8ms/99th write percentile latency - is it fast?

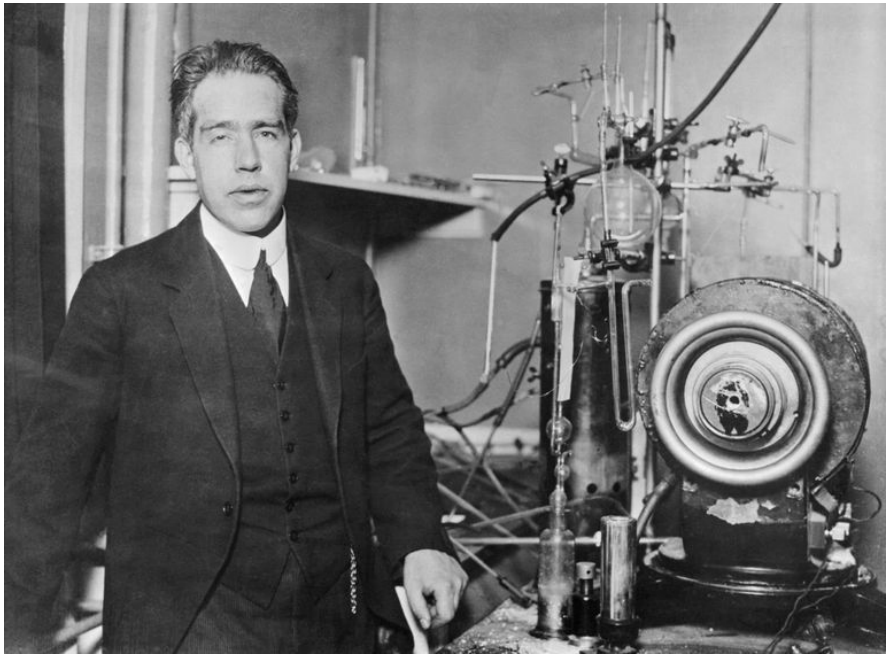
Understanding the importance of "SRE implements devops"

Maciej Lasyk

8ms/99th write percentile latency - is it fast?

Understanding the importance of "SRE implements devops"

Maciej Lasyk



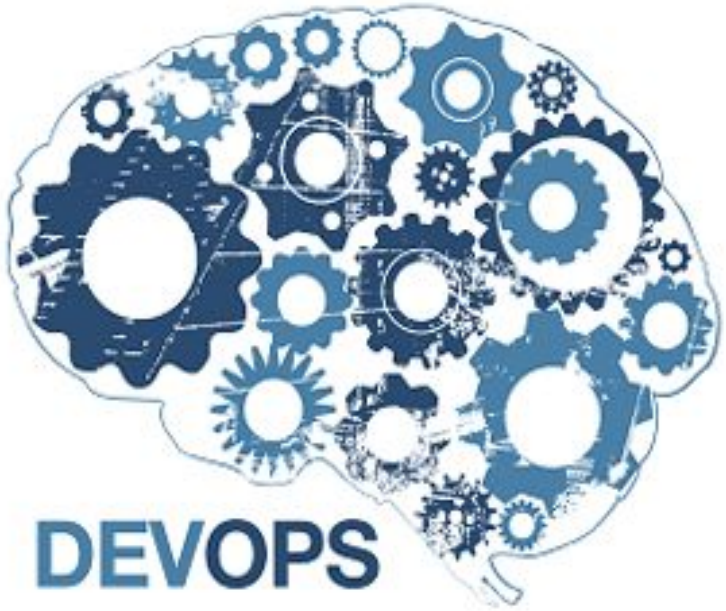
An expert is a person who has made all the mistakes that can be made in a very narrow field

Niels Bohr
(Nobel Prize, quantum physics)

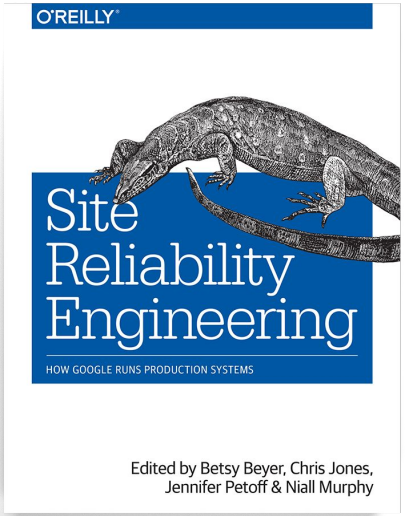

- PUCHACZ -
SP-3366

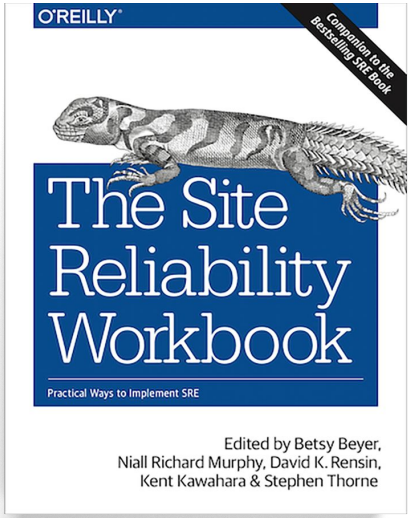
Flickr: morrosv7


/Theory/In/Practice
Web Operations
Keeping the Data on Time
O'REILLY®
John Allspaw
& Jesse Robbins


