high availability and reliability for something simple called OpenStack

i don’t see why OpenStack high availability and reliability should be hard

what does OpenStack offer?

flexible compute, storage, networking

what does OpenStack offer?

instantly available, scalable computing resources

what does OpenStack offer?

on-demand, financially-efficient use of hardware and software

right-matching real resource requirement to availability, enable scalable computing service

new expectations

on demand

it should just work

new roles

operator — vs — consumer

operator

operation of underlying infrastructure

provides virtualized resources (compute, networking, storage) … and stops at this!

consumer

operation within the virtualized environment

not concerned about underlying infrastructure

consumer

don’t care about what it takes to provide the cloud environment

it should just work

operator

responsibility to keep cloud environment running

dead

in event the service is dead: expectation no longer met

dead is not an option

if it is dead, it is too late!

when could there be risk of death?

lifecycle

deploy

operate

upgrade

deploy

can be a challenge

mess it up and it will come back to bite you later

operate

keeping services running

high-availability is not a luxury

risk of death

services encountering problems + impacting consumers

upgrade

options:

- 2n upgrades over n years
- 2n upgrades at once n years later

risk of death

downtimes and interruption during upgrade

worse, any workload killed!

risk of death

essentially in and throughout phases:

- operate
- upgrade

death during upgrade phase

simple: zero-downtime upgrade needed

solution architecture (deploy phase) has to be able to meet requirement

death during operate phase

intrinsic — vs — extrinsic

death risks

extrinsic death risk?

dealt with via highly available solution architecture

what is OpenStack?

a series of intercommunicating services

HTTP, MQ

types of services

1. with data + configuration
2. configuration only

safeguards: configuration

infrastructure-as-code operating model

must be able to re-deploy components
by re-running deployer

safeguards: data

replication, replication, replication

2-node model or quorum/odd-node model

safeguards: data

traditional high availability operations model

anything else?

what else?

http: it’s a web server! let’s treat it as such!

high availability for web servers

what else?

mq: put stuff in queue, take stuff from queue

replicate reader/writer

extrinsic death risk

keep extra copy available at all times

is that all?

yes

smarts?

it’s in the solution architecture

get it right, else it can come back to bite you!

intrinsic death risk

it’s intrinsic, not externally triggered

intrinsic death risk

traditional operation model: monitor, alert, action

monitor

look for dead or alive

if it looks like it is dead, check again x 3, to be sure

monitor

… but it’s too late if it is dead!

intrinsic death risk?

it’s intrinsic!

highly available solution architecture will not suffice

smarts?

