Interview Prep Guide

Platform & Cloud Architect

30 Practice Questions with STAR Answers

Mapped to Sathish Kumar Prakash's Experience at Takeda & Societe Generale

Contents

Interview Strategy & How to Use This Guide

This guide is designed around your actual experience at Takeda Pharmaceuticals and Societe Generale. Every answer uses real projects, real metrics, and real decisions you made. Nothing is fabricated. The goal is not to memorize answers but to practice telling your stories with structure and confidence.

What Interviewers Evaluate at the Architect Level

Platform and Cloud Architect interviews are fundamentally different from senior engineer interviews. Interviewers are not testing whether you can do the work. They are testing whether you can **design systems, make trade-offs, communicate decisions, and lead teams through change**. Here is what they weight most heavily:

- **System design thinking (40% of evaluation)** — Can you decompose a problem, identify components, explain trade-offs, and justify your choices? Every technical round will include at least one design question.
- **Production depth over theoretical knowledge (25%)** — They want to hear about systems you built that actually run in production. Your 60+ pipeline migration, EKS containerization, and Vault OIDC integration are exactly what they are looking for.
- **Quantified impact (15%)** — Numbers stick. '30% faster builds' is memorable. 'Improved build times' is forgettable. Lead with metrics in every answer.
- **Leadership and influence (20%)** — Can you get teams to adopt a platform without mandating it? Can you present technical trade-offs to non-technical leadership? Can you mentor engineers? Your Jenkins migration and Societe Generale team lead stories cover this.

How to Practice

For each question, practice answering out loud for 3 to 5 minutes. Use the STAR format strictly: Situation, Task, Action, Result. Start with the 6 System Design questions — these appear in every architect interview. Then work through CI/CD and Security, which are your strongest differentiators. Save Behavioral for last since those stories will feel natural once you have practiced the technical answers.

Your Unique Differentiators

- **Mobile CI/CD expertise** — Most cloud architects have never touched Fastlane, Xcode CLI, or self-hosted macOS runners. This is rare and valuable. Do not downplay it.
- **Vault + GitHub OIDC integration** — Most candidates say 'we used GitHub Secrets.' Your zero-static-credentials architecture is architecturally superior.
- **Pharma compliance context** — Working at Takeda gives you a compliance and audit angle that candidates from pure tech companies lack.
- **Open source contributions** — Action Log Analyser on the GitHub Marketplace shows you build beyond your day job.
- **Full-stack platform ownership** — You own the pipeline from code commit to production deployment, including mobile. That end-to-end ownership is the definition of a platform architect.

Interview Format to Expect

Most Platform Architect interviews run 3 to 5 rounds. Expect: a recruiter screen (30 min, culture fit and salary expectations), a hiring manager call (45 min, your background and why this role), a system design round (60 min, whiteboard or virtual, this is the main event), a technical deep-dive (45 to 60 min, specific technologies from your resume — AWS, Kubernetes, CI/CD, security), and a behavioral or leadership round (45 min, STAR format questions). Some companies also ask you to present a past project for 15 to 20 minutes. Your

Jenkins-to-GitHub Actions migration is perfect for this.

Self-Introduction Scripts

The first question in every interview. Practice these until they feel natural, not rehearsed. Make eye contact, speak at a steady pace, and land the metrics with confidence.

Version 1 — 60-Second Introduction (Recruiter Screen)

Use this for the initial recruiter call or when someone says 'tell me a bit about yourself.' Keep it tight. Hit the headline, one signature achievement, and why you are looking.

> "Hi, I'm Sathish. I'm a Platform and Cloud Architect with over 15 years of experience, currently at Cognizant embedded with Takeda Pharmaceuticals in the Chicago area.
>
> For the past five-plus years at Takeda, I've been leading their CI/CD platform and cloud infrastructure. The highlight has been migrating 60-plus Jenkins pipelines to GitHub Actions, which cut build times by 30 percent and reduced new service onboarding from two weeks down to two hours. I also built the mobile CI/CD pipeline from scratch using Fastlane and self-hosted macOS runners, taking iOS build times from 45 minutes to 15. On the security side, I integrated HashiCorp Vault with GitHub OIDC to eliminate all static credentials from our pipelines.
>
> I'm looking for new challenges now. I've built and optimized the platform at Takeda to a point where it runs itself, and I'm excited to take on a larger-scale architecture role where I can design cloud-native platforms from the ground up. I'm particularly interested in roles that involve Kubernetes, infrastructure as code, and platform engineering at scale."

