

## Abstract

cloud native, need for resiliency, service meshes as solution, demo, pros of istio – resiliency and fault injection

## Motivation

docker, adoption of containers, migration to clouds → microservices, devops, fast code-to-market, leave only business logic for developers

number of microservices grows, lack of visibility and control,

kubernetes, no possibility to deal with network errors – focus on pods

goals, metrics: deploy microservices app, compare resiliency with and without istio

cc project as template (refactored, adopted), deploy istio, demo in minikube, test resiliency

## Related work

other service meshes/libraries, pros/cons, trend

- libs: cons - code change (hystrix, ribbon)
- node agent (linkerd)
- sidecar (istio, linkerd2, consul)

## Major idea

**take uni cloud computing project** and deploy it with istio, because I wanted to try out istio on something, that I coded myself and not “hello world” projects or some that are given in istio documentation (Bookinfo App).

**The Twelve Factors App**, build working demo to play around, changes made, resiliency in project, no down time, user satisfaction, easy to monitor, screenshots, cpanel v1/v2

- architecture, API, REST, diagrams
- k8s to deploy
- docker – runtime / packaging
- istio as service mesh

describe **istio**, features, API installed as kubernetes CRD

- service mesh
  - security – who talks whom, trusted communication
  - observability – tracing / metrics
- architecture
  - data plane – traffic routing
  - control plane – tls, policies

**pilot** – get rules and send them to proxies, works dynamically on the fly, without restart needed, looks into all registries in system and understands topology of deployment, uses service discovery adapter (k8s, consul)

**mixer** – take telemetry to analyze, has policies, all side cars calls mixer, if request is allowed, quotas, authZ backends, turns data into info → high cpu load, has caching → not single point of failure

**citadel** – certificates mTLS

**galley** – holds configs

Manifests:

Virtual services – route traffic (headers, weight, URL), retries, timeouts, fault injection

destination rules – named subsets, circuit breaker, load balancing

gateways – Virtual Service to allow L7 routing, use default or deploy own

- ingress – to expose service with kubernetes
- egress – by default all external traffic is blocked, enabled in Service entry

Service entry – automatic from pilot, from k8s - service names and ports,

Kiali – visualize services that are deployed

Grafana with prometheus as backend

- runs in its own namespace – isolated from other procs
- fault injection:
  - http error codes, eg 400
  - delays
- resiliency possibilities:
  - + Client-side load balancing
  - Timeout – virt svc, default = 15 sec
  - Retry – virt svc, default = NO
  - circuit breaker – dest rule – LB least request
  - Pool ejection
  - + Health checks
  - Outlier Detection
- **demo:** git repo, minikube (specific versions), istio, shell scripts, makefile

## Implementation

### The Twelve Factors App

#### 1. Codebase

One codebase tracked in revision control, many deploys - **GitHub**

#### 2. Dependencies

Explicitly declare and isolate dependencies - **requirements.txt**

#### 3. Config

Store config in the environment - **env variables**

#### 4. Backing Services

Treat backing services as attached resources – **NO (json) or mount volume. It is recommended to use databases.**

#### 5. Build, release, run

Strictly separate build and run stages – **docker images with env vars and versions**

#### 6. Processes

Execute the app as one or more stateless processes – **Docker**

#### 7. Port binding

Export services via port binding - **completely self-contained, exports HTTP as a service by binding to a port, unicorn**

#### 8. Concurrency

Scale out via the process model – **LB with docker containers**

#### 9. Disposability

Maximize robustness with fast startup and graceful shutdown - **Docker**

#### 10.Dev/Prod parity

Keep development, staging, and production as similar as possible - **Docker**

#### 11.Logs

Treat logs as event streams – **logs to stdout**

#### 12.Admin Processes

**Run admin/management tasks as one-off processes - ???**

- **refactor** and **expanse** of cc project
  - 12 factor
  - unit tests
  - frontend v1/v2

- canary, blue/green deployment, user resiliency
- python + docker best practices:
  - gunicorn, root, alpine, no cache
- scaling deployment:
  - collector, image-analysis, face-recognition
- **k8s:**
  - services – fqdn, service discovery
  - deployments with pods
  - readiness/liveness - resiliency
  - resources limits – to protect pods from starvation

Istio:

istio verify install done in script

single cluster deployment

virtual services , destination rules, ingress

fault injection: delays and aborts, retries, timeouts, circuit braking

best practices: add dest rules and virt svc for all microservices

- **how to run:** github, virtualbox, curl, docker, shell scripts, yaml, minikube with kubectl, istio, install requirements (ram, cpu),
  - dirty tricks:
    - sharing containers host/guest minikube
- **how to play around:**

## Evaluation and Discussion

- demo with Makefile, tests - screenshots/results
- comparison to k8s only

Kubernetes has only round robin load balancing. Istio with the help of destinations rules extends native kubernetes load balancing and presents the following types: random, round robin, weighted least request, ring hash (#istio). In such a case istio can give any microservice replica set it's own load balancer. To show how istio load balancing can be configured, we need first to learn about routing mechanism provided by istio.