need to detect sick states

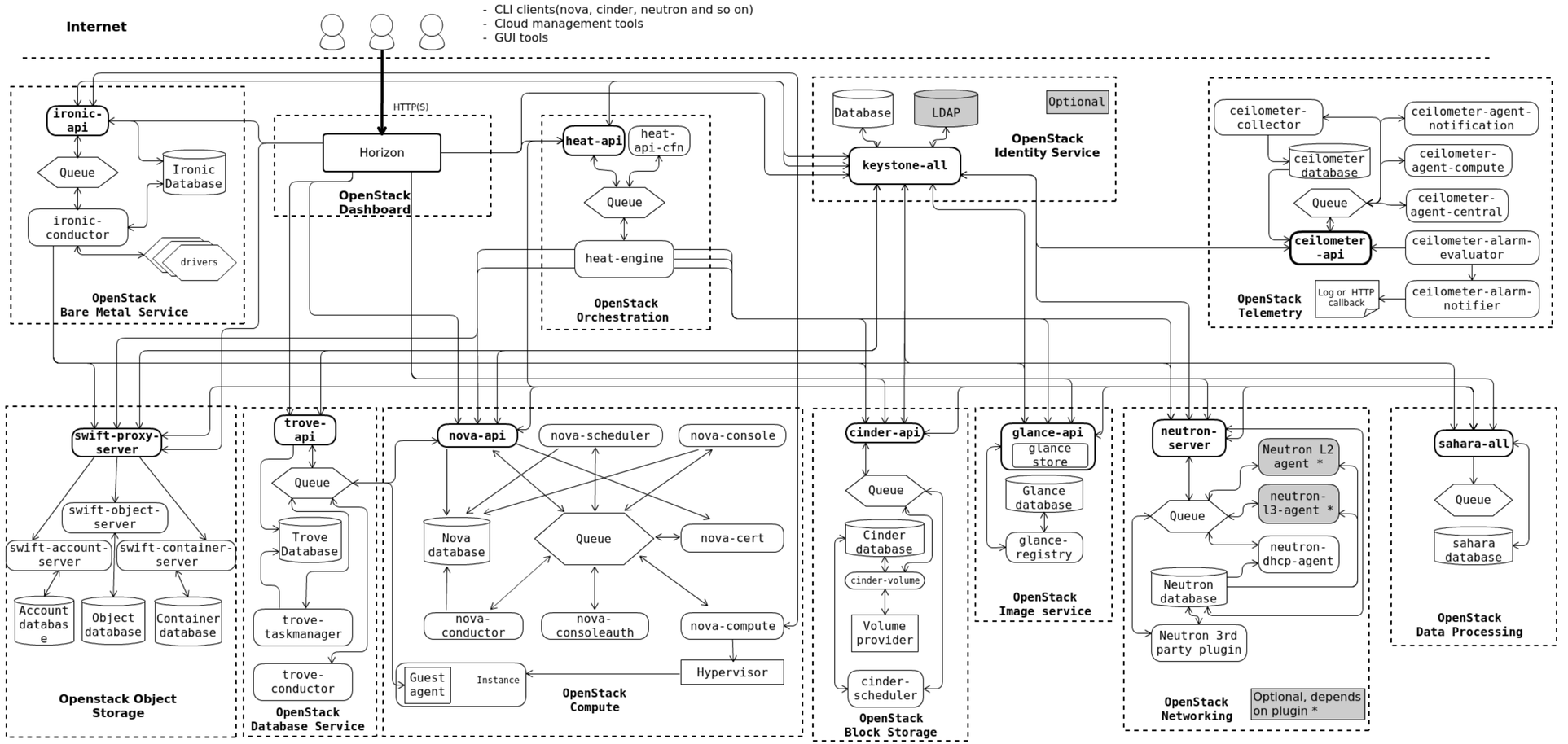
sick: alive, not healthy

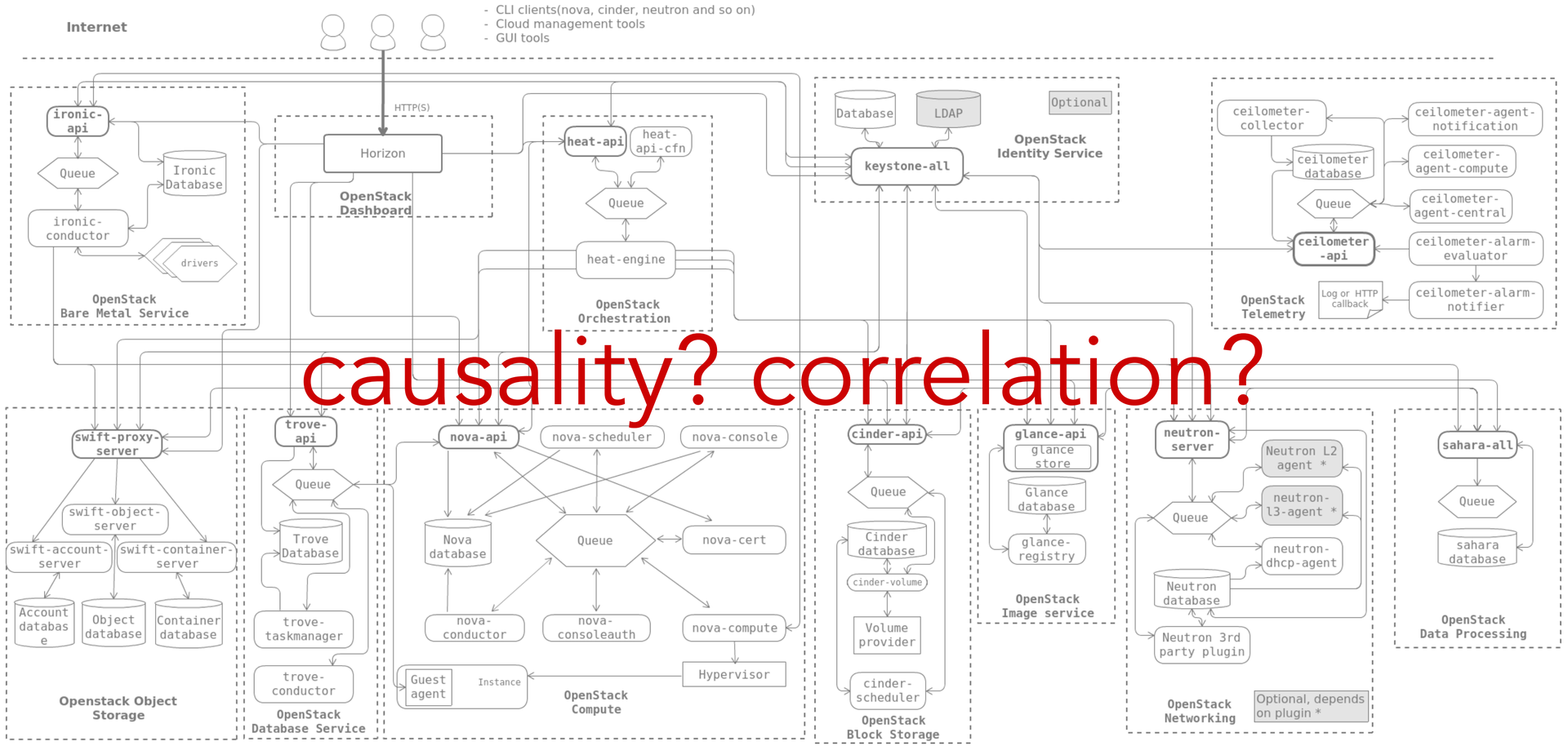
smarts

correlation between x and y

not just causality (that's easy!)

OpenStack architecture





challenge

non-trivial computational task

plus, “sick” in one environment
may not be “sick” in another

data

2 types of data

- metrics
- logs

it needs to …

example: response time behavior of Web service + disk SMART errors

detect sick states that are otherwise below the radar

how?

logs ingest from OpenStack infrastructure

time series store for machine generated data

watch log data for anomalies

analog

take continuous x-ray pictures of patient

feed through decision engine

if detect certain dark spots, the patient may be “sick”

analog

if decision engine got it wrong, need to re-train

next time, dark spots will be classed as “sick”

health

picture covers a certain configurable/variable time window

constituents of the picture: configurable

operator to trigger re-training on wrong decisions

health

shortcuts

- pre-condition: has to be alive to check for “sick”
- causality relationships

not a toy

highly available

no-polling, operate on pushed data streams

not a toy

back pressure sensitive: not to overload data pipeline

horizontal scalable data store

load balancing across multiple backends

not a toy

high availability for an evaluation system?

how much data?

4 bytes/metric

24 cores/node

