Oracle GoldenGate Microservices — Level 4 (Expert/Architect) DBA Interview Guide

This guide targets senior/architect-level GoldenGate Microservices (GGMA) roles. It assumes familiarity with classic GoldenGate concepts (Extract, Replicat, trail files, Data Pump) and focuses on the Microservices architecture, security model, HA design, performance internals, and troubleshooting depth expected at Level 4.

---

1. Architecture Fundamentals

Q1. Describe the GoldenGate Microservices Architecture (GGMA) and how it differs from Classic Architecture (CA).

GGMA replaces the classic manager/ggsci model with a set of RESTful microservices running under a **Service Manager (SM)**. Instead of one monolithic process tree per installation, GGMA introduces: - **Service Manager** – the single entry point/registry; starts, stops, and monitors all other services and deployments; exposes a REST API (default port 9100 unless overridden). - **Deployments** – logical, isolated GoldenGate environments (each with its own Extract/Replicat processes, trail files, parameter files, checkpoint files) running under one SM. Multiple deployments allow multi-tenancy on a single host. - **Administration Service** – manages configuration: creating/editing Extract, Replicat, credential stores, and parameter files via REST/UI instead of GGSCI scripting. - **Distribution Service** – replaces the classic Data Pump; pushes trail data to remote Receiver Services over HTTPS. - **Receiver Service** – replaces the classic collector process; receives trail data pushed by a remote Distribution Service and writes it to local trail files. - **Performance Metrics Service (PMSRVR)** – collects and exposes real-time and historical statistics via REST, replacing GGSCI `INFO/STATS` polling as the primary metrics path. - **Metrics Server / Notification Service** – aggregates metrics and can push events (JAgent/JMX or REST callbacks) for monitoring integration (Grafana, OEM, Prometheus scrapers via REST).

Key philosophical difference: CA is process- and file-based with manager as a supervisor; GGMA is API-first, browser/REST-managed, and containerization/cloud-friendly. GGSCI still exists but talks to the services via REST rather than directly to OS processes.

Q2. What is a “Deployment” and why would you run multiple deployments on one host?

A deployment is an isolated GoldenGate runtime environment under a Service Manager: its own OGG_HOME-like structure (`var/lib/data`, `dirprm`, `dirdat`, `dircrd`, `dirchk`, `dirrpt`), listening ports, users, and credential store. Reasons for multiple deployments: - **Multi-tenancy** — isolate different applications/schemas/business units without separate installs. - **Version isolation** — run different GGMA versions concurrently during phased upgrades. - **Security segregation** — different deployments can bind to different network interfaces/ports and have independent ADMIN users mapped to different database credentials. - **Resource isolation** — separate deployments can be assigned separate credential domains and separate certificate chains.

Q3. Walk through the port/service topology of a typical GGMA install.

- Service Manager: default port 9100 (or configured), single per host (can manage multiple deployments).
- Each deployment exposes its own Administration Service, Distribution Service, Receiver Service, and PMSRVR, each bound to a distinct port (commonly configured in a range, e.g., 9101–9199 blocks per deployment).
- Distribution Service on the source pushes over HTTPS to the Receiver Service port on the target — this is a fundamental shift from classic architecture's "target pulls via collector" model; in GGMA the **source pushes**, simplifying firewall rules (only need outbound HTTPS from source to target's Receiver port) but requiring the source to have connectivity/credentials for the target.

---

2. Security & Credential Management

Q4. How does credential management work in GGMA, and what is the credential store (`GGSCI CREDENTIALSTORE`)?

GGMA never stores clear-text passwords in parameter files. The credential store is a wallet-based (Oracle Wallet-like, using autologin/PKCS structure) repository keyed by an **alias** and **domain**. Parameter files reference `USERIDALIAS alias DOMAIN domain` instead of `USERID/PASSWORD`. Each deployment has its own credential store located under `dircrd`. Admins add credentials via `ALTER CREDENTIALSTORE ADD USER db_user, PASSWORD xxx ALIAS alias1 DOMAIN OracleGoldenGate` (or a custom domain), and this can also be done via REST/Admin Client.

Q5. Explain the certificate/TLS model for inter-service and inter-node communication.