## Istio routing mechanism

This solution can be used to make canary deployments and also make user experience more resilient - “user resilience”. For example, new version of service can be made available only to one group of users (test group). It can be as much as only 1% of of the hole traffic. Users can be filtered by headers in http request. If something goes wrong with new version of service it is very easy to rollback and switch all the traffic back to production version.

This mechanism allows also to do blue/green deployments.

\$ make deploy-app-default

\$ make deploy-istio-default

\$ make health

```
curl http://192.168.99.113:31221/status
CPanel v1 : Online
curl http://192.168.99.113:31221/cameras/1/state
{"streaming":false,"cycle":88,"fps":0,"section":"1","destination":"http://collector.default.svc.cluster.local:8080","event":"exit"}
curl http://192.168.99.113:31221/cameras/2/state
{"streaming":false,"cycle":92,"fps":0,"section":"1","destination":"http://collector.default.svc.cluster.local:8080","event":"entry"}
curl http://192.168.99.113:31221/collector/status
Collector v1 : Online
curl http://192.168.99.113:31221/alerts/status
Alerts v1 : Online
curl http://192.168.99.113:31221/sections/1/status
Section 1 v1 : Online
```

\$ make start-cameras

\$ make health

```
curl http://192.168.99.113:31221/status
CPanel v1 : Online
curl http://192.168.99.113:31221/cameras/1/state
{"streaming":true,"cycle":7,"fps":0,"section":"1","destination":"http://collector.default.svc.cluster.local:8080","event":"exit"}
curl http://192.168.99.113:31221/cameras/2/state
{"streaming":true,"cycle":5,"fps":0,"section":"1","destination":"http://collector.default.svc.cluster.local:8080","event":"entry"}
curl http://192.168.99.113:31221/collector/status
Collector v1 : Online
curl http://192.168.99.113:31221/alerts/status
Alerts v1 : Online
curl http://192.168.99.113:31221/sections/1/status
Section 1 v1 : Online
```

curl http://192.168.99.113:31221/status

CPanel v1 : Online

curl http://192.168.99.113:31221/cameras/1/state

{"streaming":true,"cycle":7,"fps":0,"section":"1","destination":"http://collector.default.svc.cluster.local:8080","event":"exit"}

curl http://192.168.99.113:31221/cameras/2/state

{"streaming":true,"cycle":5,"fps":0,"section":"1","destination":"http://collector.default.svc.cluster.local:8080","event":"entry"}

curl http://192.168.99.113:31221/collector/status

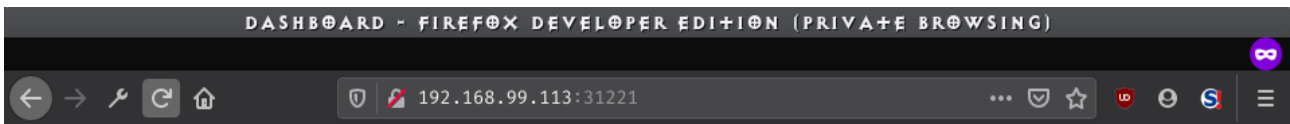
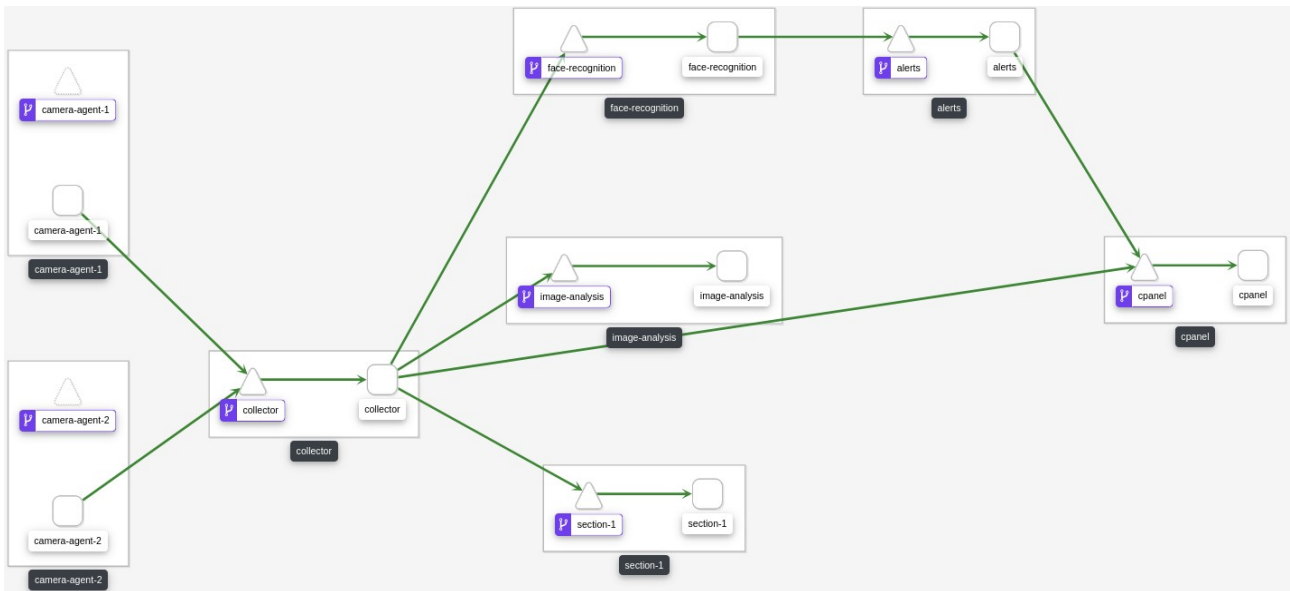
Collector v1 : Online

