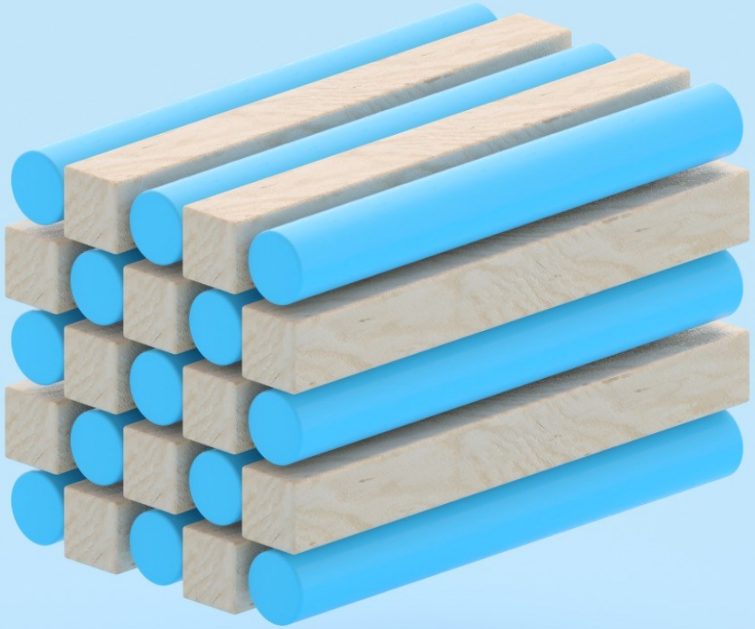
Site Reliability Engineering

Applied SRE using time series data to implement monitoring, alerting, and observability

Dustin Sorge and Aaron McCartney
NetApp BAERO SRE Team



Agenda

- About NetApp
  - BAERO Team
- About us
- What is SRE?
- Life Before Time Series Data (InfluxDB)
- Example NetApp Services
- Key Metrics for SRE
  - SLOs, SLIs, Error Budgets, MTTA, MTTR, MTBF
  - Visualization via custom dashboards
- Black Swan Events
- Measuring Key Metrics with InfluxDB
  - Service outage measurements
    - Strategic tags and fields
  - Real-time SLI calculation example with TICKscript
- Coordinated Incident Response
  - Follow the Sun
  - Slack alerting via Kapacitor
- Measuring System Resources
- Questions?

NetApp at a glance

Industry-leading cloud services

Cloud storage

Compute operations

Cloud controls

01 10 Cloud services and analytics

Industry-leading software and systems

Flash and hybrid storage

Object storage

Converged and hybrid cloud infrastructure

Protection and security

Enterprise solutions

Industry-leading solutions with an open ecosystem of partners

Microsoft Azure | Google Cloud | aws | IBM Cloud | vmware PARTNER TECHNOLOGY ALLIANCE

cisco | Lenovo | ORACLE | CITRIX | SAP Global Partner

redhat | BROCADE | accenture | Cognizant | FUJITSU

NVIDIA | COMMVAULT | veeAM | rubrik | splunk>

- **Global** cloud-led, data-centric software company
- **Industry-leading** software, systems, and cloud services
- **Founded in 1992**, headquartered in San Jose, California
- **$5.74B** FY21 revenue
- **11,000+ employees** and **4,000+ partners** helping organizations thrive in a hybrid multi-cloud world
- **Fortune 500 company** (NASDAQ: NTAP)

NetApp’s BAERO Team

Build, **A**utomation, and **E**ngineе**R**ing **O**perations

- Responsible for NetApp’s Build, Test, and Automation Infrastructure
- Driven by several critical services that support NetApp ONTAP Engineering’s development and quality
    - Data ONTAP Build Farm
    - Common Test Lab Environment
    - Continuous Integration Testing Environment
    - AI-driven Code Pre-submission Testing

About us

Dustin Sorge

- Reside in Pittsburgh, PA
- Site Reliability Engineering Technical Lead
- Been with NetApp since 2011
- Graduate of both University of Pittsburgh and Carnegie Mellon University
- Formerly High Performance Computing Operations Engineer at the Pittsburgh Supercomputing Center

Aaron McCartney

- Reside in Pittsburgh, PA
- Site Reliability Engineering
- Been with NetApp since 2013
- Graduate of both Robert Morris Univ. and Carnegie Mellon University
- Formerly Windows Sys Admin at CMU School of Computer Science

What is SRE?

- Site Reliability Engineering

- SRE is a role whose primary focus is on production engineering.

- SREs blend the skillsets of software engineering and operations to deliver and maintain highly available services.

- SRE is an implementation of the DevOps philosophy.
  - SRE is a role, DevOps is a way of doing things
  - SRE is narrowly focused on **service uptime**

- “Automate yourself out of a job”

What is SRE?

The Balloon Game!

Tenets of SRE

SRE is typically responsible for

- Availability
  - SLOs (Service Level Objectives), Error Budgets
- Latency
  - SLIs (Service Level Indicators)
- Performance
- Change Management
  - Ex: Weekly software release
- Monitoring/Alerting/Observability
  - TICK stack
- Emergency Response
  - On-call duties
- Capacity Planning

Monitoring

Valid Monitoring Output

- Alerts
  - A human should take action immediately in response to something that is happening or is about to happen.
- Tickets (JIRAs)
  - A human should take action, but not immediately. The system cannot automatically handle the situation.
- Logging
  - No one needs to look at this information, but it’s recorded for diagnostic or forensic purposes.

Life Before InfluxDB

Custom Metrics

- Small number of metrics DBs stored in relational databases.

