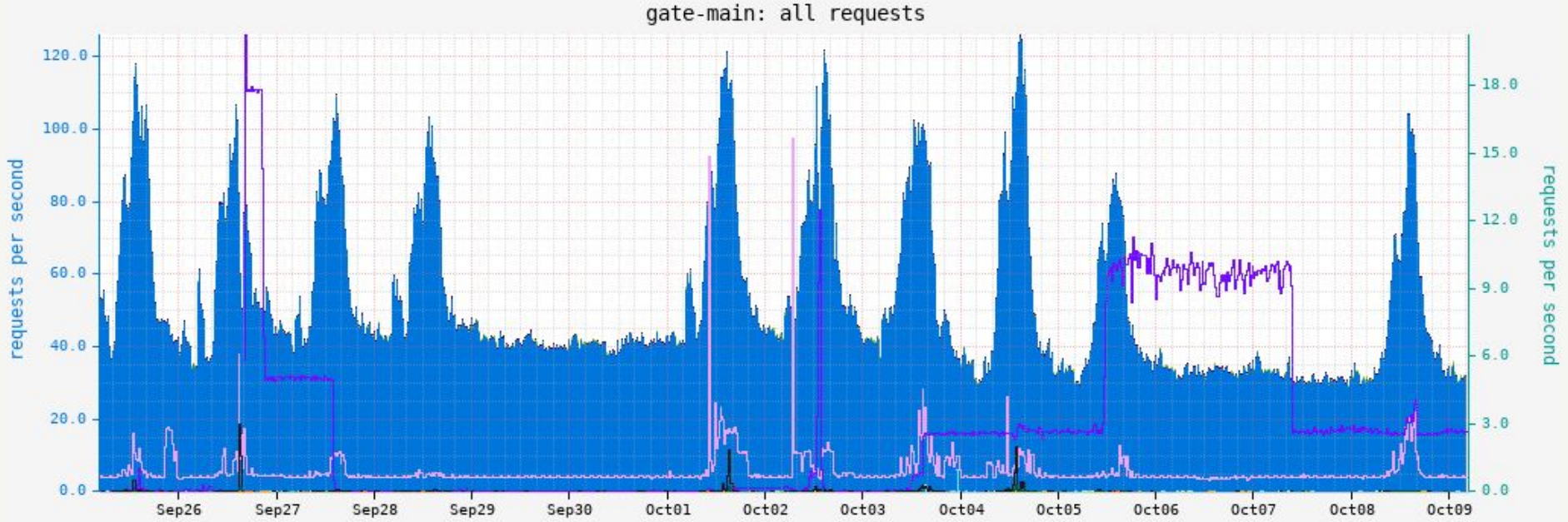
Spinnaker In Production.

LESSONS LEARNED
FROM THE TRENCHES

NETFLIX


gate-main: all requests
requests per second
120.0
100.0
80.0
60.0
40.0
20.0
0.0
18.0
15.0
12.0
9.0
6.0
3.0
0.0
Sep26
Sep27
Sep28
Sep29
Sep30
Oct01
Oct02
Oct03
Oct04
Oct05
Oct06
Oct07
Oct08
Oct09

Deployment footprint.

Single shared Spinnaker installation.

Each microservice is deployed independently.

Dedicated cluster(s) for each microservice.

Backstory.

WHY IS IT ALWAYS
AN ORIGIN STORY??

Bottlenecks.

Each microservice can scale out and scale up, however:

gate (API Service)
Poorly tuned hystrix pools get saturated.

orca (Orchestration Engine)
Stateful deploys tied to a particular instance.
Persistence of orchestration state.

clouddriver (Cloud Integration Layer)
Volume of data indexing of the state of the cloud.
Query load to aggregate the state of the cloud.

Unclogging the drain.

Eliminating a bottleneck in one location can create one in another.

gate
Better tuning and partitioning of hystrix pools.

orca
Re-architected into a distributed work queue.
Rewrote persistence backend in SQL.

clouddriver
Uh-oh.

clouddriver.

- Periodically crawl cloud provider data. **write-heavy**
- Aggregate crawled data into application-centric view for API users. **Read-heavy**
- Initially in-memory (that doesn't scale!)
- Some guy thought redis would be a good idea...

Sorry!