curl http://192.168.99.113:31221/alerts/status

Alerts v1 : Online

curl http://192.168.99.113:31221/sections/1/status

Section 1 v1 : Online



## Dashboard V1

### Section 1

timestamp: 2020-02-25T14:35:38.204522Z

gender: male | age: 38-43 | event: exit

### Alert

timestamp: 2020-02-25T14:35:27.224857Z

section: 1

event: entry

name: **PersonX**

Default app with cpanel v1 without load:



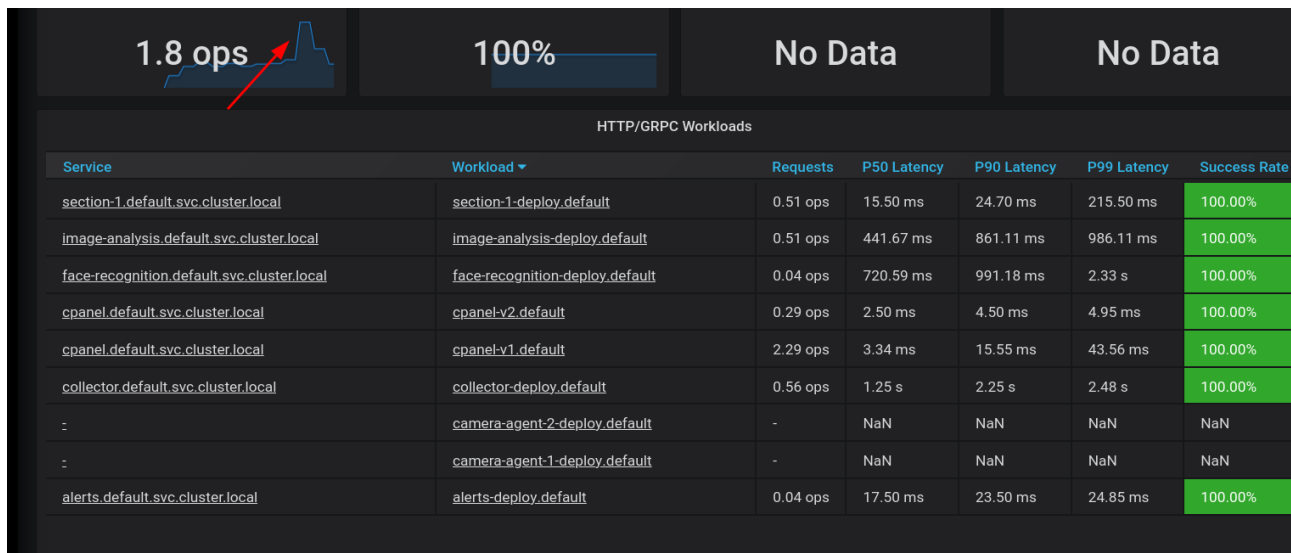
\$ make load

for i in {1..100}; do sleep 0.2; curl http://192.168.99.113:31221/status; printf "\n"; done

CPanel v1 : Online

CPanel v1 : Online



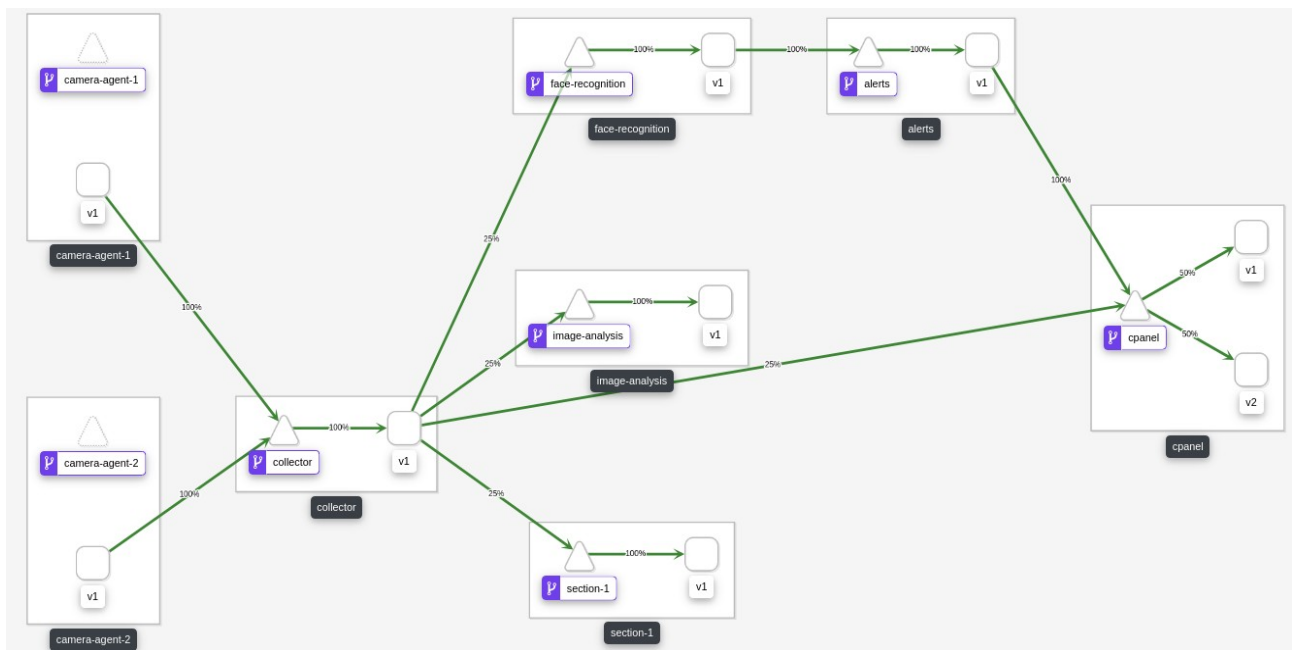


```
$ make cpanel-50-50
./kubectl apply -f istio/virt_svc_50-50.yaml
virtualservice.networking.istio.io/cpanel configured
check configuration
$ k get virtualservices cpanel -o yaml
```

