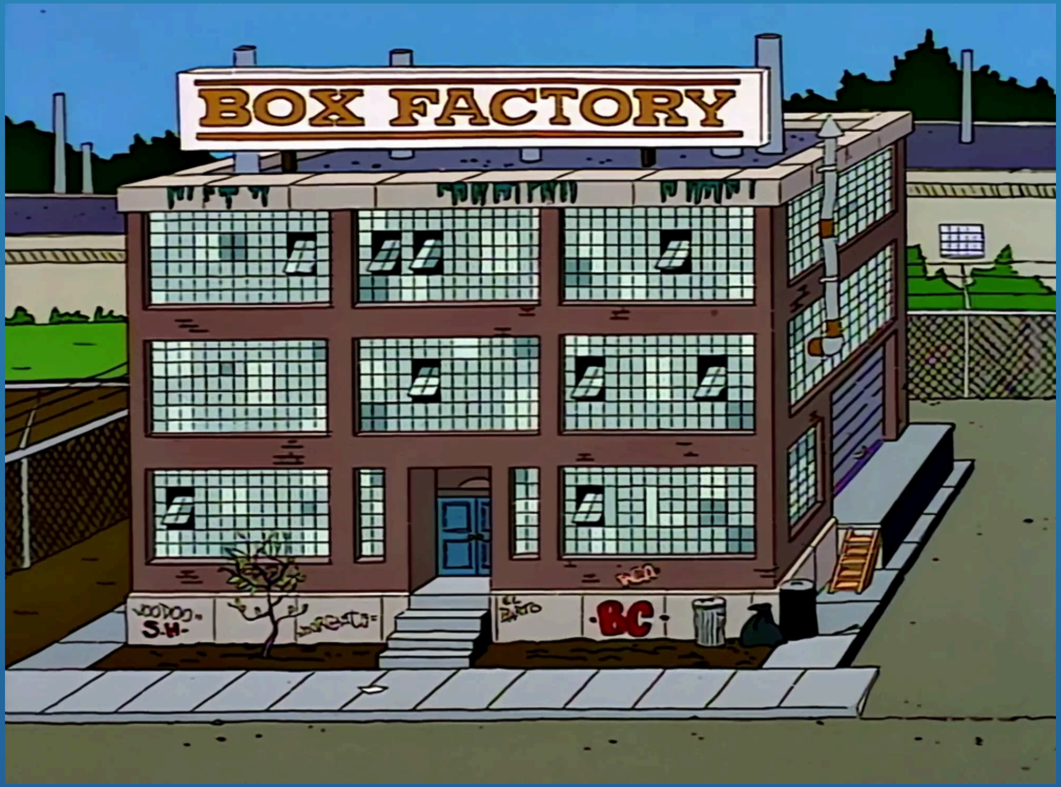
The operator antipattern

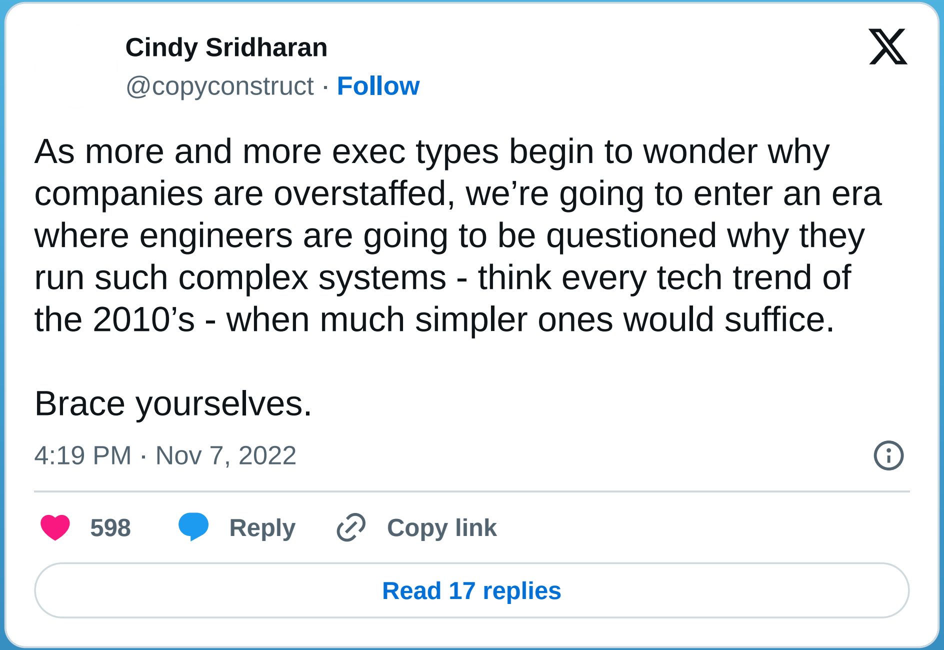
Kubernetes Community Days London 2024

Gerald Schmidt

DS Smith



Source: simpsons.fandom.com/wiki/Springfield_Box_Factory


Cindy Sridharan
@copyconstruct · Follow

