DigitalOcean

@snehainguva

observability and product release: **leveraging prometheus** to build and test new products

about me

software engineer @DigitalOcean
currently **network services**
<3 **cats**

some **stats**

90M+ timeseries

85 instances of prometheus

1.7M+ samples/sec

the **history**

ye’ olden days

use nagios + various plugins to monitor

use collectd + statsd + graphite

openTSDB

lovely prometheus

white-box monitoring

multi-dimensional data model

fantastic querying language

glorious kubernetes

easily deploy and update services

scalability

combine with prometheus + alertmanager

sneha joins networking

set up monitoring for VPC

working on DHCP

how can we use prometheus even before release?

the plan:

✔ observability DigitalOcean

build --- instrument --- test --- iterate

examples

metrics: *time-series of sampled data*

tracing: *propagating metadata through different requests, threads, and processes*

logging: *record of discrete events over time*

metrics:
what do we measure?

four golden signals

latency: *time to service a request*

traffic: *requests/second*

error: *error rate of requests*

saturation: *fullness of a service*

Utilization

Saturation

Error rate

*“**USE** metrics often allow users to solve 80% of server issues with 5% of the effort.”*

the plan:

✔ observability DigitalOcean

✔ **build --- instrument --- test --- iterate**

examples

build:

design the service

write it in go

use internally shared libraries

build: doge/dorpc - shared rpc library

```
var DefaultInterceptors = []string{ StdLoggingInterceptor, StdMetricsInterceptor, StdTracingInterceptor}

func NewServer(opt ...ServerOpt) (*Server, error) {
        opts := serverOpts{
                name:           "server",
                clientTLSAuth:   tls.VerifyClientCertIfGiven,
                intercept:       interceptor.NewPathInterceptor(interceptor.DefaultInterceptors...),
                keepAliveParams:  DefaultServerKeepAlive,
                keepAliveEnforce: DefaultServerKeepAliveEnforcement,
        }
      ...
}
```

instrument:

send logs to centralized logging

send spans to trace-collectors

set up prometheus metrics

metrics instrumentation: go-client

```
func (s *server) initalizeMetrics() {
        s.metrics = metricsConfig{
        attemptedConvergeChassis: s.metricsNode.Gauge("attempted_converge_chassis", "number of chassis converger attempting to converge"),
        failedConvergeChassis:    s.metricsNode.Gauge("failed_converge_chassis", "number of chassis that failed to converge"),
        }
}

func (s *server) ConvergeAllChassis(...) {
        ...
        s.metrics.attemptedConvergeChassis(float64(len(attempted)))
        s.metrics.failedConvergeChassis(float64(len(failed)))
        ...
}
```

Quick Q & A: Collector Interface

```
// A collector must be registered.
prometheus.MustRegister(collector)

type Collector interface {

    // Describe sends descriptors to channel.
    Describe(chan<- *Desc)

    // Collect is used by the prometheus registry on a scrape.
    // Metrics are sent to the provided channel.
    Collect(chan<- Metric)
}
```

metrics instrumentation: third-party exporters

Built using the collector interface

Sometimes we build our own

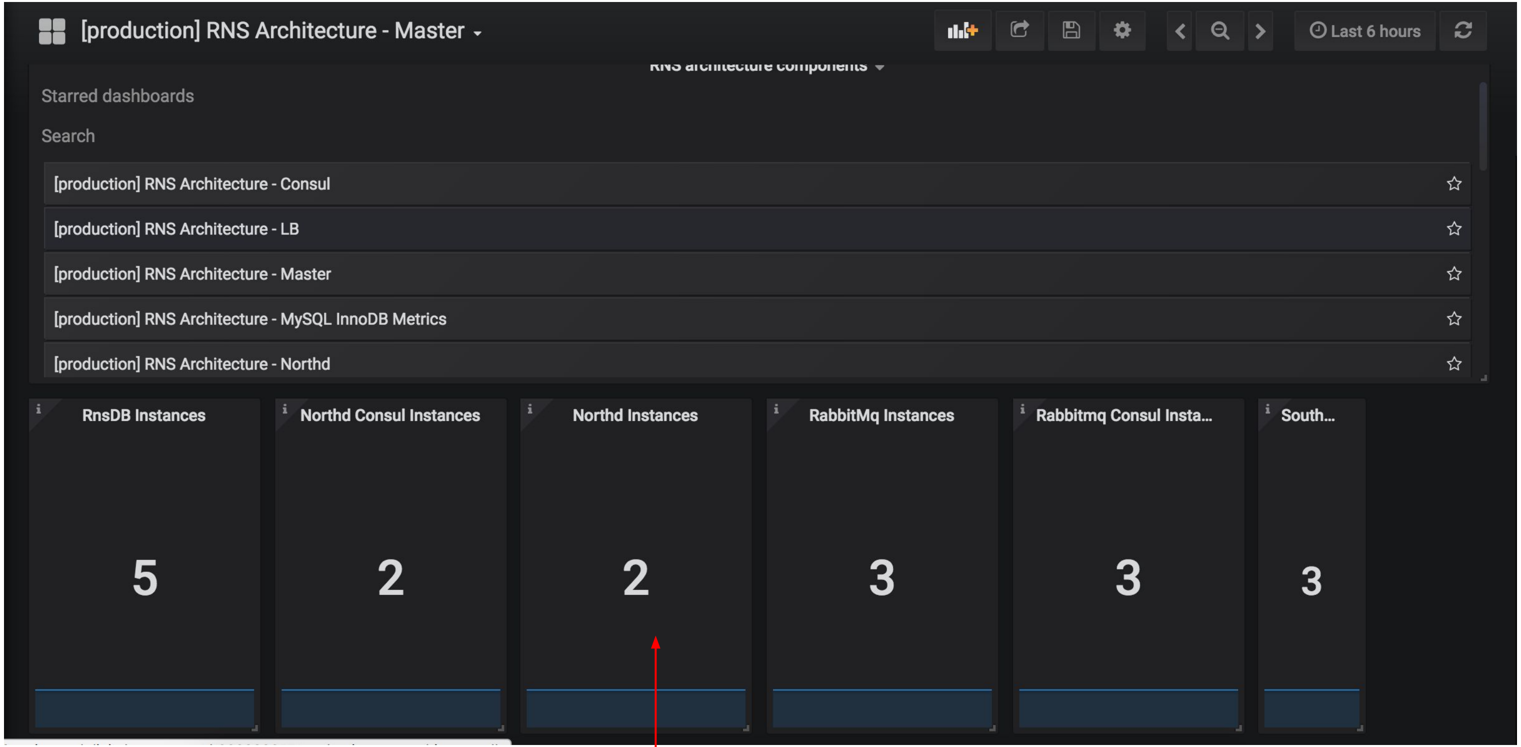
Often we use others:

- github.com/prometheus/**mysqld**_exporter
- github.com/kbudde/**rabbitmq**_exporter
- github.com/prometheus/**node**_exporter
- github.com/digitalocean/**openvswitch**_exporter

metrics instrumentation: in-service collectors

```
type RateMap struct {
        mu          sync.Mutex
        ...
        rateMap     map[string]*rate
}
var _ prometheus.Collector = &RateMapCollector{}
func (r *RateMapCollector) Describe(ch chan<- *prometheus.Desc) {
        ds := []*prometheus.Desc{ r.RequestRate}
        for _, d := range ds {
                ch <- d
        }
}
func (r *RateMapCollector) Collect(ch chan<- prometheus.Metric) {
        ...
        ch <- prometheus.MustNewConstHistogram( r.RequestRate, count, sum, rateCount)
}
```

