Migrating Oracle Databases to Azure Cloud

A typed, signed-deliverable playbook for the enterprise-scale Oracle 19c → Azure migration

KinetiStack Press

KinetiStack LLC

2026-06-30

Chapter 1 — Automated Discovery and Schema Assessment

> Most Oracle-to-Azure migrations don't fail at the cutover. They fail months earlier, in a conference room, when a director asks *"what does it cost and what could break?"* and the only honest answer is *"we don't know yet."* This chapter exists so that question never goes unanswered again.

By the time you finish this chapter you will have a signed, machine-readable **Migration Assessment Bundle (MAB)** that converts your production Oracle 19c estate into a defensible scorecard — the same scorecard every downstream chapter in the data chain uses as its input contract.

The MAB is the first of **thirteen signed JSON deliverables** that thread through the book's 15 chapters. You will meet each one in the chapter that produces it — no need to memorize the chain now, since every artifact gets a full chapter when its turn comes. The shape that matters here is just *pipeline*: each downstream artifact takes the previous one's schema-validated JSON as input, and a CI exit code gates the next step. (Two chapters — Ch.3.5 Lab Setup and Ch.5 PL/SQL Refactoring — produce reference assets and code rather than signed JSON, so the chapter count and the deliverable count are not the same.) The book also threads a running pitfall catalog (P1–P28) anchored to numbered `> **Pitfall (P##)**` blockquotes throughout — P1 is the very next callout, in Ch.2.

1.1 Why Scripted Discovery Beats Interviews

Manual discovery — DBA interviews, tribal-knowledge spreadsheets, *"ask Jim, he wrote that package in 2009"* — has a 100% failure rate at the scale we are working with: 20 TB, heavy PL/SQL, RAC, TDE, FGA, partitioning, autonomous transactions, and an unknown number of database links pointing at systems no one quite remembers. Three reasons our discovery is **fully scripted** instead:

1. **Reproducibility.** A migration program runs for 12–24 months. The discovery you ran on day 5 has to be re-runnable on day 350, byte-identically, to validate "nothing new snuck in."
2. **Auditability.** Regulated industries will be asked, sometimes years later, *"how did you know the source did not use Database Vault?"* The MAB answers that with a signed artifact, not a meeting note.
3. **Hand-off integrity.** The MAB is the *contract* between this chapter and the architecture decision in Chapter 2 and the TCO model in Chapter 3. A structured artifact lets architects and finance run their own analyses without re-pulling data.

1.2 Two Constraints That Shape Every Script

Read-only, or it never runs. The assessment must be safe to point at production at 14:00 on a Wednesday. Every query in this chapter:

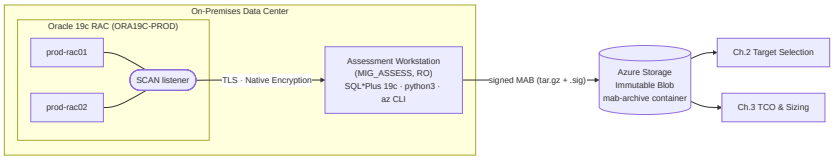
- Touches only `DBA_*` and `DBA_HIST_*` views (catalog reads, optimizer-cached, negligible latch impact).
- Runs as a dedicated, minimally-privileged account (`MIG_ASSESS`) — `SELECT_CATALOG_ROLE` only. No `SELECT ANY TABLE`, no `DBA`, never `SYSDBA`.
- Returns within minutes, not hours, on a 20 TB estate.

License-pack honesty. AWR (`DBA_HIST_*`) requires the **Oracle Diagnostics + Tuning Pack**. If your organization does not own those packs, running `02_workload_baseline.sql` is a *contract breach*. We declare the dependency in every file header, and we describe a Statspack-based fallback at the end of § 1.5.

> **Pro-Tip — Confirm the license posture in writing.** Before you run anything, ask your DBA team to confirm the Diagnostics + Tuning Pack license status in writing. Oracle LMS audits routinely cite unauthorized AWR queries as a finding, and *"the migration consultant ran it"* is not a defense. If the pack is not licensed, use the Statspack fallback.

1.3 The Discovery Topology

The assessment runs from a **dedicated hardened workstation** — never from a developer laptop, never on a RAC node directly, never as `SYS`.


