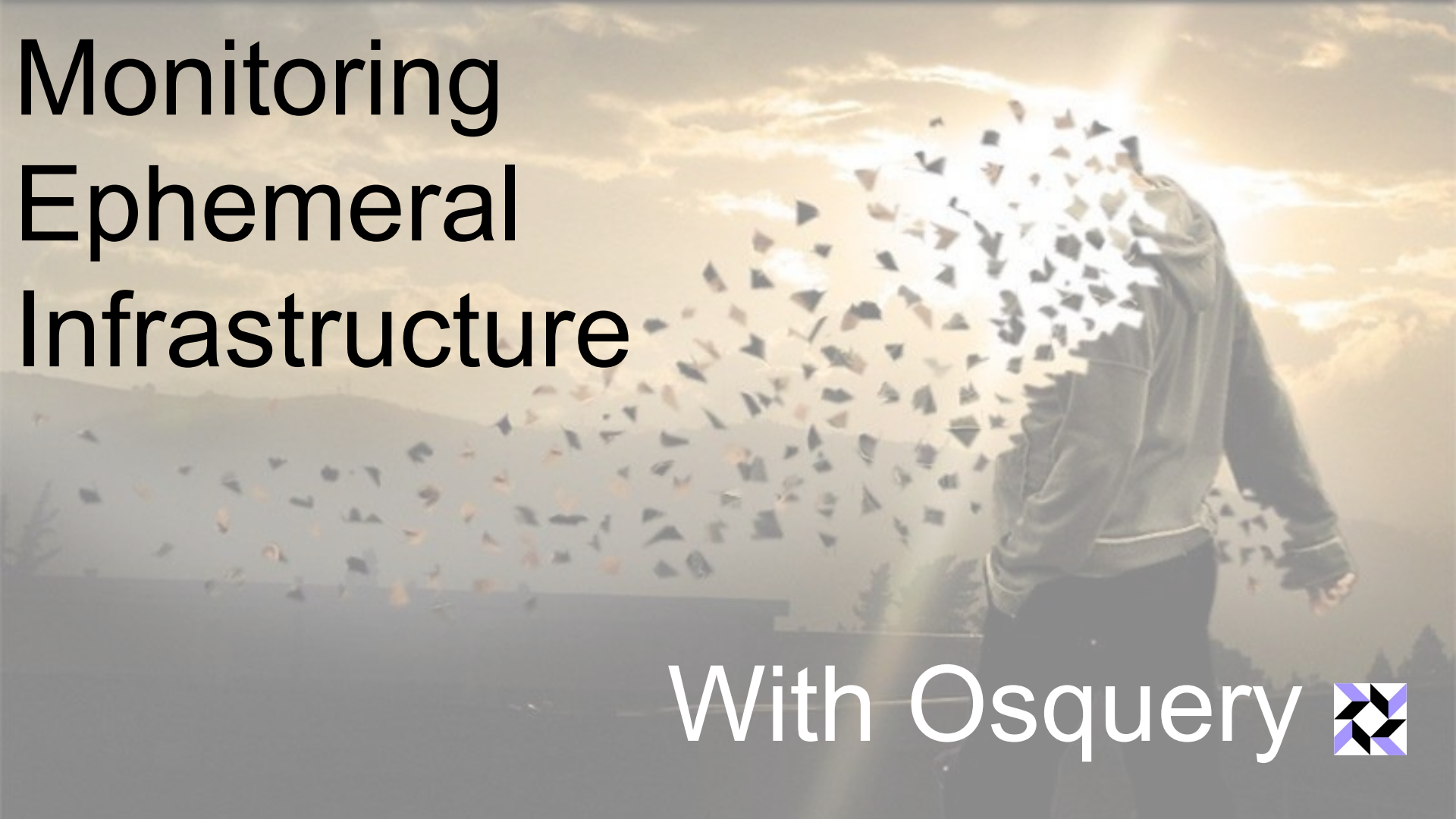
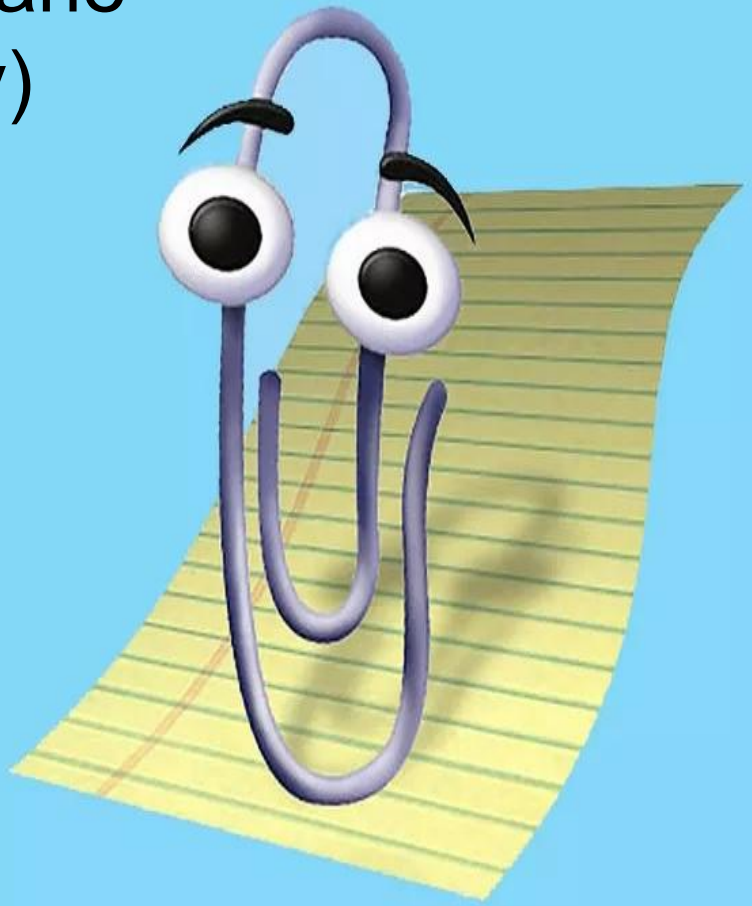
Monitoring Ephemeral Infrastructure



