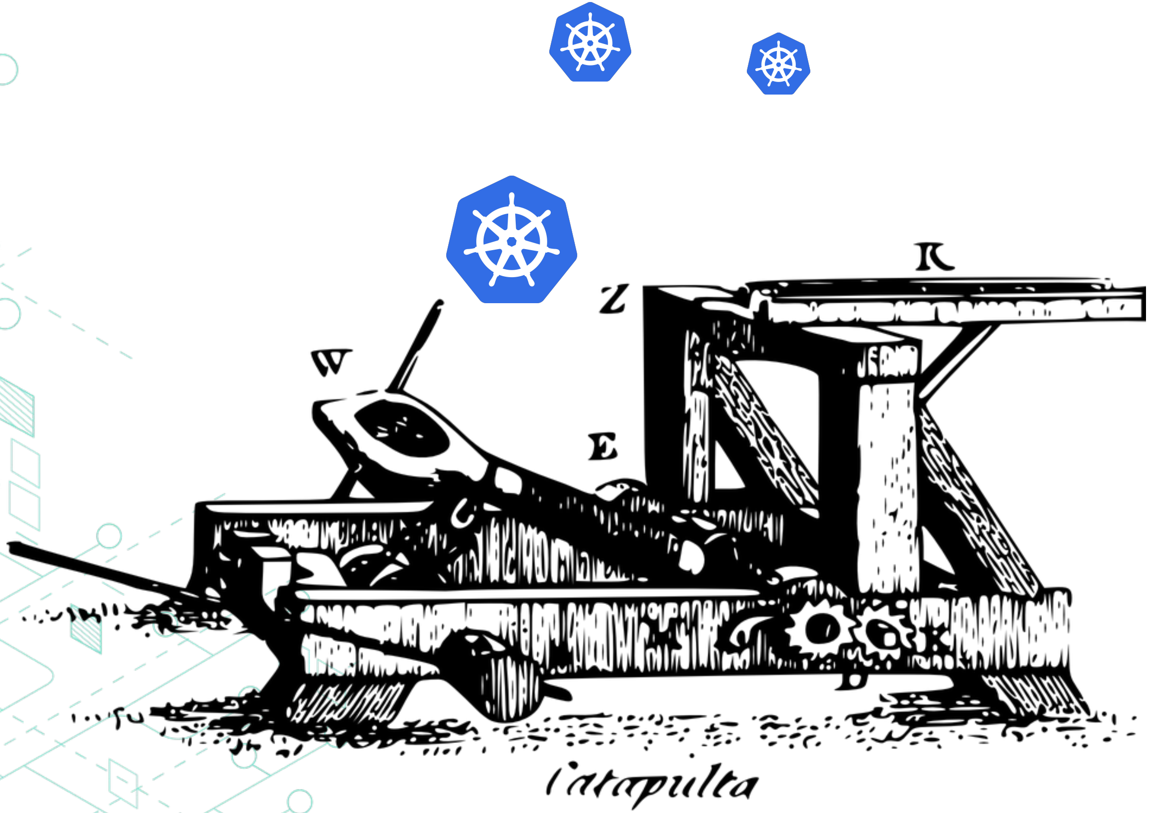
Kubernetes Deployment Strategies

Etienne Tremel
etienne.tremel@container-solutions.com

@etiennetremel

20.03.2018
Day of Cloud, Oslo

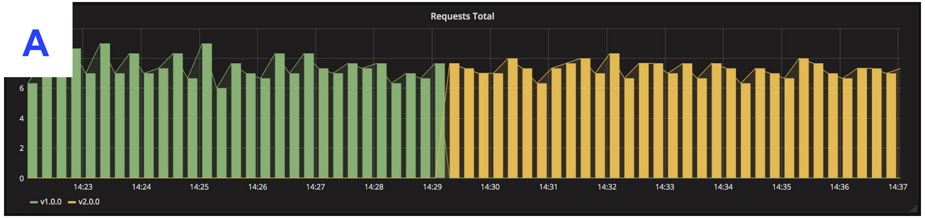


Container Solutions

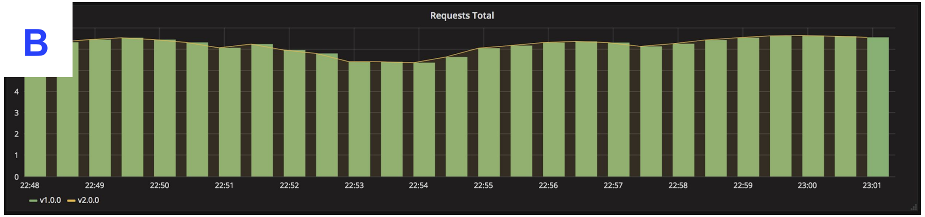
info@container-solutions.com
container-solutions.com

What is what?