On-Premises Data Center
Oracle 19c RAC (ORA19C-PROD)
prod-rac01
prod-rac02
SCAN listener
TLS · Native Encryption
Assessment Workstation
(MIG_ASSESS, RO)
SQL*Plus 19c · python3 ·
az CLI
signed MAB (tar.gz + .sig)
Azure Storage
Immutable Blob
mab-archive container
Ch.2 Target Selection
Ch.3 TCO & Sizing

1.3.1 The `MIG_ASSESS` Account

> **Pro-Tip — Do not run discovery as `SYS`.** The temptation is real (it bypasses every grant problem), but it (a) corrupts your audit trail because every query then shows up as `SYS` in unified audit, and (b) makes it impossible for a junior engineer to safely re-run the bundle later. Create `MIG_ASSESS` once. Grant the minimum. Document it in the runbook.

The one-time grant set, executed by a DBA on the source:

```
CREATE USER mig_assess IDENTIFIED BY "<replaced-by-wallet-entry>";
GRANT CREATE SESSION         TO mig_assess;
GRANT SELECT_CATALOG_ROLE    TO mig_assess;
GRANT SELECT ANY DICTIONARY  TO mig_assess;
-- AWR views inherit via SELECT_CATALOG_ROLE; we name DBA_HIST_SNAPSHOT
-- explicitly so the audit trail makes intent unambiguous.
GRANT SELECT ON dba_hist_snapshot TO mig_assess;
```

Then store the credential in an **Oracle Wallet** — not an environment variable, not a `.netrc`, not in your shell history. The orchestrator accepts a wallet alias like `/@ORA19CPROD_RO` and never touches the password directly.

1.3.2 The MAB Output Contract

Every script writes into a strict, machine-readable directory layout. **This is the contract Chapters 2 and 3 consume**, so the layout is locked:

```
/var/mab/<run-id>/
├── manifest.json                    env fingerprint + run metadata
├── feature_usage/
│   └── feature_score.csv            GREEN/AMBER/RED rubric output
├── workload_baseline/
│   ├── awr_iops.csv                 per-instance hourly IOPS
│   ├── awr_throughput.csv           MB/s read+write+redo
```

```
│   ├── awr_top_waits.csv              top-25 wait events by DB time
│   └── awr_cpu_pga.csv                CPU % + PGA MB by hour
├── blockers/
│   └── blockers_inventory.csv         banded blocker list
├── schema_complexity/
│   ├── plsql_inventory.csv            counts + LOC by owner/type
│   ├── plsql_hotspots.csv             top-50 by hotspot score
│   ├── partition_topology.csv         partitioning strategy per table
│   └── fga_policies.csv               FGA + Unified Audit inventory
├── summary/
│   └── mab_summary.json               rollup scorecard
├── checksums.sha256
└── mab.sig                            detached GPG signature
```

1.4 Script 01 — Feature Usage Audit

Full source: `scripts/01_feature_usage_audit.sql`

The script does three things:

1. **Surfaces every Oracle feature actually used**, by reading `DBA_FEATURE_USAGE_STATISTICS` and deduplicating to the latest sample per feature using `ROW_NUMBER()`.
2. **Scores each feature** against an in-script rubric encoded as a CTE.
3. **Orders the output by severity** so RED items are at the top of the CSV.

The rubric is the *opinionated* part of this chapter. It encodes our portability judgment as:

- **GREEN** — native equivalent on Azure SQL MI or PostgreSQL Flex with negligible refactor.
- **AMBER** — *different* implementation requiring code or design changes.
- **RED** — no equivalent. Re-architect (out-of-DB), pick Oracle Database@Azure (lift), or formally accept feature loss.

