SRE Responsibilities: availability, latency, perf., effectiveness, change mgmt., monitoring, emer. response and cap. planning of services.

SRE PRACTITIONER

08/13:

SLO → user exp (data)

Tool Redⁿ → Lean

TOC → Theory of constraints (out everything - can't be automated)

→ create think-tank time

SLI → parameters to measure

measurement → 360 degree view (monitoring observability)

* "slow is the new fast"

Anti-Patterns → if we do, it will backfire

1. ReBranding Ops:

* SRE → engineering work requires

* build sys. to ∧ less human intervention, to fail less often and modify existing sys. to remove emergent failure modes.

* don't bring ops closer to machines → sys. should take care of it.

(Build right alerts) modify existing sys. to remove emergent failure modes.

Autom of alerts (False positives and → (a person has spe. disease or condⁿ when the person actually doesn't have)

False negatives → Blind spot / Trap → sys. thinking knowledge)

(a person does not have a spe. disease or condⁿ when the person actually has)

we think everything is ok. but it is not discovered.

Spend more than half of their time on building better sys. → cultural shift ownership reduce toil. dev focus on user exp.

↳ ExLA (experience level agreement)

Recall: 100% if every sig. event results in an alert.
Detn Time: How long it takes to send notn in various conds.
Reset Time: How long alerts fire after an issue is resolved.
↳ (Turn your SLOs into actionable alerts that are grounded on the customer's path)

2. Users Notice an issue before you do

Precision → quality of being exact and refers to how close two or more measurements are to each other
Recall → Key attributes
Detn Time
Reset Time

when it happened fast time

Precn is 100% if every alert corresponds to a significant event.

Build solid fail-over, fault tolerance to resolve within SLO.

↳ sentiment analysis
NLP algo.

SREs → direct contributor to getting the cust. what they want.

3. measuring until my Edge
⇓
Customer's percept^n → reality

SLO → user exp. and not what your SLOs read

being close to biz.
and understand their pain

SLOs → keep on changing.

* Shared instn's, shared integn and shared commn → Key to success

* Meeting SLOs — useless if the cust. doesn't have the exp.

4. False positives are worse than no alerts.
↳ monitoring is abt. the steady flow of traffic, not a steady flow of alerts.

5. config. mgmt trap → SREs + more on homogenenizing ecosystem
Snowflake - ineffecient
design for immutability infra.

change → about
replacing
never updating or patching

* use confign mgmt. to consolidate and integrate to immutable infra.

6. DogPile: Mob Incident response → procedural framework is a mandate. good

Incident Commander: → just calm down
Incident command framework
likelihood ↑ - disruptive to engg. work

* minimize damage, make outage as short as possible

7. Point Fixing: → minimize outage with automated alerts and solid paging mech. and quick workarounds, fast rollback, fail over and fix forward

↓

Reactive ..... Proactive .... Predictive .... auto remediation

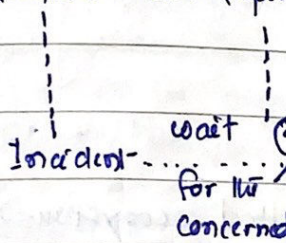
→ short term fix
→ permanent fix

class of elimenate the designer errors → leading to predective methods

i) understanding ii) goal iii) historically what we spent our time.

* Auto remediation closed loop remediations | without human intervn → goal of SRE

8. Prodn Readiness Gatekeeper:



→ increase the length of the time between the creation of a change and its prodn release (gatekeeper)/chokepoint/speed bump.

* Shift left to build in resilience by design in the devp. lifecycle

Systems → 1st line of defence

Security should be in-built.

* SREs provide dev teams with devp. frameworks and templatized configs to speed up reviews.

9. Human Error → it's a sys. error rather than a human error.

→ Prevent the occurence of the event. ⇒ making the sys. more robust.

Root cause is just the place where we decide we know enough to stop analyzing and trying to learn more.

* Q's on case story:

Monzo Bank: defense in depth works for reliability not just for security.

* Chaos Engg: helps in dist. sys. in order to build confidence in the system's capability to withstand turbulence in prod.

- Major Incident Process:

Precn / Detn (MTTD) → Incident commander → war-room

Observability

Chaos Engg.

Blameless Post-mortem

- Good post mortem
  * Blameless
  * answers to 5 whys
  * details on discovery of the inc.
  * timelines (chronology)
  * ways to prevent incident
  * log of related incidents
  * lessons learned / clear action items with timelines

Bad post mortem
  * finger pointing
  * animated language
  * missing ownership
  * missing context
  * key details omitted
  * limited audience

Module 2

SLO → proxy for cust. exp.

Establish SLOs in distributed ecosystems?

↳ varies according to persona customer journey map

Goals

SLI → key measurement of the availability of a sys.

