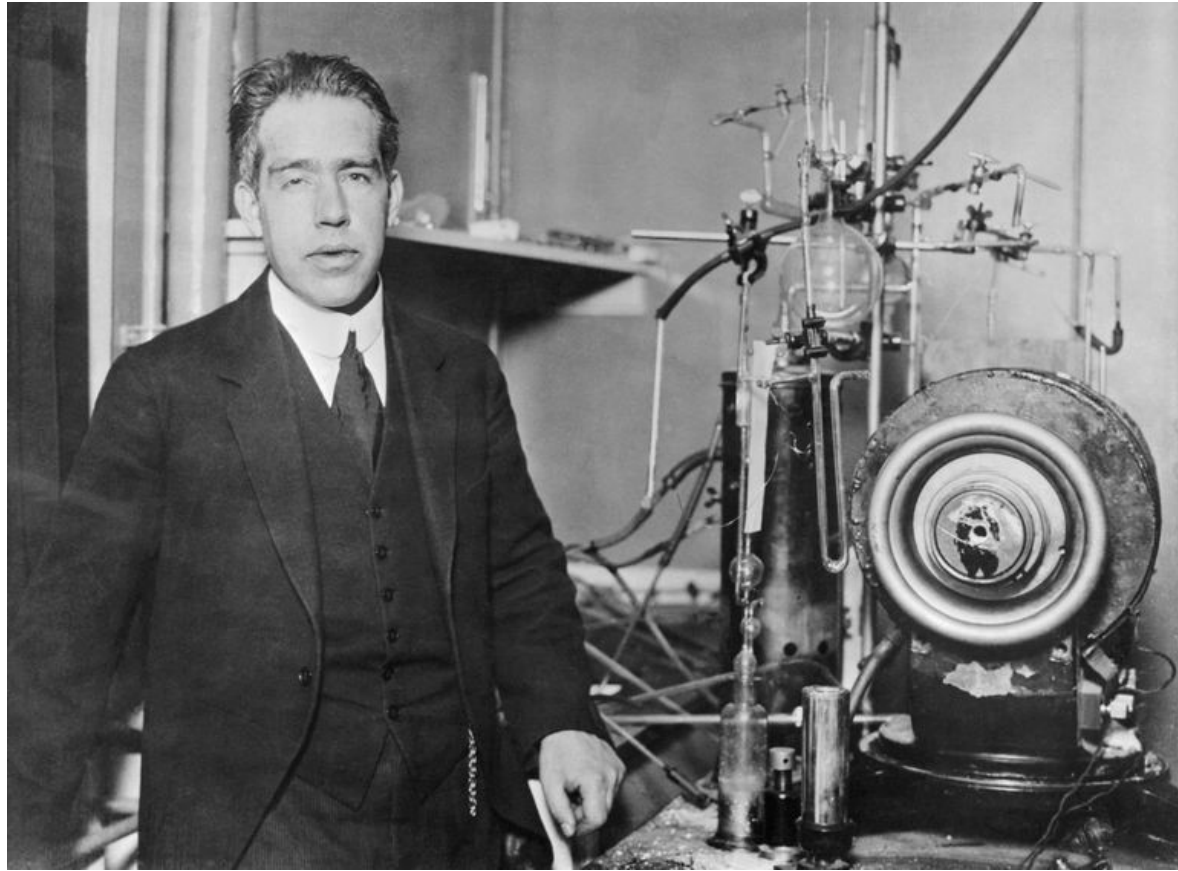


8ms/99th write percentile latency - is it fast?

Understanding the importance of
"SRE implements devops"

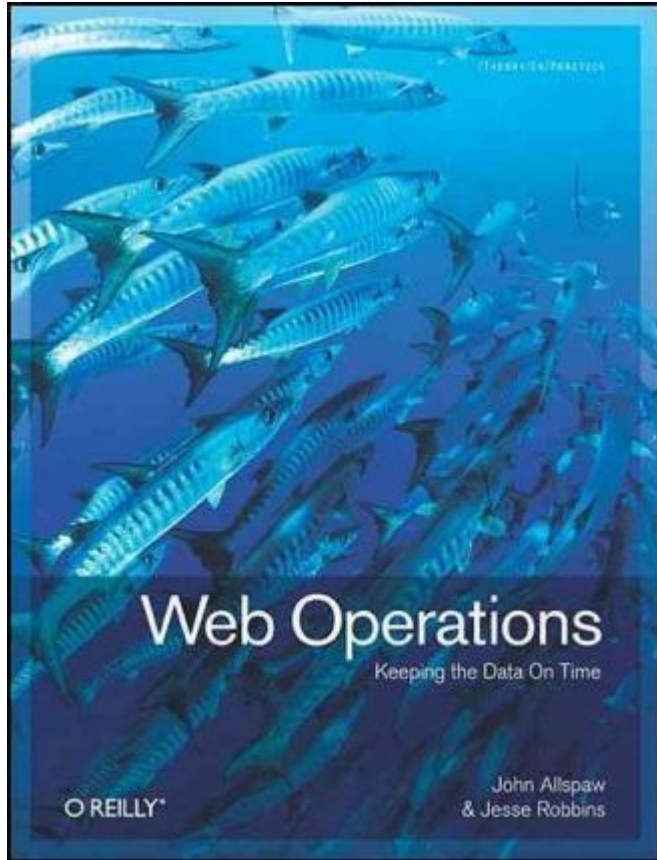
Maciej Lasyk,
geecon
Kraków 2019-05-17

Reliability Engineering?



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)



- This book is almost 10 years old
- CI, CD, monitoring, metrics, IAAC, how complex systems fail, dev & ops collaboration
- how our visitors feel and how to measure that?
- database reliability
- “How to Make Failure Beautiful: The Art and Science of Postmortems”
- cloud computing, developing career
- agile infrastructure

Reliability Engineering @Codewise?

Reliability Engineering @Codewise?