The rubric CTE excerpt:

```
WITH rubric (feature_pattern, target_score, target_notes) AS (
    SELECT 'Partitioning',                    'GREEN', 'Native on Azure SQL MI;
       ↪  emulated on PG via pg_partman'                    FROM DUAL UNION ALL
    SELECT 'Transparent Data Encryption',     'GREEN', 'TDE native on MI; PG uses
       ↪  storage-layer encryption via Azure'              FROM DUAL UNION ALL
    SELECT 'Real Application Clusters (RAC)','RED',    'No equivalent on MI/PG.
       ↪  Options: Oracle Database@Azure or Always-On AG' FROM DUAL UNION ALL
    SELECT 'Spatial and Graph',               'RED',    'Spatial: PostGIS on PG.
       ↪  Graph: out-of-DB (Azure Cosmos DB Gremlin)'       FROM DUAL UNION ALL
    SELECT 'Java',                            'RED',    'Java-in-DB not supported.
       ↪  Externalize to Azure Functions / App Service'    FROM DUAL UNION ALL
    SELECT 'Fine-Grained Auditing',           'AMBER', 'MI: SQL Audit. PG: pgAudit.
       ↪  Policy translation required (Ch.8)'              FROM DUAL UNION ALL
    -- (full rubric: 26 patterns — see source)
    SELECT 'Workspace Manager',               'RED',    'No equivalent. Re-implement
       ↪  temporal logic via system-versioned tables'    FROM DUAL
),
```

> **Deep Dive — Why a rubric CTE instead of a lookup table.** Embedding the rubric inline keeps the script self-contained, version-controllable, and inspectable in code review. A lookup table would tempt people to "tweak the scoring" out-of-band and break reproducibility. If a reader disagrees with our scoring of e.g. *Advanced Compression*, the diff is one PR — and that PR is the audit trail of *why* the score changed.

The join + ordering tail:

```
SELECT
    fu.feature_name,
    fu.currently_used,
    fu.detected_usages,
    fu.first_usage_date,
    fu.last_usage_date,
    fu.aux_count,
    NVL(r.target_score, 'AMBER')                                AS target_score,
```

```
    NVL(r.target_notes, 'No rubric entry - manual review required') AS target_notes
FROM   feature_used fu
LEFT   JOIN rubric r
       ON  UPPER(fu.feature_name) LIKE '%' || UPPER(r.feature_pattern) || '%'
WHERE  fu.rn = 1
ORDER  BY
    CASE NVL(r.target_score, 'AMBER')
        WHEN 'RED' THEN 1 WHEN 'AMBER' THEN 2 WHEN 'GREEN' THEN 3
    END,
    fu.feature_name;
```

The `LIKE` join is deliberately fuzzy — Oracle's feature names drift across versions (`"Real Application Clusters (RAC)"` vs `"RAC"` vs `"Real Application Clusters"`), and a pattern match insulates the rubric from those cosmetic shifts.

1.5 Script 02 — Workload Baseline

Full source: `scripts/02_workload_baseline.sql`

This is the script Chapter 3 reads to translate your real workload into Azure compute and storage requirements. It produces four CSVs, each a **time series** not a point-in-time snapshot.

1.5.1 Why hourly buckets over a 90-day window

- Weekly snapshots miss month-end batch peaks.
- Daily averages hide the 04:00 archive-maintenance spike that determines your peak IOPS sizing.
- Hourly × 90 days catches the seasonal envelope. P95 reads inform you; **P99.9 reads determine your Premium SSD v2 IOPS ceiling**.

1.5.2 Reading the four output files

File	What it tells you	What it sizes
`awr_iops.csv`	Avg read+write IOPS per hour per RAC instance	Azure managed-disk IOPS allocation (Premium SSD v2 / Ultra Disk)
`awr_throughput.csv`	MB/s read + write + redo MB/s	Disk throughput tier; redo MB/s sizes Ch.7 replication channel
`awr_top_waits.csv`	Top-25 wait events over 90 days	Tuning focus *before* migration — do not carry tuning debt
`awr_cpu_pga.csv`	CPU % busy + total PGA allocated	vCore count and memory tier on the target

