**SRE Interview Questions and Answers Part 1**

**1. What is the difference between SRE and DevOps?**

**Answer:**

SRE (Site Reliability Engineering) and DevOps are related but not the same.  
DevOps is a set of cultural philosophies and practices that aims to unify software development (Dev) and software operation (Ops). Its goal is to shorten the system development life cycle and provide continuous delivery with high software quality.

SRE, coined by Google, applies software engineering to infrastructure and operations problems. It enforces reliability using measurable Service Level Objectives (SLOs), focuses on toil reduction, automation, and balancing reliability with feature velocity using error budgets.

| **Comparison Area** | **SRE** | **DevOps** |
| --- | --- | --- |
| Origin | Introduced by Google | Cultural movement |
| Focus | Reliability, automation, monitoring | CI/CD, collaboration |
| Core Principles | SLOs, SLIs, Error Budgets | Automation, culture, shared responsibility |
| Tooling | Uses software engineering tools | Emphasis on CI/CD pipelines, version control |

**2. What are SLIs, SLOs, and SLAs? How do they relate?**

**Answer:**

* **SLI (Service Level Indicator):** A quantitative measure of a service's performance. Example: latency, uptime, request success rate.
* **SLO (Service Level Objective):** A target value for an SLI. Example: “99.9% of requests should succeed in a 30-day window.”
* **SLA (Service Level Agreement):** A formal agreement between service providers and customers, including penalties for failing to meet the SLOs.

**Relationship:**  
SLIs → Used to measure service behavior  
SLOs → Targets based on SLIs  
SLAs → Contracts based on SLOs

This hierarchy ensures observability and accountability.

**3. What is an Error Budget and how do you use it?**

**Answer:**

An **Error Budget** is the allowable amount of downtime or failures a system can have without breaching the SLO. It represents the balance between innovation and reliability.

**Example:**  
If your SLO is 99.9% uptime over 30 days:

* Total minutes = 43,200
* Downtime allowed = 0.1% of 43,200 = 43.2 minutes

**Usage:**

* If the error budget is used up → halt feature releases, focus on reliability.
* If not used → can afford to take more risk (release features, experiments).

It enables **data-driven decision making** between ops and dev.

**4. How do you implement monitoring for a distributed system?**

**Answer:**

Monitoring a distributed system involves collecting metrics, logs, and traces across services, components, and hosts. Implementation includes:

1. **Metric Collection**: Use tools like Prometheus, Datadog, or CloudWatch to gather CPU, memory, latency, etc.
2. **Logging**: Centralize logs with ELK (Elasticsearch, Logstash, Kibana), Fluentd, or Loki.
3. **Tracing**: Implement distributed tracing using OpenTelemetry, Jaeger, or Zipkin.
4. **Dashboards**: Visualize key metrics using Grafana or Datadog dashboards.
5. **Alerting**: Set up alerts for SLO breaches, resource exhaustion, or abnormal behavior.

**Best Practices:**

* Define key SLIs (latency, availability)
* Use black-box and white-box monitoring
* Alert on symptoms, not causes

**5. What are some common causes of high latency in web applications and how can you troubleshoot them?**

**Answer:**

**Common Causes:**

* Database slow queries
* Application-level bottlenecks (e.g., loops, locking)
* Network issues (packet loss, congestion)
* Service dependency delays (e.g., upstream APIs)

**Troubleshooting Steps:**

1. **Use APM tools**: Detect slow endpoints/functions.
2. **Check logs**: Look for timeouts or retries.
3. **Query DB slow logs**: Identify problematic SQL.
4. **Network diagnostics**: Use traceroute, ping, mtr.
5. **Load testing**: Use tools like JMeter, Locust to simulate traffic.

**Fixes** might include:

* Caching results (e.g., Redis)
* DB indexing
* Optimizing code
* Load balancing traffic

**6. Describe a runbook. What should be included in a good runbook?**

**Answer:**

A **runbook** is a documented set of procedures for handling known operations or incidents.