Timing: approximately 50-60 seconds. Practice with a stopwatch.

What this covers and why:

- **Who you are and where you work** — establishes context in one sentence. Mentioning 'Chicago area' signals location without over-explaining.
- **One signature platform achievement** — the 60+ pipeline migration with metrics. This is memorable and specific.
- **One differentiator** — mobile CI/CD. Most candidates cannot talk about this. It sets you apart immediately.
- **One security story** — Vault OIDC. Shows you think beyond just building pipelines.
- **Why you are looking** — framed positively. Not 'I want to leave' but 'I've built the platform to maturity and want a bigger challenge.' This shows growth mindset, not dissatisfaction.

Version 2 — 2-3 Minute Introduction (Hiring Manager)

Use this when the hiring manager says 'walk me through your background' or 'tell me about yourself.' This is your chance to tell your career story with an arc that leads naturally to why you are the right fit for a Platform Architect role.

"Hi, I'm Sathish. I'm a Platform and Cloud Architect with over 15 years of experience spanning cloud infrastructure, CI/CD automation, and full-stack development. I'm currently at Cognizant, embedded with Takeda Pharmaceuticals in Bannockburn, Illinois.

My career arc: I started as a front-end engineer at Verismo Networks, building HTML5 applications for set-top boxes and streaming platforms. That gave me a strong foundation in shipping consumer-facing products. I then moved to NetApp where I built internal tooling for firmware debugging, and after that spent six years at Societe Generale through Cognizant, where I grew into a Dev Chapter Lead managing a team of six-plus developers across multiple business continuity applications. That role was where I learned to drive architectural decisions, mentor engineers, and deliver under enterprise constraints. I improved team velocity by 35 percent through structured code reviews and pairing sessions.

My current role at Takeda is where everything came together. For the past five-plus years, I've owned the CI/CD platform and cloud infrastructure end to end. The biggest initiative was migrating 60-plus pipelines from Jenkins to GitHub Actions. I designed a three-tier reusable workflow architecture, integrated HashiCorp Vault with GitHub OIDC to eliminate every static credential, and set up Harness CD for deployment orchestration to EKS and ECS. The results: build times dropped 30 percent, new service onboarding went from two weeks to two hours, and we saved over 200 engineering hours per month.

I also built the mobile CI/CD pipeline from scratch. iOS builds were taking 45 minutes with manual code signing and two-day release cycles. I set up self-hosted macOS runners, automated signing with Fastlane Match, and got builds down to 15 minutes with fully automated releases in 15 minutes. That's a capability most platform teams don't have.

Beyond the day job, I've been contributing to open source. I published Action Log Analyser on the GitHub Marketplace under my SKCloudOps org. It's a GitHub Action with 42 error patterns across 16 build systems and 205 automated test cases. I also have a second action in development called AutoDocs for AI-powered documentation generation from code changes.

Why I'm looking: I've built the platform at Takeda to a mature state. The pipelines run reliably, the security model is solid, and the teams are self-sufficient. I'm looking for new challenges — ideally a role where I can architect cloud-native platforms at a larger scale, work with Kubernetes-heavy infrastructure, and have more direct ownership over the technical roadmap. I'm particularly drawn to organizations where platform engineering is treated as a product, not just a support function.

I'm open to remote roles and am ready to start contributing quickly."

Timing: approximately 2 to 2.5 minutes. If you go past 3 minutes, you are losing them.

Structure of the Long Version:

- **One-line positioning** (5 seconds) — who you are, years of experience, where you work. Sets the frame.
- **Career arc** (30 seconds) — front-end to team lead to platform architect. Shows growth and intentional career progression, not random job-hopping.
- **Current role deep-dive** (60 seconds) — this is the meat. Three stories: pipeline migration with metrics, Vault OIDC security, mobile CI/CD. Every statement has a number attached.

