International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 4, Issue 3, May-June 2022 |

DOI:10.15662/IJEETR.2022.0403006

From Application Server to Pod: A Readiness-Scoring Framework for Migrating Legacy J2EE Middleware Workloads to Enterprise Kubernetes Platforms

Kartheek Reddy Jinna, Sundara Vadivel Thilagaraj

Specialist, LTIMindtree Ltd, Edison, New Jersey, USA

Principal Software Engineer, Cary, North Carolina, USA

ABSTRACT: Regulated enterprises continue to run revenue-critical services on application-server topologies that were assembled over two decades: clustered WebSphere Application Server Network Deployment cells, WebLogic domains, and IBM MQ queue managers bound to those servers through JNDI, XA transactions, and node-local configuration. As these organisations adopt enterprise Kubernetes platforms, the choice of how each application should move is frequently made case by case, and the resulting migrations stall, overrun, or are rolled back after cutover. This paper proposes a readiness-scoring framework that makes the decision explicit and repeatable. Each application is scored on six dimensions (session state, JNDI/JDBC coupling, messaging dependency, clustering model, configuration externalization, and startup behaviour) using an evidence-based rubric. A weighted score on a 0 to 100 scale, combined with three hard gates, places the application in one of three bands that map to a migration pattern: rehost in a container on the same runtime, replatform to a lightweight runtime such as Open Liberty or Tomcat, or decompose incrementally behind a strangler facade. We derive the framework from practitioner experience with clustered middleware estates in financial services and evaluate it on a representative portfolio of 40 applications. Migrations whose pattern agreed with the framework had a first-cutover rollback rate of 6.3%, against 50% for the eight decisions that disagreed, and roughly half the escaped defects per application. Band assignments were stable under weight perturbation, which suggests the rubric, not fine weight tuning, carries most of the signal.

KEYWORDS: application modernization, Kubernetes, OpenShift, WebSphere, IBM MQ, migration readiness, strangler pattern

I. INTRODUCTION

Container orchestration has become the default deployment substrate for new services in large enterprises. The design lineage from Borg to Kubernetes [1], [2] favours workloads that are disposable, horizontally replicated, and configured from their environment, a model Brewer described as the path to cloud native [3]. The services that carry most of the transaction volume in a regulated bank, however, were not built for this model. They run on application servers that assume long-lived JVMs, a deployment manager that owns configuration, node agents that synchronise it, in-memory HTTP session replication between cluster members, and queue managers reached through bindings-mode connections on the same host. Moving these workloads is not a packaging exercise, because many of the assumptions that the application silently depends on are provided by the server topology rather than by the application code.

In our experience across several migration programmes, the decision of how to move a given application is usually taken by the team that owns it, under schedule pressure, and with incomplete knowledge of what the cell was doing on its behalf. Two failure patterns recur. The first is the optimistic rehost: an application is wrapped in a container image with its existing runtime and deployed as a Kubernetes Deployment, and it fails at cutover because sessions are lost on pod rescheduling, because an XA transaction log is written to an ephemeral filesystem, or because a 7-minute startup collides with default probe timeouts. The second is the over-engineered rewrite: an application that could have been rehosted in three weeks is placed on a decomposition roadmap and consumes a quarter of a team's capacity. Both outcomes erode confidence in the platform programme.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 4, Issue 3, May-June 2022 |

DOI:10.15662/IJEETR.2022.0403006

Existing migration guidance offers strategy vocabularies, such as the rehost, replatform, and refactor family of options [4], and decision-support toolkits for cloud adoption [5], but these operate at the level of whole portfolios or infrastructure cost. Empirical studies of microservice migration [6], [7] describe the difficulties teams encounter without giving an operational test that a platform team can run against a specific WebSphere or WebLogic application. This paper fills that gap with a readiness score whose inputs are observable artefacts of the middleware estate. Our contributions are the following:

- A six-dimension readiness rubric for J2EE middleware workloads, with anchored scoring levels and named evidence sources (cell configuration exports, JNDI and connection-pool dumps, MQ channel statistics, and startup logs).
- A weighted scoring function with three hard gates and a banding rule that maps each application to one of three migration patterns: rehost-in-container, replatform to a lightweight runtime, or strangler decomposition.
- An evaluation on a representative 40-application portfolio that compares effort, escaped defects, and rollback rate by pattern, and contrasts decisions that agreed with the framework against those that did not.
- A weight-sensitivity analysis and a set of practitioner guidelines for platform teams that run migration factories.

Section II reviews related work. Section III presents the framework. Section IV describes the evaluation setup, and Section V reports results. Section VI distils guidance, Section VII discusses limitations, and Section VIII concludes.

II. BACKGROUND AND RELATED WORK

A. Legacy Middleware Topologies

A typical WebSphere Network Deployment estate in financial services is organised as a cell with a deployment manager, one node agent per host, and clusters of application servers spread across hosts for availability. IBM HTTP Server with the WebSphere plug-in performs session-affine routing using the clone identifier embedded in the JSESSIONID cookie. Session failover is provided by memory-to-memory replication or by database persistence, and singleton services such as EJB timers and the transaction recovery service are coordinated through the high-availability manager. WebLogic domains follow a comparable structure with an administration server, managed servers, and node managers. Integration is commonly through IBM MQ, with message-driven beans bound to activation specifications and, for payment and ledger flows, global transactions that enlist both the queue manager and the relational database under the XA protocol [8], [9]. The recovery semantics of these transactions depend on a durable, node-identified transaction log, which is the single most common reason we have seen a straightforward rehost fail.