With Osquery

Matt Jane
(clippy)



Builder and Blue Teamer!

Working on…

Cloudsec
Automation
Detection
And… Osquery!

This talk is about

- Monitoring Kubernetes, ECS, Docker and why it’s different
- Deployment and Management
- Methods and strategies for dealing with these challenges

And so we begin

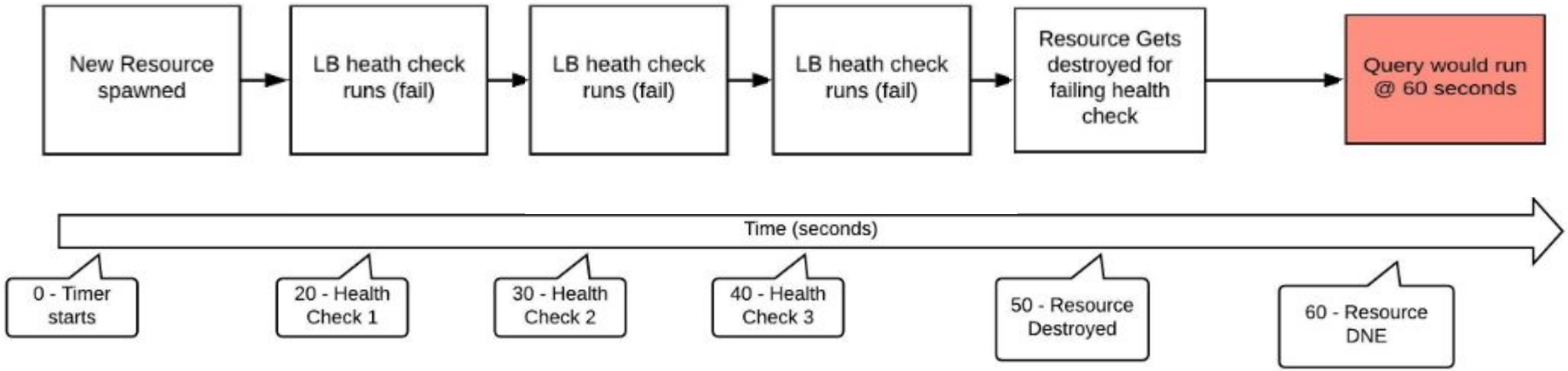
The Challenges


I challenge you to a duel.

