Comprehensive Technical Interview Questions

Prepared for: Bharath R

Table of Contents

Resume-Based Technical Questions

1. Distributed Transactions:

Question: You implemented the SAGA pattern with Azure Service Bus. Can you walk me through how you ensured data consistency across microservices?

Expected Answer: The candidate should describe:

- Implementation of compensating transactions for rollback operations
- How they used Azure Service Bus topics/queues for orchestration
- Message correlation and tracking mechanisms
- How they handled failure scenarios and retries
- Monitoring and observability of transaction flows

2. Spring State Machine:

Question: How did you implement multi-step approval workflows using Spring State Machine, and what specific event sourcing patterns did you use?

Expected Answer: Look for:

- Definition of states and transitions in the approval workflow
- Event persistence strategy
- How they implemented event sourcing for audit trails

- Recovery mechanisms from failed state transitions
- How they achieved the 40% improved process visibility

3. **Redis Cache Implementation:**

Question: Describe how you integrated Redis Cache with your Spring Boot reservation management system to achieve the 25% increase in throughput.

Expected Answer: The candidate should explain:

- Cache strategies (write-through, write-behind, cache-aside)
- TTL considerations for different types of data
- Redis data structures used (hash, sorted sets, etc.)
- Cache invalidation approach
- Performance benchmarking methods used to measure the improvement

4. **Azure Service Bus Integration:**

Question: What challenges did you face when implementing retry policies and circuit breakers for third-party integrations?

Expected Answer:

- Implementation details of retry patterns (exponential backoff, etc.)
- How they configured circuit breakers (thresholds, recovery time)
- Monitoring of circuit breaker states
- Dead-letter queue handling
- Integration testing approaches

5. **Microservices Migration:**

Question: You migrated 80% of offline workflows to online. What was your approach and what metrics did you use to measure the 50% productivity increase?

Expected Answer:

- Migration strategy (strangler pattern, etc.)
- How they prioritized which workflows to migrate
- Integration points between offline and online systems
- Performance and user experience metrics tracked
- How they quantified productivity improvements

6. **Observability Setup:**

Question: How did you configure Azure Monitor, Application Insights, and Grafana together for comprehensive monitoring?

Expected Answer:

- Data collection and correlation strategy
- Custom metrics implementation
- Alert configuration and management
- Dashboard organization in Grafana
- Integration between Azure Monitor and third-party tools

7. **Azure Blob Storage:**

Question: Describe a specific use case where you leveraged Azure Blob Storage in your current role.

Expected Answer:

- Business requirement that led to using Blob Storage
- Data organization strategy (containers, naming)
- Access control implementation
- Performance optimization techniques
- Integration with other Azure services

8. **Cloud-Native Assessment Platform:**

Question: What security measures did you implement beyond JWT authentication in your assessment platform?

Expected Answer:

- Role-based access control details
- Data encryption at rest and in transit
- Input validation and output encoding practices
- Session management approach
- Audit logging implementation

9. **Infrastructure Automation:**

Question: What specific Terraform practices did you follow in your automation pipeline that led to the 40% reduction in deployment cycles?

Expected Answer:

- Module organization and reusability
- State management approach
- Pipeline integration details
- Testing strategy for infrastructure code
- Measurement methodology for deployment cycle reduction

10. SAGA Pattern Application:

Question: Can you provide a specific example of a distributed transaction you handled with the SAGA pattern and how you managed failure scenarios?

Expected Answer:

- Concrete business transaction example
- Decomposition into steps
- Compensation transaction design
- How they ensured idempotency
- Timeout handling and recovery mechanisms

AKS (Azure Kubernetes Service)

1. How do you perform rolling updates in AKS without downtime?

Answer:

In one project, we deployed microservices using rolling updates by configuring the `Deployment` strategy in Kubernetes. We used `readinessProbes` to ensure a pod doesn't receive traffic before it's ready. If a new pod fails the readiness check, it blocks the rollout and prevents old pods from being terminated. We also monitored the rollout using `kubectl rollout status` and tied deployments to our CI/CD pipeline to verify health post-deployment.

2. What steps would you take if a node in your AKS cluster is running out of memory?

Answer:

First, I'd check with `kubectl top nodes` and `kubectl top pods` to identify the memory usage pattern. Then, I'd review the pod specs to see if the `resources.requests` and `resources.limits` are appropriately set. If a pod is consuming unbounded memory, I set stricter limits and consider vertical pod autoscaling or distribute pods across nodes using `affinity/anti-affinity` rules. Long term, I propose increasing node size or scaling out the cluster.

