SRE Leadership Interview Q&A Guide

Background & Troubleshooting Experience

Background: I have X years of experience in Site Reliability Engineering with a focus on production system troubleshooting, incident response, and leading cross-functional teams during critical outages. My background spans cloud-native applications, Kubernetes orchestration, database systems, and network infrastructure. I've led incident response for mission-critical systems processing millions of transactions daily and have experience coaching development teams on reliability best practices.

Troubleshooting Approach: My troubleshooting methodology follows a systematic approach: gather symptoms, form hypotheses, test systematically, and document findings. I believe in understanding the full system context before diving into specific components, which helps identify root causes faster than random troubleshooting.

Production Incident Response & Leadership

Taking Control During Production Issues

My Approach:

1. **Immediate Assessment (First 2 minutes):**
   - Gather basic symptoms: what's broken, since when, impact scope
   - Establish communication channels and ensure proper incident commander is present
   - Quickly assess if this is a known issue with existing runbooks
2. **Systematic Investigation:**
   - Start with monitoring dashboards and recent changes (deployments, config changes)
   - Follow service dependencies from user-facing symptoms backward
   - Check infrastructure health: network, compute, storage, database
3. **Leading the Response:**
   - I'm absolutely comfortable taking control when needed, especially when I see the investigation losing focus
   - Establish clear roles: who investigates what, who communicates with stakeholders
   - Set regular check-in intervals to avoid rabbit holes

Managing Large Incident Calls

Eliminating Unnecessary Participants:

When dealing with 60-70 person calls, I use this approach:

- **Quick Roll Call:** At 5-minute mark, ask each team to state their role and current investigation
- **Active Dismissal:** "Database team - we've confirmed DB is healthy, you can drop off but stay on standby"
- **Focused Breakouts:** Split into smaller investigation groups with clear owners reporting back to main bridge
- **Regular Pruning:** Every 15 minutes, ask "Who needs to stay for the next phase of investigation?"

Determining Right Teams to Contact:

I use a decision matrix:

1. **Immediate Impact Teams:** Those needed to restore service immediately
2. **Investigation Teams:** Those who own potentially affected components
3. **Communication Teams:** Leadership and customer communication
4. **Subject Matter Experts:** Called only when specific expertise is needed

I maintain service dependency maps and escalation matrices to make these decisions quickly under pressure.

Technical Skills & Problem-Solving

Strongest Technical Skills

- **Kubernetes Troubleshooting:** Pod debugging, cluster networking, resource management, operator development
- **Distributed Systems:** Service mesh debugging, inter-service communication, data consistency issues
- **Database Performance:** Query optimization, connection pooling, replication lag investigation
- **Network Diagnostics:** Packet analysis, routing issues, DNS resolution, load balancer configuration
- **Observability:** Building monitoring solutions, log correlation, distributed tracing

Network-Application Correlation Tool

Approach for Building Network Discovery Tool:

1. **Data Collection Strategy:**

```
- Query ServiceNow CMDB for application-to-server mappings
- Collect network topology from network management systems
- Gather application service discovery data from Kubernetes/service mesh
- Correlate using IP addresses, hostnames, and service dependencies
```

2. **Architecture:**
   - **Data Ingestion Layer:** APIs to pull from ServiceNow, network tools, K8s API
   - **Correlation Engine:** Match applications to network elements via IP/hostname mapping
   - **Query Interface:** REST API and web UI for incident responders
   - **Automated Updates:** Scheduled jobs to keep mappings current
3. **Implementation Steps:**
   - Start with pilot application with known network path
   - Build data pipeline to correlate app → server → network element
   - Create visualization showing network path dependencies
   - Add real-time status checking for each network element
   - Build alerting when network elements in app's path fail

Kubernetes Troubleshooting

Kubernetes Issue Triaging

My Systematic Approach:

1. **Application Layer (First 5 minutes):**

```bash
kubectl get pods -n <namespace>
kubectl describe pod <failing-pod>
kubectl logs <pod> --previous  # Check previous container logs
```

2. **Resource Layer:**

```bash
kubectl top nodes  # Check resource utilization
kubectl get events --sort-by=.metadata.creationTimestamp
kubectl describe node <node-name>
```

3. **Network Layer:**

```bash
kubectl get services, ingress
kubectl exec -it <pod> -- netstat -tulpn
# Test connectivity between pods
```