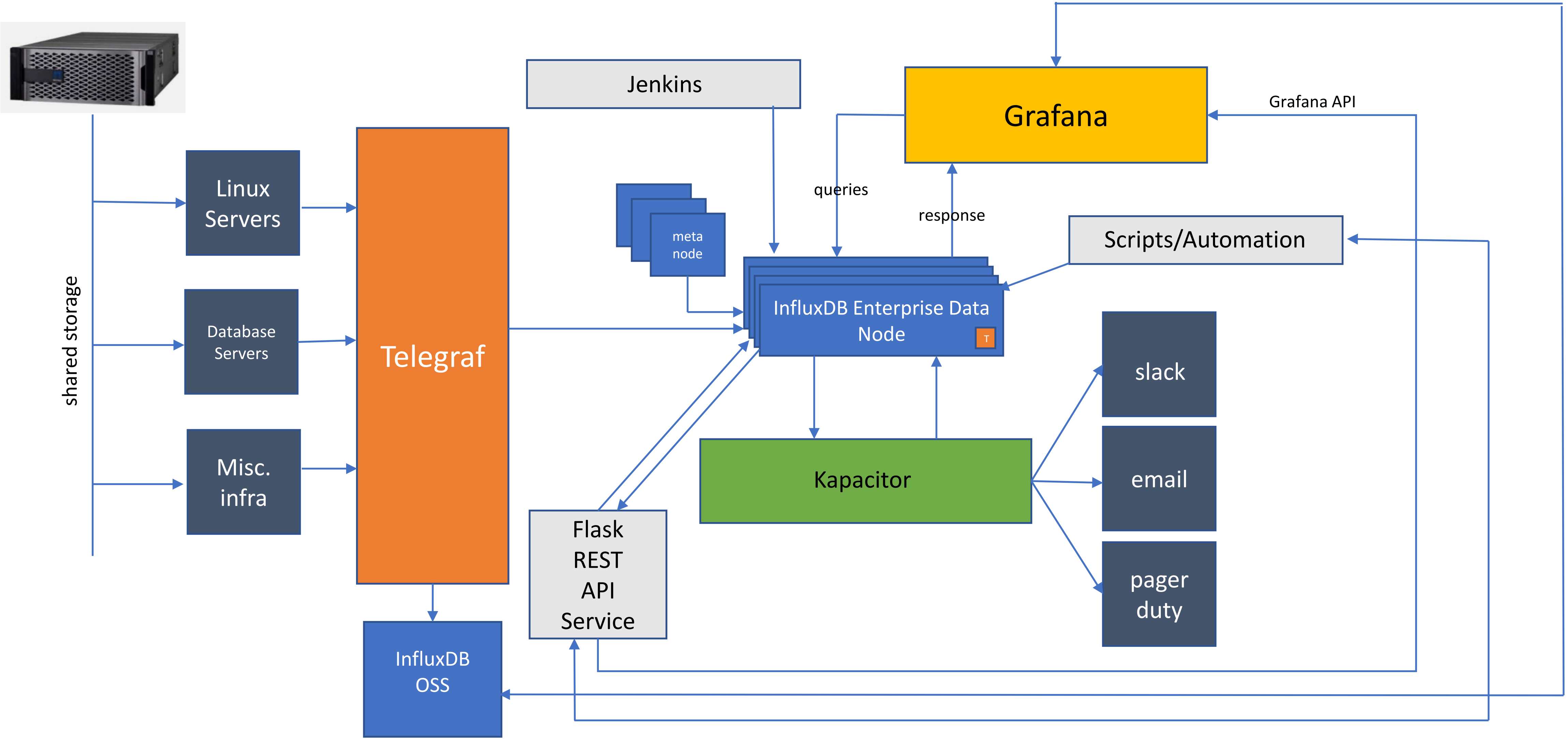
System Resource Monitoring

- Reliance on traditional Linux observability tools
  - top/htop/atop
  - mpstat
  - iostat
  - sar

Alerting

- Nagios

Architecture



Example NetApp Services

NetApp CTL Environment

NetApp's Common Test Lab (CTL) consists of thousands of nodes (physical and virtual) and thousands of client VMs to support NetApp Developer's and QA Engineer's testing requirements.

Common Workflows

- Execution
  - Submit a single test for a defined compute configuration and obtain the results via email immediately following the test.
- Reservation
  - Obtain compute and clients for a predefined period for unlimited testing in the reservation window.

~32K Compute Nodes

~100,000 hours of ONTAP testing per month

~90 unique hardware configs

~35K Client VMs

NetApp CIT Environment

NetApp’s Continuous Integration Test (CIT) Environment is crucial for protecting our code line stability.

Provide bisect and revert functionality to ensure problematic code submissions stay out of the code line.

Bisect

- Run integration tests across narrowing range of changes to narrow down the problematic change.

Revert

- Undo changes to code that caused integration test failures.

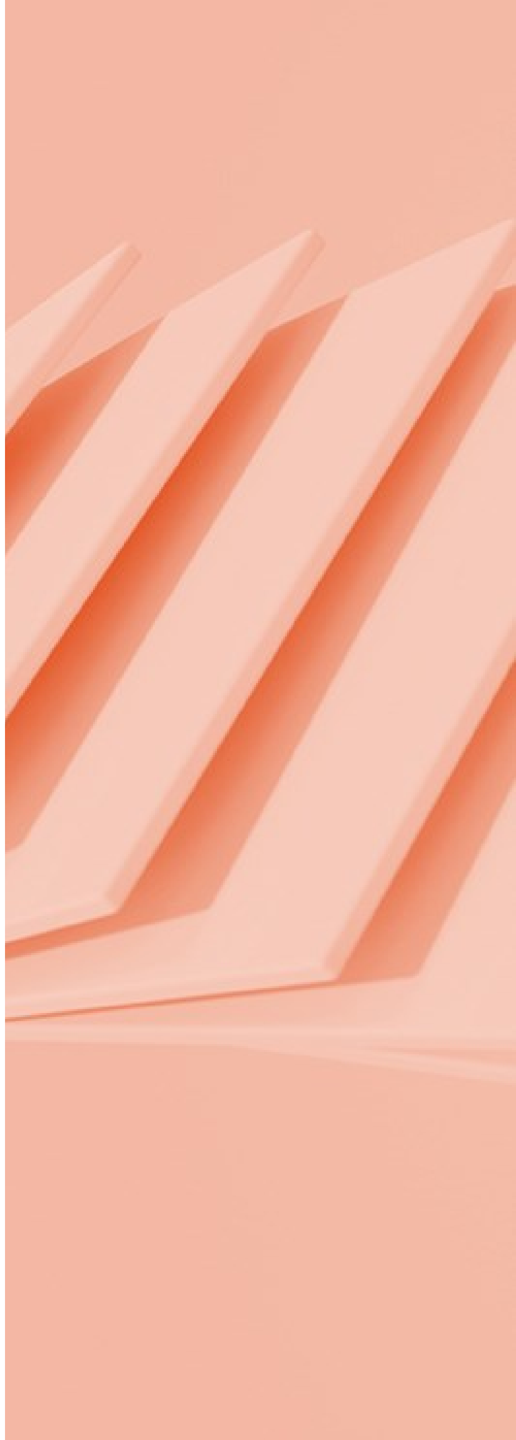
~440,000 hours of ONTAP testing per month

103 reverts per month

3.9K test and ONTAP submissions per month

~11K CIT tests running on 95% virtualized HW

Key Metrics for SRE



SLI and SLO

SLI

Service Level Indicator

- Gives us a way to quantify the level of service being provided.
- Moving targets
  - Can adjust as service improves
- Displayed as a percentage
  - (Good events / Expected events)*100