3. How do you handle certificate rotation in AKS ingress controllers (e.g., NGINX)?

Answer:

We use Azure Key Vault with certs linked via `cert-manager` and auto-sync into AKS using an ingress annotation. When the certificate in Key Vault is updated, `cert-manager` picks it up, renews the Kubernetes TLS secret, and updates the ingress resource. This setup ensures zero downtime and avoids manual redeploys during cert rotation.

4. Describe a scenario where Kubernetes readiness and liveness probes saved your production service.

Answer:

In production, a Java-based service entered a deadlock due to thread pool saturation. The `readinessProbe` kicked in, pulling the pod from the load balancer. Simultaneously, the `livenessProbe` restarted the pod after detecting the service was unresponsive. This allowed us to recover automatically without human intervention. Without probes, users would have faced 500 errors.

5. How do you manage Kubernetes manifests for multiple environments (dev, staging, prod)?

Answer:

We use a GitOps approach with Kustomize or Helm for environment-specific overlays. Each environment has its own values or overlay directory, and the base manifests are reused. During deployment, our pipeline dynamically picks the correct values or overlays depending on the environment, ensuring consistent deployments and reducing configuration drift.

Container Images & Pod Deployment

6. How do you optimize container images for faster deployment and lower attack surface?

Answer:

We use multi-stage builds to reduce image size and remove build-time dependencies. For Java apps, we use `distroless` or `Alpine` as the base image. This cuts down vulnerabilities and speeds up pull and start times. We also scan images using Trivy or Azure Defender to ensure compliance before deployment.

7. What happens if the pod crashes during initialization? How do you debug it?

Answer:

I use `kubectl describe pod` and `kubectl logs <pod-name> -c <init-container>` to check init container logs. One time, a database migration init container failed due to a missing DB endpoint in secrets. We added retry logic and improved secret loading using the Kubernetes Downward API. Using `restartPolicy: Never` helped identify issues quickly during testing.

8. How do you handle secret injection into pods?

Answer:

We mount secrets from Kubernetes Secrets or integrate with Azure Key Vault via CSI driver. For apps using environment variables, we map secrets directly. In one setup, rotating secrets in Key Vault automatically refreshed secrets in pods via CSI, avoiding redeployments and improving security posture.

9. How do you implement blue-green deployment with Kubernetes?

Answer:

We deploy a new version in parallel with the existing one under a different service selector (e.g., `app: my-app-blue`). After validation, we update the production service to point to the new pods. If anything breaks, we simply switch traffic back to the old deployment (`green`). Azure DevOps gates ensure this flow is controlled and monitored.

10. How do you manage image pull failures in AKS?

Answer:

Image pull failures usually stem from wrong credentials or incorrect tags. I use Azure Container Registry (ACR) with managed identity authentication (`aks-managed-identity`). We enable imagePullSecrets via AAD Pod Identity or by linking ACR with AKS using `az aks update`. In one case, an image tag was misconfigured, so we added tag validation in our pipeline.

Azure Key Vault & Authentication

11. How does your application authenticate to Key Vault in AKS?

Answer:

We use **Managed Identity**. The AKS cluster is assigned a system-assigned identity. Then, we assign access policies or RBAC permissions to that identity on the Key Vault. Our Spring Boot app uses Azure SDK to authenticate via `DefaultAzureCredential`, which uses the managed identity token for secure access—no secrets embedded in code.

12. How do you rotate secrets stored in Key Vault without restarting the app?

Answer:

Using Key Vault references in Azure App Configuration, our app retrieves secrets at runtime. In AKS, we mount secrets via the Key Vault CSI driver, which polls for changes periodically. A pod annotation change triggers a rolling update so that the latest secret is used without full downtime.

13. Have you ever used RBAC instead of access policies in Key Vault? Why?

Answer:

Yes. RBAC offers better consistency across Azure resources and simplifies permission auditing. In one multi-team project, switching to RBAC allowed fine-grained access via Azure AD groups instead of manual access policy entries. This made onboarding/offboarding seamless and improved security compliance.

