Terminal Velocity

Doing DevOps Right By Removing CLIs From Production Environments

Matthew Simons
@thatsimonsguy
thatsimonsguy@gmail.com

Removing CLIs From Production Environments

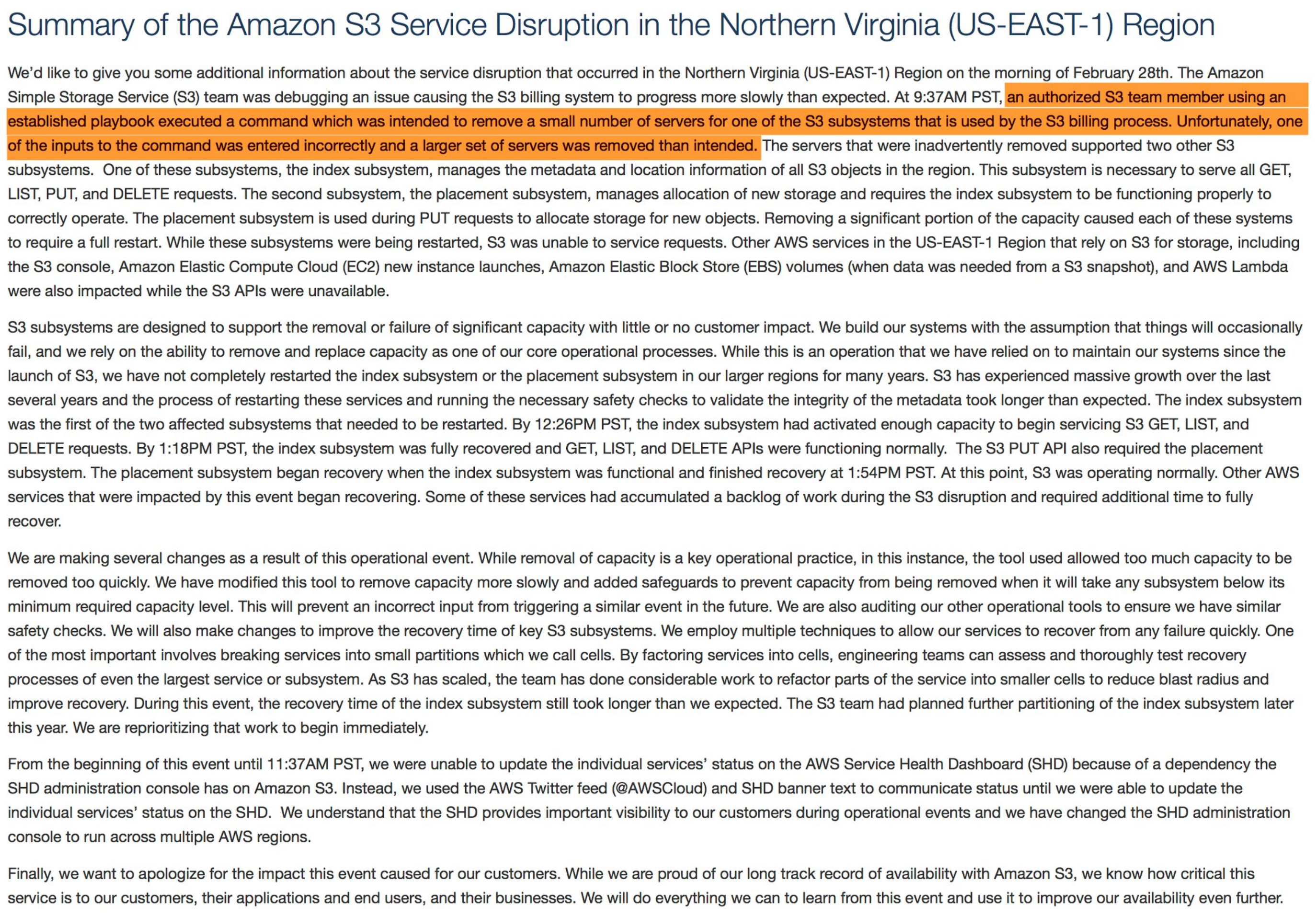
Removing

things that let you break stuff easily

from

places where breaking things is really bad

Amazon S3 - During issue remediation



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.

S3 subsystems are designed to support the removal or failure of significant capacity with little or no customer impact. We build our systems with the assumption that things will occasionally fail, and we rely on the ability to remove and replace capacity as one of our core operational processes. While this is an operation that we have relied on to maintain our systems since the launch of S3, we have not completely restarted the index subsystem or the placement subsystem in our larger regions for many years. S3 has experienced massive growth over the last several years and the process of restarting these services and running the necessary safety checks to validate the integrity of the metadata took longer than expected. The index subsystem was the first of the two affected subsystems that needed to be restarted. By 12:26PM PST, the index subsystem had activated enough capacity to begin servicing S3 GET, LIST, and DELETE requests. By 1:18PM PST, the index subsystem was fully recovered and GET, LIST, and DELETE APIs were functioning normally. The S3 PUT API also required the placement subsystem. The placement subsystem began recovery when the index subsystem was functional and finished recovery at 1:54PM PST. At this point, S3 was operating normally. Other AWS services that were impacted by this event began recovering. Some of these services had accumulated a backlog of work during the S3 disruption and required additional time to fully recover.

We are making several changes as a result of this operational event. While removal of capacity is a key operational practice, in this instance, the tool used allowed too much capacity to be removed too quickly. We have modified this tool to remove capacity more slowly and added safeguards to prevent capacity from being removed when it will take any subsystem below its minimum required capacity level. This will prevent an incorrect input from triggering a similar event in the future. We are also auditing our other operational tools to ensure we have similar safety checks. We will also make changes to improve the recovery time of key S3 subsystems. We employ multiple techniques to allow our services to recover from any failure quickly. One of the most important involves breaking services into small partitions which we call cells. By factoring services into cells, engineering teams can assess and thoroughly test recovery processes of even the largest service or subsystem. As S3 has scaled, the team has done considerable work to refactor parts of the service into smaller cells to reduce blast radius and improve recovery. During this event, the recovery time of the index subsystem still took longer than we expected. The S3 team had planned further partitioning of the index subsystem later this year. We are reprioritizing that work to begin immediately.