SLO

Service Level Objective

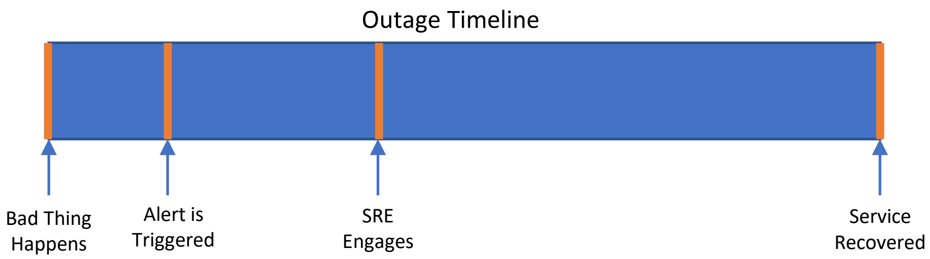
- Sets the target level of availability for a given service.
- Should not have a goal SLO of 100%.
- Meeting and/or exceeding SLO should result in happy consumers of your service.

Error Budgets

- Error budget = 100% - SLO
- Provides a clear, objective metric that states how “unreliable” a service is allowed to be.
  - Outages from both infrastructure and unstable features will be taken out of the error budget.

- Allows development teams and SRE to balance innovation and reliability.
- Larger error budgets allow development to take more risks.
- Smaller error budgets mean development should be more conservative.
- Should be constantly measured and displayed.

MTTA, MTTR, and MTBF



MTTA	MTTR	MTBF
Mean Time to Alert - The average amount of time it takes to know that there is a problem.	**Mean Time to Recovery** - The average amount of time it takes to recover the service. **Mean Time to Respond** - The average amount of time for SRE to engage.	**Mean Time Between Failure** - The average amount of time between disruptions for a given service.

Black Swan Events

An unpredictable event that has severe consequences.

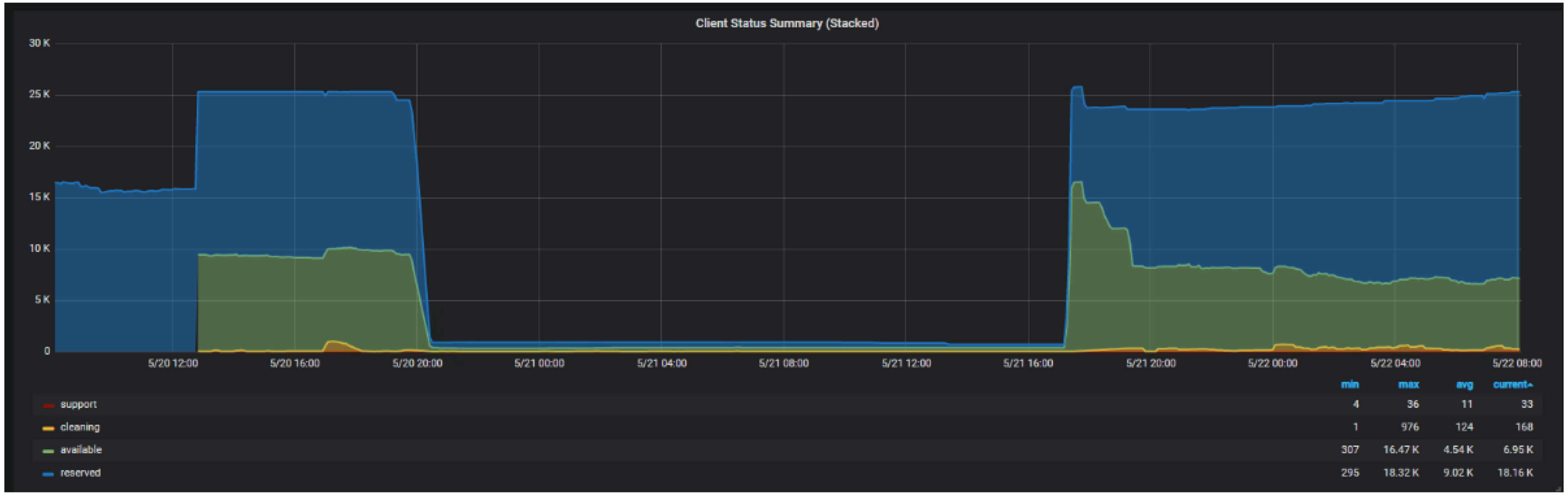
- Black swan events cause large scale outages
- Require lots of effort to recover from
- Very valuable learning experiences
  - Gain deep internal knowledge of your systems
  - automation JIRAs that strengthen your service.
  - process change

CTL Black Swan Case Study

CTL Outage on 05/21/2019

What happened?

- All of the SCS clients in the CTL (25K!) were returned at once.



	min	max	avg	current
🟥 support	4	36	11	33
🟨 cleaning	1	976	124	168
🟩 available	307	16.47 K	4.54 K	6.95 K
🟦 reserved	295	18.32 K	9.02 K	18.16 K

CTL Black Swan Case Study

What did SRE do?

- Contact SCS and have them pause their queues so all the VMs wouldn’t be deleted.
- Got lucky: only 7.5% of the clients we requested to be deleted actually were.
  - Would have possibly taken **WEEKS** to regenerate
- Worked with SCS to get their DB and the CTL Production DB homogenous
- ~500 affected reservations were swapped with new SCS clients.
  - Automation was written to do this

CTL Black Swan Case Study

What caused this to happen?

- During the SRE postmortem investigation process it was discovered that an engineer had loaded a copy of the production CTL database onto their development instance.
- An attempt was made to sanitize the data before testing but some critical tables were left behind which ended up triggering a mass delete.

Will it happen again?

- Process was implemented and documentation was introduced to educate engineers on the proper procedures for testing with production data.
- JIRAs filed with SRE and ECO to enhance visibility and monitoring of this issue in the future.