14. What's your backup plan if Key Vault becomes temporarily unavailable?

Answer:

We cache secrets locally in memory or an encrypted file. We configure exponential retries in Azure SDKs and alerting in place if secrets can't be fetched. In one case, we fell back to local encrypted copies while Azure resolved a Key Vault outage, which helped maintain uptime.

15. How do you control access to only specific secrets in a large vault?

Answer:

We use RBAC roles like `Key Vault Secrets User` and scope them to only the necessary secrets via Azure Key Vault versioned access control. Tagging secrets helps categorize access, and we regularly audit with Azure Policy and Access Reviews to ensure principle of least privilege.

Azure Service Bus

16. How do you authenticate apps with Azure Service Bus in production?

Answer:

We use **Managed Identity** and assign the role `Azure Service Bus Data Sender` or `Data Receiver` at the namespace level. Apps authenticate via Azure Identity SDK without storing connection strings. This approach removes secrets from configs and integrates smoothly with Azure RBAC.

17. How do you process messages idempotently to avoid duplicates?

Answer:

Each message carries a unique ID or business key. Before processing, we check Redis or DB for processed message IDs. If found, we skip. We also configure `MaxDeliveryCount` to control retries and avoid infinite re-processing loops. This is critical for payment or booking microservices.

18. How do you handle poison messages in queues?

Answer:

Messages that exceed retry limits are moved to the dead-letter queue (DLQ). We monitor the DLQ with alerts and have a custom tool to reprocess or inspect those messages. In one case, an invalid JSON format caused deserialization errors. We updated schema validation to fail fast and dead-letter early.

19. How do you scale Service Bus consumers?

Answer:

We deploy consumers as Kubernetes pods and use **horizontal pod autoscaling** based on queue length metrics exposed via Prometheus. We also optimize concurrency using `maxConcurrentCalls` in the client SDK. This helps during seasonal spikes or product launches.

20. How do you track end-to-end message flow in a distributed system using Service Bus?

Answer:

We attach a `correlationId` to each message. Logs across services use this ID for traceability. We

use Application Insights or Azure Monitor to visualize message journey across consumers. In one issue, it helped us debug message delay caused by one consumer running out of threads.

Azure Blob Storage & Authentication

21. How do you authenticate apps accessing Blob Storage?

Answer:

We use **Managed Identity** with role `Storage Blob Data Contributor` assigned at container level. Apps authenticate using Azure Identity SDK. No need for storage keys or SAS tokens. This approach works well inside AKS and improves auditability via Azure AD logs.

22. How do you securely share blob access externally?

Answer:

We generate **SAS tokens** with precise permissions and expiry. For example, a read-only SAS link valid for 30 minutes is shared with a customer for download. Tokens are generated server-side and never exposed in frontend code. We log every access via storage analytics.

23. How do you handle file uploads in Spring Boot to Blob Storage?

Answer:

We use `Azure Blob Storage SDK` and authenticate using `DefaultAzureCredential`. Files are streamed to the target container using `BlockBlobClient`. We validate file type, size, and virus-scan before upload. Upload metadata is stored in CosmosDB for later retrieval.

24. How do you monitor and manage blob lifecycle?

Answer:

We use **Azure Blob Lifecycle Management** rules to auto-delete or tier blobs after a certain time. For backups, blobs are moved to cool/archive tiers. Alerts notify us if blob size grows unexpectedly, and we audit access via logs integrated into Log Analytics.

25. How do you version or back up blobs in a multi-tenant app?

Answer:

Each tenant has a logical folder path in blob storage (`tenant-id/documents`). We enable **Blob**

Versioning, so any overwrite stores a new version. This helps in rollback or compliance checks. We also generate SAS links for auditors to access specific versions without modifying content.

Additional Technical Scenarios

26. How do you ensure secrets stored in Azure Key Vault are rotated and picked up by your AKS application automatically?

Answer:

To automate secret rotation and consumption:

- Use **Azure Key Vault event triggers** with **Event Grid** to notify a logic app or Azure Function on secret change.
- The logic app can update the Kubernetes secret via `kubectl` or call an internal API in the AKS app to reload configuration.
- Use a **CSI Secret Store driver** with Key Vault integration. It allows auto-sync of secrets into pods, reducing the need for redeployment.
- Monitoring is set up with Application Insights and alerts if secrets are about to expire.