route:

- destination:
  - host: cpanel.default.svc.cluster.local
  - port:
    - number: 8080
    - subset: v1
  - weight: 50
- destination:
  - host: cpanel.default.svc.cluster.local
  - port:
    - number: 8080
    - subset: v2
  - weight: 50

```
$ make start-cameras
```

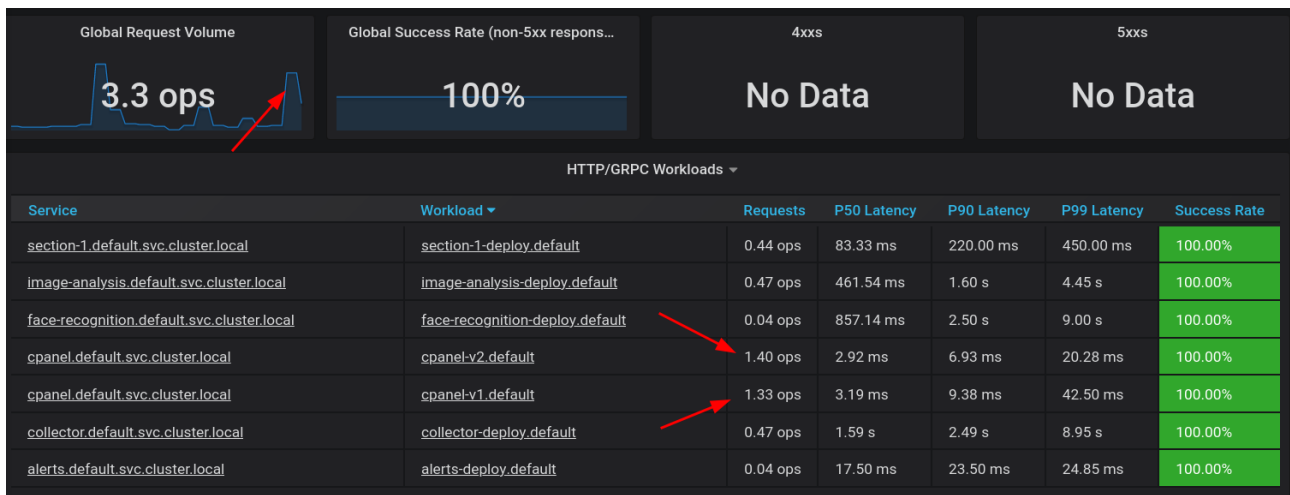


```

$ make load
for i in {1..100}; do sleep 0.2; curl http://192.168.99.113:31221/status; printf "\n"; done
CPanel v1 : Online
CPanel v1 : Online
CPanel v1 : Online
CPanel v2 : Online
CPanel v2 : Online
CPanel v2 : Online

```





```
$ make cpanel-v2
./kubectl apply -f istio/virt_svc_v2.yaml
virtualservice.networking.istio.io/cpanel configured
check configuration
$ k get virtualservices cpanel -o yaml
```

```
route:
- destination:
  host: cpanel.default.svc.cluster.local
  port:
    number: 8080
  subset: v1
  weight: 0
- destination:
  host: cpanel.default.svc.cluster.local
  port:
    number: 8080
  subset: v2
  weight: 100
```

```
$ make start-cameras
```

## Dashboard V2

### Section 1



timestamp: 2020-02-25T15:27:21.900453Z

gender: male | age: 25-32 | event: entry

gender: male | age: 25-32 | event: entry

### Alert



timestamp: 2020-02-25T15:26:55.022111Z

section: 1

event: exit

name: **George W**

\$ make load

for i in {1..100}; do sleep 0.2; curl http://192.168.99.113:31221/status; printf "\n"; done

