

Prod is down.

The first 15 minutes.



RULE 0

Stop the bleeding first. Find the root cause after.

00 → 02 MIN

Own it

Acknowledge the page so it stops escalating. Open one incident channel. Name an incident commander.

02 → 05 MIN

Size it

Who is affected, which feature, how many users? Set a severity. Say it out loud in the channel.

05 → 10 MIN

What changed?

Most outages follow a change. Check the last 60 minutes: deploys, config, feature flags, infra, dependencies.

10 → 15 MIN

Mitigate

Roll back, fail over, turn the flag off, scale out, shed load. Pick the fastest safe option.

WHO DOES WHAT

Commander	Runs the incident, makes the calls. Does not debug.
Ops lead	Hands on keyboard. Investigates and mitigates.
Comms	Updates stakeholders and the status page.
Scribe	Writes the timeline with timestamps as it happens.

SEVERITY AT A GLANCE

SEV1	Core service down or data at risk. All hands.	every 15m
SEV2	Major feature degraded for many users.	every 30m
SEV3	Minor or partial impact. Workaround exists.	daily

Update cadence is a common default. Use your team's own policy if it has one.

STATUS UPDATE TEMPLATE • COPY, FILL, POST

[SEV2] [INVESTIGATING] Checkout errors at ~12% since 21:04 IST.
Impact: some users can't pay. Browsing works.
Doing: rolled back v2.3.1, watching error rate.
Next update: 21:45 IST or sooner.

Don't

Debug in five private DMs. Ship a hotfix nobody reviewed. Go silent for 40 minutes. Ask “who broke it?”

You're done when

Metrics are back to normal for 15+ minutes, users confirm, and a postmortem owner is named.



The box is slow.

Linux triage in 60 seconds.



SSH in and run these ten, in order. Together they show whether you have a CPU, memory, disk or network problem before you open a single dashboard. Based on Brendan Gregg's well-known 60-second checklist.

01	uptime	Load averages for 1, 5, 15 min. Rising or falling?
02	dmesg -T tail	Kernel errors: OOM killer, disk I/O errors, dropped packets.
03	vmstat 1	r above CPU count = saturation. si/so above 0 = swapping.
04	mpstat -P ALL 1	One core pinned at 100% = a single-threaded hot spot.
05	pidstat 1	Which process is eating CPU, second by second.
06	iostat -xz 1	High %util or await = the disk is the bottleneck.
07	free -m	Look at available , not free. Low = memory pressure.
08	sar -n DEV 1	Network throughput per interface. Near the link limit?
09	sar -n TCP,ETCP 1	Retransmits (retrans/s) point to network trouble.
10	top	The final sanity check. Anything you missed?

WHEN IT'S A SPECIFIC SYMPTOM

Disk full

```
df -h
du -xh / 2>/dev/null | sort -h | tail
lsof +L1 # deleted but still open
```

Service won't start

```
systemctl status app
journalctl -u app --since "15 min ago"
ss -tulpn # port already taken?
```

Slow HTTP calls

```
curl -o /dev/null -s -w \
"dns %{time_namelookup} \
ttfb %{time_starttransfer}\n" URL
```

It's probably DNS

```
dig +short api.example.com
cat /etc/resolv.conf
getent hosts api.example.com
```



The pod is broken.

Read the status first.



STATUS	USUAL CAUSE	FIRST COMMAND
Pending	No node fits: CPU/memory requests, taints, node selector, unbound PVC.	<code>kubectl describe pod P</code>
ImagePullBackOff	Wrong image name or tag, private registry without imagePullSecrets.	<code>kubectl describe pod P</code>
CrashLoopBackOff	The app starts, then exits. Bad config, missing env var, failing dependency.	<code>kubectl logs P --previous</code>
OOMKilled	Memory limit too low, or a leak. Exit code 137.	<code>kubectl top pod P</code>
CreateContainer-ConfigError	A ConfigMap or Secret the pod references doesn't exist.	<code>kubectl get cm,secret</code>
Running, 0/1 Ready	Readiness probe failing. Wrong path, port, or the app is still warming up.	<code>kubectl describe pod P</code>
Service, no traffic	Service selector doesn't match the pod labels, so it has no endpoints.	<code>kubectl get endpoints S</code>
Node NotReady	Kubelet down, disk or memory pressure, network partition.	<code>kubectl describe node N</code>

THE LOOP YOU'LL RUN ALL NIGHT

```
kubectl get pods -o wide
kubectl get events --sort-by=.lastTimestamp
kubectl describe pod P
kubectl logs P -c C --previous
kubectl exec -it P -- sh
```

BAD DEPLOY? UNDO IT

```
kubectl rollout history deploy/D
kubectl rollout undo deploy/D
kubectl rollout status deploy/D
```

Roll back first. Read the diff once traffic is healthy again.