Cameron Fieber
Senior Software Engineer
Delivery Engineering at Netflix

cfieber@netflix.com

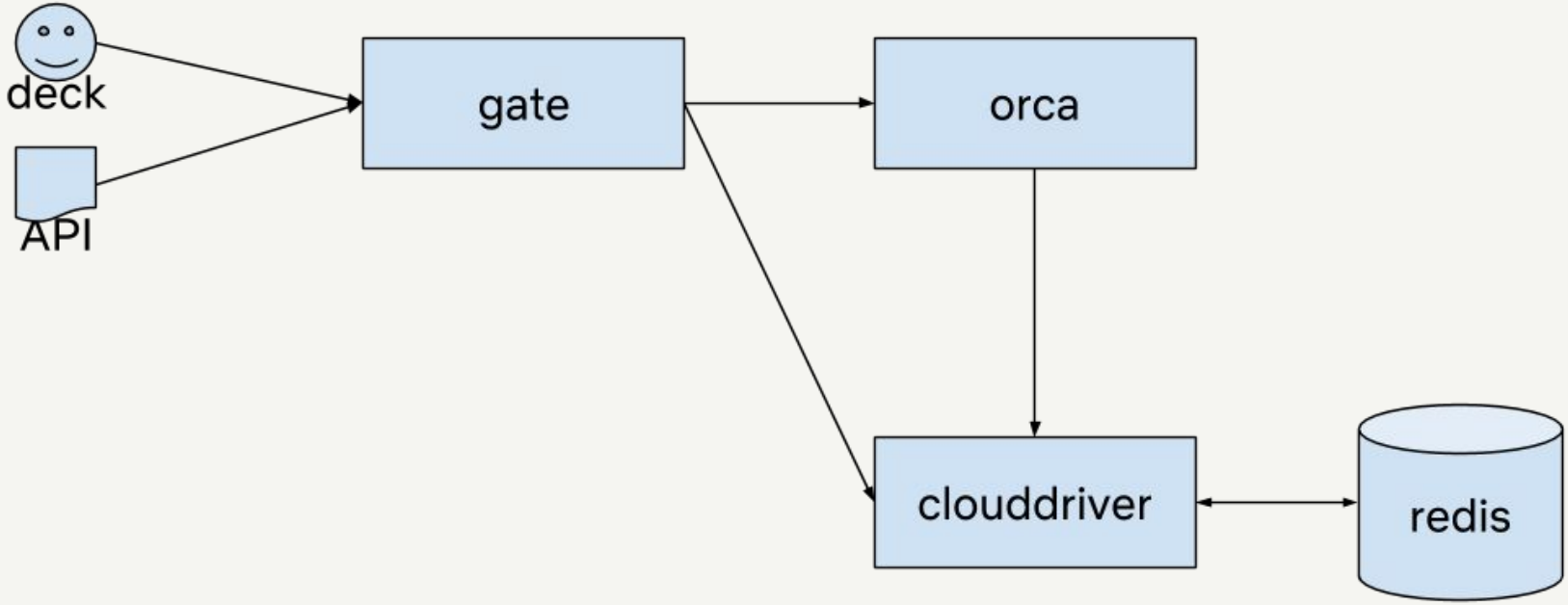
Contents.

- Scaling clouddriver
- Managing Redis
- Scaling clouddriver even more
- Metrics and Monitoring
- Maintenance
- Q&A

Scaling clouddriver.