As more and more exec types begin to wonder why companies are overstaffed, we're going to enter an era where engineers are going to be questioned why they run such complex systems - think every tech trend of the 2010's - when much simpler ones would suffice.

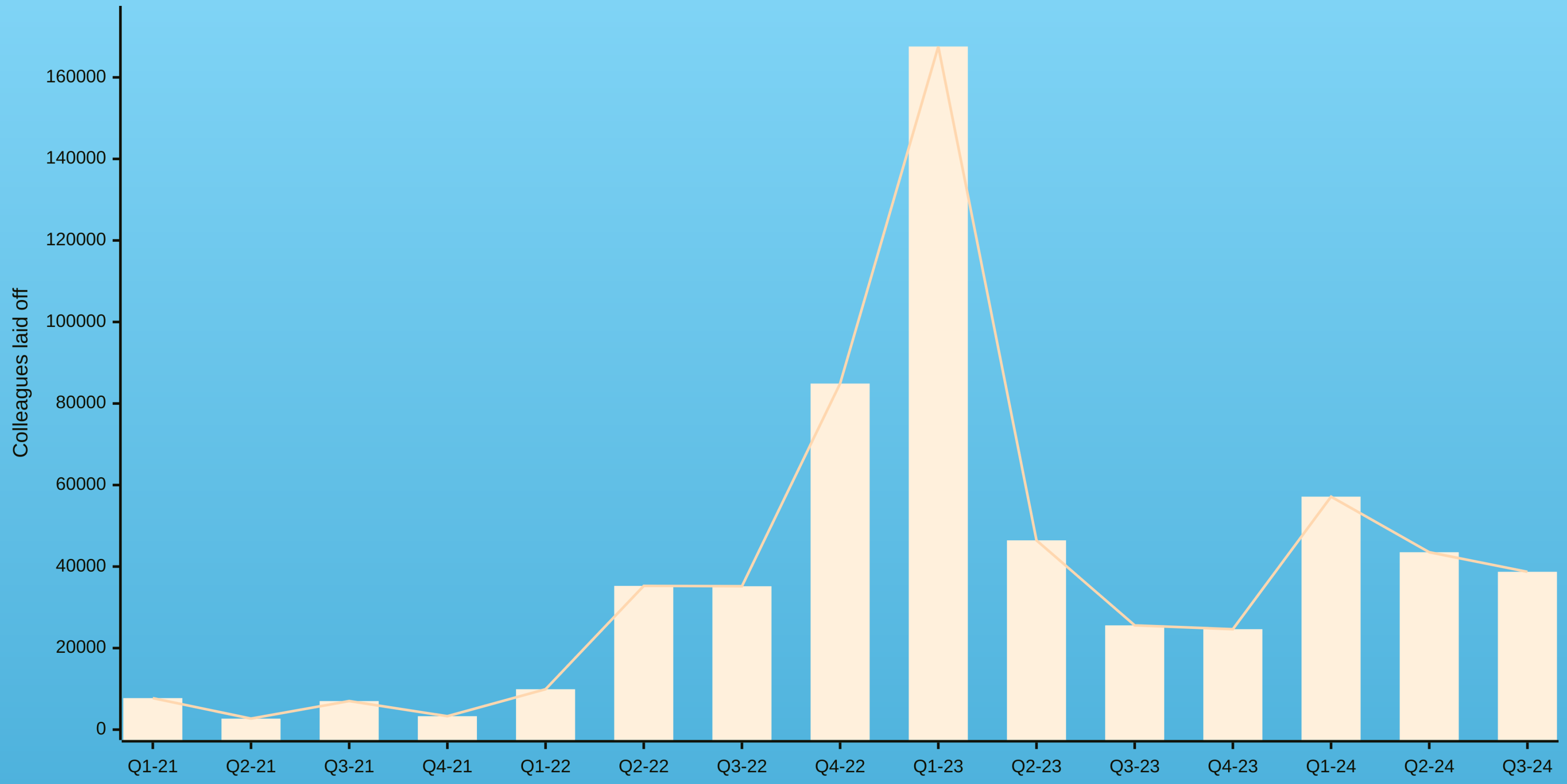
Brace yourselves.