From the beginning of this event until 11:37AM PST, we were unable to update the individual services' status on the AWS Service Health Dashboard (SHD) because of a dependency the SHD administration console has on Amazon S3. Instead, we used the AWS Twitter feed (@AWSCloud) and SHD banner text to communicate status until we were able to update the individual services' status on the SHD. We understand that the SHD provides important visibility to our customers during operational events and we have changed the SHD administration console to run across multiple AWS regions.

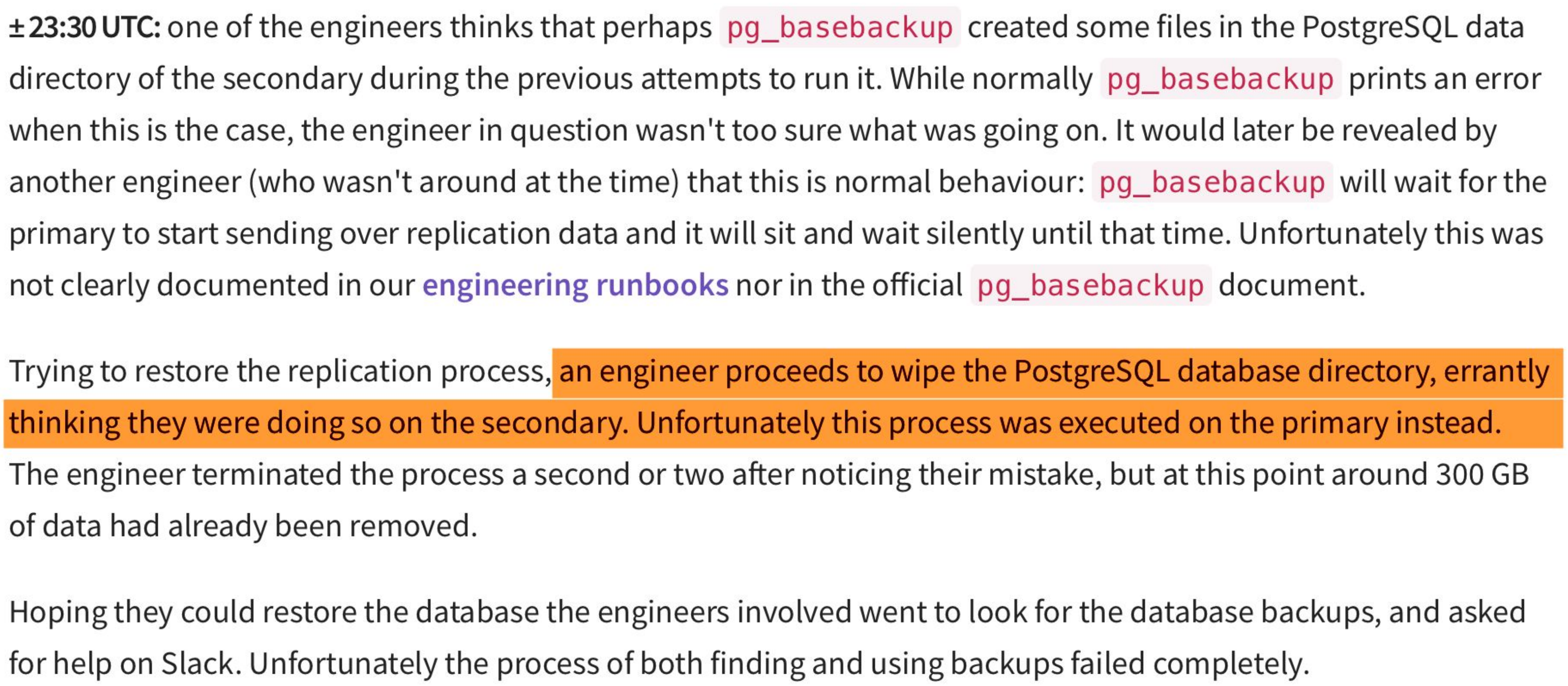
Finally, we want to apologize for the impact this event caused for our customers. While we are proud of our long track record of availability with Amazon S3, we know how critical this service is to our customers, their applications and end users, and their businesses. We will do everything we can to learn from this event and use it to improve our availability even further.

Overview of the Storage Incident

Azure - During deployment

We are continuously looking for ways to improve performance of all aspects of our platform. In this case, we developed a software change to improve Azure Storage performance by reducing CPU footprint of the Azure Storage Table Front-Ends. We deployed the software change using the described flighting approach with the new code disabled by default using a configuration switch. We subsequently enabled the code for Azure Table storage Front-Ends using the configuration switch within the Test and Pre-Production environments. After successfully passing health checks, we enabled the change for a subset of the production environment and tested for several weeks. While testing, the fix showed notable performance improvement and resolved some known customer issues with Azure Table storage performance. Given the improvements, the decision to deploy the fix broadly in the production environment was made. During this deployment, there were two operational errors: 1. The standard flighting deployment policy of incrementally deploying changes across small slices was not followed. The engineer fixing the Azure Table storage performance issue believed that because the change had already been flighted on a portion of the production infrastructure for several weeks, enabling this across the infrastructure was low risk. Unfortunately, the configuration tooling did not have adequate enforcement of this policy of incrementally deploying the change across the infrastructure. 2. Although validation in test and pre-production had been done against Azure Table storage Front-Ends, the configuration switch was incorrectly enabled for Azure Blob storage Front-Ends. Enabling this change on the Azure Blob storage Front-Ends exposed a bug which resulted in some Azure Blob storage Front-Ends entering an infinite loop and unable to service requests. Automated monitoring alerts notified our engineering team within minutes of the incident. We reverted the change globally within 30 minutes of the start of the issue which protected many Azure Blob storage Front-Ends from experiencing the issue. The Azure Blob storage Front-Ends which already entered the infinite loop were unable to accept any configuration changes due to the infinite loop. These required a restart after reverting the configuration change, extending the time to recover.