- **Open source** (15 seconds) — brief mention of Action Log Analyser. Shows initiative beyond the day job.
- **Why you are looking** (20 seconds) — positive framing. 'I built it to maturity, now I want bigger challenges.' Not 'I'm bored' or 'I want more money.' Ends with what you want next, which makes it easy for the interviewer to connect your goals to their role.
- **Availability** (5 seconds) — remote, ready to start. Clean close.

Common Mistakes to Avoid

- **Do not recite your resume chronologically.** The interviewer has your resume. They want your narrative — the arc, the highlights, and the why.
- **Do not start with 'I have 15 years of experience in...'** and then list technologies. Start with your name and a positioning statement, then prove it with stories.
- **Do not apologize for what you lack.** Never say 'I don't have CKA yet' or 'I haven't worked with Azure.' Focus on what you bring.
- **Do not forget the 'why I'm looking' part.** If you skip it, the interviewer will ask anyway. Better to control the narrative.
- **Do not speak in third person.** Say 'I built' not 'The team built.' Take ownership of your contributions.

1. System Design

Highest-weighted questions. Start with requirements, then components, then trade-offs. Always draw from real experience.

Q1 Design a CI/CD platform that serves 50+ engineering teams. Walk me through your architecture.

WHY THEY ASK THIS

They want to see platform-scale thinking, not pipeline-by-pipeline. They evaluate self-service, standardization, and governance.

YOUR STAR STORY — TAKEDA

Situation: 60+ pipelines on Jenkins — fragmented, no standardization, each team maintaining their own Jenkinsfiles. Build reliability was poor, onboarding a new service took 2 weeks.

Task: Lead the migration to a modern CI/CD platform that scales across all teams with consistent quality and security.

Action: Designed a GitHub Actions platform with reusable composite actions and workflow templates. Three layers: shared template library (org-level), team-specific overrides, environment-specific configs. Integrated HashiCorp Vault with GitHub OIDC for dynamic IAM role assumption — zero static credentials. Self-hosted macOS runners with Fastlane for mobile. Harness CD for deployment orchestration to EKS and ECS.

Result: Migrated **60+ pipelines**, reduced build time by **30%**, cut onboarding from **2 weeks to 2 hours**, saved **200+ engineering hours/month**.

HOW TO SAY IT

Draw three boxes on the whiteboard: Source & Templates (GitHub org-level reusable workflows), Build & Test (GitHub Actions runners — Linux for backend, self-hosted macOS for mobile), Deploy (Harness CD to EKS/ECS with environment promotion). Then cross-cutting concerns: Secrets (Vault + OIDC), Artifacts (JFrog Artifactory), Observability (ELK + Grafana), Governance (branch protection, approval gates).

LIKELY FOLLOW-UPS

"How did you handle teams that didn't want to migrate?" — Ran Jenkins and GitHub Actions in parallel, let early-adopter teams prove the value, then made it the default.

"How do you handle a pipeline failure at 2 AM?" — Alerting through Slack + PagerDuty integration, observability with ELK Stack.

Q2 Design a migration strategy for moving a monolithic application to microservices on Kubernetes.

YOUR STAR STORY — TAKEDA

Situation: Monolithic EC2 workloads — hard to scale, deploy independently, or update without risk.

Action: Strangler fig approach. Identified least-coupled modules first, containerized with Docker, created Helm charts for EKS, routed traffic progressively using ALB path-based routing. Split shared databases using bounded-context model with dual writes during transition.

Result: 99.9% deployment success rate, independent scaling per service, faster release velocity.

HOW TO SAY IT

Four phases: 1) Assess (map dependencies, identify low-risk services), 2) Containerize (Docker + Helm, no behavior changes), 3) Migrate traffic (ALB routing, canary rollouts), 4) Decouple data (bounded contexts, dual writes, then cut over). Emphasize incremental approach — never a big-bang switch.

Q3 How would you design a high-availability architecture for a production application on AWS?

YOUR STAR STORY — TAKEDA

Situation: Pharma applications with compliance requirements and zero tolerance for extended downtime.