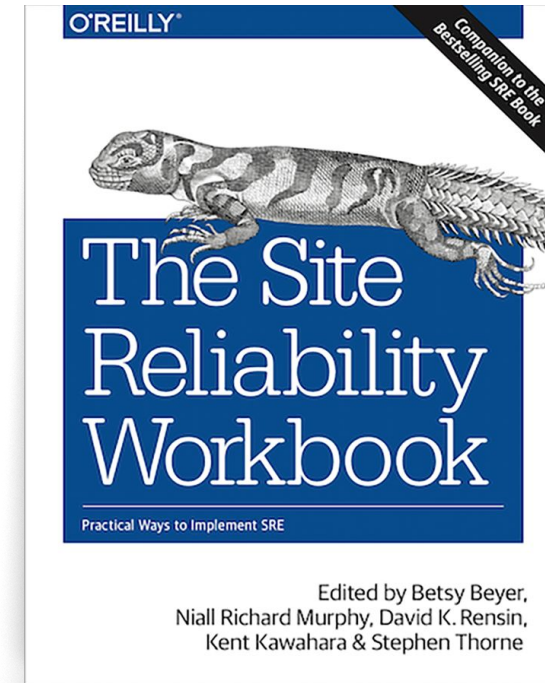
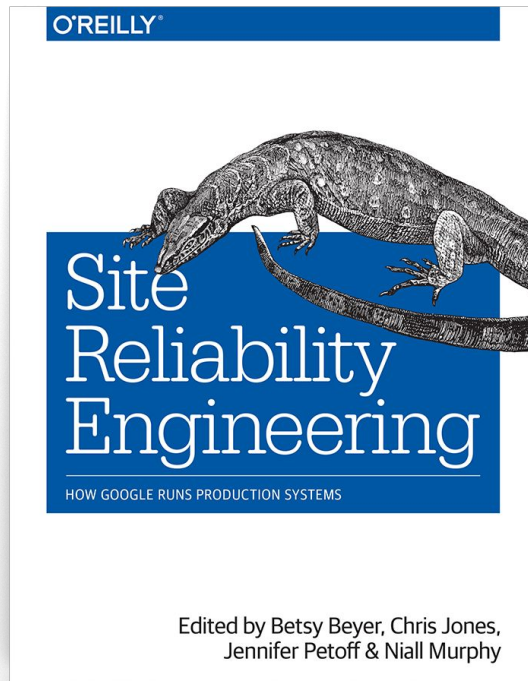
- **~1k EC2s**
- **300-400k req/s**
- **1PB of data out / monthly**

Reliability Engineering @Codewise?



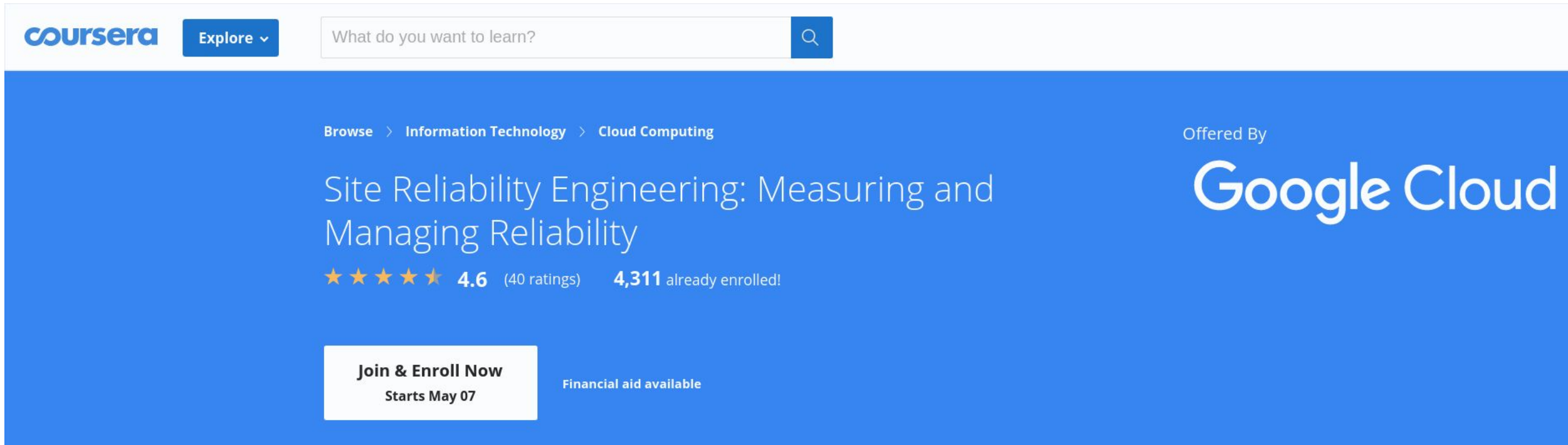


The Google approach



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

The Google approach



The screenshot shows the Coursera website interface. At the top, there is a navigation bar with the Coursera logo, an 'Explore' dropdown menu, and a search bar containing the text 'What do you want to learn?'. Below the navigation bar, the breadcrumb trail reads 'Browse > Information Technology > Cloud Computing'. The main heading for the course is 'Site Reliability Engineering: Measuring and Managing Reliability'. To the right of the heading, it says 'Offered By Google Cloud'. Below the heading, there are five yellow stars, a rating of '4.6 (40 ratings)', and the text '4,311 already enrolled!'. At the bottom left, there is a white button that says 'Join & Enroll Now' with 'Starts May 07' below it. To the right of the button, it says 'Financial aid available'.

Join & Enroll Now
Starts May 07

Financial aid available

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

What is SRE about?

Are our products working reliable so users are happy?

What is SRE about?

Are our products working reliable so users are happy?

How much reliability - related work is required?