Gitlabs - During issue remediation

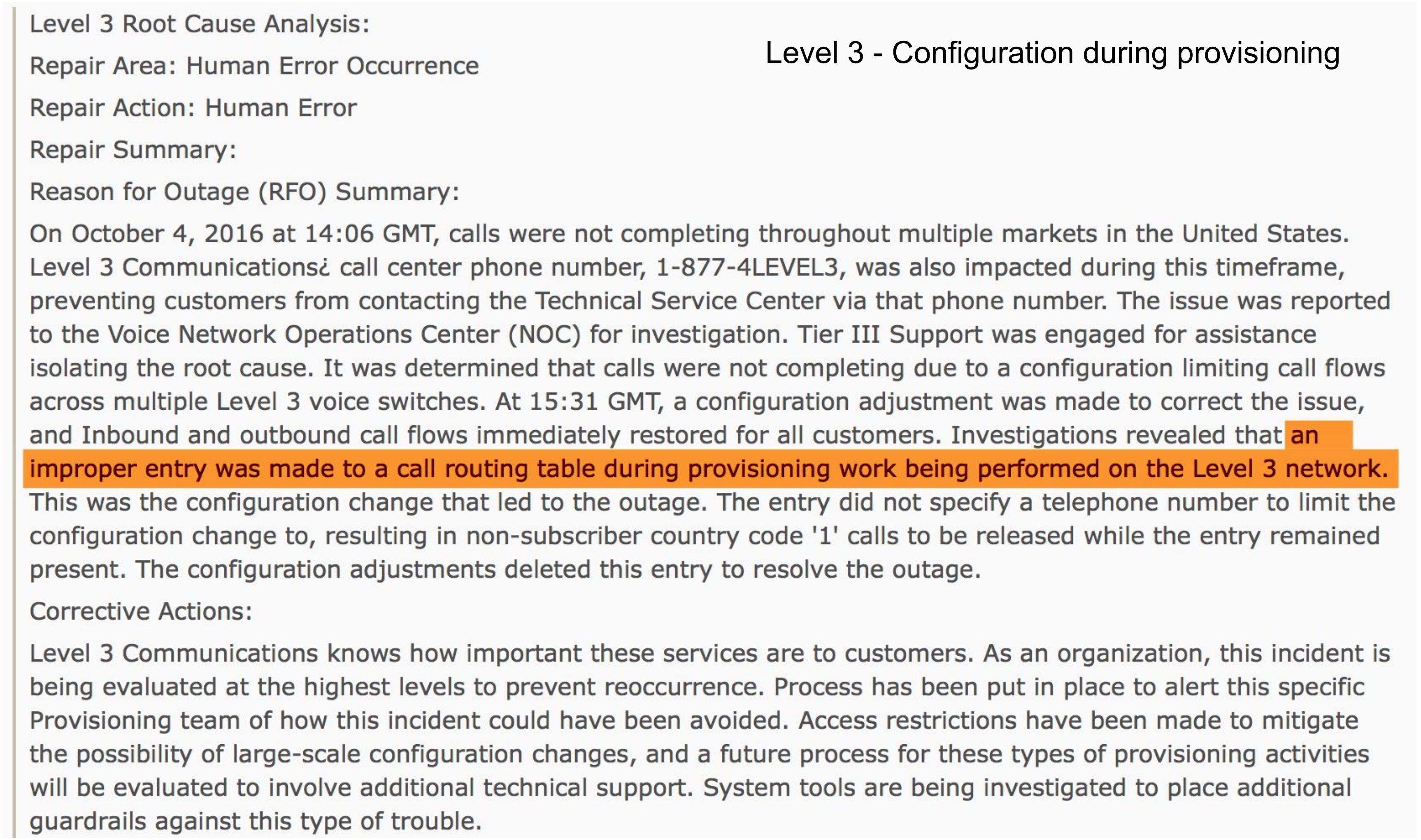


±23:30 UTC: one of the engineers thinks that perhaps `pg_basebackup` created some files in the PostgreSQL data directory of the secondary during the previous attempts to run it. While normally `pg_basebackup` prints an error when this is the case, the engineer in question wasn't too sure what was going on. It would later be revealed by another engineer (who wasn't around at the time) that this is normal behaviour: `pg_basebackup` will wait for the primary to start sending over replication data and it will sit and wait silently until that time. Unfortunately this was not clearly documented in our engineering runbooks nor in the official `pg_basebackup` document.

Trying to restore the replication process, an engineer proceeds to wipe the PostgreSQL database directory, errantly thinking they were doing so on the secondary. Unfortunately this process was executed on the primary instead. The engineer terminated the process a second or two after noticing their mistake, but at this point around 300 GB of data had already been removed.

Hoping they could restore the database the engineers involved went to look for the database backups, and asked for help on Slack. Unfortunately the process of both finding and using backups failed completely.

Level 3 - Configuration during provisioning



Level 3 Root Cause Analysis:

Repair Area: Human Error Occurrence

Repair Action: Human Error

Repair Summary:

Reason for Outage (RFO) Summary:

On October 4, 2016 at 14:06 GMT, calls were not completing throughout multiple markets in the United States. Level 3 Communications¿ call center phone number, 1-877-4LEVEL3, was also impacted during this timeframe, preventing customers from contacting the Technical Service Center via that phone number. The issue was reported to the Voice Network Operations Center (NOC) for investigation. Tier III Support was engaged for assistance isolating the root cause. It was determined that calls were not completing due to a configuration limiting call flows across multiple Level 3 voice switches. At 15:31 GMT, a configuration adjustment was made to correct the issue, and Inbound and outbound call flows immediately restored for all customers. Investigations revealed that an improper entry was made to a call routing table during provisioning work being performed on the Level 3 network. This was the configuration change that led to the outage. The entry did not specify a telephone number to limit the configuration change to, resulting in non-subscriber country code '1' calls to be released while the entry remained present. The configuration adjustments deleted this entry to resolve the outage.