4:19 PM · Nov 7, 2022

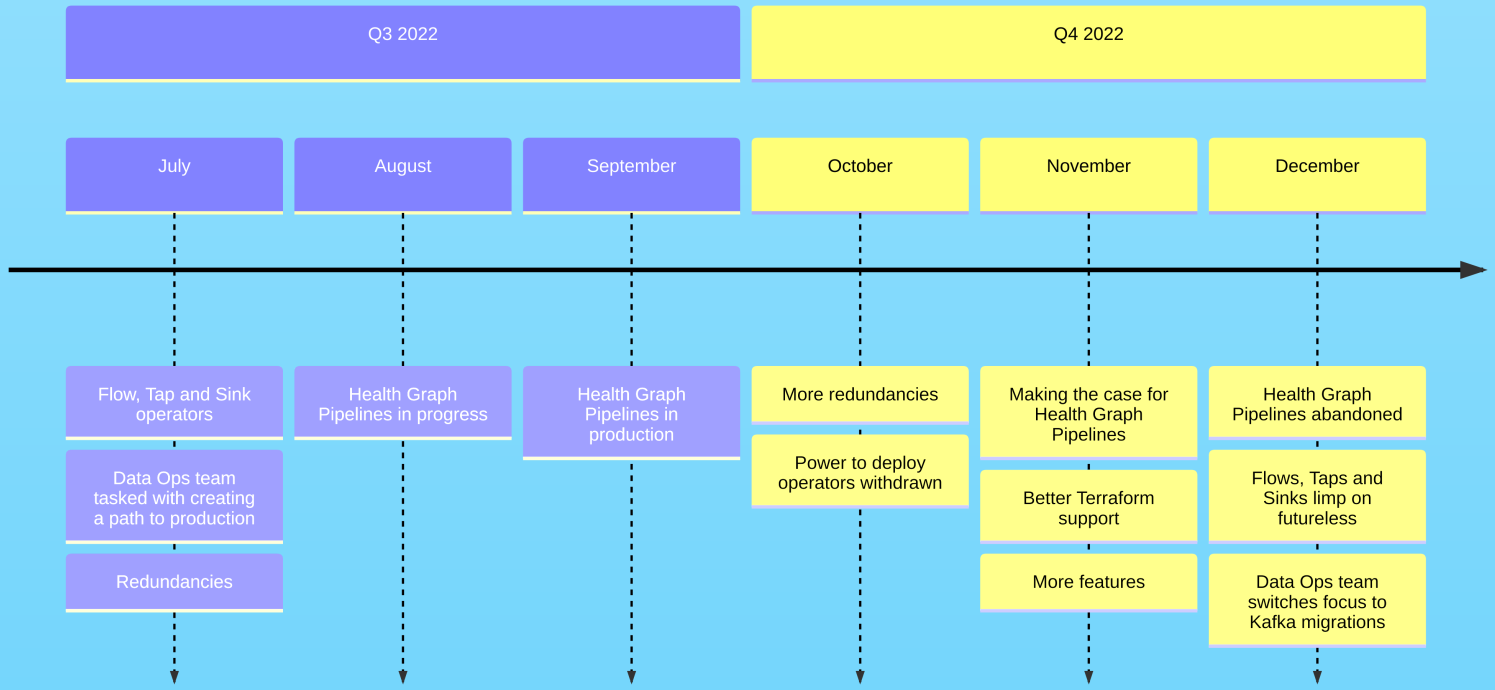
598 Reply Copy link

Read 17 replies

Classic Sridharan:
Testing Microservices, the sane way (2017) | Testing in Production, the safe way (2018) | Testing in Production: the hard parts (2019)



