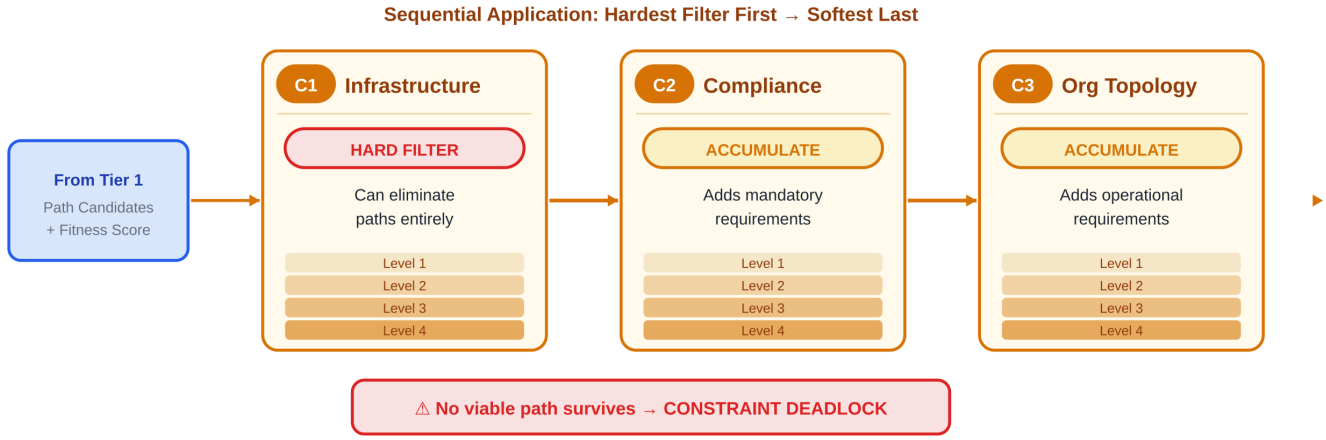
Aggregation Rules

Classification	Score Range	Conditions	Primary Path Candidates
HIGH	≥ 10	No dimension below 2	Container-Native (full GitOps, HPA, canary/rolling)
MEDIUM	6–9	Or: any dimension = 1 (caps at MEDIUM)	Container-with-Constraints, PaaS, or Hybrid
LOW	≤ 5	—	VM/IaaS, PaaS with abstraction, or Refactor First
BLOCKED	N/A	**Any dimension = 0**	Must address architectural blocker before proceeding

Special rule: Any dimension scoring 1 automatically caps the overall fitness at MEDIUM, regardless of the total score. This prevents a single weak area from being masked by strong scores elsewhere.

TIER 2: Extrinsic Constraints

Tier 2 filters the path candidates from Tier 1 against real-world constraints. Unlike Tier 1's scoring model, Tier 2 uses a **filtering-and-accumulation** model: constraints either eliminate paths or add mandatory requirements to surviving paths. Constraints are applied sequentially from hardest filter to softest.



Constraint Reference

	Constraint	Level 1	Level 2	Level 3	Level 4
C1	**InfrastructureAvailability**	IA1: Full Cloud-Native (managed K8s + ecosystem)	IA2: Container-Capable (runtime, limited orchestration)	IA3: PaaS-Constrained (Cloud Run, Heroku, etc.)	IA4: Traditional Only (VMs/bare metal)
C2	**CompliancePosture**	CP1: Unrestricted (team discretion)	CP2: Standard (org policy, scanning, RBAC, audit)	CP3: Regulated (SOC2, HIPAA, PCI, GDPR)	CP4: Highly Regulated (air-gap, FIPS, SBOM, multi-party)
C3	**OrganizationalTopology**	O1: Single Team (full autonomy)	O2: Multi-Team Shared (coordinated releases)	O3: Multi-Team Distributed (independent pipelines)	O4: Cross-Organizational (contractual boundaries)

Application Order & Mechanics

C1 Infrastructure is applied first because it is the hardest filter — it can eliminate entire deployment paths. If Kubernetes is simply not available (IA4), the Container-Native path is eliminated regardless of how fit the application is.

