CTRL+CHAOS FREE RESOURCE

Technical Interview Readiness Checklist

For Linux, SRE, DevOps, Cloud, and Platform Engineering roles

> **The goal is not to know everything.**
> Strong candidates can explain how systems behave, how they fail, what they would check, and how they would prove what is happening.

Use this before every technical interview

- ☐ Score yourself honestly before you start preparing.
- ☐ Spend most of your prep time on the weak areas that matter for this specific role.
- ☐ Practice answers out loud. Reading an answer is not the same as explaining one.
- ☐ Repeat the final readiness check the night before the interview.

Your starting score

Role fit	Technical depth	Troubleshooting	Communication	Evidence
0 1 2 3 4	0 1 2 3 4	0 1 2 3 4	0 1 2 3 4	0 1 2 3 4

Circle one number in each column. You will score yourself again on the final page.

> **A strong technical interview is a conversation, not a trivia contest.**
> The interviewer is evaluating what you know, how you reason, how you communicate uncertainty, and whether your approach would be safe in a real system.

01 BEFORE YOU STUDY

Build the interview map

Do not start by randomly reviewing commands. First decide what this interview is actually likely to test.

Role map

- ☐ I highlighted the 5-8 skills the job description emphasizes most.
- ☐ For every important skill, I can point to one project, incident, or real example from my experience.
- ☐ I identified any term, platform, or responsibility I cannot explain yet.
- ☐ I know the interview format: live troubleshooting, coding, system design, take-home, panel, or mixed.
- ☐ I know what seniority this role expects and I am preparing at that level.

Resume map

- ☐ I can explain every technical claim on my resume without guessing.
- ☐ For each major project, I can explain the problem, architecture, decisions, tradeoffs, result, and what I would change now.
- ☐ I can explain the scale I actually worked at without exaggerating it.
- ☐ I can separate what I personally owned or changed from what the team owned.
- ☐ I remember the tools and technologies listed on older roles well enough to discuss them honestly.

Company and team map

- ☐ I understand what the company builds and who uses it.
- ☐ I looked for clues about scale, reliability, cloud, Kubernetes, security, or operational responsibilities in the job description.
- ☐ I prepared 3 questions that would help me understand the team, not questions I could answer from the homepage.

Logistics

- ☐ Time zone, date, interview length, and interviewer names are confirmed.
- ☐ Camera, microphone, screen sharing, terminal/IDE, and internet connection are tested.
- ☐ For an onsite interview, transportation and arrival time are under my control.
- ☐ I know whether external documentation, notes, AI tools, or internet access are allowed.

Core readiness matrix

Score each domain from 0-3. Use the role description to decide which rows matter most.

Domain	0 - Not ready	1 - Explain	2 - Troubleshoot	3 - Tradeoffs
Linux	need review	processes, filesystems, permissions, systemd	CPU, memory, disk, services, logs	performance, safety, failure modes
Networking / DNS	need review	TCP/IP, ports, DNS, HTTP/TLS	reachability, name resolution, latency	routing, LB, caching, timeouts
Git	need review	commits, branches, remote refs	push/pull conflicts, history issues	rebase, force-with-lease, protections
Scripting / Automation	need review	read and write basic scripts	debug failed automation	idempotency, error handling, safety
Containers	need review	images, layers, processes, networking	startup, image, resource, connectivity issues	isolation, lifecycle, security
Kubernetes	need review	pods, deployments, services, scheduling	pending/crashloop/ network/node problems	HA, rollout, resources, failure behavior
CI/CD	need review	build/test/package/deploy stages	failed pipeline and deployment investigation	rollback, gates, speed vs safety
Cloud / IaC	need review	core compute/network/storage + IaC concepts	permissions, state, drift, dependency issues	availability, cost, blast radius
System Design	need review	request flow, load balancing, caching, queues, storage	bottlenecks, dependency failure, scaling issues	HA, consistency, capacity, failure domains
Observability	need review	metrics, logs, traces, alerts	choose signals for an incident	SLOs, cardinality, noise, cost
Security	need review	least privilege, IAM, secrets, network controls	access, policy, secret exposure, containment	risk, defense-in-depth, operational tradeoffs
Reliability / Resilience	need review	HA, backups, capacity, failure domains	failover, recovery, durability, dependency failure	RTO/RPO, resilience, cost, complexity

Prioritize by role, not by ego.

