CLOUD NATIVE COMPUTING FOUNDATION

Deployment Strategies on Kubernetes

By Etienne Tremel

Software engineer at Container Solutions

@etiennetremel
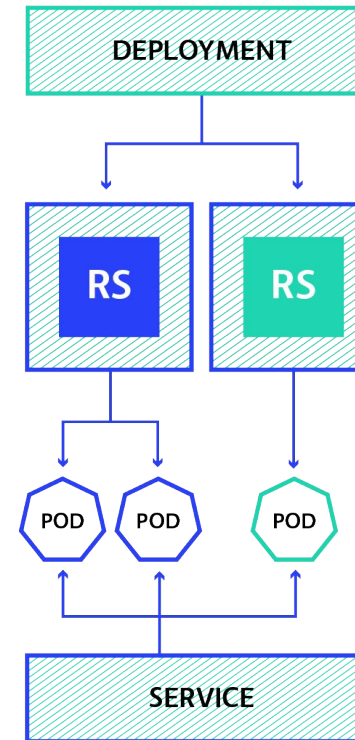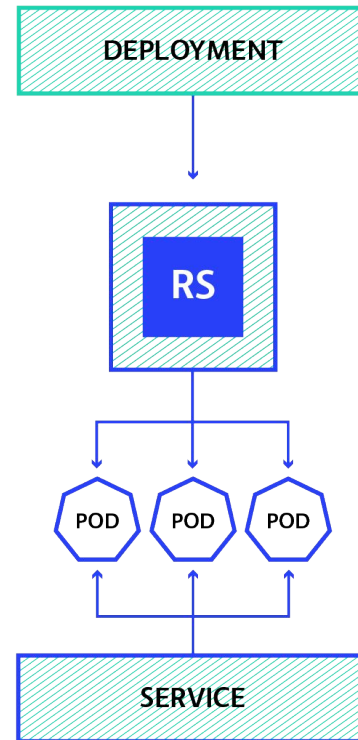
February 13th, 2017

Agenda

- Kubernetes in brief
- Look at 6 different strategies
  - Recreate
  - Ramped
  - Blue/Green
  - Canary
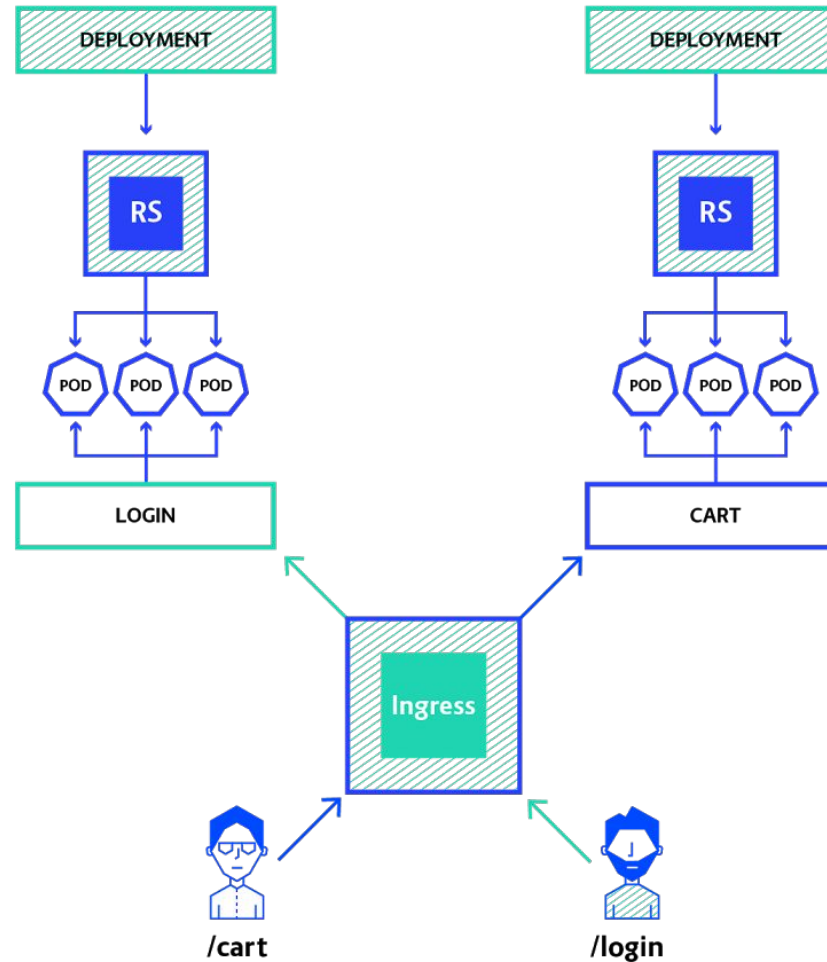 - A/B Testing
  - Shadow
- Sum-up
- Next

CLOUD NATIVE
COMPUTING FOUNDATION

Kubernetes in brief
Deployments, replica-sets, pods and services

CLOUD NATIVE
COMPUTING FOUNDATION

Kubernetes in brief
Advanced routing using Ingress

Ingress controllers:
- Nginx
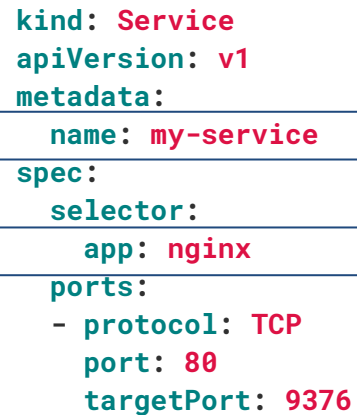- HA Proxy
- Traefik
- Istio
- Linkerd
- GKE
- etc.

CLOUD NATIVE
COMPUTING FOUNDATION

Kubernetes in brief
Configuration

Deployment configuration:

Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
```

ReplicaSet

```
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
```

Pod

```
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```
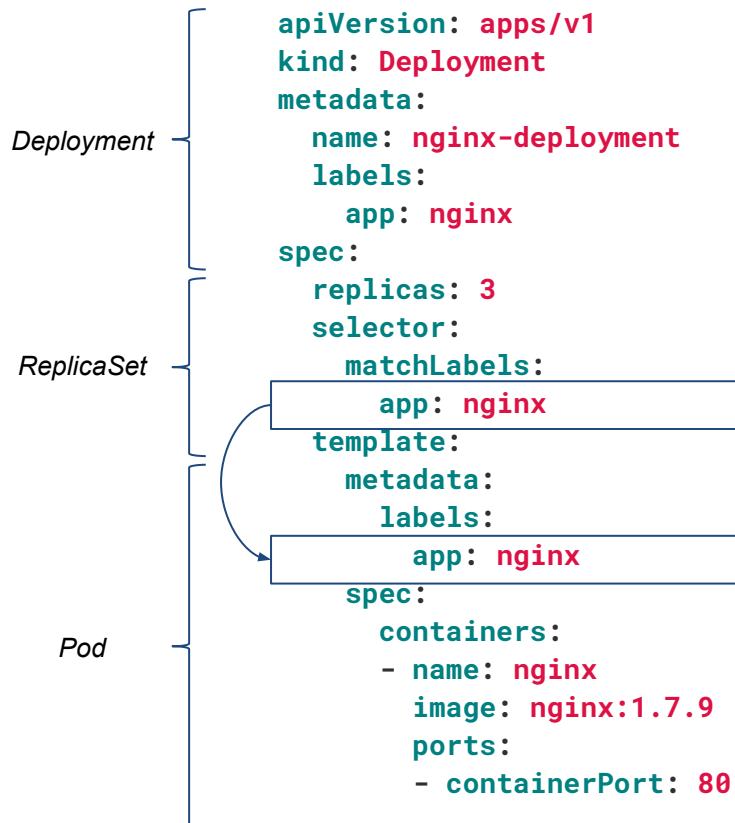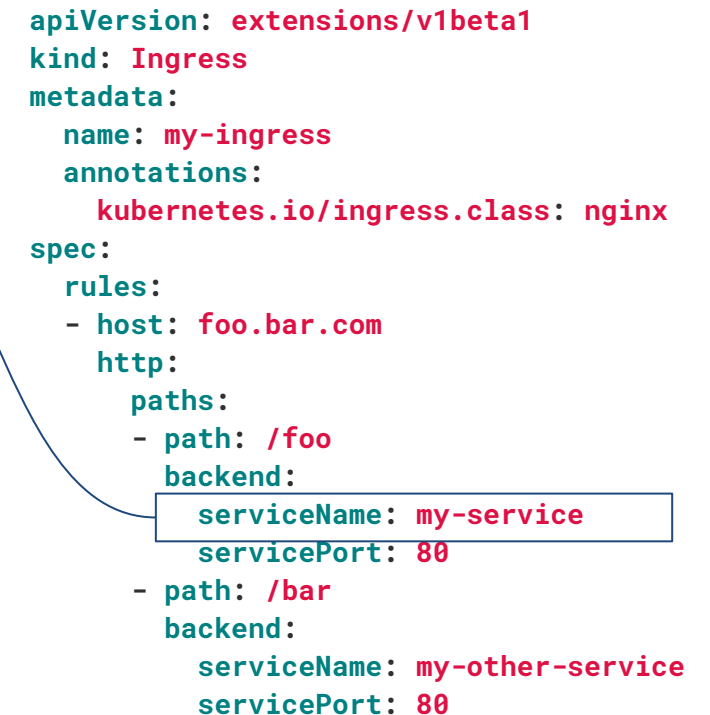
Service configuration:

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

Ingress configuration:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: my-service
          servicePort: 80
      - path: /bar
        backend:
          serviceName: my-other-service
          servicePort: 80
```

CLOUD NATIVE
COMPUTING FOUNDATION

Deployment strategies

- Recreate *native*

- Ramped *native*

- Blue/Green *extra step needed*

- Canary *extra step needed*

- A/B Testing *require additional component*
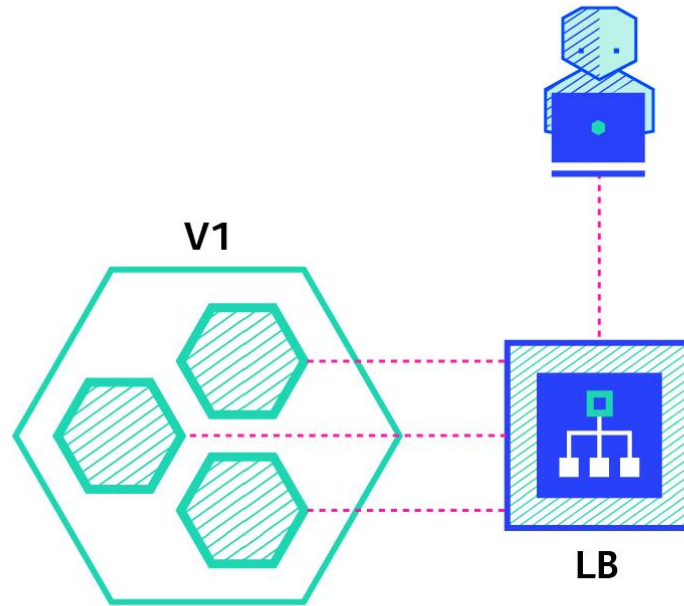
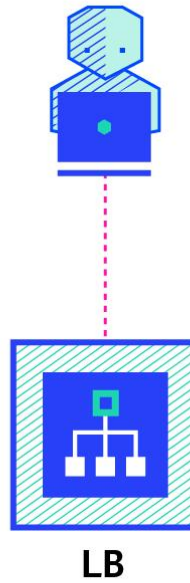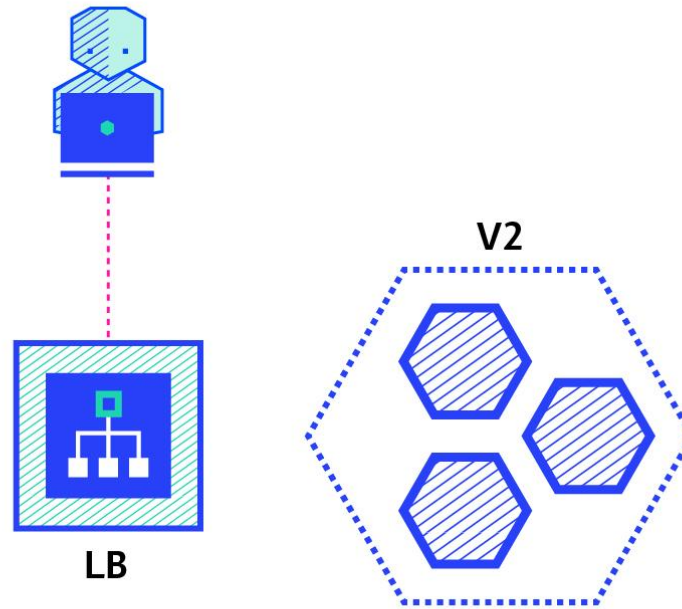- Shadow *require additional component*

Get your hands on: https://github.com/ContainerSolutions/k8s-deployment-strategies

CLOUD NATIVE
COMPUTING FOUNDATION

Recreate

CLOUD NATIVE
COMPUTING FOUNDATION

Recreate

V1

LB

In this case [LB] *is a Kubernetes Service*

CLOUD NATIVE
COMPUTING FOUNDATION

Recreate

LB

In this case [LB] *is a Kubernetes Service*

Recreate

V2

LB

In this case [LB] *is a Kubernetes Service*

CLOUD NATIVE
COMPUTING FOUNDATION

Recreate

In this case [LB] *is a Kubernetes Service*

CLOUD NATIVE
COMPUTING FOUNDATION

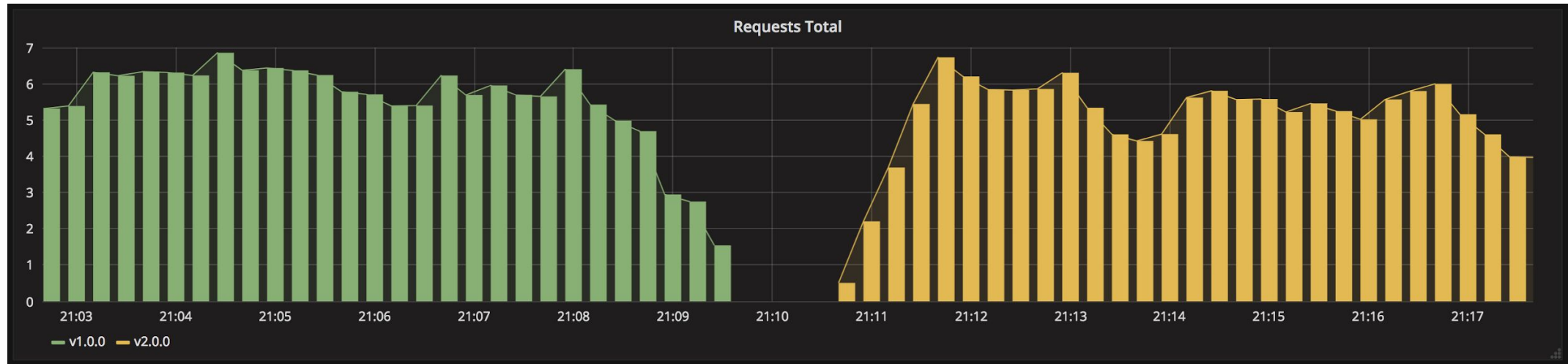# Recreate

```
[...]
kind: Deployment
spec:
    replicas: 3
    strategy:
        type: Recreate
[...]
```