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: 99.9999% (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: 99.9999% (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: 99.9999% (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: 99.9999% (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. Zeropark 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)**
- **$\text{SLO} = \text{successful events} / \text{valid events} [\%]$**

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 $\geq 1s$

Error Budget

- **Example:**
 - **PM together w/engineers defines an SLO -> an expectation of target uptime per month**
 - The actual uptime is measured by a neutral third party: Tornimo ;)
 - The difference is the “budget”: how much “unreliability” is remaining for the month
 - As long as there is error budget remaining—new releases can be pushed
 - Otherwise we must work on reliability

Error Budget

- **Example:**
 - **PM together w/engineers defines an SLO -> an expectation of target uptime per month**
 - **The actual uptime is measured by a neutral third party: Tornimo ;)**
 - The difference is the “budget”: how much “unreliability” is remaining for the month
 - As long as there is error budget remaining—new releases can be pushed
 - Otherwise we must work on reliability

Error Budget

- **Example:**
 - **PM together w/engineers defines an SLO -> an expectation of target uptime per month**
 - **The actual uptime is measured by a neutral third party: Tornimo ;)**
 - **The difference is the “budget”: how much “unreliability” is remaining for the month**
 - As long as there is error budget remaining—new releases can be pushed
 - Otherwise we must work on reliability

Error Budget

- **Example:**
 - **PM together w/engineers defines an SLO -> an expectation of target uptime per month**
 - **The actual uptime is measured by a neutral third party: Tornimo ;)**
 - **The difference is the “budget”: how much “unreliability” is remaining for the month**
 - **As long as there is error budget remaining—new releases can be pushed**
 - **Otherwise we must work on reliability**

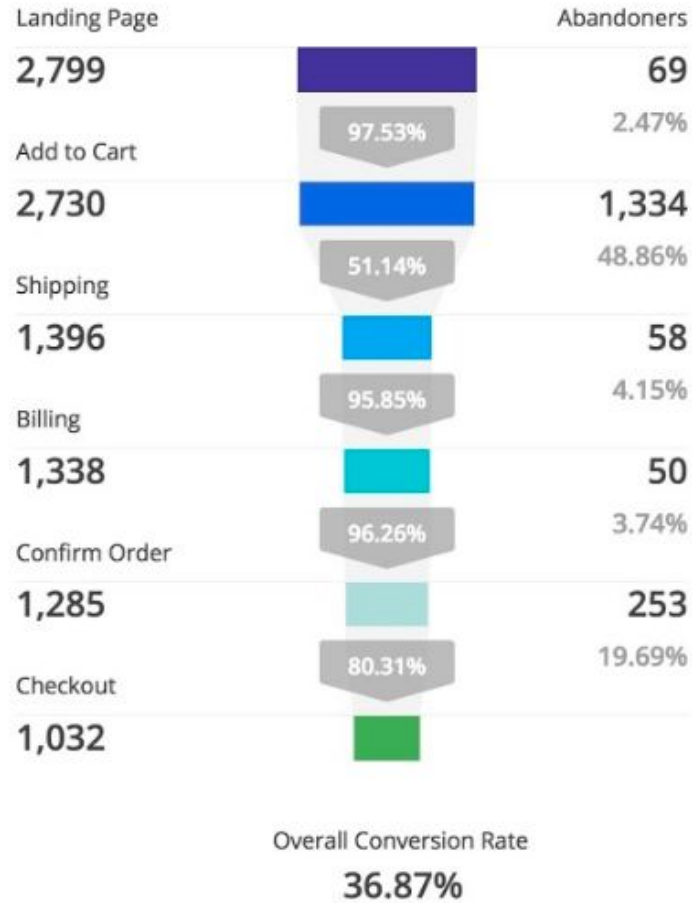
Error Budget

- **Example:**
 - **PM together w/engineers defines an SLO -> an expectation of target uptime per month**
 - **The actual uptime is measured by a neutral third party: Tornimo ;)**
 - **The difference is the “budget”: how much “unreliability” is remaining for the month**
 - **As long as there is error budget remaining—new releases can be pushed**
 - **Otherwise we must work on reliability**

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)	

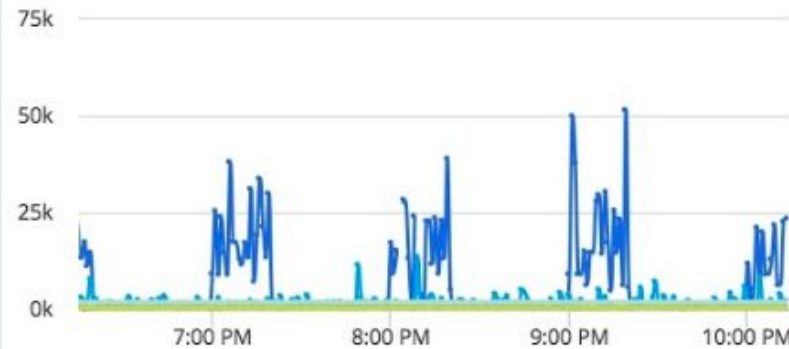
Visualizing reliability



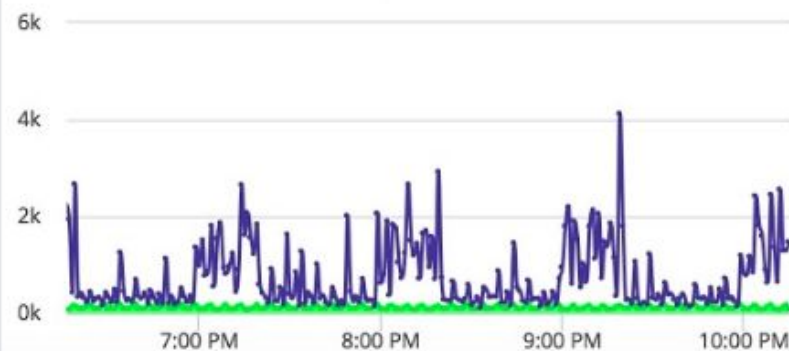
SLO



End User Response Time (ms)