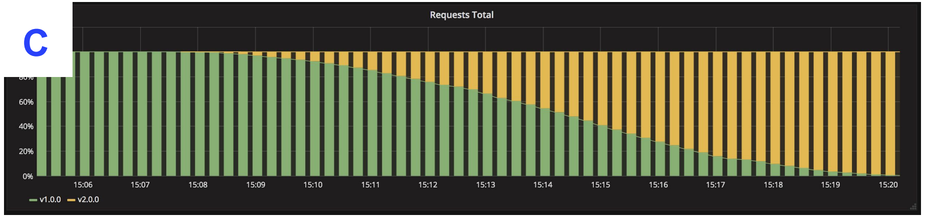
A



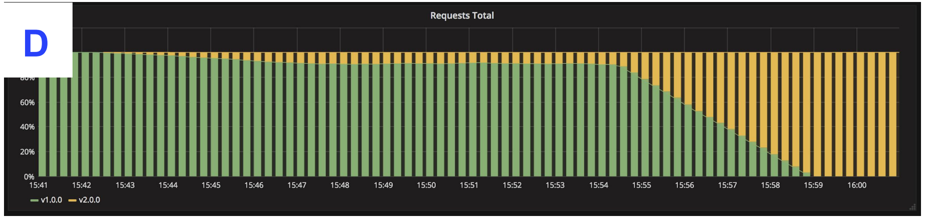
B



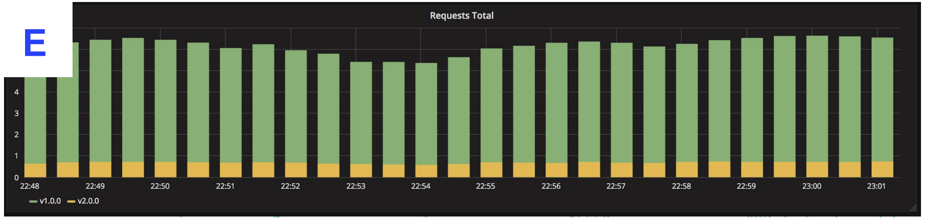
C



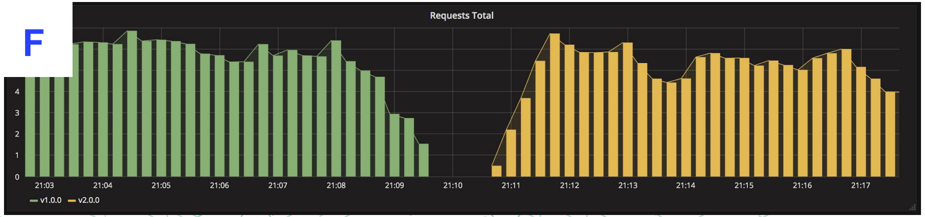
D



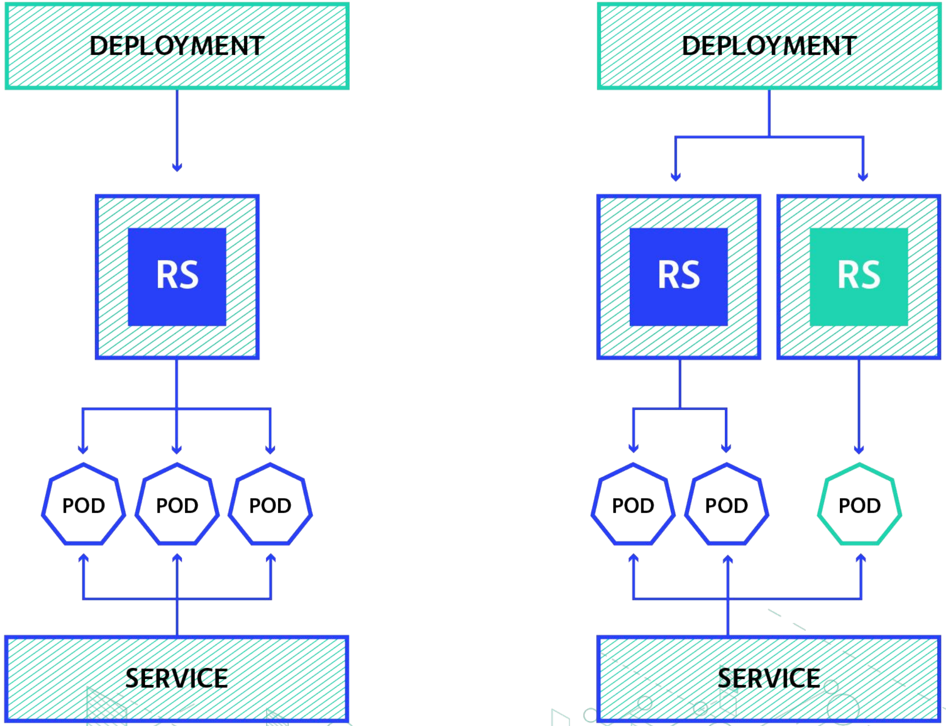
E



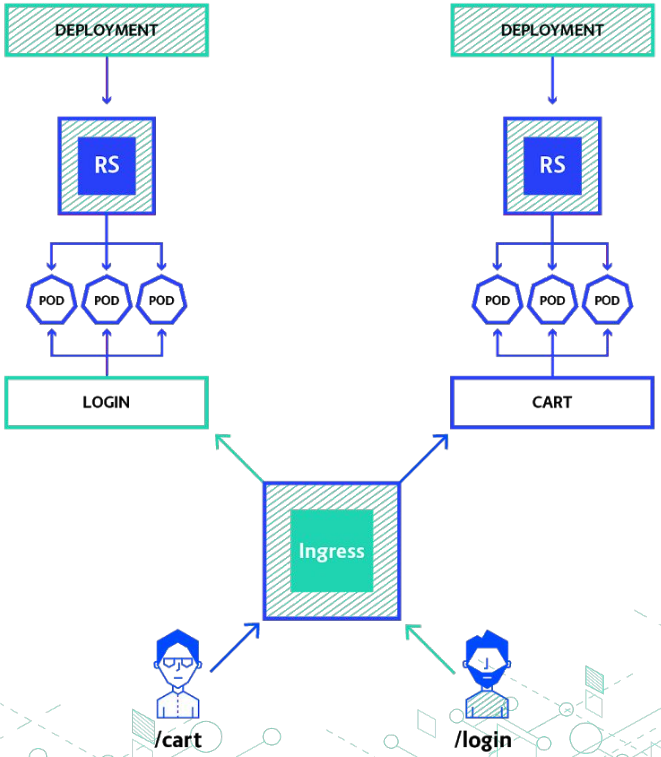
F



Kubernetes deployment: in brief



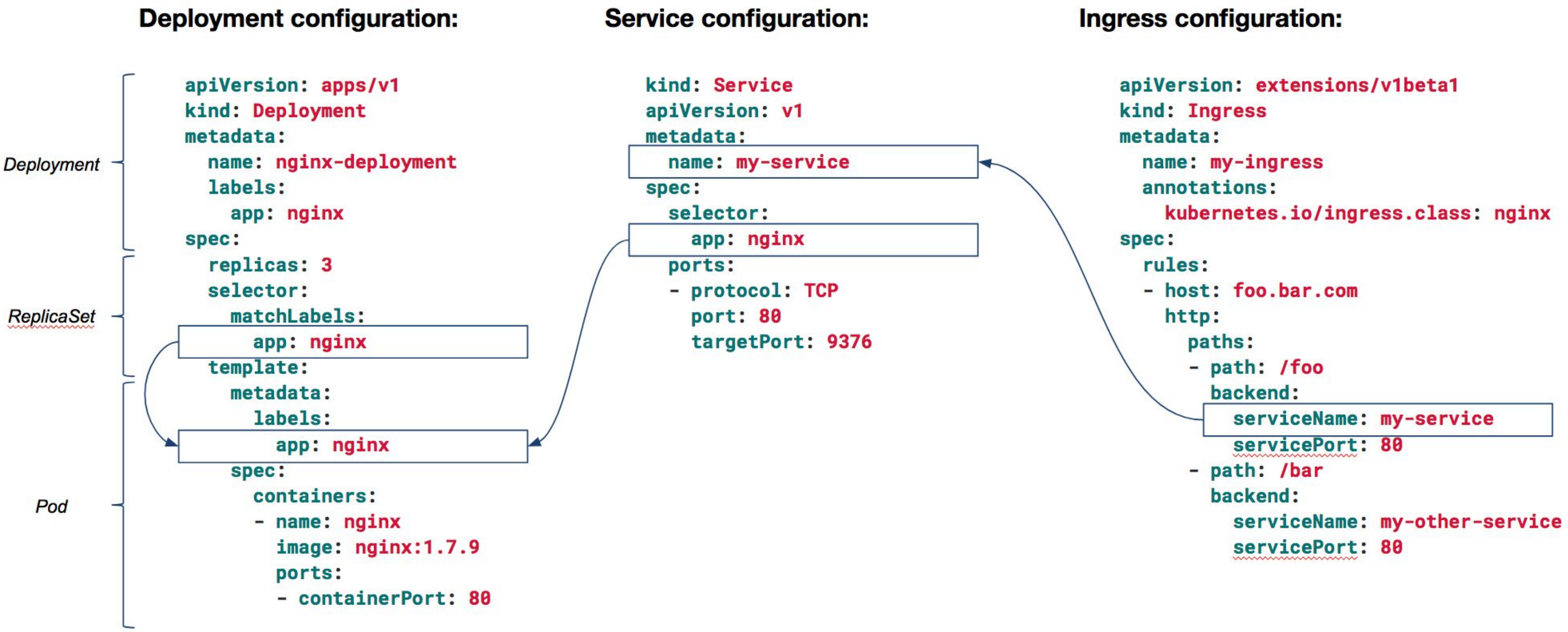
Kubernetes deployment: complex routing



Ingress controllers:

- Nginx
- Traefik
- Istio
- GKE
- etc.

Kubernetes deployment: configuration



Deployment configuration:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Service configuration:

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

Ingress configuration:

```yaml
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
```

Readiness/Liveness probe

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20
```

Readiness

Liveness

HTTP request:

```yaml
livenessProbe:
  httpGet:
    path: /healthz
    port: liveness-port
```

Any code greater than or equal to 200 and less than 400 indicates success. Any other code indicates failure.

Shell command:

```yaml
livenessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