All REST communication (browser-to-SM, SM-to-deployment, Distribution-to-Receiver) rides over HTTPS. GGMA requires: - A **keystore** containing the server's private key + certificate (self-signed or CA-issued) for each listening service. - A **truststore** on the calling side containing the CA or peer certificate to validate the handshake. - Mutual TLS (mTLS) can be configured so the Distribution Service also presents a client certificate to the Receiver Service, and the Receiver validates it — critical in zero-trust network designs. - Certificates are managed via `ADMIN CLIENT` commands (`ALTER... ADD TRUSTEDCERT`, keystore create/import) or REST endpoints; rotation requires updating both keystore and any dependent truststores and restarting/reloading the affected service, not the whole SM.

Q6. How do you implement role-based access control (RBAC) across deployments?

GGMA ships built-in roles: **Administrator, Operator/POC (Security), Analyst/User, and Distribution Admin/Receiver Admin (with recent versions adding fine-grained roles)**. Users are created at the Service Manager level or per-deployment; a user can be granted different roles per deployment (e.g., Analyst on Deployment A, Administrator on Deployment B). Best practice at L4: bind GGMA authentication to an external identity provider (LDAP/OAuth2/OIDC where supported) rather than local users, and enforce least privilege — Distribution/Receiver service accounts should not carry Administrator rights on the deployment, only the rights needed to push/receive trail data.

Q7. What's different about network exposure risk in GGMA vs. CA, and how do you mitigate it?

CA processes (Extract/Replicat/Manager/Collector) communicate over plain TCP by default unless you layer SSL manually; GGMA is REST/HTTPS by default, meaning every deployment is effectively a web service — increasing attack surface (each port is an API endpoint). Mitigations: put GGMA behind a reverse proxy/load balancer with WAF rules, disable unused services on a deployment (e.g., disable PMSRVR external exposure if metrics are scraped internally), enforce mTLS between Distribution/Receiver, rotate certs, and restrict the Service Manager's admin port to a management VLAN/bastion.

---

3. Deployment, HA, and DR Design

Q8. Design a highly available GGMA topology for an active-active or active-passive Oracle environment.

Key components: - **Service Manager HA**: SM itself is lightweight; typical pattern is to run SM under OS-level clustering (e.g., pacemaker/Grid Infrastructure resource, or on Kubernetes as a pod with a Deployment/StatefulSet) so it restarts automatically. SM state (deployment registry) persists in local config; back it up and replicate to standby node storage. - **Deployment failover**: place `dirdat`, `dirchk`, `dircrd`, `dirprm` on shared/replicated storage (ACFS, shared NFS, or a clustered filesystem) so that if the node running the deployment fails, a peer node's SM can start the same deployment against the same checkpoint/trail data — avoiding full resync. - **Distribution/Receiver redundancy**: configure multiple Distribution Service instances (or Distribution Paths) targeting Receiver Services on multiple candidate target nodes; use DNS/VIP fronting so a Receiver Service failure fails over transparently. - **Extract/Replicat checkpoint durability**: since checkpoints live in `dirchk` on shared storage, a failed-over process resumes from last checkpoint with minimal data loss (near-zero if trail files were also on shared/replicated storage). - For **Oracle RAC** sources/targets, GGMA integrates via the same thread-merge/tanking mechanisms as CA — Extract on RAC needs `TRANLOGOPTIONS` for ASM/thread-specific redo access, unaffected by the microservices layer itself.

Q9. How do you handle DR (cross-region) with GGMA?

Two common patterns: 1. **Trail file shipping continues as normal replication** — GGMA's Distribution Service pushes trails to a DR-region Receiver Service continuously, so the DR site's Replicat lags only by network + apply latency (same principle as classic pump-based DR, just push-based). 2. **Cold-standby SM/deployment**: keep an inactive Service Manager + deployment config (via `oggca.sh`/REST templates or exported deployment configuration JSON) in the DR region; on failover, deploy from the saved configuration onto DR infrastructure and point it at replicated trail/checkpoint storage.

At L4 you should be able to discuss **RPO/RTO trade-offs**: push-based Distribution reduces RPO because writes reach the target continuously (vs. classic scheduled pump batches), but RTO depends on how quickly SM/deployment can be reinstantiated — hence favoring shared/replicated storage for `dirchk` and `dirdat` to skip resync.

Q10. What is Parallel Replicat, and how does it interact with GGMA's architecture?

Parallel Replicat is an apply-engine feature (available in both CA and GGMA) that replaces the classic Coordinated Replicat model with a single logical Replicat that internally partitions and applies transactions across multiple threads/mappers based on dependency

analysis (using an in-memory or on-disk **Performance Metrics** and dependency graph). In GGMA it's configured and monitored the same way (via `MAP/PARALLELISM` parameters and the Admin Service), but PMSRVR gives much richer per-thread throughput/latency visibility via REST than classic `SEND REPLICAT STATUS` polling — important for tuning apply parallelism (`MAX_PARALLELISM`, `MIN_PARALLELISM`) at scale.