Action: Layered HA: public NLB (TCP) to internal ALB (HTTP routing) to backend services. EKS across 3 AZs with pod anti-affinity. RDS multi-AZ with automated failover. Route 53 geo-DNS with health checks. Cross-VPC via NLB-to-ALB chaining with PrivateLink.

Result: Zero unplanned downtime during entire tenure for architected services.

PRO TIP

Always ask about RTO and RPO first. "Before I design, I'd ask what the recovery time objective and recovery point objective are — that determines active-active vs active-passive." This shows architect-level thinking.

Q4 Walk me through how you would design an iOS/Android mobile CI/CD pipeline from scratch.

WHY THEY ASK THIS

Mobile CI/CD is rare. Most cloud architects cannot speak to this. This differentiates you immediately.

YOUR STAR STORY — TAKEDA

Situation: iOS builds took 45 minutes. Releases took 2 days of manual work — code signing, provisioning, TestFlight uploads all done manually.

Action: Self-hosted macOS runners for GitHub Actions. Fastlane Match for centralized code signing (certs in private Git repo). Separate Fastlane lanes for dev/staging/prod. Full automation: PR triggers lint + tests, merge to main triggers build + sign + TestFlight/Play Store upload. Cached CocoaPods and node_modules.

Result: iOS build time **45 to 15 minutes** (67% reduction). Release cycle **2 days to 15 minutes**.

LIKELY FOLLOW-UPS

"What specifically cut the build time?" — CocoaPods/SPM caching, derived data caching, removing redundant clean builds, Fastlane lane optimization, dedicated runners eliminating queue wait time.

"How did you manage provisioning profiles?" — Fastlane Match with private Git repo. Each environment had its own profile, automatically synced.

Q5 How would you migrate an on-premises artifact management system to the cloud with zero downtime?

YOUR STAR STORY — TAKEDA

Situation: JFrog Artifactory and Jira running on-premises — maintenance burden, slow retrieval, single points of failure.

Action: Phased migration: synced repositories to cloud using built-in replication, validated checksums, switched pipelines to cloud endpoint in batches. Ran dual-write/read period, monitored discrepancies, then cut over DNS. Same URLs, same SSO authentication.

Result: Zero-downtime cutover. Artifact retrieval **40% faster**. Eliminated on-prem hardware maintenance.

HOW TO SAY IT

Six steps: 1) Replicate (sync to cloud while on-prem stays live), 2) Validate (checksums), 3) Dual-run (both active), 4) Switch readers (pipelines point to cloud), 5) Cut writers (cloud becomes source of truth), 6) Decommission (on-prem off after bake period).

Q6 How do you approach cross-account AWS migrations while preserving IAM policies and data integrity?

YOUR STAR STORY — TAKEDA

Situation: Needed to migrate ECS Fargate workloads and AWS Glue jobs across accounts as part of consolidation.

Action: Codified all infrastructure with Terraform. Wrote plans for target account with equivalent IAM roles, security groups, VPC configs. Re-deployed ECS task definitions pointing to same ECR images (cross-account via resource-based policies). Exported Glue definitions, recreated with updated IAM. S3 synced via cross-account replication. Integration tests before switching Route 53 DNS.

Result: Full data integrity, IAM policies, and service continuity preserved. No data loss, no interruption.

PRO TIP

Mention gotchas: KMS-encrypted S3 objects require cross-account KMS key policies. ECS task role ARNs change across accounts so all IAM trust policies need updating.

---

2. AWS Architecture

They will ask WHY you chose one service over another. Always lead with the business requirement.

Q7 Why did you use both NLB and ALB in your architecture? Why not just one?

HOW TO SAY IT

"NLB at the front for non-HTTP traffic (TCP) and static IPs for partner allowlisting. ALB behind it for path-based routing, host-based routing, and TLS termination. Architecture: public NLB to internal ALB to backend targets. Clean security boundary — NLB in public subnet, ALB in private subnet, backends in isolated subnets."

LIKELY FOLLOW-UPS

"Couldn't you use ALB alone?" — ALB does not provide static IPs. External partners needed IP allowlisting. NLB also handles millions of requests/sec at Layer 4 with ultra-low latency.