At the heart of the issue...

```
{
  "schedule": {
    "installed_pkgs": {
      "query": "SELECT * FROM deb_packages;",
      "interval": 60
    }
  }
}
```

Normally this works pretty well, but…



The query never runs

So… that's not good.

But!

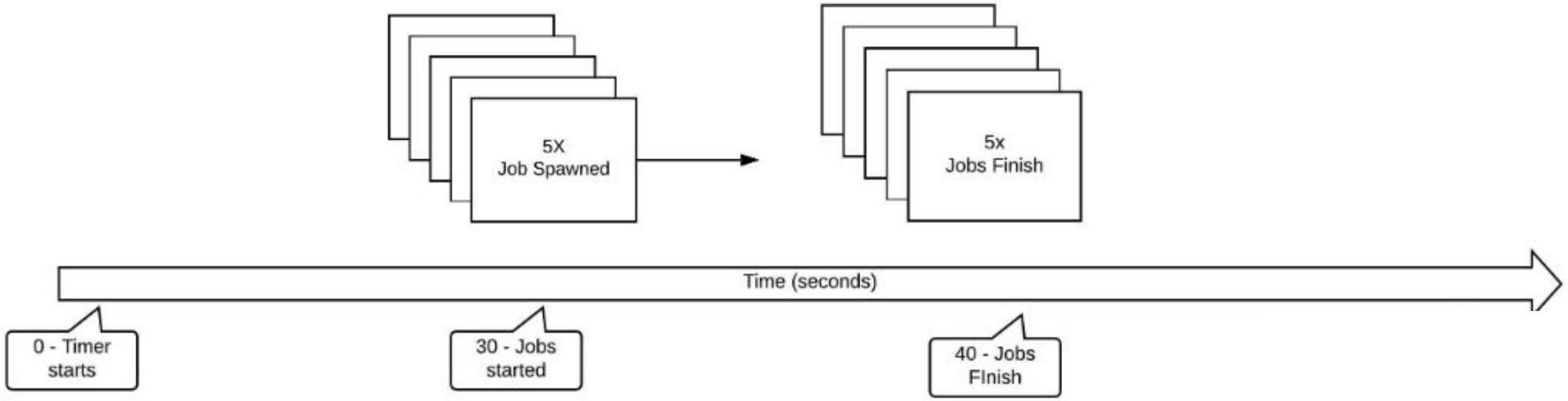
Let's try a Kubernetes Node

```
{
  "schedule": {
    "list_docker_containers": {
      "query": "SELECT * FROM docker_containers;",
      "interval": 60
    }
  }
}
```