metrics instrumentation: dashboards #1



[production] RNS Architecture - Master

Last 6 hours

RNS architecture components

Starred dashboards

Search

[production] RNS Architecture - Consul

[production] RNS Architecture - LB

[production] RNS Architecture - Master

[production] RNS Architecture - MySQL InnoDB Metrics

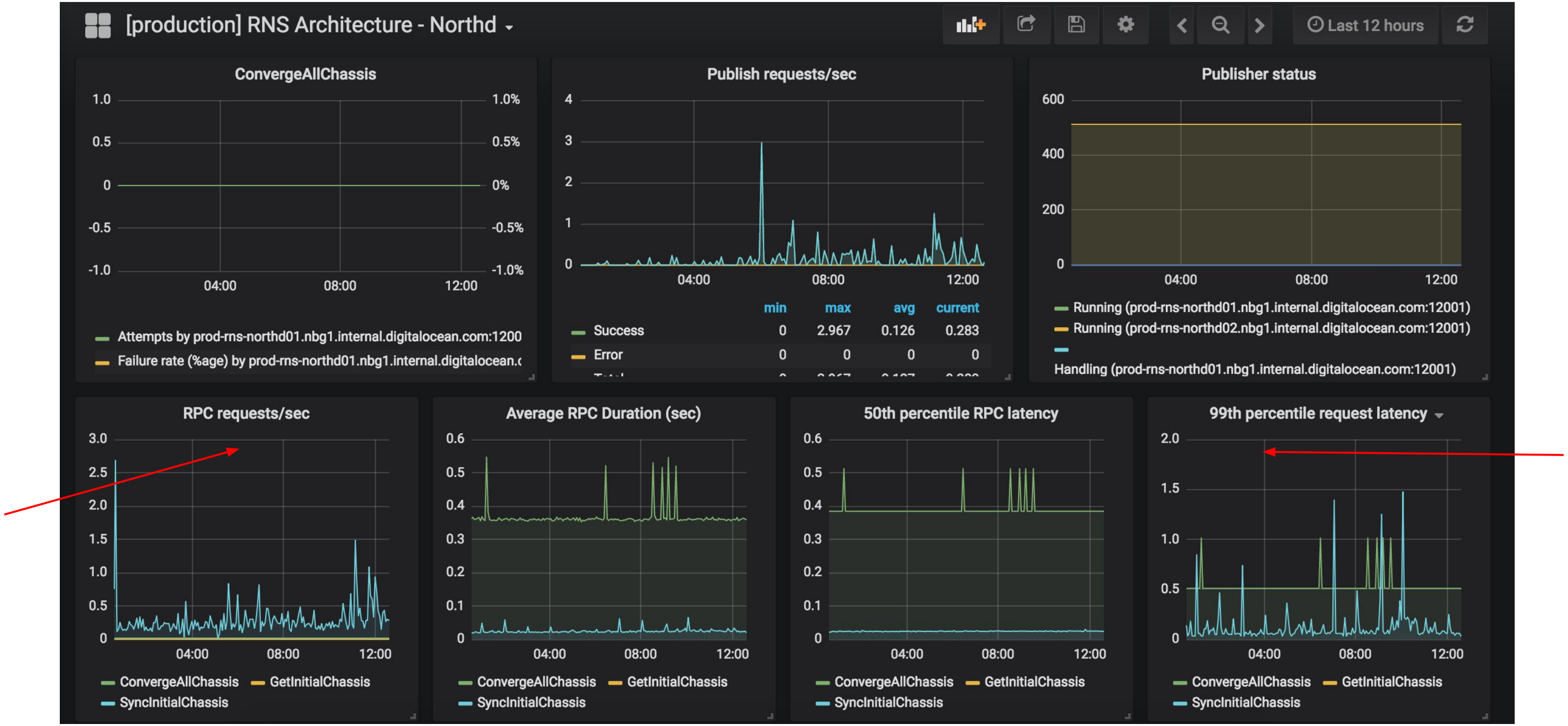
[production] RNS Architecture - Northd

RnsDB Instances	Northd Consul Instances	Northd Instances	RabbitMq Instances	Rabbitmq Consul Insta...	South...
5	2	2	3	3	3