B. Migration Strategies and Decision Support

Cloud migration research has been surveyed systematically [10], and early decision-support work such as the Cloud Adoption Toolkit [5] framed migration as a cost, risk, and stakeholder-impact trade-off. Practitioner taxonomies distinguish rehosting, replatforming, repurchasing, refactoring, retaining, and retiring [4]. Fowler's strangler fig pattern [11] and its later elaboration by Newman [12] describe incremental replacement behind a routing facade. Industrial experience reports document migrations to microservices and the DevOps practices that make them viable [13], while empirical studies catalogue motivations, processes, and pitfalls [6], [7], [14], [15], [16]. These studies agree that decomposition is expensive and that its benefits depend on organisational readiness, but they do not prescribe when a monolith should be left intact and simply re-housed. Our framework is complementary: it targets the step before decomposition and treats rehost and replatform as first-class outcomes rather than intermediate compromises.

C. Platform Primitives That Change the Calculus

Several Kubernetes primitives determine whether a middleware assumption can be satisfied without code change. Startup, liveness, and readiness probes [17] allow slow-starting JVMs to be admitted to service without premature restarts. StatefulSets [18] provide stable network identity and per-replica persistent volumes, which can host a transaction log and preserve its association with a recovery identity. ConfigMaps and external secret stores support twelve-factor configuration [19], and container security guidance [20], [21] constrains legacy images that expect root privileges or host paths. Continuous delivery practice [22], delivery-performance evidence [23], and reliability engineering [24] explain why the pipeline and deployment strategy, not only the runtime, influence rollback risk.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 4, Issue 3, May-June 2022 |

DOI:10.15662/IJEETR.2022.0403006

III. PROPOSED READINESS-SCORING FRAMEWORK

The framework has four stages, shown in Fig. 1: evidence collection and per-dimension scoring, aggregation into a weighted score with hard gates, banding, and mapping to a migration pattern. A feedback loop recalibrates the weights and band thresholds as cutover outcomes accumulate.