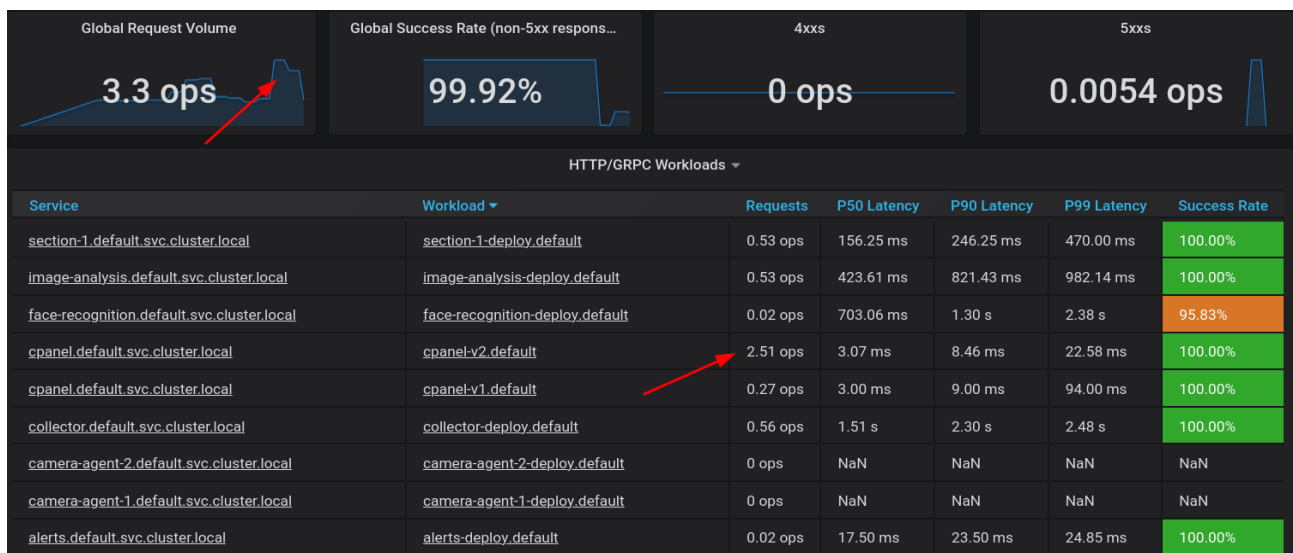
CPanel v2 : Online

CPanel v2 : Online

CPanel v2 : Online

CPanel v2 : Online

CPanel v2 : Online



### Load balancing

Default round robin between v1 and v2 cpanel (should be 1:3)

\$ make scale\_v2\_x3

```
kubectl scale deployment cpanel-v2 --replicas=3
collector-deploy-558dd7dd45-8rlwq      2/2   Running   3      9h
cpanel-v1-8446d9dd45-wx6mz             2/2   Running   2      9h
cpanel-v2-8445ff5964-lgj84             1/2   Running   0      6s
cpanel-v2-8445ff5964-qdhk8             0/2   Running   0      6s
cpanel-v2-8445ff5964-r4r2d             2/2   Running   3      9h
face-recognition-deploy-7b954c454-fdphg 2/2   Running   3      9h
```

Here we can see how kubernetes scales our service.

```
$ make load_balancing
./kubectl apply -f istio/round_robin.yaml
```

```
route:
- destination:
    host: cpanel.default.svc.cluster.local
    port:
        number: 8080
```

```
$ make load
for i in {1..100}; do sleep 0.2; curl http://192.168.99.113:31221/status; printf "\n"; done
CPanel v2 : Online
CPanel v2 : Online
CPanel v2 : Online
CPanel v2 : Online
CPanel v2 : Online
CPanel v2 : Online
CPanel v1 : Online
CPanel v2 : Online
CPanel v1 : Online
```

```
$ make random
./kubectl apply -f istio/random_lb.yaml
destinationrule.networking.istio.io/cpanel configured
$ k get destinationrules cpanel -o yaml
```

```
$ make load
for i in {1..100}; do sleep 0.2; curl http://192.168.99.113:31221/status; printf "\n"; done
CPanel v2 : Online
CPanel v2 : Online
CPanel v1 : Online
CPanel v2 : Online
CPanel v1 : Online
CPanel v2 : Online
CPanel v1 : Online
CPanel v2 : Online
```

```
$ make all-reset
./kubectl delete service --all
```

### **fault injection**

Internal istio mechanism for chaos testing. Allows simulating network and service errors without touching the source code of microservice at all. All faults are done by sidecar proxy.

```


http:
- fault:
  abort:
    httpStatus: 503
    percentage:
      value: 50
route:
- destination:
  host: alerts.default.svc.cluster.local
  subset: v1

route:
- destination:
  host: cpanel.default.svc.cluster.local
  port:
    number: 8080
  subset: v1
  weight: 50
fault:
  delay:
    fixedDelay: 10s
    percentage:
      value: 50
- destination:
  host: cpanel.default.svc.cluster.local
  port:
    number: 8080
  subset: v2
  weight: 50

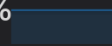
```

before fault injection

Global Request Volume

2.5 ops

Global Success Rate (non-5xx responses)

