Ryan Gerstenkorn

ryan_gerstenkorn@fastmail.fm

(360) 296-8611

CODING RELATED SIDE PROJECTS / VOLUNTEER WORK

- (1) *SeaUtils* – Docker like client + API gateway built in Golang with swagger.
  - Run anywhere. No docker daemon, IAM credentials. Enforces least/zero trust privilege.
- Terraform Route53 Authorization – Original PR and discussion, also see (1).
- ryanjarv/awsconfig – Config rules for detecting IMDS persistence, also see (3).
- *ryanjarv/coderun* and dockersnitch – Toy project with big ambitions.
- *sous-chefs/varnish* – Previous primary maintainer and top overall contributor.
- *ryanjarv/randrust* and puppet-randrust – Ops example custom service best practice.
- *ryanjarv/gocash* – Experiment to emulate Redis SAVE functionality in Golang.
- *ryanjarv/nettomidi* – Listen to your network in Python!
- Distributed API scraping cluster – C#, RabbitMQ, and the S2 geo library.
- Advised PokeVision devs on database scaling to handle peak loads during initial release.
  - Resulted in a three fold increase to peak concurrent users the next day.
- Previously volunteered at the Bellingham RElectronics repair and recycling store.

SECURITY RESEARCH

- (2) IMDS Persistence – Malware like behavior in AWS using malicious IMDS routing.
- Abusing the AWS SDK – Confused deputy like* type attack on the local subnet.
- SSM Parameter Store Permissions – Unintuitive IAM behavior when deny listing secrets.
- (3) Backdooring Route53 – Side effect's of missing Route53 authorization enumeration Action's
- Abusing Forwarded-For – Easy mistake to make, even easier to exploit.
- OpenBSD package signing – Non-existent (circa. 2013), fixed shortly after with signify
- Malicious Vagrant Boxes – Takeaway: Vagrant is not meant for isolation.
- VirtualBox NAT – Direct network access to the host through VirtualBox NAT behavior.
  - Received a Security-In-Depth Contributor credit from Oracle for reporting this.
  - Takeaway: Default settings offer poor privacy and isolation between the guest and host.

SR DEVOPS ENGINEER, SAILPOINT; JAN 2020 - SEP 2020

- Developed a Serverless container build system for use outside of AWS.
  - SailPoint has kindly allowed me to continue working on after leaving. Also see (1) above.
- CI/CD update automation. Backup, deploy, health checks and restore on failure of JCASC.
- Lead team migration to MFA'd and encrypted AWS credentials in a multi account setup.
- Lead plan for reducing risk on high risk Amazon SES changes.
  - * Implemented feature flagged A/B rollout to allow for gradual go live over several weeks.
  - Published customer facing post on what/why/when. Worked directly with customers to resolve any related concerns.

SR DEVOPS ENGINEER, BRAVE; MAR 2018 - OCT 2018

- Lead client update build process and service migration for the Chromium based Brave browser.
- Managed and improved build infrastructure for both Muon and Chromium based browsers.
- Automated existing AWS infrastructure and processes using Terraform and Ansible.
- Security on-call contact for infrastructure related issues.

DEVOPS ENGINEER; CURSE SEP 2014 - MAR 2018

Curse was acquired by Twitch and (in turn) Amazon in 2016. Since I worked closely with teams across both companies during this time I've split this period up in to separate sections.

CURSE SEP 2014 - MAR 2018

Lead operations and networking with the goal of reducing IT overhead by automating and codifying common tasks and best practices. The majority of the time I acted as the sole DevOps engineer across Curse, but had significant help from the two other individuals on the IT team on many projects. Later on my roles included building infrastructure and mentoring others in order to take on tasks previously handled solely by the DevOps team with the goal of creating self sufficient teams.

During our migration to AWS I lead architecture and tooling efforts while my manager worked with me and upper management to rethink how we approached operations across the organization. Despite me needing to take personal leave for the second half of the AWS migration, the IT team was able to successfully finish our zero-downtime migration effort's and unplug our colocated site using the architecture and tools I had worked on up to that point.