OUR STARTING STATE AND
FIRST SCALE OUT

Base state of Spinnaker

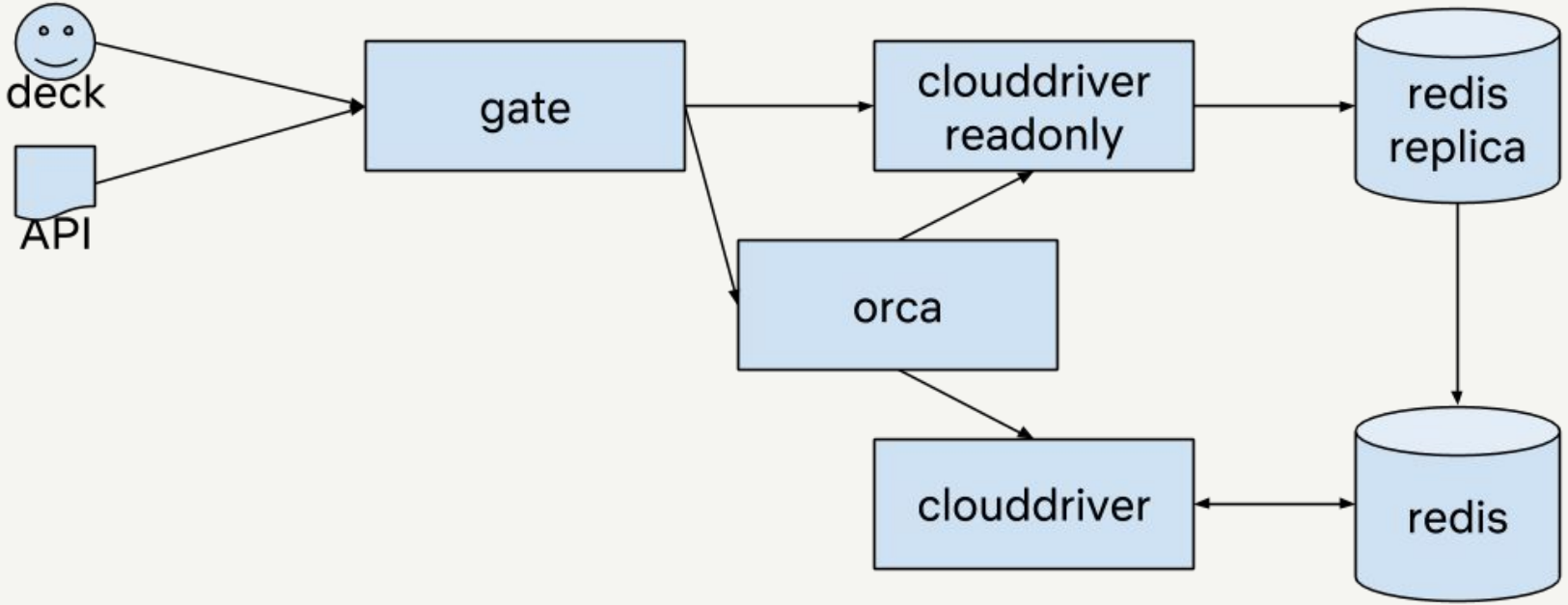


Our first pass at sharding.

We have identified that we have two different workloads

Caching is very write heavy

Serving API requests is very read heavy


deck
API
gate
clouddriver
readonly
redis
replica
orca
clouddriver
redis

What did we just do?

Added a redis read replica.

Added a dedicated **clouddriver-readonly** cluster for that replica.

- New ELB with new unique hostname.
- Caching agents disabled

```
caching.writeEnabled: false
```

gate points API query traffic at **clouddriver-readonly**

```
gate.baseUrl: http://clouddriver-readonly
```

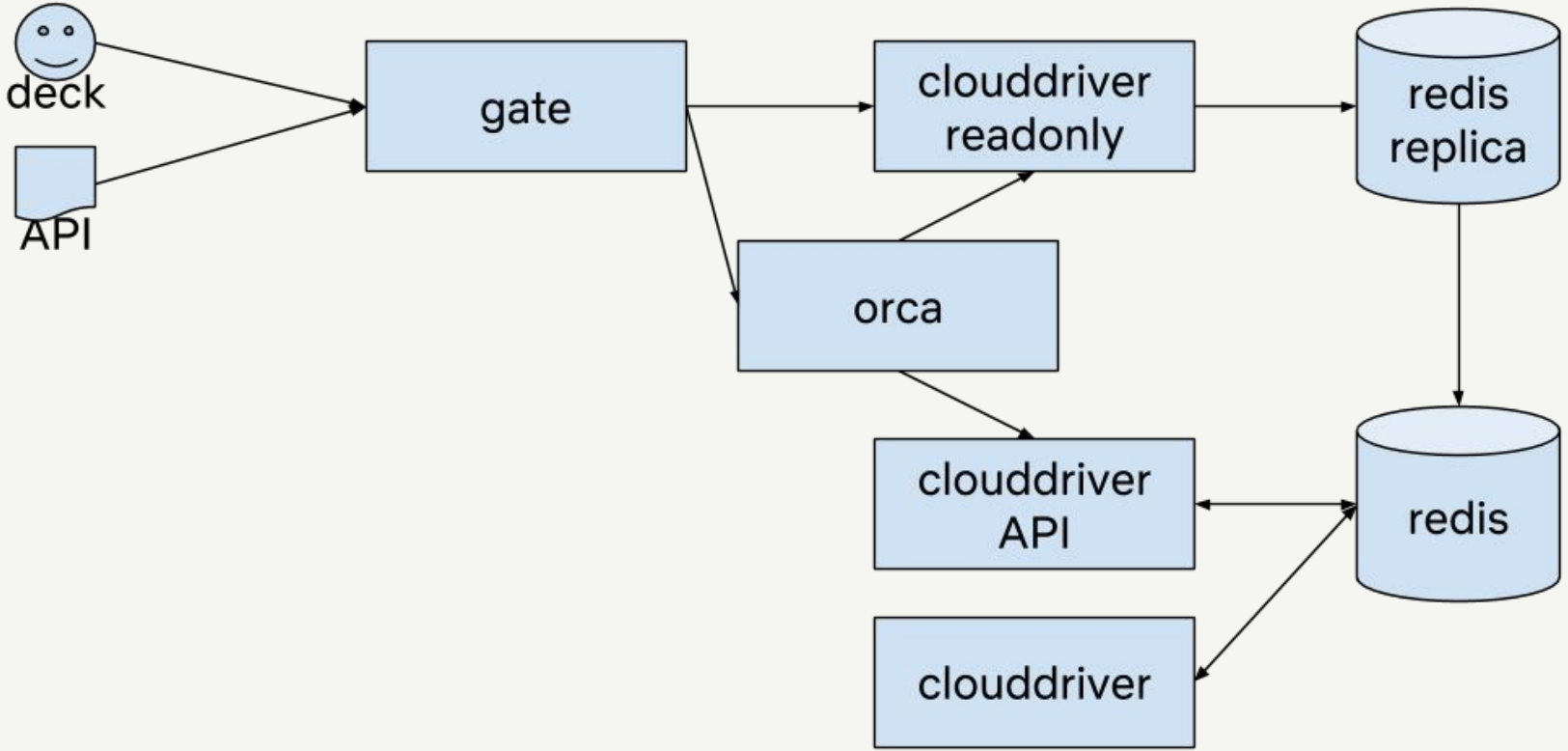
orca monitors orchestration state from **clouddriver-readonly**

```
clouddriver.readonly.baseUrl: http://clouddriver-readonly
```