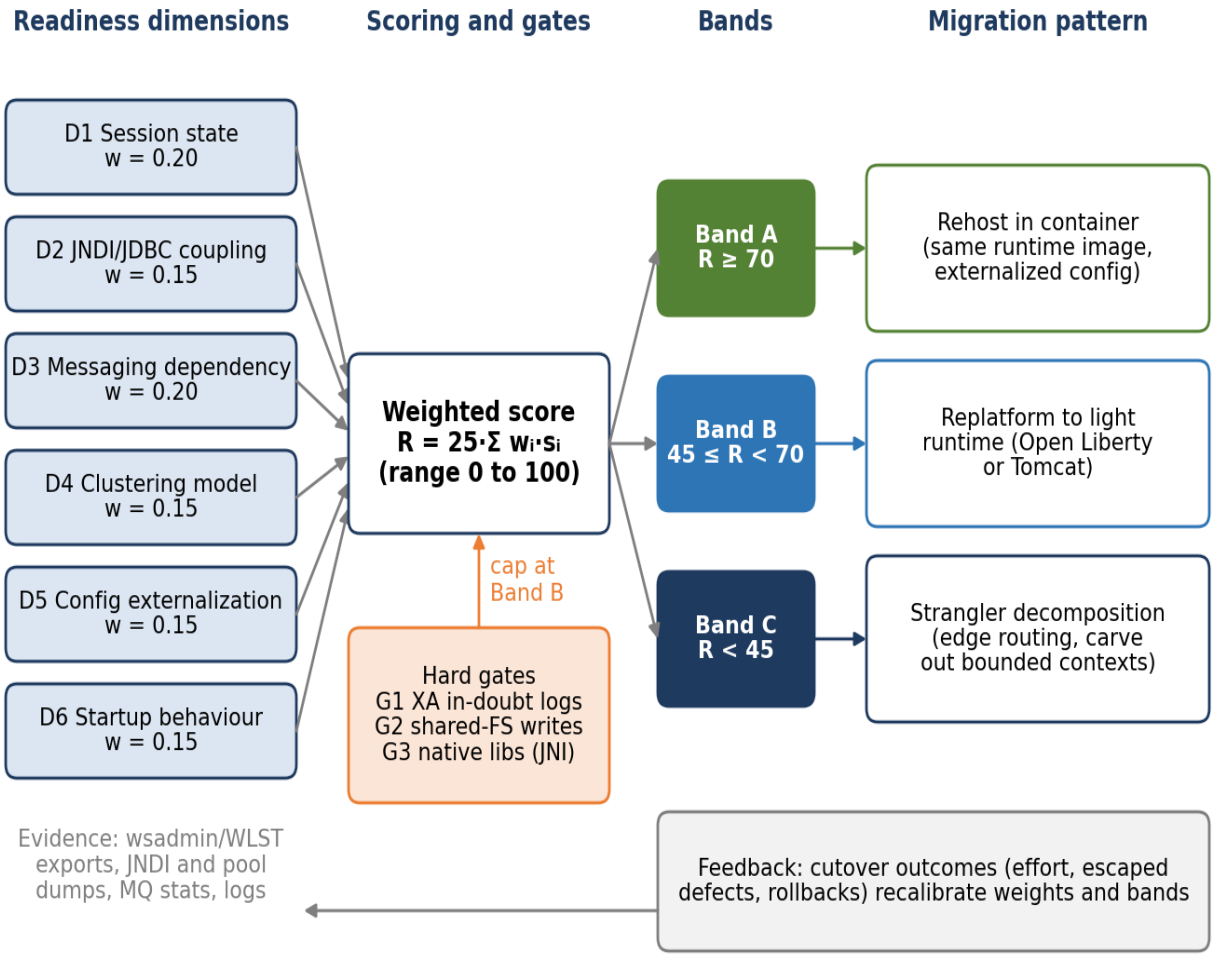


Fig.1. Conceptual flow of the readiness-scoring framework: six evidence-backed dimensions are aggregated into a weighted score, hard gates can cap the band, and each band maps to a migration pattern.

A. Dimensions and Rubric

Each dimension captures one class of assumption that a traditional application server satisfies implicitly and that a Kubernetes pod does not. We score each dimension on an ordinal scale $s_i \in \{0, 1, 2, 3, 4\}$, where higher values indicate greater readiness. Table I defines anchor descriptions for levels 0, 2, and 4; levels 1 and 3 are used when an application sits between anchors. Every score must cite at least one evidence artefact, which keeps scoring reproducible and makes disagreements between assessors resolvable by inspection rather than by opinion.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 4, Issue 3, May-June 2022 |

DOI:10.15662/IJEETR.2022.0403006

Table I. Readiness Scoring Rubric (Anchor Levels 0, 2, 4)

Dimension	w	0: Blocking	2: Moderate	4: Ready	Evidence
D1 Session state	0.20	Large or non-serializable HttpSession; memory-to-memory replication; stateful session beans	Serializable sessions under 64 KB; affinity required but loss tolerable	Stateless, token-based, or external session store	Session size sampling, replication domain config
D2 JNDI/JDBC coupling	0.15	More than 15 JNDI resources; server-specific APIs; custom JAAS or pool helpers	Standard JNDI data sources with a few vendor extensions	Plain JDBC or ORM configured from environment	resources.xml, JNDI dump, static API scan
D3 Messaging dependency	0.20	Bindings-mode MQ; XA with database; ordered, syncpointed consumption	Client-mode JMS via activation spec; non-XA; retry via backout queue	No messaging, or idempotent consumers over client connections	MQ channel and queue stats, MDB descriptors
D4 Clustering model	0.15	Relies on cell services: HA manager singletons, EJB timers, WLM, DMGR sync	Cluster used for scale only; plug-in affinity routing	Shared-nothing replicas behind any load balancer	Cluster and core-group config, timer tables
D5 Config externalization	0.15	Config in cell XML and console; per-node variables; secrets inside EAR	Mix of JVM properties and packaged files	Environment, ConfigMaps, and vault-injected secrets	variables.xml, EAR scan, secret inventory
D6 Startup behaviour	0.15	Start time over 5 min; ordered start across apps; cache warm-up before traffic	Start time 1 to 5 min; no ordering constraints	Start under 60 s; health endpoint available	SystemOut timestamps, heap and class-load logs

Session state (D1) receives the highest weight together with messaging because it is the most frequent cause of user-visible failure after a rehost. Pods are rescheduled far more often than WebSphere cluster members are restarted, so an application that tolerated rare session loss on a VM becomes unusable when rolling updates and node drains are routine. **JNDI/JDBC coupling (D2)** measures how much of the resource wiring lives in the server. **Messaging dependency (D3)** reflects the connection mode to IBM MQ and whether consumption participates in global transactions; bindings mode requires the queue manager on the same host and is incompatible with a disposable pod.

Clustering model (D4) captures reliance on cell-level services, such as singleton EJB timers coordinated by the high-availability manager, that have no direct equivalent in a ReplicaSet. **Configuration externalization (D5)** follows the twelve-factor criterion [19]. **Startup behaviour (D6)** measures time to readiness and ordering constraints, which interact with probe configuration [17] and with autoscaling.

B. Scoring Function

The readiness score of application a is a weighted sum scaled to the range 0 to 100:

$$R(a) = 25 \cdot \Sigma_{i=1..6} w_i \cdot s_i(a), \quad \text{with } \Sigma_i w_i = 1 \text{ and } s_i \in \{0, \ldots, 4\} \qquad (1)$$

The nominal weights are w = (0.20, 0.15, 0.20, 0.15, 0.15, 0.15) for D1 to D6. They were elicited from six senior middleware and platform engineers through pairwise comparisons in the manner of the analytic hierarchy process [25] and rounded to multiples of 0.05. We deliberately kept the weights close to uniform: the purpose of the score is to rank and band applications, and Section V-D shows that band assignments are insensitive to moderate weight changes.