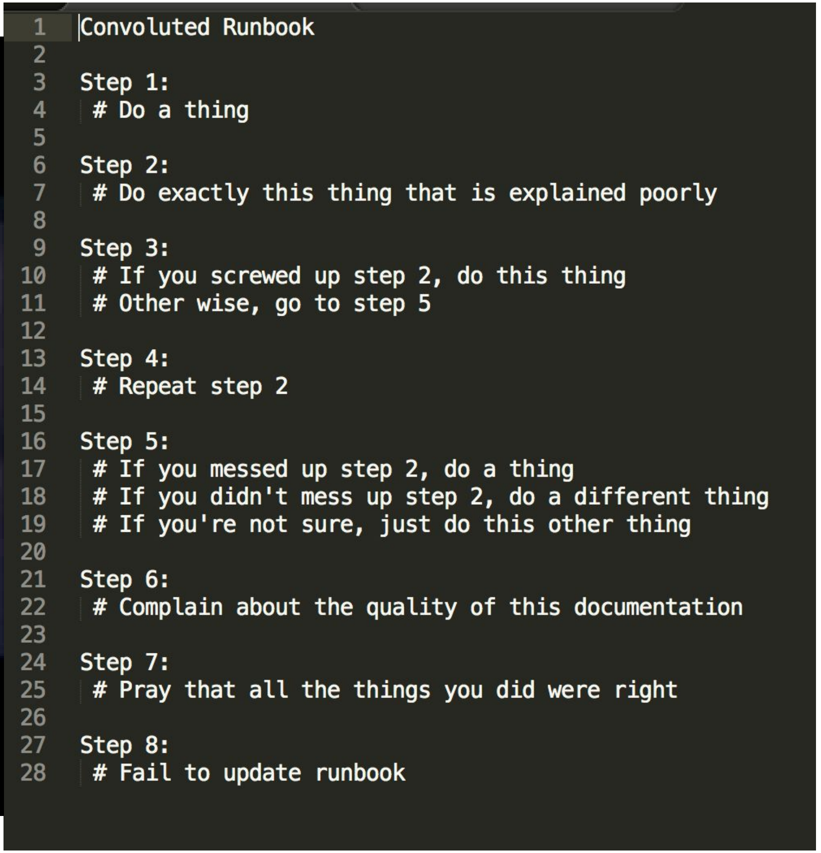
Corrective Actions:

Level 3 Communications knows how important these services are to customers. As an organization, this incident is being evaluated at the highest levels to prevent reoccurrence. Process has been put in place to alert this specific Provisioning team of how this incident could have been avoided. Access restrictions have been made to mitigate the possibility of large-scale configuration changes, and a future process for these types of provisioning activities will be evaluated to involve additional technical support. System tools are being investigated to place additional guardrails against this type of trouble.

You run arbitrary code in production

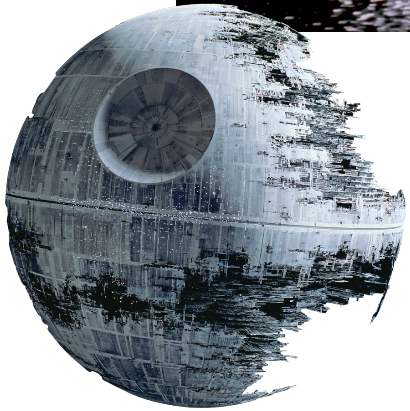
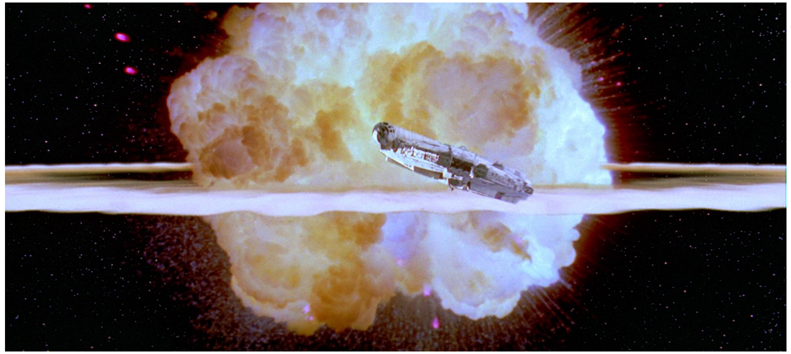
Rapid Response: A Cautionary Tale

Single Points of Failure in Process



```
Convoluted Runbook

Step 1:
 # Do a thing

Step 2:
 # Do exactly this thing that is explained poorly

Step 3:
 # If you screwed up step 2, do this thing
 # Other wise, go to step 5

Step 4:
 # Repeat step 2

Step 5:
 # If you messed up step 2, do a thing
 # If you didn't mess up step 2, do a different thing
 # If you're not sure, just do this other thing

Step 6:
 # Complain about the quality of this documentation

Step 7:
 # Pray that all the things you did were right

Step 8:
 # Fail to update runbook
```