Partitioning caching agents.

Execution of caching agents is quite memory and CPU intensive.

Execution of these agents on the same instances serving API requests introduces significant variance in request latencies.


deck
API
gate
clouddriver
readonly
redis
replica
orca
clouddriver
API
redis
clouddriver

What did we just do?

Separated caching from operations.

Added a dedicated **clouddriver-api** cluster for that redis master.

- Existing clouddriver ELB.
- Performs cloud operations and force cache refreshes.
- Caching agents disabled

```
caching.writeEnabled: false
```

Made the **clouddriver** cluster caching only.

- No ELB (no callers)
- Tunable caching concurrency for memory sizing

```
redis.agent.maxConcurrentAgents: N
```

Managing redis.

TURNS OUT WE ARE GOING
TO DO A FAIR BIT OF THIS

Why not ElastiCache?

Maintaining stateful services sucks - let someone else do it!

Unfortunately:

- Unpredictable behaviour during maintenance windows.
- Redis throughput triggering sentinel failover.
- Metrics and monitoring mismatch.

Our redis configuration.

Pre-allocated Elastic Network Interfaces (ENIs)
- Tagged by cluster name
- Associated with route53 address

Cluster naming convention
- clouddriver-redis-main as master
- clouddriver-redis-main-replica for replicas

Redis AMI leveraging conventions.

spinredis AMI

Redis on our internal base image.

Support for metric publishing / system metrics.

Redis sidecar app for redis metric publishing
- spring-boot / spectator metrics via redis INFO command
- eureka health status when redis responds to PING

On startup
- Find an ENI matching cluster name and attach it.
- If we are a replica cluster, issue SLAVEOF command

Scaling clouddriver even more.