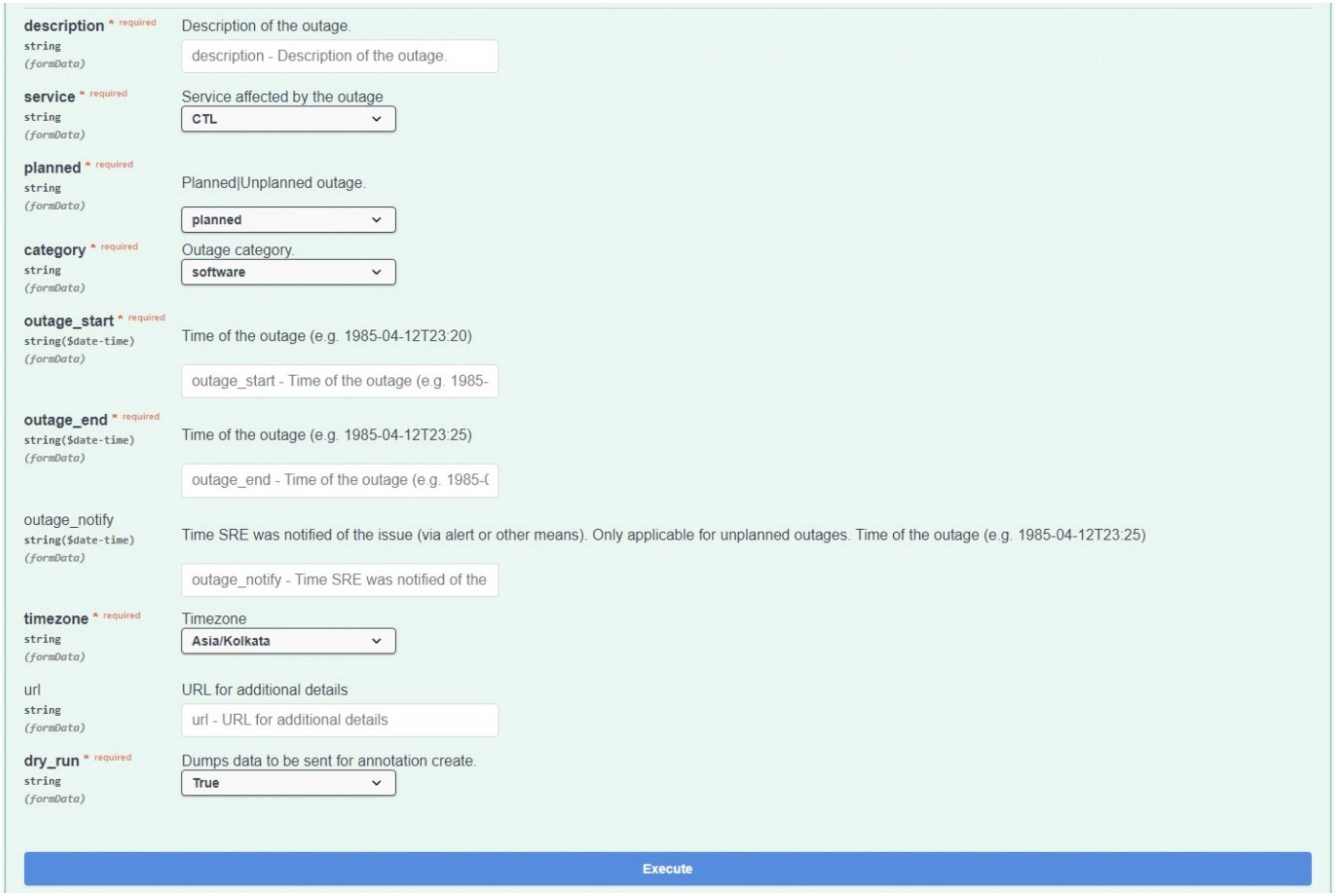
Measuring Key Metrics with InfluxDB

Outage Measurement

- Service specific outage information is stored in an outage measurement
- Record is populated by SRE REST API Service following incident Postmortem Analysis.
- Custom scripts read ctl_outage measurement and generate current SLO, MTTA, MTTR, and MTBF metrics to be displayed by SRE metrics dashboard in Grafana.

- Example: CTL Service Outage

- Database: ctl
- Measurement: ctl_outage
- Tags:
  - site
  - service
  - planned
  - category
- Fields:
  - description
  - outage_start (epoch seconds)
  - outage_end (epoch seconds)
  - outage_notify (epoch seconds)



description * required
string
(formData)
Description of the outage.
description - Description of the outage.

service * required
string
(formData)
Service affected by the outage
CTL

planned * required
string
(formData)
Planned|Unplanned outage.
planned

category * required
string
(formData)
Outage category.
software

outage_start * required
string($date-time)
(formData)
Time of the outage (e.g. 1985-04-12T23:20)
outage_start - Time of the outage (e.g. 1985-

outage_end * required
string($date-time)
(formData)
Time of the outage (e.g. 1985-04-12T23:25)
outage_end - Time of the outage (e.g. 1985-(

outage_notify
string($date-time)
(formData)
Time SRE was notified of the issue (via alert or other means). Only applicable for unplanned outages. Time of the outage (e.g. 1985-04-12T23:25)
outage_notify - Time SRE was notified of the

timezone * required
string
(formData)
Timezone
Asia/Kolkata

url
string
(formData)
URL for additional details
url - URL for additional details

dry_run * required
string
(formData)
Dumps data to be sent for annotation create.
True