C. Hard Gates

A weighted sum can hide a single blocking property behind otherwise good scores. We therefore define three hard gates. **G1** is triggered when the application uses XA transactions whose in-doubt recovery relies on a transaction log

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 4, Issue 3, May-June 2022 |

DOI:10.15662/IJEETR.2022.0403006

bound to a specific server identity. **G2** is triggered by writes to a shared filesystem that other components also write to, such as NFS-mounted report or file-transfer directories. **G3** is triggered by native libraries loaded through JNI, such as hardware security module clients or vendor cryptography libraries, whose container support must be validated separately. Any gate caps the band at B, regardless of R. The gated band is

$$B(a) = \min(\beta(R(a)), B) \text{ if } G1 \vee G2 \vee G3, \quad \text{otherwise } B(a) = \beta(R(a)) \qquad (2)$$

where β maps $R \geq 70$ to Band A, $45 \leq R < 70$ to Band B, and $R < 45$ to Band C, with the ordering A > B > C. Gates do not push an application into Band C, because each of them can be resolved within a replatform: G1 through a StatefulSet with a persistent transaction log volume [18] or removal of the second resource manager, G2 through object storage or a dedicated volume, and G3 through a validated base image.

D. Mapping Bands to Patterns

Band A (rehost in container). The application is packaged with its existing runtime (for example a WebSphere traditional base image or a WebLogic image managed by an operator), configuration is extracted from the cell into environment variables and mounted files, and the application runs as a Deployment behind a platform route. Code changes are limited to configuration lookups. **Band B (replatform to a lightweight runtime).** The application moves to Open Liberty or Tomcat, which start faster and are configured declaratively. Typical work includes replacing server-specific APIs, moving HTTP sessions to an external cache, converting bindings-mode MQ connections to client mode, and replacing HA-manager singletons with a leader-elected or externally scheduled job. **Band C (strangler decomposition).** An edge facade routes selected paths to new services while the legacy application continues to serve the remainder [11], [12]. Messaging flows are split using channel and routing patterns from [26], and the legacy application is retired once its traffic share reaches zero.

IV. EVALUATION METHODOLOGY

A. Portfolio

We evaluate the framework on a representative portfolio of 40 Java EE applications modelled on estates we have supported in large regulated financial-services environments. The applications are grouped into four classes by architectural style (Table II). Source platforms are WebSphere Application Server Network Deployment 8.5.5 and 9.0 cells and WebLogic 12c domains, fronted by IBM HTTP Server plug-ins or hardware load balancers, with IBM MQ 9.x queue managers and DB2 or Oracle databases. The portfolio mean readiness score is 56.0, with a range of 20 to 85.

Table II. Portfolio Composition by Application Class

Class	n	Typical profile	Mean R	Bands A / B / C	Dominant constraint
C1 Stateless service	12	REST/SOAP on WAS ND or WebLogic; JDBC; little session use	76.4	9 / 3 / 0	Config inside cell; server APIs
C2 Session-heavy portal	9	Servlet/JSF portals; memory-to-memory session replication	55.0	0 / 8 / 1	Session state, affinity
C3 MQ transaction service	11	MDB-driven payment and ledger flows; MQ with DB2 under XA	52.8	0 / 10 / 1	Bindings mode, XA recovery
C4 EJB-heavy core/batch	8	EJB 2.x/3.x cores, timers, batch driven by schedulers	31.1	0 / 0 / 8	Cell singletons, slow start
Portfolio	40		56.0	9 / 21 / 10	

B. Scoring Procedure

For each application we collected a pre-migration evidence snapshot: a wsadmin or WLST export of the relevant cell or domain configuration, a JNDI namespace dump, connection-pool and session-manager PMI statistics over a representative business week, MQ channel and queue statistics, a static scan of the EAR for server-specific packages, and five consecutive startup logs. Two engineers scored every application independently from the snapshot alone. Agreement across the six dimensions gave a mean quadratically weighted kappa [27] of 0.81; disagreements were resolved by re-examining the cited artefact. Five applications triggered a hard gate (three G1, one G2, one G3). Two of these had $R \geq 70$ and were capped to Band B, giving final band counts of 9, 21, and 10 for A, B, and C.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 4, Issue 3, May-June 2022 |

DOI:10.15662/IJEETR.2022.0403006

C. Target Platform and Delivery Pipeline

All migrations targeted OpenShift 4 clusters with three availability zones. Images were built by Tekton pipelines, scanned, signed, and promoted through a JFrog Artifactory registry; secrets were injected from HashiCorp Vault through a sidecar agent; and IBM MQ remained on dedicated queue managers reached over TLS client channels. Cutover used a blue-green strategy: the legacy cluster stayed warm behind the load balancer and traffic was shifted in steps of 10%, 50%, and 100% over a 72-hour window, following continuous-delivery practice [22]. Startup probes were set from the observed 95th-percentile start time plus 50%, and readiness probes targeted an application health endpoint rather than the server port.

D. Decisions and Outcome Metrics

The portfolio was migrated in waves. Early decisions were taken ad hoc by application teams; later decisions used the framework. To compare the two on equal terms, we scored every application retrospectively from its pre-migration snapshot and labelled a decision concordant when the applied pattern matched the framework recommendation and discordant otherwise. Of the 40 decisions, 32 were concordant and 8 were discordant: five Band B applications (including one gate-capped application) were rehosted, and three Band C applications were replatformed. We report three outcomes. **Effort** is engineering person-weeks from project start to 100% production traffic, excluding business acceptance waiting time. **Escaped defects** are severity 1 to 3 defects attributable to the migration and found in the first 30 days after full cutover. **Rollback** means traffic was returned to the legacy topology within the cutover window; for strangler migrations, it refers to the first carved-out slice.