I HAVE A HAMMER. NOW EVERYTHING LOOKS LIKE A REPLICA PROBLEM

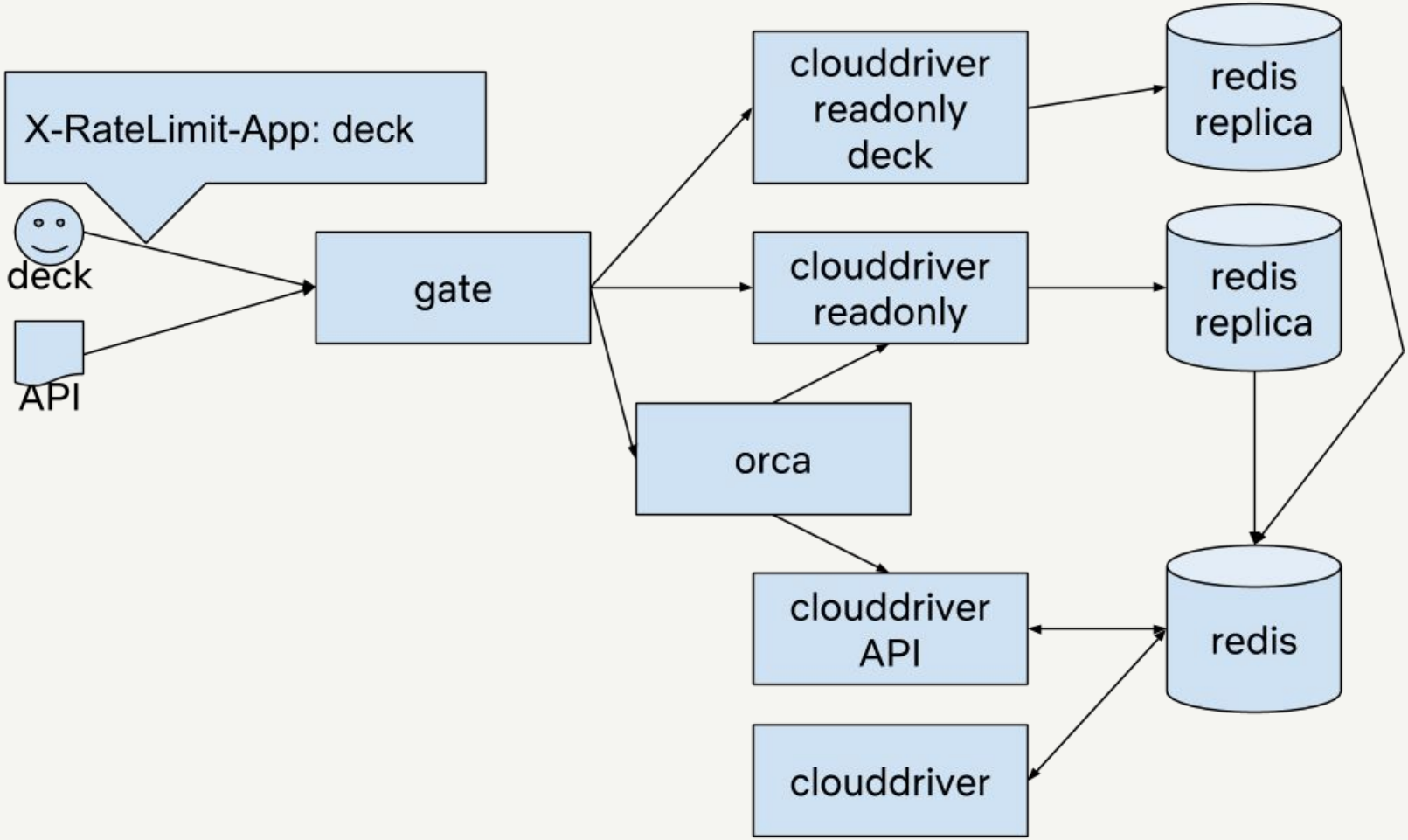
Ensuring
UI responsiveness.

Some traffic patterns can overwhelm the redis read replica.
- Heavy API users.
- Large number of concurrent orchestrations in flight.

Manifests as slow API response times.

Manifests as slow or unresponsive UI.

Manifests as angry users in #spinnaker.


X-RateLimit-App: deck
deck
API
gate
clouddriver
readonly
deck
clouddriver
readonly
orca
clouddriver
API
clouddriver
redis
replica
redis
replica
redis

What did
we just do?

Added a redis read replica dedicated to UI users..

Added a dedicated **clouddriver-readonly-deck** cluster for that replica.
New ELB with new unique hostname.

deck includes an `X-RateLimit-App: deck` request header

gate routes UI query traffic at **clouddriver-readonly-deck**

```
services.clouddriver.config.dynamicEndpoints:
    deck: http://clouddriver-readonly-deck
```

Isolating heavy hitters.