← SLO → goals we set for how much availability we expect out of a sys.

* Setting SLIs and SLOs on system boundaries.

between components in an archi., it can mean producing component and consuming comp.

complex system: understanding - limited

current state → enhancing → enhanced state

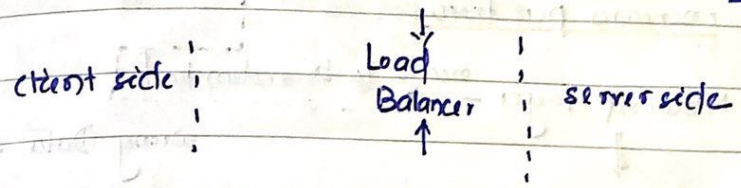
- Complex Sys. Traits: i) cascading failures ii) diff. to determine boundaries
iii) diff. to determine to model behavior
iv) dynamic network of multiplicity
v) emergent phenomena
vi) rel^{n}ship — nonlinear vii) rel^{n}ship contain feedback loops.

- Relationships: non-linear

- QOS
  - Quality of service → Synonymous → SLO
  - SLO → Targets for SLIs.
    - e.g. avg. resp. time, resp. time percentile, throughput, MTTI

- Kudo's SLO journey:

where the SLO needs to be measured → imp.



SLI → identify the options where SLO carries ~~outage~~ weightage on the overall system

* ElasticSearch HeartBeat → to monitor our external only APIs and services

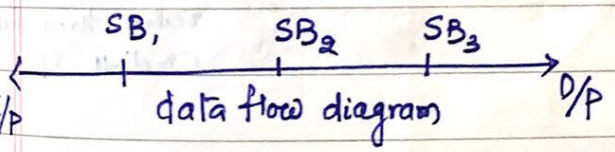
System Boundaries: → Identify the sys. boundaries within your platform.
→ identify the customer-facing capabilities that exist at each sys. boundary
‖
(UI)

- articulate a defn of what it means for each capability to be available.
- Define SLI for that defn

define a SLO for each capability and start measuring to get a baseline.

→ iterate and refine your system and fine tune the SLOs over time.

understanding arch. is a must. / system boundaries as a service owner.



SLO ↓ (manageable) → reduced but not 0%

- Horizontally scaled → Load Balancer
  Hard Sharded → 1-1 mapping ↓
  SLO = 0% for a set of users

- MTTR → mean time to repair
  ⇓
  increases per fixes

  wrong UI → isolated
  wrong service → UI will be impacted but UI caching will reduce impact.
  wrong Data → service + UI fixes will go down.
  wrong Network → everything goes down

  Network Layer → needs to be redundant!

- I. System Boundaries (data domain driven app.)
  D. Cap for sys. boundaries (address cross functn issues in a dis. sys. to keep single service resp.)
  define SLI for each capability
  define SLOs targets
  measure baseline → revise & repeat
  data routing SLI
  one SLO per capability goal

* MTTR increases with count of impacted service tiers.

* Graceful degradation ↓
  ⇓
  significantly decrease the amt. of work or time needed by decreasing the quality of response.
  diff. to implement for network outage.

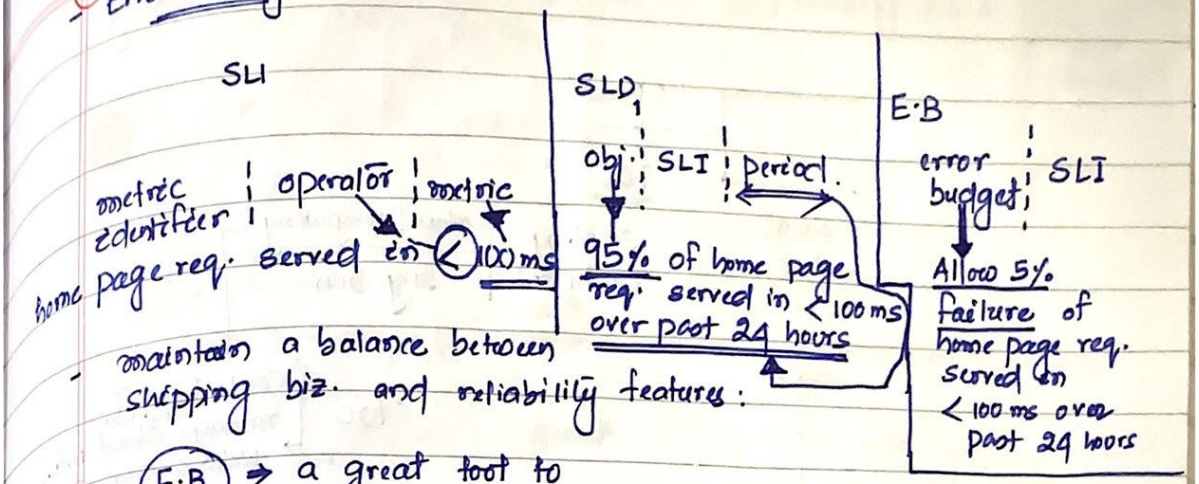
e.g: A search appln might only search a subset of data stored in an in-memory cache rather than full On disk DB.