1.5.3 The IOPS query, annotated

```
WITH snaps AS (
    SELECT snap_id, instance_number, dbid,
           begin_interval_time, end_interval_time,
           (CAST(end_interval_time AS DATE) - CAST(begin_interval_time AS DATE))
   ↪  * 86400 AS interval_secs
    FROM   dba_hist_snapshot
    WHERE  begin_interval_time >= SYSTIMESTAMP - INTERVAL '90' DAY
),
stat AS (
    SELECT s.snap_id, s.instance_number, s.dbid,
           SUM(CASE WHEN ss.stat_name = 'physical read total IO requests'  THEN
              ↪  ss.value END) AS read_reqs,
           SUM(CASE WHEN ss.stat_name = 'physical write total IO requests' THEN
              ↪  ss.value END) AS write_reqs
    FROM   dba_hist_sysstat ss
    JOIN   snaps s
```

```
            ON s.snap_id = ss.snap_id AND s.instance_number = ss.instance_number
               ↪  AND s.dbid = ss.dbid
    WHERE  ss.stat_name IN ('physical read total IO requests','physical write total
       ↪  IO requests')
    GROUP  BY s.snap_id, s.instance_number, s.dbid
),
delta AS (
    SELECT
        s.instance_number,
        TRUNC(CAST(s.end_interval_time AS DATE), 'HH24') AS hour_bucket,
        SUM(GREATEST(stat.read_reqs  - LAG(stat.read_reqs)
                     OVER (PARTITION BY s.instance_number ORDER BY s.snap_id), 0))
                        ↪  AS read_reqs_delta,
        SUM(GREATEST(stat.write_reqs - LAG(stat.write_reqs)
                     OVER (PARTITION BY s.instance_number ORDER BY s.snap_id), 0))
                        ↪  AS write_reqs_delta,
        SUM(s.interval_secs) AS secs
    FROM   snaps s
    JOIN   stat ON stat.snap_id = s.snap_id AND stat.instance_number =
       ↪  s.instance_number AND stat.dbid = s.dbid
    GROUP  BY s.instance_number, TRUNC(CAST(s.end_interval_time AS DATE), 'HH24')
)
SELECT
    instance_number,
    TO_CHAR(hour_bucket, 'YYYY-MM-DD"T"HH24:00:00') AS hour_utc,
    ROUND(read_reqs_delta  / NULLIF(secs, 0), 1)    AS read_iops_avg,
    ROUND(write_reqs_delta / NULLIF(secs, 0), 1)    AS write_iops_avg
FROM   delta
ORDER  BY hour_bucket, instance_number;
```

The `GREATEST(..., 0)` wrap around `LAG` is intentional: when an instance restarts mid-window, the cumulative counter resets and the raw delta goes negative. We floor at zero so a restart appears as a missing bucket rather than a spurious negative IOPS spike.

Pro-Tip — Read the top-waits CSV by *class*, not by absolute time. When you open `awr_top_waits.csv`, ignore the absolute numbers and look at *which wait class dominates*. **User I/O**-dominant workloads thrive on Azure Premium SSD v2 with provisioned IOPS. **Concurrency**-dominant workloads (`gc buffer busy`, `enq: TX`) often expose application contention that will be **worse** on a single-instance target — flag those for the application team in Chapter 8.5.

Pro-Tip — AWR retention silently caps the window. If your AWR retention is set to the 8-day default, the script returns 8 days of data and your sizing model is sampling-biased. Before running, verify retention:

```
SELECT retention FROM dba_hist_wr_control;
```

For migration assessment you want **at minimum 35 days, ideally 90**. Adjust with:

```
EXEC DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(retention => 129600);
```

1.5.4 Statspack fallback (no Diagnostics + Tuning license)

Statspack uses `PERFSTAT.STATS$*` tables; the metric names align approximately 1:1 with the AWR equivalents (`STATS$SYSSTAT`, `STATS$SYSTEM_EVENT`, `STATS$OSSTAT`, `STATS$SNAPSHOT`). The chapter companion repository carries `scripts-statspack/02_workload_baseline_statspack.sql` which produces CSVs with the **same column contract**, so downstream chapters require no change.

1.6 Script 03 — Blocker Detection

Full source: `scripts/03_blocker_detection.sql`

The blocker list is deliberately **opinionated and short** — nine categories, not ninety. We triage from real-world engagements; these are items that change the architecture when discovered in week 1, and blow up the program timeline when discovered in month 6.

Remediation bands map to engineering effort, not data volume:

- **LOW** — configuration or driver change.
- **MEDIUM** — code refactor, no architectural change.
- **HIGH** — out-of-DB re-architecture, or a different target platform.