```
$ kubectl apply -f ./manifest.yaml
```

CLOUD NATIVE
COMPUTING FOUNDATION

Recreate

Pattern of the traffic during a release

Service unavailable
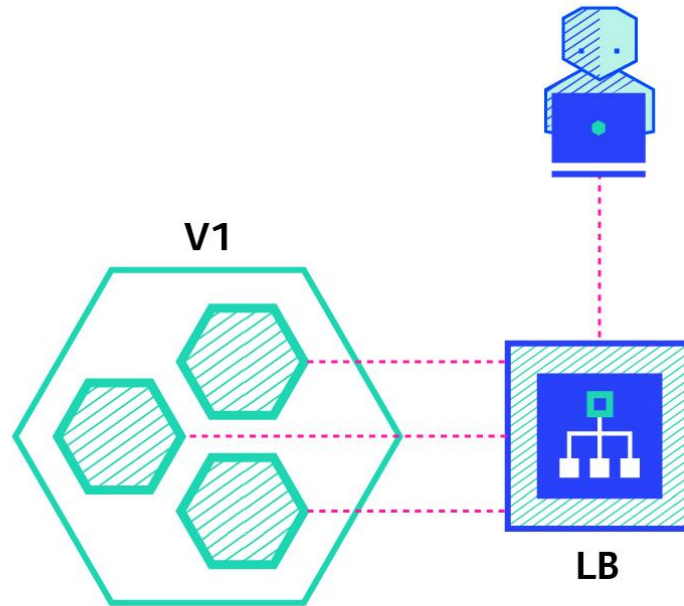
CLOUD NATIVE
COMPUTING FOUNDATION

Recreate

Pros:
- easy to setup

Cons:
- high impact on the user, expect downtime that depends on both shutdown and boot duration of the application
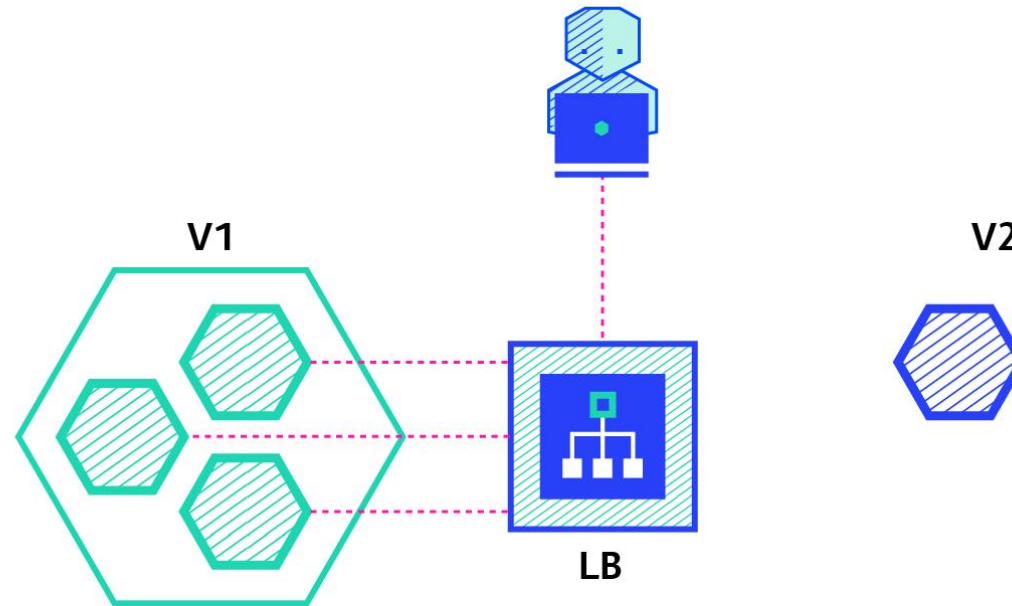
CLOUD NATIVE
COMPUTING FOUNDATION

Ramped

aka incremental, rolling update

CLOUD NATIVE
COMPUTING FOUNDATION

Ramped - aka Incremental, Rolling

V1

LB

In this case [LB] *is a Kubernetes Service*

CLOUD NATIVE
COMPUTING FOUNDATION

Ramped - aka Incremental, Rolling

V1

V2

LB

In this case [LB] *is a Kubernetes Service*

CLOUD NATIVE
COMPUTING FOUNDATION

Ramped - aka Incremental, Rolling

In this case [LB] *is a Kubernetes Service*

CLOUD NATIVE
COMPUTING FOUNDATION

Ramped - aka Incremental, Rolling

V1

V2

LB

In this case [LB] *is a Kubernetes Service*

CLOUD NATIVE
COMPUTING FOUNDATION

Ramped - aka Incremental, Rolling

V2

LB

In this case [LB] *is a Kubernetes Service*

CLOUD NATIVE
COMPUTING FOUNDATION
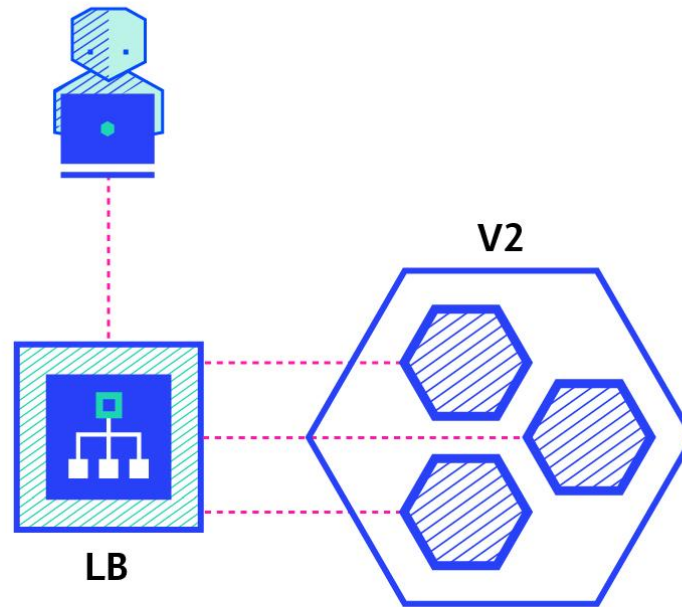
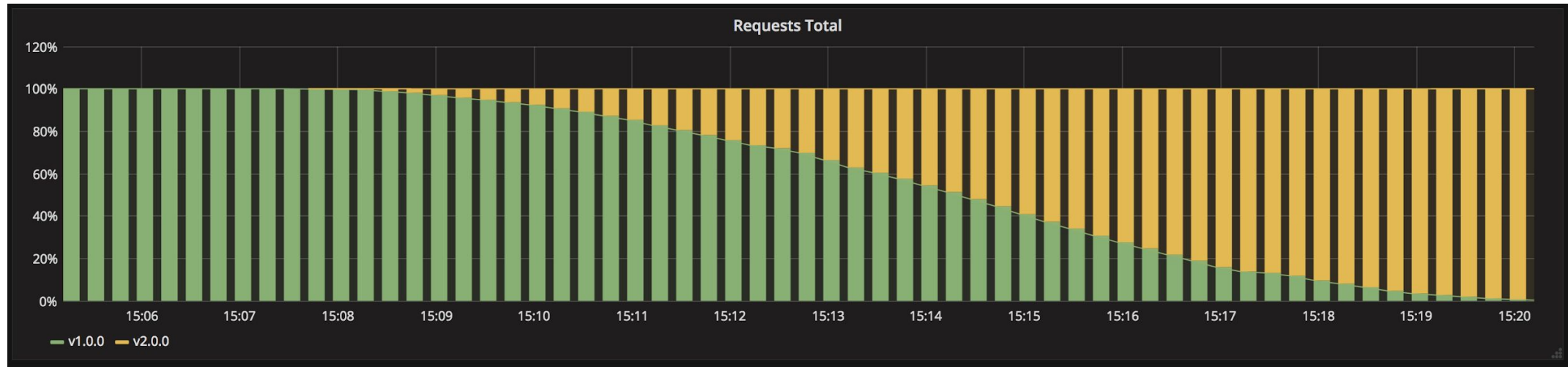# Ramped - aka Incremental, Rolling

```
[...]
kind: Deployment
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 2          # how many pods we can add at a time
      maxUnavailable: 0    # maxUnavailable define how many pods can be
                           # unavailable during the rolling update
[...]
```