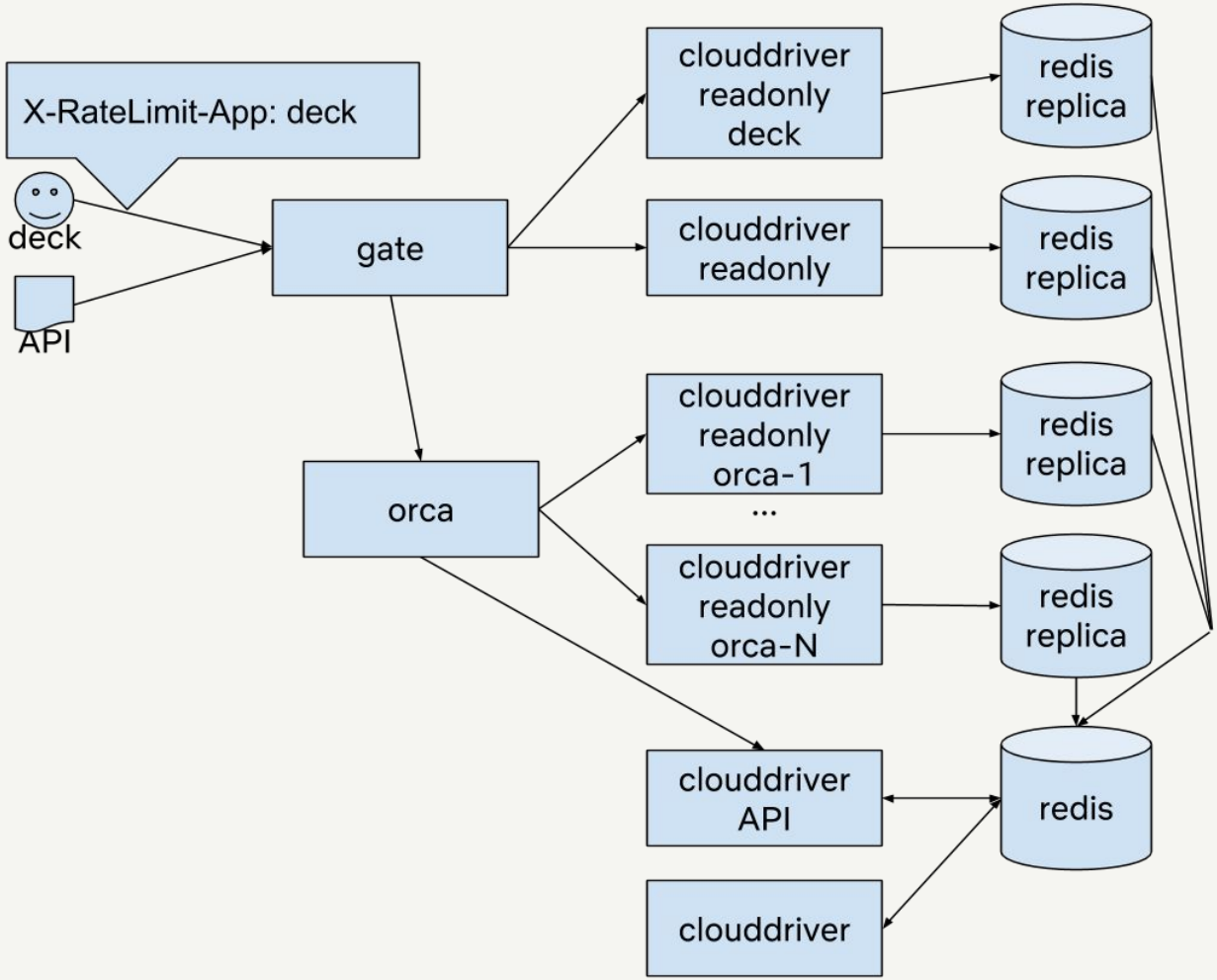
Ensure that

- Large external automation clients.
- Large footprint applications.

Don’t impact the rest of the users of the system.

Maintain client ignorance of sharding strategies

- Don’t want a project management exercise in migrating users
- Want flexibility to adapt strategies in the future


X-RateLimit-App: deck
deck
API
gate
clouddriver
readonly
deck
clouddriver
readonly
orca
clouddriver
readonly
orca-1
...
clouddriver
readonly
orca-N
clouddriver
API
clouddriver
redis
replica
redis
replica
redis
replica
redis
replica
redis

What did
we just do?

Added dedicated redis read replicas dedicated for monitoring orchestration workloads.

Added a dedicated **clouddriver-readonly-orca-N** cluster for each replica.
    New ELB with new unique hostname.

orca routes API traffic via service selectors.

orca
service selectors.