4. **Infrastructure Layer:**
   - Check cloud provider dashboards (AWS/GCP/Azure)
   - Validate DNS resolution within cluster
   - Review service mesh configuration if applicable

Pinpointing Platform Issues

Real Example: Recently diagnosed an issue where pods were getting OOMKilled but resource requests looked appropriate. My approach:

- Analyzed actual memory usage patterns vs requested limits
- Discovered memory leak in application during specific API calls
- Correlated with application logs to find the triggering condition
- Worked with dev team to implement circuit breaker pattern

Connection Methods: I connect to Kubernetes using multiple methods depending on the situation:

- `kubectl` with proper context switching for different clusters
- K9s for interactive cluster exploration during investigations
- Direct API calls when automation is needed
- Web UI dashboards for visual correlation during incidents

Disaster Scenario Response

Complete Data Center Outage

Catastrophic Network Failure - My Thought Process:

1. **Immediate Actions (0-5 minutes):**
   - Verify scope: Is it truly everything or specific services?
   - Check if secondary data center/cloud region is accessible
   - Establish out-of-band communication channels

- Alert disaster recovery teams

2. **Investigation Approach:**
   - **Physical Layer:** Check power, cooling, physical connectivity
   - **Network Layer:** Border routers, core switches, DNS servers
   - **External Dependencies:** ISP connections, BGP routing
   - **Recent Changes:** Any network maintenance, configuration changes
3. **Failover Strategy:**
   - Activate disaster recovery procedures
   - Route traffic to backup data center
   - Communicate expected recovery time to stakeholders
   - Document timeline for post-incident review

Network Troubleshooting Starting Points

When network is suspected:

1. **Connectivity Testing:** Can we reach border routers? Core infrastructure?
2. **DNS Resolution:** Are internal DNS servers responding?
3. **Routing Tables:** Check if routes to external networks exist
4. **Load Balancers:** Are they routing traffic properly?
5. **Service Discovery:** Can services find each other internally?

Database Operations

Cassandra Failover Process

My Approach:

1. **Pre-Failover Assessment:**
   - Check cluster health and node status
   - Verify replication factor and consistency levels
   - Ensure backup data center has recent data
2. **Failover Execution:**

```bash
```

```
# Check cluster status
nodetool status

# If doing datacenter failover:
# Update application connection strings to point to backup DC
# Adjust replication strategy if needed
# Monitor for data consistency issues
```

3. **Post-Failover Validation:**
   - Verify application connectivity
   - Check query performance and latency
   - Monitor for any data inconsistencies
   - Plan for eventual failback procedures

Problem-Solving Philosophy

How I Get Things Done

My Approach to Overcoming Obstacles:

When I encounter blockers like "we can't give you access," I use this escalation approach:

1. **Direct Negotiation:** Explain the business impact and time sensitivity
2. **Alternative Solutions:** "Can you run these commands and share the output?"
3. **Management Escalation:** Involve incident commanders or management when needed
4. **Creative Workarounds:** Find other data sources or monitoring tools
5. **Documentation:** Always document what was blocked for post-incident review

Real Example: During a database performance issue, the DBA team initially refused monitoring access. I:

- Explained the revenue impact of the ongoing outage
- Offered to have a DBA run queries while I guided them remotely
- Escalated to my manager and theirs simultaneously
- Found alternative monitoring through application-level metrics
- Result: Got access within 30 minutes and resolved the issue

Taking Initiative: I don't accept "no" as a final answer during production incidents. I always:

- Provide regular updates on what I've tried and what I'm trying next
- Suggest alternative approaches when blocked
- Escalate appropriately when I need organizational help
- Document blockers for process improvement discussions

Leadership & Coaching

Mentoring Development Teams

I focus on teaching sustainable practices:

- **Observability First:** Help teams instrument their code properly
- **Failure Mode Analysis:** Work through potential failure scenarios
- **Incident Response Training:** Shadow junior developers during incidents
- **Proactive Monitoring:** Build dashboards and alerts before problems occur

Communication During Incidents

- Provide clear, technical updates without overwhelming non-technical stakeholders
- Keep investigation timeline realistic and update it regularly
- Ensure lessons learned sessions happen after every major incident
- Coach team members on effective troubleshooting techniques in real-time

This role requires someone who can be both a technical expert and an organizational leader - driving results through influence, technical expertise, and persistence when facing obstacles.