- Large Scale Cultural Shift:

  SLO Buy-in
  Become a SLO Expert
  create standard artifacts for use
  SLO pilot and rollout
  communicate (iterate)

(37) cascading → Error budget

(38) Error Budget:



- maintain a balance between shipping biz. and reliability features:

(E.B) ⇒ a great tool to drive data-driven discussn.

- Burn of E.B ⇒ determine the weakest link in your dist. ecosystem

* use E.B to improve resiliency
* expt' confidence

E.B → error-budget → great tool to drive data driven discn

Prometheus monitoring tool → Grafana dashboard.

* use Error Budget: turn off heritage components to see impact.

- SLO Inflectn Point:

* E.B burndown charts are i/ps for decn making on disaster recovery / data center drills.

- Home Depot's SLO Journey:

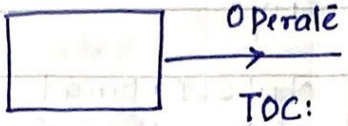
good strategy
executive buy-in
easy adoptn patterns
and patience

VALET
volume
availability
latency
error
tickets

* SLO culture: bring prod. dev. & ops close together

Module-III (Non-Abstract Large System Design)

Non-Abstract Large Scale Design – capacity planning



TOC:

Small biz.

Scaling cashflow | cloud computing sig. ben

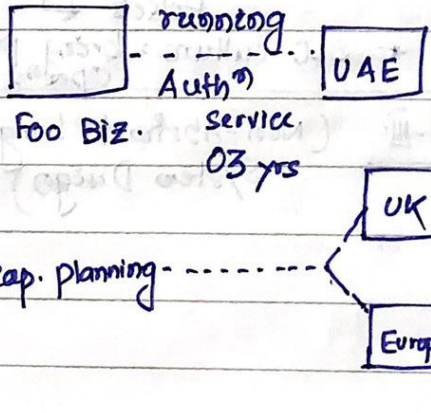
TOC → Control complex manage Cloud infra. with a small IT Team

may not accomodate reactive approach to infra. provisioning and test new config on live env.

→ costly
time-consuming
inactive operational risk

TOC

Simulation → eff. and efficient evaluation and experiment of cloud infra. config and provisioning/de-provisioning of capacity by examining how infra. reacts to demand in the short/med. term.



SLI

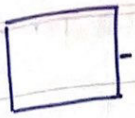
2. user routed to MFA page in < 100 ms

3. ~~user login~~ 95th percentile of login latency over Rain 3 mins < 200 ms

SLO

2. 95% of routing req served in < 100 ms ~~for last~~ over past 24 hrs.

3. 99% of 95th percentile of login latency over 3 mins < 200 ms for the past month



up & running ⇒ SLA

Foo Biz. Authn Service at 5 nines of reliability

Current Fact

concurrent auth req/sec - 10K.

Cluster - 20 machines in pub. cloud.

Future Fact

* diff. SLOs and SLAs per region
* Conc. auth req/sec - 20K.
* Cluster should grow automatically based on load
* traffic flow monitoring should be enabled to enfor future predictability

SLI	SLO
initial	
① user login req served < 100 ms	95% of user init log req served in < 100 ms over last 24 hrs.