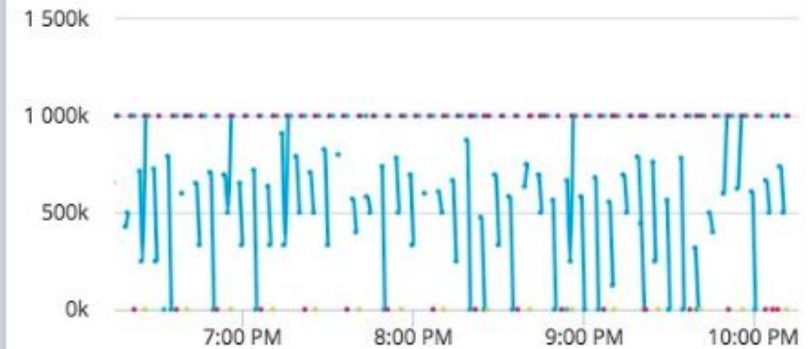
Business Transaction Response Time/Load



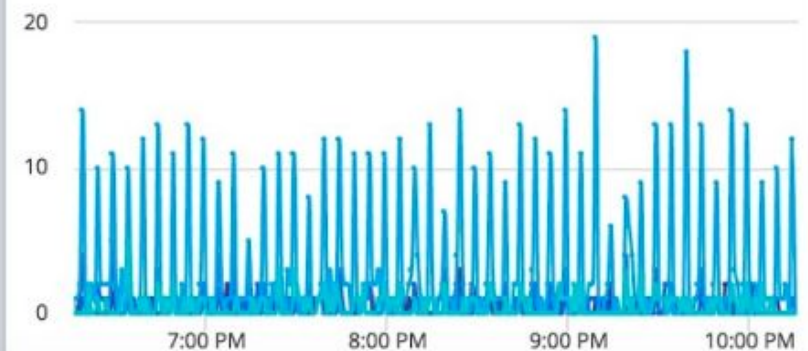
Error Budget



Availability - Synthetic Jobs



Business Transaction Error Load

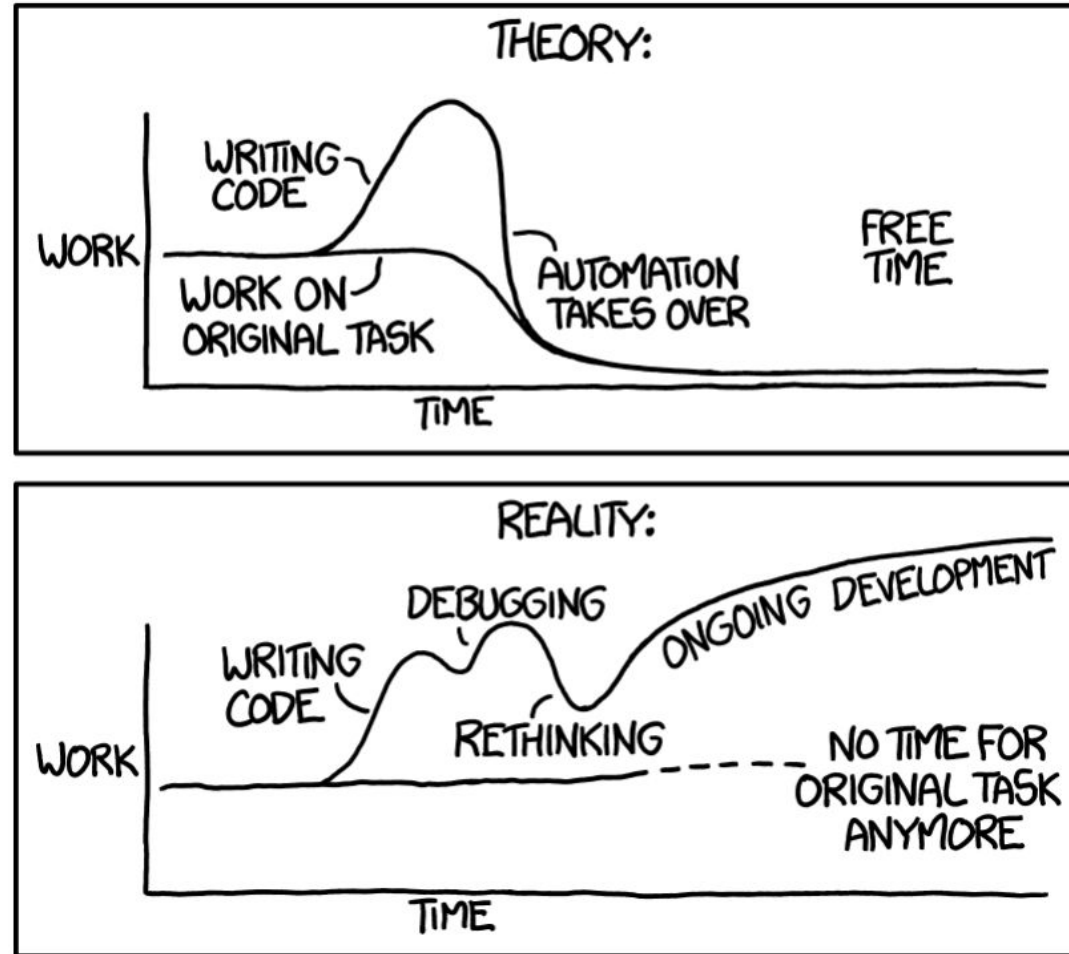


Toil

- 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!

Toil

"I SPEND A LOT OF TIME ON THIS TASK.
I SHOULD WRITE A PROGRAM AUTOMATING IT!"

