Production Population Control: My Cattle Are Rabbits!

Alex Nauda
CTO, Nobl9
Evolution of infrastructure environments

until 2010

- Pets
  - High maintenance
  - Long-lived
  - Precious
  - Named

2011-2015

- Cattle
  - Lower maintenance
  - Shorter-lived
  - Disposable
  - Tagged

2016-2020

- Rabbits
  - Unmaintainable
  - Temporary
  - Pests
  - Forgotten

TOO MANY
How many environments?

Per application:
- ~1 Production env
  - (max 2-3 if HA)
- 3 Test envs
  - Int, QA, Staging

Until 2010

Per application:
- 2-3 Production envs
  - Active/Warm/Cold
- 6-9 Test envs
  - Each team, CI, branch

2011-2015

Per application:
- >10 Production envs
  - Multi-region and -cloud
- Countless Test envs
  - Dynamically stood up and torn down

2016-2020

MULTIPLYING
How many prod environments do we need?

Production:
- Regional HA - One for every supported region of every supported cloud
- DR - Active/active/passive/warm/cold
- Blue/Green - A blue for every green

Often no traffic
How many test environments do we need?

Test:
- Shared, hosted development environment
  - ...per team
- Individual developers' hosted environments
  - ...for at least some back end devs, but everyone wants one
- CI environment for every pull request
- QA
- Staging
- UX validation
- Sales demo
- Pen testing

Often no traffic

Also, who responds to alerts for these?
Problem statement

All of these environments are
• Unmanageable
• Expensive
• Overstimulating
• Diverse
  ○ Variety of use cases and traffic patterns
  ○ Differing requirements for reliability

We need appropriate SLOs to monitor these environments
Properties of SLOs: 1. Low context

Not like this

Like this!

SET AND FORGET
Properties of SLOs: 2. Easily tunable

Not like this

Like this!

TURN THAT DOWN (OR UP)
Must be automatable (Gitops, IaC)

CI / CD

Infrastructure (hosts, clusters, network)

Applications

SLOs

Repos
What to pick for SLIs?

Traditional SLIs depend on user activity
- Key business metrics
- Tied to user satisfaction
- Who even are the users of your test envs and your DR cold site?

What can we do instead?

- First of all, we can use an occurrences-based SLO
- Next, we can monitor for lights-on behavior
- Finally, we can use synthetics
  ...and we can do that across the entire stack
Occurrences-based SLOs

(a/k/a Event-based SLOs)
Measure a straight ratio of successful events (requests, jobs, attempts)
...as opposed to time slices - good minutes vs bad minutes
This can be used with any counter type metric

So they automatically adjust for low traffic periods
Lights-on SLIs for infrastructure and clusters

- **kube-state-metrics**
  - Are my nodes on?
    - \( \text{count(kube_node_status_condition\{condition="Ready", status="false"\})} == 0 \)
  - Are my pods on?
    - \( \text{sum(kube_deployment_status_replicas_unavailable)} == 0 \)
    - \( \text{sum(kube_daemonset_status_number_unavailable)} == 0 \)

- **Is my k8s not broken?**
  - Kubernetes API response time (Occurrences method SLO)
  - Pod start latency (Occurrences method SLO)
Lights-on SLIs for applications

- Hosting-specific ingress or load balancing
  - e.g. AWS ALB unhealthy count == 0

- Latency of requests/events (Occurrences method SLO)

- Success of requests/events (Occurrences method SLO)
Synthetic SLIs for infrastructure and clusters

Kuberhealthy

○ DNS resolution
  ■ \( \frac{\text{avg(kuberhealthy_check\{check=~"kuberhealthy/dns-status.*", status="1"\})}}{\text{avg(kuberhealthy_check\{check=~"kuberhealthy/dns-status.*"\})}} \)

○ Deployment works
  ■ \( \frac{\text{avg(kuberhealthy_check\{check="kuberhealthy/deployment", status="1"\})}}{\text{avg(kuberhealthy_check\{check="kuberhealthy/deployment"\})}} \)

○ Daemonset works
  ■ As above; check="kuberhealthy/daemonset"
Synthetic SLIs for applications

- Have automated functional tests
  - Popular tools in this space
    - Mabl
    - Runscope (back end)
    - Ghost Inspector (front end)
- Run them regularly in every environment
  - In production, use an isolated account or tenant
  - In test environments, run them every so often to give your SLOs something to measure to detect bad deploys
  - Example SLO:
    - Total tests succeeding / total tests run over 1 hour
    - Very strict target in production
    - Lower target for lower order environments
Define clear ownership of non-production SLOs

Developers need autonomy in their workflow, so encourage it

- Developers should own their own individual hosted environments
- Shared development environments, QA, etc.
  - Consider assigning a rotating daytime on-call sprint by sprint
- Business-critical environments need special care
  - Treat internal business-facing environments as production
  - Lower criticality during non-working hours
Rabbit control

- Prerequisite: automatable SLOs-as-code
- Create a set of simple SLOs that work in any environment
  - Set and forget (until they alert)
- Work around the issues caused by sparse traffic
  - Monitoring lights-on behaviors
  - Making the most of synthetics
- Define clear alert policies and ownership for these generated SLOs
Thank You

Alex Nauda
CTO, Nobl9

SREcon Slack Alex Nauda

Twitter @alexnauda

Email alex@nobl9.com