Example SLI: reservation_add() API latency

api.log

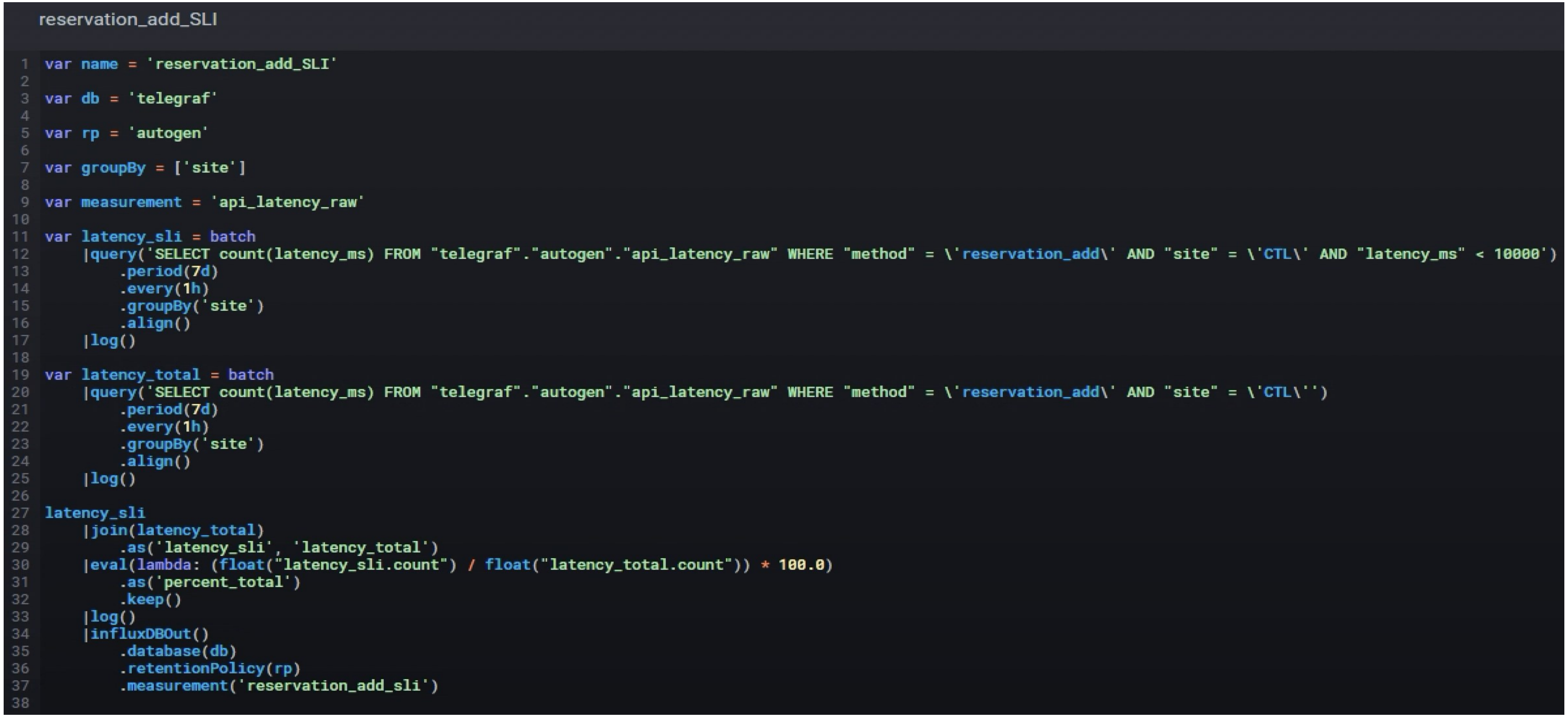
```
2022/07/11 03:16:31 api::Controller::Default::jsonrpc INFO: {"remote_ip_source":"X-Forwarded-
For","api_user":”<sanitized>","remote_ip":”<sanitized>","api_caller":"gui","params":{"priority":7,"testbed_attributes":"","testbed_type":"1
Vsim","project":"Wafl","bad_model_combo_check":1,"init":"config/1Vsim.nsha.sdot","secondary_owner":"","build_variant":"x86_64.sim","project
_type":"other","pool":"common","project_reason":"WRR CP Trigger
test","owner":”<sanitized>","build_promoted":0,"build_root":”<sanitized>","duration":14,"build_resolved":"later","lab":"common","clients":"
","label":""},"site":"CTL","latency_ms":2824,"request_id":”<sanitized>","method":"reservation_add"} [YzahrwVy7-f2] (ea1af4cc)
```

telegraf.conf

```
[[inputs.logparser]]
  files = [”<path_to_api_log>"]
  watch_method = "poll"
  from_beginning = false

  [inputs.logparser.tags]
    service = "api"

  [inputs.logparser.grok]
    timezone = "America/New_York"
    measurement = "api_latency_raw"
    patterns = [
      '%{TS_UNIX:timestamp:ts-"2006/01/02 15:04:05"} api::Controller::Default::jsonrpc %{LEVEL:level:tag}: %{GREEDYDATA:api_json} %{REQUESTID:instance:drop}
%{SERVICEID:instance:drop}',
      '%{GREEDYDATA:garbage:drop}'
    ]
    custom_patterns = '''
TS_UNIX %{YEAR}/%{MONTHNUM}/%{MONTHDAY} %{HOUR}:%{MINUTE}:%{SECOND}
LEVEL (\bINFO\b)
REQUESTID (\[.*\])
SERVICEID (\(.*\))
'''

[[processors.parser]]
  merge = "override"
  parse_fields = ["api_json"]
  data_format = "json"
  tag_keys = ["api_user", "method", "site", "api_caller"]
  json_string_fields = ["params", "latency_ms"]
```

Example SLI: reservation_add() API latency cont.