```
$ kubectl apply -f ./manifest.yaml
```

CLOUD NATIVE
COMPUTING FOUNDATION

Ramped - aka Incremental, Rolling

Pattern of the traffic during a release

Ramped - aka Incremental, Rolling

Pros:

- easy to use

- version is slowly released across instances

- convenient for stateful applications that can handle ongoing rebalancing of the data
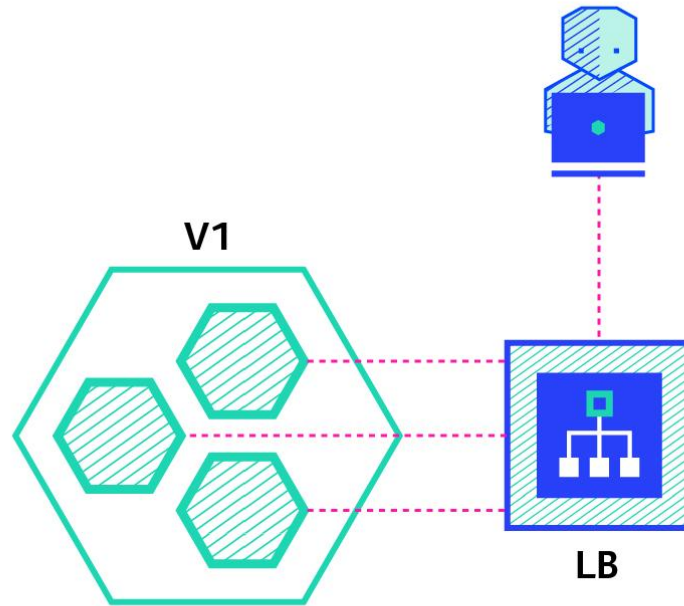
Cons:

- rollout/rollback can take time

- no control over traffic

CLOUD NATIVE
COMPUTING FOUNDATION

Blue/Green

aka red/black

CLOUD NATIVE
COMPUTING FOUNDATION

Blue/Green - aka Red/Black

Blue/Green - aka Red/Black

CLOUD NATIVE
COMPUTING FOUNDATION

Blue/Green - aka Red/Black

CLOUD NATIVE
COMPUTING FOUNDATION
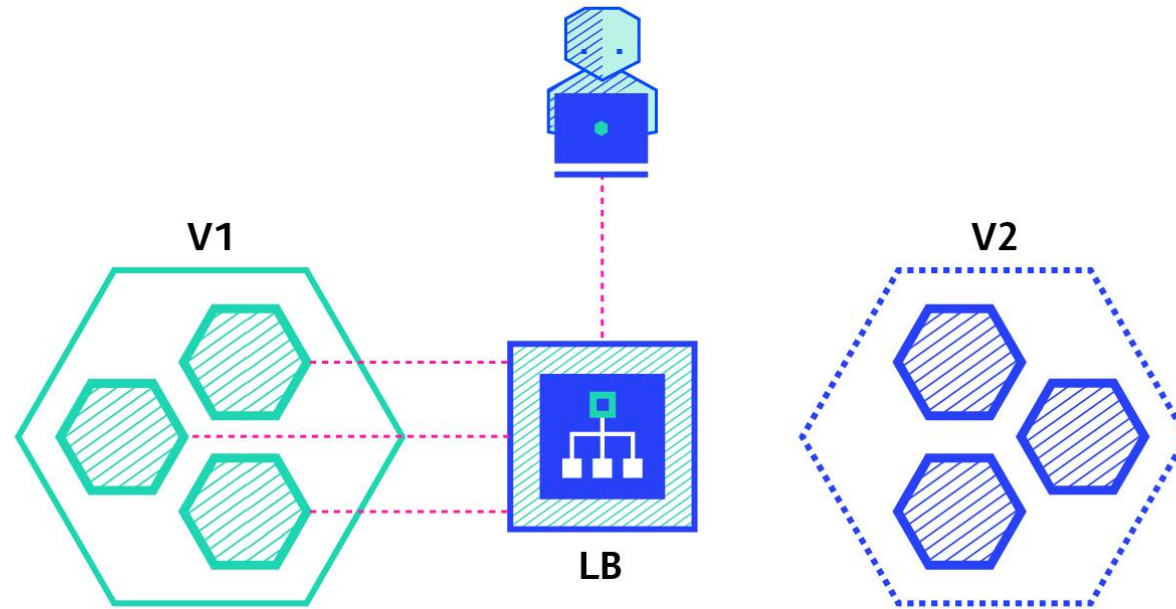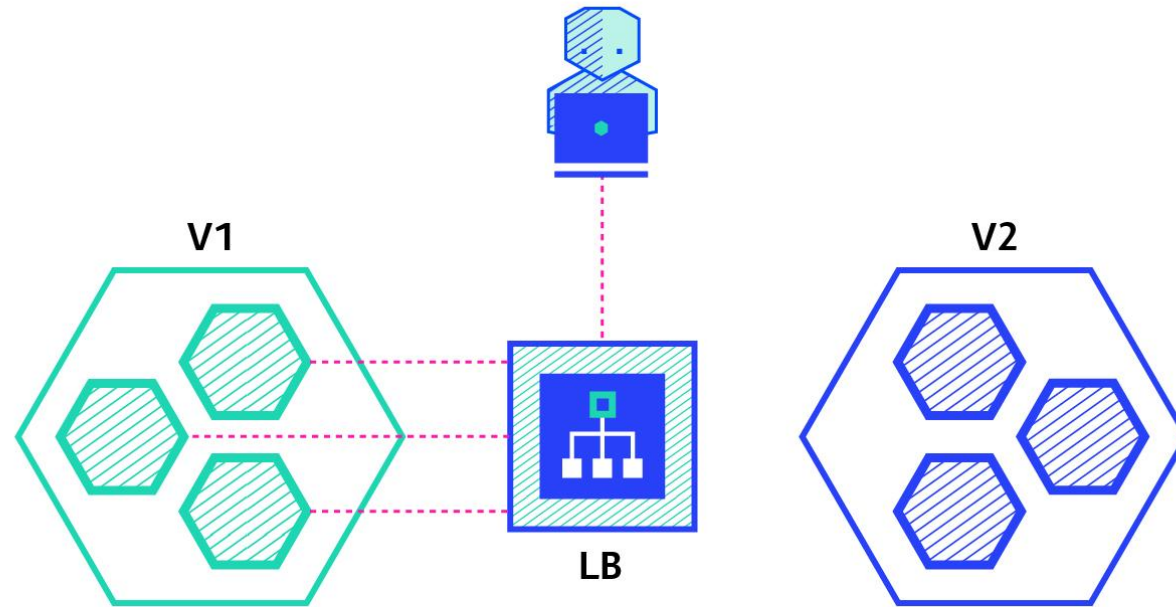
Blue/Green - aka Red/Black
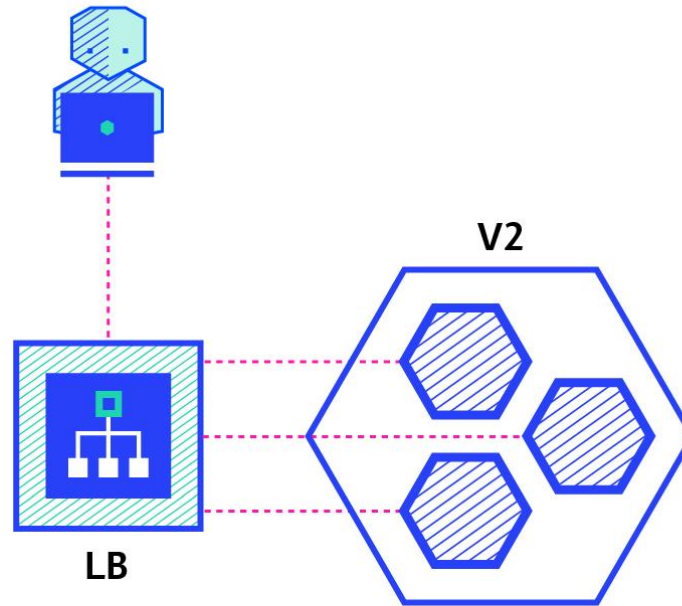
Blue/Green - aka Red/Black
Single service deployment

```
[...]
kind: Service
spec:
  # Note here that we match both the app and the version.
  # When switching traffic, update the label "version" with
  # the appropriate value, ie: v2.0.0
  selector:
    app: my-app
    version: v1.0.0
[...]
```

```
$ kubectl apply -f ./manifest-v2.yaml
$ kubectl patch service my-app -p \
    '{"spec":{"selector":{"version":"v2.0.0"}}}'
$ kubectl delete -f ./manifest-v1.yaml
```

CLOUD NATIVE
COMPUTING FOUNDATION

Blue/Green - aka Red/Black
To rollout multiple services at once, use Ingress

```
[...]
kind: Ingress
spec:
  rules:
  - host: login.domain.com
    http:
      paths:
      - backend:
          serviceName: login-v2
          servicePort: 80
  - host: cart.domain.com
    http:
      paths:
      - backend:
          serviceName: cart-v2
          servicePort: 80
[...]
```

```
[...]
kind: Service
metadata:
    name: login-v2
spec:
    selector:
      app: login
      version: v2.0.0
[...]
```

```
[...]
kind: Service
metadata:
    name: cart-v2
spec:
    selector:
      app: cart
      version: v2.0.0
[...]
```