Rule-based routing for clouddriver endpoints.

baseUrl determines the specific endpoint to hit if the rule matches
priority controls evaluation order
selectorClass determines the rule
config configures the rule

Selector types:

- ByExecutionTypeServiceSelector
- ByAuthenticatedUserServiceSelector
- ByOriginServiceSelector
- ByApplicationServiceSelector

orca
service selectors.

```
clouddriver.readonly.baseUrls:
- baseUrl: http://clouddriver-readonly-orca-1
  priority: 10
  config:
    selectorClass: c.n.s.o.c.c.ByApplicationServiceSelector
    applicationPattern: bigapp1|titus.*
- baseUrl: http://clouddriver-readonly-orca-2
  priority: 20
  config:
    selectorClass: c.n.s.o.c.c.ByOriginServiceSelector
    origin: deck
    executionTypes:
      - orchestration
- baseUrl: http://clouddriver-readonly-orca-3
```

Metrics and Monitoring.

IDEALLY NOT OUR USERS
IN #SPINNAKER

Key indicators that we are having a bad time.

Spinnaker is heavily instrumented, and we leverage metrics for alerting.

Some key indicators of clouddriver unhappyness are:

- `controller.invocations` - a timer giving us response timing from apis
- `executionCount[status: failure]` - counter of failed caching agent execution
- `redis.replication.slaveDelta` - bytes offset from the master

Maintenance.

WHAT DO YOU MEAN I HAVE
TO APPLY SECURITY PATCHES?

Stateful systems are hard.

We built this monstrosity of redis and redis replicas.

Now we need a way to upgrade all those nodes to keep them current.

Also we don’t have any window when someone is going to be okay with us being offline…

I know!
Lets add more redii.

Support the ability to deploy onto a new redis stack.

Read from the old stack to ensure we don’t lose any work in progress.

We update our cluster naming convention to include two stacks: -a and -b.

At any one time, one of those stacks is active, and the other we can deploy onto.

Both of those stacks share the same ELBs.

Enter the flippy-floppy-flush.

Steady state

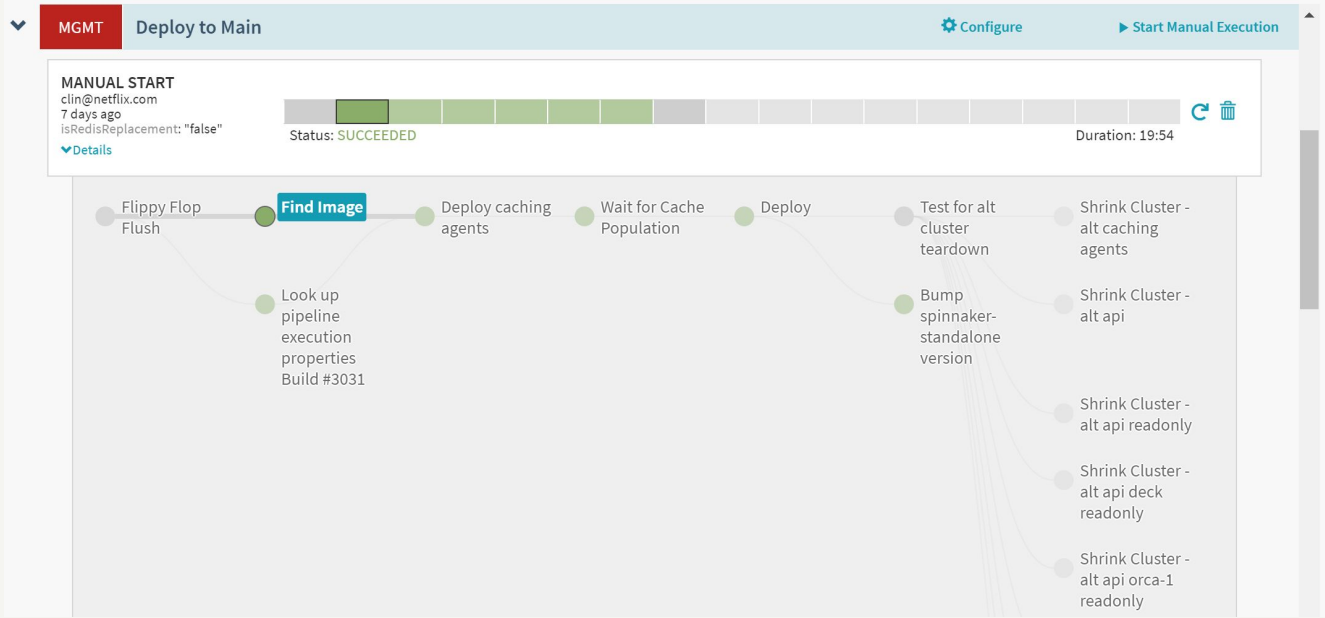
Simple red/black in the existing active redis cluster