With a kubernetes job

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: sleepy
spec:
  parallelism: 5
  template:
    metadata:
      name: sleepy
    spec:
      containers:
      - name: sleepy
        image: registry.gitlab.com/clippy/querycon/sleepy
        command: ["sleep", "10"]
      restartPolicy: OnFailure
```

Kubernetes Batch Job (parallelism 5)



Well....

The query runs, but we miss all the containers we actually wanted to know about.

Our troubles so far

1. Scheduled queries won’t catch everything
2. Even if you shorten your query interval, you still can’t guarantee you’ll catch everything.
3. In environments where things change rapidly, the likelihood that you’ll miss something is greatly increased.

Note: These are contrived examples

Scheduling queries to run every 60 seconds (or less) should be done with caution

It’s generally a good idea to target only data you need

(Don’t use ‘*’ for everything)

What about event tables? Don't they solve this?

Event tables in osquery behave differently than regular tables. They hold a number of events and return the results in a query.

What are events?

Osquery exposes a pubsub framework for aggregating operating system information asynchronously at event time, storing related event details in the osquery backing store, and performing a lookup to report stored rows query time.

Event Tables Vs. Regular Tables

Event Tables: 14

disk_events, file_events, hardware_events, osquery_events, powershell_events, process_events, process_file_events, selinux_events, socket_events, syslog_events, user_events, user_interaction_events, windows_events, yara_events

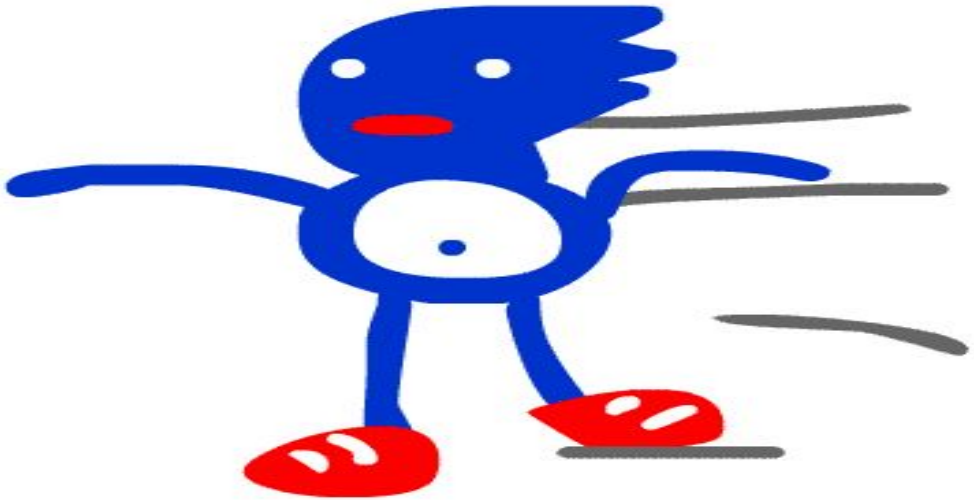
Regular Tables: 215

Disk info, dns resolversj, docker container labels, docker container mounts,docker container networ, docker container ports, docker container processes, docker container stats, docker containers, docker image labels, docker networks, docker version, docker volume labels, docker volumes, drivers, ec2 instance metadata, ec2 instance tags, elf dynamic, elf info .elf sections , etc protocols.etc services. Event taps, example, extended attributes.....

Events are extremely useful, but they don’t capture everything.

There are a lot of non-event tables which are extremely valuable.

So.. Just schedule queries every 5 seconds!!



At which point, whatever you're monitoring probably looks like this…

So maybe that's not the answer.

Queries have an associated cost

Scheduling several queries rapidly that take more than a second or two to run can (and will) rapidly overwhelm your system.

Rapidly scheduling queries can result in a lot of issues.

- Instance OOM
- CPU stuck @ 100%
- Overlapping queries
- Watchdog killing the osquery process repeatedly


BUT...
WHAT AM I SUPPOSE TO DO NOW?
quickmeme.com

And now for the actual informative part of the talk...

Monitoring

Ephemeral

Infrastructure!

Deployment and Management

Deployment and management could probably a talk all by itself

Deployments are very different based on environment.

Clippy's Osquery Law

The resource that doesn't have osquery installed is approximately 1000% more likely to get popped and that you can't see.

So, when deploying...

Get your osquery installation as close to a guarantee as possible

- AMI
- Base Image
- First run of chef/puppet/etc
- Init Scripts

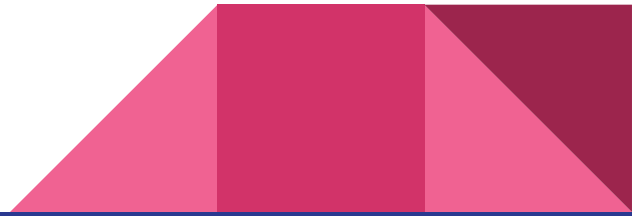
Management:

Opinions

A TLS server is mandatory

You will need to change much of your configuration on the fly. No configuration management in the world will do this fast enough for you.

On-Demand queries are essential. You will need to make heavy use of them to catch as much data as possible during a resource's lifetime.



Capabilities of a Management server