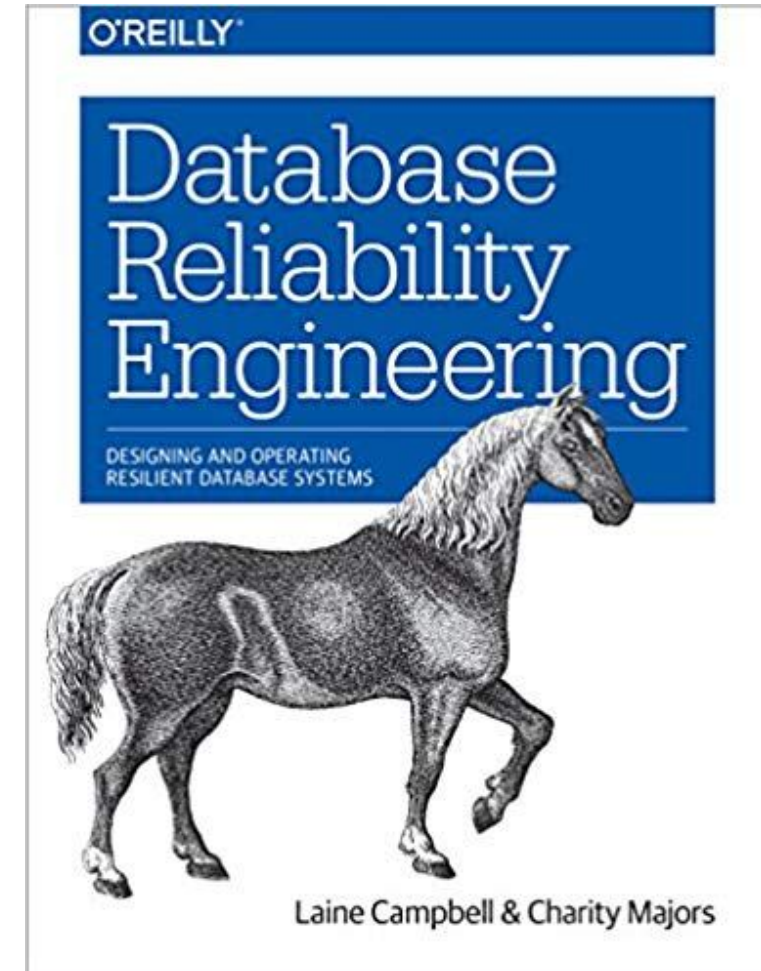


devops & SRE?

- **devops: practice of operations and development engineers participating together in the entire service lifecycle**
- **devops defines the overall behavior of the system, but the implementation details are left up to the author**
- **SRE: strict methods for working on service reliability and feature development across business and dev teams. SRE implement devops**

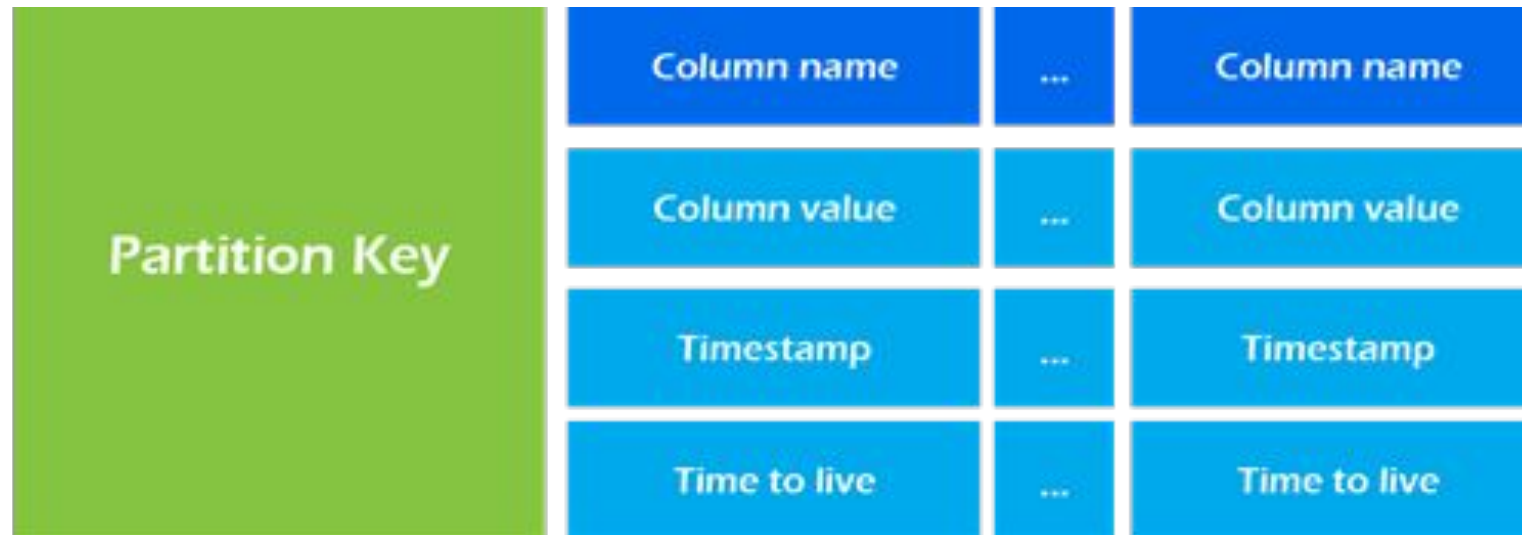
Database reliability

- how to learn it?
- Charity Major's: "put your developers oncall"
- <https://charity.wtf>



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://shermadigital.com/blog/designing-a-cassandra-data-model/>