Sample set of adv. optimiz conclus

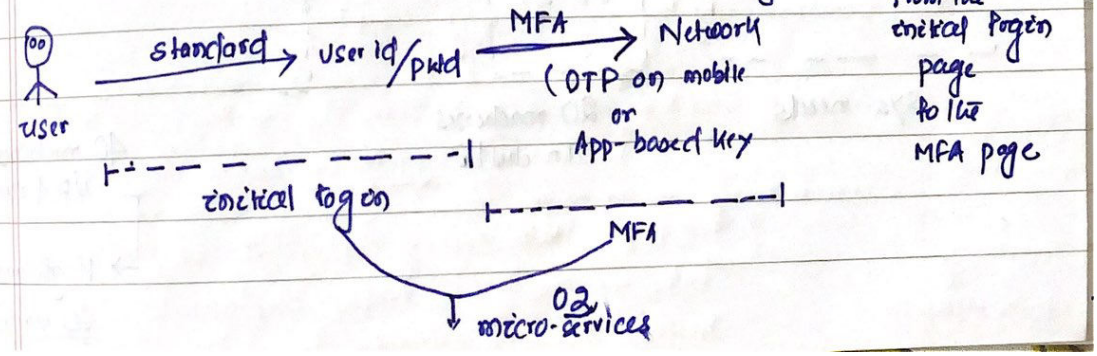
→ SLOs: obj | SLI | period

→ reliable network

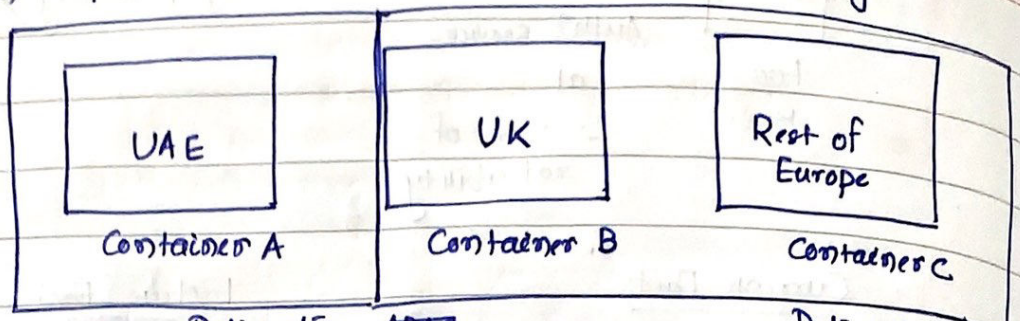
→ globally dis. user base

→ availability regions - UK, Rest of Europe

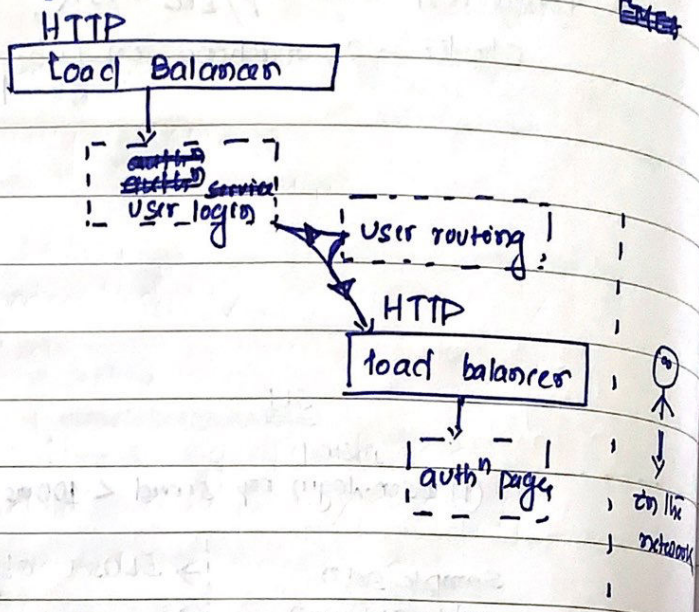
→ H/w: HDD mach. SSD mach.



Scaling: 2) Replication and sharded microservices per region



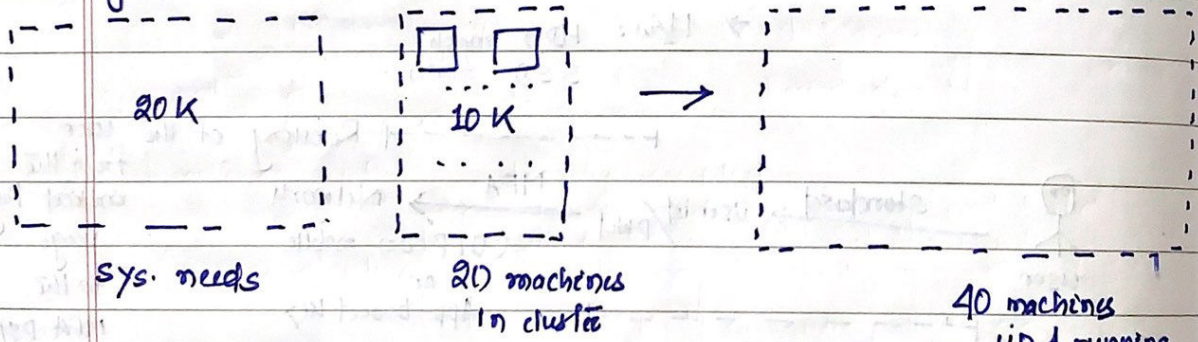
3)



4) How many machines do we need to allocate to each micro-service?

Can we meet our SLO latency req.?

Provisioning:



Conc. users.

UAE: 10K : 20 machines ⇒ 30 machines → availability zone

UK/Europe: 10K : 20 machines. ⇒ 30 machines → availability zone

15 mach. | 15 machines.