AWS architecture design

Scalability and security was met by implementing a loosely coupled design between ephemeral and non-ephemeral resources as well as segmenting end user services by AWS account. Designing flexible terraform modules as well as grouping common patterns across our org allowed us to stamp out the infrastructure as needed for the wide range of web stacks we ran.

Avoiding shared networking completely, as well as shared resources where possible allowed for a naturally resilient infrastructure where blast radius from attacks or user error was greatly reduced. Due being the sole DevOps engineer alongside a small IT team with limited resources, I

leaned in favor of optimizing for expertise amongst a smaller toolset, rather than always using the right tool for the job.

We relied on both Terraform and BeanStalk heavily, extending their capabilities when needed to suit our need. Terraform code was organized a separate module repo, referenced in environment templates by SerVer based version tags. Environment templates where deduplicated using a crude Git branching model which attempted to hierarchically define all of our environments, i.e the main branch being a common subset of all environments, stack type (.NET and linux) branched off main, branching again with either stack subtype branching or concrete leaf environments. This worked well enough for our purposes at the time but ended up difficult to maintain later on. This complexity was expected with the decision to consolidate on minimal tooling.

With this setup we where able to manage our infrastructure, fully defined in terraform, with approximately 50 terraform modules as well as 50 functional leaf Git branches representing over 200 distinct clustered services across 50 AWS accounts.

Access across accounts was handled by a well defined naming scheme using assume role permissions. This resulted in a somewhat rigid but powerful authorization system but was robust and easy to use. An example of this is our Jenkins system was able to drop IAM privileges before executing individual deployment jobs without any central or managed configuration, a well known source account and naming guarantees between disparate systems made this possible.

Twitch considered adopting this design after seeing it work in practice, however changing existing naming schemes in AWS often prove to be difficult. This situation has likely changed with recent session tagging improvements added by AWS, implementing a similar solution could likely be done without worrying about resource naming changes.

Colocated architecture design (replaced by AWS architecture above)

The original network consisted of an in-house BGP/OSPF network of around 2 thousand VM's, 200 clustered services, and 50 public facing websites all running across a various of web stacks. This was fronted by a distribution network serving 20 billion requests monthly. Clustered services where split amongst three availability zones with each zone representing a separate cluster of HyperV hypervisors. While failure over was possible both between hypervisors in a zone and application level clustering, the later was preferred.

All services ran in either an active active or read/write primary, or readonly secondary setup. Client side HA Proxy configurations where used for load balancing, routing and health checking between services. This setup was surprisingly robust, simple to maintain, and core to allowing us to operate at scale with minimal resources. Accounting for public routing issues outside of our control we achieved 99.999% uptime despite many catastrophic hardware, software and internal network failures.

During the early days of the Twitch desktop app (aka Curse Voice) we also managed a separate AWS network of considerable size. Efforts and on-call burden was largely focused on our need for low latency world-wide replication of data. At this time (2014-'15) networking between AWS datacenter's wasn't nearly as reliable as it is today, and presented a major hurdle to operating our voice service at peak efficiency. To work around this at the time we ran infrastructure across a OSPF routed full mesh VPN network. While OSPF convergence handled major outages

(The Curse II: The Curse Continues Again — Colocated architecture design)

automatically a secondary system was built to monitor and route around brief intermittent issues and large TCP round trip times.

Early on I realized attempting to predict outages via monitoring was not a good use of our time and caused considerable unnecessary after hours on-call work. Shifting to a focus around building infrastructure that could consistently withstand failure as well as investing heavily in observability allowed us to significantly reduce on-call work at the same time as improving overall uptime. This allowed us to only alert on things that actively needed maintenance and could not fail safely, this, along side APM (NewRelic) and a well designed metric and graphing system (1 billion distinct metrics a minute) covering insight to underlying infrastructure performance was critical to managing a sane level of after hours on-call work.

Our logging infrastructure (not including APM) existed mostly to detect network issues outside of our control. BGP routing issues where common, but just about anything that could happen did happen and we where typically the first to notice and report issues to our upstream providers.

The following projects where joint efforts across our IT and/or DevOps team.