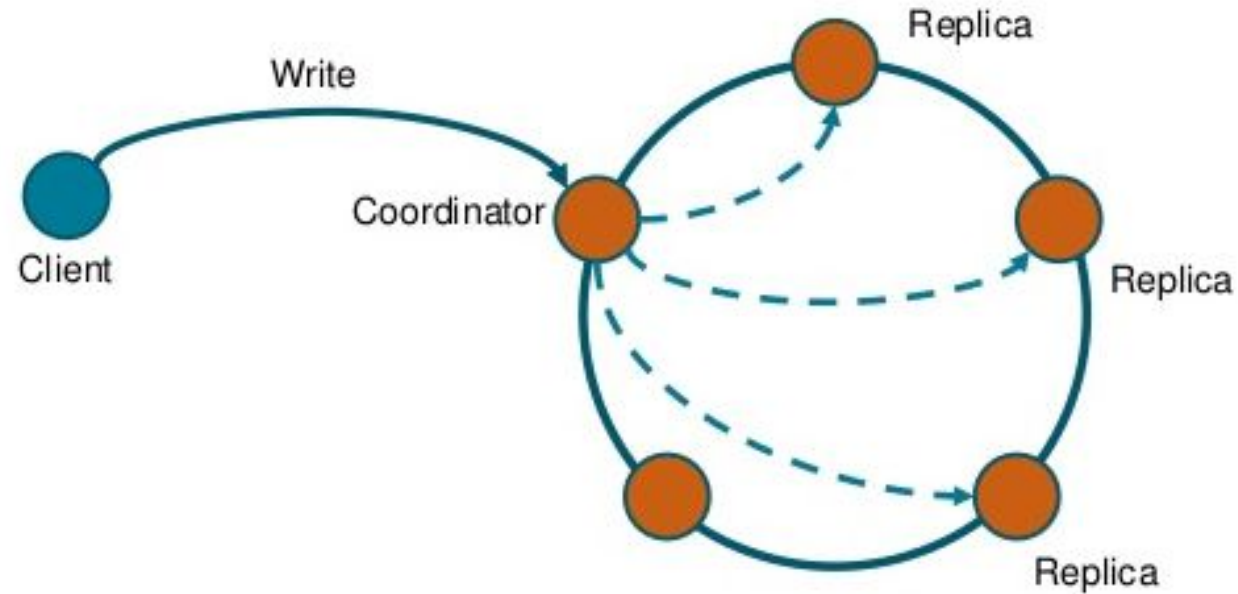
Cassandra basics - data modeling

2 main rules:

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

<https://shermigital.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 ($\text{nodes}/2+1$)**
- **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**

Setting up the reliability



- We're not Google (size, culture, etc)
- We won't have SRE teams
- We always customize

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: SRE follow - up: SLA/SLO/SLI
- SRE meeting #11: SRE follow - up: SLA/SLO/SLI (part 2)

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) - put here anything, we
infrastructure	Cassandra	3	TF & Ansible	click	Sentinel, Datadog, Tornimo	Pagerduty, Tornimo	no	we're working on self - healing (Datadog triggering Rundec
infrastructure	Cassandra-Reaper	2	TF & Ansible	click	Datadog, Tornimo	Pagerduty, Tornimo	no	we're working on IT; ASG + S3
infrastructure	Rundeck	3	TF & Ansible	click	Datadog, Tornimo	Pagerduty, Tornimo	no	we're working on IT; ASG + S3 & RDS
infrastructure	etcd	3	Ansible	click	Sentinel, Datadog	Pagerduty	no	etcd is going down in the nearest future
infrastructure	etcd-browser	2	-	-			no	to be decomissioned after DSP migration to config-server

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)

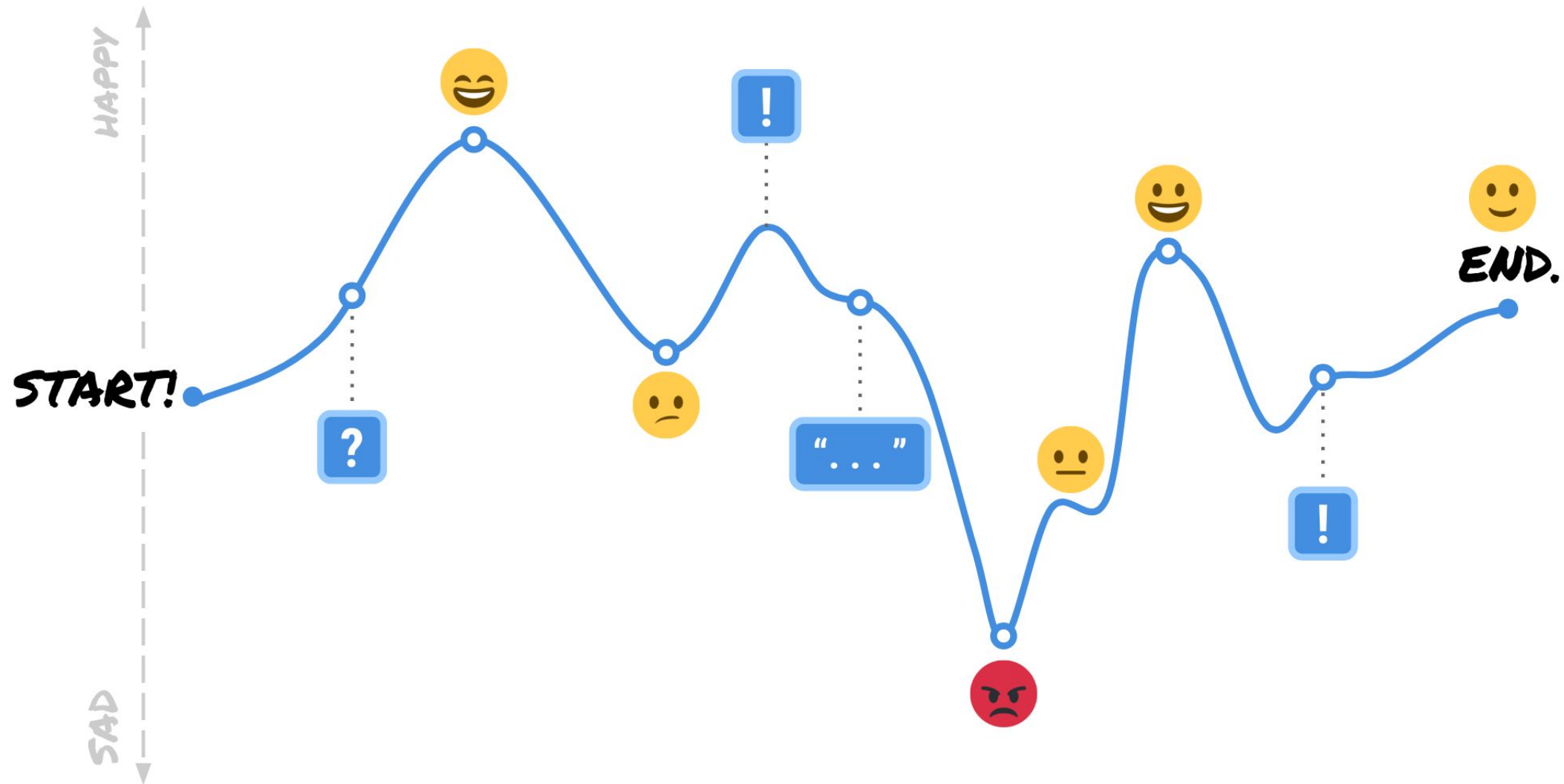
A couple of months ago we created SRE chapter here @Codewise. However, this is an engineering group and **we need our business to take part in this**. Customer Reliability Engineering happens between technology, product and the client. **Thus this invitation for you guys <3**

