Help protect your datacenters with safety constraints

Etienne Perot and Christina Schulman, Google
Datacenter automation at Google

- Datacenter machine management is complex
- It’s easier to safeguard the automation than to fix everything that uses it.
Datacenter automation at Google

Google uses automation to handle datacenter machine activity

- repairs
- installs
- decommissions
Educational Experience #1

That time we erased our entire Content Delivery Network
Educational Experience #1

- Engineer attempts to manually send 1 rack of CDN machines to Diskerase
Educational Experience #1

- Query bug causes ALL the CDN machines to go to Diskerase
  - Result: slow user queries, internal network congestion, 2 days of manual cleanup
Educational Experience #2

That time we decommissioned all our Tunneling Load Balancers
Educational Experience #2

- Dedicated switches used to be used as TLBs for all traffic entering the datacenters
Educational Experience #2

- A utility script was used to send retired switches to decom
Educational Experience #2

- Whoops. The underlying data has changed.
- Specifically, the script now matches all the TLBs as retired.
Educational Experience #2

- We got lucky: TLBs kept serving because they didn’t know they’d been decommed.
How can we prevent this?

- Completely different root causes
- But: common patterns for root causes
Common Failure Patterns

- Overmatching / inadequate limiting
- Code rot / changing nature of data
- Complex interdependent systems
- Unsafe releases and rollouts
So how do we protect our machines?

- Common patterns, but different systems
- Different root causes
- Same mechanism of destruction
So how do we protect our machines?

So use a central mechanism to mitigate risk
So how do we protect our machines?

So use a central mechanism to mitigate risk

- and bake it into your automation
{{Magic transition slide}}
Safety Constraint Checking as a Service (SCCaaS)

- Production infrastructure at Google: It's Complicated™.
- But:

  "Production Shall Keep Running."
  
  *(encoded as: "SLOs Shall Be Respected.")*

- Let's write an RPC service to keep this true!
SRSly?

- "Are you serious?"
- Est. 2009
- Prevented many outages.
What can go wrong at all?

● Enumerate production workflows.
● Figure out blast radius.
Example workflows

- Machine upgrades
- Storage drains
- Migrating VMs
- Pushing datacenter-wide configs
- Shutting down racks
Now what?

- Sanity-checks and rate-limits
- Look at your SLOs for inspiration!
Constraint pattern #1

- Rate limits:
  Allow $N$ things per period per bucket.

"Allow at most 1% of TLBs per 1h per datacenter to be sent to decom."
Constraint pattern #2

- Concurrency limits:
  Allow at most $N$ concurrent things per bucket.

"Allow at most 5% of CDN machines per datacenter to be rebooting before allowing more."
Constraint pattern #3

- Sanity/policy checks:
  Only approve `thing` if `condition` is true.

"Can only reboot a machine that has no VMs running on it."
Constraint pattern #4

- Service-specific health checks:
  Prevent disruption to service if it is bad.

"Can't impact Google Web Search if its oncaller got paged recently."
Constraint pattern #5

- **Automatic braking:**
  Stop approving *things* if *recent* approvals caused *pain*.

"*Don't upgrade rackswitches* if *recent* *rackswitch upgrades* resulted in *broken rackswitches.*"
API

Check(\texttt{Entity}, \texttt{Intent}) \rightarrow (\texttt{bool}, \texttt{string})

- \texttt{Entity}: What is being affected.
- \texttt{Intent}: What is being done.
- Returns:
  Whether it's safe to go ahead, and why/why not.
Request handling

- Data gathering
- Map to constraints
- Evaluate constraint 1
- Evaluate constraint 2
- ... (dotted)

Evaluate constraint 1
Evaluate constraint 2

Result

- All "yes"
- Nope (no wai)

Record approval

Return "yep"

Return "nope"
Safety² constraint service

● SRSly's configuration itself can be bad

How to avoid?

● Regression tests for config mapping
● Internal sanity checks
● Big Red Button™
● Shard it! Slow rollout!
Behavior overrides

- Want to do Something Special™?
  - Roll out kernel faster to patch a vulnerability
  - Prevent extra disruptions during demos

- Override behavior!
  - Force approval/rejection, disable constraint, tweak params
  - Auto expiry & max duration
  - Keyed by Entity and/or Intent
Enforcing safety checks
● Production gets more complicated over time
● Automation can go horribly wrong
● Apply defensive design
  ○ Protect it {early, often, well}
Questions?

Etienne Perot and Christina Schulman, Google