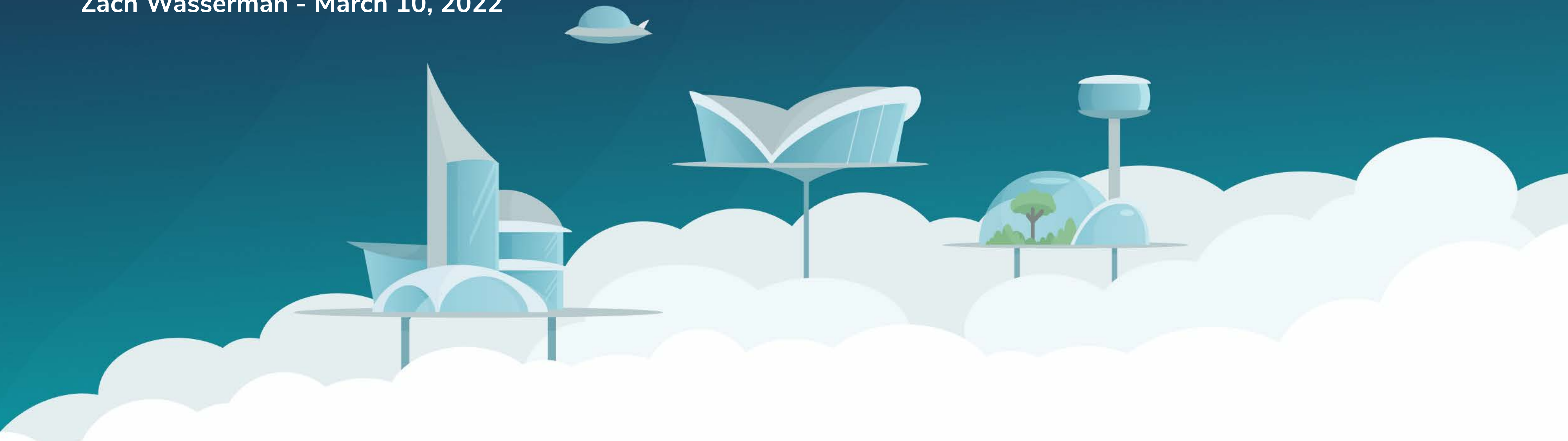


Zach Wasserman - March 10, 2022



Glitches in the Matrix, Or Taming Agent Chaos

Conf42 Chaos Engineering 2022





Performant endpoint visibility

Transform your OS into a virtual database.

- Developed at Facebook, now a project of the Linux Foundation.
- Open source (MIT License) - github.com/osquery/osquery
- Cross platform - macOS, Windows, Linux, BSD (limited).
- Read only - limited by design.
- Agent deployed across all endpoints (production, corporate, workstations) for security, IT, or operations use cases.



Open source device management

Deploy and manage osquery on 100,000+ devices.

- Open core (MIT/Proprietary License) - github.com/fleetdm/fleet
- Linux server + cross platform CLI tool (fleetctl).
- Two classes of client:
 - Osquery agents - Retrieving configurations, sending collected data.
 - API clients - Humans (or scripts) modifying configurations or retrieving data.

Engineering Resilience

Engineering Resilience

Focus

- Identify areas of key risk, apply mitigations focused to those areas.
- Availability > Integrity > Cost (to some extent)
- Pareto Principle/Amdahl's law
 - Focus on the greatest contributors to risk.



Osquery (Agent)

Production availability

+

Workstation usability

+

Monitoring integrity

+

Compute cost

=

High Risk

Fleet (Server)

Monitoring availability

+

Monitoring latency

+

Monitoring integrity

+

Compute cost

=

Medium Risk

Osquery

Watchdog

- Dual process worker/watcher model.
- Osquery self-watches for CPU and memory utilization, terminating any query that exceeds the set utilization limits, and blocking that query for 24 hours.
- Mitigates:
 - Production availability.
 - Workstation usability.
- Downside: Blocking queries reduces monitoring integrity.



Osquery

Linux cgroups

- Ask the kernel to maintain strict limits on the amount of CPU and memory utilized by the osquery process.
- Mitigates:
 - Production availability.
- Downside: Only compatible on Linux.