DEVOPS

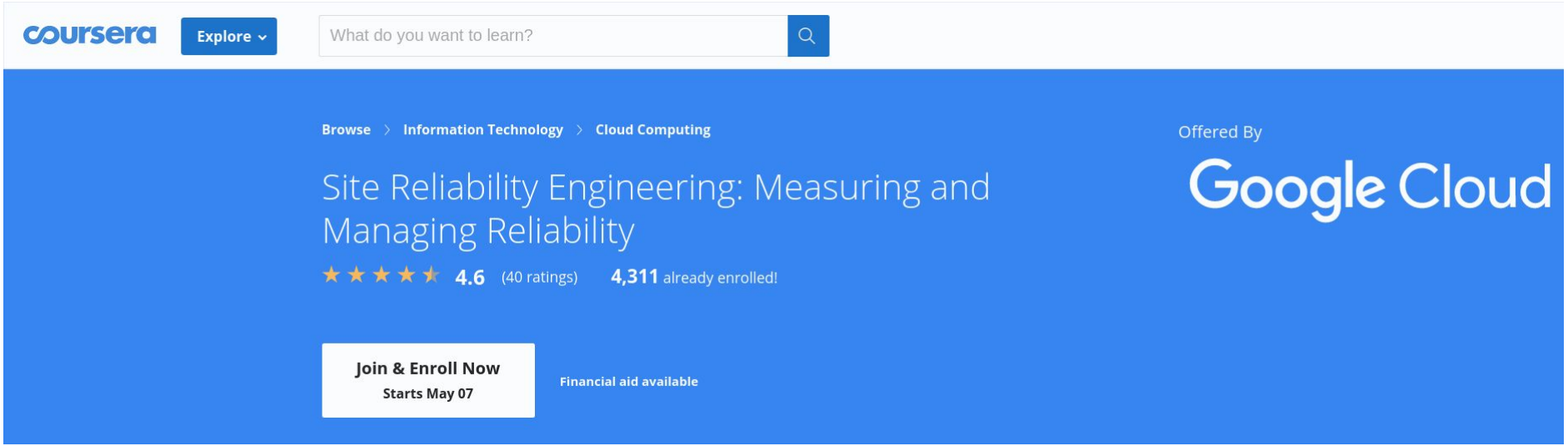
The Google approach



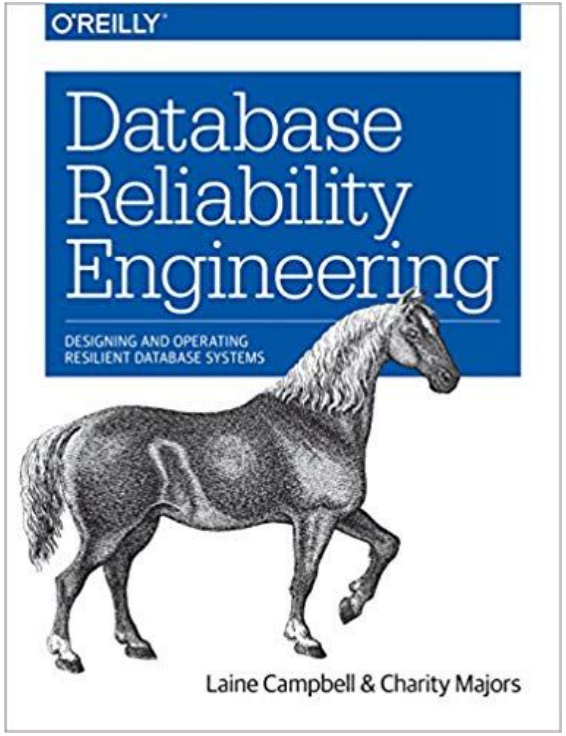


https://landing.google.com/sre/books/

The Google approach



https://www.coursera.org/learn/site-reliability-engineering-slos



https://charity.wtf

Awesome Site Reliability Engineering

A curated list of awesome Site Reliability and Production Engineering resources.

What is Site Reliability Engineering?

> "Fundamentally, it's what happens when you ask a software engineer to design an operations function." - Ben Treynor Sloss, VP Google Engineering, founder of Google SRE

Contributing

Please take a look at the contribution guidelines first. Contributions are always welcome!

Contents

https://github.com/dastergon/awesome-sre

What is SRE about?

Are our products working reliable so users are happy?

How much reliability - related work is required?

How to maximize effort on feature development?

Tell me more about SRE

SREs take care of products reliability

Tell me more about SRE

SREs take care of products reliability

At Codewise developers are oncall

So.. developers are SREs!

The CRE principles

- **reliability is the most important feature**
- **users, not monitoring, decide reliability**
- **in order to increase availability there is a need for well engineered:**
  - **software: 99.9% (40min / 28 days)**
  - **operations: 99.99% (4min / 28 days)**
  - **business: 99.999% (24 sec / 28 days)**
  - **above time is the error budget**

The CRE principles

- reliability is the most important feature
- **users, not monitoring, decide reliability**
- in order to increase availability there is a need for well engineered:
  - software: 99.9% (40min / 28 days)
  - operations: 99.99% (4min / 28 days)
  - business: 99.999% (24 sec / 28 days)
  - above time is the error budget

The CRE principles

- reliability is the most important feature
- users, not monitoring, decide reliability
- **in order to increase availability there is a need for well engineered:**
  - **software: 99.9% (40min / 28 days)**
  - operations: 99.99% (4min / 28 days)
  - business: 99.999% (24 sec / 28 days)
  - above time is the error budget