Single Points of Failure

Single
Point
of
Failure

You do not scale

My SQL

M
S

M
S

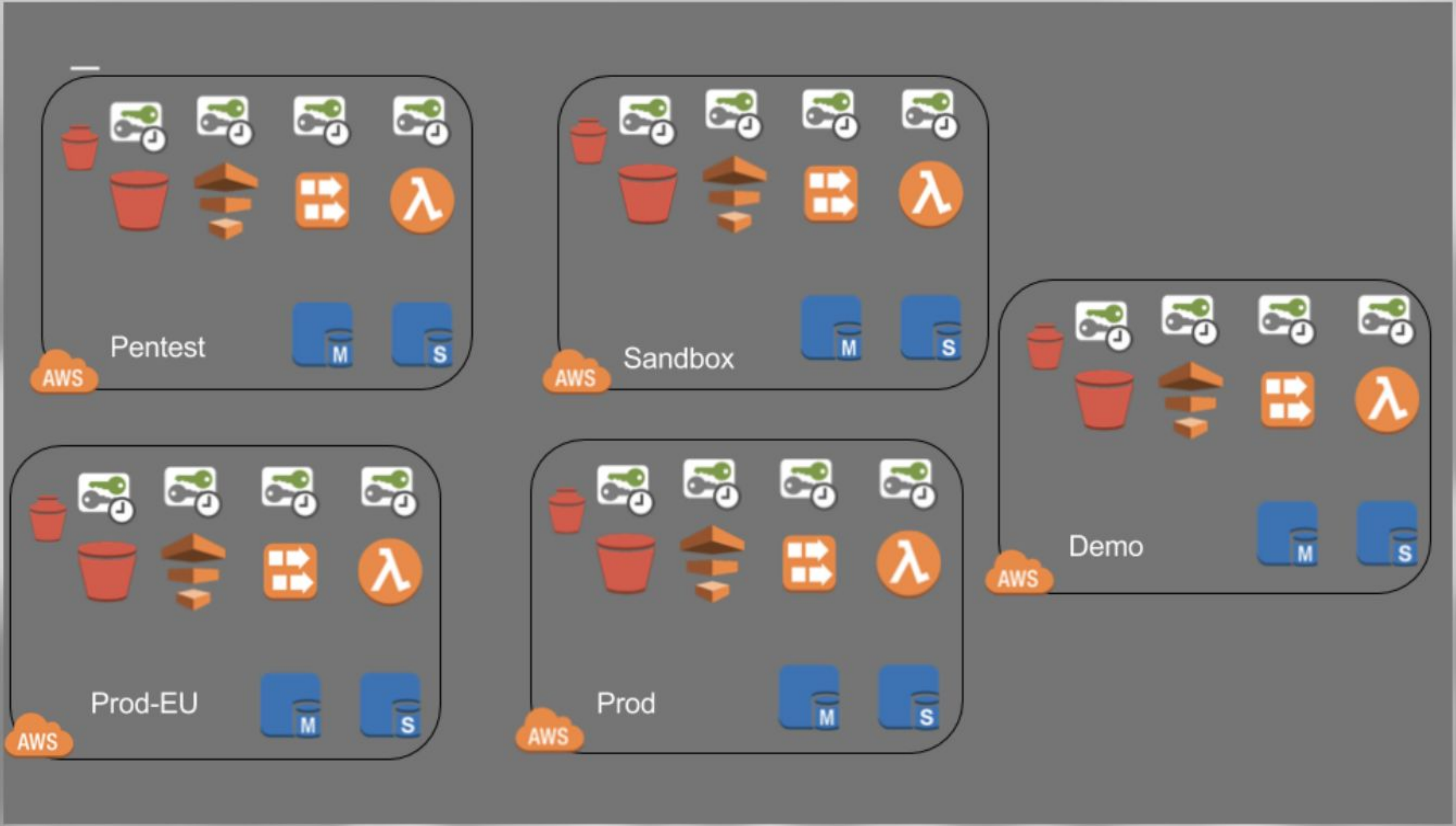
M
S

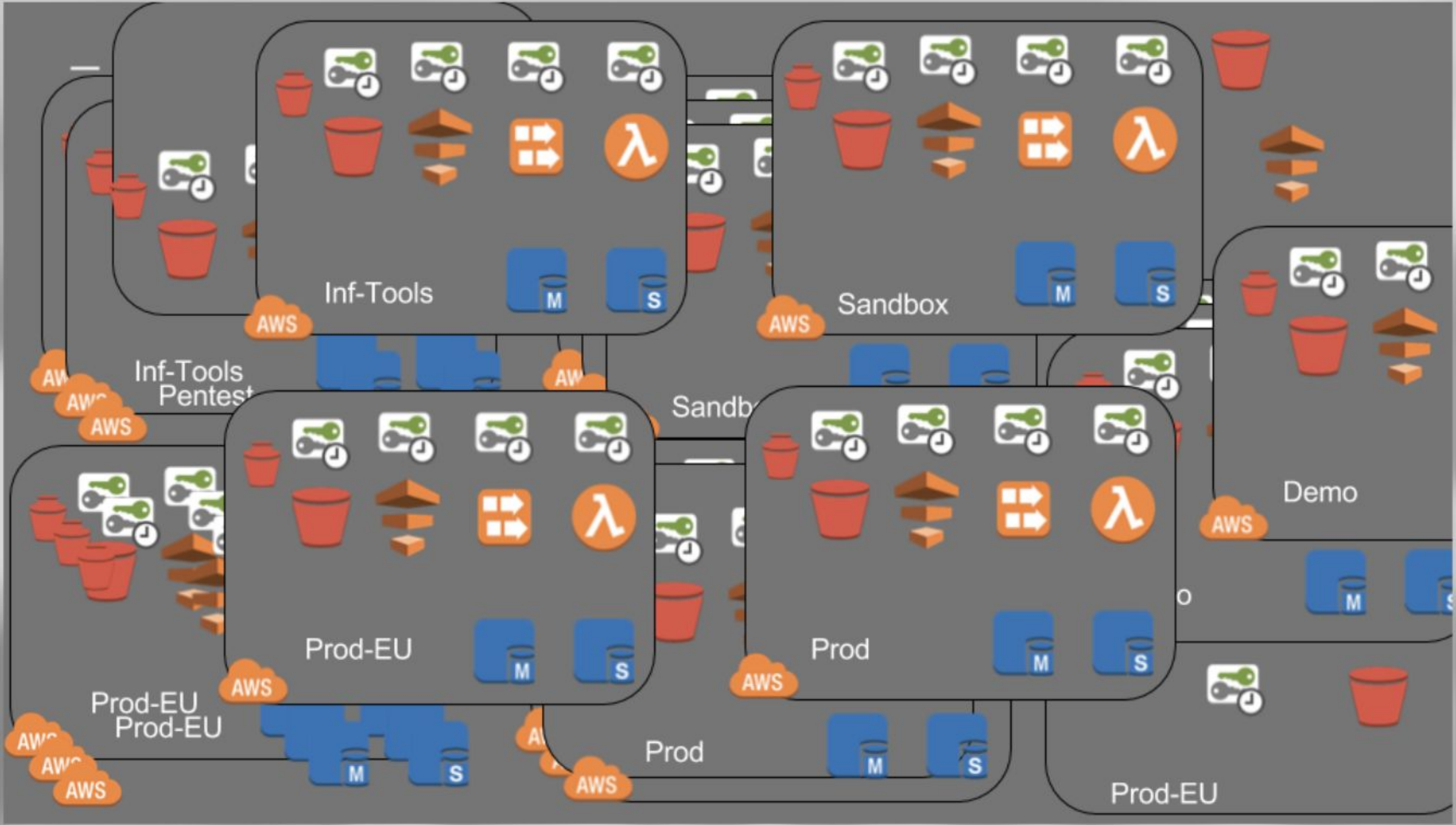
M
S

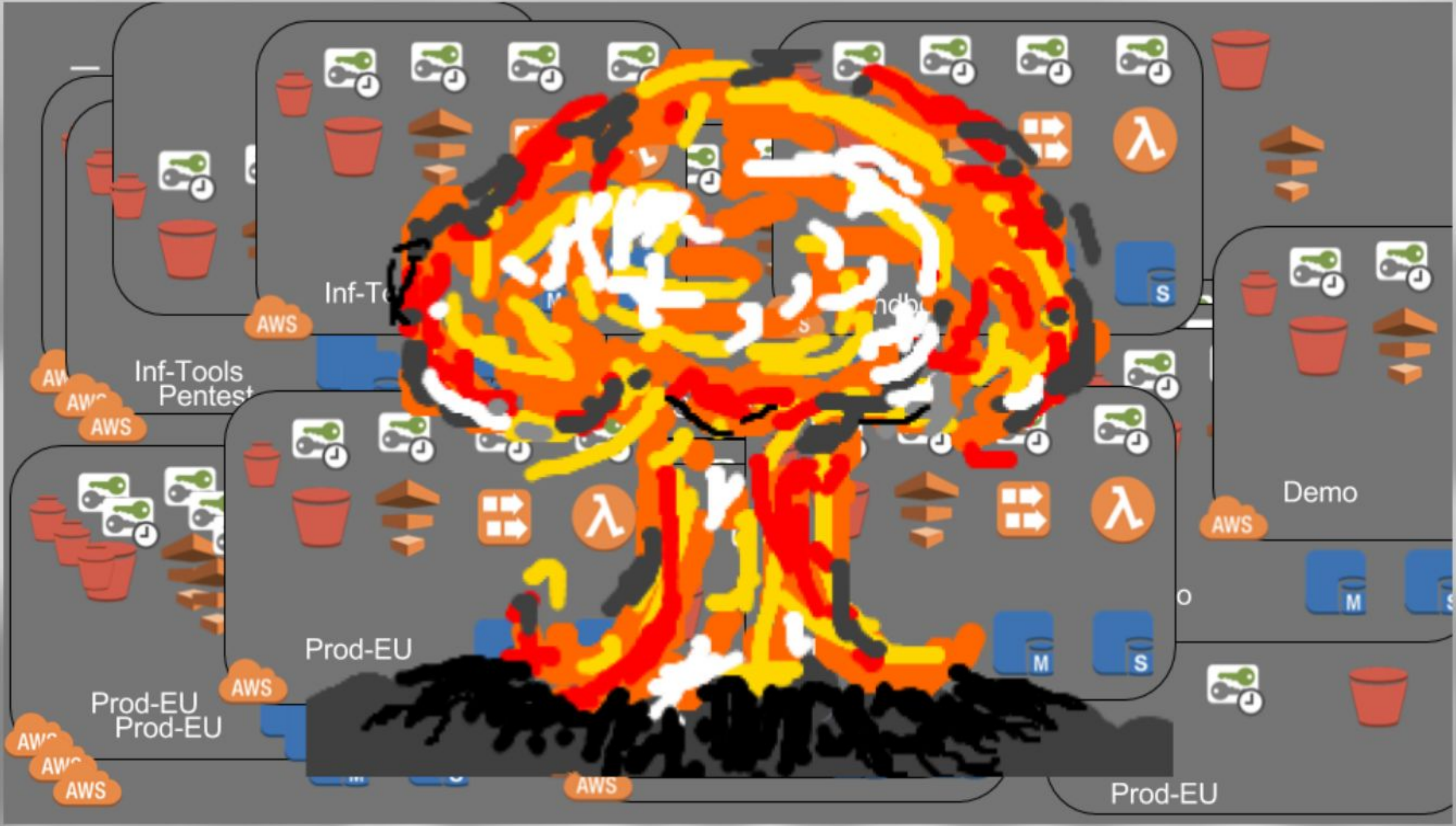
M
S