**Should Include:**

* Description of the problem
* Preconditions or triggers
* Step-by-step resolution actions
* Validation steps post-fix
* Rollback plan
* Owner/team to escalate to

**Example:**  
Runbook for “Disk Full on Web Server”

* Step 1: SSH into the server
* Step 2: Identify large files (du -sh \*)
* Step 3: Archive or delete logs
* Step 4: Verify disk space

**Benefits**:

* Faster MTTR (Mean Time To Recovery)
* Reduces need for on-call escalation
* Ensures consistency

**7. What is toil in SRE, and how can it be eliminated?**

**Answer:**

**Toil** is repetitive, manual, automatable, and non-value-add work that doesn't require human judgment.

**Examples**:

* Restarting failed services
* Manually rotating logs
* Handling routine tickets

**Elimination Techniques:**

* **Automation**: Write scripts, use Ansible/Terraform
* **Self-healing**: Use Kubernetes probes and auto-restart
* **Better alerting**: Suppress noise with smart alerts
* **Bots**: Use Slack bots for common actions

Goal is to keep toil < 50% of SRE workload.

**8. How would you design a high-availability architecture in a cloud environment?**

**Answer:**

Designing for high availability includes redundancy and fault tolerance at every layer.

**Core Principles:**

* **Multi-Zone**: Use multiple availability zones (AZs)
* **Load Balancers**: Distribute traffic across instances
* **Auto-scaling**: Maintain performance under load
* **Stateless services**: Easier to replicate
* **Managed DBs**: Use HA RDS, CosmosDB, etc.
* **Health checks**: Auto-restart failed nodes

**Example in AWS:**

* ALB → Auto Scaling Group with EC2 across 3 AZs
* Aurora DB with cross-AZ failover
* Route53 for multi-region DNS

Design should be **resilient**, **scalable**, and **self-healing**.

**9. What are the differences between horizontal and vertical scaling? When would you use each?**

**Answer:**

| **Scaling Type** | **Description** | **Use Case** |
| --- | --- | --- |
| **Horizontal** | Add more machines/instances | Cloud-native apps, stateless services |
| **Vertical** | Increase resources (CPU/RAM) on one node | Monolithic apps, legacy databases |

**Horizontal scaling** is preferred in cloud environments because it’s more fault-tolerant and supports auto-scaling.  
**Vertical scaling** is used when application architecture doesn’t support clustering or sharding.

**10. Explain blue-green deployment and its benefits.**

**Answer:**

**Blue-Green Deployment** is a strategy where two environments (Blue = current, Green = new) are used.

**Steps:**

1. Deploy new version to Green.
2. Test internally.
3. Switch traffic from Blue to Green.
4. Rollback by redirecting traffic back to Blue if needed.

**Benefits:**

* Zero downtime
* Easy rollback
* Safe experimentation

Used in CI/CD pipelines for risk mitigation during production releases.

**11. How do you ensure fault tolerance in a microservices architecture?**

**Answer:**

Fault tolerance ensures that the system continues functioning even if components fail.

**Techniques:**

* **Retries with exponential backoff**
* **Circuit Breakers** (e.g., Hystrix, Resilience4j)
* **Bulkheads**: Isolate failures to specific service components
* **Fallback mechanisms**: Return default values or cached data
* **Health checks**: Remove unhealthy instances from service mesh/load balancer

**Best Practices:**

* Monitor each microservice independently
* Use service meshes (like Istio) for traffic management
* Container orchestration (like Kubernetes) for resilience

**12. What is chaos engineering, and why is it important?**

**Answer:**

Chaos engineering is the practice of intentionally injecting faults into a system to test its resilience.

**Tools**: Chaos Monkey, Gremlin, Litmus

**Purpose:**

* Identify weak points
* Validate alerting and monitoring
* Build confidence in production readiness

**Example Scenarios**:

* Terminate instances randomly
* Block network between services
* Increase latency to simulate real-world conditions

**Outcome**: More resilient systems through proactive failure discovery.

**13. How do you handle noisy alerts?**

**Answer:**

Noisy alerts overwhelm engineers and lead to alert fatigue.

**Solutions:**

* Use **deduplication** and **rate-limiting**
* Apply **threshold tuning** on alerts
* Categorize alerts: P1 (urgent), P2 (important), etc.
* Implement **alert silencing** during maintenance
* Use **machine learning-based anomaly detection** for smarter alerting
* Only alert on **symptoms**, not causes

Use postmortems to refine alerting continuously.

**14. How does SRE measure the reliability of a system?**

**Answer:**

Reliability is measured through **SLIs and SLOs**, often based on:

* **Availability (uptime)**: e.g., 99.99%
* **Latency**: e.g., 95% of requests < 200ms
* **Durability**: e.g., no data loss
* **Error rate**: e.g., < 0.1% failed requests

**Metrics Tools**:

* Prometheus
* CloudWatch
* New Relic
* Datadog

**Reliability Score** = How closely your system meets its SLOs.

**15. What is MTTR, MTBF, and MTTD? Why are they important?**

**Answer:**

* **MTTR (Mean Time to Repair)**: Time to fix a failure.
* **MTBF (Mean Time Between Failures)**: Average uptime between failures.
* **MTTD (Mean Time to Detect)**: Time to detect an issue after it occurs.