27. Your pod fails with an image pull error. What are the steps to troubleshoot and resolve it?

Answer:

1. Check `kubectl describe pod <pod-name>` for imagePullBackOff or ErrImagePull.
2. Verify image name, tag, and registry are correct.
3. Ensure the image is publicly accessible or the correct **imagePullSecrets** are configured.
4. If using ACR, verify the AKS cluster has **proper access via managed identity** or service principal.
5. For private registry: create a Docker registry secret and attach it to the service account.

28. Describe a real-world use case where you used Azure Blob Storage in a containerized microservice setup.

Answer:

In our onboarding app, user-uploaded documents (PDFs, images) are sent from frontend to a backend AKS pod.

- The pod generates SAS tokens for secure uploads.

- Files are stored in a container with a structured path: `users/<user-id>/documents/`.
- Blob events trigger an Azure Function to scan for viruses and tag metadata.
- Access is managed via **Azure AD authentication** for backend pods using managed identities.

29. **How do you version and rollback container images in AKS during blue-green deployments?**

Answer:

- All builds push versioned images to ACR: e.g., `myapp:v1.0.0`, `v1.0.1`.
- Blue-green strategy uses two deployments (blue and green). Only one serves traffic via a stable service label.
- Rollback is as simple as switching the service selector to the previous deployment or reapplying a previous manifest with the older image.
- Canary testing is optionally done before full switch.

30. **How does authentication work when accessing Azure Blob Storage from a pod in AKS?**

Answer:

- Use **Azure Workload Identity** to assign a user-assigned managed identity to the pod.
- The pod uses Azure SDKs (e.g., `DefaultAzureCredential`) which pick up the identity.
- Identity is granted **Storage Blob Data Contributor** role in RBAC for secure access.
- This avoids using access keys or connection strings in secrets.

31. **Your service needs to access Key Vault and Blob Storage securely. How do you configure authentication?**

Answer:

- Use **Azure Workload Identity** or **AAD Pod Identity** to assign a managed identity to the pod.
- Assign roles:
  - **Key Vault Secrets User** for Key Vault.
  - **Storage Blob Data Contributor** for Blob.
- SDKs use identity to authenticate (no secrets or client secrets are stored).
- All configurations use `DefaultAzureCredential` so devs can run locally via `az login`.

32. **How do you configure private container registries like ACR with AKS in a secure way?**

Answer:

- Use **managed identity** of AKS to authenticate to ACR via Azure RBAC.
- Run `az aks update` to attach ACR to AKS.
- Avoid using imagePullSecrets with credentials.
- Validate access by pulling a private image using `kubectl run`.

33. **Describe how you handled scaling pods in AKS for a high-throughput Service Bus consumer.**

Answer:

- Created an HPA (Horizontal Pod Autoscaler) based on custom metrics like Service Bus message count.
- Used **KEDA (Kubernetes Event-driven Autoscaler)** to scale pods based on queue length.
- KEDA watches the queue using a trigger and scales from 0 to N as needed.
- Pods used **managed identity** to connect securely to the queue.

34. **Your AKS application is getting unauthorized errors while accessing Service Bus. How do you resolve it?**

Answer:

- Confirm pod has the correct **workload identity** or service principal.
- Check the role assignment: should have **Azure Service Bus Data Sender/Receiver** on the queue/topic.
- Verify the identity is using correct SDK auth path like `DefaultAzureCredential`.
- Check if token expiration or network firewall rules block access.

35. **What are some best practices for deploying sensitive configuration in AKS pods?**

Answer:

- Store secrets in Azure Key Vault, not in config maps.
- Use CSI driver or init containers to fetch secrets at runtime.
- Enable **RBAC** and **PodSecurityPolicy** to restrict access.
- Ensure secrets are mounted read-only and do not log sensitive info.

36. **How do you monitor container logs and app metrics in AKS?**

Answer:

- Enabled **Azure Monitor for Containers**.
- Integrated **Application Insights SDK** in Java services for custom events and traces.
- Used `kubectl logs`, Prometheus, and Grafana dashboards for fine-grained metrics.
- Alerts are configured on resource usage, error logs, and latency SLAs.

37. How would you transfer large files securely between your AKS app and Azure Blob Storage?

Answer:

- Use SAS tokens with expiration and specific permissions.
- Transfer files via client-side or backend-streaming.
- For large files, use **block blob** strategy with parallel uploads.
- Validate file type and size before processing. Virus scanning via event-based triggers.