Osquery

Query Performance Profiling

- Use tooling to estimate relative performance of queries.
- Mitigates:
 - Monitoring integrity
 - Compute cost



```
Profiling query: select * from processes
```

```
U:1 C:0 M:2 F:0 D:0 processes (1/1): utilization: 9.8 cpu_
0.099889228 memory: 18640896 fds: 4 duration: 0.5181262435
```

```
Profiling query: select * from users join user_groups using
groups using (gid)
```

```
U:2 C:1 M:2 F:0 D:2 user_groups (1/1): utilization:
28.299999999999997 cpu_time: 0.570734208 memory: 19369984
```

```
Profiling query: select * from time
```

```
U:0 C:0 M:2 F:0 D:0 time (1/1): utilization: 5.35 cpu_time
0.056201881999999995 memory: 16080896 fds: 4 duration: 0.5
```

Osquery

Query Performance Monitoring

- Record statistics for real life query execution.
 - Shown here with simplified rendering in Fleet UI
- Mitigates:
 - Monitoring integrity
 - Compute cost

All teams ▾ Advanced

Schedule queries to run at regular intervals across all of your hosts.

9 queries

<input type="checkbox"/>	Query	Frequency	Performance impact	
<input type="checkbox"/>	Get syslog events	6 hours	Minimal	Actions
<input type="checkbox"/>	Get USB devices	1 day	Minimal	Actions
<input type="checkbox"/>	Detect dynamic linker hijacking on Linux (MITRE. T1574.006)	1 week	Minimal	Actions
<input type="checkbox"/>	Get network interfaces	1 day	Minimal	Actions
<input type="checkbox"/>	Get installed Safari extensions	1 day	Minimal	Actions
<input type="checkbox"/>	Get disk encryption status	1 day	Minimal	Actions
<input type="checkbox"/>	Detect machines with Gatekeeper disabled	1 day	Minimal	Actions
<input type="checkbox"/>	Get installed Chrome Extensions	1 day	Minimal	Actions
<input type="checkbox"/>	Count Apple applications installed	1 day	Minimal	Actions

Fleet

Common Service Practices

- Multiple Fleet server processes run behind a load balancer.
- MySQL/Redis dependencies clustered with failover.
- Utilize autoscaling for efficient infra sizing.
- Mitigates:
 - Monitoring availability, latency, integrity
- Downside: Without proper sizing, can increase compute costs.



Fleet

Backpressure (Buffering)

- Buffer data on the clients (osquery) until the server is ready to ingest it.
- Requires coordination between client and server.
- Client buffers logs until the server confirms receiving them successfully.
- Mitigates:
 - Monitoring integrity, compute costs
- Downside: (Possibly) increased latency, integrity compromised in extreme cases.



Engineering Resilience

Engineering Resilience

Self Managed

- Fleet's software (both agents and servers) is entirely self-managed by customers.
- Self Managed Challenges:
 - Environments are inconsistent.
 - Deploys are slow (not in our control).
 - Debugging feedback loop is slow.



Engineering Resilience

Consistency

- More heterogeneous deployments = more edge cases.
- MySQL... or MariaDB, Aurora, etc.
- Redis... Cluster, Sentinel, etc.
- Encourage deployment consistency.
 - Infrastructure as code (Fleet Terraform).
 - Reference Architectures (Fleet Reference architecture).



Up to 25000 hosts

Fleet instances	CPU Units	RAM
10 Fargate task	1024 CPU Units	4GB

Dependencies	Version	Instance type
Redis	6	m6g.large
MySQL	5.7.mysql_aurora.2.10.0	db.r6g.large

Up to 150000 hosts

Fleet instances	CPU Units	RAM
30 Fargate task	1024 CPU Units	4GB

Dependencies	Version	Instance type	Nodes
Redis	6	m6g.large	3
MySQL	5.7.mysql_aurora.2.10.0	db.m6g.8xlarge	1

Engineering Resilience

Testing

- Automate, automate, automate.
- Metric: How many experiments can we run per week?
- More experiments = more chaos = more edge cases = more issues detected
- Infrastructure as Code comes into play here again.
 - Easier to spin up and down and change parameters of test environments.



Engineering Resilience

Testing

- Generic HTTP testing tools may not cover your edge cases, and don't know the production hot-paths.
- Build custom tooling!
- For Fleet, we created custom tooling to simulate osquery agents.
 - Hot path: Agent check-ins, processing received data.
 - Simulate agents efficiently (no need for a full osquery process, just pretend to be a remote osquery).



```
38     },
39     "system_info": {
40         "computer_name": "{{ .CachedString "hostname" }}",
41         "cpu_brand": "Intel(R) Core(TM) i7-4770HQ CPU @ 2.20GHz\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000",
42         "cpu_logical_cores": "8",
43         "cpu_physical_cores": "4",
44         "cpu_subtype": "Intel x86-64h Haswell",
45         "cpu_type": "x86_64h",
46         "hardware_model": "MacBookPro11,4",
47         "hardware_serial": "D02R835DG8WK",
48         "hardware_vendor": "Apple Inc.",
49         "hardware_version": "1.0",
50         "hostname": "{{ .CachedString "hostname" }}",
51         "local_hostname": "{{ .CachedString "hostname" }}",
52         "physical_memory": "17179869184",
53         "uuid": "{{ .UUID }}"
54     }
55 },
56 "host_identifier": "{{ .CachedString "hostname" }}",
57 "platform_type": "16"
58 }
59 {{- end }}
```