If Kubernetes is central to the role and you score yourself 1, that matters more than scoring 3 in a technology the team barely uses.

03 HOW TO ANSWER

Use the TRACE framework

For systems, architecture, and troubleshooting questions, strong answers usually explain a journey. TRACE gives you a structure without making your answer sound memorized.

Step	What it means	What to say / do
T - Target	Clarify the goal, symptom, and scope.	"When you say the site is down, is it all users or one region?"
R - Route	Walk through the system end to end.	Client -> DNS -> load balancer -> service -> dependency -> response.
A - Assumptions	State what you are assuming and what could differ.	"I am assuming traffic reaches the load balancer. If not, I would branch earlier."
C - Checks	Name the signals, failure modes, and tradeoffs you would inspect.	Metrics, logs, traces, events, recent changes, config, dependencies.
E - Evidence	Explain how you would verify the answer or the fix.	Re-test the user path and confirm the relevant signals return to normal.

If you get stuck

- ☐ Say what you know: "Let me start with the part I am confident about."
- ☐ Make an explicit assumption: "Can I assume DNS is resolving correctly?"
- ☐ Separate concept from syntax: "I do not remember the exact flag, but I would inspect the process environment and verify..."
- ☐ Ask for one useful clue instead of guessing randomly.
- ☐ If you realize an earlier assumption was wrong, correct it and continue. Do not defend a bad path.

Example: "The website is down. What do you do?"

- Target: clarify user impact, scope, time of onset, and whether anything changed.
- Route: walk the request path from DNS and network entry points through the application and dependencies.
- Assumptions: state which layers appear healthy and where you are narrowing the problem.
- Checks: correlate metrics, logs, traces, deployment/config events, and dependency health.
- Evidence: verify the customer path after mitigation and watch the signals long enough to know it is stable.

04 TROUBLESHOOTING

Show a method, not a command dump

Interviewers learn more from the order of your investigation than from how many commands you can name.

Before touching anything

- ☐ What changed recently? Deploy, config, certificate, dependency, traffic, capacity, infrastructure?
- ☐ What is the blast radius? One user, one host, one AZ/region, one service, or everything?
- ☐ What is the customer impact and urgency?
- ☐ What does a known-good baseline look like?
- ☐ Is there a safe mitigation or rollback while investigation continues?

Senior SRE signal: If the service has defined SLOs, identify which SLI is degrading, whether error-budget burn rate changes the urgency, and whether mitigation or rollback should happen before deeper diagnosis.

Build evidence

- ☐ Metrics: saturation, errors, latency, throughput, resource pressure.
- ☐ Logs: errors and state changes around the same time window.
- ☐ Traces: where time or failures appear in the request path.
- ☐ Events and changes: deployments, config, autoscaling, certificates, DNS, dependencies.
- ☐ Host/container signals: process state, disk, memory, CPU, network, filesystem, kernel/system logs.

Run the hypothesis loop

- ☐ Form one hypothesis that explains the evidence.
- ☐ Choose the safest test that can confirm or weaken it.
- ☐ Observe the result before changing something else.
- ☐ Update the hypothesis and continue narrowing the scope.

Red flags interviewers notice

Weak signal	Better behavior
"I would restart it."	First explain what you would learn before destroying evidence.
Listing 15 commands immediately	Choose commands only after you know what question you are answering.
Locking onto one cause	Keep alternative hypotheses until evidence eliminates them.
Ignoring user impact	Separate mitigation from root-cause investigation.
Claiming the root cause early	Say what evidence would be required to prove it.

10 journey questions worth practicing

For each question, practice a 2-minute answer, a 5-minute answer, and at least two follow-up questions.