The CRE principles

- reliability is the most important feature
- users, not monitoring, decide reliability
- **in order to increase availability there is a need for well engineered:**
  - **software: 99.9% (40min / 28 days)**
  - **operations: 99.99% (4min / 28 days)**
  - **business: 99.999% (24 sec / 28 days)**
  - **above time is the error budget**

The CRE principles

- reliability is the most important feature
- users, not monitoring, decide reliability
- **in order to increase availability there is a need for well engineered:**
  - **software: 99.9% (40min / 28 days)**
  - **operations: 99.99% (4min / 28 days)**
  - **business: 99.999% (24 sec / 28 days)**
  - above time is the error budget

The CRE principles

- reliability is the most important feature
- users, not monitoring, decide reliability
- **in order to increase availability there is a need for well engineered:**
  - **software: 99.9% (40min / 28 days)**
  - **operations: 99.99% (4min / 28 days)**
  - **business: 99.999% (24 sec / 28 days)**
  - **above time is the error budget**

SLA

- Service Level Agreement: a commitment between a service provider (e.g. Codewise) and a customer
- agreement on unreliability of the service in the reliability model
- e.g. Voluum can be down for 1 hour per month and that's fine (only 99.86% availability!)

SLO

- Service Level Objectives, a key element of SLA
- SLO is a measurement characteristic (availability, throughput, response time, quality)
- SLO = successful events / valid events [%]

SLI

- Service Level Indicator
- low level metrics and monitoring checks
- SLI provide data for SLOs
- it’s a measurement of user’s expectations

Error Budget

- How much of downtime is acceptable?
- availability = successful requests / total requests
- 100% - target SLO = error budget

Error Budget

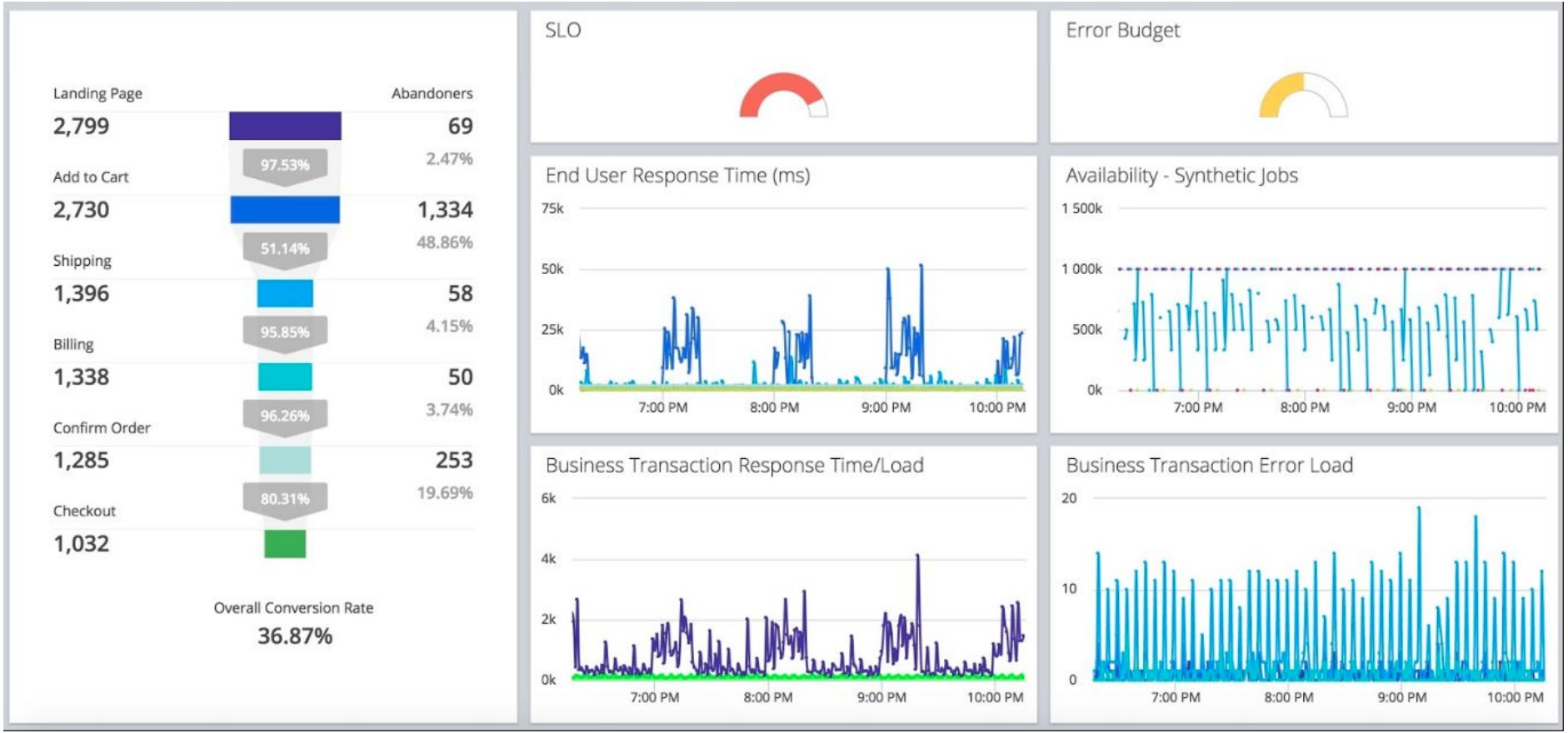
SLO = 95% of responses return within < 1s

Error Budget = 5% responses >= 1s

Error Budget

Target SLO	Allowable Downtime (per 30 days)	Likely Requires
99.999% (5 nines)	0.43 minutes	Automated Failover
99.99% (4 nines)	4.32 minutes	Automated Rollback
99.95% (3.5 nines)	21.6 minutes	
99.9% (3 nines)	43.2 minutes	Comprehensive monitoring and on-call system in place
99.5% (2.5 nines)	216 minutes (~3.5 hours)	
99% (2 nines)	432 minutes (~7 hours)	

Visualizing reliability



Toil

- manual and repetitive work that could be automated
- nontactical / reactive work (e.g. flapping monitoring)
- resolving performance issues with no value for the future
- SRE defines Toil as a waste
- Toils should be removed - perfectly by automation!