38. What happens when a container keeps restarting due to crash loop? How do you debug it?

Answer:

- Run `kubectl describe pod` to check reason (`CrashLoopBackOff`).
- Fetch logs using `kubectl logs <pod-name> --previous` to see failing logs.
- Validate:
  - Resource limits (memory/CPU)
  - Dependency services are reachable
  - Configs and secrets are correct
- Use liveness/readiness probes properly to avoid killing healthy pods.

39. How do you deploy a Java Spring Boot container on AKS with environment-based configurations?

Answer:

- Use Helm charts with values.yaml for environment-specific values.
- Spring profiles (`-Dspring.profiles.active`) are passed as arguments in deployment.
- Secrets and config maps injected as env vars or mounted volumes.
- Images tagged per environment like `myapp:prod`, `myapp:qa`.

40. **Your AKS app connects to a premium Azure Service Bus topic. How do you ensure high availability and retries?**

Answer:

- Enable message sessions and duplicate detection.
- Configure retry policies in SDK.
- Use **geo-disaster recovery** by aliasing the namespace.
- Auto-retry logic built into Spring Boot listener. Messages are dead-lettered after N retries.

41. **How do you secure Azure Key Vault access when using GitHub Actions to deploy to AKS?**

Answer:

- GitHub OIDC is configured with a federated credential in Azure AD.
- Action uses `azure/login` to fetch access token.
- Key Vault secrets are fetched via CLI or SDK during deployment.
- No secrets are stored in GitHub or pipeline YAML.

42. **How do you avoid downtime during pod rollouts in production?**

Answer:

- Use **rolling updates** with `maxUnavailable` set to 0.
- Health probes (readiness/liveness) ensure only ready pods serve traffic.
- Use **PodDisruptionBudgets** to control voluntary disruptions.
- Service has stable selector pointing to updated pods.

43. **How do you implement access tiering and lifecycle for Azure Blob data used in AKS?**

Answer:

- Define **lifecycle management rules** to move blobs to Cool or Archive after 30/90 days.
- Access tier is set per blob: Hot (default), Cool, Archive.
- Policies reduce storage cost automatically.
- Application tracks last access time for usage-based tiering.

44. **How do you restrict access to a specific Service Bus topic from a single AKS namespace?**

Answer:

- Create a **user-assigned managed identity** and bind it to that namespace.
- Assign **Azure Service Bus Data Sender/Receiver** roles only to the topic.
- Use Kubernetes RBAC to restrict which services can use that identity.
- Validate token access with audit logs.

45. What is the use of initContainers in AKS deployment scenarios?

Answer:

- Run a pre-task before main container starts, e.g.:
  - Wait for database to be ready.
  - Fetch secrets from an external source.
  - Run schema migration jobs.
- Helps separate concerns and keeps main container lean.

46. How do you configure blob change notifications to trigger downstream services?

Answer:

- Enable **Event Grid integration** on Blob Storage.
- Configure event subscription for `BlobCreated`, `BlobDeleted`.
- Event triggers Azure Function or Service Bus queue.
- Use metadata to route processing logic.

47. Describe how you use labels and annotations to organize AKS resources.

Answer:

- Labels: for selectors, grouping (`env=prod`, `team=payments`).
- Annotations: for metadata (`prometheus.io/scrape: true`).
- Used in deployment pipelines and monitoring tools.
- Labels are used in NetworkPolicies and HPA targeting.

48. How do you handle retries for failed Service Bus message processing in pods?

Answer:

- SDK retry policy with exponential backoff.

- Manual retry logic with DLQ (dead letter queue) after N failures.
- Health check for queue lag triggers autoscaling via KEDA.
- Alerts notify ops if queue length exceeds threshold.

49. **Explain how pod-to-pod secure communication is achieved in AKS.**

Answer:

- Use **NetworkPolicies** to allow traffic only between allowed pods/namespaces.
- Enable **TLS mutual authentication** if needed.
- Use service mesh (e.g., Istio or Linkerd) for advanced mTLS.
- Restrict egress traffic using Azure NSGs.

50. **What steps do you take to clean up stale container images in ACR and AKS?**

Answer:

- Enable **ACR retention policy** to delete untagged images after X days.
- Use a GitHub Action or script to delete images older than N versions.
- Avoid `latest` tag in production to prevent ambiguity.
- Clean up unused pods and images via `kubectl rollout restart`.