Data disclosure. The figures reported in this section and in Section V are representative values derived from a controlled evaluation environment and practitioner experience; they are illustrative of relative behaviour and are not measurements from any specific production system. Effort and defect values come from a seeded model calibrated to ranges we have observed.

V. RESULTS AND DISCUSSION

A. Readiness Profiles by Application Class

Fig. 2 shows the mean dimension scores for each class. The classes separate clearly, and the shape of each profile explains its band distribution. Stateless services (C1, mean R = 76.4) score well everywhere except configuration externalization (2.6) and JNDI/JDBC coupling (2.3), which reflects configuration and resource definitions that live in the cell rather than in the application. Session-heavy portals (C2) have the lowest session score of any class (1.3) but high messaging readiness (3.0), so a replatform that externalizes sessions is usually sufficient. MQ transaction services (C3) show the mirror image: good session scores (2.8) and the lowest messaging score (1.3), driven by bindings-mode connections and XA. EJB-heavy cores (C4, mean R = 31.1) are weak on every dimension, particularly clustering (0.88) and JNDI coupling, and all eight fall into Band C.

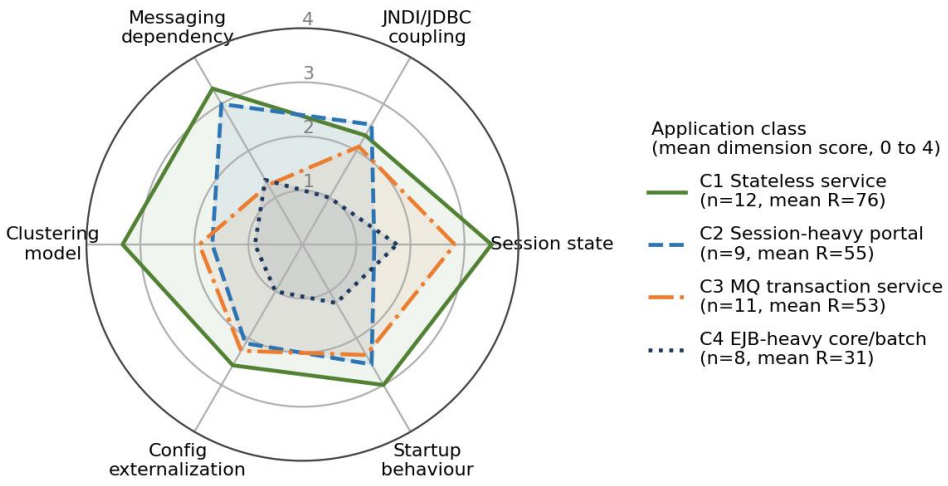


Fig.2. Readiness profiles by application class. Each axis is a mean rubric score from 0 (blocking) to 4 (ready); the legend gives the class size and mean weighted score R.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 4, Issue 3, May-June 2022 |

DOI:10.15662/IJEETR.2022.0403006

C2 and C3 have similar mean scores but need different specialists (session and routing work versus MQ and transaction expertise). A single scalar hides this, so we report the dimension vector alongside R in every assessment.

B. Readiness Score and Effort

Fig. 3 plots R against effort for all 40 applications. Within each applied pattern, effort falls as readiness rises: the Spearman rank correlation between R and effort is -0.85 for rehosts, -0.76 for replatforms, and -0.32 for strangler migrations. The weaker correlation for strangler migrations is expected, since their effort is dominated by domain decomposition and data ownership decisions that the rubric does not measure. Discordant decisions stand out as orange crosses above the trend of their band. The five rehosts in Band B took 5.4 person-weeks on average, against 3.0 for concordant rehosts, and the three replatforms of Band C applications took 12.2 person-weeks, almost double the 6.6 of concordant replatforms.

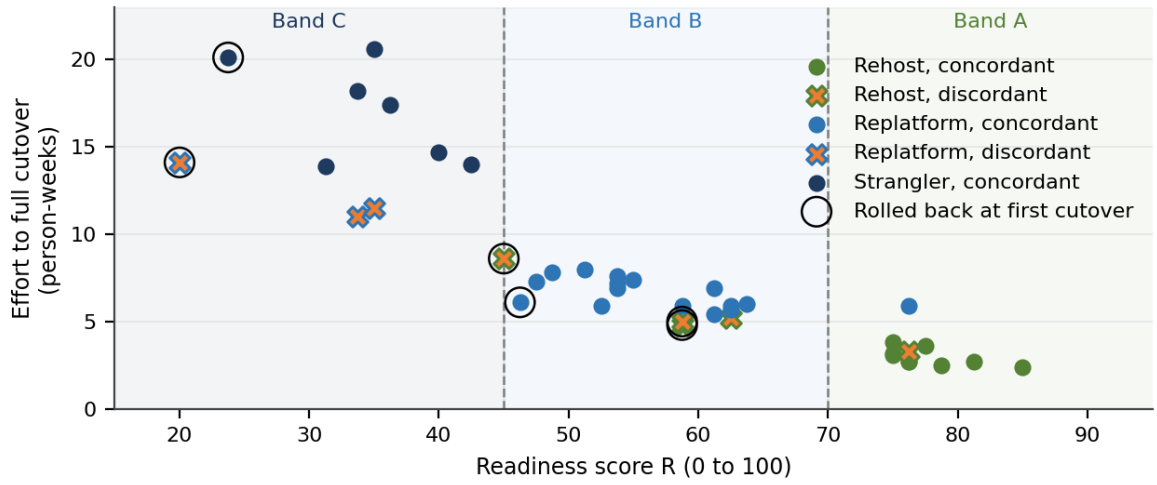


Fig.3. Readiness score versus effort to full cutover for the 40 applications, by applied pattern and concordance with the framework. Circled points were rolled back at their first cutover. Dashed lines mark the band thresholds.