Setting up the reliability

Setting up the reliability: the culture debt

SRE roadmap

Created by Maciej Lasyk
Aug 17, 2018

During one of our meetings we've discussed a list of subjects we'd like to take care of in this SRE chapter.

Following is a picture of that discussion followed by a list of those topics and results of our vote for importance:

1. self - healing (7 votes)
2. improve monitoring, metrics, alerting (6)
3. high availability (3)
4. rest
    a. planning for failure (1)
    b. logging (1)
    c. chaos monkey (1)
    d. disaster recovery (1)
    e. capacity planning
    f. operational knowledge

Setting up the reliability: the culture debt

SRE group meets regularly. Here you'll find meeting notes as well as any other informations related to it:

- SRE meeting #1: Hello world!
- SRE meeting #2: Where to?
- SRE meeting #3: The Roadmap
- SRE meeting #4: Self - healing (part 1)
- SRE meeting #5: Self - healing (part 2)
- SRE meeting #6: Self - healing (part 3)
- SRE meeting #7: reactivation
- SRE meeting #8: Infra as a Code (part 1)
- SRE meeting #9: Infra as a Code (part 2)
- SRE meeting #10,#11: SRE follow - up: SLA/SLO/SLI
- SRE meeting #12,#13, #14: planning for the failure
- SRE meeting #15: logging

Setting up the reliability: postmortems

SRE group will discuss how this page should look like, what the postmortem template should consist of and any other details.

- Post-mortem 2018-07-06: Cassandra upgrade 2.2->3.11
- Post-mortem: Cassandra, you naughty! 2018-03-02
- Post-mortem: Config-server unstable/down - TEST - 15.06.2018
- Post-mortem: h2o down - GitHub API increased error rates - TEST - 9.08.2018
- Post-mortem: Removed AMI used in production - 10.11.2017
- Post-mortem: Zeropark, 2018-06-29
- Post-mortem: Zeropark, 2018-07-04

Old VoluumDB Post-Mortens / System-Owner's log: https://docs.google.com/document/d/1viuWr6Gs6r3Ex-kOtMWCSFv_-mmWl3sw9yAtUZ978N8

"attack the problem, not the people"

Setting up the reliability: service catalog

Team	Service name	Severity of service (1-3, where 3 is critical and 1 is TV dashboard)	infra provisioning tools	infra provisioning repos	Monitoring tool (Sentinel, Cloudwatch, Datadog etc)	Alerting (Pagerduty, Pingdom etc)	Self - healing (yes / no / partial)	Self - healing (yes / no / partial)	Failure scenarios (yes / no)	Failure scenarios related URLs
infrastructure	Cassandra	3	TF & Ansible	click	Sentinel, Datadog, Tornimo	Pagerduty, Tornimo	no	we're working on self - healing (Datadog triggering Rundeck jobs for resolving problems)	yes	GH milestone: https://github.com/codewise Test reports: https://docs.google.com/docu Documentation for procedures: https://githu
infrastructure	Cassandra-Reaper	2	TF & Ansible	click	Datadog, Tornimo	Pagerduty, Tornimo	no	we're working on IT; ASG + S3	no, working on it	
infrastructure	Rundeck	3	TF & Ansible	click	Datadog, Tornimo	Pagerduty, Tornimo	no	we're working on IT; ASG + S3 & RDS	no, working on it	
infrastructure	etcd	3	Ansible	click	Sentinel, Datadog	Pagerduty	no	etcd is going down in the nearest future	yes	Documentation for full recovery: https://gith

Introducing SRE between product and devs

Event Details Find a Time

Googlers shared publically how they work with their customers in terms of applications reliability and they want everyone to learn from their mistakes.

One of my favourite quotes from the training I did on this whole subject is: “targeting specific level of reliability is a key to establish balance between need to maintain system reliability and providing new features to drive user acquisition and revenue growth”.

So - this talk is an introduction to Site and Customer Reliability Engineering (SRE/CRE) where you will learn about:

- Making applications reliability a feature so it can be prioritized
- What level of reliability is enough?
- Measuring the reliability and acknowledging some degree of unreliability
- Defining SLAs (Service Level Agreements) and introducing the “happiness test”
- Setting reliability targets, defining and understanding SLOs (Service Level Objectives)
- Working on key metrics / figuring out SLIs (Service Levels Indicators)

A couple of months ago we created SRE chapter here @Codewise. However, this is an engineering group and **we need our business to take part in this**. Customer Reliability Engineering happens between technology, product and the client. **Thus this invitation for you guys <3**