Categories detected:

Blocker	Source	Band
Java in Database	`dba_objects` (JAVA SOURCE/CLASS/ RESOURCE)	HIGH
Spatial / `SDO_GEOMETRY` columns	`dba_tab_columns`	HIGH
`XMLType` columns	`dba_tab_columns`	MEDIUM
User-defined object/collection types	`dba_types` (predefined='NO')	MEDIUM
Outgoing Database Links	`dba_db_links`	MEDIUM
Materialized views (refresh chains)	`dba_mviews`	MEDIUM
External tables	`dba_external_tables`	MEDIUM
Advanced Queuing queues	`dba_queues`	MEDIUM
Code referencing `UTL_FILE` / `UTL_HTTP` / `UTL_SMTP` / `UTL_TCP` / `DBMS_PIPE`	`dba_dependencies`	MEDIUM

Each subquery is bounded by `ROWNUM <= 10` for sample objects, and concatenated via `LISTAGG(... ON OVERFLOW TRUNCATE WITHOUT COUNT)` for safety against `ORA-01489`.

> **Deep Dive — Why Database Links are MEDIUM, not LOW.** `CREATE DATABASE LINK` reads as five lines of DDL. The catch: every DB link in the source points to a system that *also* has to be either migrated, exposed via Azure Private Link, or retired. The link is trivial; the dependency graph it implies is not. We score the *dependency*, not the syntax.

1.7 Script 04 — Schema Complexity

Full source: `scripts/04_schema_complexity.sql`

Four CSVs that drive **Chapter 5 effort estimation** (PL/SQL refactor) and **Chapter 8 compliance migration** (FGA → pgAudit / SQL Auditing).

The **hotspot score** in `plsql_hotspots.csv` blends two signals:

```
hotspot_score = lines_of_code + dependents × 50
```

The `× 50` weight reflects an empirical observation: a 1,000-line package with no dependents is a small project; a 200-line package with twelve dependents is the riskiest object in your refactor backlog because changing it ripples. Tune the weight in your fork if your codebase looks different — but ship the score to stakeholders unmodified for comparability across runs.

The FGA section unions Fine-Grained Audit policies (legacy) with Unified Audit policies (12c+) into a single inventory, with a `source_type` column distinguishing them — feeding directly into Chapter 8's compliance migration mapping.

1.8 Script 05 — The Orchestrator

Full source: `scripts/05_assessment_bundle.sh`

The orchestrator is a single Bash script that:

1. Verifies tools (`sqlplus`, `jq`, `tar`, `gpg`, `sha256sum`, `az`, `uuidgen`, `python3`) exist.
2. Verifies a read-only connection to the source.
3. Captures an environment fingerprint (`manifest.json`).
4. Runs the four discovery SQL files in order, passing the output directory as `&1`.
5. Aggregates the per-section CSVs into `mab_summary.json` using a CSV-aware Python helper.
6. Computes `checksums.sha256`, signs it with GPG (detached `mab.sig`).
7. Tars and uploads the archive to an immutable Azure Blob container.

Run it:

```
# One-time setup
export ORACLE_CONNECT='/@ORA19CPROD_RO'      # wallet alias — no password in env
export AZ_STORAGE_ACCOUNT='stmabarchive001'
export AZ_CONTAINER='mab-archive'
export GPG_SIGN_KEY='migration-lead@example.com'
az login --identity                          # or `az login` interactively

# Run. Typical wall time on a 20 TB estate: 6–12 minutes.
./05_assessment_bundle.sh
# stdout: 8f9a1b3c-2d4e-4a6f-9c1b-7e2d3a4b5c6d   (the run_id)
```

The script exits non-zero on any failure (`set -Eeuo pipefail`, `trap` on `ERR`) and leaves partial output on disk for forensics — **it never auto-deletes the working directory.**

> **Pro-Tip — Do not put the password on the command line.** Environment variables holding secrets are readable from `/proc/<pid>/environ` by anyone with the same UID. The orchestrator's documented mode is Oracle Wallet (`/@alias`). If you must use a connect string with credentials, scope it to a per-run shell with `env -i` so it does not pollute your interactive session.

