Case Study

"**QuickBasket**", a grocery delivery startup, currently uses **Traditional Ops**.

Current Metrics

- Average monthly downtime: **6 hours**
- Current uptime: **99.17%**
- Peak order surge: **4× traffic every weekend**
- Checkout API latency:
    - P95 latency: **650 ms**
    - Expected: **< 300 ms**
- Ops team toil: **65%** of total work
- Weekly repetitive tasks:
    - 80 manual pod restarts
    - 40 manual scale-up actions
    - 25 ticket-based approvals
    - 12 manual config updates
- Deployment:
    - 100% manual, done at midnight
    - Failure rate: **18%**

The CTO wants to shift to **SRE model** using **three principles**.

Task

Q1 — Embrace Risk Using SLOs & Error Budgets

Using the data above:

1. Propose **two SLOs** for QuickBasket (Availability + Latency).
2. Calculate the **monthly error budget (in minutes)** for your availability SLO.
3. Explain how the error budget should influence release decisions during weekends (4× traffic surge).
4. Explain how this replaces the "zero downtime" mindset of Traditional Ops.

Q2 — Reduce Toil (Automation Required)

Using the toil numbers given:

1. Identify **three major toil items** from the list.
2. Calculate **total weekly toil actions** (sum them).
3. Recommend which tasks should be automated first and **why**, using numbers.
4. Explain how this reduction in toil will improve:
    - Reliability
    - MTTR
    - Team productivity

Q3 — Engineering-Focused Operations (Blameless, Monitoring, Coding Ops)

Based on the scenario:

1. Describe **how blameless postmortems** would improve reliability compared to current blame-driven ops.
2. Propose at least **3 SLIs (latency, availability, freshness, etc.)** to rebuild monitoring.
3. Explain how adopting **Infrastructure as Code / Monitoring as Code** will fix the current repetitive manual tasks.
4. Explain how Dev + Ops collaboration will reduce the **18% deployment failure rate**.

Model Answer

Q1: Embrace Risk Using SLOs & Error Budgets

1. Proposed SLOs

- **Availability SLO:** 99.5% monthly
- **Checkout API Latency SLO:** P95 < 300 ms

2. Monthly Error Budget

For **99.5% SLO**:

- Month = 30 days = 43,200 mins
- Allowed downtime = 0.5%
- **Error Budget = 216 minutes per month**

3. How Error Budget Influences Weekend Releases

Since traffic spikes **4× every weekend**, releases should follow rules:

- If more than **150 mins** of error budget already consumed → **freeze releases**
- If less than **50 mins** consumed → allow releases with rollback alarms
- If latency SLO is violated (P95 > 300 ms) → reduce release speed and fix hot paths

This transforms decisions into **data-driven governance** rather than gut feeling.

4. Replacing Zero Downtime Mindset

Traditional Ops: "No downtime allowed → delay releases → slow innovation."
SRE: "Downtime is allowed within budget → controlled risk → faster delivery + predictable reliability."

Q2: Reduce Toil

1. Three Major Toil Items

- 80 manual pod restarts
- 40 manual scale-ups
- 25 ticket-based approvals

2. Total Weekly Toil Actions

80 + 40 + 25 + 12 = **157 repetitive actions per week**

3. What to Automate First & Why

- **Pod restarts (80/week)** → Highest volume & easily scriptable
- **Scale-up actions (40/week)** → Causes latency drops during spikes
- **Ticket approvals (25/week)** → Causes deployment delays and long MTTR

Automating these removes **145/157 = 92%** of repetitive actions.

4. Reliability Benefits

- **Lower MTTR:** Auto-healing replaces manual restarts
- **Better Availability:** Auto-scaling handles spikes instantly
- **Improved Productivity:** 65% toil reduced → engineers focus on redesigning slow services
- **Fewer Errors:** No human mis-clicks / wrong config edits

Q3: Engineering-Focused Operations

1. Blameless Postmortems

Current method:

- "Who caused it?" → engineers hide mistakes → repeated issues

SRE method:

- Ask "What failed?" → focus on systemic causes
- Creates shared learning → eliminates recurring incidents
- Allows bold changes → improves culture

2. Proposed SLIs

- **API latency (P95)**
- **Availability (successful requests / total)**
- **Order freshness time** (time from cart → backend processing)
- **Queue lag** for inventory updates

3. Monitoring & Infrastructure as Code

- "Manual config updates (12 per week)" replaced with Git-based IaC
- Dashboard + alert rules become versioned code
- Alerts tied to SLOs → no more noisy thresholds
- Faster rollbacks using Git history

4. Reduce 18% Deployment Failure Rate

- Use CI/CD pipelines with automated tests
- Shift-left reliability checks
- SRE + Dev collaboration to define SLO-based gating
- Gradual rollouts (blue/green, canary)

Outcome: Deployment failure rate drops from **18%** → **< 5%** over time.