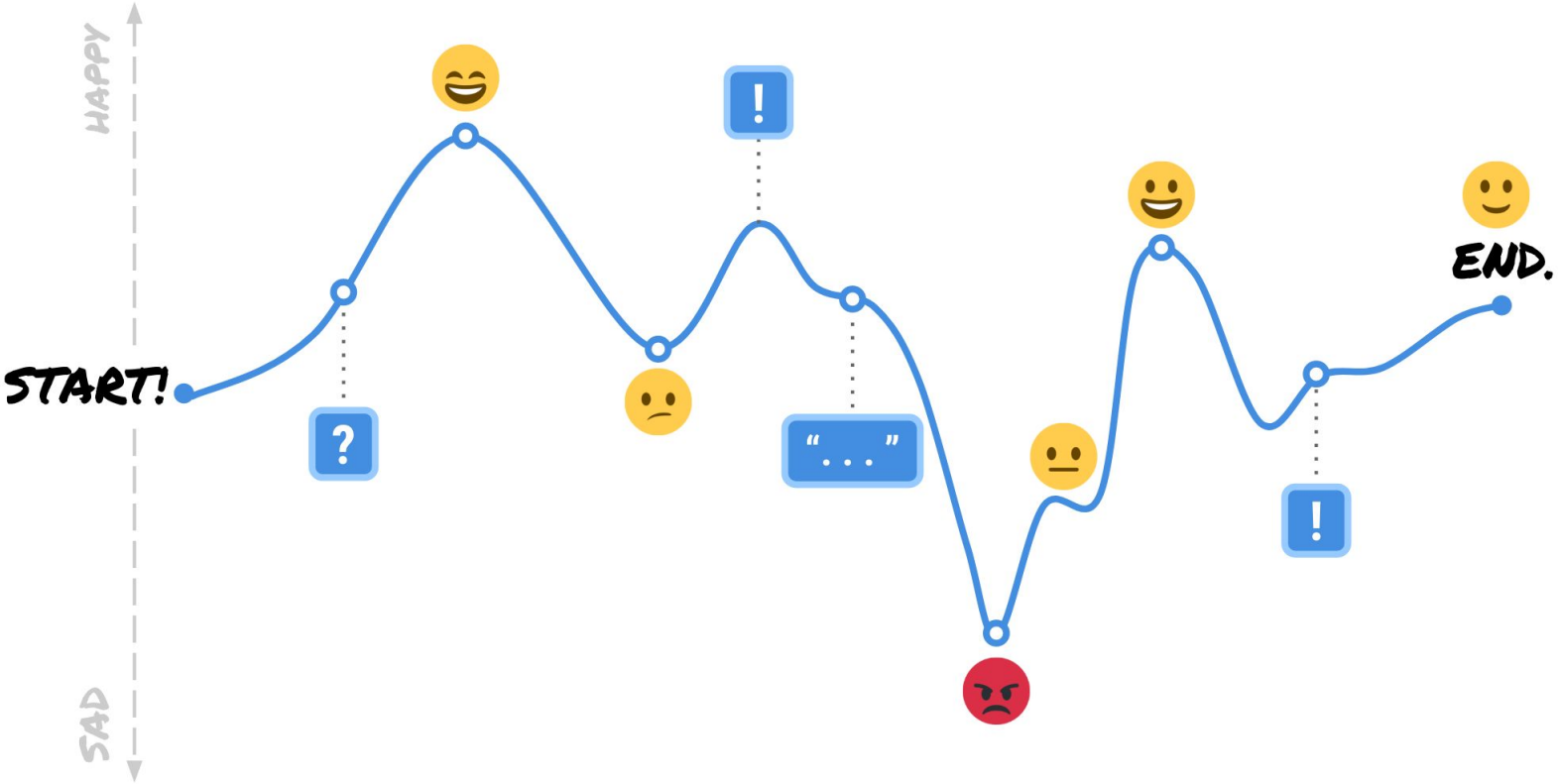
Possible business buyins for reliability

- when everything is stable - focusing 100% on features
- business will know before customers about problems
- lower chance for unhappy customers
- status pages for customers
- dashboards with reliability information
- business may decide to override sometimes

Introducing SRE between product and devs

- Get 3 perspectives:
  - developers’
  - TLs’
  - business’

User journeys and happiness (SLOs!)



https://blog.fullstory.com/customer-journey-maps-session-replay-and-the-power-of-empathy/

Consistency is the key!

Observability & monitoring

- Does your monitoring always informs you about problems?
- How long from alert to finding the source of problem?
- Are alerts actionable?
- Can you snooze notifications?
- Are devs creating checks?

Observability & monitoring

TORNIMO

Application monitoring for developers & teams that move fast

Self - healing kit

1. Bring all procedures into one place
2. Automate procedures
3. Test, use manually, build trust
4. Build more trust
5. Attach automation jobs to monitoring
6. Profit

