Invisible OS and Platform Upgrades

Adam McKenna, Site Reliability Engineer, Pinterest
@deathtocss
/in/admckenna
Pronouns: He, his
Our mission

To bring everyone the inspiration to create a life they love.
Core SRE @ Pinterest

- Overall Uptime
- Internal Services
- Tech Debt / “Ownerless” services

- Linux user since 1993, Sysadmin since 1998
- DevOps practitioner since 2012
Defining our problem
Software goes out of date

- **OS/Container Runtimes**
  - Ubuntu
    - 12.04, 14.04 EOL
  - Docker
    - Quarterly Updates
  - Windows
    - Windows 7 EOL (Pinterest not affected)

- **Language Runtimes**
  - Python
    - 2.x to 3.x
  - Java
    - Oracle to OpenJDK
    - Java version updates
  - Go
  - C++
We Must Upgrade!

The business needs:

- Retain support
  - Security bug fixes

- Access to new features
  - New EC2 Instance Families/Generations
  - Hardware support
  - Performance
  - New storage technologies (Instance store -> EBS)

- “Containerization War Stories”
  https://www.usenix.org/conference/srecon18americas/presentation/wong
Developers want

- **Access to new packages / features**
  - Updated runtimes (Go, Python, Java, etc.)
  - GPU Drivers for AI/ML workloads

- **Platform Efficiency**
  - Desire to avoid supporting multiple OS or runtime versions
But we hates upgrading!

- **Complexity**
  - Hundreds of microservices across tens of thousands of hosts

- **Service owners**
  - Don’t like downtime.
  - Migration work is generally not a preferred task

- **Developers**
  - Consistent API/ABI
  - Tests written for specific versions of language runtimes
Solution:
Automated Canary Analysis
What is ‘Canary Analysis’?

- Compare the behavior and performance of two clusters - a canary (test) cluster and a control cluster.
- Has inherent risk, but lower risk than full rollout of bad code.

Canary Analysis

Requirements and Goals

● Automate and normalize the most mundane migration tasks
  ○ Depend on successful integration and acceptance testing
  ○ Automate looking at charts, comparing metrics
  ○ Normalize - Process can be applied to many similar systems.

● Make migrations ‘invisible’ to service owners
  ○ Address reliability concerns; Services are analyzed at the cluster level.
  ○ Can be run by an operator, freeing up developer time
  ○ No more ‘finger crossing’
Typical migration ‘toil’ pattern

1. New Image is released
2. Service owner updates Image ID in deploy system
3. Operator triggers rolling cluster upgrade
4. Humans watch charts and cross fingers
5. Did things go OK? Good!
6. Did things not go OK? Bad! Rollback! Outage?
“Looks good to me!”

Can you make a Go / No-Go decision based on these two charts?
How about these?
Well, Let’s ask the service owners!
The human factor
What might these Service Owners think?

Gene
- Extremely Risk Averse
- Hates change
- His service has 99.999999% uptime, but runs on Ubuntu 10.04
Gene’s call

“No way. Look at that I/O line! Let’s meet on Monday and figure out what’s going on.”
Angus

- Last job was at a Bitcoin startup
- Likes to “Move fast and break stuff”
- Yells “TONIGHT WE TEST IN PROD” at least once a day
Angus’s call
“LGTM. SHIP IT!”
Carol

- Has 100% test coverage on all her code
- Is researching statistical models to improve fault detection
- Analysis is her middle name
Carol’s call

“I’ll need a couple hours to run some additional tests against the new hosts. Stand by.”

CPU, Memory and I/O - Control

CPU, Memory and I/O - Canary
Karen

- The employee who left the company
- All of the knowledge they didn’t write down about their service left with them!
A better way - Automate!

Benefits of automated Canary Analysis

- A software-defined, repeatable process for validation of new code
- Accumulated knowledge remains in the system as code, rather than leaving with employees
- Can be generalized/normalized with sane defaults
- (Optionally) continue to provide manual go/no-go approval (You know, for Gene)
- Bad results? Update the model! (In source control!)
How it works
Components

- **Spinnaker + Kayenta**
  - Workflow orchestration
  - [https://www.spinnaker.io/](https://www.spinnaker.io/)

- **Docker**
  - Execution environment
  - [https://www.docker.io](https://www.docker.io)

- **Jenkins**
  - Script execution
  - [https://jenkins.io/](https://jenkins.io/)

- **Python**
  - Script logic implementation
  - [https://www.python.org](https://www.python.org)

- **Teletraan**
  - Pinterest’s deploy system
  - [https://github.com/pinterest/teletraan](https://github.com/pinterest/teletraan)
Simplified Process Flow

Create Canary and Control Clusters

Build fresh (1/n) instances per cluster

Clusters at desired size?