state metrics

digitalocean.com

metrics instrumentation: dashboard #2

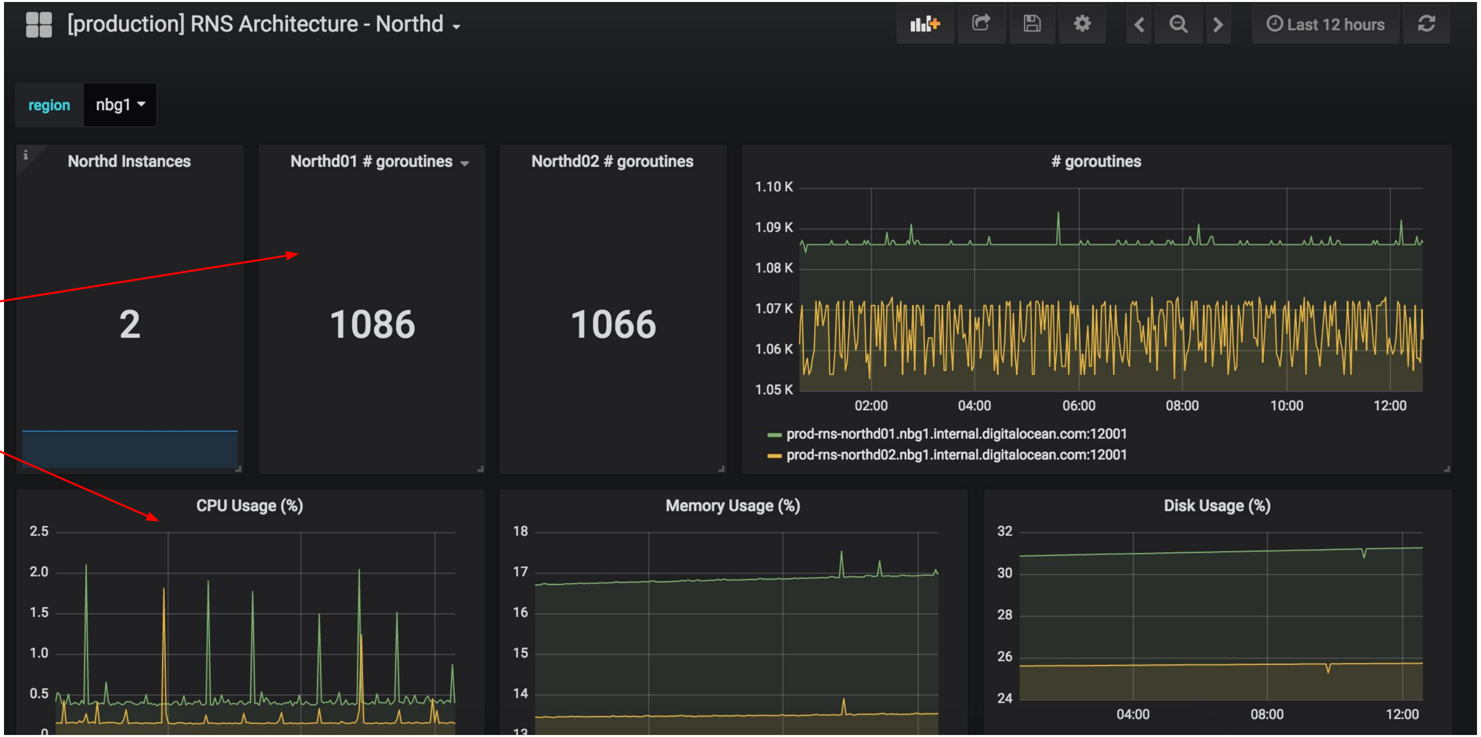


	min	max	avg	current
Success	0	2.967	0.126	0.283
Error	0	0	0	0

request rate

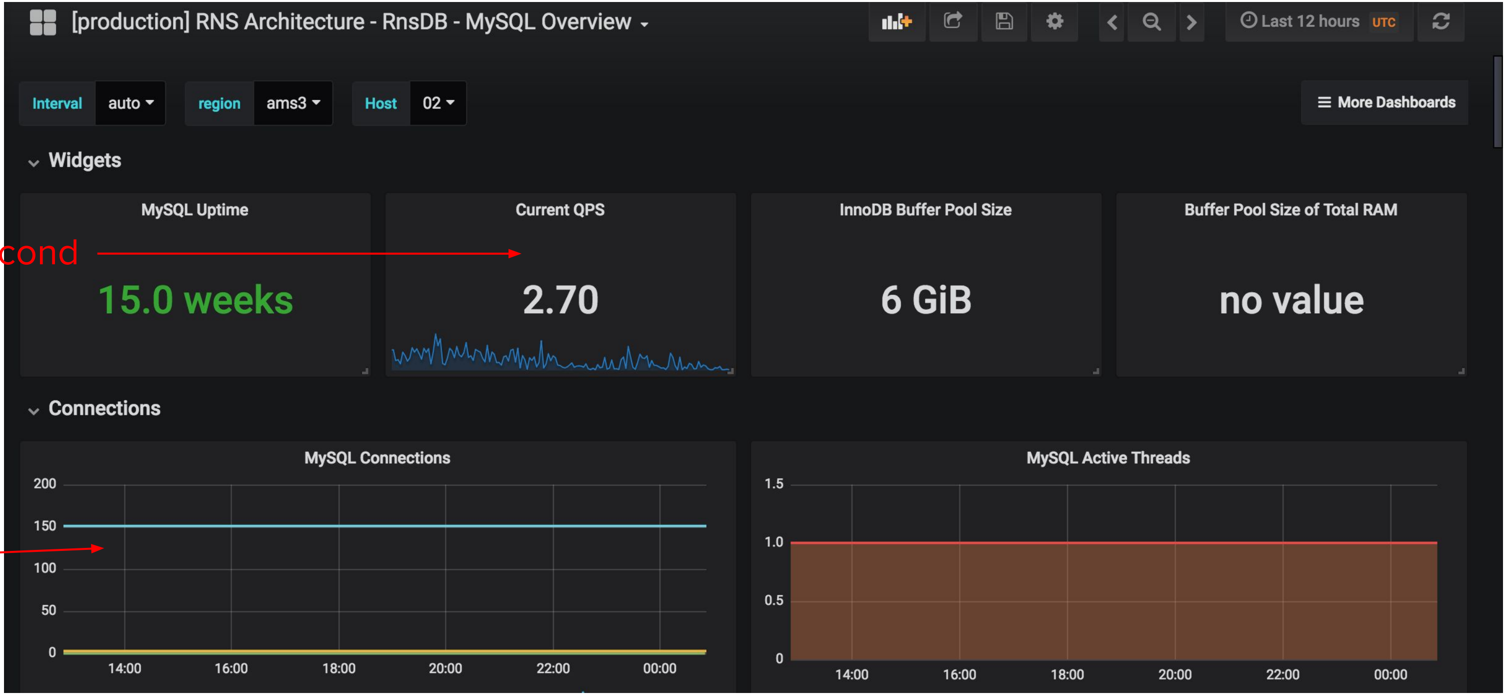
request latency

metrics instrumentation: dashboard #3



utilization metrics

metrics instrumentation: dashboard #4



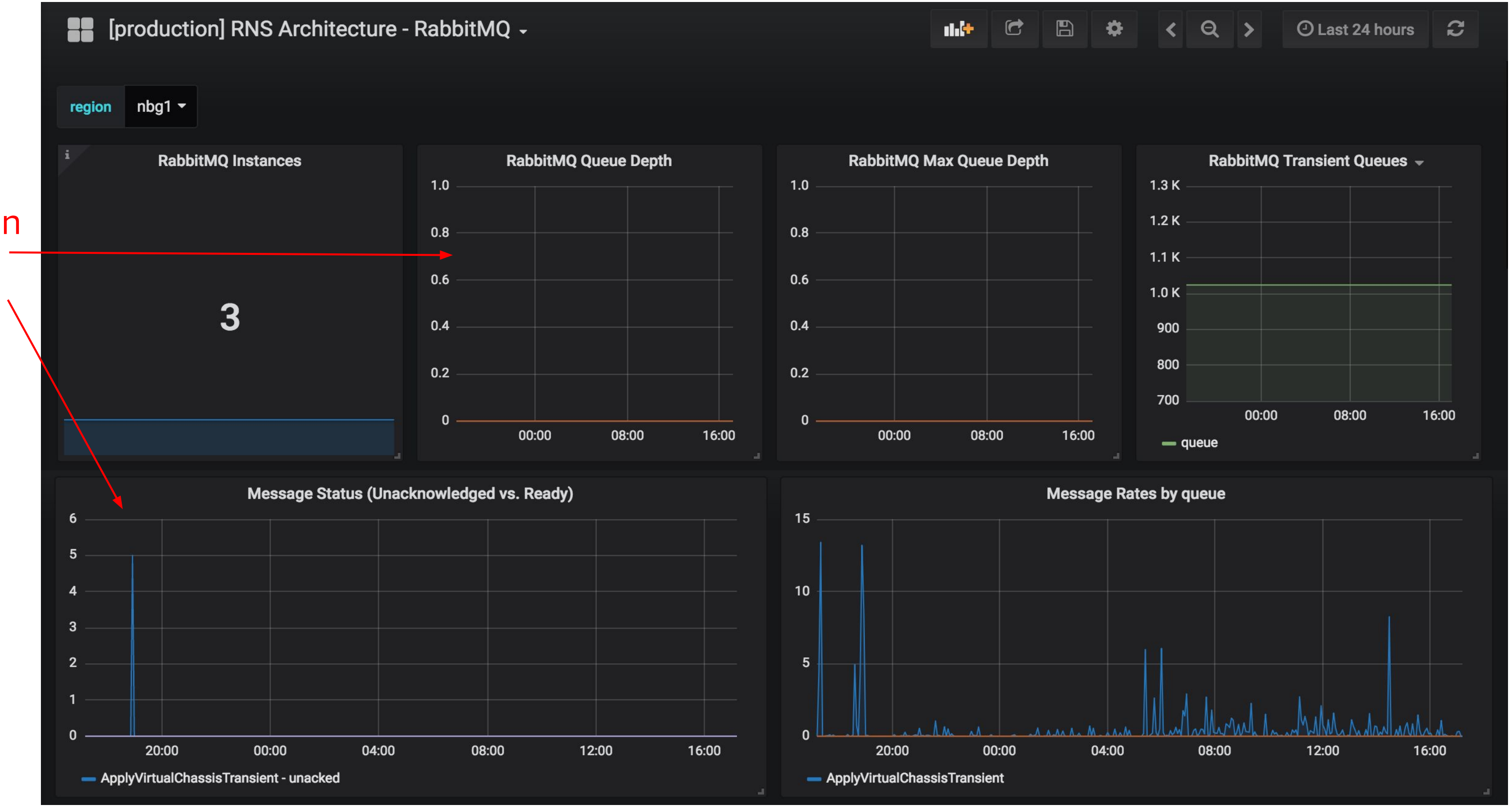
queries/second

utilization

digitalocean.com

metrics instrumentation: dashboard #5

saturation metric



digitalocean.com

test:

load testing:

grpc-clients and goroutines

chaos testing:

take down a component of a system

integration testing:

how does this feature integrate with the cloud?

testing: identify key issues

how is our latency? → *use tracing to dig down*

is there a goroutine leak? → *use a worker pool*

does resource usage increase with traffic? → *use cpu and memory profiling*

is there a high error rate? → *check logs for types of error*

how are our third-party services?

testing: tune metrics + alerts

do we need more labels for our metrics?

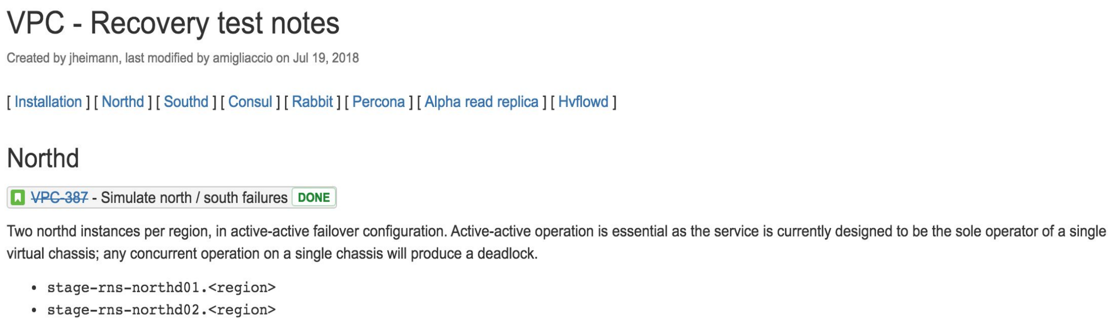
should we collect more data?