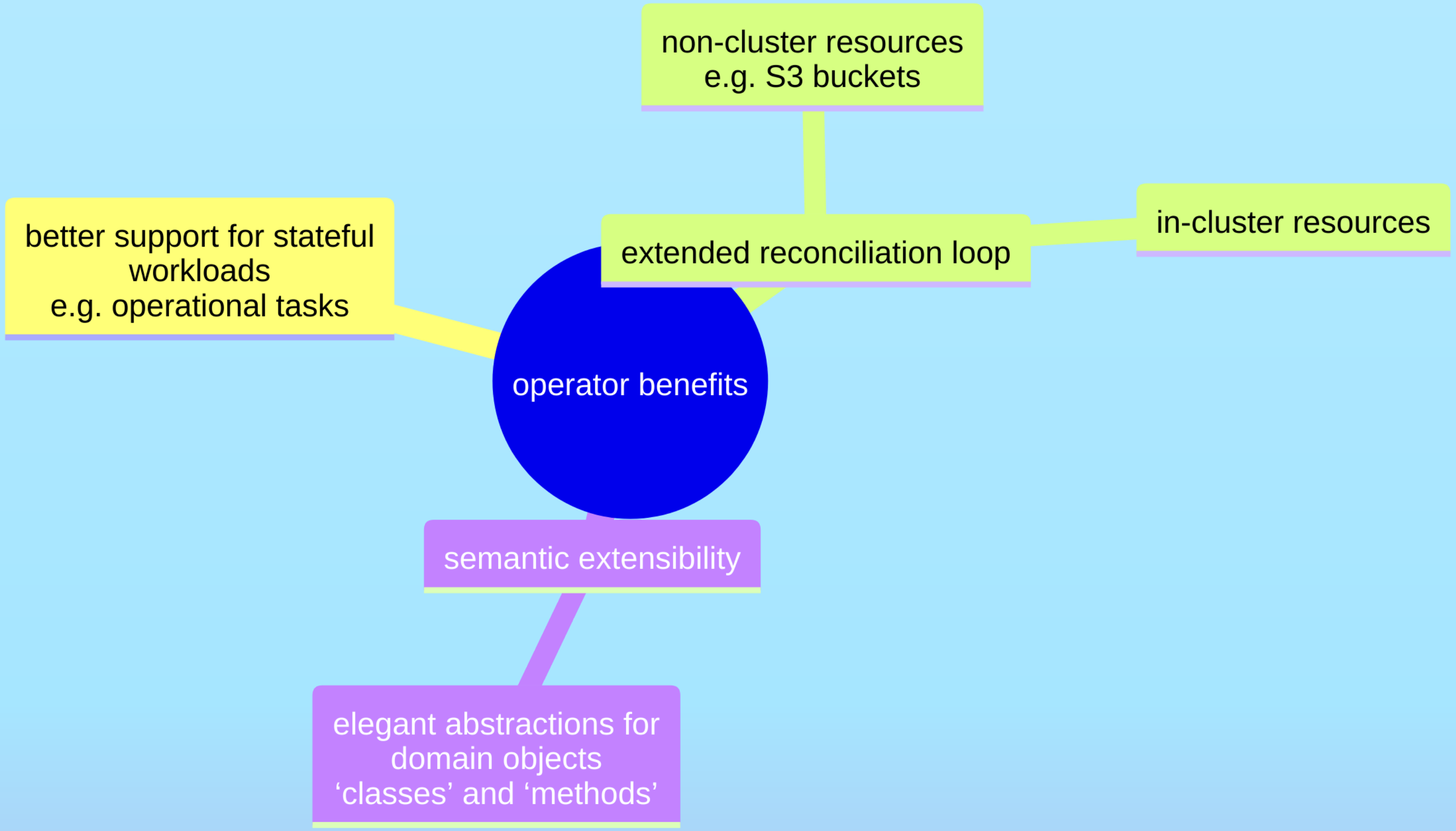
Source: layoffs.fyi


Q3 2022
Q4 2022
July
August
September
October
November
December
Flow, Tap and Sink operators
Data Ops team tasked with creating a path to production
Redundancies
Health Graph Pipelines in progress
Health Graph Pipelines in production
More redundancies
Power to deploy operators withdrawn
Making the case for Health Graph Pipelines
Better Terraform support
More features
Health Graph Pipelines abandoned
Flows, Taps and Sinks limp on futureless
Data Ops team switches focus to Kafka migrations

The premise

An Operator is an application-specific controller that extends the Kubernetes API to create, configure, and manage instances