```
$ kubectl apply -f ./manifest-v2.yaml
$ kubectl apply -f ./ingress.yaml
$ kubectl delete -f ./manifest-v1.yaml
```

CLOUD NATIVE
COMPUTING FOUNDATION

Blue/Green - aka Red/Black

Pattern of the traffic during a release

CLOUD NATIVE
COMPUTING FOUNDATION
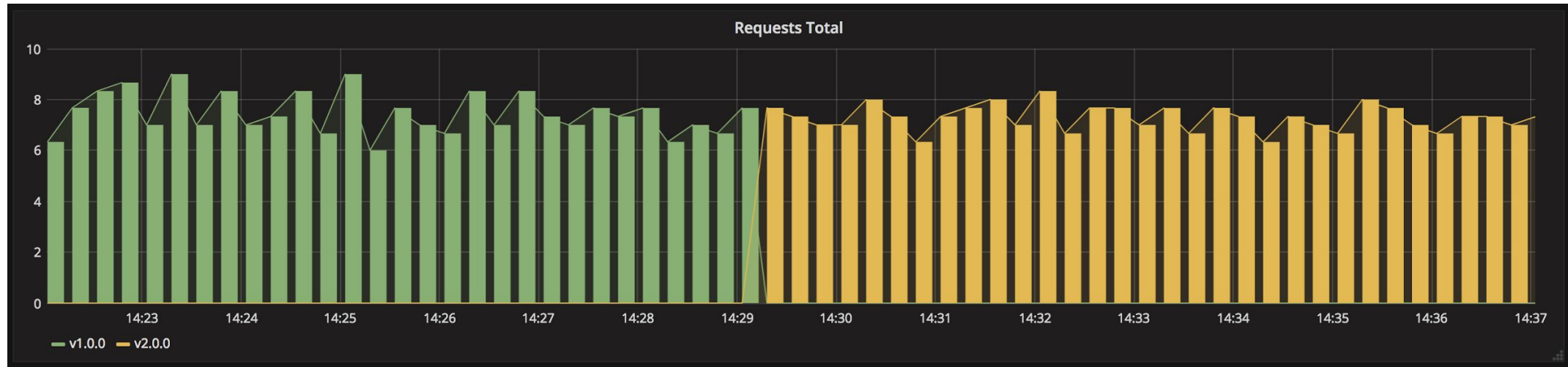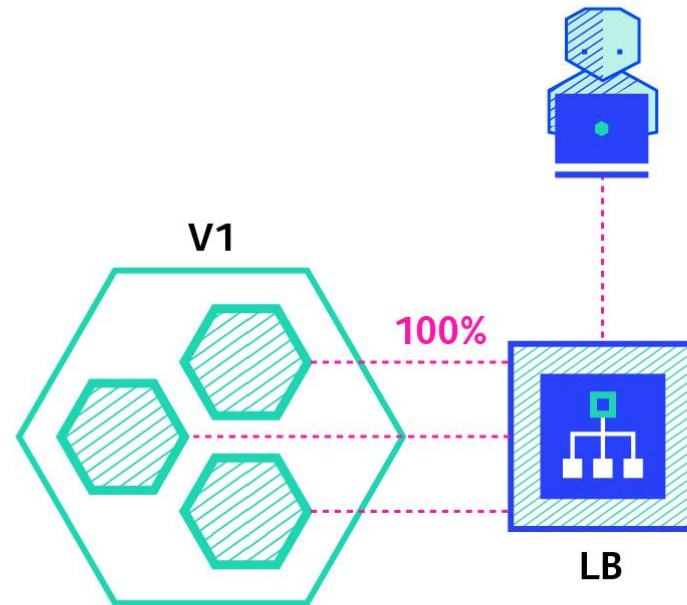
Blue/Green - aka Red/Black

Pros:

- instant rollout/rollback

- good fit for front-end that load versioned assets from the same server
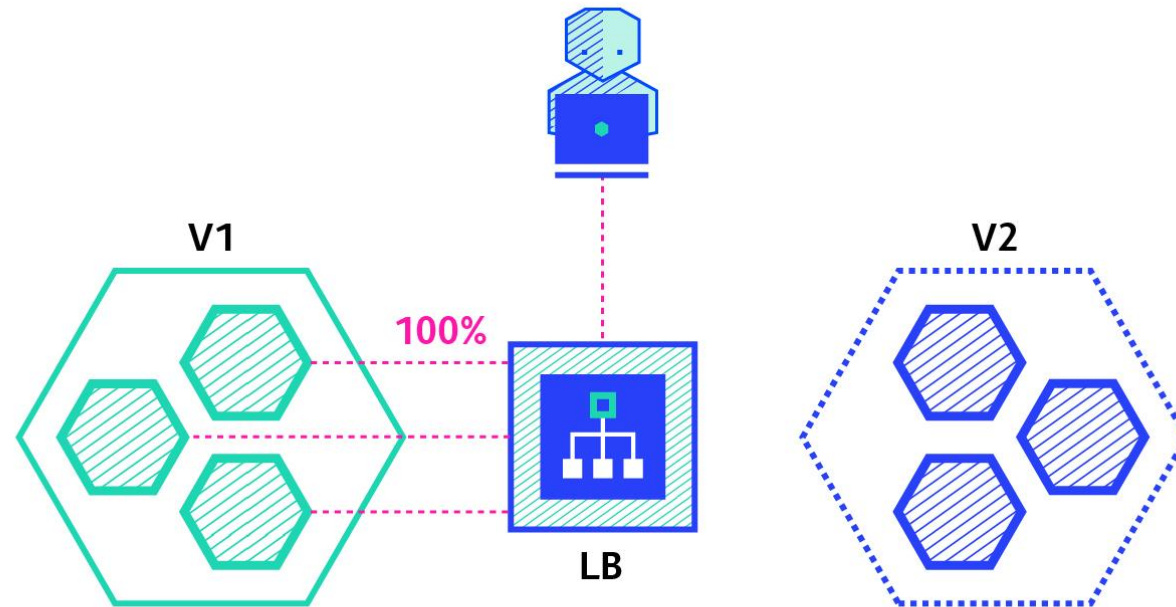
- dirty way to fix application dependency hell

Cons:

- expensive as it requires double the resources

- proper test of the entire platform should be done before releasing to production
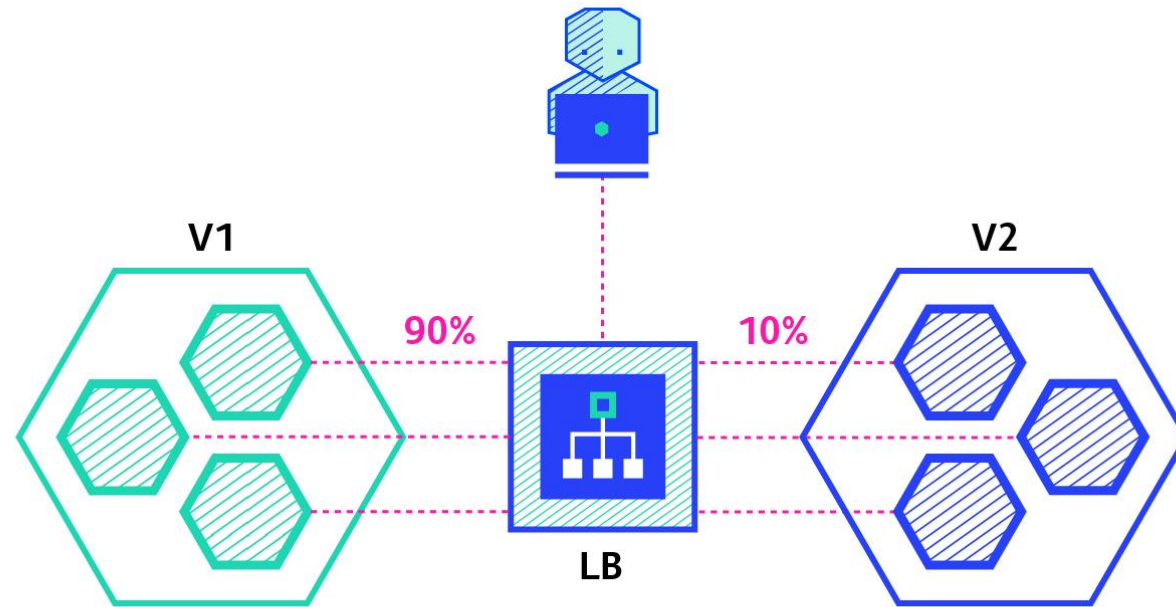
CLOUD NATIVE
COMPUTING FOUNDATION

Canary

CLOUD NATIVE
COMPUTING FOUNDATION

Canary

V1

100%

LB

CLOUD NATIVE
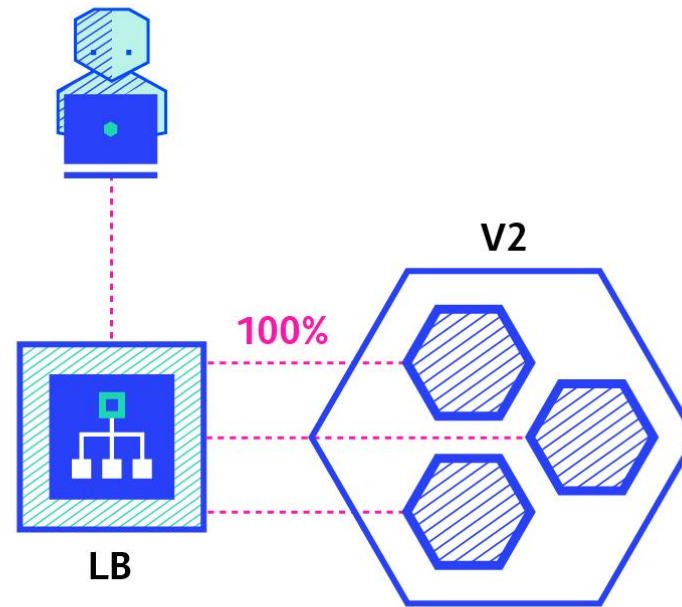**COMPUTING FOUNDATION**

Canary

Canary

Canary

Canary