---

4. Configuration & Operations

Q11. How do you create a deployment and add Extract/Replicat purely via REST/Admin Client vs. classic GGSCI?

At L4 you should be comfortable both ways: - **GUI/REST**: use the Service Manager's `/services/v2/deployments` endpoint (or the browser console) to create a deployment, specify OGG_HOME software location, ports, and admin credentials; then call the Administration Service REST endpoints (`/services/v2/config/extracts`, `/config/replicats`) to define processes, or push parameter files directly via the `/parameters/{file}` endpoint. - **GGSCI (still supported)**: `ADMIN CLIENT` connects to the Administration Service via REST under the hood; commands like `ADD EXTRACT`, `ADD REPLICAT`, `ADD TRANDATA`, `REGISTER EXTRACT` behave nearly identically to CA syntax, but execution is delegated to the Admin Service rather than directly manipulating OS-level files. - Discuss automation: Terraform/Ansible modules and the GGMA REST API are commonly used to fully automate deployment provisioning in CI/CD pipelines for database platform teams.

Q12. What replaces `GLOBALS` and `mgr.prm` in GGMA?

There's no classic Manager process; port/purge/resource settings that used to live in `mgr.prm` are now deployment-level configuration set through the Service Manager/Administration Service (e.g., trail purge policies configured via `PURGEOLDEXTRACTS` still exist as Extract parameters, but scheduling/housekeeping that Manager used to do — like dynamic port allocation, autostart, and resource limits — is handled by SM/Admin Service settings and deployment properties, not a `GLOBALS` file).

Q13. How is trail file management and purging different in GGMA?

Trail files still live under `dirdat` per deployment. Purging policy is still controlled largely via `PURGEOLDEXTRACTS/PURGEOLDREPLICAT` type parameters on the ADD TRANDATA or via `PURGE` retention settings, but disk-space and file lifecycle monitoring is exposed through PMSRVR metrics (trail growth rate, lag) so you can build proactive alerting instead of relying on manual `dirdat` disk checks.

Q14. Explain how DDL replication and Integrated Extract/Replicat work under GGMA — are there architectural differences from CA?

No — Integrated Capture/Apply (using the database's logmining server and inbound/outbound server processes) works identically at the database integration layer; GGMA changes only the *control plane* (how you configure/monitor/start/stop), not the *data plane* mechanics of Integrated Extract mining redo via the LogMiner-based streaming API, or Integrated Replicat applying via the inbound server. DDL trigger-based capture, `DDL INCLUDE/EXCLUDE MAPPED`, and DDL history table (`GGS_DDL_HIST`) work the same. The key operational difference: DDL/perf statistics for these processes are visible in richer REST/JSON form via PMSRVR.

Q15. How do you migrate an existing Classic Architecture GoldenGate environment to Microservices?

High-level path: 1. Install GGMA software (new Oracle home) alongside or replacing CA. 2. Create a Service Manager and a deployment pointing its `dirdat`/`dirprm`/`dirchk` at (or migrated from) the CA instance's directories — Oracle provides utilities/documented steps to reuse existing trail and checkpoint files where compatible. 3. Recreate credential store entries (CA's `ORAPWD`-obfuscated or clear passwords must be re-added into the GGMA wallet-based credential store). 4. Re-point Extract/Replicat parameter files to use `USERIDALIAS` instead of `USERID`. 5. Convert classic Data Pump configuration to Distribution Service / Receiver Service configuration (source pushes instead of target pulls) — this typically requires updating remote trail definitions and firewall rules (outbound HTTPS from source to target). 6. Cut over incrementally: run CA and GGMA in parallel briefly, validate lag/checksums, then decommission CA processes. 7. Update monitoring integrations (OEM plug-ins, custom scripts polling GGSCI) to use PMSRVR/REST instead of GGSCI text parsing.

---

5. Performance Tuning & Troubleshooting

Q16. A Distribution Service shows growing backlog while Receiver Service on the target looks idle. How do you diagnose?