of complex stateful applications on behalf of a Kubernetes user. It builds upon the basic Kubernetes resource and controller concepts but includes domain or application-specific knowledge to automate common tasks.

Brandon Philips, Introducing Operators: Putting Operational Knowledge into Software (2016)


operator benefits
better support for stateful workloads
e.g. operational tasks
extended reconciliation loop
non-cluster resources
e.g. S3 buckets
in-cluster resources
semantic extensibility
elegant abstractions for domain objects
‘classes’ and ‘methods’

The CRD/controller split

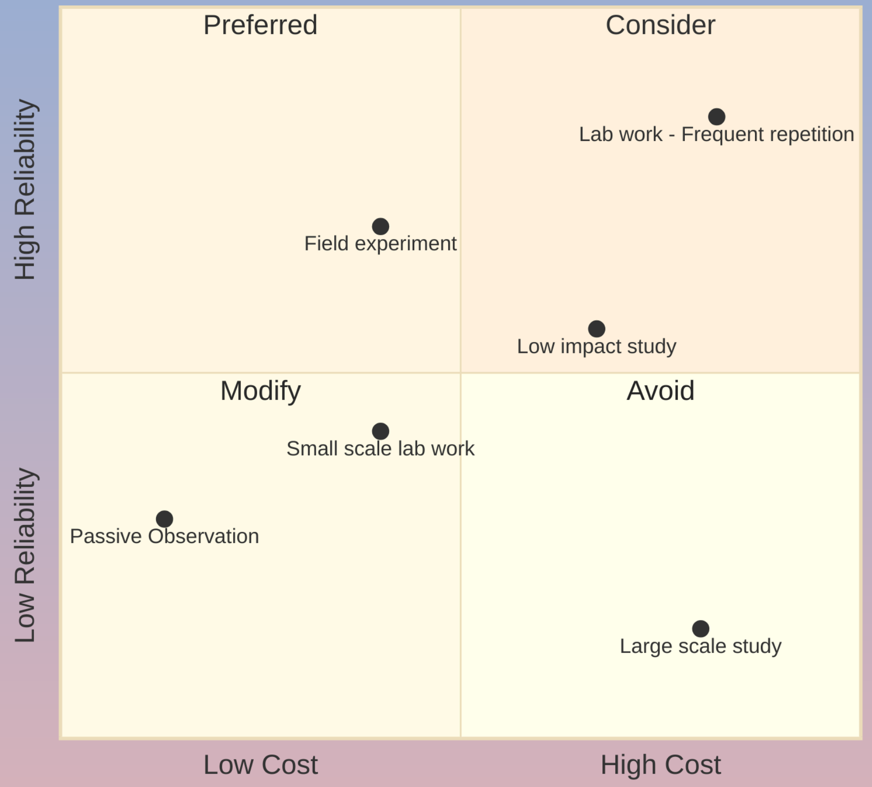
Weighting

The benefits of operators centre on the controller.