Calculate and Publish SLI Value in Realtime

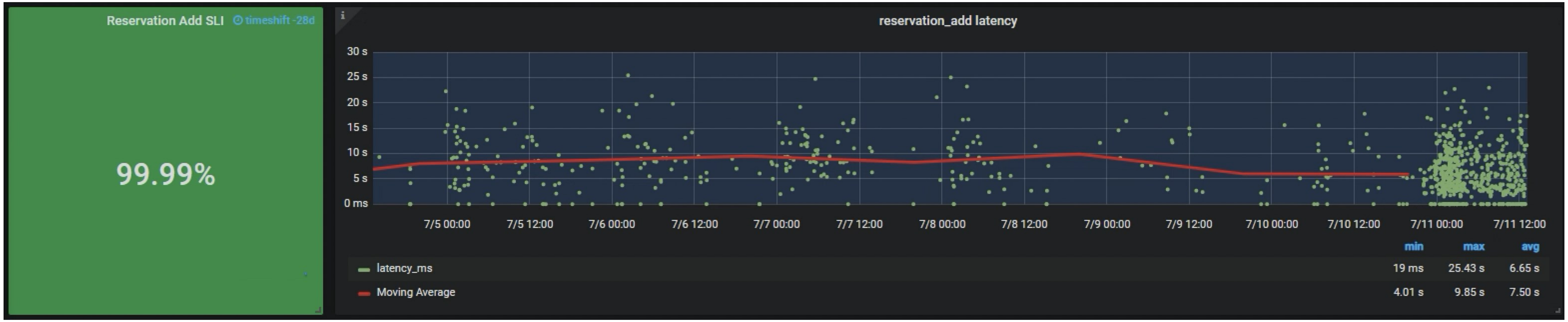


reservation_add_SLI

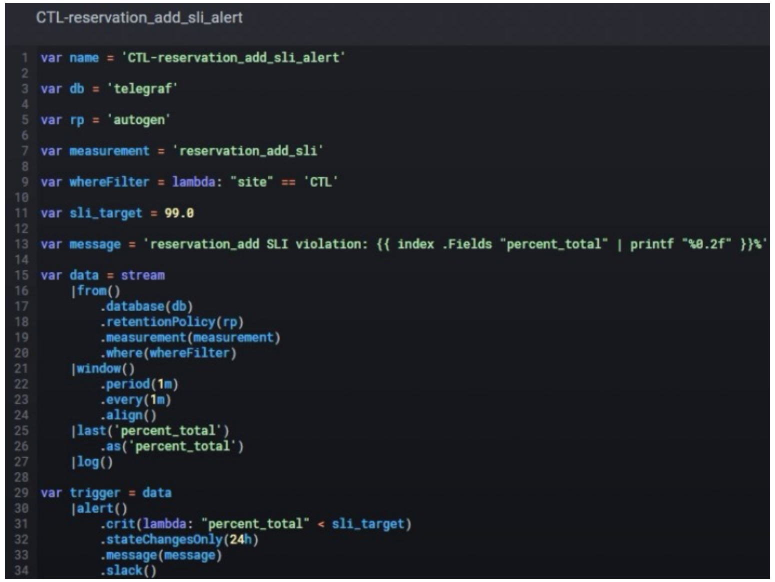
```
var name = 'reservation_add_SLI'

var db = 'telegraf'

var rp = 'autogen'

var groupBy = ['site']

var measurement = 'api_latency_raw'

var latency_sli = batch
    |query('SELECT count(latency_ms) FROM "telegraf"."autogen"."api_latency_raw" WHERE "method" = \'reservation_add\' AND "site" = \'CTL\' AND "latency_ms" < 10000')
        .period(7d)
        .every(1h)
        .groupBy('site')
        .align()
    |log()

var latency_total = batch
    |query('SELECT count(latency_ms) FROM "telegraf"."autogen"."api_latency_raw" WHERE "method" = \'reservation_add\' AND "site" = \'CTL\'')
        .period(7d)
        .every(1h)
        .groupBy('site')
        .align()
    |log()

latency_sli
    |join(latency_total)
        .as('latency_sli', 'latency_total')
    |eval(lambda: (float("latency_sli.count") / float("latency_total.count")) * 100.0)
        .as('percent_total')
        .keep()
    |log()
    |influxDBOut()
        .database(db)
        .retentionPolicy(rp)
        .measurement('reservation_add_sli')
```

Example SLI: reservation_add() API latency cont.

Display on Grafana Dashboard



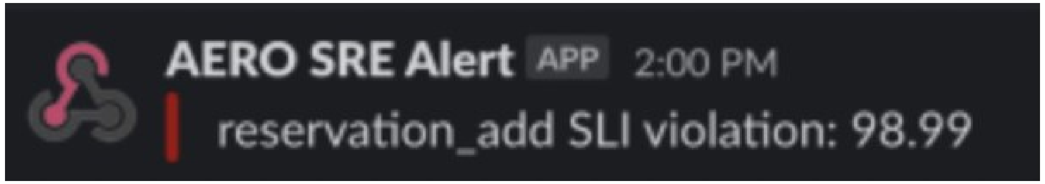
	min	max	avg
🟩 latency_ms	19 ms	25.43 s	6.65 s
🟥 Moving Average	4.01 s	9.85 s	7.50 s



CTL-reservation_add_sli_alert

```
var name = 'CTL-reservation_add_sli_alert'

var db = 'telegraf'

var rp = 'autogen'

var measurement = 'reservation_add_sli'

var whereFilter = lambda: "site" == 'CTL'

var sli_target = 99.0

var message = 'reservation_add SLI violation: {{ index .Fields "percent_total" | printf "%0.2f" }}%'

var data = stream
    |from()
        .database(db)
        .retentionPolicy(rp)
        .measurement(measurement)
        .where(whereFilter)
    |window()
        .period(1m)
        .every(1m)
        .align()
    |last('percent_total')
        .as('percent_total')
    |log()

var trigger = data
    |alert()
        .crit(lambda: "percent_total" < sli_target)
        .stateChangesOnly(24h)
        .message(message)
        .slack()
```