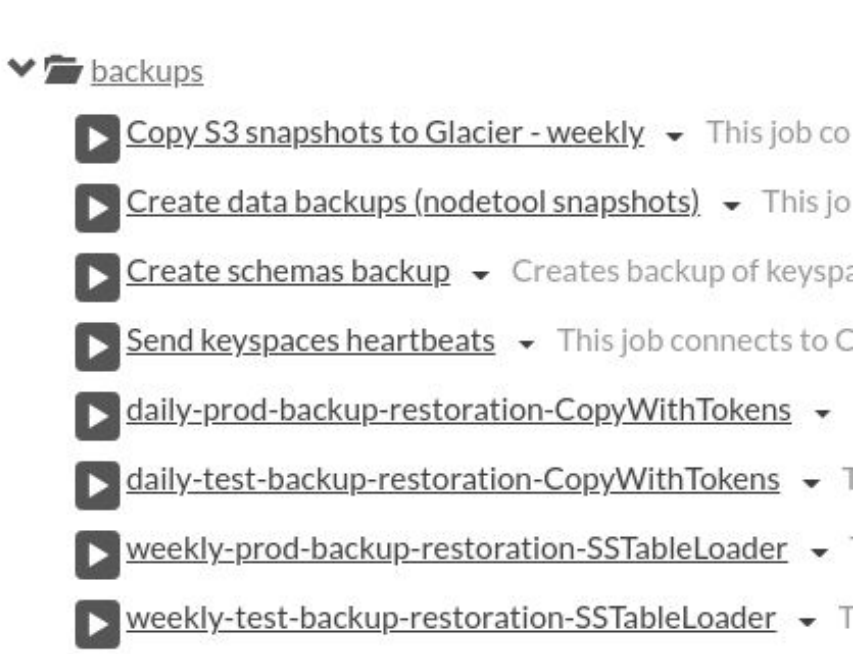
Our generic operational flow

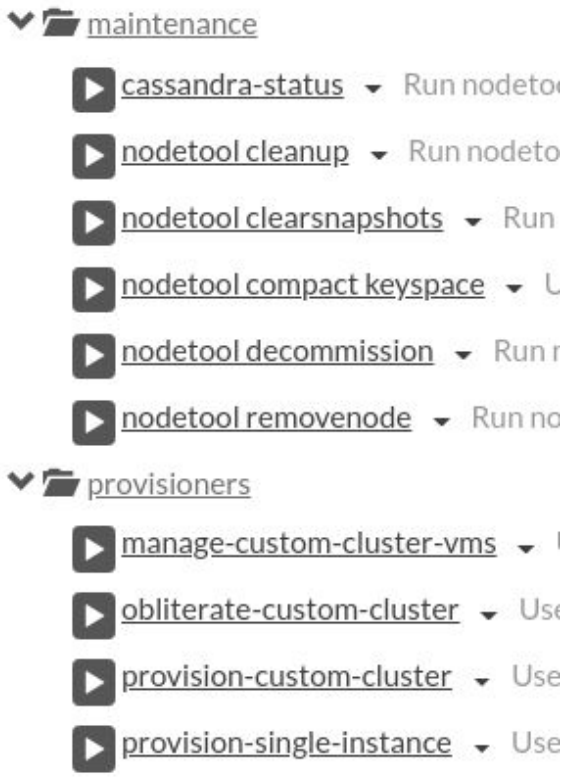
- get all procedures in one place
- put into repo
- use one scheduler (Rundeck)
- Cassandra as a service
- KISS: no k8s & Docker; VMS+ASG+LB=<3
- Just do perf tuning per instance type

Our operational flow for Cassandra

- Observability
  - Accessing and managing Cassandra logs
  - Accessing and broswing Cassandra graphs
- Operations
  - Accessing and managing container logs
  - Replacing running instance
  - Replacing dead instance
  - Whole AZ down scenario
  - Adding brand new node to the cluster
  - Removing node from the cluster
  - Adding new AZ to the DC in cluster
  - Removing AZ from the DC in cluster
  - Adding new DC to the cluster
  - Compacting keyspaces / tables
  - Cleaning up keyspaces / tables
  - Repairing keyspaces / tables
  - Changing node IP address without bootstrapping new node
  - Cluster restore
  - Resizing Cassandra data filesystem