***State-based alerting**: Is our service up or down?*

***Threshold alerting**: When does our service fail?*

testing: documentation

set up operational playbooks
document recovery efforts



VPC - Recovery test notes

Created by jheimann, last modified by amigliaccio on Jul 19, 2018

[Installation] [Northd] [Southd] [Consul] [Rabbit] [Percona] [Alpha read replica] [Hvflowd]

Northd

~~VPC-387~~ - Simulate north / south failures DONE

Two northd instances per region, in active-active failover configuration. Active-active operation is essential as the service is currently designed to be the sole operator of a single virtual chassis; any concurrent operation on a single chassis will produce a deadlock.

- `stage-rns-northd01.<region>`
- `stage-rns-northd02.<region>`

iterate!

(but really, let’s look at some examples…)

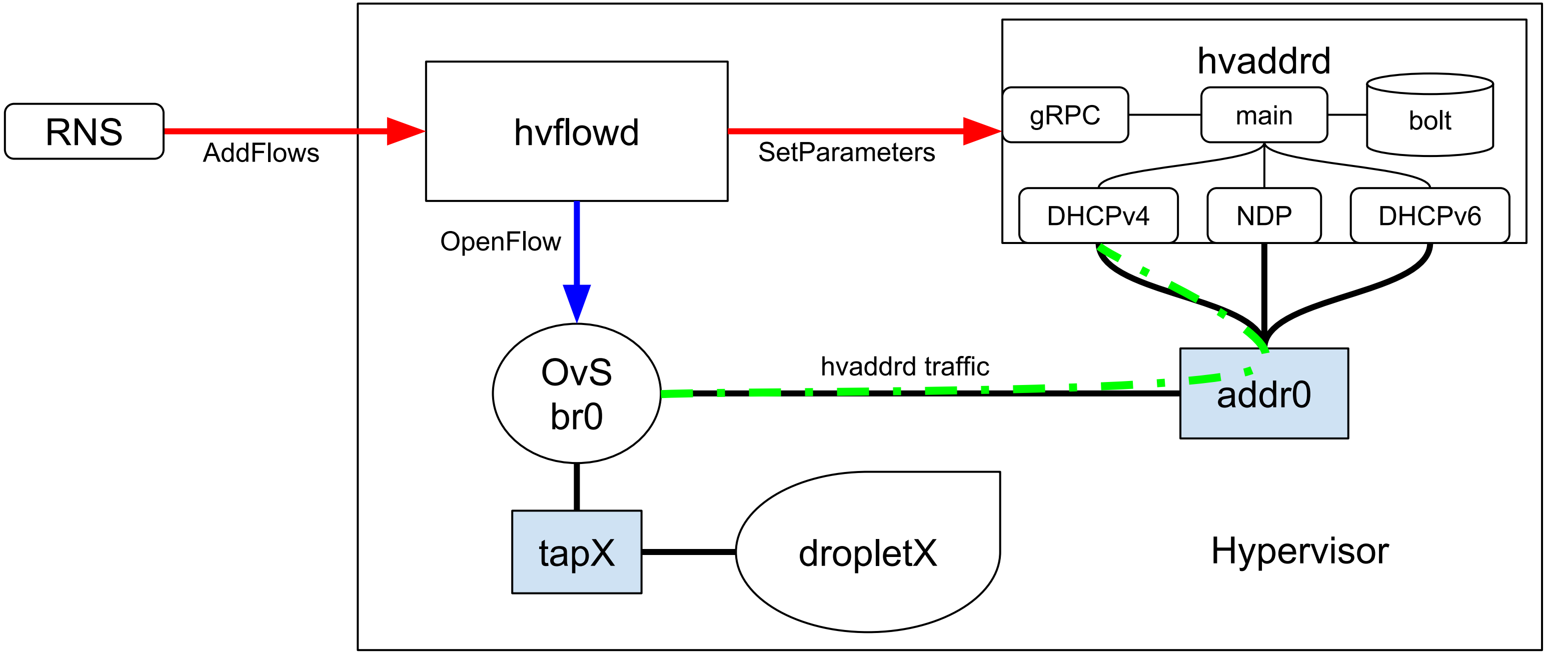
the plan:

- ✔ observability DigitalOcean
- ✔ **build --- instrument --- test --- iterate**
- ✔ examples

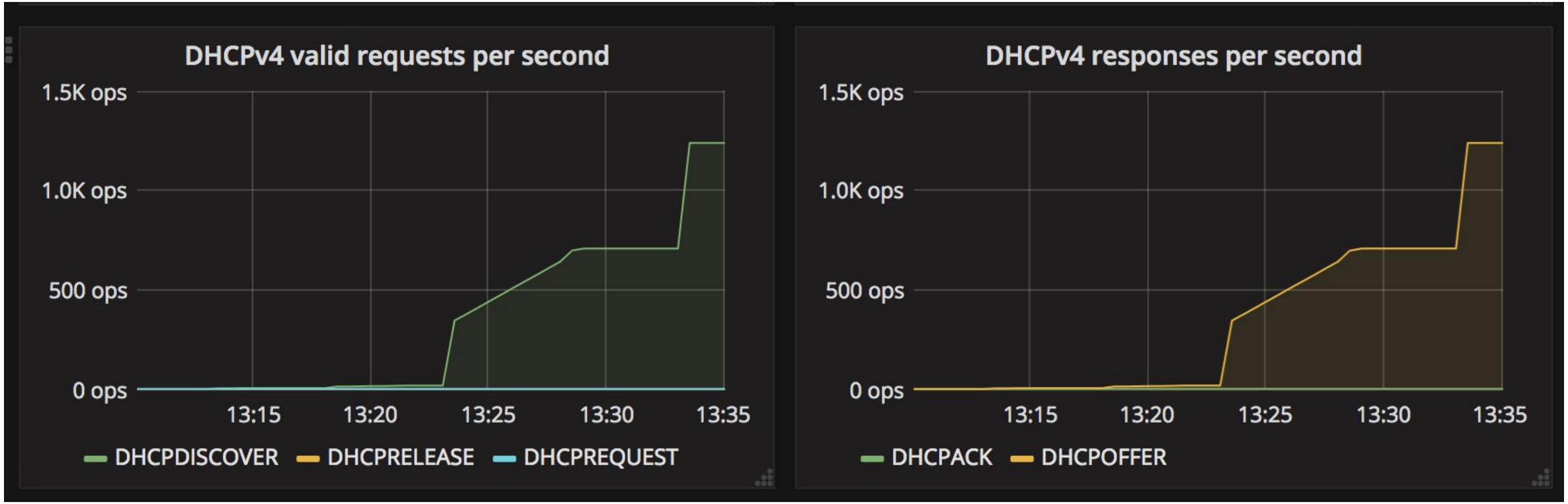
product #1: DHCP

(hvaddrd)