"How did you handle TLS?" — Terminated at ALB using ACM certificates. NLB passed through TCP — no double encryption overhead.

Q8 Walk me through your AWS cost optimization strategy.

YOUR STAR STORY

Action: Three-pronged: 1) Rightsizing — CloudWatch metrics + Compute Optimizer, downgraded ~40% of instances. 2) Spot instances — non-critical workloads (CI runners, batch, dev/staging) with graceful interruption handling. 3) Reserved capacity — 1-year RIs for stable production baselines after 3-month usage analysis.

Result: 25% reduction in monthly cloud spend.

PRO TIP

Frame as a business conversation: "I presented savings to leadership with a before/after dashboard, which built trust for future infrastructure investments."

Q9 When would you choose EKS over ECS, and vice versa?

HOW TO SAY IT

"Used both at Takeda. ECS Fargate when: small team, straightforward workload, want AWS to manage everything, no need for Kubernetes-specific features like CRDs or service mesh. Simpler and cheaper. EKS when: need pod-level networking control, custom operators, Helm ecosystem, complex scheduling. At Takeda — EKS for main microservices platform, ECS Fargate for standalone batch jobs and Glue-adjacent workloads."

Q10 How do you design cross-VPC communication securely?

HOW TO SAY IT

"Hub-and-spoke VPC model with Transit Gateway. For service-to-service calls, VPC PrivateLink via NLB endpoints — traffic stays on AWS backbone. Security groups at ENI level, VPC Flow Logs + CloudTrail for audit."

LIKELY FOLLOW-UPS

"VPC Peering vs Transit Gateway vs PrivateLink?" — Peering for simple 1:1 (no transitive routing). Transit Gateway for hub-and-spoke at scale. PrivateLink for exposing a single service endpoint securely across accounts/VPCs.

Q 11 How do you implement observability at scale?

YOUR STAR STORY

Action: Three-pillar observability: Metrics (Prometheus + Grafana), Logs (ELK Stack), Traces (AppDynamics). Each new service auto-onboarded via Terraform-provisioned dashboards and alerting rules.

Result: Covered **50+ services**. MTTD dropped significantly with proactive alerts.

PRO TIP

Mention the four golden signals: latency, traffic, errors, saturation. "Every service dashboard starts with these four. If an engineer can't see these in 30 seconds, the dashboard has failed."

---

3. Kubernetes

Without CKA, demonstrate production depth. Anchor every answer in Takeda experience, not exam theory.

Q 12 How did you design your EKS cluster architecture? Namespaces, RBAC, and scaling.

HOW TO SAY IT

"Namespaces — one per team/domain for isolation + quotas. Shared services in a platform namespace. RBAC — integrated with AWS IAM via aws-auth ConfigMap. Developers: read/exec to their namespace only. Platform team: cluster-admin. Service accounts with IRSA for pod AWS access. Scaling — HPA on CPU/memory, Cluster Autoscaler for nodes, VPA for stateful services."

LIKELY FOLLOW-UPS

"How do you handle noisy neighbors?" — Resource quotas per namespace, LimitRanges for default pod requests, pod priority classes for critical services.

Q 13 How do you handle Kubernetes deployments — rolling update vs. blue-green vs. canary?

HOW TO SAY IT

"Different strategies by risk: Rolling updates (default) — maxSurge=1, maxUnavailable=0, readiness probes gate traffic. Blue-green via Harness CD — for high-risk deployments (schema changes). New deployment, smoke tests, switch selector. Instant rollback. Canary — for customer-facing services. Harness managed 5-10% traffic split with automated rollback on error threshold."

Q 14 How do you manage Helm charts across multiple environments?

HOW TO SAY IT

"One Helm chart per service with environment-specific values files: values-dev.yaml, values-staging.yaml, values-prod.yaml. Template identical — only values change (replicas, limits, image tags, flags). Charts versioned in JFrog Artifactory as Helm repo. CI pushes chart on merge, Harness CD deploys with correct values file. Helmfile for multi-chart orchestration."

