Debugging under fire

Keeping your head when systems have lost their mind

Bryan Cantrill
CTO

bryan@joyent.com

@bcantrill

The genesis of an outage

Joyent

[20:05:33] <Operator-1> I am gonna reboot all empty RB's shortly here, just getting the list right now
[20:07:57] <Operator-2> you let me know when it ti s a go and I will do it towards the end of the shift so you have a few hours
[20:14:08] <Operator-3> uh
[20:14:13] <Operator-3> us-east-1 is being rebooted
[20:14:17] <Operator-3> ?
[20:14:30] <Operator-4> ??
[20:14:39] <Operator-3> Broadcast Message from root (???) on headnode Tue May 27 20:13:53...
THE SYSTEM headnode IS BEING SHUT DOWN NOW ! ! !
Log off now or risk your files being damaged
[20:14:54] <Engineer-1> wtf
[20:14:54] <bot> the case that blows this fucker up is pushing large files.
[20:15:11] <Operator-5> what? :o
[20:15:15] <Engineer-1> please don't be me, please don't be me, please don't be me

“Please don’t be me, please don’t be me”

Joyent

[20:16:09] <Operator-1> it is me
[20:16:31] <Engineer-1> deliberate or not?
[20:16:39] <Operator-1> accident
[20:16:49] <Operator-1> I ewas rebooting an rb
[20:16:52] <Engineer-1> ok, whew
[20:16:53] <Operator-1> forgot to put -n
[20:17:02] <Operator-1> I fucked up
[20:17:03] <Operator-1> I am sorry!
[20:17:10] <Operator-1> I truly truly sorry
[20:17:12] <Operator-5> Operator-1: so you got just the headnode, or you got all of east1?
[20:17:39] <Operator-1> so I did this:
[20:17:39] <Operator-1> sdc-oneachnode *redacted*
[20:18:32] <Support-Personnel-1> so is all of East going to reboot?
[20:18:32] <Engineer-2> you just rebooted all of us-east-1.

“…doesn’t begin to describe it”

Joyent

[20:18:32] <Engineer-2> you just rebooted all of us-east-1.
[20:18:43] <Support-Personnel-2> whee
[20:18:44] <Engineer-1> yes
[20:18:48] <Support-Personnel-1> need to know what the impact is here… ok
[20:18:49] <Operator-5> WHEE!
[20:18:59] <Operator-6> fuck
[20:19:10] <Engineer-2> Support-Personnel-1: every customer intsance is about to go down.
[20:19:10] <Bryan> "Fuck" doesn't begin to describe it.

“WHEE!”

Joyent

The Register®
Biting the hand that feeds IT

DATA CENTER SOFTWARE SECURITY TRANSFORMATION DEVOPS BUSINESS PERSONAL TECH SCIENCE EMERGENT TECH BOOTNOTES

Data Center ▸ Cloud

95

Fat-fingered admin downs entire Joyent data center

Cloud operator now home to most mortified sysadmin in the USA

“Fat-finger”?

Joyent

- Not just a “fat-finger”; even this relatively simple failure reflected deeper complexities:

[20:19:50] <Operator-5> this is gonna need a postmortem - i've almost done what Operator-1 just did a *number* of times.
[20:20:42] <Operator-5> Operator-1: don't stress, we work on recovery now.
[20:21:47] <Operator-5> interestingly i don't see any alarms for other hosts in east1 yet.
[20:21:51] <Operator-5> and by now i should be seeing some.
[20:21:53] <Operator-1> neither do I
[20:22:14] <Bryan> Your alarms probably depend on some nodes in us-east-1 being up.
[20:22:14] <Support-Personnel-1> yeah ^
[20:22:23] <Bryan> Tweets are coming in.

- Outage was instructive — and lucky — on many levels…

It could have been much worse!

Joyent

- The (open source!) software stack that we have developed to run our public cloud, Triton, is a complicated distributed system
- Compute nodes are PXE booted from the headnode with a RAM-resident platform image
- It seemed entire conceivable that the services needed to boot compute nodes would not be able to start because a compute node could not boot…
- This was a condition we had tested, but at nowhere near the scale — this was a failure that we hadn't anticipated!

How did we get here?