- Maintained a site-wide job scheduler and coordinator written in python + celery which ran jobs across approximately 1 thousand nodes.
- Security incident response and post mortem.
- Migrated linux infrastructure from Puppet to Chef. We also investigated managing windows systems using IaC at this time, but ultimately decided against it.
- Maintained Curse's OSPF and BGP networks, occasionally deep diving into network troubleshooting and making configuration changes to fix identified issues.

Below are efforts I primarily lead myself.

- Added simplified dependency management, unit tests, and a CI pipeline for our Chef infrastructure composed of about 100 private cookbooks. This was critical in facilitating PR's from other people, and removing myself from being the sole maintainer.
- Designed and implemented a hardened LDAP password reset web app exposed through 2FA enforcing proxy and WAF.
- Tracked and fixed a long standing, inconsistent and difficult to pinpoint issue with POST requests failing only when sent through CloudFlare's CDN.
- Identified and fixed various security issues, working across several teams to resolve issues.
- Integrating our wide range of app's and services with a common secret management system.

TWITCH – CURSE AUG 2016 - MAR 2018

- Managed a tight deadline and exceeded the Curse IT acquisition requirements set by Twitch.
  - My effort here was recognized with the first employee of the month award under the then new Curse subsidiary of Twitch/Amazon.

(Curse III: Eternally Cursed)

AWS/AMAZON – CURSE AUG 2016 - MAR 2018

Lead infrastructure design and deployment of amazonforums.com replacing the previous Amazon digital and device forums hosted under the amazon.com domain. Working heavily with developers, management and internal Amazon teams to turn around a project that was seemingly destined to fail.

To expand on this our core issue revolved around the fact that no one at Curse was ever trained on Amazon's internal processes, tools, or relevant policies. We made a surprising amount of progress given our situation, however our lack of broad high level experience in the area repeatedly sent us down the rabbit hole chasing blocker after blocker, going from team to team asking for advice.

To resolve this, I decided to push for calculated high level design changes with our clients at Amazon which ended up eliminating 90% of our work. It also decreased our risk classification greatly, resulting in eliminating several open positions and significantly reduced our expected ongoing maintenance workload.

The reason I was able to make a drastic change in direction here so quickly was largely thanks to Amazon's internal documentation and an impromptu office seating change. After outlining the processes for each decision we could have made using, it was clear that none of these options would work well for us on our timeline and resources. Adjusting what we where trying to do from an infrastructure perspective allowed us to safely side step many issues, and work in a way that we already had the people and skills to execute on. We ended up having the system operational and ready for initial testing in just weeks after my suggested changes.

The impromptu seating change resulted in working closer with developers on this project, and catching this issue before it became a problem. The success of this experiment helped push an org wide change in thinking that focused on better communication between operations and development.

ENTERPRISE NETWORK TECHNICIAN, API DIGITAL; JUN 2014 - SEP 2014

- Provided emergency network support for Cisco and Adtran networking equipment.

SYSTEMS ADMINISTRATOR, DIGITAL FORTRESS; APR 2012 - JUN 2014

- Provided support for Unix, OS X, and Windows systems.
- Lead consolidation of several disparate Nagios systems due to a previous merger.
- Replaced the aging backup system with a Bacula based infrastructure secured through an out of band backup network and a private x509 cert authority for encryption.
- Identified and fixed several near catastrophic failure issues with authoritative DNS.
- Managed authoritative DNS, hosting approximately 50 thousand records.
- Rebuilt Unix web infrastructure as well as investigated provisioning services with IaC.

REPAIR TECHNICIAN, VARIOUS; OCT 2008 - APR 2012

- Repaired Windows, Apple and Linux devices
- Circuit board soldering/repair and SMD reflows

Notes, Opinions and other Dangerous Things

- Currently fluent in Golang, Bash, Python.
  - Also familiar with: Ruby, C#, and Rust.
- Recently been focusing on Serverless in AWS utilizing API Gateway, Swagger and SAM (Thoughts on serverless).
- Experience designing secure, highly available systems and infrastructure (Thoughts on Terraform and Design).
- Proficient in a wide range of AWS services and tools (list available on request).
- Proficient with various container technologies, mainly Docker and ECS Fargate. But previously FreeBSD jails, LXC and some Kubernetes (Thoughts on Kubernetes).
- Deep knowledge of networking and system fundamentals with plenty experience debugging issues in complex production environments.