C* operational tasks and automation





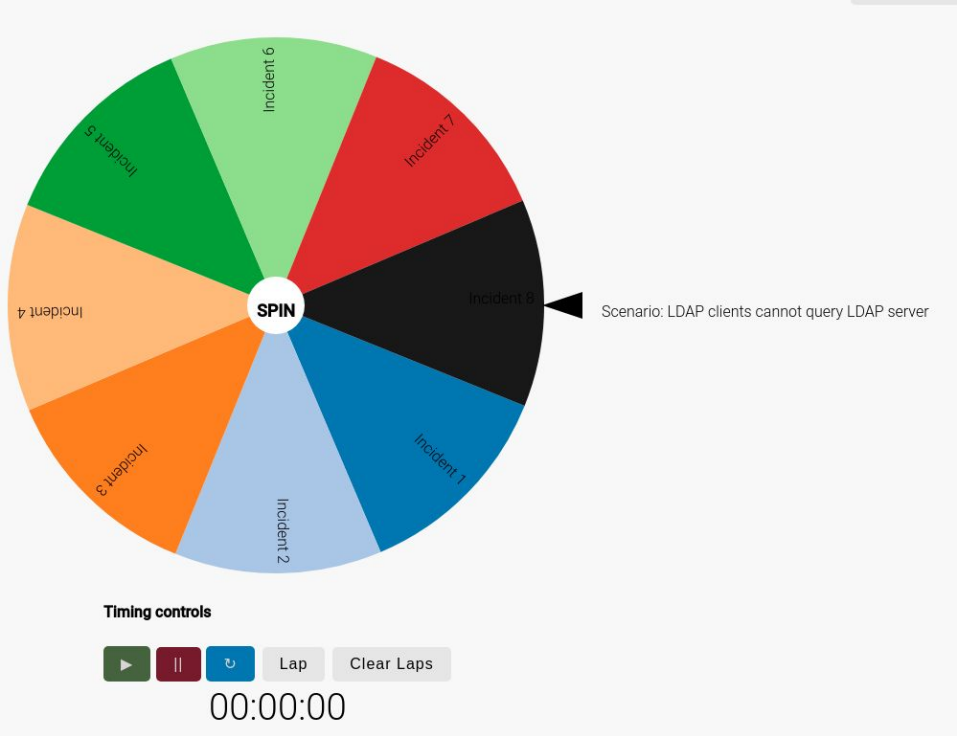
Prepare for the inevitable failure

- 3 nodes down (1 full AZ and one in different AZ) cassandra mst-c-recovery-scenarios
  #71
- 3 nodes down scenario (1 in each AZ) cassandra mst-c-recovery-scenarios
  #70
- 2 AZs down scenario cassandra mst-c-recovery-scenarios
  #69
- 1 AZ down scenario cassandra mst-c-recovery-scenarios
  #68
- Whole cluster fail scenario cassandra mst-c-recovery-scenarios
  #67
- EU failure scenario cassandra mst-c-recovery-scenarios
  #66
- US failure scenario cassandra mst-c-recovery-scenarios
  #63

Drills?

Workshops! And RPG

Or Wheel of misfortune



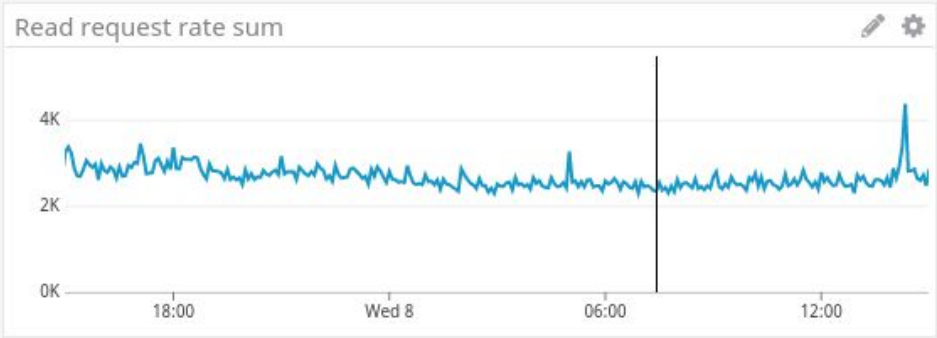
https://dastergon.gr/wheel-of-misfortune/

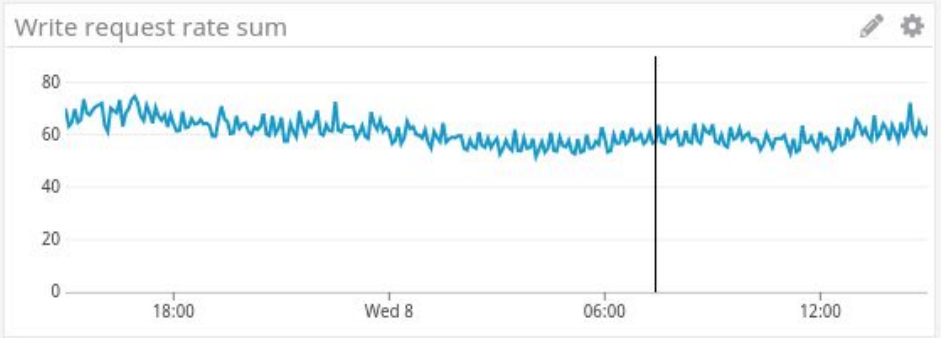
Review regularly performance and logs!

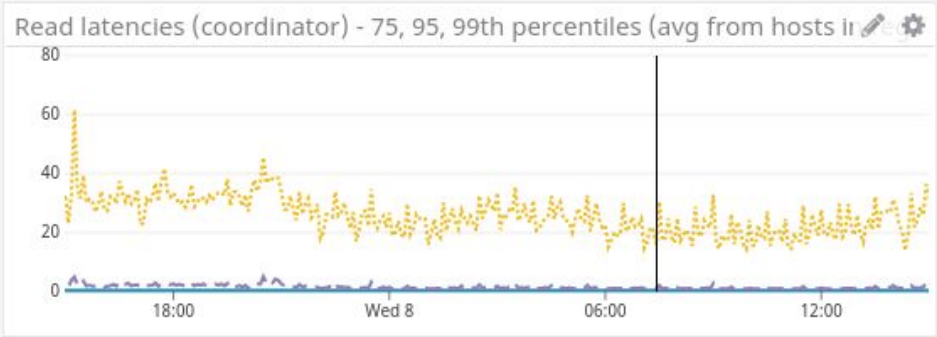
So what exactly means "reliable enough" now?

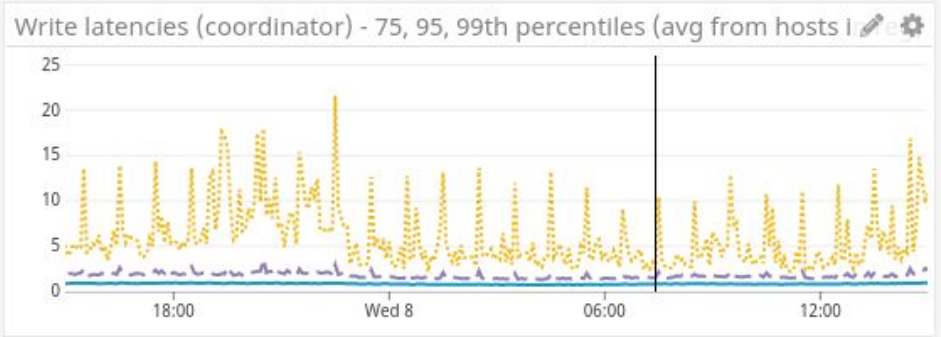
1. on-calls doesn’t really receive C* - related calls
2. But it they did they’d manage
3. Latencies are stable (we’re working on C* SLOs)
4. Latencies are not too low (we are at 50% of possible VM size)
5. neither too high (developers are happy, customers are happy)

So what exactly means "reliable enough" now?