The custom resource component is to a meaningful extent syntactic sugar.

At the same time it is responsible for most of the problematic side-effects of operators: custom resource definitions are cluster-level objects, which brings with it a whole set of permission issues. Versioning is another problematic aspect that we’ll come back to.

Cost and Results of experiments



Prometheus without operator

```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/port: "2112"
    prometheus.io/scrape: "true"
```

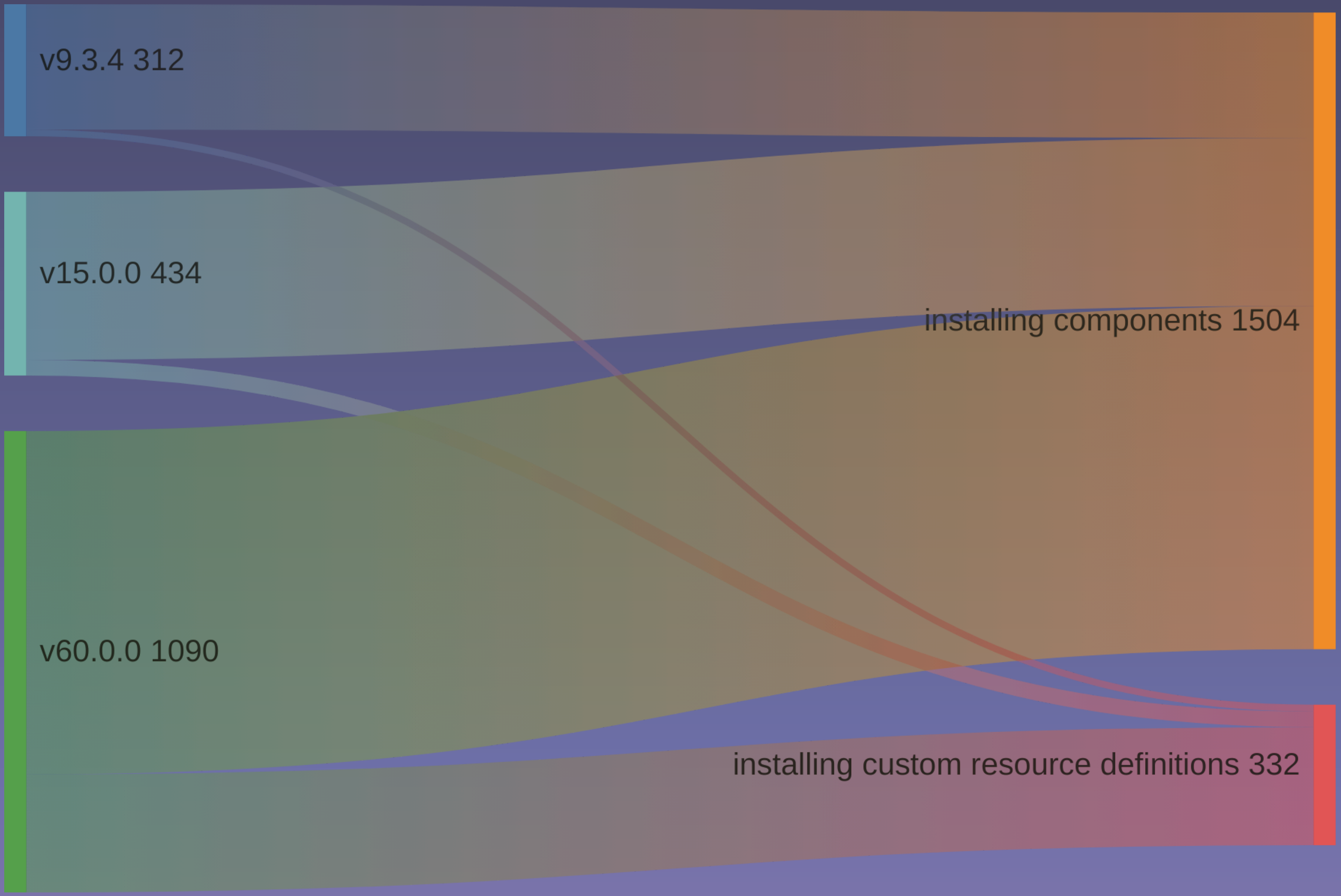
Prometheus with operator

```
$ kubectl get crd -o custom-columns=NAME:.metadata.name \
  | grep "monitoring\.coreos"
alertmanagerconfigs.monitoring.coreos.com
alertmanagers.monitoring.coreos.com
podmonitors.monitoring.coreos.com
probes.monitoring.coreos.com
prometheusagents.monitoring.coreos.com
prometheuses.monitoring.coreos.com
prometheusrules.monitoring.coreos.com
scrapeconfigs.monitoring.coreos.com
servicemonitors.monitoring.coreos.com
thanosrulers.monitoring.coreos.com
```