Based on data,

	bandwidth	storage	machines reqd.
users in each zone (UAE)	?	?	

UK

Europe

* Machines per DC → No. of DCs × Infra. Tax + more load spikes.

* Advanced optimizn's:

capacity growth

privacy reg. (GDPR) → General Data Regulatn Protn

↑ peak users

* design for changing landscape: containers, service mesh, API gateway, orchestration improves fault tolerance.

* using micro-services, building layers of security control and managing cross-service comm's with a service mesh improves resilience.

08/14: Module-III

Intentional Arch: version should be known.
Know the des. principles.

↓ multi-variance tested

↓ Perf. engineering

* des. for changing landscape:
  ⇓
  CICD
  ↓
  more freq. releases; less changes/release

SonarQube - testing must
Containers & micro-services

* des. for changing security: IAM, SaaS adoption
  ↓
  Zero trust networking
  ↓
  requires a modern identity and access security parameter

* multi-grained service arch: microservices ⇒ more complex arch.
  ⇓
  expose APIs using open standards

* Containers: management
  ↳ Container orchestn: core capability of dist. arch.

Kubernetes: Extending Kub. API, 3rd party configs
Istio (service mesh) — Data Plane
Prometheus (monitoring)
Control Plane
↳ Pilot- (config)
Mixer (Policy, telemetry)
Citadel (encrypt, authn)

* Pod: represents a unit of deployment which may consist of either a single Container or small no. of Containers that are tightly coupled.

KEY: automated service delivery, appln monitoring, distributed tracing

D3 Load Balancing: Round robin

Reactive System: → loosely coupled.

← Canary deployment — releasing to small set of users (5%)
↳ consists of gradually shifting prod. traffic from version A to version B and releasing to a subset of users
↳ feature flagging or routing logic can be used.
mostly useful for error rate and perf. monitoring with faster rollback

Service mesh - manages comm's between microservices.
↳ helps with canary deployment for traffic splitting.
↳ helps → C.D for traffic deployments
higher level of security for each service inside a cluster.

Reactive Systems: Manifesto

Responsive → cornerstone of usability.
Resilient → resilience achieved by replicn, containment, isolation and delegn
← Elastic
message driven → asynchronous message passing to establish a boundary between components that ensures loose coupling, isoln and locn transparency.

Sys. stays responsive under a varying workload.

- Rolling upgrade:

old new versions of the code at the same time. This requires your appln handle N-1 compatibility.

recommended

| max. surge | max. unavailable |

- Blue-Green: → expensive

→ The new version of an appln is deployed in an identical env.

Tech. to understand

* instant rollout/rollback
* avoids versioning issue
* stateful apps → result → few glitches

⊛ Router used → select the appln version which will be exposed to the users.

- Threat:

Design for Security: MITRE ATT&CK → framework of adversary tactics and tech. based on real-world obsns.

STRIVE
↳ Spoofing
intrusion
tampering

Cyber-security

idenn & analytics
threat intelligence
adversary emulation & red teaming

⊛ Design for Resiliency: → ability of a sys. to gracefully handle and recover from failures.

⊛ Circuit-Breaker: → to protect from cascading failures and to block traffic to downstream components

caching

⊛ Supervisor agent - service that continuously monitors the appln/daemons/services and restarts them if they fail

any failure during execution, the ability to gracefully rollback a trans to its original state

← compensating transn: graceful roll-back

master selection: Kubernetes →

⊛ desn for scalability: → horizontal scaling
asynchronous design

ability of a sys. to handle increase in load without impact on perf.

statelessness → asynchronous design

* Design for performance: → (measured using the responsiveness of the system on any given period of time, especially during peak loads.)
↓
use serverless arch. (event driven and cost eff. impln)
experiment more often → Observability
go global and be elastic

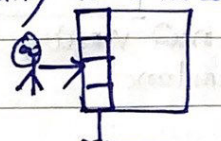
mechanical empathy: when you use the tool of an understanding how it operates best

* design for availability: (MTTR / MTBF)
↓ HA
service level / hypervisor level / load balancer level / network level / zone level / region level

* design for reliability:
↳ test recovery procedures
↳ automatically recover from failure:
scale horizontally to increase sys. availability

deployment of service such as kubernetes pods

Castle and Moat model: →



Big door

stop guessing capacity
manage change using automn
→ Zero Trust.

ID based model

Full-stack Observability

Observability: → granular insight into the implicit failure modes of the system.
↳ uncover deeper, systemic issues.

high-level overviews of sys. health

* An observable sys. furnishes ample context about its inner workings, unlocking the ability to uncover deeper, systemic issues.

Perf. in complex arch:

practitioner's view on observability:
↓
aggn is the biggest enemy. ∧ kill variety that will making the infra totally useless!
↓
Complexity, elasticity and transience are the key considerations.

