Interview Questions for CVS/Aetna Solution Architect Role

# Medium Difficulty Questions

## Technical – API Design

**Question:**

* How would you design an API gateway strategy for the CVS SuperApp to handle multiple backend services, versioning, and authentication?

**Possible Answers:**

* Use AWS API Gateway with multiple stages (dev, test, prod), integrate Lambda authorizers for auth, and deploy per service using REST/HTTP APIs.
* Include API versioning in the URL (e.g., /v1/resource) or headers.
* Use throttling, caching, and custom domain mappings.

**Bad Answer:**

* Anything that exposes everything through an API endpoint or ignores versioning and security needs.

**Best Architecture:**

* **API Gateway per domain** → separates concerns, allows granular throttling.
* **Lambda authorizers** → decouples auth logic.
* **CloudFront in front of API Gateway** → for caching and global acceleration.

**Worse Architectures:**

* **Single monolithic API Gateway** → tight coupling, no domain separation.
* **Direct access to backend services** → no unified control plane, no throttling.

**Follow-up:**

* What trade-offs would you consider when using GraphQL instead of REST for this architecture?

## Procedural/Process – Regulatory Design

**Question:**

* How would you ensure ADA and HIPAA compliance are addressed in the development lifecycle for the SuperApp? I do not expect you to know the specifics of the regulatory environment…just speak the problem space.

**Possible Answers:**

* Embed compliance reviews into sprint planning.
* Use automated tools like Axe for accessibility testing, static analysis for PHI handling, and integrate with CI/CD.
* Partner with legal/compliance teams in design phases.

**Bad Answer:**

* Where the test is at the end or hands the problem off to a compliance officer.

**Follow-up:**

* How would you enforce these checks across multiple teams and services in an Agile release train?

## Interpersonal – Team Collaboration

**Question:**

* How do you handle disagreements between a backend developer and a frontend developer over API design that could delay delivery?

**Possible Answers:**

* Facilitate a quick design workshop, clarify requirements, and mediate a decision.
* Document trade-offs and escalate if needed.
* Use API standards and shared contracts (e.g., OpenAPI) as a guide.

**Bad Answer:**

* Answers that ignore mediation or shunt the issue off to the developers to settle rather than exercising senior architects’ expertise and authority in decision making.

**Follow-up:**

* How would you reduce future conflicts?

# Difficult Questions

## Scalability – Elastic Design on AWS

**Question:**

* Describe a scalable, elastic architecture for a SuperApp feature like "Benefits Lookup" that spikes during open enrollment (Nov–Dec) using AWS.

**Possible Answers:**

* Use **API Gateway + Lambda** for request handling.
* **Aurora Serverless or DynamoDB** for database.
* **SQS for decoupling**, **CloudWatch Alarms + Auto Scaling Groups** for backend workloads.
* Cache static content in **CloudFront**, use **WAF** for security.

**Bad Answer:**

* Simple EC2 scaling or solutions that ignore open enrollment traffic patterns.

**Best Architecture:**

* **Serverless-first design (Lambda, DynamoDB, SQS)** → pay-per-use, elastic.
* **Aurora Serverless** → if relational data required.

**Worse Architectures:**

* **Monolithic EC2 instances behind an ALB** → higher cost, manual scaling.
* **No caching** → increased latency.

**Follow-up:**

* What gotchas might we find with Lambda or API Gateway in this architecture (replace Lambda/API with their solution components)?

## API Aggregation

**Question:**

* How would you design an API aggregation layer that consumes multiple backend services with differing SLAs and authentication mechanisms?

**Possible Answers:**

* Use an **API Gateway + Lambda** pattern for aggregation.
* Implement **circuit breakers, retries, timeouts**.
* Normalize authentication via **token exchange** or **API proxy layer**.

**Bad Answer:**

* Any answer that does not include a gateway pattern, but keep an ear out for creative alternatives. Deep design like circuit breakers and timeouts are a bonus, but not required for a satisfactory answer.

**Follow-up:**

* Why not use a GraphQL federation for this layer? If they suggested this, skip the follow up.

**Regulatory Guardrails**

**Question:**

* How would you enforce PHI/PII masking and encryption in the SuperApp architecture, considering multiple microservices?