Whatever as a service in #noops

1. Do you really need to use self - managed service?
2. All 3rd party services might be managed this way
3. Developers on-call
4. High level of automation
5. Think Redis, MongoDB, Zookeeper, Kafka etc
6. The most important thing: developers have to understand technology (e.g. Cassandra)

Summary

1. Introduce reliability to your devs
2. Introduce reliability to your business
3. Work on users journeys w/your business
4. Define SLAs/SLOs/error budgets
    - Work on SLIs
5. Gather procedures, automate, test
6. Run reliability workshops across teams
7. Attach automation to monitoring
8. Keep meetings and discussing reliability items across company

URLs

1. https://github.com/dastergon/awesome-sre
2. https://landing.google.com/sre/books/
3. https://www.coursera.org/learn/site-reliability-engineering-slos
4. https://github.com/dastergon/awesome-sre
5. https://shermandigital.com/blog/designing-a-cassandra-data-model/
6. https://www.datastax.com/dev/blog/basic-rules-of-cassandra-data-modeling
7. https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlAboutDataConsistency.html
8. https://www.slideshare.net/DataStax/replication-and-consistency-in-cassandra-what-does-it-all-mean-christopher-bradford-datastax-c-summit-2016
9. https://blog.fullstory.com/customer-journey-maps-session-replay-and-the-power-of-empathy/

Thank you, questions?

maciej@lasyk.info
maciej.lasyk.info
@docent-net
github.com/docent-net/

Cloud sphere.it

Join the conversation #sphereIT #CloudSphere

Cassandra basics - data model

- 2 main rules:
  - Spread Data Evenly Around the Cluster
  - Minimize the Number of Partitions Read
- Those rules conflict with each other
- You have to find the sweet spot

https://shermandigital.com/blog/designing-a-cassandra-data-model/

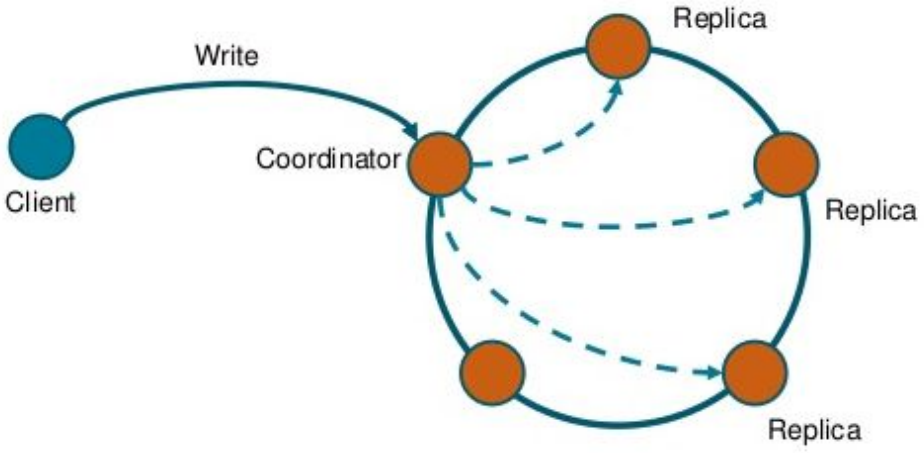
Cassandra basics - data modeling

2 main rules:

- **optimize data model for queries: duplicate data (no JOINs!)**
- **don’t minimize the number of writes, optimize for reads**

https://shermandigital.com/blog/designing-a-cassandra-data-model/

Cassandra basics - replication factor



{'class': 'NetworkTopologyStrategy', 'us-east': '4', 'eu-central': '3'};

https://www.slideshare.net/DataStax/replication-and-consistency-in-cassandra-what-does-it-all-mean-christopher-bradford-datastax-c-summit-2016

Cassandra basics - tunable consistency

- ALL (strong consistency, higher latency)
- LOCAL_QUORUM (nodes/2+1)
- QUORUM (same as above but DC-aware)
- ONE (eventual consistency, lowest latency)

Cassandra basics - tunable consistency

- read CL: how many replicas must respond to a read request before returning data to the client application
- write CL: how many replicas must respond to a write request before the write is considered successful

C* reliability notes

1. always use NetworkTopologyStrategy/EC2Snitch
2. repairs and deletes are tricky (see later)
3. don't use DTCS compaction strategy
4. use racks (multiAZ on AWS) and DCs
5. automate all operational tasks
6. rethink backup & restore: test and automate
7. disable dynamic snitch
8. GC tuning for your workload (reads / writes / mixed)

C* repairs

1. sub - range repairs and stop using incremental repairs
2. repairs are also needed for preventing resurrections
3. tombstone: new entry telling, that record is removed
4. repairs must be run on clusters where deletions occur
5. deletions are also:
    a. inserting null values
    b. inserting values into collection columns
    c. Expiring Data with TTL
    d. and smt more
6. run repairs more often than gc_grace_period