Engineering Resilience

Debugging

- Debugging tooling plays a dual role with staging (load test) and production:
 - Detect issues before they become incidents.
 - Resolve incidents quicker.
- Collect first, ask questions later.
- `fleetctl debug archive`



```
~ fleetctl debug archive --context preview
Ran allocs
Ran block
Ran cmdline
Ran errors
Ran goroutine
Ran heap
Ran mutex
Ran profile
Ran threadcreate
Ran trace
Failed db-locks: get /debug/db/locks received status 500
Failed db-innodb-status: get /debug/db/innodb-status received status 500
Ran db-process-list
Archive written to fleet-profiles-archive-2022-03-01T16:06:16-08:00.tar.gz
~ █
```

<https://fleetdm.com/docs/using-fleet/monitoring-fleet#generate-debug-archive-fleet-3-4-0>

Focus

Consistency

Testing Automation



Debug Tooling.



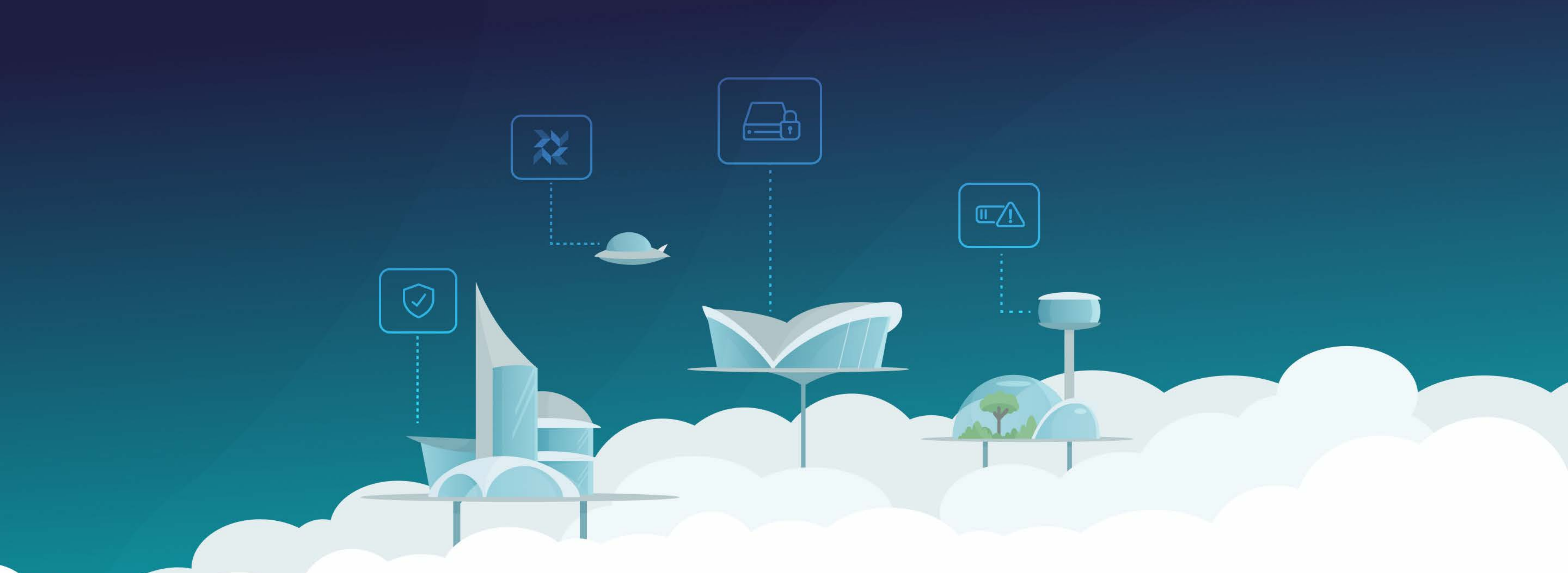
Thank you
zach@fleetdm.com

 [@thezachw](https://twitter.com/thezachw)

 [@zwass](https://www.linkedin.com/company/zwass)

We're hiring:
fleetdm.com/jobs





 **fleet**