→ good observability makes detecting, investigating diagnosing and responding to problems easier.

Observability: → granular visibility into the unexpected behavior of complex sys. along with rich context

Key elements: EMTC

- events (appln/sec. logs)
- metrics (number describing a process or activity measured overtime)
- tracing (data showing which lines of code are failing from APMs & SFX)
- context

MELT:
- metrics
- events
- logs
- tracing

04

* Google's Golden signals → traffic, errors, latency, saturation

Observability is about building better insights to your system.

TELS

03 Pillars of Observability (DTR)

- detect → metrics (Do I have a problem)
- troubleshoot → traces (where is the problem)
- root cause → logs. (what is causing the prob.)

* saturation: degree to which the resource has extra work which it can't service, often queued

Coursera: SRE - Google

Synthetic and End-user monitoring:

Synthetic	Real User
measured thru AppDynamics, DynaTrace	
* event programatically generated	* event from user traffic

- distributed tracing: → including trace libraries and configure spans that will allow code en tracing.
  trace spans generated → asynchronously and despatched to a persistence store hooked to the backend analytics eng.
- Observability driven dev.:

Developers → [You build / you run / you monitor] ..... → Merge → only proper observability are baked in the code

developers will have the ability to debug in prod.

* Tech. should enable dist. tracing and tracing the breadcrumbs built in the sys.

Platform Engg. & AI Ops

* Platform SRE: approach to solve the org scalability challenges such as fragm, inconsistency & unpredictability.

- AIOps → anamoly detn
- AI Ops Greases ITOps Wheels → Slide

AI Ops → automate response, intelligent routing
→ anamoly detn
→ understanding of customer patterns

Data Ops: data transform & enrichment → to ensure actionable insights
← operationalizing high-perf. and scalable data fabric.
⇓
which can handle mixed workloads, diff. data types from sensors

consolidtn to a single source of truth

sre.google.com/sre-book

* Prescriptive : 30 mins before the incident occurs.

08/15:

Incident Mgmt:

Time between incidents → (MTBF) → mean time between failures.

* OODA Loop

observe - orient - decide - act

* Incident metrics to Biz. impact

CLR - closed loop automated response

SRE - improve the first call resol^n rate (FCR)

Break the Glass: → should be highly restricted.
→ available to SRE Team.

for zero trust networking should be available only from specific loc^{n}s. | e.g: panic rooms

mech. for BCP/DR.

* ~~Fault~~ in unplanned

should be tested regularly by the team(s) responsible for prod^n services, to make sure it funct^n when you need it.

Swarming: collabor^n based process
no teeming support groups

AI/ML use cases: increase first-touch resolution

⊛ Telemetry - process of recording the behavior of your sys.

Atomic Habits - Book
Lateral Thinking → Edward

* AI-ML Use cases

Chaos Engg: ID based zero trust arch.
adversely focussed

most common adopt^{n} driven - cloud transform^{n}
not abt breaking things in Prod^{n}
Failure modes should be derived from the real world

Game Day → What
Who
When
Where
How

max^{m} visibility

GameDays → $/p → C.E

- Pre-requisites:

Instrument^{n}
social awareness

- DERT (Disaster Recovery Testing) → wheel of misfortune

Module-8 (SRE is the purest form of DevOps)

Table-top exercises
VQMOM
vision/values/methods/obstacle/measures

SRE and Incident Response Management

- Ability to respond and resolve incidents
  - ↳ true indicator of operational capabilities

- Key Responsibilities:
  - Manage incidents
  - Debugging the sys.
  - right mitigation
  - Org. the incident response if it req. broader coordination.

- defect prevn
- Strategies for deploy / roll back / roll forward (feature flags, blue green, canary)
- auto remediation
- reverting sys. to safe state

- Best practices for inc. mgmt:

Prioritize	Prepare	trust	introspect	alternatives	practice	Change it around
↓	↓	↓	↓	↓	↓	↓
Stop the bleeding restore service and preserve evidence for RCA	document IM procedures in adv.	give full autonomy to all inc. participants	pay attn to your emotional state while responding to an incident	consider your options and re-evaluate	use the process routinely	encourage every team member to acquire familiarity with each me.

- Key principles of incident response:
  - maintain a clear line of command (recursive separatn of responsibilities)
  - designate clearly defined roles (inc commander, operational work, comm, planning)
  - Keep a working record of debugging and mitign
  - declare incidents early and often

3 C's:

- Coordinate → response effort
- Communicate → between incident responders within the org and outside also.
- Control → maintain control over incident response

Vertical functional aligned teams that operate inside a command and control approval system,

(Traditional ITSM)

Migration

Horizontal alignment self-regulatory system

(DevOps + SRE)

OODA Loop

Observe → Orient → Decide → Act

Observe

- Monitoring
- observability

M → metrics (aggregated set of meas. grouped or collected at reg. intervals)
e.g error rate, response time throughput

E → events
L → logs
T → traces

→ (discrete actn happening at any moment of time. Events - records of sig. things)

→ debugging data and detailed context around an event.