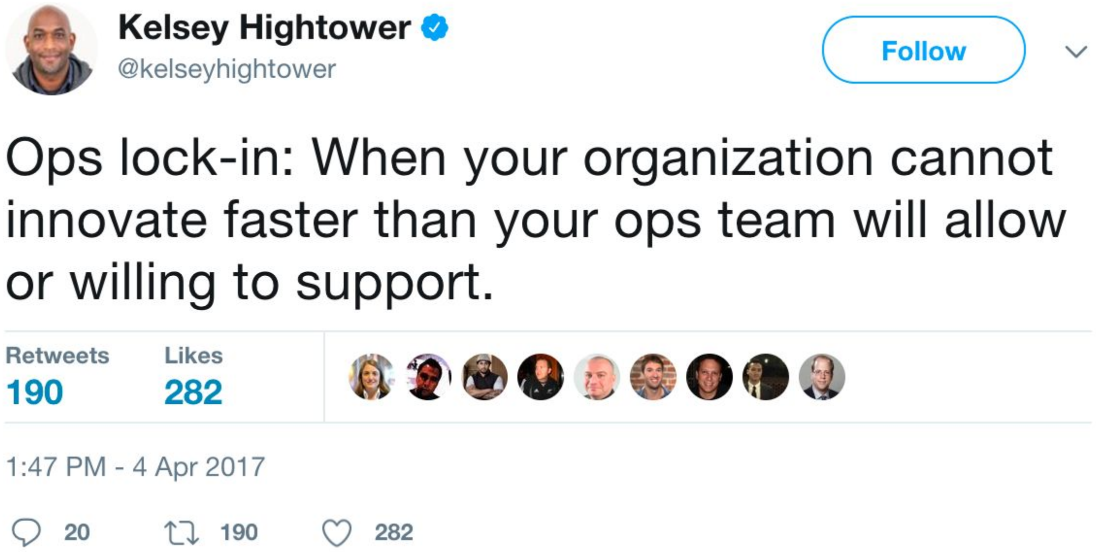
M
S

M
S


Pentest
AWS
Sandbox
AWS
Demo
AWS
Prod-EU
AWS
Prod
AWS
M
S


Inf-Tools
Inf-Tools
Pentest
Sandbox
Sandbox
Demo
Prod-EU
Prod-EU
Prod-EU
Prod-EU
Prod
Prod
AWS
M
S


Inf-Tools
Pentest
Inf-T
AWS
Demo