4 VMs/core

200+ metrics/VM

360 Gbps, 44 GB/s

how much data?

256 bytes/entry

1 entry/s/service

80+ services

20 MB/s

72 GB/hr

smarts?

smart data ingest reduction

imbalanced I/O pattern: large volume of small writes, small number of large reads

deal with risk of death and …

know imminent fault

deal with latent threat, before threat becomes patent

meet consumer expectations

avert death

other lifecycle challenges

ease and flexibility of deployment, idempotent

infrastructure-as-code

resource management

zero-downtime upgrade

deployment

zero-to-showtime: 1-cup of coffee

highly available: default

idempotent: flexible to change at any time

resource management

operating system-based model, throughout entire facility

dynamic, efficient: lower OpEx + CapEx

… topics for another day

this is Sardina Systems

full-lifecycle OpenStack automation: deploy, operate, upgrade

AI-driven smart, efficient, super-scalable automation technology

SARDINA

enable organizations to rapidly experience the value of OpenStack cloud and maximize utility of their resources

sectors: finance, government, aerospace, research, academia

in 2015, Sardina FishOS won the IDC HPC Innovation Award

SARDINA

top-10 trader on New York market

U of Edinburgh: top-5 UK academic and research site

150k VMs at classified government site

how to achieve OpenStack high availability and reliability

dr kenneth tan
kenneth.tan@sardinasystems.com
+44 798 941 7838