→ Shapes of causal chain of events when applied to a single thread of req.

Orient

* Inc. commander
* mobilizn
* Coordinate
* Communicate
* control
* wartime vs peacetime

Decide

* Identify investign and repair actns and delegate actns to service experts.

*
bad deployment - roll it back

web applⁿ stuck - do a rolling restart

event flood - validate if automatic throttling is sufficient

data centre outage - validate if autⁿ has removed bad data center

degraded service behavior without load - gather forensic data and rolling restart

- Tie Incident metrics to Biz. impact:
  - no. of outbound notn per sec.
  - no. of orders per sec.
  - no. of stream starts per sec.

- Incident Response Roles:
  - Commander
  - Liaison
  - Operations
  - (CLO)

- CLR: Closed loop/automated response

* SRE ····> Key responsibility → automate the response processes
  manual response: expensive on time and resources

* Short circuit response time through CLR:
  ↳ escalate to the concerned team for investign

- Break Glass: (act of checking out a sys. acct. p/w to bypass normal access control procedures for a critical emergency.)
  e.g. a junior physician would be able to perform certain tasks of a senior physician in case of emergency.

* all uses of BG mech. → should monitored
* BG mech should be tested regularly to ensure proper functneng.
* BG mechn for zero-trust networkeng should be available from specific locn (e.g panic room)
* BG mechn → restricted usage → SRE team

Swarming: (apply organizational context)

* collabor^{n} based process
* no tiered support groups
* case should move directly to the person most likely to resolve it (direct alert to the SRE of the product)
* no escalat^{n}s from one group to another

Escalat^{n} based process:

L1 ——→ L2 ——→ L3 (streaming)

AI/ML use cases:

* auto categ^{n} of enc. using chatbot
* remove blind spots (ML algo. to detect how changes impact dynamic IT landscape)
* intelligent routing (enables alerting with context and notifying teams with actionable alerts) → minimizing MTTI)
* automatically fulfil basic requests (NLP can help chat agents handle categories of req. and enc.)
* increase first-touch resolutions
* anamoly detn by flagging unusual repeat incidents (control the blast radius by clustering related incidents)
* use predictive analytics to flag reqs that could violate SLAs
* predict impact
* RCA
* automate remediation (Semi-autonomously remediate with little human interv^{n})
* event resolut^{n} guidance

CHAOS ENGG:

- drivers for CE:
  - Complexity of sys.
  - messy sys. engg.
  - changing security landscape (ID based zero-trust arch. adversety focussed)

* 50% of incedents → due to internal issues

remaining 50% → malicious or criminal attack

- CE: discipline of experimenting on a dist. sys. in order to build confidence in the sys. ability to withstand turbulent cond^n^s

* most common adopt^n^ driver — cloud transf^n^
* not an advanced tech.
* not about breaking things in prod^n^
* Failure modes — derived from the real world
* minimize the blast radius
* have an emergency stop.
* not about injecting random chaos exp. into the sys. and seeing what happens
* start — in a controlled env.
  stop — immediately if things go badly

- Stop your docker service.
  add delay to your network and chk for resiliency
  burn CPU with stress
  disaster recovery → periodically switch over sites
  fill up storage and see sys. behavior
  detach storage volumes
  use traffic shaping to introduce Network packet loss and network packet corruption
  Kill services and check for graceful restore
  mess up with your /etc/host and check behavior

- Gamedays:
  - * get relevant people in a room who are responsible for a sys. or a set of sys.
  - * shut off a component that the sys. should be robust enough to tolerate without it
  - * record the learning outcomes and bring the comp. outcome

UAT Env.

- Chaos MonKey story:
  - * born from cloud transf^{n} (migr^{n} from DC to cloud)
  - * put coel defined prob.. in front of engg.
  - * Terminate VMs or random VPC instances

- CE Myths:
  - * advanced capability (certain level of sophistication within your sys. in order to undertake CE)
  - * adds more chaos into the sys.
  - * only valuable in prod^{n}

- Gremlin: destruct^{n} as a service
- Properties of a Chaos Exp:

Game Days allows you to perform exp. with max^{m} resebility.

