A declarative and distributed interface for scheduling collection tasks with...

ZABBIX

Parker Green
F5 Networks | Silverline
About us

- F5 Silverline offers Security-aaS
- We proxy traffic through a global proxy network
- Use ANYCAST IPs/Wide-IPs
- Rely on public transit providers
- Know that sooner or later infrastructure fails
- It is in our best interests to be aware of those failures
Deeply invested in supporting our customers
Configuration is constantly in-flight, and changes bring opportunity for failure
Outage inquiries can arrive lacking technical information needed for troubleshooting
Global and HA deployments create divergent behavior
We need to collect network vitals, from an end-user’s perspective
We need to regularly monitoring thousands of entities
Canned solutions?
There are great products, but they become costly at scale. We need a cheap, reliable, scalable collection network to help us get ahead of issues.

10k applications =~ $150,000/year

20*5*10000*.0004 =~ $400/hr or $3,504,000/year

Wants an server-side agent, an area we do not control
Can Zabbix do it?

- We already use it for infrastructure monitoring
- Low-level discovery is declarative
- User-parameters facilitate execution of arbitrary logic
- Orchestration is centralized
- Proxies and agents offload work from the server
- Zabbix is available in containers
- Mechanisms for handling problem creation and alerting
Meet COSMOS

Zabbix meets distributed scheduling
COSMOS

Big Picture

- Zabbix agents monitor apps, not hosts
- LLD to install and remove work
- User-parameters wrap common Unix utilities (curl, etc.)
- User-parameters return data via Zabbix Traps
- User-parameters also generate rich JSON for Elasticsearch
- Redis houses placement state, facts, and buffers documents to elastic

Core node runs a lot of supporting services:

- Zabbix server (orchestration)
- Zabbix GUI
- Logstash (ingestion)
- Curator (retention management)
- Kibana (visualization)

While workers run:

- Many many Zabbix agents (parallelism)
At Scale

COSMOS Regions

- Scale our presence by adding Regions
- Increase capacity by creating more t3.micros
- Self-sufficient
- Collects and stores terabytes of data

COSMOS Aggregator

- Query aggregation point
- Executes complex alerting logic external to Zabbix

Diagram:

- ES
- NGNIX
- Kibana
- Zabbix

Regions:
- US west
- US east
- EU east
- EU central
- APAC southeast
The Collection Loop
Collection loop

Using facts, perform tests

Fact lookup

Factor Cache (Redis)

Collection Script

Output single values

Zabbix Agent

Start collection...

Output rich JSON doc

Redis

Persisted in Zabbix controlled index

Postgres

Elasticsearch

Zabbix

Logstash
What are facts?

COSMOS facts are a place to store variables used for collection. They always have a TTL and are accessed by a “COSMOS Key”.

A “COSMOS Key” looks like this

```
cosmos-v1-4THS:ducks_2891_1900_www.ducky.example.com_
```

Items within Zabbix, just receive the COSMOS Key as input, because we want to:

- Occasionally send more than nine item parameters
- Avoid unnecessary item destroys, due to changing item parameters
- Avoid the Zabbix item key length restriction
A fact entry

KEY = cosmos-v1-4THS:goo_1_443_www.google.com_

```json
{
  "#CUSTOMER_NAME": "Google.com",
  "#CID": "goo",
  "#IDENTIFIER": "Google Homepage",
  "#HTTP_PROTOCOL": "https",
  "#SAFE_HOSTNAME": "www.google.com",
  "#VPORT": "443",
  "#VIP": "172.217.14.228",
  "#HTTP_PATH": "",
  "#HTTP_EXPECTED_CODE": "200",
  "#COSMOS_KEY": "cosmos-v1-4THS:goo_1_443_www.google.com_"
}
```
How does the system take on workloads (collection tasks)?
# Configuration Lifecycle

<table>
<thead>
<tr>
<th>Feed list</th>
<th>Core</th>
<th>Zabbix Server</th>
<th>Zabbix Agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intent</td>
<td>S3</td>
<td>Auto-reg</td>
<td>Election Item</td>
</tr>
<tr>
<td>Fetch</td>
<td>Redis</td>
<td>Templates</td>
<td>Reaper Item</td>
</tr>
<tr>
<td>State</td>
<td>change</td>
<td>Config change</td>
<td>LLD</td>
</tr>
<tr>
<td>change</td>
<td></td>
<td>LLD</td>
<td>User parameters</td>
</tr>
</tbody>
</table>

- **Agent join**
- **Workload Enrollment**
- **Agent config**
- **Collection**
Making of the LLD
Distributing the workload
Feed pull

Receives a feed list contains 9 entities from S3, there is zero state in factor.
Refresh

Core node runs refresh script, we start inserting things into factor

Fact entries get populated, join queues fill

Note 3,4,6 missing for SSL. The refresh script has determined these entities don't utilize SSL.
Election

Workers run election the loop

As a precondition the loop checks the worker’s capacity, causing a NOOP if it is already at capacity.

Else, the loop:
1. Pops a task
2. Checks if entity is already claimed by another worker (discards entity)
3. Assuming it is not...
4. Records in Active Set
5. Records ownership under it’s worker set
## Election

Second iteration of an election loop.

Note that tasks may not be consumed at the same rate by each individual worker.
### Election

All tasks are claimed

<table>
<thead>
<tr>
<th>Fact entries</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cosmos-joining-v1:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-joining-v1:cosw_http</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cosmos-worker-v1:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-worker-v1:cosw1_http</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-worker-v1:cosw2_http</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-worker-v1:cosw3_http</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cosmos-active-v1:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-active-v1:cosw_http</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cosmos-joining-v1:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-joining-v1:cosw_dns</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-joining-v1:cosw1_dns</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-joining-v1:cosw2_dns</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-joining-v1:cosw3_dns</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cosmos-active-v1:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-active-v1:cosw_dns</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cosmos-joining-v1:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-joining-v1:cosw_ssl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-joining-v1:cosw1_ssl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-joining-v1:cosw2_ssl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-joining-v1:cosw3_ssl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cosmos-active-v1:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosmos-active-v1:cosw_ssl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**LLD** is the union of the worker set and the corresponding active set.

For `cosw3_dns` it would be

Removing **6** from the active set would cause `cosw3_dns` LLD to only return **5**
Feed pull (2)

Given the same feed list:

- Join queues are repopulated
- Facts are refreshed
- Fact TTLs are reset (this is important, keeps entities in the system)
Feed pull (2)

Work is redundant so workers throw it away!
Reaping

Fact 4 hasn’t been refreshed in a while, this triggers a reaper which removes associations from the factor.

After reaping Zabbix LLD process will no longer return this entity.

Low TLL!
Reaping...

Once the associations are removed the fact is left to naturally decay, using the native Redis TTL.
Reaping...

The fact has decayed, making room for other work.
Demo Videos
(time allowing)
Simplicity

- Write a wrapper for anything we want
  - very extensible
- It's not a magic black box
  - familiar utilities
- Easy to reproduce collection results
  - (copy & paste a Unix command)
- Compartmentalized fault tolerance
- Distributes the workload in a scalable way
- Easy to move into a Kubernetes environment (hyper scale)
COSMOS Wishlist 😊

1. Parallelism for active item checks
   - Allows for less significantly fewer agents

2. Mechanism for distributing item checks across a collection of Zabbix hosts
   - Replaces 1/3rd of COSMOS

3. Continued Elasticsearch integration and Container development
Thanks!

Questions?
Twitter @hwshadow
Email p.green@f5.com