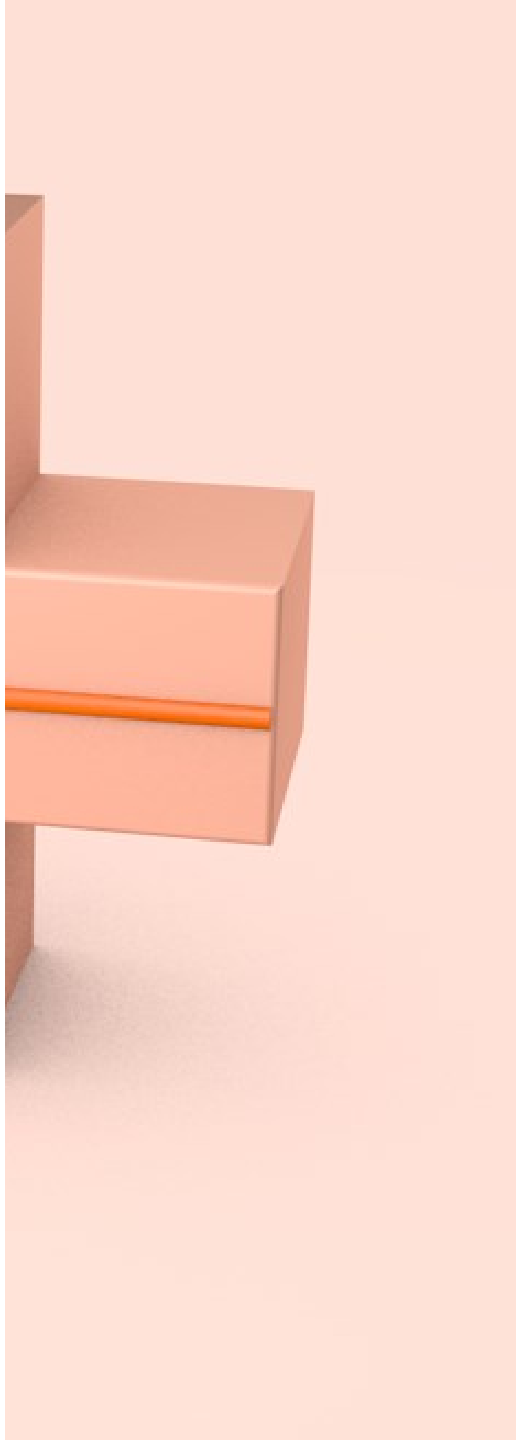
Alert to Slack if Necessary



AERO SRE Alert APP 2:00 PM

reservation_add SLI violation: 98.99

Coordinated Incident Response



Follow The Sun

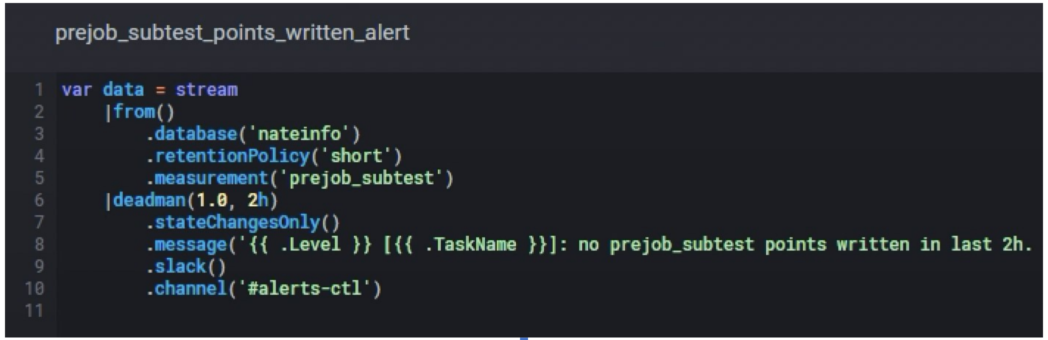
- SRE is best implemented with a “follow the sun” model.
- Geographic distribution allows for more complete coverage of services.
- Reduces off-hour pages

Slack Alerting with Kapacitor

- Custom alerts created with TICKscripts are routed to service-specific alerting channels in Slack.
- Allows for immediate response and global collaboration in a single location.
  - SREs from US and India able to triage issues in real time which minimizes service downtime.

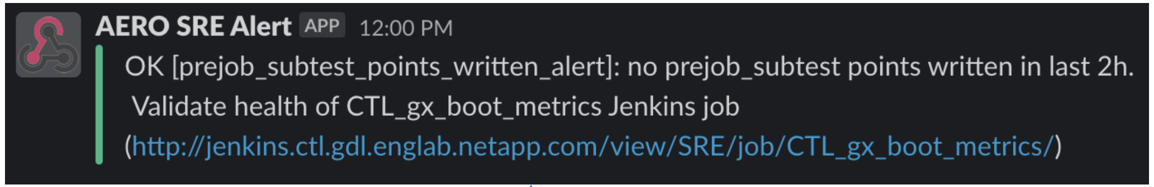
Slack Alerting with Kapacitor cont.

Ex: Using deadman alert to detect potential issues



prejob_subtest_points_written_alert

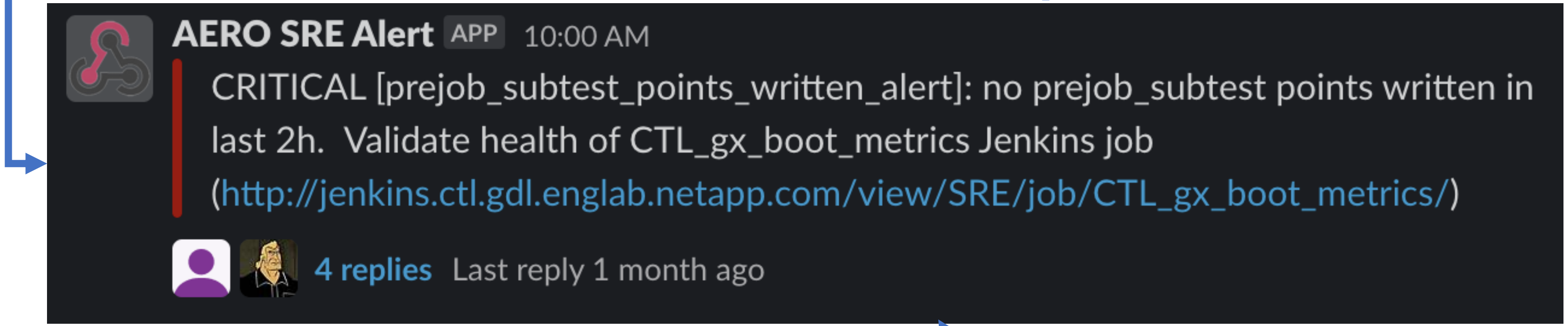
```
var data = stream
    |from()
        .database('nateinfo')
        .retentionPolicy('short')
        .measurement('prejob_subtest')
    |deadman(1.0, 2h)
        .stateChangesOnly()
        .message('{{ .Level }} [{{ .TaskName }}]: no prejob_subtest points written in last 2h.
        .slack()
        .channel('#alerts-ctl')
```



AERO SRE Alert APP 12:00 PM

OK [prejob_subtest_points_written_alert]: no prejob_subtest points written in last 2h. Validate health of CTL_gx_boot_metrics Jenkins job (http://jenkins.ctl.gdl.englab.netapp.com/view/SRE/job/CTL_gx_boot_metrics/)

Back to normal!



AERO SRE Alert APP 10:00 AM

CRITICAL [prejob_subtest_points_written_alert]: no prejob_subtest points written in last 2h. Validate health of CTL_gx_boot_metrics Jenkins job (http://jenkins.ctl.gdl.englab.netapp.com/view/SRE/job/CTL_gx_boot_metrics/)