**Importance**:

* Helps track operational efficiency
* Indicates reliability trends
* Guides incident response planning

Lower MTTR and MTTD → faster recovery. Higher MTBF → fewer outages.

**16. What is the difference between canary and rolling deployment?**

**Answer:**

| **Type** | **Description** | **Benefit** |
| --- | --- | --- |
| **Canary** | Deploy to a small subset of users first | Safer validation, rollback scope |
| **Rolling** | Gradual update across all servers | Less resource-intensive |

**Canary** is better for risk detection.  
**Rolling** is suitable for stable environments with minor changes.

**17. How does Kubernetes help in SRE?**

**Answer:**

Kubernetes simplifies operations and enables reliability through:

* **Auto-scaling**: Horizontal and vertical
* **Self-healing**: Restart failed pods
* **Rolling updates**: Minimal downtime deployments
* **Health checks**: Liveness and readiness probes
* **Namespaces & RBAC**: Better isolation and security

SREs leverage Kubernetes to automate deployments, monitor workloads, and maintain service uptime efficiently.

**18. What is a blameless postmortem, and why is it important?**

**Answer:**

A **blameless postmortem** focuses on learning from incidents without attributing fault to individuals.

**Why it matters:**

* Encourages transparency
* Builds psychological safety
* Uncovers systemic weaknesses

**Postmortem Includes**:

* Summary of the incident
* Timeline
* Root cause analysis
* Action items
* Lessons learned

Promotes a culture of continuous improvement.

**19. How do you handle database scaling in high-traffic systems?**

**Answer:**

**Vertical Scaling**:

* Add more CPU/RAM to DB server (limited scalability)

**Horizontal Scaling**:

* **Sharding**: Split data across nodes
* **Read Replicas**: Distribute read traffic
* **Caching**: Use Redis/Memcached for frequent reads
* **Connection pooling**: Optimize database connections

Use **proxy layers** (like ProxySQL) and managed DB services (like Amazon Aurora) for easier scaling.

**20. What is service mesh, and how does it help in observability?**

**Answer:**

A **service mesh** is an infrastructure layer that manages service-to-service communication.

**Examples**: Istio, Linkerd, Consul

**Features**:

* Traffic management (routing, retries)
* Security (mTLS)
* Observability (metrics, traces)
* Fault injection (chaos testing)

**In Observability**:

* Automatically collects telemetry data (latency, errors)
* Integrates with tracing and logging tools
* Enables fine-grained metrics per service

**21. How do you reduce Mean Time to Recovery (MTTR)?**

**Answer:**

MTTR can be reduced by:

* **Robust monitoring**: Real-time alerts to catch issues quickly
* **Runbooks**: Fast, documented response
* **Auto-remediation**: Scripts to self-heal (e.g., restart services)
* **Incident drills**: Prepare responders via simulations
* **CI/CD rollback**: Enable rapid reversion to last known good state

**Tooling**: PagerDuty, Opsgenie, Lambda/Functions for automation

**22. Describe a typical SRE on-call process.**

**Answer:**

1. **Alerting system** (Prometheus, CloudWatch) detects an issue
2. **Notification** sent to on-call engineer via PagerDuty, Opsgenie
3. Engineer assesses logs, dashboards
4. **Mitigation**: Hotfix, restart, or scale-out
5. **Escalate** if necessary
6. After issue → **Postmortem** created

Goals: fast detection, minimal downtime, documented resolution

**23. How do you handle secret management in production systems?**

**Answer:**

Use tools like:

* **HashiCorp Vault**
* **AWS Secrets Manager**
* **Azure Key Vault**
* **Kubernetes Secrets**

**Practices:**

* Encrypt secrets at rest and in transit
* Rotate secrets regularly
* Use IAM for controlled access
* Never hardcode secrets in code or CI/CD

**24. How do you manage resource limits in Kubernetes?**

**Answer:**

Set **resource requests and limits** in Pod specs:

resources:

requests:

cpu: "250m"

memory: "512Mi"

limits:

cpu: "500m"

memory: "1Gi"

**Benefits**:

* Prevent resource starvation
* Enable efficient scheduling
* Avoid noisy neighbor problems

Monitor using Prometheus and Grafana.