Q 15 What's your approach to Kubernetes networking and service discovery?

HOW TO SAY IT

"AWS VPC CNI plugin — each pod gets a real VPC IP. No overlay network overhead, native integration with security groups and Flow Logs. Service discovery: ClusterIP + CoreDNS for internal traffic. External: AWS Load Balancer Controller provisioning ALBs from Ingress resources with path-based routing."

PRO TIP

"We evaluated Istio but didn't adopt it — operational overhead wasn't justified. I always prefer the simplest solution that meets requirements." Shows pragmatic decision-making.

---

4. CI/CD & Platform Engineering

Your strongest area. Go deep. Interviewers rarely meet candidates who have built platforms at this scale.

Q16 How did you design reusable GitHub Actions workflows for 60+ teams?

HOW TO SAY IT

"Three-tier template architecture: Tier 1 — Org-level reusable workflows in a central .github repo, defining standard stages (checkout, build, test, scan, publish, deploy). Called via workflow_call. Tier 2 — Composite actions for specific tasks (Docker build+push, Helm deploy, Fastlane iOS). Tier 3 — Team overrides for team-specific steps. New service: zero to fully working pipeline in under 2 hours."

Q17 How do you manage secrets in CI/CD pipelines securely?

HOW TO SAY IT

"Eliminated all static secrets. HashiCorp Vault + GitHub OIDC — GitHub Actions authenticates using OIDC tokens. No long-lived credentials anywhere. Workflow requests short-lived token scoped by repo + branch + environment. IAM Role Assumption — Vault-issued token assumes IAM role via STS with least-privilege policies. Only Vault address and role ID stored in GitHub Secrets. Rotating a secret is a Vault operation, not updating 60 repos."

PRO TIP

This is one of your strongest differentiators. Most candidates say "we used GitHub Secrets." Your Vault + OIDC + dynamic IAM story is architecturally superior.

Q18 How do you measure CI/CD platform health and performance?

HOW TO SAY IT

"Four DORA metrics: deployment frequency, lead time for changes, change failure rate, mean time to recovery. Built Grafana dashboards from GitHub Actions API and Harness logs. Beyond DORA: build queue time, flaky test rate, pipeline cost per build, template adoption rate."

Q19 Tell me about a pipeline failure that taught you something important.

HOW TO SAY IT

"What broke — A Fastlane code signing certificate expired over a weekend; Monday's iOS build failed for all mobile teams. How I diagnosed — Error logs pointed to provisioning profile mismatch. Traced to expired cert in Fastlane Match repo. How I fixed — Regenerated certificate, pushed to Match, re-ran builds. 45-minute resolution. What I changed — Added certificate expiry monitoring that alerts 30 days before expiration. Never happened again."

PRO TIP

They want your debugging process and prevention strategy. "What did you change so this never happens again?" is the real question being asked.

5. Security & DevSecOps

Pharma experience at Takeda gives you a compliance angle most candidates lack.

Q 20 How do you implement least-privilege IAM in a large AWS environment?

HOW TO SAY IT

"Three principles: 1) No human access to production — all changes through CI/CD. Engineers read logs/metrics but cannot SSH or kubectl exec in prod. 2) Dynamic, scoped credentials — Vault + OIDC issues short-lived tokens scoped to specific repos/branches. Staging physically cannot access production secrets. 3) IAM boundary policies — permission boundaries cap what teams can do. AWS Organizations SCPs for account-level guardrails."

Q 21 How do you handle compliance requirements in your infrastructure?

HOW TO SAY IT

"In pharma, compliance was built into every decision. Infrastructure as Code — every change is Terraform, auditable, version-controlled, peer-reviewed. Audit trails — CloudTrail + VPC Flow Logs + GitHub audit log, shipped to tamper-proof S3 with object lock. Automated compliance checks — pre-commit hooks and CI checks validate Terraform plans (encryption enabled, public access blocked, logging enabled). Access reviews — quarterly IAM reviews with automated reports."

Q 22 Explain how the HashiCorp Vault + GitHub OIDC integration works technically.