4 replies Last reply 1 month ago

Issue is triaged in channel thread

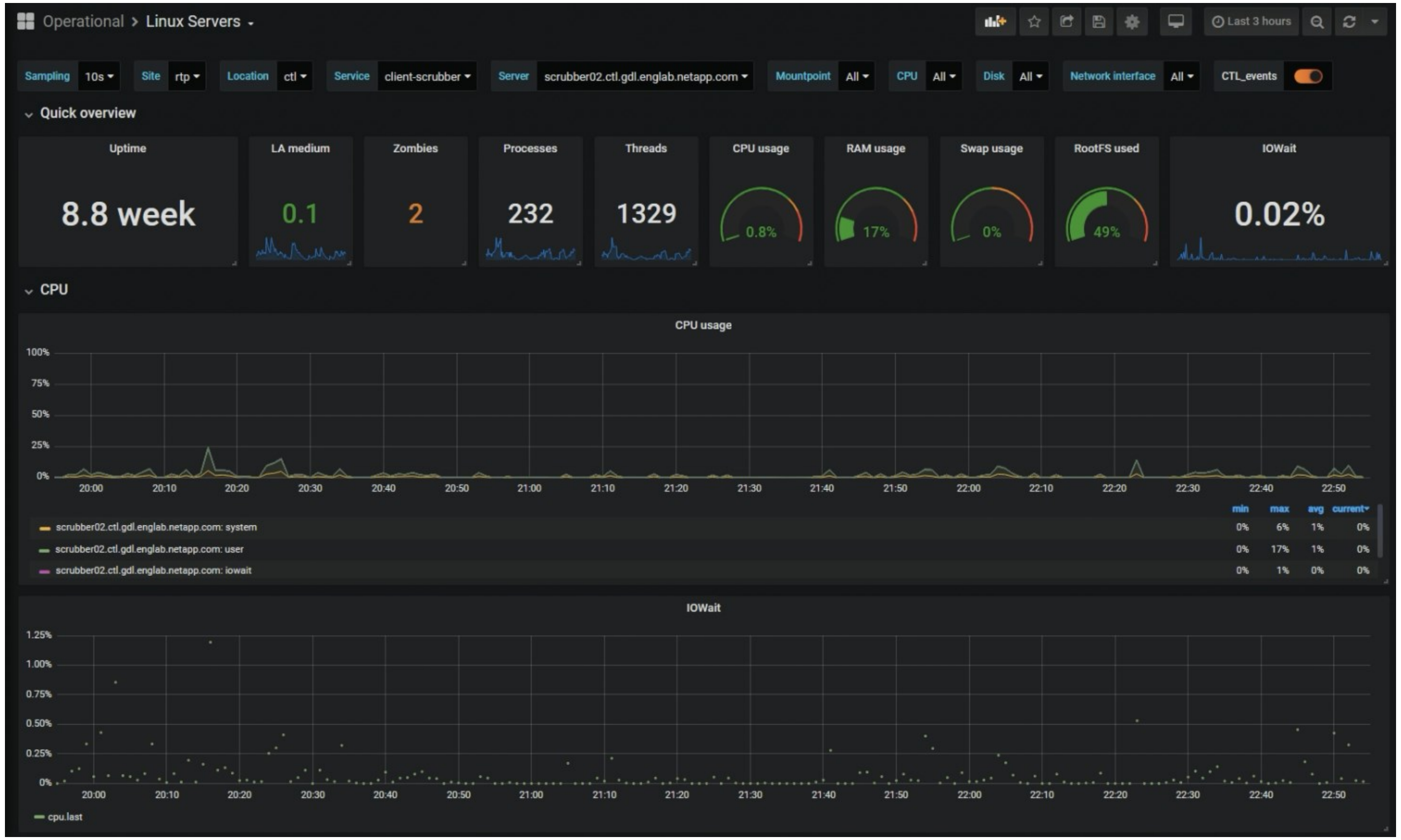
Measuring System Resources

Telegraf For The Win!

- System stats collected by default with Telegraf out of the box!

- Once a Grafana template is established new hosts can be added in seconds!

1. Install telegraf RPM
2. Copy telegraf config to /etc/telegraf/telegraf.conf
3. Start telegraf service



Operational > Linux Servers

Last 3 hours

Sampling 10s · Site rtp · Location ctl · Service client-scrubber · Server scrubber02.ctl.gdl.englab.netapp.com · Mountpoint All · CPU All · Disk All · Network interface All · CTL_events

Quick overview

Uptime	LA medium	Zombies	Processes	Threads	CPU usage	RAM usage	Swap usage	RootFS used	IOWait
8.8 week	0.1	2	232	1329	0.8%	17%	0%	49%	0.02%

CPU

CPU usage

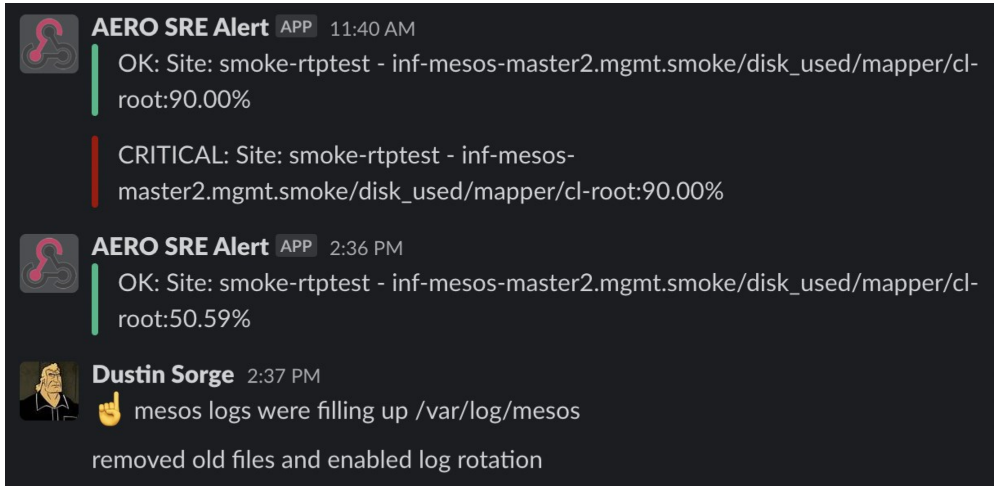
	min	max	avg	current
scrubber02.ctl.gdl.englab.netapp.com: system	0%	6%	1%	0%
scrubber02.ctl.gdl.englab.netapp.com: user	0%	17%	1%	0%
scrubber02.ctl.gdl.englab.netapp.com: iowait	0%	1%	0%	0%

IOWait

Alerting for Resource Utilization

You can alert on whatever system stats you wish!

Know your file systems are filling up before the box tips over!



AERO SRE Alert APP 11:40 AM

OK: Site: smoke-rtptest - inf-mesos-master2.mgmt.smoke/disk_used/mapper/cl-root:90.00%

CRITICAL: Site: smoke-rtptest - inf-mesos-master2.mgmt.smoke/disk_used/mapper/cl-root:90.00%

AERO SRE Alert APP 2:36 PM

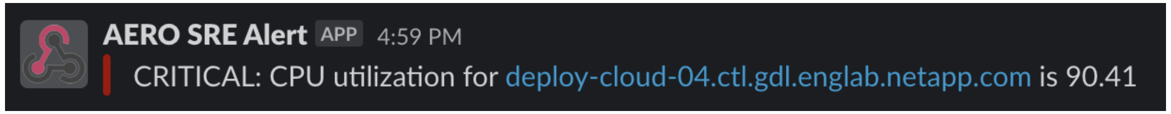
OK: Site: smoke-rtptest - inf-mesos-master2.mgmt.smoke/disk_used/mapper/cl-root:50.59%

Dustin Sorge 2:37 PM

☝ mesos logs were filling up /var/log/mesos

removed old files and enabled log rotation

Know when CPU is spiking on critical infrastructure!



AERO SRE Alert APP 4:59 PM

CRITICAL: CPU utilization for deploy-cloud-04.ctl.gdl.englab.netapp.com is 90.41

Questions?