100%

4xxs

No Data

5xxs

No Data

HTTP/GRPC Workloads ▾

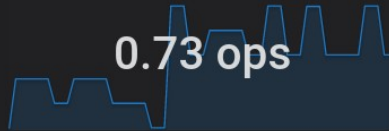
Service	Workload ▾	Requests	P50 Latency	P90 Latency	P99 Latency	Success Rate
<a href="#">section-1.default.svc.cluster.local</a>	<a href="#">section-1-deploy.default</a>	0.47 ops	7.71 ms	18.70 ms	24.37 ms	100.00%
<a href="#">image-analysis.default.svc.cluster.local</a>	<a href="#">image-analysis-deploy.default</a>	0.47 ops	437.50 ms	1.45 s	2.40 s	100.00%
<a href="#">face-recognition.default.svc.cluster.local</a>	<a href="#">face-recognition-deploy.default</a>	0.04 ops	826.92 ms	1.98 s	2.45 s	100.00%
<a href="#">cpanel.default.svc.cluster.local</a>	<a href="#">cpanel-v2.default</a>	0.27 ops	2.73 ms	4.91 ms	9.40 ms	100.00%
<a href="#">cpanel.default.svc.cluster.local</a>	<a href="#">cpanel-v1.default</a>	0.24 ops	3.44 ms	8.17 ms	9.82 ms	100.00%
<a href="#">collector.default.svc.cluster.local</a>	<a href="#">collector-deploy.default</a>	0.47 ops	1.61 s	3.69 s	4.87 s	100.00%
<a href="#">camera-agent-2.default.svc.cluster.local</a>	<a href="#">camera-agent-2-deploy.default</a>	0 ops	NaN	NaN	NaN	NaN
<a href="#">camera-agent-1.default.svc.cluster.local</a>	<a href="#">camera-agent-1-deploy.default</a>	0 ops	NaN	NaN	NaN	NaN
<a href="#">alerts.default.svc.cluster.local</a>	<a href="#">alerts-deploy.default</a>	0.04 ops	17.50 ms	23.50 ms	24.85 ms	100.00%

client workloads - workloads that are calling this service  
 service workloads - workloads that are providing this service

\$ make fault-injection-500

## SERVICE: image-analysis.default.svc.cluster.local

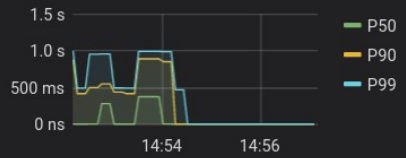
Client Request Volume



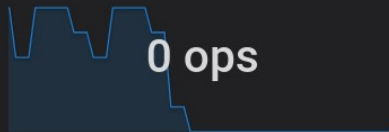
Client Success Rate (non-5xx responses...)



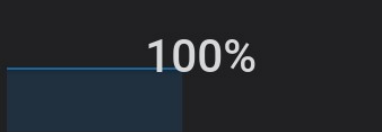
Client Request Duration



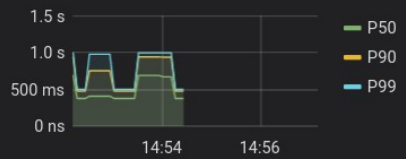
Server Request Volume



Server Success Rate (non-5xx responses...)

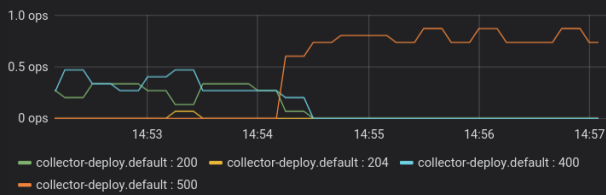


Server Request Duration



## CLIENT WORKLOADS

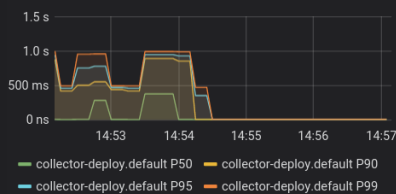
Incoming Requests by Source And Response Code



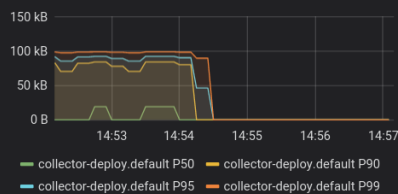
Incoming Success Rate (non-5xx responses) By Source



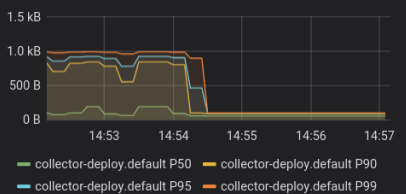
Incoming Request Duration by Source



Incoming Request Size By Source

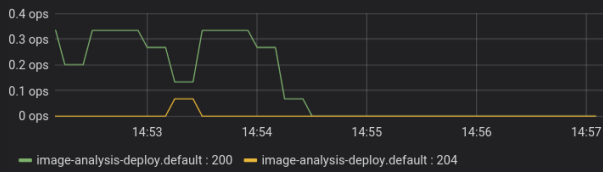


Response Size By Source



## SERVICE WORKLOADS

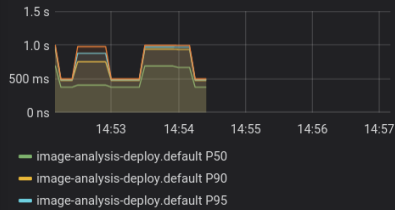
Incoming Requests by Destination And Response Code