Joyent

- Software is increasingly delivered as part of a *service*
- Software configuration, deployment and management is increasingly *automated*
- But automation is not total: humans are still in the loop, even if only developing software
- Semi-automated systems are fraught with peril: the arrogance and power of automation — but with human fallibility

Human fallibility in semi-automated systems

Joyent

Human fallibility in semi-automated systems

Joyent

Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region

We'd like to give you some additional information about the service disruption that occurred in the Northern Virginia (US-EAST-1) Region on the morning of February 28th. The Amazon Simple Storage Service (S3) team was debugging an issue causing the S3 billing system to progress more slowly than expected. At 9:37AM PST, an authorized S3 team member using an established playbook executed a command which was intended to remove a small number of servers for one of the S3 subsystems that is used by the S3 billing process. Unfortunately, one of the inputs to the command was entered incorrectly and a larger set of servers was removed than intended. The servers that were inadvertently removed supported two other S3 subsystems. One of these subsystems, the index subsystem, manages the metadata and location information of all S3 objects in the region. This subsystem is necessary to serve all GET, LIST, PUT, and DELETE requests. The second subsystem, the placement subsystem, manages allocation of new storage and requires the index subsystem to be functioning properly to correctly operate. The placement subsystem is used during PUT requests to allocate storage for new objects. Removing a significant portion of the capacity caused each of these systems to require a full restart. While these subsystems were being restarted, S3 was unable to service requests. Other AWS services in the US-EAST-1 Region that rely on S3 for storage, including the S3 console, Amazon Elastic Compute Cloud (EC2) new instance launches, Amazon Elastic Block Store (EBS) volumes (when data was needed from a S3 snapshot), and AWS Lambda were also impacted while the S3 APIs were unavailable.

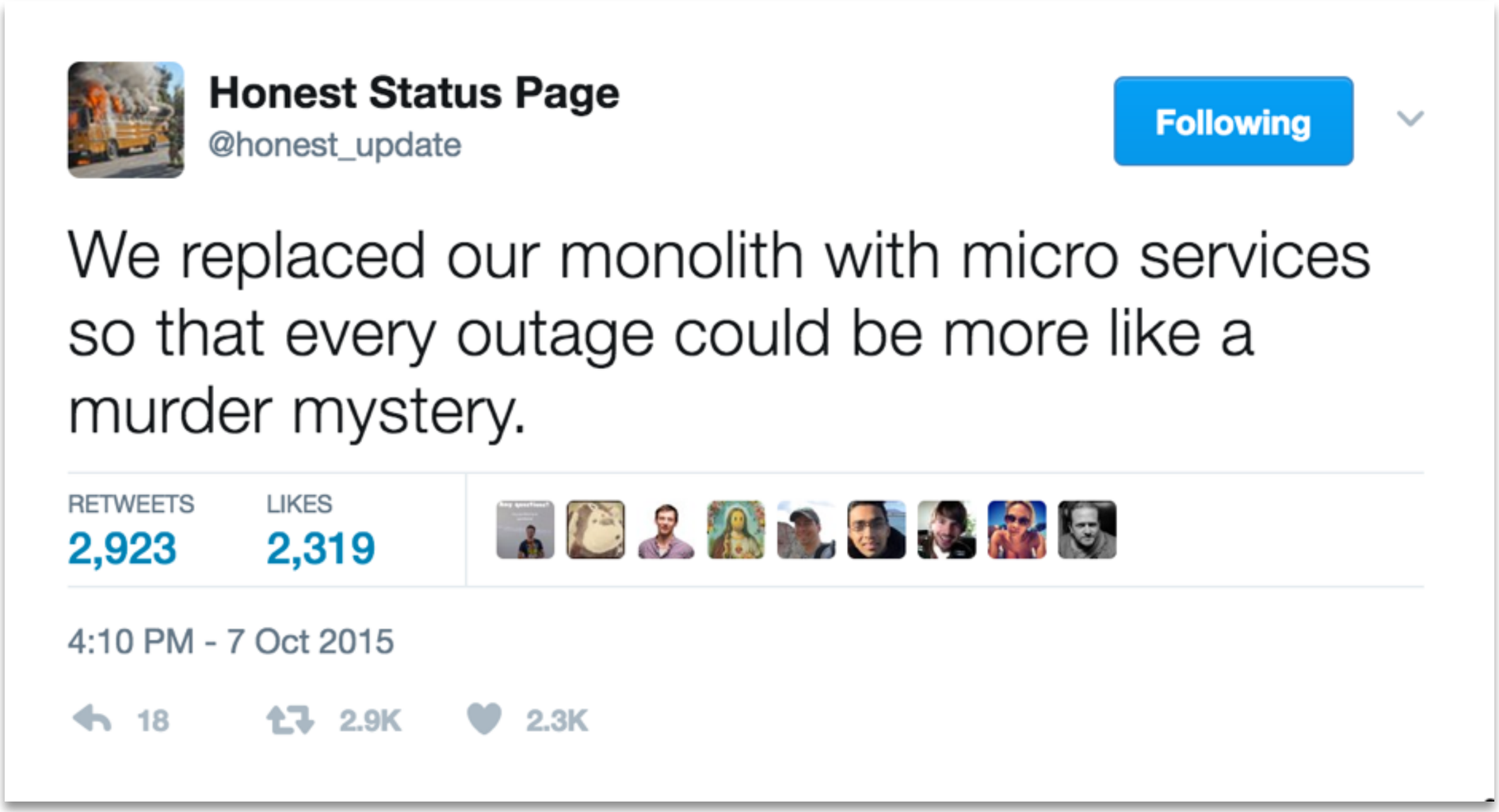
Whither microservices?