Yes

Manual Go / No Go Decision

No

Service Health Check

Convergence Period

Canary Analysis
Configuring the Pipeline

Canary Analysis Configuration

Analysis Config

Analysis Type

- Real Time (Manual)
- Retrospective

Config Name
control-canary-devapptool

Lifetime
0 hours 20 minutes

Delay
0 minutes before starting analysis

Interval

Step
1 seconds

Lookback Type
Growing
# Configuring Scoring

## Metric Scope

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_scope_key</td>
<td>host_type</td>
</tr>
</tbody>
</table>

### Add Field

## Scoring Thresholds

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Marginal</strong></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td><strong>Pass</strong></td>
<td>75</td>
<td></td>
</tr>
</tbody>
</table>
# Execution Options

<table>
<thead>
<tr>
<th><strong>If stage fails</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ halt the entire pipeline</td>
</tr>
<tr>
<td>☐ halt this branch of the pipeline</td>
</tr>
<tr>
<td>☐ halt this branch and fail the pipeline once other branches complete</td>
</tr>
<tr>
<td>☐ ignore the failure</td>
</tr>
</tbody>
</table>

- ☐ Restrict execution to specific time windows
- ☐ Fail stage on failed expressions
- ☐ Conditional on Expression

## Notifications

- ☐ Send notifications for this stage
# Adding a Metric

## Configure Metric

<table>
<thead>
<tr>
<th><strong>Group</strong></th>
<th>System Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>CPU Total Time</td>
</tr>
<tr>
<td><strong>Fail on</strong></td>
<td>![Select Either] Increase Decrease Either</td>
</tr>
<tr>
<td><strong>Criticality</strong></td>
<td>Fail the canary if this metric fails</td>
</tr>
<tr>
<td><strong>NaN Strategy</strong></td>
<td>![Select Remove] Default (remove) Replace with zero Remove</td>
</tr>
<tr>
<td><strong>Scope Name</strong></td>
<td>default</td>
</tr>
<tr>
<td><strong>OpenTSDB Metric</strong></td>
<td>tc.proc.stat.cpu.total.*</td>
</tr>
<tr>
<td><strong>Aggregator</strong></td>
<td>sum:rate</td>
</tr>
<tr>
<td><strong>Downsample</strong></td>
<td>1m-sum</td>
</tr>
</tbody>
</table>
# Metrics Groups

**SYSTEM METRICS**

<table>
<thead>
<tr>
<th>METRIC NAME</th>
<th>GROUPS</th>
<th>Edit</th>
<th>Move Group</th>
<th>Copy</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Total Time</td>
<td>System Metrics</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Used</td>
<td>System Metrics</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disk IO</td>
<td>System Metrics</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Add Group**

**Add Metric**
Metric Thresholds / Weights

Thresholds

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marginal</td>
<td>50</td>
</tr>
<tr>
<td>Pass</td>
<td>75</td>
</tr>
</tbody>
</table>

Judge

NetflixACAJudge-v1.0

Metric Group Weights

- System Metrics: 50
- Service Framework Metrics: 50
Canary Output

FAIL

33.33

SCOPES
devapp-srecon-control

LOCATION

canary-devapptool

METRIC NAME | DEVIATION | RESULT
--- | --- | ---
CPU Total Time | +136.3% | High
Disk IO | +2789.0% | Pass
Memory Used | +9.7% | High
## Canary Output

### System Metrics

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Deviation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Total Time</td>
<td>+136.3%</td>
<td>High</td>
</tr>
<tr>
<td>Disk IO</td>
<td>+2789.0%</td>
<td>Pass</td>
</tr>
<tr>
<td>Memory Used</td>
<td>+9.7%</td>
<td>High</td>
</tr>
</tbody>
</table>

### Diagram

The diagram illustrates the system metrics over a period, with two lines representing baseline and canary performance. The chart shows a comparison of performance metrics such as CPU, Memory, and Disk usage over time from 11:49 pm to 12:07 am.
Sizing your clusters

<table>
<thead>
<tr>
<th></th>
<th>Typical</th>
<th>Good</th>
<th>Bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>5%</td>
<td>95%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10%</td>
<td>80%</td>
<td>10%</td>
<td>98%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1%</td>
<td></td>
</tr>
</tbody>
</table>

The table shows the distribution of successful and unsuccessful outcomes in different scenarios.
Overall Effort

- Originally planned as a six-month effort for 2-3 FTE
- Actual effort was ~1 year due to various delays
- Cycle.py - ~700 lines of python
- Kayenta - had to write our own OpenTSDB plugin
Future Plans

• Integration of additional Machine Learning to Canary process
• Direct integration into our build system
• Additional tooling to improve user experience
Additional Resources

- Spinnaker Canary Analysis doc: https://www.spinnaker.io/setup/canary/
We’re hiring! Come work with us!

hiring-srecon@pinterest.com
Questions?