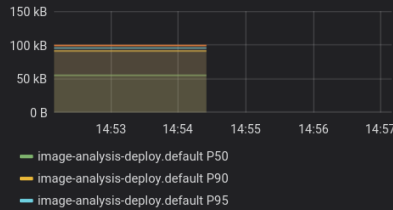
Incoming Success Rate (non-5xx responses) By Source



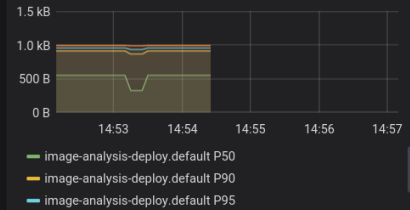
Incoming Request Duration by Source



Incoming Request Size By Source



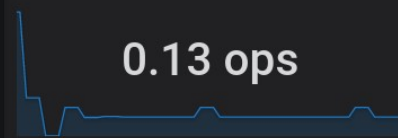
Response Size By Source



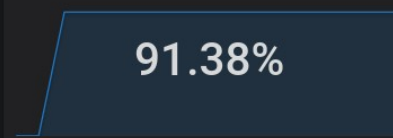
\$ make fault-injection-delay10

## SERVICE: image-analysis.default.svc.cluster.local

Client Request Volume



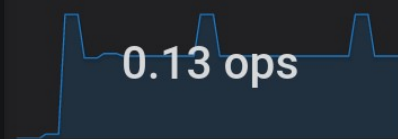
Client Success Rate (non-5xx respons...



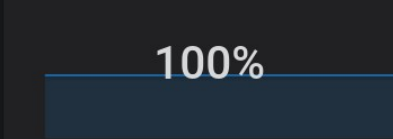
Client Request Duration



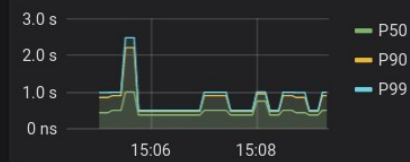
Server Request Volume



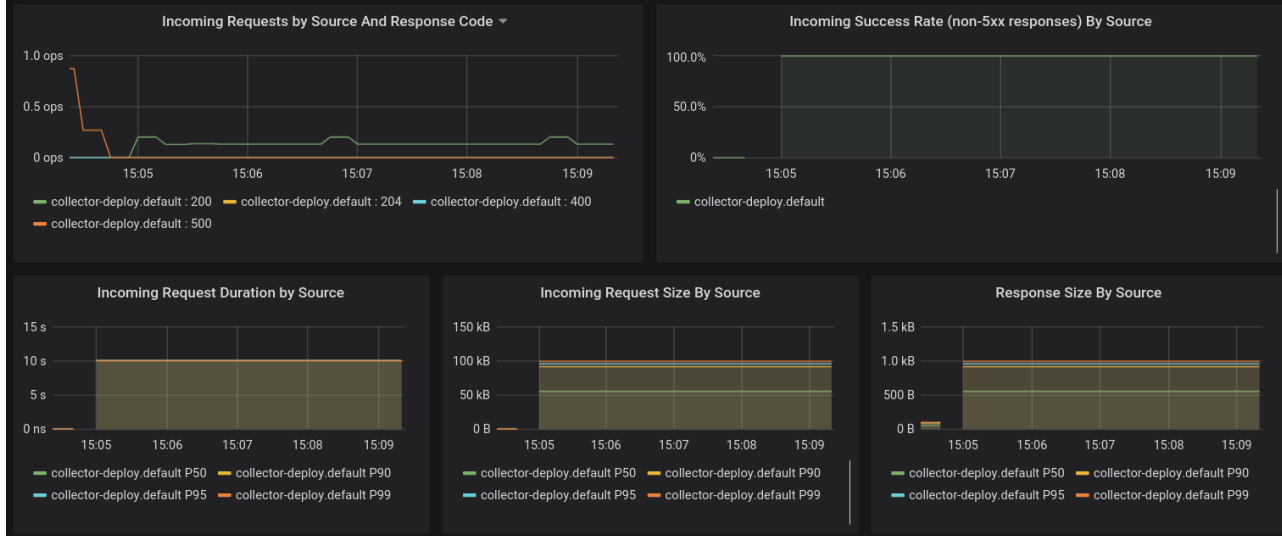
Server Success Rate (non-5xx respons...