KEY P pod C container D deployment S service N node · add -n NAMESPACE to every command



You can't fix what you can't see.



THREE LENSES • PICK THE ONE THAT FITS WHAT YOU'RE LOOKING AT

Golden signals

Any user-facing system

Latency how slow

Traffic how much

Errors how broken

Saturation how full

RED

Request-driven services

Rate requests per second

Errors failed requests

Duration latency distribution

USE

Resources: CPU, disk, network

Utilization % busy

Saturation queued work

Errors error counts

METRICS

Tell you **that** something is wrong.
Cheap, aggregated, alertable.

TRACES

Tell you **where**. Follow one request
across every service.

LOGS

Tell you **why**. The detail behind
one event or error.

PROMQL STARTER PACK • RENAME METRICS TO MATCH YOUR SETUP

Request rate	sum by (service) (rate(http_requests_total[5m]))
Error ratio	sum(rate(http_requests_total{status=~"5.."}[5m])) / sum(rate(http_requests_total[5m]))
p95 latency	histogram_quantile(0.95, sum by (le) (rate(http_request_duration_seconds_bucket[5m])))
CPU busy %	1 - avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m]))
Memory left	node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes
Disk full in 4h	predict_linear(node_filesystem_avail_bytes[1h], 4 * 3600) < 0
Restarting pods	increase(kube_pod_container_status_restarts_total[15m]) > 3

Log like you'll read it at 3 AM Structured JSON. Always include timestamp, level, service, trace_id. Never log secrets or full card numbers.



SLOs, not vibes. Alerts, not noise.



SLI

What you measure. % of requests under 300 ms.

SLO

Your internal target. 99.9% over 30 days.

SLA

The promise in a contract, with penalties. Looser than the SLO.

ERROR BUDGET • HOW MUCH DOWNTIME EACH TARGET ALLOWS IN 30 DAYS

99%		7h 12m
99.5%		3h 36m
99.9%		43m 12s
99.95%		21m 36s
99.99%		4m 19s

Each extra nine costs roughly 10× more effort. Pick the target users actually notice.

BURN-RATE ALERTS • FROM THE GOOGLE SRE WORKBOOK

PAGE NOW

14.4× burn

2% of the monthly budget gone in 1 hour.

PAGE

6× burn

5% of the budget gone in 6 hours.

TICKET

1× burn

10% of the budget gone in 3 days.
Fix it this week.

THE 5-POINT ALERT TEST • DELETE ANY ALERT THAT FAILS

- ☐ **Actionable.** A human must do something right now.
- ☐ **User-facing.** Alert on symptoms (errors, latency), not causes (CPU 80%).
- ☐ **Has a runbook.** The alert links to what to check first.
- ☐ **Grouped.** One incident, one page. Not forty.
- ☐ **Reviewed.** Noisy alerts get fixed or deleted every week.



After the fire.

The blameless postmortem.



Write it within 48 hours, while memory is fresh. Ask what made the mistake easy to make, never who made it.

1 · SUMMARY

Two sentences a VP can read. What broke, for how long, now fixed.

2 · IMPACT

Users affected, duration, failed requests, money, SLO budget spent.

3 · TIMELINE

Detected, acknowledged, mitigated, resolved. Timestamps with time zone.

4 · ROOT CAUSE

Ask “why” until you hit a process or system gap, not a person.

5 · WHAT WENT WELL / BADLY / LUCKY

“Lucky” is where the next outage is hiding.

6 · ACTION ITEMS

Each with an owner, a due date and a ticket. No owner, no action.

TIMELINE FORMAT

21:04 IST Error rate alert fires (5xx at 12%)
21:06 IST On-call acknowledges, opens #inc-checkout
21:11 IST Deploy v2.3.1 identified as the last change
21:14 IST Rollback started → 21:19 errors back to 0.2%

01

02

03

04

05

06

THIS WAS 6 PAGES. THERE'S A WHOLE LIBRARY.

Every cheat sheet, in one place: DevOps Park.

The complete cheat sheet library lives on DevOps Park, InfraThrone's learning platform. Bookmark it for your next incident.

Open the full library →

infrathrone.xyz/dashboard/assets/cheatsheets