**Possible Answers:**

* Centralize sensitive data handling via shared libraries or API Gateway filters.
* Use **AWS Macie**, **KMS**, **GuardDuty**.
* Encrypt at rest with KMS, in transit with TLS.

**Bad Answer:**

* Simple encryption solutions. They do not suffice because of the many opportunities for data slippage. Answer should have some form of post-hoc surveillance but listen for other creative approaches.

**Follow-up:**

* How would you ensure consistent application of masking across teams?

## API Chaining Latency

**Question:**

* What is the impact of API chaining in the backend, and how would you reduce end-to-end latency?

**Possible Answers:**

* Discusses **fan-out vs. fan-in**, **aggregation**, and **parallelization**.
* Understands failover and retry implications
* Suggests **event-driven architecture (SNS/SQS/Kinesis)** to decouple calls.
* Includes root-cause handling and exception management

**Bad Answer:**

* Addresses response velocity but not include mid-chain failure.
* Recommends aggregation. Be careful, aggregation can be an effective strategy at times. Do not prejudge.

**Follow-up:**

* Would you recommend AWS AppSync for this use case? Why or why not?

**Inter-Org Design Decisions**

**Question:**

* How would you approach architectural decisioning when Aetna’s design choices (e.g., APIs, data models) differ from CVS’s, but the SuperApp must integrate both?

**Possible Answers:**

* Facilitate architectural governance forums.
* Use **anti-corruption layers**.
* Define **shared contracts** or **experience APIs** to decouple.

**Bad Answer:**

* Answers that do not get at the meat of the design disconnect.
* Answers that assume more control over partner approach than is typically available in challenges of this nature.
* Answers that do not show familiarity with concordance

**Follow-up:**

* How would you handle conflicts in data definitions across both organizations?

## Experience API vs. GraphQL

**Question:**

* When would you choose an Experience API layer over GraphQL for a consumer-facing healthcare portal? Describe trade-offs.

**Possible Answers:**

* Experience APIs: simpler, clear ownership, caching. Often better for the C U and D of CRUD
* GraphQL: flexible, single endpoint, but risk of over-fetching/under-fetching and complexity in authorization.

**Bad Answer:**

* Says “GraphQL is always better” or “REST is always better” without context.
* GraphQL without understanding project overhead/complexity of synthetic transfer object management

**Follow-up:**

* Would GraphQL be a good fit for a high-traffic, heavily cached healthcare search feature? Why or why not?

## Past Experience and Reflection (Two Questions)

* **Q1:** Describe a past project where you built a complex API ecosystem. What did you learn about balancing scalability, security, and delivery speed?
* **Q2:** Share an experience where a major design change was introduced mid-project. How did you adapt, and what trade-offs were made?

**Follow-ups (for both):**

* What would you do differently today?
* Why didn’t you choose an alternative approach at the time?

## DevOps Question

**Question:**

* How would you ensure architecture principles like DDD, security, and compliance are enforced in a CI/CD pipeline for the SuperApp?

**Possible Answers:**

* Use **linting, static analysis, IaC templates**, **SonarQube**, **Open Policy Agent**.
* Integrate **security scans (Snyk, CodeQL)**.
* Enforce reviews in PRs.
* Better if they have some DevOps specific approaches.
* Even better if it is AWS oriented.

**Bad Answer:**

* Listen for an answer implying the interviewee has not thought about or encountered this issue in prior engagements. Not a deal breaker. Walk them through at this point to see if they “get it”.

**Follow-up:**

* What challenges do you foresee with this approach in a multi-team environment?

# Harder Design Question

**Question:˙**

* You have a SuperApp feature for prescription refills using a REST API, with caching at CloudFront. The business now requires **real-time pharmacy inventory visibility**. How would you modify the architecture to support this requirement while maintaining performance?

**Possible Answers:**

* Add a **real-time event stream (Kinesis, DynamoDB Streams)** to push inventory updates.
* Update caching policies to differentiate between static and dynamic data.
* Consider **WebSocket or GraphQL subscriptions** for real-time needs.

**Bad Answer:**

* Listen for solutions that do not modify existing architectures. Something like “create a new capability” or “rework the existing framework”. These are obvious examples but they make the point. The idea here is to modify the existing architecture, not create a new one.

**Follow-up:**

* How would you ensure real-time data doesn’t degrade system performance during high-traffic periods?