Kube Prometheus stack

```
FILE=charts/kube-prometheus-stack/README.md
for TAG in "9.3.4" "15.0.0" "60.0.0"; do
  git checkout "kube-prometheus-stack-${TAG}" 2>/dev/null
  echo "# ${TAG}"
  head -n -100 "${FILE}" | wc -l charts/kube-prometheus-stack/README.md
  grep -v "\(^kubectl .*crd\|CRD\)" "${FILE}" | wc -l
done
```

Kube-prometheus-operator



Antipattern 1: operators in developer workflows

Flow operator

Antipattern 2: tight coupling with external resources

Strimzi

Antipattern 3: versioning trouble

Incrementing CRD versions is a serious matter.

Is the old version still served? Have we provided a conversion webhook?

So far from reducing complexity, we are introducing new error conditions, failure modes and edge cases.

See AWS Controllers for Kubernetes. (Still on alpha.)

Antipattern 4: overpromising

The association of operators with complex *stateful* applications has not displaced managed databases such as the Relational Database Service. The CRD that allows me to create a VectorDatabase resource does not magically make it a good, fault-tolerant. A backup method is helpful and appreciated, but it does not rival a mature point-in-time recovery facility.

Whisper it: Kubernetes does not have a ‘stateful workload’ problem. It has a persistent volume problem. The solution is object storage, and the challenge is working around the limitations of object storage when it comes to read and write speed.

See Object storage for stateful applications on Kubernetes

Kyverno

Wins first prize for an implementation that feels as if it should be an in-tree policy engine. Policy violations create detailed events and the new resources (Policy, ClusterPolicy) fit well into the existing set of resources.

Controller revival

Grafana has bucked the trend of CRD sprawl.

To load a dashboard on startup, Grafana seeks out ConfigMaps that have label `grafana_dashboard` set to value `1`.

There is no need for a GrafanaDashboard CRD.

Grafana dashboards are JSON objects following a well-established structure.

Teams store dashboards they wish to keep in a folder:

```
for DASHBOARD in \
  $(ls kube-prometheus-stack/dashboards/*.json)
do
  CONFIGMAP=$(basename "${DASHBOARD}" | cut -d'.' -f1)
  kubectl create configmap "${CONFIGMAP}" \
    -n monitoring \
    --dry-run=client \
    --from-file="${DASHBOARD}" -o yaml | \
    kubectl apply -f -
  kubectl label configmap "${CONFIGMAP}" \
    -n monitoring \
    --overwrite grafana_dashboard="1"
done
```

gerald1248/operator-antipattern-slides
www.linkedin.com/in/gerald1248
03spirit
Slides built with Markdeck

DS Smith