Prod-EU
Prod-EU
Prod-EU
Prod-EU
M
S



Kelsey Hightower
@kelseyhightower

Follow

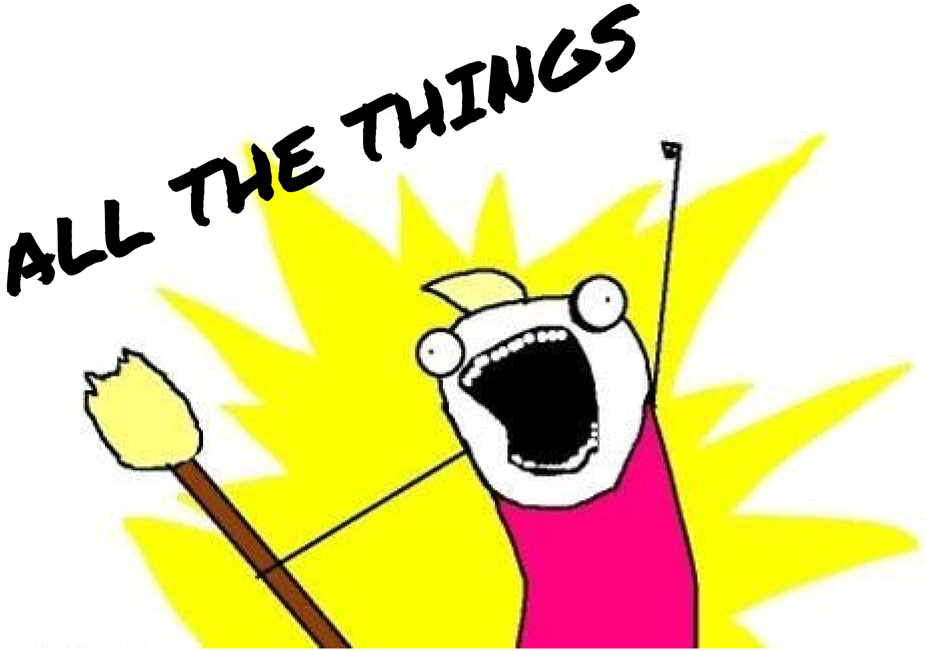
Ops lock-in: When your organization cannot innovate faster than your ops team will allow or willing to support.