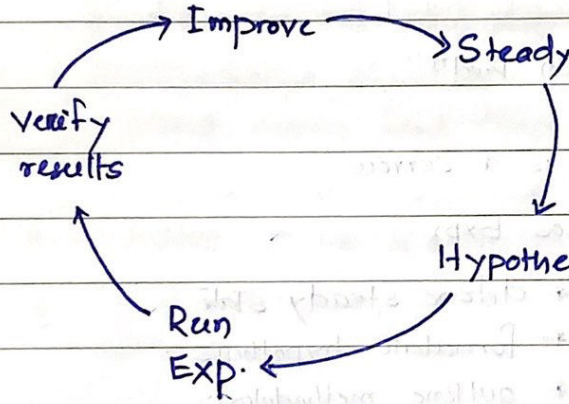
  - * define steady state
  - * formulate hypotheses
  - * outline methodology
  - * identify blast radius
  - * observability - key
  - * readily abortable

- Develop a learning culture about failure:
  * sys. safety
  * building safety margin into sys.
  * replace blame culture with learning culture
  * telemetry, expn and enstrumn

- GameDay Exercises:
  * 2-4 hrs in length
  * cross-functnal teams
  * focussed on increasing resilience
  * used for manual CE
  * great intro. to CE

Recommendations
i) use GameDays for need C. Exp
ii) Integral Exp. Deployment on need targets
iii) Proving need C.E Tools
iv) get everyone in the same loc^n

- State of Chaos:



Quantifying Recall:
* time to detect
* time for notn/time for escaln
* time when graceful degn kicked-in
* time for self-healing to happen
* time to recovery
* time to all-clear and back to normalcy

Experiment LifeCycle:

1) perform a GameDay exercise
2) validate exp. hypotheses
3) remediate findings and repeat exp.
4) once successful: automate exp.

GameDays: (The Basics)

* plan and org. gameday exercise
* dev. and evaluate C. Exp
* execute five GameDay oprns
* conduct pre-incident reviews
* automate & evangelize results & actn taken

Pre-requisites for CE:

* instn to be able to detect degraded state in your sys.
* social awareness (buy-in)
* expectn that hypotheses should be upheld (fix what is known to be broken) before C. exp starts
* alignment to respond
* cultural infrastructure

→ able to detect diff. in degradation between a control grp and an experimental grp

- pro-grade observability (before CE) is a myth.

- CE experiments focusses on
  - security weaknesses
  - security inc. response
  - security observability

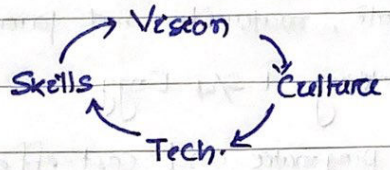
- ChaoSlingr: open source framework for security focussed C. Exp

- CE @ Google: (improving dist. sys. reliability)
  * DiRT (Disaster Recovery Testing)
    ⇓
    developed to
    find critical vulnerabilities
    determine tech. robustness
  * comm^n protocol
  * revert/rollback for every step
  * processes, humans, comm^n → test everything that is involved in the MTTR processes.

SRE - Purest form of DevOps

- SRE: making machines work for humans
  - ↳ implement sys. that fix themselves.
- metrics for success:
  - * deployment freq
  - * lead time for changes
  - * time to restore service
  - * change failure rate
- Impln Pattern:
  - * orgn current state, maturity and priorities
  - * SRE: sys. thinking + s/w Engg.
  - * SRE - create pragmatic and cost-effective decns across the entire product life cycle
  - * SREs: bring singularity
    - ↓ (collaborators)
    - : create envs and autn that will self-enable sys. to run meeting SLAs
    - : work on the entire continuum between s/w design, dev, testing, deployment, oprns and cust. exp.
- 1. understand cust. expectations
- 2. start small on obvious areas and scale out
- 3. a) identify need areas of sys. engg and apply innovn as needed
  - b) Target areas: cloud mgn
    - CI/CD autom
    - Observability
    - ops autom
    - } strategies.

- Impln Roadmap:
  - i) Blueprint for SRE
  - ii) Assess / Maturity / Roadmap : identify bottlenecks in flow of value
  - iii) Best Practices
  - iv) Enablement → training strategy defined
  - v) Governance → enterprise perf. dashboards
  - vi) Evangelism → cross pollinate learnings
  - vii) Rinse and repeat : refine & review (continuous improvement)
  - viii) Rapid SRE transf^{n}



- Build Stage Observability:

* measure and optimize DevOps perf. to maximize ROI
  - → optimize s/w delivery with goal based KPIs
  - → support dec^{n}s with data driven recommendations
  - → identify trends and predict release risk

- Run stage observability:

* Server
  - infrastructure
  - appln ecosystem
    - appln
  - biz. parameters
  - security analytics

- SRE Execution:
  - SREs are
* ^ coders and Know the toolset of the product thoroughly

* ensure consistency of tooling
* error budgets remove conflict and help self-regulate flow.