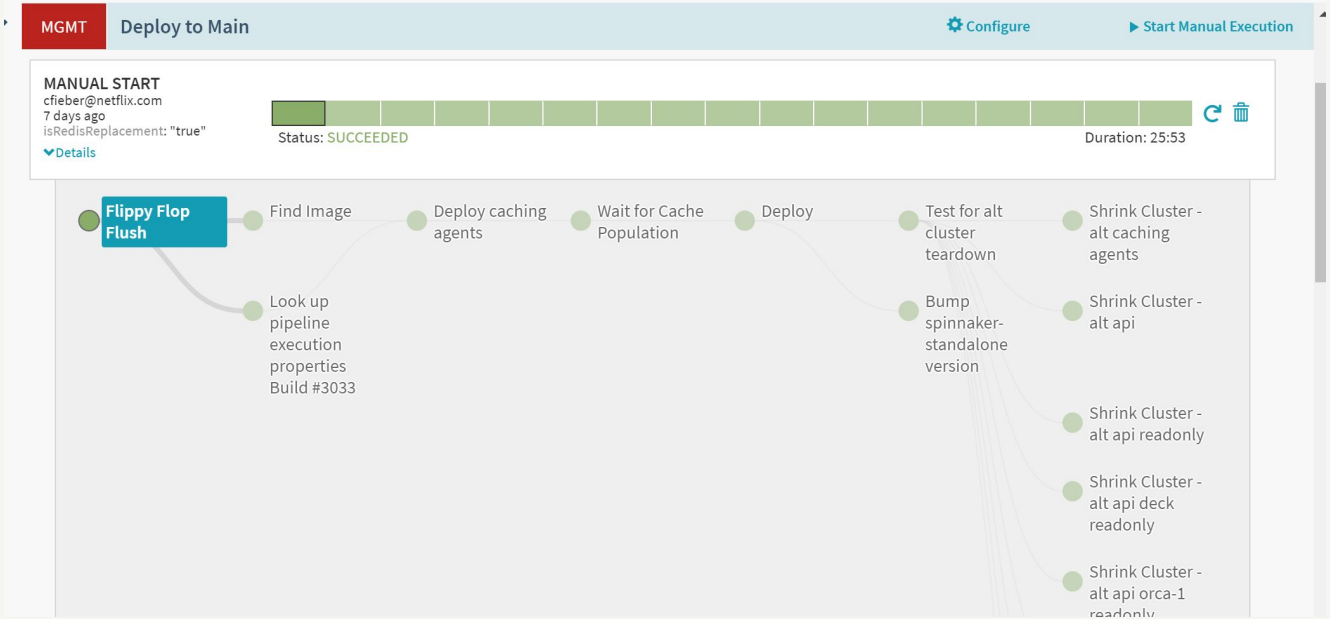
Redis maintenance (or redis emergency)

Flip onto the other redis stack

Scripting queries the existing cluster to figure out which is active.

Parameterizes the deployment pipeline to indicate where to deploy.


MGMT
Deploy to Main
Configure
Start Manual Execution
MANUAL START
clin@netflix.com
7 days ago
isRedisReplacement: "false"
Details
Status: SUCCEEDED
Duration: 19:54
Flippy Flop Flush
Find Image
Deploy caching agents
Wait for Cache Population
Deploy
Test for alt cluster teardown
Shrink Cluster - alt caching agents
Look up pipeline execution properties Build #3031
Bump spinnaker-standalone version
Shrink Cluster - alt api
Shrink Cluster - alt api readonly
Shrink Cluster - alt api deck readonly
Shrink Cluster - alt api orca-1 readonly


MGMT
Deploy to Main
Configure
Start Manual Execution
MANUAL START
cfieber@netflix.com
7 days ago
isRedisReplacement: "true"
Details
Status: SUCCEEDED
Duration: 25:53
Flippy Flop Flush
Find Image
Look up pipeline execution properties Build #3033
Deploy caching agents
Wait for Cache Population
Deploy
Test for alt cluster teardown
Bump spinnaker-standalone version
Shrink Cluster - alt caching agents
Shrink Cluster - alt api
Shrink Cluster - alt api readonly
Shrink Cluster - alt api deck readonly
Shrink Cluster - alt api orca-1 readonly

Questions?

Thank you!

NETFLIX