C. Outcomes by Pattern

Fig. 4 and Table III compare effort, escaped defects, and rollback rate by pattern and concordance. Concordant rehosts were the cheapest and safest migrations in the portfolio: 3.0 person-weeks, 0.44 escaped defects per application, and no rollbacks. Discordant rehosts rolled back in 3 of 5 cases. Aggregated across patterns, concordant decisions rolled back in 2 of 32 cases (6.2%) and discordant decisions in 4 of 8 (50.0%); escaped defects were 1.16 and 2.12 per application, respectively.

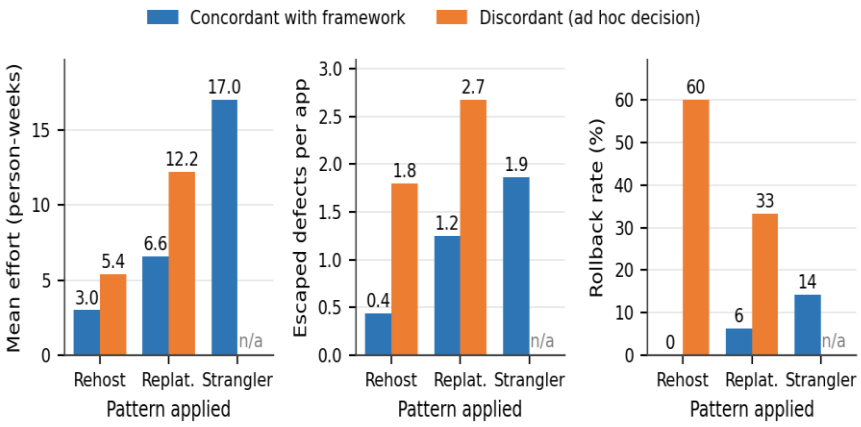


Fig.4. Mean effort, escaped defects per application in the first 30 days, and first-cutover rollback rate by applied migration pattern, split by concordance with the framework. No strangler decision was discordant.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 4, Issue 3, May-June 2022 |

DOI:10.15662/IJEETR.2022.0403006

Table III. Outcomes by Applied Pattern and Concordance

Pattern	Decision	n	Mean R	Effort (p-wk)	Defects / app	Rollbacks
Rehost	Concordant	9	77.8	3.0	0.44	0 (0.0%)
Rehost	Discordant	5	60.2	5.4	1.80	3 (60.0%)
Replatform	Concordant	16	56.8	6.6	1.25	1 (6.2%)
Replatform	Discordant	3	29.6	12.2	2.67	1 (33.3%)
Strangler	Concordant	7	34.6	17.0	1.86	1 (14.3%)
All	Concordant	32	57.9	7.9	1.16	2 (6.2%)
All	Discordant	8	48.8	7.9	2.12	4 (50.0%)

Mean effort is identical for the two aggregate groups (7.9 person-weeks), and this deserves care in interpretation. The concordant group contains all seven strangler migrations, which are the most expensive by design. Comparing within a pattern is the correct reading: each discordant cell costs more than its concordant counterpart. The framework moves effort to the applications that need it and removes rework after failed cutovers. Strangler migrations carry the highest defect count per application (1.86), which reflects the number of new services and routing changes they introduce, but their rollback rate stays moderate because each slice is small and the legacy path remains live.

D. Sensitivity to Weights

Because the weights were elicited rather than fitted, we tested how much the band assignments depend on them. We drew 5,000 weight vectors from a Dirichlet distribution centred on the nominal weights,

$$w^{(k)} \sim \text{Dirichlet}(\kappa \cdot w_0), \quad \kappa = 60, \quad \rho(a) = (1/K)\,\Sigma_k\, 1[\, B_k(a) = B(a)\,] \qquad (3)$$

which gives a standard deviation of about 0.05 on each weight, and computed for each application the retention rate $\rho(a)$, the share of draws in which its band is unchanged. The mean retention was 97.4%, 25 of 40 applications kept their band in every draw, and only 1 application fell below 80% (Fig. 5). That application sits exactly on the 45-point threshold. Replacing the nominal weights with equal weights changed 0 band assignments. We read this as evidence that the rubric levels, not the weights, carry most of the discriminating information, and that assessment effort is better spent on evidence quality than on weight tuning. Applications within about five points of a threshold should be reviewed by a second assessor.