**25. How do you conduct capacity planning?**

**Answer:**

Steps:

1. **Gather metrics**: CPU, memory, disk, traffic trends
2. **Forecast usage**: Based on historical data and business growth
3. **Test scalability**: Stress/load tests
4. **Plan buffer**: Typically 30% headroom
5. **Reassess periodically**

Tools: Grafana, CloudWatch, Datadog

**26. What is observability? How is it different from monitoring?**

**Answer:**

* **Monitoring**: Predefined metrics and alerts
* **Observability**: Ability to understand internal system state from outputs (metrics, logs, traces)

**3 Pillars of Observability**:

1. Metrics (quantitative)
2. Logs (textual)
3. Traces (request flows)

Observability helps debug unknown unknowns. Monitoring detects known issues.

**27. How do you ensure zero-downtime deployments?**

**Answer:**

Techniques:

* **Blue-Green or Canary deployments**
* **Load balancer health checks**
* **Readiness/liveness probes**
* **Preload containers before switching traffic**
* **Use rolling updates with maxSurge and maxUnavailable**

Kubernetes and service meshes help manage zero-downtime rollouts.

**28. What is infrastructure as code (IaC), and how does it benefit SREs?**

**Answer:**

IaC is managing infrastructure using code (declarative/imperative).

**Tools**: Terraform, Pulumi, AWS CDK, Ansible

**Benefits**:

* Version control for infra
* Reproducibility and audit trails
* Easier testing and rollback
* Automation of provisioning

Enables GitOps workflows for environment management.

**29. What is GitOps and how is it applied in SRE workflows?**

**Answer:**

**GitOps** uses Git as a single source of truth for infra and app config.

**Process**:

* Desired state in Git
* Git triggers deployment via agents (e.g., ArgoCD, Flux)
* Reconciliation loops correct drift

**Benefits**:

* Auditable changes
* Easy rollback
* Consistency across environments

**30. How do you test disaster recovery in cloud systems?**

**Answer:**

**Plan**:

* Define RPO (Recovery Point Objective)
* Define RTO (Recovery Time Objective)

**Testing**:

* Simulate region outage
* Practice DB failovers
* Validate backup restores
* Chaos engineering for DR

Use **playbooks** and document DR results.

**31. Explain rate limiting and why it’s important.**

**Answer:**

**Rate limiting** restricts number of requests per client/service to prevent abuse or overload.

**Types**:

* IP-based
* User-based
* Token-bucket algorithms

**Tools**: NGINX, Envoy, API Gateway

Prevents:

* DDoS
* API overuse
* Resource exhaustion

**32. How do you ensure security in CI/CD pipelines?**

**Answer:**

Steps:

* Use static analysis tools (e.g., SonarQube, Checkov)
* Scan for secrets (e.g., truffleHog)
* Sign artifacts (e.g., Cosign)
* Use least privilege for CI/CD tokens
* Enable multi-factor authentication (MFA)

Security should be integrated into each pipeline stage (DevSecOps).

**33. What’s the difference between a symptom-based and cause-based alert?**

**Answer:**

| **Type** | **Description** |
| --- | --- |
| **Symptom-based** | Alerts on impact (e.g., high latency) |
| **Cause-based** | Alerts on root issue (e.g., CPU spike) |

Prefer **symptom-based alerts** because they align with user experience.

**34. How do you handle cascading failures?**

**Answer:**

Cascading failures spread from one service to another.

**Prevention**:

* Implement circuit breakers
* Add timeouts and retries
* Use bulkheads to isolate services
* Monitor dependency health

Service meshes can enforce limits to prevent overload.

**35. How does load shedding work?**

**Answer:**

Load shedding drops low-priority traffic during high load to protect the system.

**Methods**:

* Reject requests with 503
* Queue overflow handling
* Prioritize traffic tiers (e.g., VIP vs free users)

Preserves system availability under stress.

**36. What is distributed tracing and how do you use it?**

**Answer:**

**Distributed tracing** tracks requests across microservices.

**Tooling**: OpenTelemetry, Jaeger, Zipkin

**Use Case**:

* Find latency bottlenecks
* Diagnose service dependencies
* Correlate logs and spans

**Output**: Span trees with timestamps and service names

**37. What is the principle of least privilege in access control?**

**Answer:**

Only give access necessary for a user/app to perform their task.

**Applies to**:

* IAM roles
* Kubernetes RBAC
* API keys and credentials

Improves security and reduces breach impact.