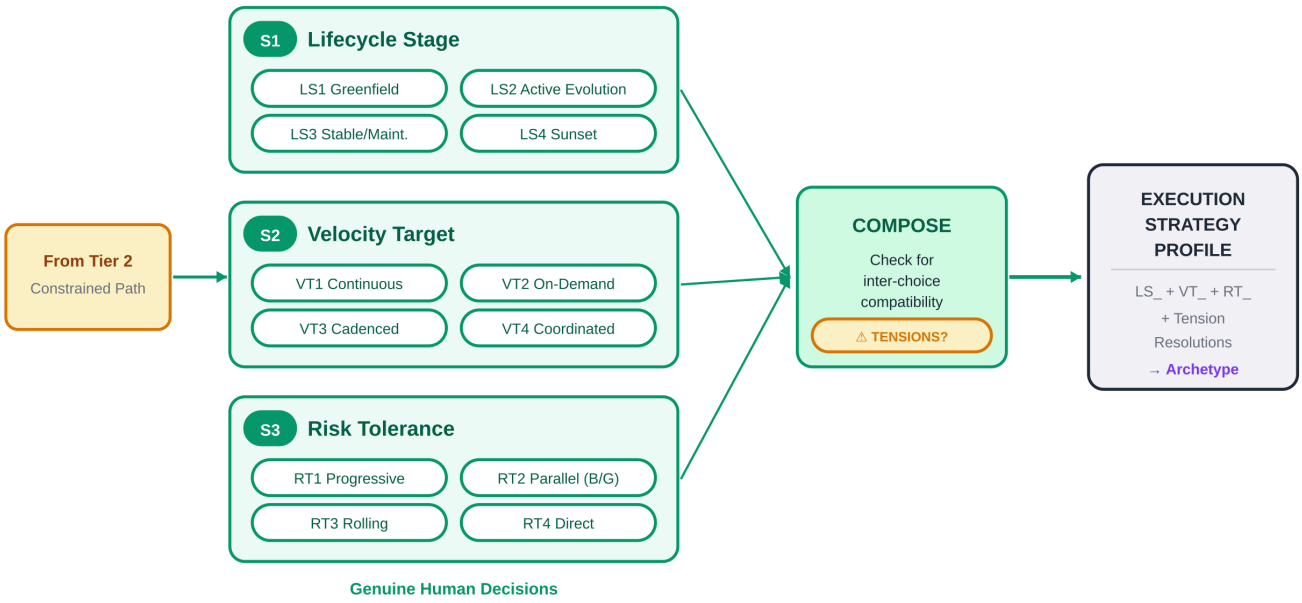
C2 Compliance is applied second. It rarely eliminates paths outright but accumulates mandatory requirements: image scanning, approval gates, audit logging, environment isolation, etc. Higher compliance levels add progressively more requirements.

C3 Org Topology is applied third as the softest constraint. It primarily adds operational requirements (pipeline-per-team, contract testing, API versioning) rather than eliminating paths. Conway's Law operates in full force here: team structure shapes system boundaries.

CONSTRAINT DEADLOCK occurs when no viable path survives all three constraints. Example: HIGH fitness (container-ready) + IA4 (no containers available). Resolution order: (1) invest to change the hardest constraint, (2) accept a suboptimal path with documented trade-offs, (3) return to Tier 1 reassessment.

TIER 3: Strategic Choices

Tier 3 determines ***how*** to execute the deployment path, not ***which*** path. It uses a **selection-and-composition** model: you select from available options in three choice areas, and the selections compose into an Execution Strategy Profile. Unlike Tiers 1 and 2, this involves genuine human decision-making — reasonable people with identical upstream outputs could choose differently.



Choice Area Reference

S1: Lifecycle Stage — Where is this application in its lifecycle?

Level	Description	Investment Justification	Key Implication
LS1 Greenfield	New application, no existing deployment	High (building foundations)	Max design freedom; adopt target-state from day 1
LS2 Active Evol.	Existing app under active feature development	Medium-High (changes compound)	Incremental adoption; strangler fig for architecture changes
LS3 Stable	Bug fixes, security patches, minor updates only	Low-Medium (short ROI window)	"If it works, document it and leave it" is legitimate
LS4 Sunset	Being decommissioned or replaced	Minimal (investment is waste)	Safe operation until decommission; minimize changes

S2: Velocity Target — How frequently must changes reach production?

Level	Description	Pipeline Requirement	Natural Fit
VT1 Continuous	Commits flow to production automatically	Trunk-based dev, full automated testing, auto-promotion	LS1/LS2 + HIGH fitness + O1/O2 teams
VT2 On-Demand	Always releasable; deploy is a deliberate decision	CI keeps trunk releasable, feature flags, 1-click deploy	LS2 where business timing matters
VT3 Cadenced	Releases on regular schedule (sprint, weekly, monthly)	Release branches, scheduled promotion, integration testing	LS2/LS3 + CP2/CP3 compliance + O2 teams
VT4 Coordinated	Multiple services/teams release together in sync	Release train, dependency mapping, cross-service testing	O3/O4 teams. **If forced by coupling, revisit Tier 1**

S3: Risk Tolerance — What is the acceptable blast radius of a failed deployment?