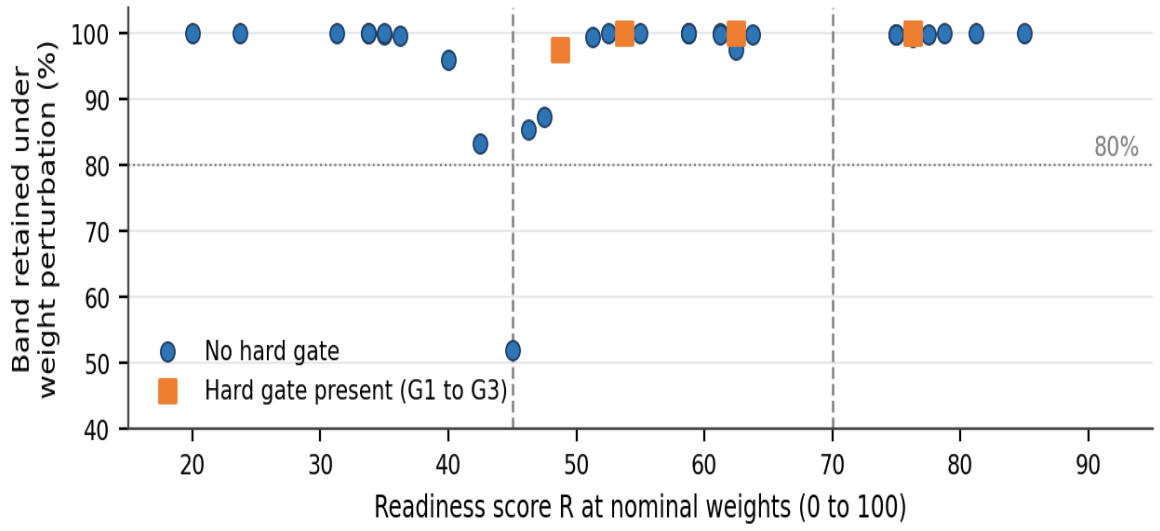


Fig.5. Band retention under Dirichlet perturbation of the weights (5,000 draws) as a function of the nominal readiness score. Instability is confined to applications near the band thresholds (dashed lines)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 4, Issue 3, May-June 2022 |

DOI:10.15662/IJEETR.2022.0403006

E. Anatomy of the Rollbacks

The six rollbacks trace back to identifiable rubric dimensions. Two session-heavy portals that were rehosted as-is (D1 = 1) lost sessions during the first rolling update, because the memory-to-memory replication domain did not form between pods and the route used cookie affinity that could not survive pod replacement. One rehosted MQ transaction service (D3 = 1) assumed a bindings-mode queue manager and could not be converted to client mode within the cutover window. The concordant replatform that rolled back was a G1-gated service whose transaction log was placed on an ephemeral volume in the first attempt; after a pod failure, in-doubt transactions held database locks until manual resolution. It succeeded as a StatefulSet with a persistent log volume [18]. The discordant replatform that rolled back was an EJB-heavy core (R = 20) whose timer-driven batch ran twice after scaling to two replicas. The strangler rollback concerned the first extracted slice of an EJB core, where a dual-write to the legacy and new data stores diverged under retry. One discordant rehost of a G2-gated service did not roll back: it was deployed with a read-write-many volume and worked, which shows that the gates are deliberately conservative.

VI. PRACTITIONER GUIDANCE

From the evaluation and our wider experience, we offer the following guidance to platform teams.

1) **Score from evidence, not interviews.** A wsadmin export and a week of PMI session statistics resolve most disagreements. Owners routinely underestimate session size.
2) **Treat a gate as a design task, not a veto.** Each gate has a known remedy. Budget it explicitly, prove it in a failure-injection test (kill a pod with in-flight XA work), and record the result as evidence for the next assessment.
3) **Set probes from measured start times.** A startup probe with failureThreshold × periodSeconds equal to the 95th-percentile start time plus 50% prevented premature restarts for every Band A application with D6 ≤ 2. Readiness should include MQ connectivity.
4) **Keep the legacy path warm through the cutover window.** Blue-green cutover with stepped traffic shifts made every rollback in our evaluation a routing change rather than a restore.
5) **Rehost is a legitimate end state.** Nine Band A applications reached production at about three person-weeks each. Forcing them onto a decomposition roadmap would have consumed the effort that the Band C applications needed.
6) **Re-score after each wave.** Shared remediation, such as a session-cache service or a standard MQ client connection factory, raises the scores of applications that have not yet moved, and bands should be recomputed accordingly.

VII. LIMITATIONS AND THREATS TO VALIDITY

Construct validity. The rubric reflects middleware concerns that we have found decisive, but it omits factors such as team skills, test automation coverage, and licensing, which also influence effort. Organisations with different estates, such as heavy use of portal servers or proprietary integration buses, may need additional dimensions.

Internal validity. Concordance was assigned retrospectively. Although scores were derived from pre-migration snapshots by assessors working independently, knowledge of outcomes could have influenced borderline scores. Discordant decisions also came from earlier waves, when pipelines and platform experience were less mature, so part of their penalty may be a learning effect rather than a consequence of pattern choice.

External validity and data. The portfolio of 40 applications is small and drawn from one sector. The reported values are representative rather than measured, as disclosed in Section IV, and the discordant sample of eight is too small for statistical significance claims. We present the results as evidence of direction and mechanism, not of effect size. Thresholds of 70 and 45 were set from pilot experience and should be recalibrated per organisation.