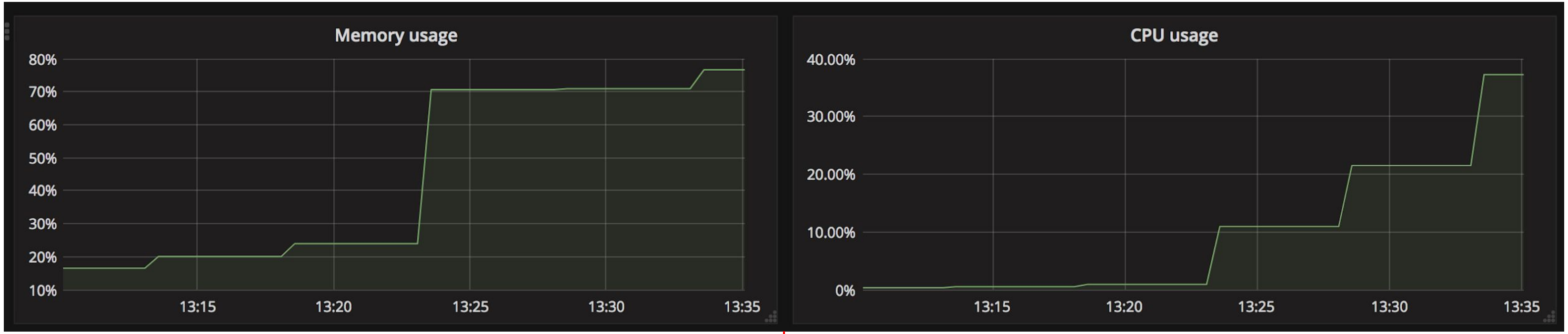
product #1: DHCP

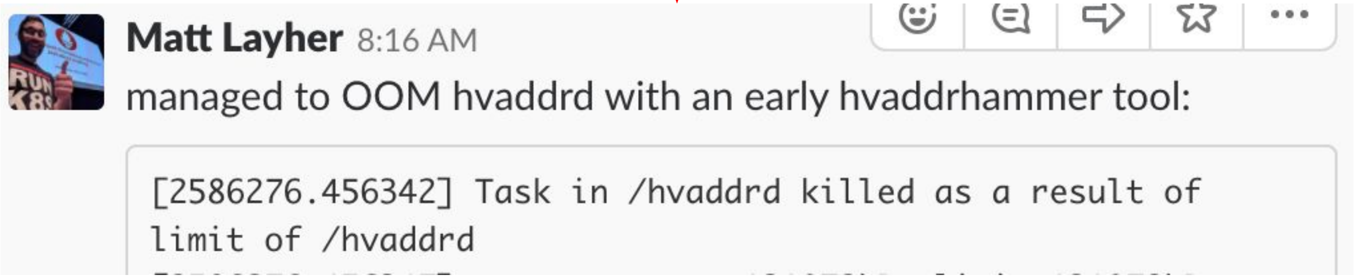


DHCP: load testing



DHCP: load testing (2)




Matt Layher 8:16 AM

managed to OOM hvaddrd with an early hvaddrhammer tool:

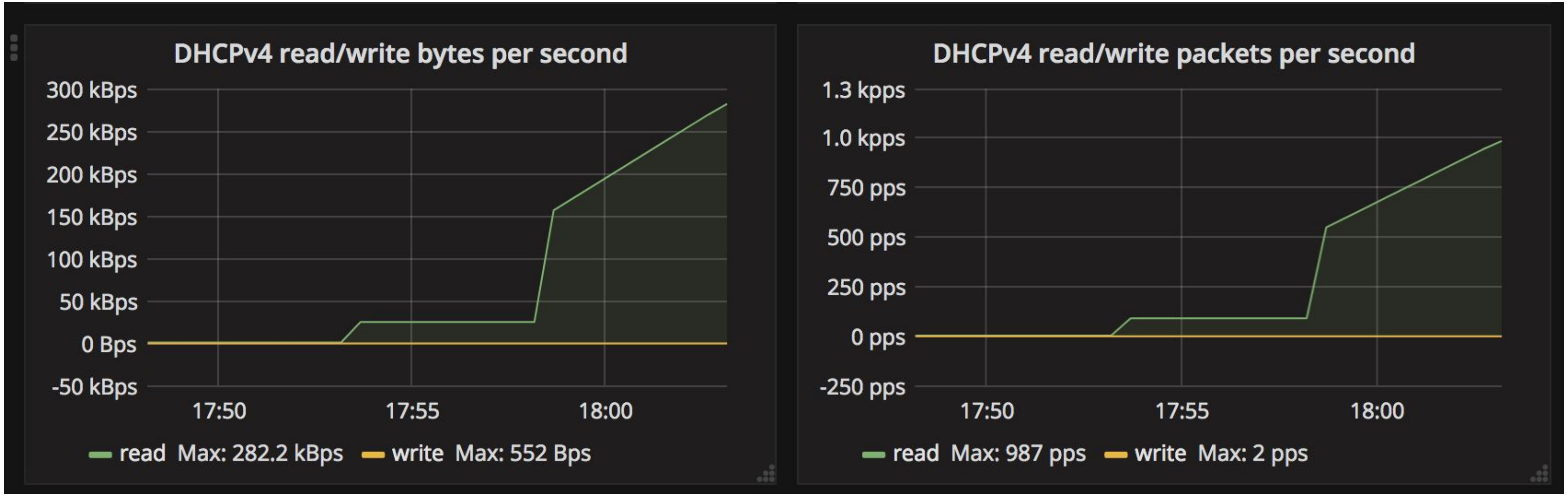
```
[2586276.456342] Task in /hvaddrd killed as a result of
limit of /hvaddrd
```

DHCP: custom conn collector

```go
package dhcp4conn

var _ prometheus.Collector = &collector{}

// A collector gathers connection metrics.
type collector struct {
        ReadBytesTotal    *prometheus.Desc
        ReadPacketsTotal  *prometheus.Desc
        WriteBytesTotal   *prometheus.Desc
        WritePacketsTotal *prometheus.Desc
}
```

Implements the net.conn interface and allows us to process ethernet frames for validation and other purposes.

DHCP: custom conn collector



DHCP: goroutine worker pools

```
workC := make(chan request, Workers)
for i := 0; i < Workers; i++ {
    go func() {
        defer workWG.Done()
        for r := range workC {
            s.serve(r.buf, r.from)
        }
    }()
}
```

Uses buffered channel to process requests, limiting goroutines and resource usage.

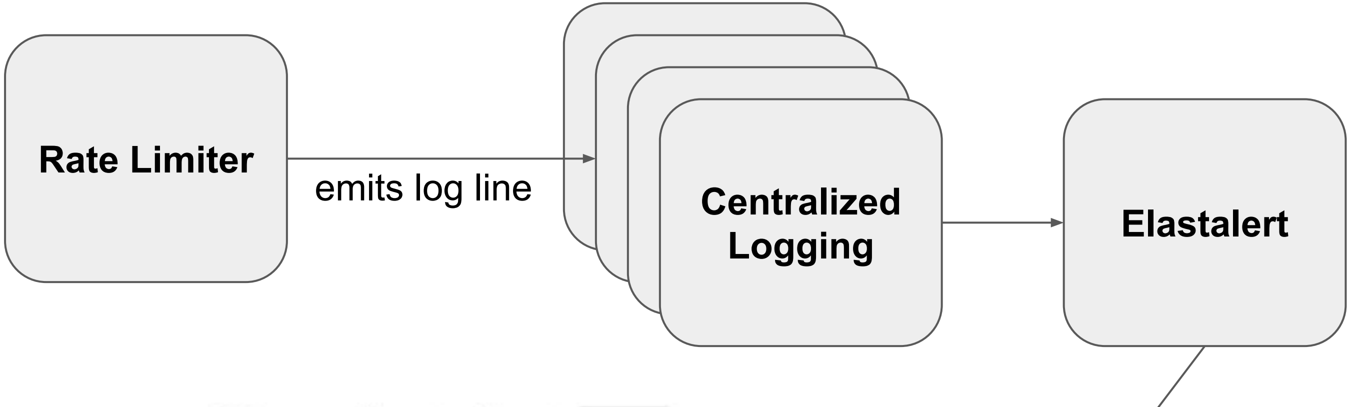
DHCP: rate limiter collector

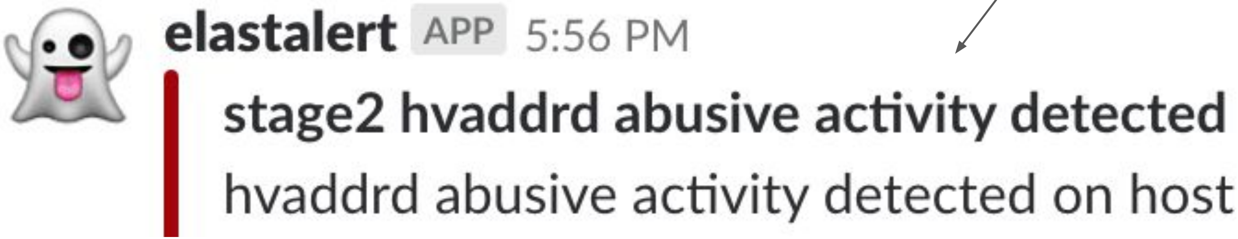
```
type RateMap struct {
        mu          sync.Mutex
        ...
        rateMap       map[string]*rate
}

type RateMapCollector struct {
        RequestRate *prometheus.Desc
        rm          *RateMap
        buckets     []float64
}

func (r *RateMapCollector) Collect(ch chan<- prometheus.Metric) {
        ...
        ch <- prometheus.MustNewConstHistogram(
                r.RequestRate,
                count, sum,
                rateCount)
}
```