```
[...]
kind: Deployment
metadata:
  name: my-app-v1
spec:
  replicas: 9
  template:
    labels:
      app: my-app
      version: v1.0.0
[...]
```

```
[...]
kind: Deployment
metadata:
  name: my-app-v2
spec:
  replicas: 1
  template:
    labels:
      app: my-app
      version: v2.0.0
[...]
```

```
[...]
kind: Service
metadata:
  name: my-app
spec:
  selector:
    app: my-app
[...]
```

```
$ kubectl apply -f ./manifest-v2.yaml
$ kubectl scale deploy/my-app-v2 --replicas=10
$ kubectl delete -f ./manifest-v1.yaml
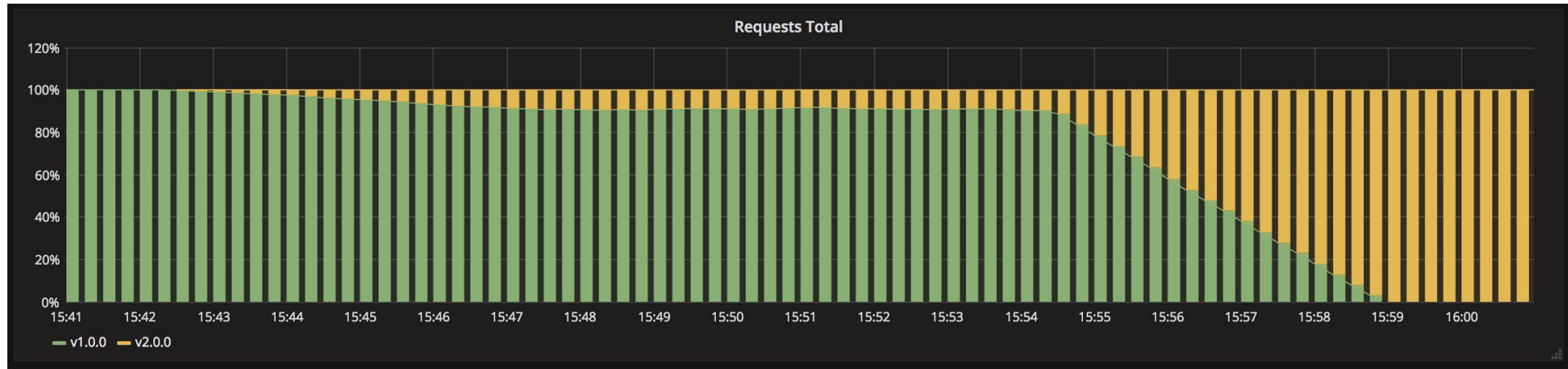```

CLOUD NATIVE
COMPUTING FOUNDATION

# Canary

Example of shifting traffic based on weight (percentage) using *Istio*

```
[...]
kind: RouteRule
metadata:
  name: my-app
spec:
  destination:
    name: my-app
  route:
  - labels:
      version: v1.0.0
    weight: 90          # 90% traffic
  - labels:
      version: v2.0.0
    weight: 10          # 10% traffic
[...]
```

```
$ kubectl apply -f ./manifest-v2.yaml
$ kubectl apply -f ./routerule.yaml
```

CLOUD NATIVE
COMPUTING FOUNDATION

Canary

Pattern of the traffic during a release

Canary

Pros:

- version released for a subset of users

- convenient for error rate and performance monitoring
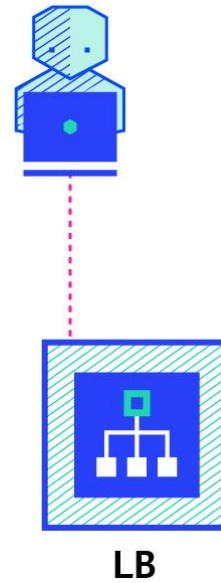
- fast rollback

Cons:

- slow rollout

- sticky sessions might be required

- precise traffic shifting would require additional tool like Istio or Linkerd

A/B Testing

CLOUD NATIVE
COMPUTING FOUNDATION

A/B Testing

LB

CLOUD NATIVE
COMPUTING FOUNDATION

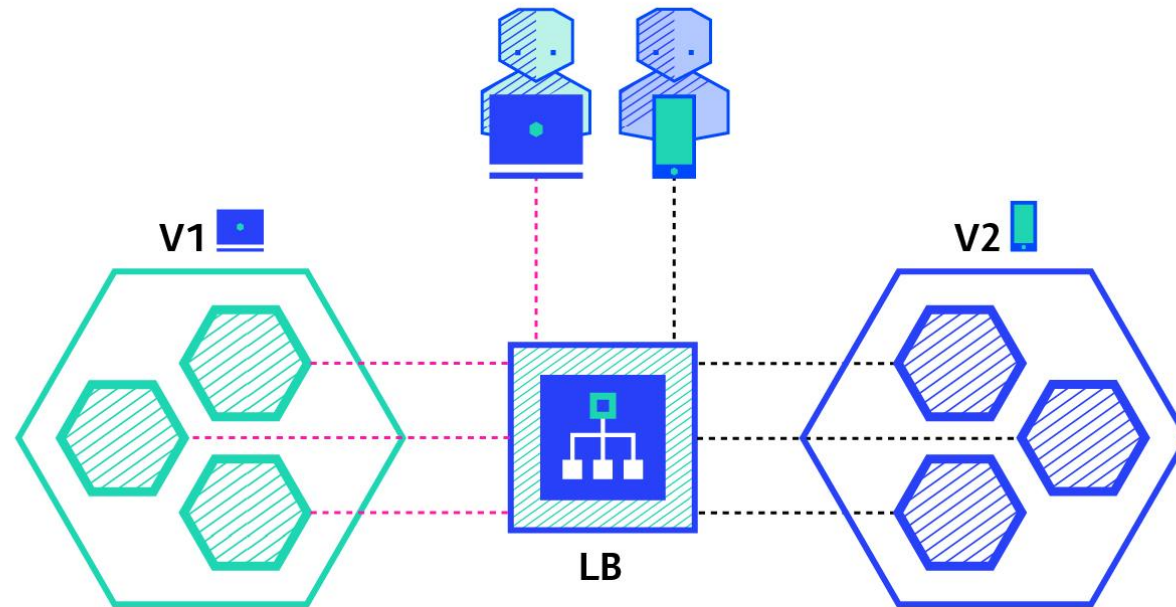# A/B Testing

A/B Testing

CLOUD NATIVE
COMPUTING FOUNDATION
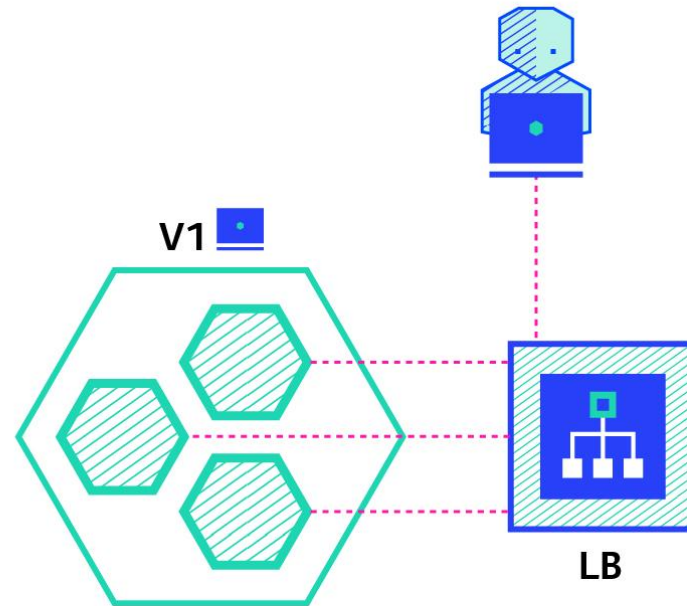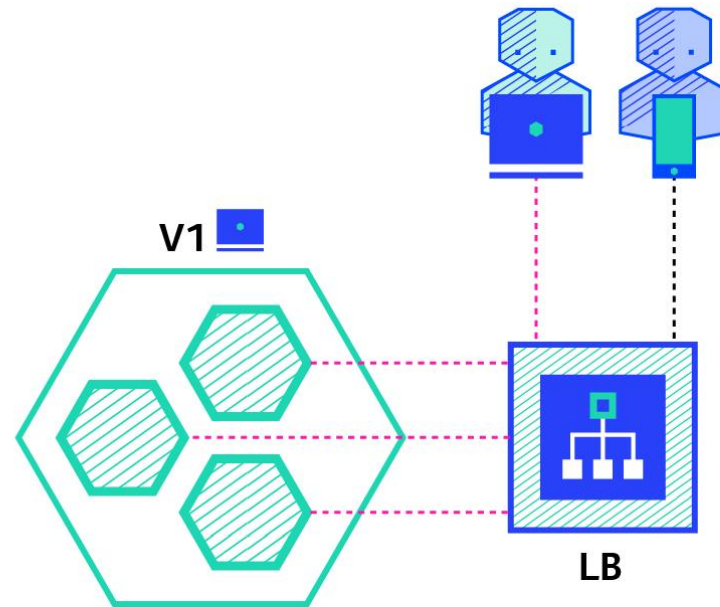
A/B Testing

Possible conditions:
- *Geolocalisation*
- *Language*
- *Cookie*
- *User Agent (device, OS, etc.)*
- *Custom Header*
- *Query parameters*

A/B Testing

Example of shifting traffic based on request Headers using *Istio*

```
[...]
kind: RouteRule
metadata:
  name: my-app-v1
spec:
  destination:
    name: my-app
  route:
  - labels:
      version: v1.0.0
  match:
    request:
      headers:
        x-api-version:
          exact: "v1.0.0"
[...]
```
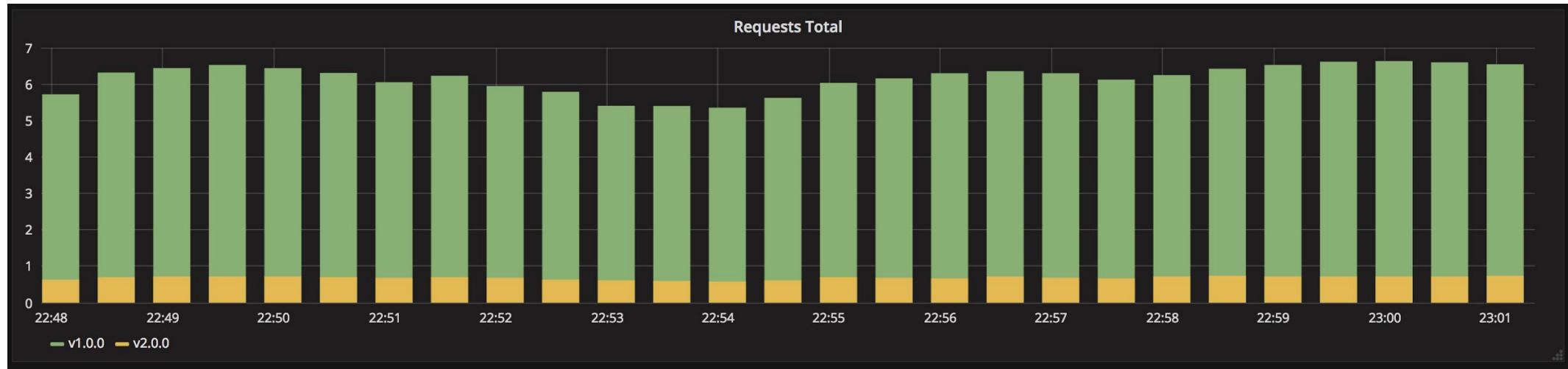
```
[...]
kind: RouteRule
metadata:
  name: my-app-v2
spec:
  destination:
    name: my-app
  route:
  - labels:
      version: v2.0.0
  match:
    request:
      headers:
        x-api-version:
          exact: "v2.0.0"
[...]
```

```
$ kubectl apply -f ./manifest-v2.yaml
$ kubectl apply -f ./routerule.yaml
```

CLOUD NATIVE
COMPUTING FOUNDATION

A/B Testing

Pattern of the traffic during a release

CLOUD NATIVE
COMPUTING FOUNDATION

A/B Testing

Pros:

- several versions run in parallel

- full control over the traffic distribution

- great tool that can be used for business purpose to improve conversion

Cons:

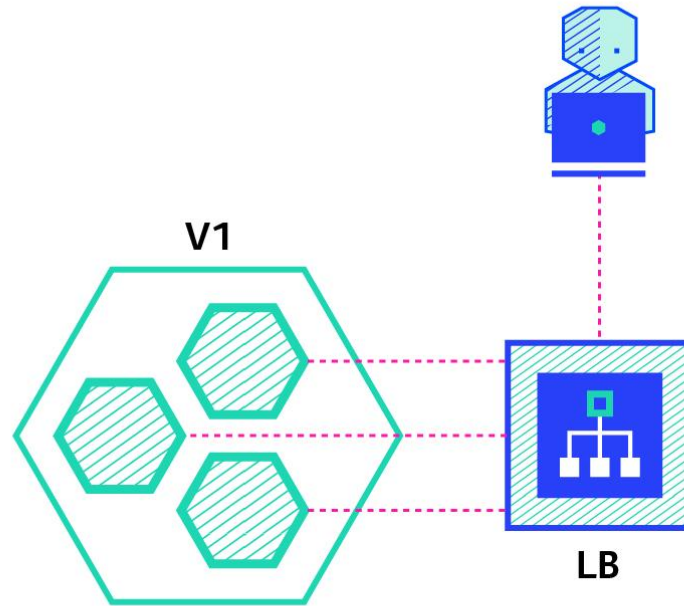- requires intelligent load balancer (Istio, Linkerd, etc.)

- hard to troubleshoot errors for a given session, distributed tracing becomes mandatory