Systematic approach: 1. Check PMSRVR metrics for the Distribution Service: throughput (bytes/sec, records/sec), current file being read, and read lag vs. Extract's write position. 2. Verify network path: HTTPS handshake latency, TLS renegotiation overhead, or a saturated link between source and target (Distribution pushes are HTTP POST-based; a slow/lossy WAN link manifests as backlog even though both ends look "idle" individually). 3. Check Receiver Service logs for authentication/certificate errors that silently reject batches, or disk write bottlenecks on the target's `dirdat`. 4. Confirm Distribution Path configuration (`ADD DISTPATH` equivalent) isn't misconfigured to a stale/incorrect Receiver endpoint or port, and that keepalive/timeout parameters aren't too aggressive for the link's latency. 5. Compare with `LAG AT CHKPT` / `LAG REPORTED` (still exposed via REST `/statistics`) between Extract and Distribution, and Distribution vs. Receiver, to isolate which hop is actually behind.

Q17. How do you tune Parallel Replicat parallelism, and what metrics does PMSRVR expose to guide this?

Start with `MAX_PARALLELISM`/`MIN_PARALLELISM` (or `PARALLELISM` depending on version) based on target hardware (CPU cores, I/O subsystem) and workload characteristics (transaction dependency density). Use PMSRVR's per-mapper/per-thread metrics — records applied/sec per thread, average transaction size, and time spent in dependency-wait vs. actual apply — to identify whether you're CPU-bound, I/O-bound, or dependency-serialized (i.e., high lock contention forcing near-serial apply despite high parallelism settings). If dependency-wait dominates, increasing parallelism further won't help; instead look at reducing transaction interdependency (e.g., via `TRANSACTION SPLIT` tuning) or batching (`BATCHSQL`/array processing tuning at the target).

Q18. What are common causes of Extract "abend" in GGMA and how does troubleshooting differ from CA?

Root causes are the same as CA (archived log gaps, insufficient supplemental logging, corrupted trail, resource exhaustion, credential expiry). The differentiator in GGMA: - Error reporting surfaces through the Administration Service's REST `/status` and the report file, plus can trigger webhook/notification callbacks if configured —

enabling automated alerting without polling GGSCI. - Credential-related failures often trace to the wallet/credential store (expired DB password not updated in the store, or wallet corruption) rather than a plaintext `PASSWORD` typo — check `ALTER CREDENTIALSTORE` state and DB-side password expiry policies first. - Since deployments are isolated, confirm you're diagnosing the right deployment/service — GGMA hosts can run several deployments with similarly named processes, so REST calls or GGSCI sessions must target the correct deployment context.

Q19. How would you benchmark end-to-end replication latency (source commit to target apply) in a GGMA pipeline, and what's the breakdown of contributing components?

Break the pipeline into hops and use PMSRVR timestamps at each stage: 1. **Capture lag** — Extract's read position vs. database SCN/commit time (mining lag). 2. **Trail write lag** — time from capture to trail file write. 2. **Distribution lag** — time from trail write to successful POST/ack at Receiver. 4. **Receiver-to-trail lag** — time for Receiver to persist to target-side trail. 5. **Apply lag** — Replicat/Parallel Replicat time from trail read to commit on target.

Sum these using the `LAG` and `CHECKPOINT` timestamps exposed via `/services/v2/.../statistics` REST endpoints (rather than shell-scripting GGSCI `INFO EXTRACT/INFO REPLICAT` output parsing, which was the CA-era approach). At L4, be ready to discuss building a Grafana/Prometheus dashboard on top of PMSRVR's metrics endpoint for continuous latency SLO tracking.

Q20. How does conflict detection and resolution (CDR) work in GGMA for active-active configurations, and what changed operationally?

CDR logic (before/after image comparison, `COMPARECOLS`, `RESOLVECONFLICT`, procedures like `USEMAX`/`OVERWRITE`/`DISCARD`) is unchanged mechanically — it's still configured via Replicat parameters/`MAP` statement CDR clauses. What's different operationally is visibility: conflict counts, resolution outcomes, and discard-file growth are exposed as first-class metrics in PMSRVR, making it easier to alert when conflict rates exceed a baseline (a sign of application-level bugs or clock-skew issues across active-active sites) instead of grepping discard files.

---

6. Integration, Cloud, and Advanced Scenarios

Q21. How does GGMA integrate with Oracle GoldenGate Stream Analytics, Kafka, or Big Data adapters?

The Big Data/Kafka Handler Replicat still runs as a Replicat process within a deployment, configured with the relevant Java Adapter properties file. GGMA's value-add is centralizing credential management for the target Kafka cluster (SASL/SSL credentials in the same wallet-based store) and exposing handler-specific throughput metrics via PMSRVR alongside database Replicats — giving a single monitoring pane across heterogeneous targets.