#	Question	2 min	5 min	Follow-ups
1	A user types a URL into a browser and presses Enter. What happens until the page loads?	☐	☐	☐
2	What really happens after you run git push?	☐	☐	☐
3	What happens after kubectl apply -f deployment.yaml?	☐	☐	☐
4	How does a Kubernetes Service route traffic to a Pod?	☐	☐	☐
5	Walk me through how a Pod gets scheduled and starts running on a node.	☐	☐	☐
6	What happens if the node hosting your Pod suddenly dies?	☐	☐	☐
7	I give you SSH access to a Linux server you have never seen. How do you figure out what it does?	☐	☐	☐
8	A Linux server is out of disk space. Walk me through your investigation and recovery.	☐	☐	☐
9	A file was deleted, but disk space was not freed. What is happening and what would you do?	☐	☐	☐
10	Walk me through everything from git push to a production deployment. Where does Git end and CI/CD begin?	☐	☐	☐

Senior / SRE bonus

Design a service with a 99.9% availability target. What would you measure, where can it fail, and how would SLOs and error-budget burn influence your decisions during an incident?

> **Practice complete journeys.**
> A stronger answer is not simply an answer with more technical words. It explains the system clearly, at the right depth, and shows where failures and verification fit.

Sound experienced without sounding rehearsed

Level	Typical answer	Signal to interviewer
Junior	Explains core components, the normal flow, and obvious failure points.	Shows solid fundamentals and asks useful clarifying questions.
Mid-level	Explains the path, common failures, evidence to inspect, and a safe troubleshooting approach.	Can operate independently in familiar systems and verify results.
Senior	Explains the path, tradeoffs, failure modes, evidence, operational safety, and what changes with context.	Can reason through ambiguity, protect production, and connect decisions to reliability goals when relevant.

Communication checklist

- ☐ I clarify ambiguous questions before solving the wrong problem.
- ☐ I explain my reasoning, not only my conclusion.
- ☐ I separate facts, assumptions, and guesses.
- ☐ I can say "I do not know" without stopping the conversation.
- ☐ I explain why I chose an approach and what I traded away.
- ☐ I avoid filling silence with jargon or unrelated details.
- ☐ I scale the depth of my answer to the question and interviewer cues.
- ☐ For production scenarios, I consider customer impact, blast radius, reversibility, and reliability objectives when relevant.

Evidence bank

Have these examples ready because technical interviews often move from theory into your real experience.

- ☐ 3 production incidents I can explain clearly.
- ☐ 2 projects where I made meaningful technical decisions.
- ☐ 1 failure or wrong decision and what changed afterward.
- ☐ 1 performance/reliability improvement with measurable impact.
- ☐ 1 example where I disagreed with an approach and used evidence to resolve it.
- ☐ 1 system I can draw from memory and discuss at multiple levels of detail.

07 FINAL CHECK

Interview-day readiness

Before the call / onsite

☐ I know the role, interview format, and what this round is likely to test.
☐ I have water, notes, working audio/video, and a quiet environment.
☐ I can access any allowed terminal, IDE, whiteboard, or diagramming tool without setup delays.
☐ I am not depending on last-minute transportation or another person to arrive on time.

During the interview

☐ Listen to the whole question before answering.
☐ Clarify scope and assumptions when needed.
☐ Structure the answer before diving into details.
☐ Think out loud enough that the interviewer can follow the reasoning.
☐ If stuck, recover methodically instead of guessing faster.
☐ End with how I would verify the result.

Final readiness score

Area	Score 0-4	Question
Role understanding		Can I predict what this role is likely to test?
Technical depth		Can I explain the important systems beyond definitions?
Troubleshooting		Can I investigate methodically without random actions?
Communication		Can I make my reasoning easy to follow?
Evidence / stories		Can I connect answers to things I have actually done?

17-20: Ready. 13-16: Close - target the weakest area. 0-12: Stop broad review and do focused practice.

If you only have 60 minutes

Time	Focus
10 min	Map the job description to the 5 most important technical areas.
20 min	Practice 3 journey questions out loud using TRACE.
15 min	Review the projects/incidents most likely to come from your resume.
10 min	Practice one troubleshooting scenario and one system-design explanation.
5 min	Test logistics, close distractions, and stop cramming.

> **You do not need the secret answer in the interviewer's head.**
> You need a clear, evidence-based way to reason through the problem in front of you.