Joyent

- Microservices have yielded simpler components — but more complicated systems
- …and open source has allowed us to deploy many more kinds of software components, increasing complexity again
- As abstractions become more robust, failures become rare, but arguably more acute: service outage is more likely due to *cascading failure* in which there is not one bug but several
- That these failures may be in discrete software services makes understanding the system very difficult…

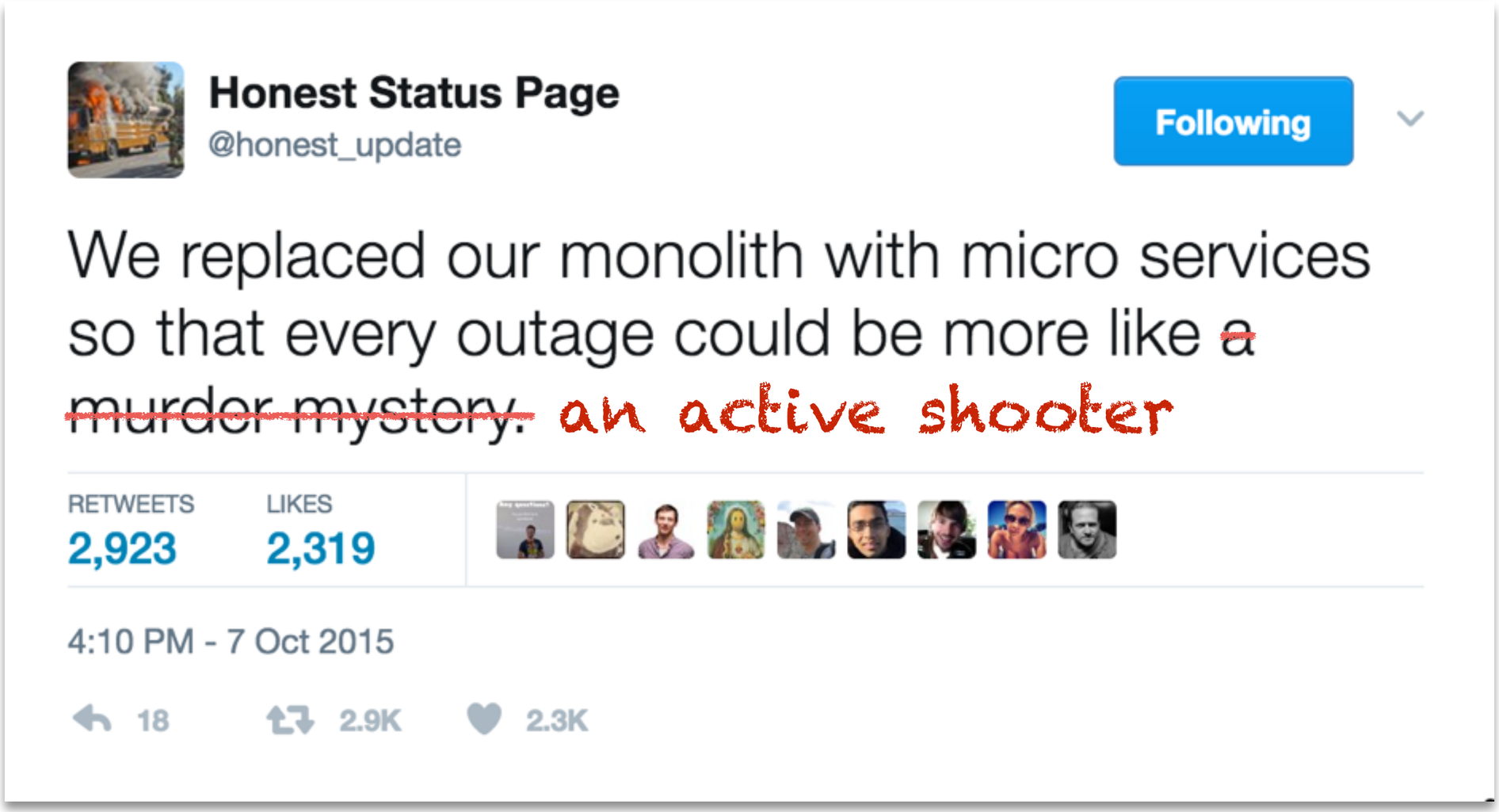
The Microservices Complexity Paradox

Joyent



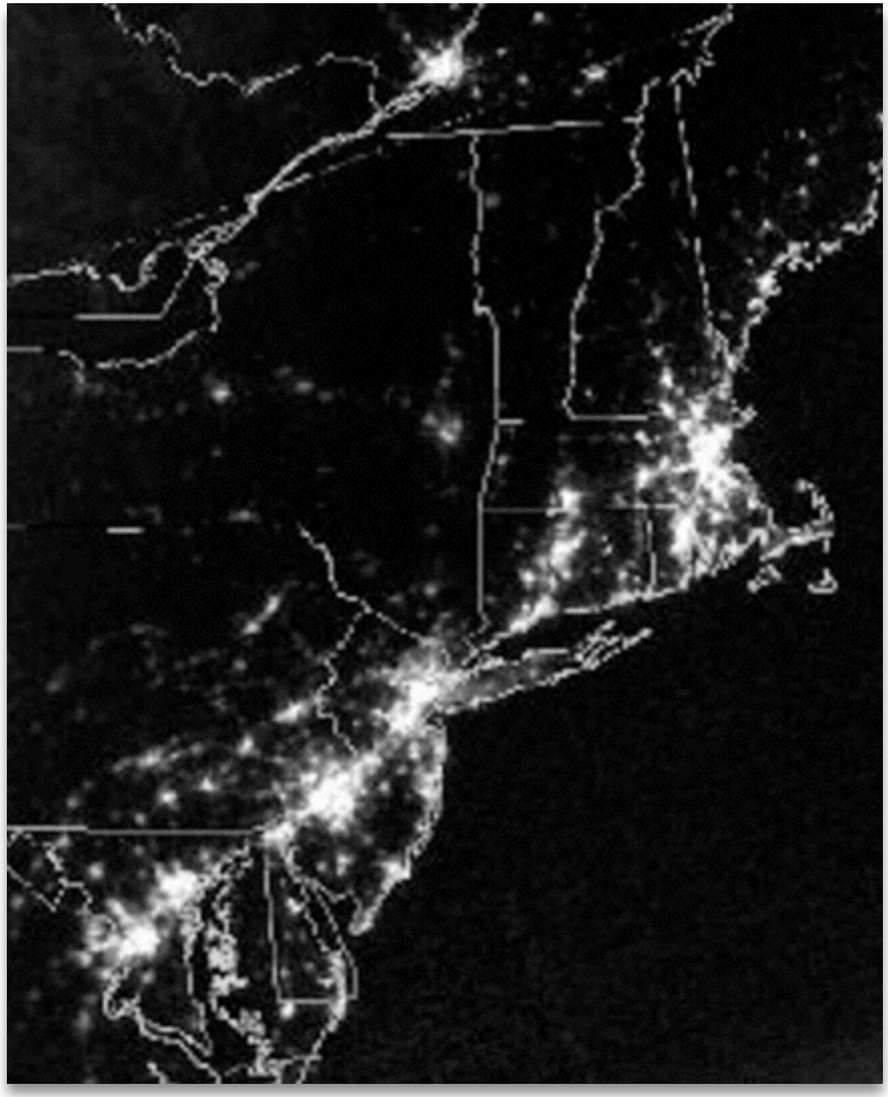
The Microservices Complexity Paradox