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/>

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**

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

Observability & monitoring



Sentinel
...and its future



Observability & monitoring

- No HA
- development
- Check isolation
- Pingdom integration
- Not possibility for muting, scheduling downtime
- Some well known bugs
- No alert history
- Actionable alerts

Observability & monitoring



—

Application
monitoring for
developers & teams
that move fast

The operational flow

- Using Ansible Playbooks
 - via Vagrant box (suggested way)
 - directly from own laptop (more complicated)
 - Running a playbook
- Cassandra management
 - Accessing and managing container logs
 - Replacing live node without bootstrapping (only on TEST for now)
 - Dead node replacement
 - Cluster restore
 - Resizing Cassandra data filesystem
- Cassandra-Reaper
 - Provisioning Cassandra Reaper
 - Reaper logs
- Etcd
 - Adding new node to cluster
 - Removing node from cluster
 - Upgrading to new etcd version

The operational flow

- get all procedures in one place
- put into repo
- use one scheduler (Rundeck, cloud-agnostic)
- don't use Jenkins (audit, logs, credentials sharing etc)

About automation

use proper tools! (Terraform, Ansible, not Python/Java/Go)

don't snowflake automation!

declarativeness over imperativeness!

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

Prepare for problems during upgrade



Doing a deploy without rollback ability

<https://devopsreactions.tumblr.com/post/93861075847/doing-a-deploy-without-rollback-ability>

Self - healing creation 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**

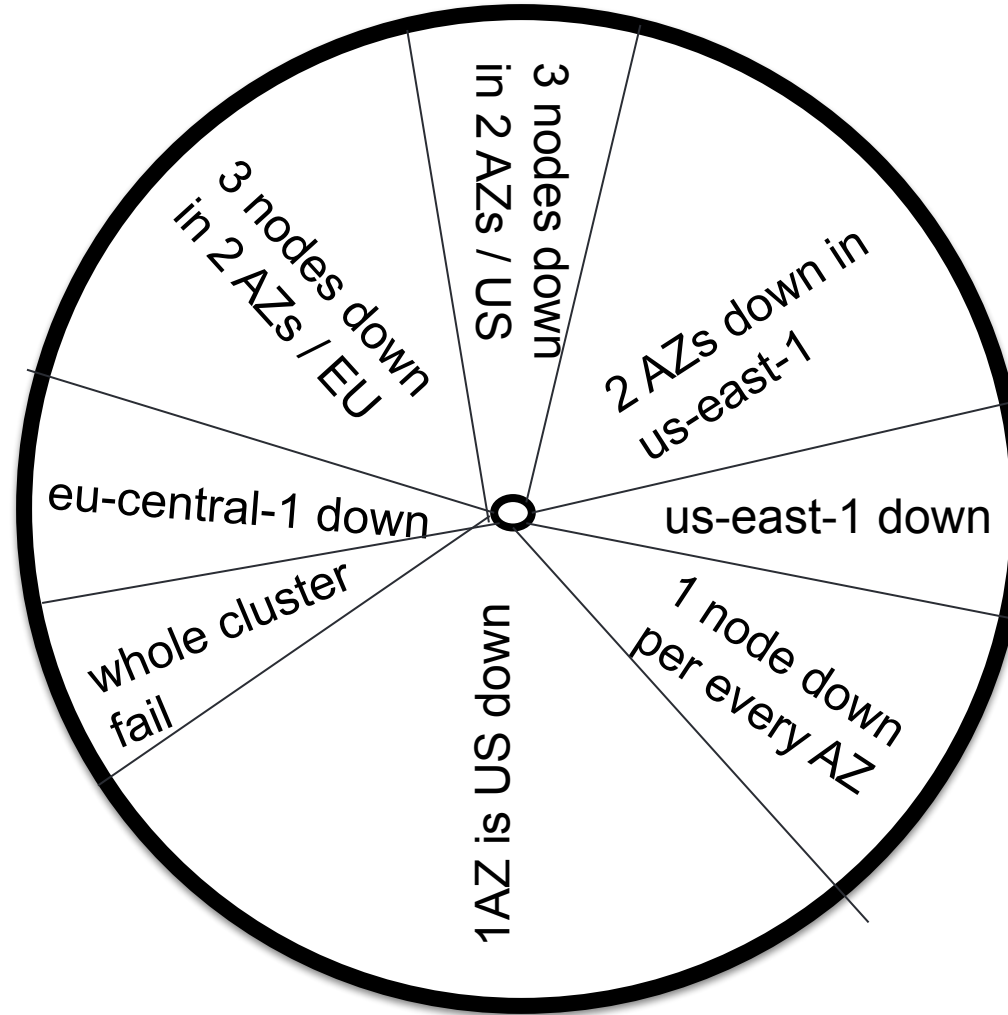
Preventing recurring outages

1. Not always full incident analysis and postmortem
2. simple procedure:
 - a. handle outage
 - b. analyse impact, source of the problem and whether in the future it may happen again (even 1:1 between 2 ppl)
 - c. take notes, create tasks
 - d. reset

Drills, workshops



Wheel of misfortune



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`

C* operational tasks and automation

▼ backups

- ▶ Copy S3 snapshots to Glacier - weekly ▼ This job co
- ▶ Create data backups (nodetool snapshots) ▼ This jo
- ▶ Create schemas backup ▼ Creates backup of keyspa
- ▶ Send keyspaces heartbeats ▼ This job connects to C
- ▶ daily-prod-backup-restoration-CopyWithTokens ▼
- ▶ daily-test-backup-restoration-CopyWithTokens ▼ 1
- ▶ weekly-prod-backup-restoration-SSTableLoader ▼
- ▶ weekly-test-backup-restoration-SSTableLoader ▼ T