Q22. How would you run GGMA in a containerized/Kubernetes environment, and what are the persistence considerations?

Oracle publishes container images for GGMA. Key design points: - `dirdat`, `dirchk`, `dircrd`, `dirprm`, `dirrpt` must be on **persistent volumes** (PV/PVC), not ephemeral container storage, or a pod restart loses trail/checkpoint state. - Service discovery: use Kubernetes

Services/Ingress to expose SM and per-deployment ports; TLS termination can be at the ingress or maintained end-to-end for zero-trust requirements. - Secrets (DB passwords used to seed the credential store) should be injected via Kubernetes Secrets and loaded into the wallet at container init, not baked into the image. - StatefulSets are preferred over Deployments (k8s object) for GGMA pods needing stable network identity/storage per replica, especially when running multiple GGMA deployments per node group.

Q23. Explain sharded database (Oracle Sharding) replication considerations with GGMA.

For Oracle Globally Distributed Database (Sharding), GoldenGate is often used for cross-shard consistency or for replicating into/out of a sharded topology. Considerations: you typically need one Extract per shard (each shard is its own database with its own redo stream), aggregated into a single logical trail stream on the hub/coordinator side, or independent Distribution Paths per shard feeding a consolidated Replicat. GGMA's multi-deployment model maps naturally here — one deployment (or one Extract within a shared deployment) per shard catalog/shard, all monitored centrally through one Service Manager's aggregated PMSRVR view.

Q24. How do you handle GGMA software patching/upgrades with minimal downtime?

- GGMA supports **rolling upgrades** at the deployment level: install the new OGG_HOME, register it with the Service Manager, and migrate deployments one at a time (stop, point deployment metadata at new home, start) rather than a full-environment outage.
- Because checkpoints/trail files are version-tolerant across minor versions (per Oracle's compatibility matrix — always verify the specific matrix for the source/target versions in play), Extract/Replicat can resume from existing checkpoints post-upgrade without a resync in most cases.
- Best practice: upgrade Receiver/target-side deployments before Distribution/source-side deployments when there are protocol-level enhancements, to maintain backward compatibility of the push protocol during the transition window; consult the specific version's upgrade guide for any required ordering.

Q25. What's your approach to monitoring/alerting strategy for a large-scale GGMA estate (dozens of deployments across multiple hosts)?

- Centralize metrics collection via PMSRVR's REST/metrics endpoints, scraped by Prometheus (or fed into OEM's GoldenGate plug-in), rather than host-by-host GGSCI polling.
- Define SLOs per deployment: max acceptable lag, max discard rate, min throughput — alert on trend breaches, not just absolute thresholds (sudden lag *acceleration* often precedes an abend).
- Correlate GGMA metrics with source/target database health metrics (redo generation rate, AWR data) so lag spikes can be attributed to capture-side vs. apply-side vs. network causes quickly.
- Maintain a configuration-as-code repository (deployment JSON/Terraform) so any deployment can be recreated from source control — critical for audit and fast DR rebuild.

---

7. Rapid-Fire Conceptual Checks (good for screening candidates)

Question	Expected Answer
What replaced the classic Manager process?	Service Manager (control plane) + per-deployment services
What replaced Data Pump / Collector?	Distribution Service (push) / Receiver Service (receive)
Default communication protocol?	HTTPS/REST
Where are passwords stored?	Wallet-based credential store, referenced via `USERIDALIAS`
What service exposes metrics?	Performance Metrics Service (PMSRVR)
Can GGSCI still be used?	Yes, as a REST client (`ADMIN CLIENT`) to the Administration Service
Is Integrated Capture/Apply mechanically different in GGMA?	No — only the control/monitoring plane changes
Key HA design point for deployments?	Shared/replicated storage for `dirdat`/`dirchk` + SM under cluster management
What enables multi-tenancy on one host?	Multiple isolated deployments under one Service Manager
What must be reconfigured when migrating CA→GGMA?	Credential store entries, `USERID`→`USERIDALIAS`, Data Pump→Distribution/Receiver Paths

---

This guide is intended as an interview-prep/reference document. Always validate version-specific behavior (port defaults, parameter names, upgrade compatibility matrices) against the current Oracle GoldenGate documentation for the specific release in scope, since GGMA has evolved significantly across 19c/21c/23ai releases.