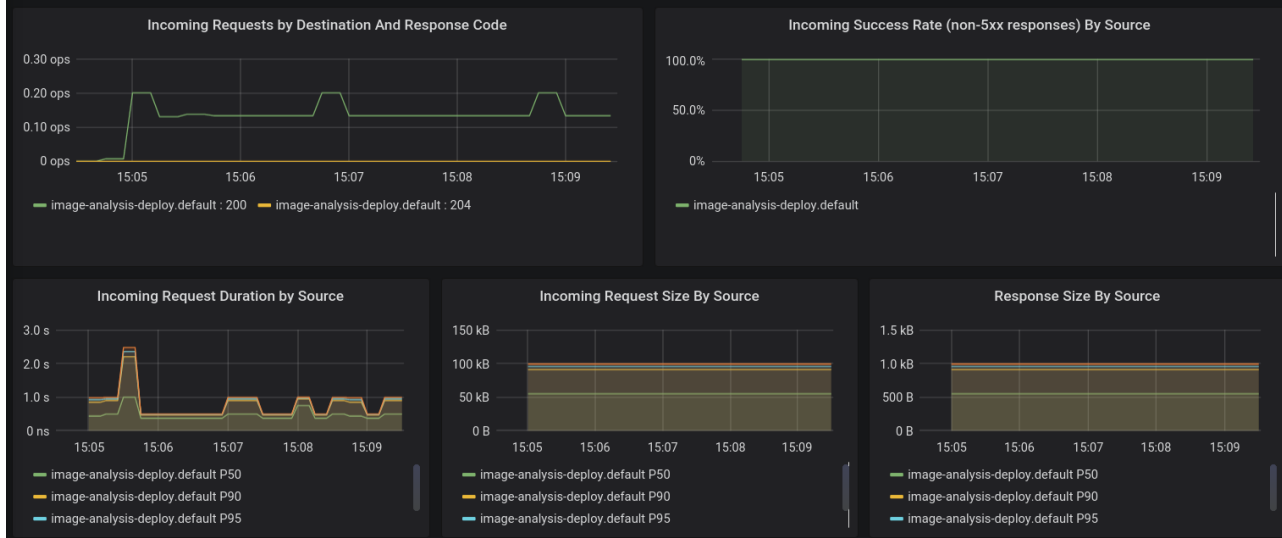
Server Request Duration



## CLIENT WORKLOADS



## SERVICE WORKLOADS



Healthchecks ??? need to retest

```

13:12 $ kubectl get pods --watch
NAME                                READY    STATUS    RESTARTS   AGE
alerts-deploy-d58cf7647-9m2g9      2/2     Running   0           15m
camera-agent-1-deploy-8468f948f8-5m8h2 2/2     Running   0           15m
camera-agent-2-deploy-545c4d9bd5-ltb49 2/2     Running   0           15m
collector-deploy-558dd7dd45-6p6ng    2/2     Running   0           15m
cpanel-v1-8446d9dd45-wnlxs           2/2     Running   0           15m
cpanel-v2-8445ff5964-hpfh8           2/2     Running   0           15m
face-recognition-deploy-7b954c454-79qdg 2/2     Running   0           15m
image-analysis-deploy-696d85448f-jxjdj 2/2     Running   0           15m
section-1-deploy-6b8b74f786-6wg75    2/2     Running   0           15m
alerts-deploy-d58cf7647-9m2g9      1/2     Running   0           18m
alerts-deploy-d58cf7647-9m2g9      1/2     Running   1           18m
alerts-deploy-d58cf7647-9m2g9      2/2     Running   1           18m

```

```

Mem: 522436K used, 11196688K free, 500500K shrd, 72200K buff, 2415832K cached
CPU: 14% usr 9% sys 0% nic 75% idle 0% io 0% irq 0% irq
Load average: 1.30 1.76 1.70 4/1824 14

```

PID	PPID	USER	STAT	VSZ	%VSZ	CPU	%CPU	COMMAND
6	1	appuser	S	30072	0%	1	0%	/usr/local/bin/python /app/main.py
1	0	appuser	S	28740	0%	0	0%	python main.py
9	0	appuser	S	1644	0%	1	0%	sh
14	9	appuser	R	11580	0%	3	0%	top

```

/app$ kill 6
/app$ command terminated with exit code 137
X-KILL /uni/ws2019/bachelor/resilient_istio [master|+3]
13:15 $ kubectl describe pods alerts-deploy-d58cf7647-9m2g9 | grep Liveness\|Readiness
Liveness: tcp-socket :8080 delay=15s timeout=3s period=7s #success=1 #failure=3
Readiness: tcp-socket :8080 delay=10s timeout=3s period=7s #success=1 #failure=3
Readiness: http-get http://:15020/healthz/ready delay=1s timeout=1s period=2s #success=1 #failure=30
Warning Unhealthy 19m kubelet, minikube Readiness probe failed: HTTP probe failed with statuscode: 503

```

## Conclusions and Future Work

- pros of istio resiliency features
- expanse of service meshes
- complexity of operations (# of micro services, agile)
- advice - move to production step by step incremental, complexity of debugging
  - configure log level to error – otherwise too much traffic \$\$\$

## References

- Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.
- <https://www.martinfowler.com/articles/microservices.html>
- <https://snyk.io/blog/10-docker-image-security-best-practices/>
- Cloud computing assignment
- <https://12factor.net/>
- <https://kubernetes.io/>
- <https://istio.io/>
- <https://www.docker.com/>

## Appendix / Supplemental Material

- cc assignment
- commands