1.9 The MAB Rubric — Worked Example

Applied to the anchor environment (20 TB `ORA19C-PROD`), a representative `mab_summary.json` looks like:

```
{
  "run_id": "8f9a1b3c-2d4e-4a6f-9c1b-7e2d3a4b5c6d",
  "source": {
    "db_name":      "ORA19CPROD",
    "version":      "19.21.0.0.0",
    "cdb":          "YES",
    "rac_nodes":    2,
    "characterset": "AL32UTF8",
    "platform":     "Linux x86_64"
  },
  "scorecard": {
    "red_features":   3,
    "amber_features": 11,
    "green_features": 7,
    "high_blockers":  1
  }
}
```

Three reds, one high blocker. From the rubric in § 1.4 this implies: **lift to Oracle Database@Azure is the lowest-risk target** for this environment; an MI/PG refactor would require formal feature-loss acceptance for the three reds. That decision is made — *with this evidence on the table* — in Chapter 2.

1.10 Troubleshooting Matrix

Symptom	Likely Cause	Fix
`ORA-00942: table or view does not exist` on `DBA_HIST_*`	Account lacks `SELECT_CATALOG_ROLE`, or Diag+Tuning pack disabled at instance level	Confirm grant; check `CONTROL_MANAGEMENT_PACK_ACCESS = 'DIAGNOSTIC+TUNING'`
CSV files exist but have **no header row**	`PAGESIZE 0` left in place — known SQL*Plus quirk with `MARKUP CSV`	Use `PAGESIZE 50000` — the chapter scripts already do
`ORA-01489: result of string concatenation is too long`	LISTAGG exceeded VARCHAR2(4000)	Chapter scripts use `ON OVERFLOW TRUNCATE WITHOUT COUNT` — verify you have the post-validator version
`awr_iops.csv` returns only 7 days	AWR retention at the 8-day default	`EXEC DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS( retention => 129600);` (90 days, in minutes)
Connection works for one node, fails for the other	SCAN listener round-robin hitting a stale instance, or NSG asymmetry	Verify SCAN VIPs resolve; test direct VIPs as a diagnostic only — production traffic must use SCAN
Blob upload fails with `AuthorizationFailure`	Managed identity lacks `Storage Blob Data Contributor` role	Assign the role on the storage account; RBAC propagation can take 5 minutes
Orchestrator hangs at *"Verifying read-only connection"*	Wallet not loaded, or `TNS_ADMIN` unset	`export TNS_ADMIN=/etc/oracle/wallet && sqlplus /@ORA19CPROD_RO` to reproduce in isolation

Symptom	Likely Cause	Fix
`mab_summary.json` shows zeros across the board	The summary helper looked up CSV columns by SQL*Plus's emitted header case	Column headers from `MARKUP CSV` are uppercase — pass `TARGET_SCORE` / `BAND` exactly as in the script

1.11 Chapter Closeout

The shape of what this chapter just produced is worth pausing on. A signed MAB archive in immutable Azure Blob Storage is not a status report — it is the *contract* every later chapter reads. The portability scorecard, the 90-day workload baseline, the blocker inventory with banded remediation cost, and the PL/SQL complexity model are not five separate documents you might or might not consult; they are five fields on one JSON object that the next chapter's `target_select.py` will refuse to run without. That refusal is the design — it is the architectural reason discovery is fully scripted, why the MAB is signed, and why a junior engineer can re-run this chapter on day 350 and get a byte-identical answer. The discipline starts here because *every* chapter after this one depends on the discovery you just signed being trustworthy.

In **Chapter 2 — Target Architecture Selection Matrices** we feed `mab_summary.json` into four decision tables that map this scorecard to one of four target architectures — Oracle Database@Azure, Azure IaaS RAC, Azure SQL Managed Instance, or PostgreSQL Flexible Server — and produce a defensible written recommendation.

Chapter Artifacts (shipped with this chapter)

File	Purpose
`scripts/01_feature_usage_audit.sql`	Feature usage audit + portability rubric
`scripts/02_workload_baseline.sql`	90-day AWR workload baseline
`scripts/03_blocker_detection.sql`	Banded blocker inventory
`scripts/04_schema_complexity.sql`	PL/SQL surface, partitioning, FGA inventory
`scripts/05_assessment_bundle.sh`	Orchestrator + bundle signer + uploader