ratemap calculates the exponentially weighted moving average on a per-client basis and limits requests

collector gives us a snapshot of rate distributions

DHCP: rate alerts

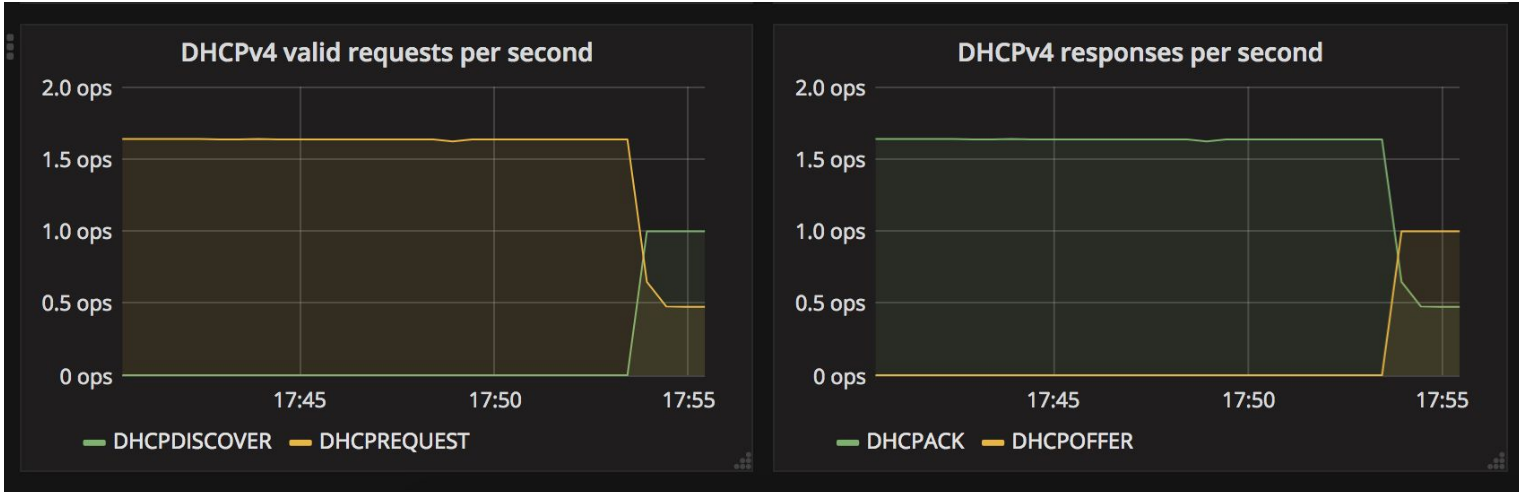




stage2 hvaddrd abusive activity detected

hvaddrd abusive activity detected on host

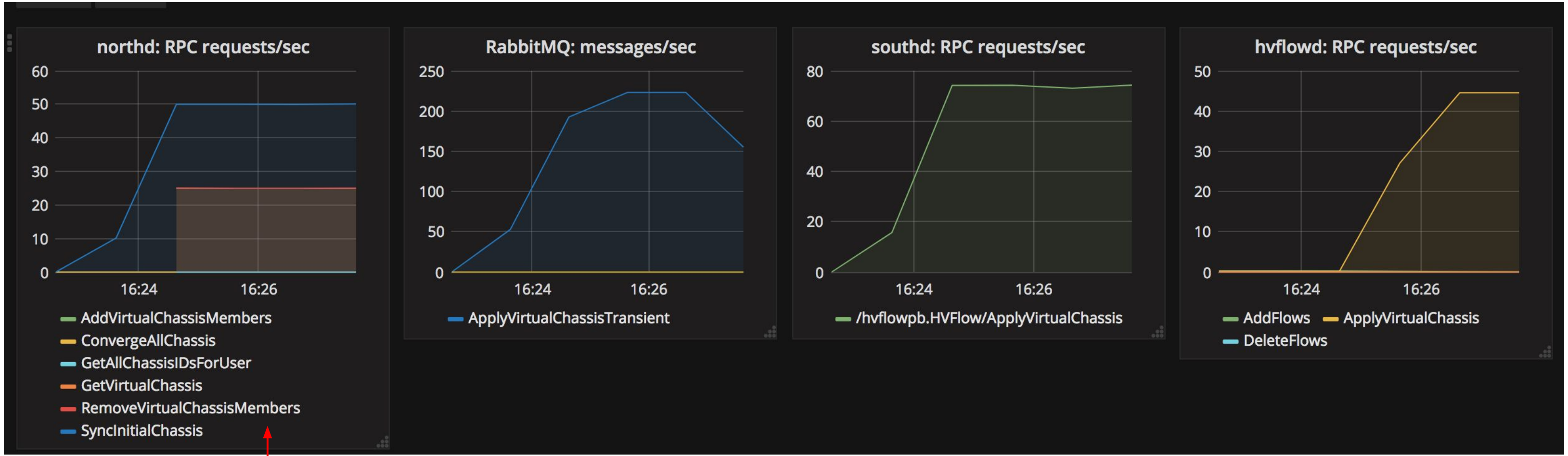
DHCP: the final result



product #2: VPC

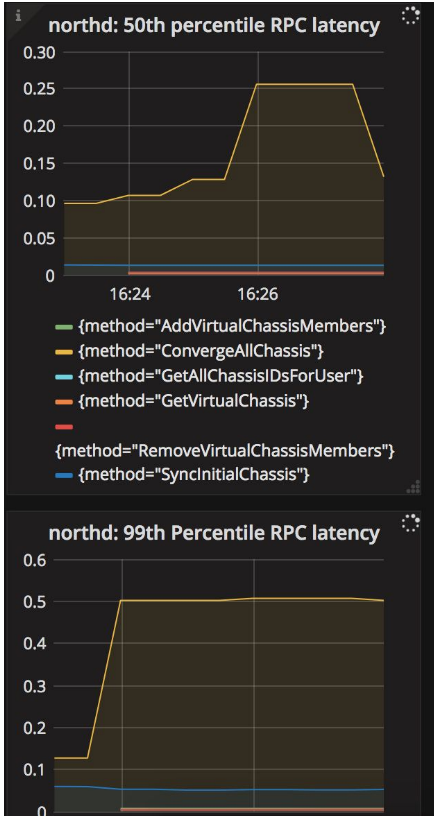
product #2: VPC

VPC: load-testing



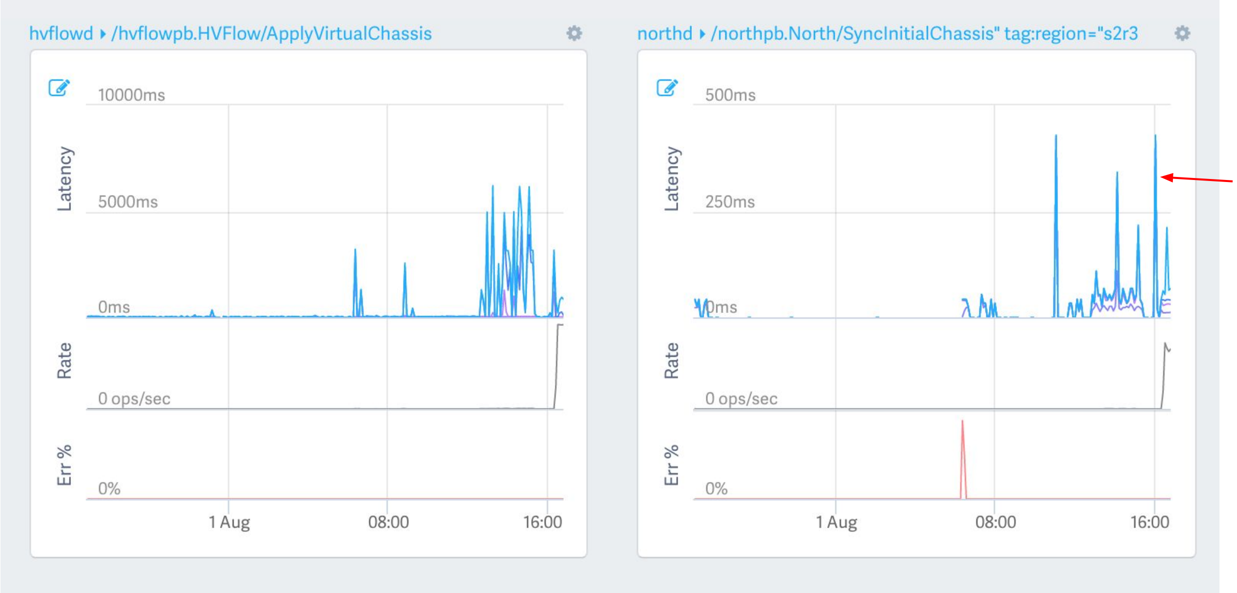
load tester repeatedly makes some RPC calls