CLOUD NATIVE
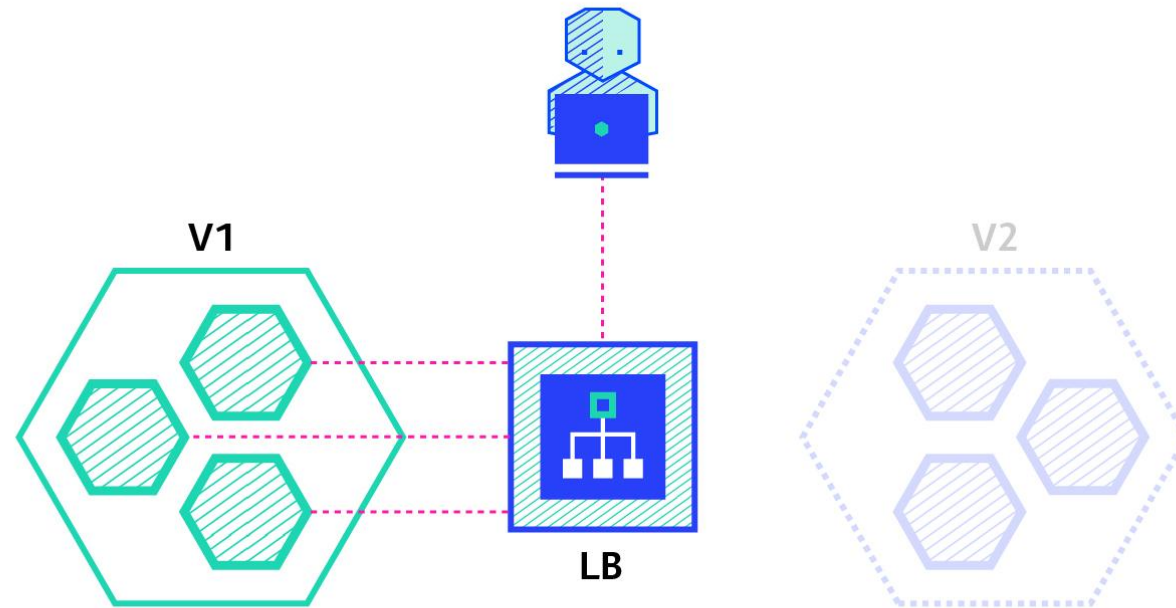**COMPUTING FOUNDATION**

Shadow

aka Mirrored, Dark

Shadow - aka Mirrored, Dark

Shadow - aka Mirrored, Dark

CLOUD NATIVE
COMPUTING FOUNDATION

Shadow - aka Mirrored, Dark

CLOUD NATIVE
COMPUTING FOUNDATION
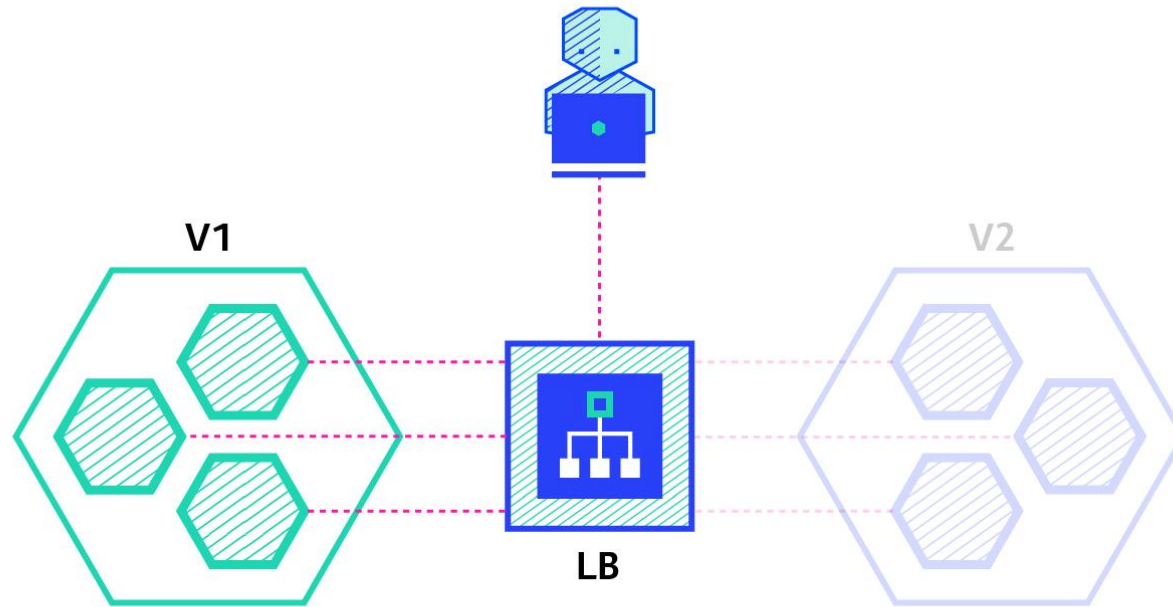
Shadow - aka Mirrored, Dark
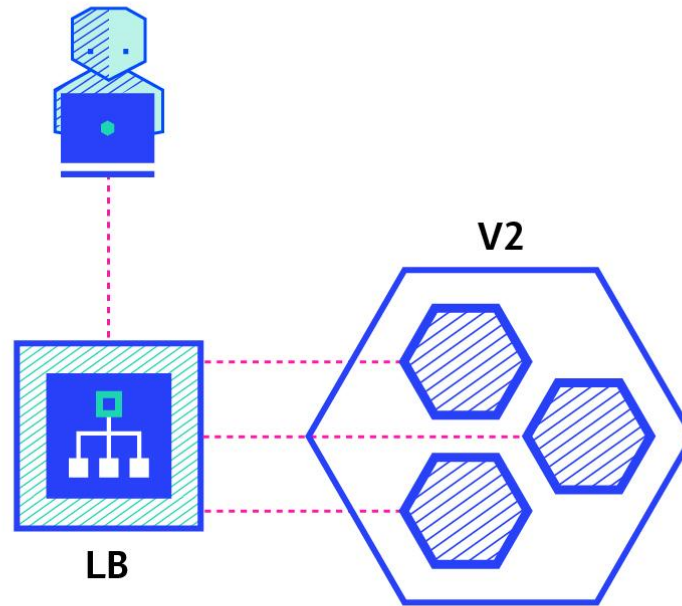
Shadow - aka Mirrored, Dark

Example of mirroring traffic using *Istio*

```
[...]
kind: RouteRule
spec:
  destination:
    name: my-app
  route:
  - labels:
      version: v1.0.0
    weight: 100
  - labels:
      version: v2.0.0
    weight: 0
  mirror:
    name: my-app-v2
    labels:
      version: v2.0.0
[...]
```

```
$ kubectl apply -f ./manifest-v2.yaml
$ kubectl apply -f ./routerule.yaml
```

Shadow - aka Mirrored, Dark

Pattern of the traffic during a release

CLOUD NATIVE
COMPUTING FOUNDATION
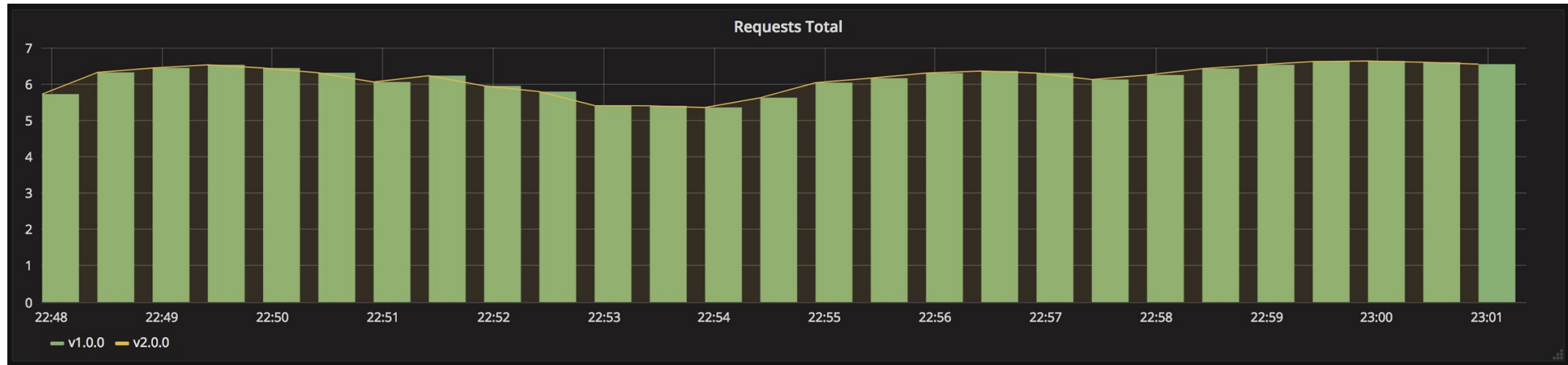
Shadow - aka Mirrored, Dark

Pros:

- performance testing of the application with production traffic
- no impact on the user
- no rollout until the stability and performance of the application meet the requirements

Cons:

- complex to setup
- expensive as it requires double the resources
- not a true user test and can be misleading
- requires mocking/stubbing service for certain cases

Sum-up

- **recreate** if downtime is not a problem

- **recreate** and **ramped** doesn't require any extra step (kubectl apply is enough)

- **ramped** and **blue/green** deployment are usually a good fit and easy to use

- **blue/green** is a good fit for front-end that load versioned assets from the same server

- **blue/green** and **shadow** can be expensive

- **canary** and **a/b** testing should be used if little confidence on the quality of the release

- **canary**, **a/b testing** and **shadow** might require additional cluster component

CLOUD NATIVE
COMPUTING FOUNDATION

Sum-up

Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE — version A is terminated then version B is rolled out	✗	✗	✗	■□□	■■■	■■■	□□□
RAMPED — version B is slowly rolled out and replacing version A	✓	✗	✗	■□□	■■■	■□□	■□□
BLUE/GREEN — version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■■■	□□□	■■□	■■□
CANARY — version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■□□	■□□	■□□	■■□
A/B TESTING — version B is released to a subset of users under specific condition	✓	✓	✓	■□□	■□□	■□□	■■■
SHADOW — version B receives real world traffic alongside version A and doesn't impact the response	✓	✓	✗	■■■	□□□	□□□	■■■

CLOUD NATIVE COMPUTING FOUNDATION

Next

Hands on ***Kubernetes deployment strategies***:

https://github.com/ContainerSolutions/k8s-deployment-strategies

Blog post about strategies:

https://container-solutions.com/kubernetes-deployment-strategies

https://thenewstack.io/deployment-strategies

CLOUD NATIVE
COMPUTING FOUNDATION

**CLOUD NATIVE
COMPUTING
FOUNDATION**

Thank You