Level	Description	Requires	Blast Radius
RT1 Progressive	Canary: gradual traffic shifting with automated evaluation	Traffic splitting, observability, automated rollback	Minimal: starts small, expands on success
RT2 Parallel	Blue-green: full parallel environments, atomic switch	2x resources during transition, atomic routing, DB compatibility	Full on cutover, but instant rollback available
RT3 Rolling	Incremental instance replacement within same environment	Orchestrator with rolling updates, health checks	Incremental; rollback = roll forward to previous version
RT4 Direct	All-at-once: stop old, start new (may involve downtime)	Minimal infrastructure	Maximum: everything affected at once

Execution Archetypes

Common combinations that map to recognizable patterns. Not prescriptive — they serve as a sanity check: if your selections match a known archetype, you're in well-charted territory.

Archetype	Combination	Characteristics
Cloud-Native Ideal	LS1 + VT1 + RT1	Greenfield, continuous deployment, canary. Full GitOps, HPA, service mesh. Highest investment, highest payoff.
Enterprise Standard	LS2 + VT3 + RT3	Active development, sprint-aligned releases, rolling updates. Balanced investment, proven patterns.
Controlled Evolution	LS2 + VT2 + RT2	Active development, deliberate releases, blue-green safety. Common in regulated environments.
Regulated Continuous	LS2 + VT1 + RT1	Continuous delivery with automated compliance gates. Velocity within regulatory bounds. Requires mature automation.
Pragmatic Maintenance	LS3 + VT3 + RT3/RT4	Maintenance mode, scheduled releases, minimal deployment sophistication. Low investment, adequate for purpose.
Sunset Minimal	LS4 + VT3 + RT4	Decommissioning app, scheduled patches only, direct deployment. Minimum viable operational investment.

Strategy Tensions

STRATEGY TENSION is Tier 3's failure mode, but softer than Tiers 1–2. Tensions arise when selected choices create friction with each other or with upstream outputs. They require conscious trade-off acceptance, not a hard stop.

Tension	Why It Conflicts	Resolution
VT1 + RT4	Deploying continuously with maximum blast radius	Upgrade to RT3 minimum; continuous + no risk management is reckless
LS4 + VT1	Investing in continuous deployment for a sunset app	Downgrade to VT3; CD investment is waste for a dying app
VT4 + RT1	Coordinated releases with canary is extremely complex	Degrade to RT2 for coordinated releases, or break coordination
LS3 + RT1	Canary infrastructure for a maintenance-mode app	RT3 is sufficient; canary investment unjustified
LS1 + VT4	New app already needing coordinated releases	Question if truly greenfield, or new service in coupled architecture (Tier 1)

Tension resolution order: (1) Adjust the selection in the tensioned choice area. (2) Invest in capability to support the desired combination. (3) Accept with documented trade-offs. (4) Escalate to Tier 2 if the tension reveals the constrained path is insufficient.

Failure Modes Across Tiers

Each tier has a distinct failure mode, calibrated to the tier's nature. Severity decreases from Tier 1 to Tier 3.

Failure Mode	Tier	Meaning	Resolution
BLOCKED	Tier 1	Any dimension scores 0. Architectural issue prevents containerization.	Fix the blocker: consolidate distributed monolith, fix data ownership, analyze hidden coupling, evaluate lifecycle modifiability.
CONSTRAINT DEADLOCK	Tier 2	No viable path survives constraint filtering (e.g. HIGH fitness + no containers available).	(1) Invest to change the hardest constraint, (2) accept suboptimal path with documented trade-offs, (3) return to Tier 1.
STRATEGY TENSION	Tier 3	Selected choices create friction but are not impossible. Soft warning, not a hard stop.	(1) Adjust selection, (2) invest in capability, (3) accept with documented trade-offs, (4) escalate to Tier 2.

What This Framework Does NOT Decide

The framework produces an Execution Strategy Profile that makes implementation decisions tractable, but it defers certain choices to the concrete implementation phase:

Deferred To Implementation	Informed By
Specific tooling (ArgoCD vs Flux, Jenkins vs GitHub Actions)	Velocity Target, Org Topology, Infrastructure Availability
Environment topology (count, naming, promotion order)	Velocity + Compliance + Org Topology
Testing strategy (unit/integration/e2e ratio)	Velocity Target + Risk Tolerance
Monitoring/observability stack	Risk Tolerance (RT1/RT2 require it) + Infrastructure Availability