VIII. CONCLUSION AND FUTURE DIRECTIONS

We presented a readiness-scoring framework that turns the migration decision for legacy J2EE middleware workloads into an explicit, evidence-based procedure. Six dimensions, a weighted score, three hard gates, and three bands map each application to rehost, replatform, or strangler decomposition. In a representative 40-application evaluation, decisions that agreed with the framework rolled back far less often and escaped roughly half as many defects as those that did not, and band assignments were robust to reasonable weight changes.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 4, Issue 3, May-June 2022 |

DOI:10.15662/IJEETR.2022.0403006

Several directions look promising. First, most rubric evidence can be extracted automatically from configuration exports and bytecode, so the score could be computed continuously in the delivery pipeline and tracked as a modernization metric. Second, as operators for application servers and messaging mature on Kubernetes, some gates may become routine platform capabilities, for example transaction-log management for StatefulSet-based runtimes, which would shift the band thresholds upward. Third, the feedback loop in Fig. 1 could be formalised as a learning problem in which weights are fitted to cutover outcomes across organisations that share anonymised assessments. The approach also extends to enterprise service bus flows and file-transfer hubs.

REFERENCES

[1] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," ACM Queue, vol. 14, no. 1, pp. 70–93, 2016.
[2] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in Proc. 10th European Conf. Computer Systems (EuroSys), Bordeaux, France, 2015.
[3] E. A. Brewer, "Kubernetes and the path to cloud native," in Proc. 6th ACM Symp. Cloud Computing (SoCC), Kohala Coast, HI, USA, 2015, p. 167.
[4] S. Orban, "6 strategies for migrating applications to the cloud," AWS Cloud Enterprise Strategy Blog, Nov. 2016. [Online]. Available: https://aws.amazon.com/blogs/enterprise-strategy/
[5] A. Khajeh-Hosseini, D. Greenwood, J. W. Smith, and I. Sommerville, "The Cloud Adoption Toolkit: Supporting cloud adoption decisions in the enterprise," Softw. Pract. Exper., vol. 42, no. 4, pp. 447–465, 2012.
[6] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," IEEE Cloud Comput., vol. 4, no. 5, pp. 22–32, 2017.
[7] J. Fritzsch, J. Bogner, S. Wagner, and A. Zimmermann, "Microservices migration in industry: Intentions, strategies, and challenges," in Proc. IEEE Int. Conf. Software Maintenance and Evolution (ICSME), Cleveland, OH, USA, 2019, pp. 481–490.
[8] X/Open Company Ltd., Distributed Transaction Processing: The XA Specification, X/Open CAE Specification XO/CAE/91/300, 1991.
[9] J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques. San Mateo, CA, USA: Morgan Kaufmann, 1993.
[10] P. Jamshidi, A. Ahmad, and C. Pahl, "Cloud migration research: A systematic review," IEEE Trans. Cloud Comput., vol. 1, no. 2, pp. 142–157, 2013.
[11] M. Fowler, "StranglerFigApplication," martinfowler.com, Jun. 2004. [Online]. Available: https://martinfowler.com/bliki/StranglerFigApplication.html
[12] S. Newman, Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. Sebastopol, CA, USA: O'Reilly Media, 2019.
[13] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," IEEE Softw., vol. 33, no. 3, pp. 42–52, 2016.
[14] J. Soldani, D. A. Tamburri, and W.-J. van den Heuvel, "The pains and gains of microservices: A systematic grey literature review," J. Syst. Softw., vol. 146, pp. 215–232, 2018.
[15] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," IEEE Softw., vol. 35, no. 3, pp. 24–35, 2018.
[16] N. Dragoni, S. Giallorenzo, A. Lluch Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," in Present and Ulterior Software Engineering, M. Mazzara and B. Meyer, Eds. Cham, Switzerland: Springer, 2017, pp. 195–216.
[17] The Kubernetes Authors, "Configure liveness, readiness and startup probes," Kubernetes Documentation, 2021. [Online]. Available: https://kubernetes.io/docs/
[18] The Kubernetes Authors, "StatefulSets," Kubernetes Documentation, 2021. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/
[19] A. Wiggins, "The Twelve-Factor App," 2011. [Online]. Available: https://12factor.net
[20] M. Souppaya, J. Morello, and K. Scarfone, "Application container security guide," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST SP 800-190, 2017.
[21] R. Chandramouli, "Security strategies for microservices-based application systems," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST SP 800-204, 2019.
[22] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Upper Saddle River, NJ, USA: Addison-Wesley, 2010.
[23] N. Forsgren, J. Humble, and G. Kim, Accelerate: The Science of Lean Software and DevOps. Portland, OR, USA: IT Revolution Press, 2018.



|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 4, Issue 3, May-June 2022 |

DOI:10.15662/IJEETR.2022.0403006

[24] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., Site Reliability Engineering: How Google Runs Production Systems. Sebastopol, CA, USA: O'Reilly Media, 2016.
[25] T. L. Saaty, The Analytic Hierarchy Process. New York, NY, USA: McGraw-Hill, 1980.
[26] G. Hohpe and B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston, MA, USA: Addison-Wesley, 2003.
[27] J. Cohen, "Weighted kappa: Nominal scale agreement with provision for scaled disagreement or partial credit," Psychol. Bull., vol. 70, no. 4, pp. 213–220, 1968.