The Evolution of Network Automation at Roblox

Mayuresh Gaitonde | Network Reliability Engineer
Agenda

- Background
- Initial automation framework
- Challenges
- Automation 2.0
- Takeaways / Learnings
- Q&A
What is Roblox

- Massively multiplayer online game creation platform
- Gaming + social
- Core audience is children aged 9-12
- Over 100 million monthly active players
Background

- Single Data Center in Chicago
- Legacy hardware and outdated network design
- Player growth
- SOS!
Roblox Cloud

- New US-Central Datacenter with in house compute
- Spine-leaf fabric
- 9+ new POPs with game servers
- Single-vendor
- Automation first approach
Enter the NRE team...

- Network Reliability Engineering
- Build a scalable, robust and reliable automation stack
  - Automation should facilitate network growth
- Focus on network reliability
  - Monitoring and alerting was the #1 priority
- Customer centric view
Source of Truth

- Netbox
- Network builder
  - Populate Netbox based on user defined intent
  - Declarative tool to express network intent in the form of YAML templates
  - Idempotent resource allocation engine (ASNs, Interfaces, IPs)
  - Convert rendered templates into netbox objects
Monitoring and Alerting

- Priority # 1
- Collect, store, visualize
- Custom collector
  - Netconf + JSON
  - Based on Open Source vendor libraries
- TSDB
  - Leverage org wide data store infra
  - TICK Stack
  - Alerting required writing TICK scripts
- Dashboards
  - Grafana
- Syslog
Configuration Management and Device Provisioning

- Ansible
  - Host + Group vars + Netbox data
  - Collection of playbooks to serve various operational needs
  - Use vendor modules wherever possible
  - AWX for UI based jobs

- Device Provisioning
  - Ansible playbooks for device provisioning over console
  - Used legacy console scripts from vendors
Not exactly a bed of roses...

- Limited resources across different pods
- 2-3 member NRE team to maintain the entire stack  
  - Majority of time spent in KTLO
- We needed to re-think the automation stack  
  - Overcome challenges with current stack  
  - Enable self-service where possible
Challenges with SOT

- Netbox API limitations
- Config generation required making multiple API calls
- NetBuilder v1.0 was showing its age
  - Lacked a proper schema
  - Lacked unit tests
  - Too many templates / variations
The Solution...

- Build Custom APIs that pertain to the business logic
- Initially maintained as a Netbox plugin
- Network API
  - Decouple business logic from SOT
- Netbuilder v2.0
  - Schema based templating engine
  - Redesigned the templating schema
  - Rewrote resource manager
  - Slew of unit tests
Challenges with Config Management and Deployment

- Ansible’s weaknesses start showing at scale
  - Too slow for our use case
  - Cryptic internals make debugging hard
  - Ended up writing Python modules for a lot of use cases
  - Scaling issues with AWX

- Device provisioning was slow and error prone
  - Console playbooks had only a 50% success rate
  - Lacked any checks

```shell
shell = "(?P<shell>%!#|\-\$)"\s*$"
conn = self.connect()
self.expect_prompt(shell)
conn.send(username)
self.expect_prompt("Password: ")
conn.send(password)
self.expect_prompt(shell)
conn.send(config[:10])
conn.send(config[10:50])
conn.send("The 90s called...")
```
The Solution...

- **Nornir**
  - Pure Python
  - Base functionality was not enough
- Built a config management and tooling framework on top
  - Collection of operational Python scripts
  - Extensible, multi-vendor support
  - Retained Ansible style filtering and variable management
- **Zero Touch Provisioning**
  - Multi vendor support
  - Added pre and post provisioning checks
- **Job Runner UI**
  - Push button device provisioning
The Common theme so far ..?

- Question the status quo !!
- Regular “Customer” feedback is critical
- v2.0 = best_of(v1.0) - worst_of(v1.0) + ...
Challenges with Alerting

- Naively sent all alerts to a Slack channel
- Hard to distinguish noise = alert fatigue
- No unified view of alerts
- Hundreds of trivial repetitive alerts
- Needed a comprehensive alert management framework
The Solution...

- Alert Manager
  - In house tool written in Go for alert management
- Single pane of glass for all alerts
- Visualize and interact with alerts
  - API driven
- Tune out the noise
  - Alert aggregation and inhibition
- Plug into S.O.T
  - Automatic alert suppressions
- Enabled more advanced workflows
  - Auto remediation

Challenges with Monitoring

- **Storage issues with InfluxDB**
  - Inefficient storage engine
  - Retention limits

- **Writing alerts meant writing TICK scripts**
  - Steep learning curve
  - Debugging and adding new alerts was time consuming
  - Started hitting scaling limits over time

- **Dashboarding required InfluxQL knowledge**
  - SQL-like but not quite
  - Complex visualizations require trial and error

- **Did not fit into our “self-service enablement” model**
The Solution...

- Prometheus
- 5x Storage efficiency for our metrics, YMMV
- Easier to learn and write PromQL
  - Enables other engineers to create their own dashboards and queries
- Built in alerting system using the familiar PromQL
  - No separate alerting component needed
The NRE Self-Service Model

- Build frameworks, not scripts
- Plugin Driven Development
- Allow network engineers to write their own scripts, audits etc.
- Allow network engineers to create their own dashboards
- Provide UI tools to offload trivial tasks (e.g. TOR provisioning)
- Look for collaboration opportunities outside Networking
Key Learnings

- Automation first approach
- Hire for automation!
  - No longer a nice-to-have
  - Automation adds strategic value
  - Give it equal importance as your core product or SWE teams
- Iterate - done is better than perfect
- Standardize where possible
  - Clean, robust automation goes hand in hand with clean, standardized network designs
- Does out-of-the-box functionality work for you?
  - Built in APIs and features may be limiting
  - Be prepared to write extensions to serve your business logic
Key Learnings contd..

- Open Source is not always the right answer
  - Don't fall for the number of github stars!
  - Control your own destiny
  - Think strategic

- Avoid one-stop-shop automation solutions
  - Unless something is better than nothing

- Automation as an enabler
Thank you !