▼ maintenance

- ▶ cassandra-status ▼ Run nodetool
- ▶ nodetool cleanup ▼ Run nodetool
- ▶ nodetool clearsnapshots ▼ Run
- ▶ nodetool compact keyspace ▼ L
- ▶ nodetool decommission ▼ Run r
- ▶ nodetool removenode ▼ Run no

▼ provisioners

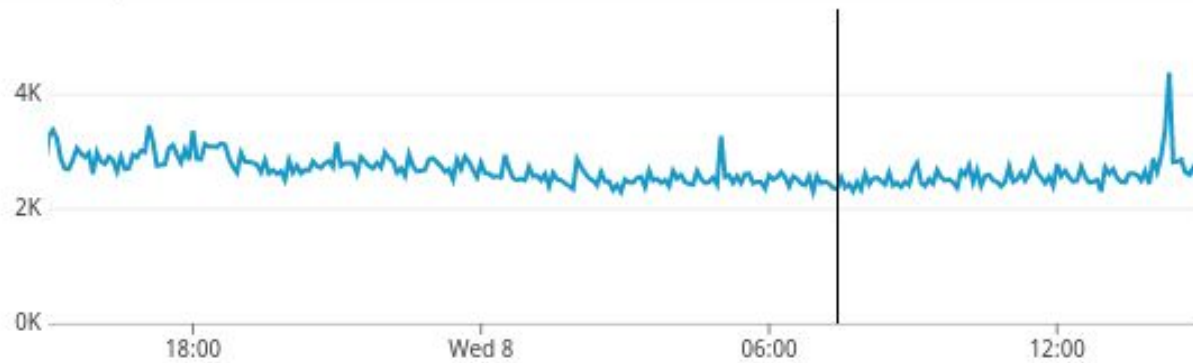
- ▶ manage-custom-cluster-vms ▼
- ▶ obliterate-custom-cluster ▼ Use
- ▶ provision-custom-cluster ▼ Use
- ▶ provision-single-instance ▼ Use

So what exactly means "reliable enough" now?

1. on-calls doesn't have any C* - related calls
2. business is not impacted by C* performance
3. C* cluster tries to self - heal in case of incident
4. Latencies are stable (we're working on C* SLOs)
5. Latencies are not too low (we are at 50% of possible VM size)
6. neither too high (developers are happy, customers are happy)

So what exactly means "reliable enough" now?

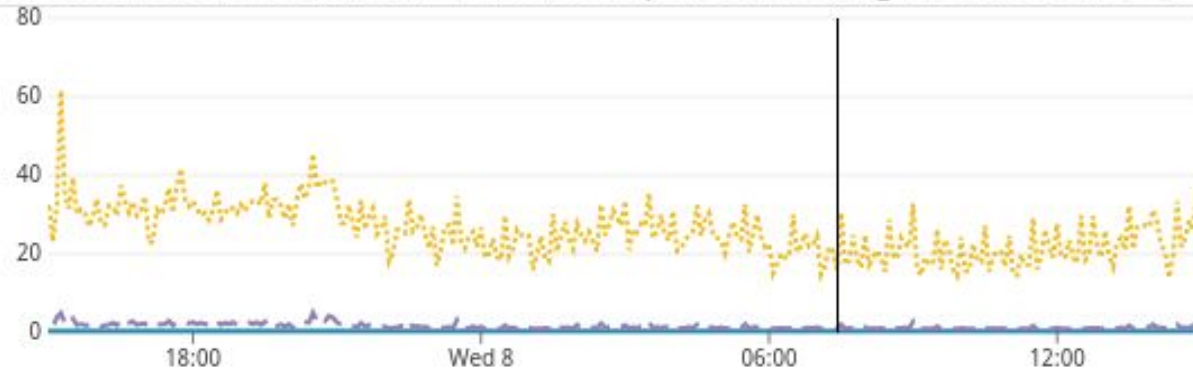
Read request rate sum



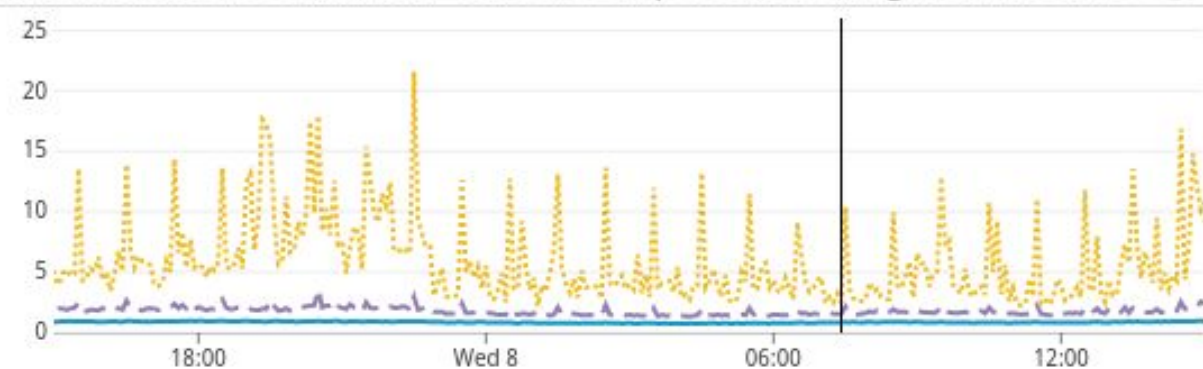
Write request rate sum



Read latencies (coordinator) - 75, 95, 99th percentiles (avg from hosts in



Write latencies (coordinator) - 75, 95, 99th percentiles (avg from hosts in



Do you still wanna host own:

- Cassandra
- etcd
- Kafka
- Zookeeper
- MongoDB
- Redis
- ...any other?

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://shermigital.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/