**38. How do you track configuration drift?**

**Answer:**

**Configuration drift** occurs when systems diverge from declared state.

**Solutions**:

* IaC enforcement (Terraform, Ansible)
* GitOps reconciliation loops
* Drift detection tools (e.g., Terraform Plan)

Automated alerts and enforcement reduce risk.

**39. What is alert fatigue and how do you combat it?**

**Answer:**

Alert fatigue happens when engineers receive too many irrelevant alerts.

**Solutions**:

* Triage alerts into severity levels
* Silence non-critical alerts
* Use better thresholds and anomaly detection
* Run weekly alert review meetings

Keep alerts actionable and minimal.

**40. Describe the incident lifecycle.**

**Answer:**

1. **Detection** (monitoring/alerts)
2. **Response** (triage, mitigation)
3. **Resolution** (restore service)
4. **Postmortem** (blameless, documented)
5. **Action Items** (to prevent recurrence)

Tools: PagerDuty, Slack, Statuspage

**41. How do you design a high-availability system?**

**Answer:**

**High availability (HA)** ensures minimal downtime. Key components:

* **Redundancy**: Use multiple instances/load balancers
* **Failover mechanisms**: Auto switch on failure
* **Health checks**: Remove unhealthy nodes
* **Geographic distribution**: Multi-region or zone setup
* **Stateless design**: Easier failover

Use services like AWS ALB, Azure Availability Zones, or Kubernetes with anti-affinity rules.

**42. What’s the difference between a soft and hard outage?**

**Answer:**

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| **Soft** | Degraded performance, partial issue | Slow response, 503 errors |
| **Hard** | Complete service failure | Website down, DB unreachable |

Soft outages can be harder to detect without SLIs/SLOs.

**43. How do you scale systems to handle sudden traffic spikes?**

**Answer:**

**Strategies**:

* Auto-scaling groups (AWS ASG, K8s HPA)
* Queueing mechanisms (SQS, RabbitMQ)
* CDN offloading (Cloudflare, Akamai)
* Caching (Redis, Varnish)
* Pre-warm critical services (Lambda provisioned concurrency)

Plan ahead with **load testing** and scale limits.

**44. What’s the purpose of an error budget, and how do you enforce it?**

**Answer:**

**Error budget** = 100% - SLO target  
(e.g., 99.9% SLO → 0.1% budget for failure)

**Usage**:

* Track system reliability
* Pause deployments if budget exhausted
* Encourage balance between innovation and reliability

Monitored via dashboards; decisions tied to deployment frequency.

**45. How do you handle ephemeral environments for testing?**

**Answer:**

Ephemeral environments are temporary, disposable staging/test setups.

**Tools**:

* Terraform with short TTL
* Preview environments via CI (e.g., GitHub Actions, GitLab)
* Use Kubernetes namespaces per PR

Benefits: consistent testing, cost efficiency, no long-lived test infra

**46. What’s the difference between a playbook and a runbook?**

**Answer:**

| **Type** | **Description** |
| --- | --- |
| **Runbook** | Step-by-step resolution instructions |
| **Playbook** | High-level strategy for incident types |

**Runbook**: "How to restart Redis."  
**Playbook**: "What to do during DB outage."

**47. How do you monitor service dependencies in microservices?**

**Answer:**

**Approach**:

* Distributed tracing (OpenTelemetry, Jaeger)
* Service mesh observability (Istio, Linkerd)
* Dependency graphs (e.g., Dynatrace, New Relic)

Set up **alerts** for downstream failures and latency spikes.

**48. What’s toil in SRE and how do you reduce it?**

**Answer:**

**Toil** = Manual, repetitive, automatable work with no lasting value

**Examples**:

* Manual deployments
* Rebooting services
* Updating dashboards

**Reduction Strategies**:

* Automation (scripts, CI/CD)
* Self-healing systems
* Standardized templates

Google SRE recommends keeping toil under 50% of SRE work.

**49. What is auto-remediation, and give examples.**

**Answer:**

**Auto-remediation** automates incident response without human intervention.

**Examples**:

* Restart a crashed pod via K8s liveness probe
* Auto scale on high CPU usage
* Replace failed EC2 instance using ASG
* Lambda function to restart service on failure alert

Helps reduce MTTR and improve uptime.

**50. How do you prepare for major product launches from an SRE perspective?**

**Answer:**

Steps:

* **Load testing**: Predict traffic behavior
* **Chaos drills**: Validate resilience
* **Capacity planning**: Scale resources ahead
* **Rollback plan**: Tested and ready
* **Observability checks**: Dashboards, alerts verified
* **On-call rotation**: Adjusted for event

**War room** setup often used during live launch.