Joyent



Modern software failure modes

Joyent

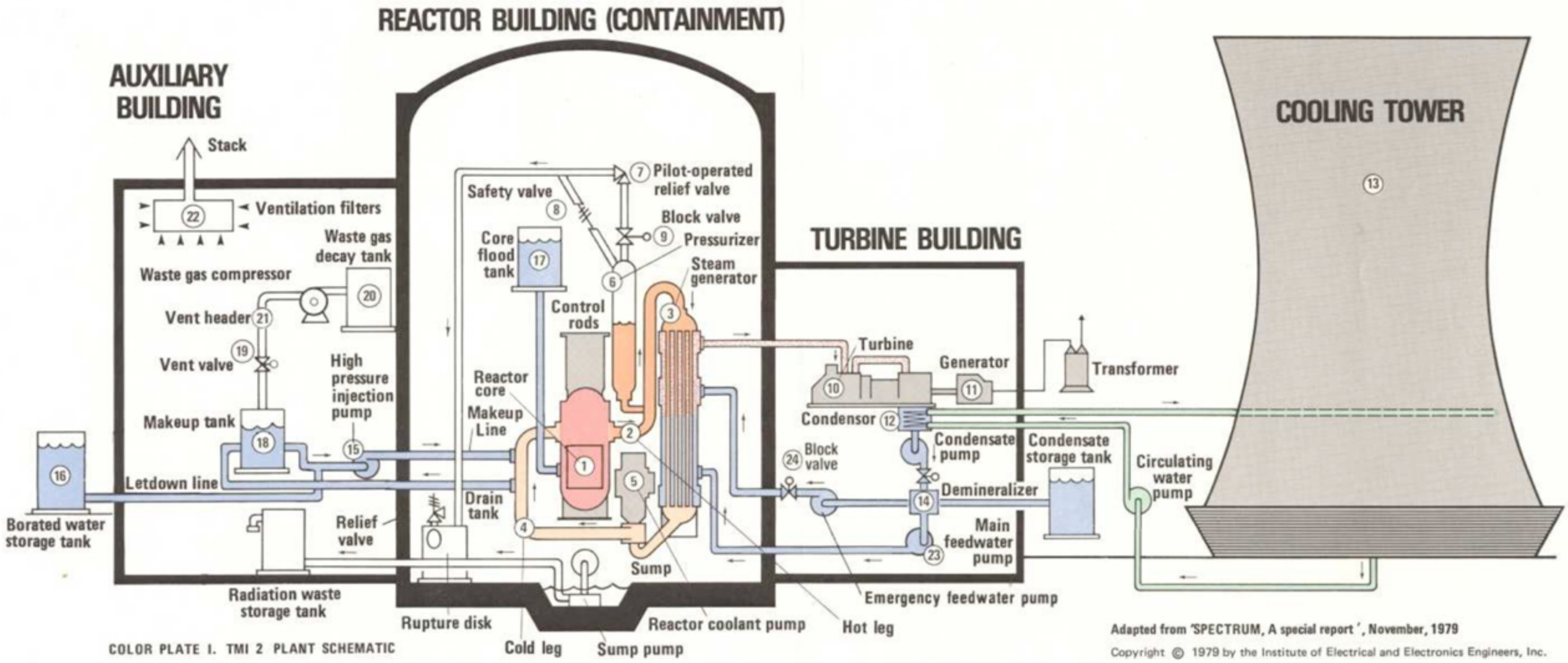


An even more apt metaphor

Joyent

A mechanical distributed system

Joyent



COLOR PLATE I. TMI 2 PLANT SCHEMATIC

Adapted from 'SPECTRUM, A special report', November, 1979

But… but… alerts and monitoring!