VPC: latency issues (1)



as load testing continued, started to notice latency in different rpc calls

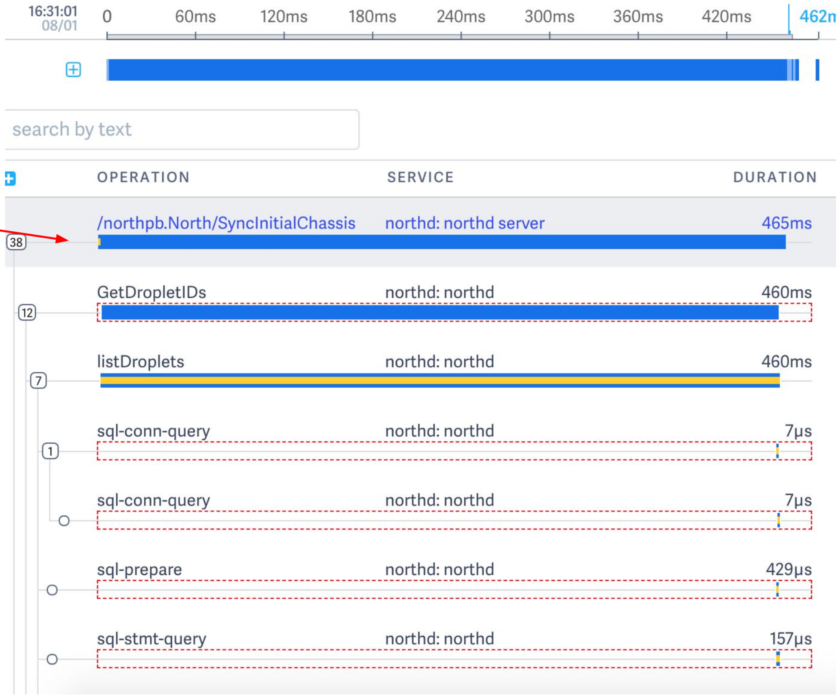
VPC: latency issues (2)



use tracing to take a look at the /SyncInitialChassis call

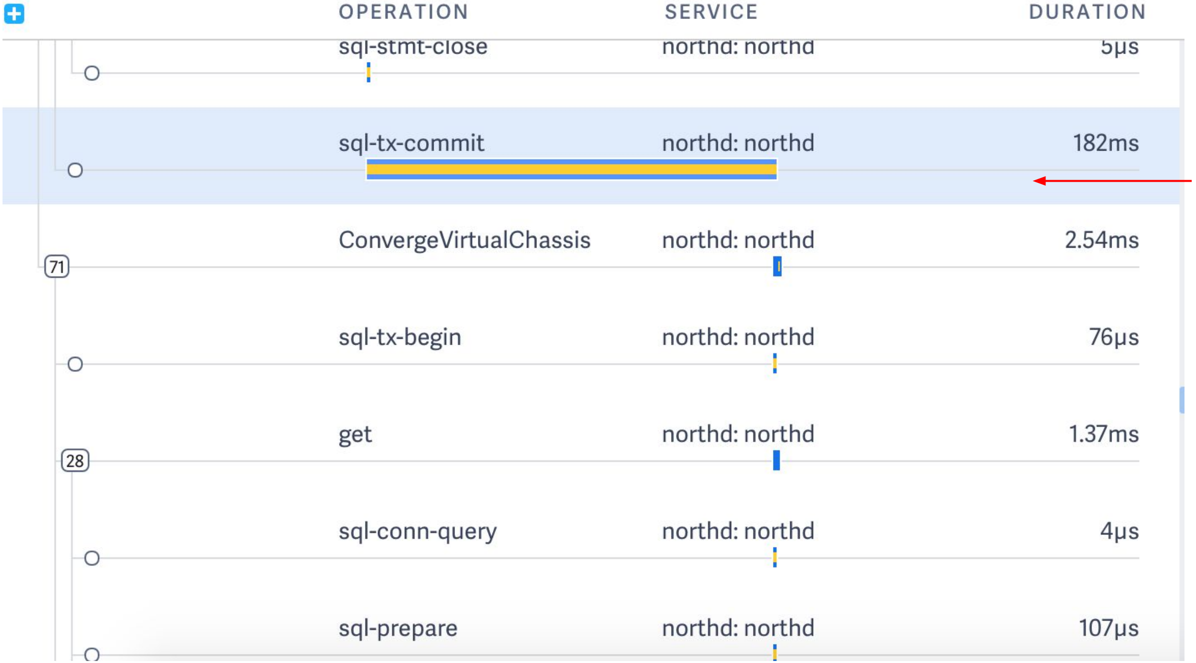
VPC: latency issues (3)

Note that spans for some traces were being dropped. Slowing down the load tester, however, eventually ameliorated that problem.



OPERATION	SERVICE	DURATION
/northpb.North/SyncInitialChassis	northd: northd server	465ms
GetDropletIDs	northd: northd	460ms
listDroplets	northd: northd	460ms
sql-conn-query	northd: northd	7µs
sql-conn-query	northd: northd	7µs
sql-prepare	northd: northd	429µs
sql-stmt-query	northd: northd	157µs

VPC: latency issues (4)