HOW TO SAY IT

"Step by step: 1) GitHub Actions starts, GitHub mints a short-lived OIDC JWT with claims (repo, branch, environment, workflow). 2) Workflow sends JWT to Vault's JWT/OIDC auth backend. 3) Vault validates against GitHub's OIDC discovery endpoint. 4) Vault checks claims against a bound role — this role only accepts tokens from repo X, branch main, environment production. 5) Vault issues short-lived token with policies controlling secret access. 6) Workflow uses token to read secrets or assume AWS IAM role via Vault's AWS secrets engine. No long-lived credentials anywhere."

---

6. Behavioral & Leadership

At the architect level, they evaluate whether you can drive decisions, handle ambiguity, and influence teams. Use STAR format strictly.

Q 23 Tell me about a time you had to convince a team to adopt a new technology.

YOUR STAR STORY — JENKINS TO GITHUB ACTIONS

Situation: Teams comfortable with Jenkins. Some had invested years in custom Groovy libraries. Resistance to change.

Task: Get buy-in from 60+ pipeline owners.

Action: Did not mandate. Picked two early-adopter teams, migrated first, documented results. Presented before/after metrics at tech all-hands. Offered hands-on support: 'I'll pair with your team for a day.' Ensured templates replicated all Jenkinsfile functionality.

Result: Within 3 months, every team voluntarily migrated. No mandates needed.

PRO TIP

This shows data-driven decision making, empathy for affected teams, and leading through influence rather than authority.

Q 24 Describe a situation where a migration or deployment went wrong.

HOW TO STRUCTURE

Pick a real incident. Structure: Situation — "During the ECS Fargate cross-account migration, we discovered KMS-encrypted S3 objects couldn't be read in target account." Task — "Fix without data loss or delaying migration window." Action — "Updated KMS key policy, re-ran sync, validated checksums. Set up pre-migration checklist." Result — "Resolved within [time]. Added check to runbook. Never recurred."

PRO TIP

End with what you changed, not just what you fixed. "What did you change so this never happens again?" — this shows systems thinking.

Q 25 How do you mentor engineers and build platform adoption across teams?

YOUR STAR STORY

Societe Generale: Led 6+ developers. Mentored on Angular, Node.js, backend integration. Improved team velocity by **35%** through pairing sessions, code review standards, design pattern workshops.

Takeda: Built self-service platform teams could adopt independently. Created documentation, example pipelines, ran office hours. Adoption hit 100% without mandates.

HOW TO SAY IT

"The best platform engineers make themselves unnecessary. If teams need me to deploy their service, the platform has failed. If they can onboard themselves in 2 hours — that is success."

---

7. Additional Topics

These questions fill gaps in the core 25. Companies increasingly ask about technical debt, database trade-offs, API strategy, DR planning, and stakeholder communication.

Q 26 How do you prioritize and address technical debt alongside feature work?

HOW TO SAY IT

"I frame technical debt as a business conversation, not a tech complaint. At Takeda, the Jenkins-to-GitHub Actions migration was fundamentally a technical debt initiative. I built the case by showing: maintenance cost of 60+ fragmented Jenkinsfiles, hours lost to flaky builds, security risk of static credentials. I proposed it alongside a feature epic — migration happened in parallel, not instead of delivery. I tied every refactoring initiative to a measurable outcome: build time reduction, engineering hours saved, security posture improvement. Leadership approved because the ROI was clear."

PRO TIP

Never frame technical debt as "we need to clean up code." Frame it as "this costs us X hours per month and creates Y risk." Business language gets budget.

Q 27 When would you split a database? How do you choose between SQL and NoSQL?

HOW TO SAY IT

"I consider splitting a database when write contention, hot partitions, or team autonomy bottlenecks emerge. At Takeda during the monolith-to-microservices migration, the shared database was the hardest part. I used bounded contexts from DDD to define ownership boundaries. Migration approach: introduce a data access layer first, then use dual writes (or CDC via DMS) to migrate one context at a time with verification and rollback plans. For the new services: PostgreSQL (RDS multi-AZ) for transactional data where consistency matters, and DynamoDB for high-throughput read-heavy workloads. The choice always depends on access patterns, consistency requirements, and team familiarity."