Joyent

> “It is a difficult thing to look at a winking light on a board, or hear a peeping alarm — let alone several of them — and immediately draw any sort of rational picture of something happening”
>
> — *Nuclear Regulatory Commission’s Special Report on incident at Three Mile Island*

The debugging imperative

Joyent

- We suffer from many of the same problems as nuclear power in the 1970s: we are delivering systems that we think can't fail
- In particular, distributed systems are vulnerable to software defects — we must be able to debug them *in production*
- What does it mean to develop software to be debugged?
- Prompts a deeper question: how *do* we debug, anyway?

Debugging in the abstract

Joyent

- Debugging is the process by which we understand pathological behavior in a software system
- It is not unlike the process by which we understand the behavior of a natural system — a process we call *science*
- Reasoning about the natural world can be very difficult: experiments are expensive and even observations can be very difficult
- Physical science is *hypothesis-centric*

The exceptionalism of software

Joyent

- Software is entirely synthetic — it is *mathematical* machine!
- The conclusions of software debugging are often mathematical in their unequivocal power!
- Software is so distilled and pure — experiments are so cheap and observation so limitless — that we can structure our reasoning about it differently
- We can understand software by simply *observing* it

The art of debugging

Joyent

- The art of debugging isn’t to guess the answer — it is to be able to ask the right questions to know how to answer them
- Answered questions are facts, not hypotheses
- Facts form constraints on future questions and hypotheses
- As facts beget questions which beget observations and more facts, hypotheses become more tightly constrained — like a cordon being cinched around the truth

The craft of debuggable software

Joyent

- The essence of debugging is asking and answering questions — and the craft of writing debuggable software is allowing the software to be able to answer questions about itself
- This takes many forms:
  - Designing for postmortem debuggability
  - Designing for *in situ* instrumentation
  - Designing for *post hoc* debugging

A culture of debugging

Joyent

- Debugging must be viewed as the process by which systems are *understood* and *improved*, not merely as the process by which bugs are made to go away!
- Too often, we have found that beneath innocent wisps of smoke lurk raging coal infernos
- Engineers must be empowered to understand anomalies!
- Engineers must be empowered to take the extra time to build for debuggability — we must be secure in the knowledge that this pays later dividends!

Debugging during an outage

Joyent

- When systems are down, there is a natural tension: do we optimize for recovery or understanding?
  - “Can we resume service without losing information?”
  - “What degree of service can we resume with minimal loss of information?”
- Overemphasizing recovery with respect to understanding may leave the problem undebugged or (worse) exacerbate the problem with a destructive but unrelated action

The peril of overemphasizing recovery

Joyent

- Recovery in lieu of understanding normalizes broken software
- If it becomes culturally engrained, the dubious principle of software recovery has toxic corollaries, e.g.:
  - Software should tolerate bad input (viz. “npm isntall”)
  - Software should “recover” from fatal failures (uncaught exceptions, segmentation violations, etc.)
  - Software should not assert the correctness of its state
- **These anti-patterns impede debuggability!**

Debugging after an outage

Joyent

- After an outage, we must debug to *complete* understanding
- In mature systems, we can expect cascading failures — which can be exhausting to fully unwind
- It will be (very!) tempting after an outage to simply move on, but every service failure (outage-inducing or not) represents an opportunity to advance understanding
- Software engineers must be encouraged to understand their *own* failures to encourage designing for debuggability

Enshrining debuggability

Joyent

- Designing for debuggability effects *true* software robustness: differentiating operational failure from programmatic ones
- Operational failures should be handled; programmatic failures should be debugged
- Ironically, the more software is designed for debuggability the less you will need to debug it — and the more you will leverage it to debug the software that surrounds it

Debugging under fire

Joyent

- It will always be stressful to debug a service that is down
- When a service is down, we must balance the need to restore service with the need to debug it
- Missteps can be costly; taking time to huddle and think can yield a better, safer path to recovery and root-cause
- In massive outages, parallelize by having teams take different avenues of investigation
- Viewing outages as opportunities for understanding allows us to develop software cultures that value debuggability!

Hungry for more?

Joyent

- If you are the kind of software engineer who values debuggability — and loves debugging — Joyent is hiring!
- If you have not yet hit your Cantrillian LD50, I will be joining Brigit Kromhout, Andrew Clay Shafer, Matt Stratton as “Old Geeks Shout At Cloud”
- **Thank you!**