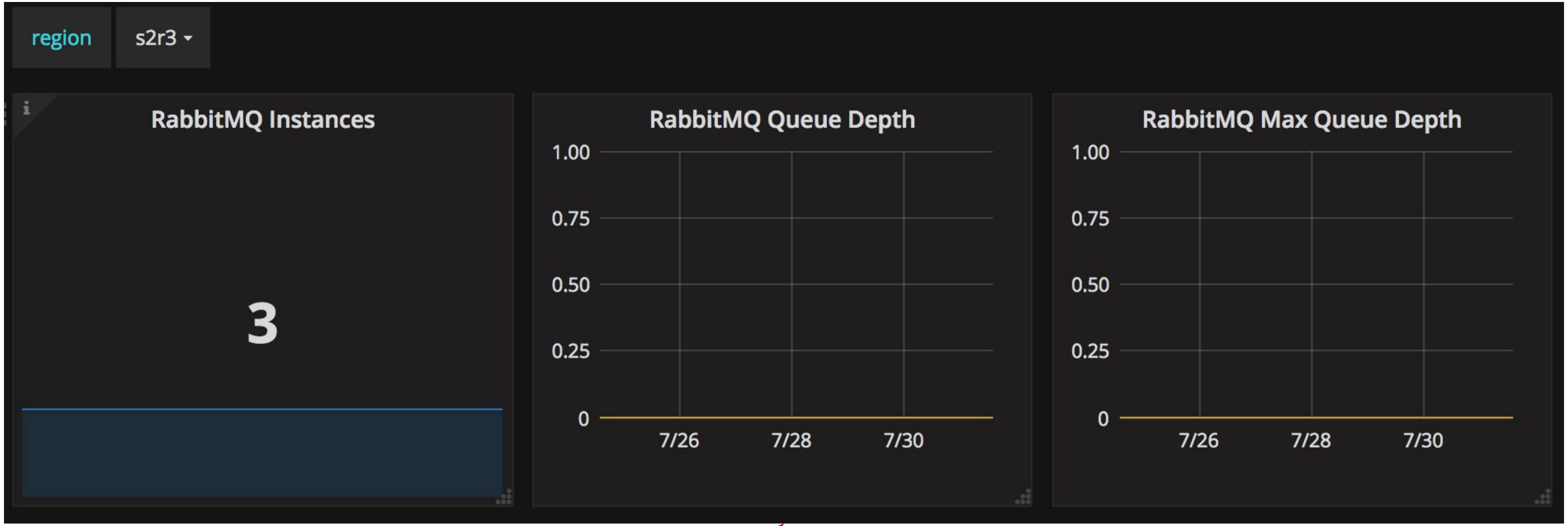


OPERATION	SERVICE	DURATION
sql-stmt-close	northd: northd	5µs
sql-tx-commit	northd: northd	182ms
ConvergeVirtualChassis	northd: northd	2.54ms
sql-tx-begin	northd: northd	76µs
get	northd: northd	1.37ms
sql-conn-query	northd: northd	4µs
sql-prepare	northd: northd	107µs

"The fix was to be smarter and do the queries more efficiently. The repetitive loop of queries to rnsdb really stood out in the lightstep data."

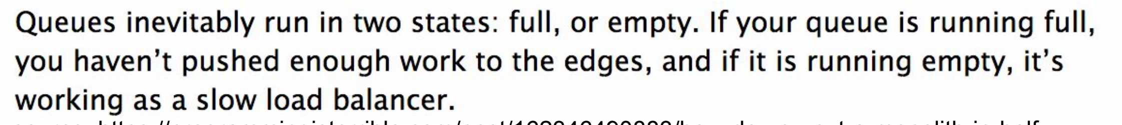
- Bob Salmi

VPC: remove component



region s2r3

can queue be replaced with simple request-response system?

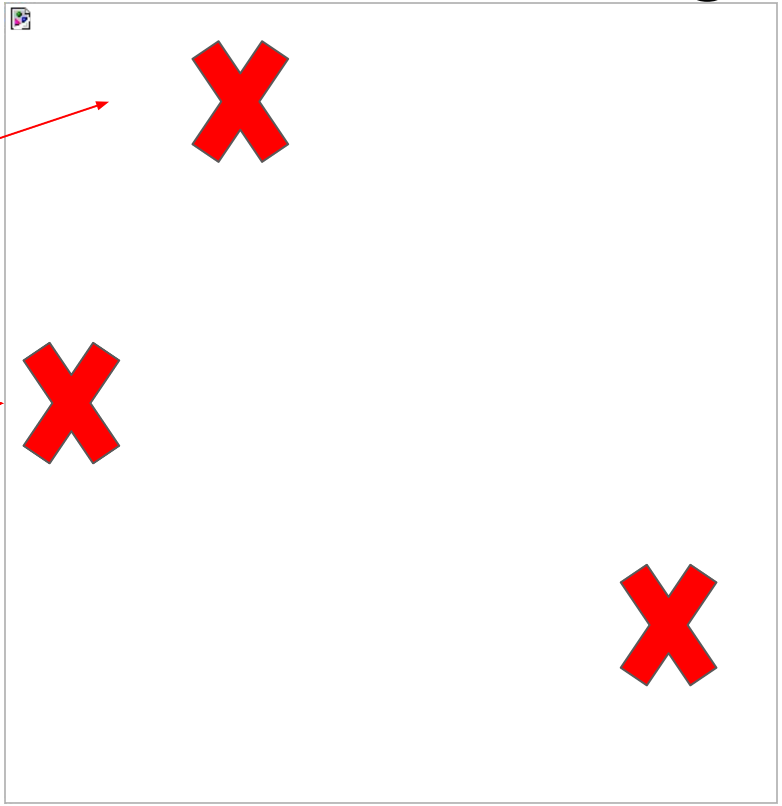


Queues inevitably run in two states: full, or empty. If your queue is running full, you haven't pushed enough work to the edges, and if it is running empty, it's working as a slow load balancer.

source: https://programmingisterrible.com/post/162346490883/how-do-you-cut-a-monolith-in-half

digitalocean.com

VPC: chaos testing



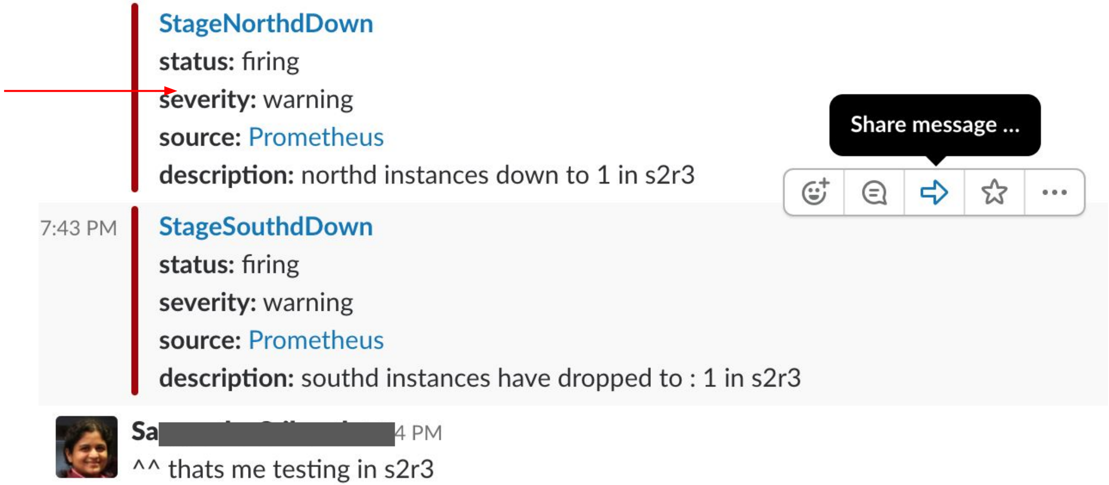
Induce northd failure and ensure failover works

Drop primary and recovery from secondary

Induce south service failure and see how rabbit responds

VPC: add alerts (1)

state-based alerts



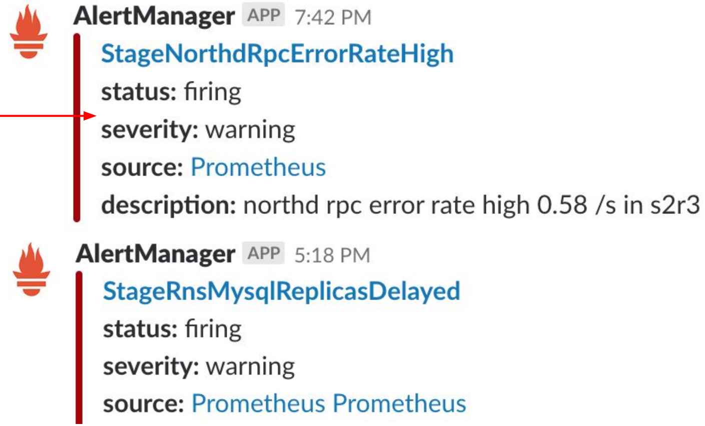
StageNorthdDown
status: firing
severity: warning
source: Prometheus
description: northd instances down to 1 in s2r3

7:43 PM StageSouthdDown
status: firing
severity: warning
source: Prometheus
description: southd instances have dropped to : 1 in s2r3

Sa[redacted]4 PM
^^ thats me testing in s2r3

VPC: add alerts (2)

threshold alert



AlertManager APP 7:42 PM
StageNorthdRpcErrorRateHigh
status: firing
severity: warning
source: Prometheus
description: northd rpc error rate high 0.58 /s in s2r3

AlertManager APP 5:18 PM
StageRnsMysqlReplicasDelayed
status: firing
severity: warning
source: Prometheus Prometheus

conclusion

what?

four golden signals, USE metrics

when?

as early as possible

how?

combine with profiling, logging, tracing

thanks!

@snehainguva