Retweets 190 Likes 282

1:47 PM - 4 Apr 2017

20 190 282

_______________ as code



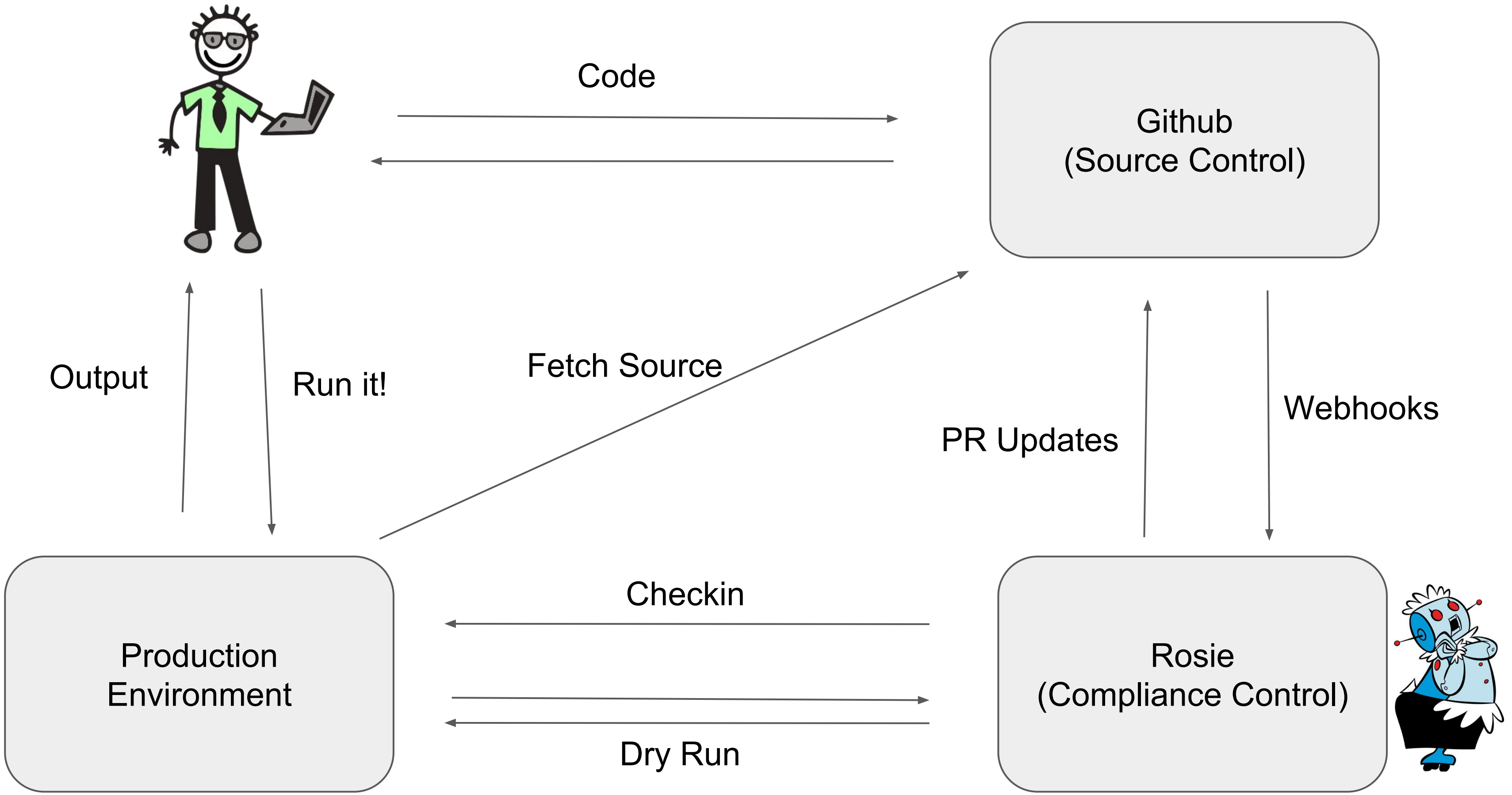
as code

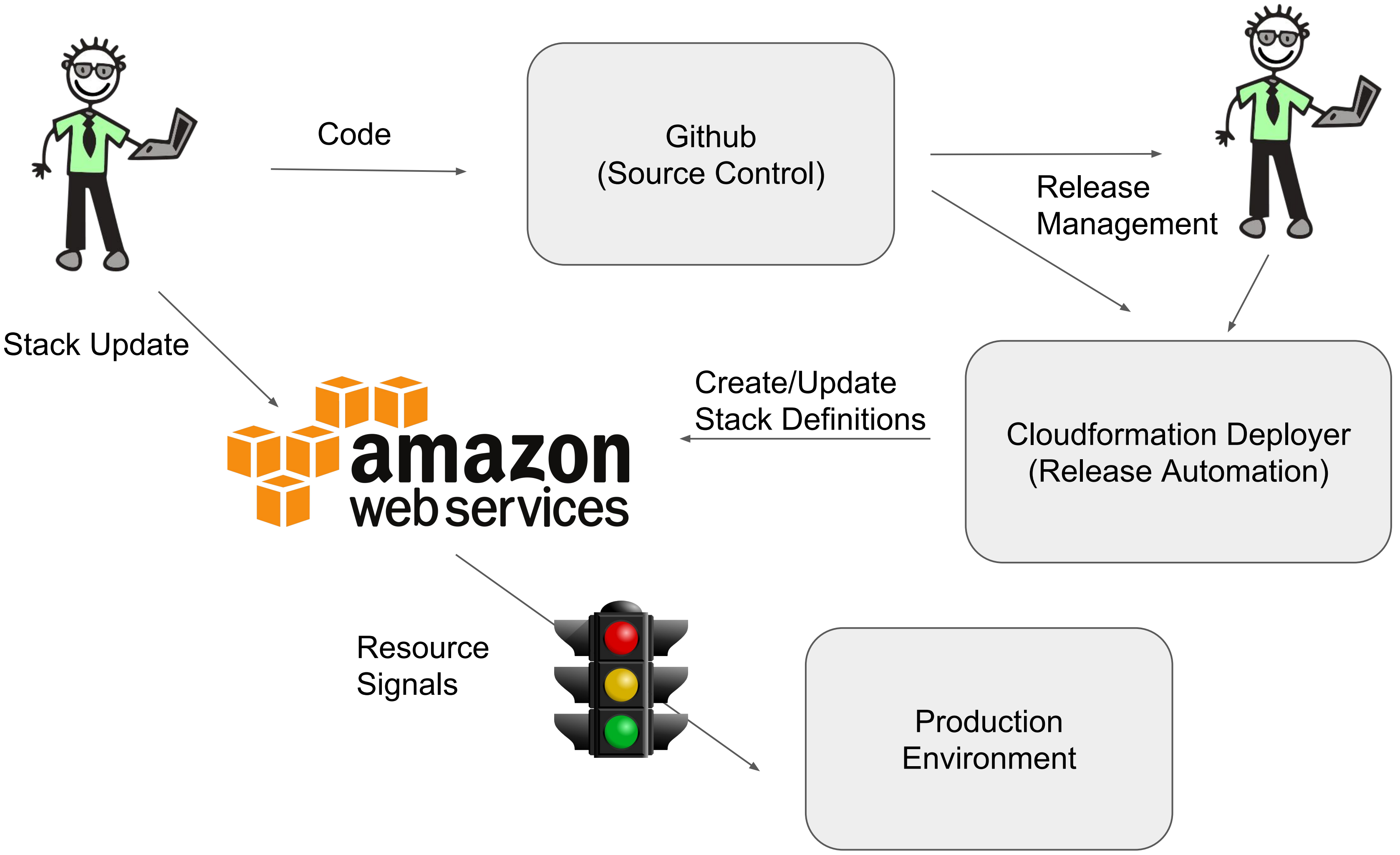
	CODE	>_
Compliant	✓	✗
Immutable	✓	✗
Auditable	✓	✗
Repeatable	✓	✗
Scalable	✓	✗
Makes you feel like a ~~badass~~ dumbass	✗	✓

Code is safe(r)

Common Components

1. Source control
2. Review
3. Testing
4. Sanity checking at runtime


Code
Github
(Source Control)
Output
Run it!
Fetch Source
PR Updates
Webhooks
Checkin
Production
Environment
Rosie
(Compliance Control)
Dry Run


Code
Github
(Source Control)
Release
Management
Stack Update
Create/Update
Stack Definitions
Cloudformation Deployer
(Release Automation)
amazon
web services
Resource
Signals
Production
Environment

Summary

Remove things that let you break stuff easily from places where breaking things is bad.

Operations as code

Crisis response as code

Everything as code

Don't be a single point of failure

Do it for you!

Matthew Simons @ workiva