Q 28 How would you design an API platform for internal and external consumers?

HOW TO SAY IT

"At the platform level, API design is about developer experience and governance. I would set up an API gateway (AWS API Gateway or Kong) for rate limiting, authentication, versioning, and monitoring. Internal APIs use gRPC or REST depending on performance needs. External APIs always use REST with OpenAPI specs. Versioning via URL path (/v1/, /v2/) for external, header-based for internal. Every API gets generated documentation from the spec, contract testing in CI, and automated backward-compatibility checks before deployment. The goal: a developer from another team can discover, understand, and integrate with your API in under 30 minutes without asking anyone."

Q 29 Design a disaster recovery strategy spanning multiple AWS regions.

HOW TO SAY IT

"I always start with business requirements: what is the RTO and RPO? For Takeda, pharma workloads needed sub-1-hour RTO and near-zero RPO. Architecture: active-passive across two regions. Primary region runs all traffic. Secondary region has warm standby — EKS cluster running, RDS read replica with async replication, S3 cross-region replication enabled. Route 53 health checks detect primary failure and auto-switch DNS. For truly critical services: active-active with DynamoDB global tables and application-level conflict resolution. I always run DR drills quarterly to validate the failover actually works — untested DR is not DR."

PRO TIP

"Untested DR is not DR" is a great line to use in interviews. It shows operational maturity.

Q 30 How do you communicate technical decisions to non-technical leadership?

HOW TO SAY IT

"At Takeda, I regularly presented to leadership. The key is translating technical complexity into business impact. For the AWS cost optimization initiative, I did not present instance types and pricing models. I presented: here is what we spend, here is what we could spend, here is the risk profile of each option, and here is my recommendation. One slide with a before/after cost chart and three bullet points. For the Jenkins-to-GitHub Actions migration, I framed it as: current state costs us X hours/month in maintenance and carries Y security risk. Proposed state delivers Z improvement. Every technical decision gets a one-page business case before I bring it to leadership."

PRO TIP

If asked to present a past project (some companies do this as a separate round), use this structure: Problem (1 min), Approach (3 min), Architecture with diagram (5 min), Results with metrics (2 min), Lessons learned (2 min). Keep it to 15 minutes max.

Quick Reference — Your Key Metrics

Have these numbers ready. Interviewers remember specifics.

Achievement	Before	After	Impact
Jenkins to GitHub Actions	60+ fragmented pipelines	Unified platform	30% faster, 200+ hrs/mo saved
Service onboarding	2 weeks	2 hours	98% reduction
iOS build time	45 minutes	15 minutes	67% reduction
Mobile release cycle	2 days	15 minutes	99% reduction
Static credentials in CI/CD	Multiple long-lived keys	Zero	100% via Vault OIDC
Artifactory migration	On-premises	Cloud	40% faster, zero downtime
AWS cloud spend	Baseline	Optimized	25% monthly reduction
Team velocity (SocGen)	Baseline	Improved	35% increase
Impact tool adoption	0	10+ business units	60% faster assessment
Monitoring coverage	Partial	Full stack	50+ services covered
EKS deployment success	Variable	Consistent	99.9% success rate

Interview Day Reminders

→ Always start with requirements before designing. Ask: 'What's the RTO/RPO? Traffic pattern? Team size?'

→ Draw on the whiteboard. Visual thinkers get hired at the architect level. Practice drawing your NLB to ALB to EKS to RDS diagram until it is muscle memory.

→ Lead with metrics. '60+ pipelines, 30% faster, 2 hours not 2 weeks' — these stick in an interviewer's mind.

→ When you do not know something, say so honestly and explain how you would find out. Architects navigate ambiguity.

→ End every answer with impact. Not 'I set up Vault OIDC' but 'I eliminated 100% of static credentials, reducing attack surface.'

→ Your mobile CI/CD experience is rare. Do not downplay it.

→ For behavioral questions, always end with what you changed to prevent recurrence.

→ Prepare a 15-minute project presentation of the Jenkins-to-GitHub Actions migration in case they ask for one.

→ Research the company's tech stack before each interview. Tailor your examples to match their tools and challenges.