Must Haves:

- The ability to assign a default configuration to a node
- Assignment of configs via enrollment attributes
  - EG: Hostname contains **db-01-ro-xx-xxx-xx.**
    - Assign this the DB configuration

Must haves, cont.

- Auto-Enrollment (no node approval)
- On-Demand Queries
- Enrollment Queries (Not mandatory, but HIGHLY recommended)

The Management Server Must Scale

Just like the rest of your infra, the management server needs to be able to scale (up) and (hopefully) down.

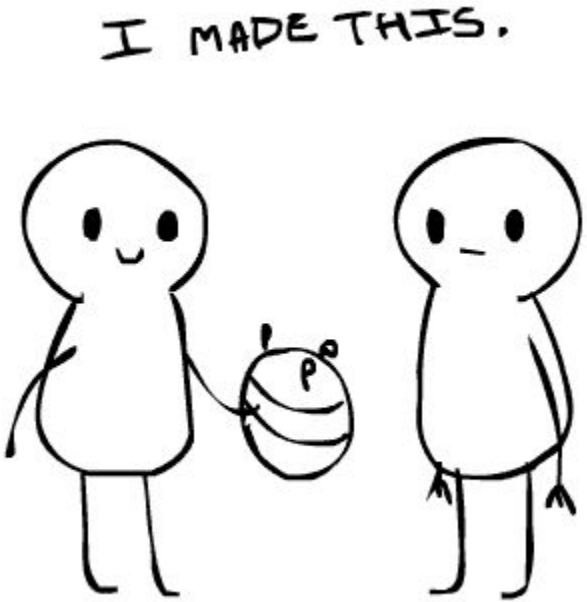
Bursts in infra will result in (likely) equal bursts of need from your management server.

Watch out for DBs and on-demand query queues getting overwhelmed.

Management Database

- Client entries should become stateless-ish
- Use a TTL for entries that have not been updated recently to remove them
- Rapid registration and update intervals can result in a massive hit. DB caching of packs, queries, etc can alleviate an enormous load due to their repetitive nature

Putting it all together



When starting out

- Start simple
- Target data you need that you know
- Plan for your data to scale in and out
  - Bursts of autoscaling activities will generate corresponding spikes in osquery data. Plan accordingly so you don’t lose data
  - Buffering data is better than dropping it on the floor, even if it means slower alerting. At least its still there.

Host Configuration

Do the bare minimum to get your hosts to a point where they can get the rest of your configuration from your management server

BE AWARE!! Some flags can **ONLY** be set in the flag file on the host and NOT by the management server

Use Enrollment Queries

- Enrollment queries are on-demand queries that are queued for a node immediately upon enrollment and run only once
- Enrollment queries ensure that any data you absolutely MUST have about a node is obtained
- This can be either done via eventing or triggered directly on your management server if your management server supports them

Trigger on-demand queries with monitoring events

- Correlate data you want to have with events that create them
- Example: The process of creating a docker container from the `process_events` table can be used to trigger a docker containers query.
- Use a monitoring system/webhook to kick off on-demand queries to that host with the data you want to use

Decorators save lives

- Establish what data you MUST ABSOLUTELY HAVE for a given query to associate it with host, env, etc.
- Be aware of attribute re-use like IP address, friendly container names, etc.
  - Ip address
  - Hostname
  - Device hash
  - Docker-image-labels
  - Image-hash
  - mac-addresses/network interfaces
  - Instance metadata (Tags)

Why are decorators important?

Imagine looking at netflow logs from 3 weeks ago in an environment where hosts frequently re-use IP addresses from a pool.

Now try correlating which resource made an outbound connection from an IP that has been used by 50 different resources.

If you had an IP decorator in your queries, you'd know :)

Understand your Max Time To Alert:

- The time between even generation <-> the time the alert fires in your monitoring system
- Add your on-demand query polling interval to the TTA
- This is the fastest you can **GUARANTEE** that you can execute an on-demand query in response to an event. The average of this time may be significantly less, but don't plan for things to work "optimally".
- Try to reduce your MTTA without blowing up your instances or your management server. Burning heaps of slag tend to be less useful

That’s all I’ve got!

Thanks!

Slides: https://github.com/securityclippy/QueryCon

Twitter: @PansyMcCoward

Email: Securityclippy@securelyinsecure.com

Segment: We’re Hiring!