```

Exit code return 0: healthy
Exit code return 1: unhealthy

Getting started

1. git clone -b gke https://github.com/ContainerSolutions/k8s-deployment-strategies

2. Play around with the different strategies

   - Recreate
   - Ramped
   - Blue/Green
   - Canary
   - A/B Testing
   - Shadow

Recreate

Version A is terminated then version B is rolled out

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

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

Recreate

Version A is terminated then version B is rolled out

Pros:

- easy to setup

Cons:

- high impact on the user, expect downtime that depends on both shutdown and boot duration of the application

Ramped (aka incremental, rolling update)

Version B is slowly rolled out and replacing version A

```
[...]
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
```

Ramped (aka incremental, rolling update)

Version B is slowly rolled out and replacing version A

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

Blue/Green (aka Red/Black)

Version B is released alongside version A, then the traffic is switched to version B

```
[...]
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
```

Blue/Green (aka Red/Black)

Version B is released alongside version A, then the traffic is switched to version B

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

Canary

Version B is released to a subset of users, then proceed to a full rollout

```
[...]
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
```

Canary

Version B is released to a subset of users, then proceed to a full rollout

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

Version B is released to a subset of users under specific condition

```
[...]
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
```

A/B Testing

Version B is released to a subset of users under specific condition

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

Shadow (aka mirrored)

Version B receives real-world traffic alongside version A and doesn’t impact the response.

```
[...]
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)

Version B receives real-world traffic alongside version A and doesn't impact the response.

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

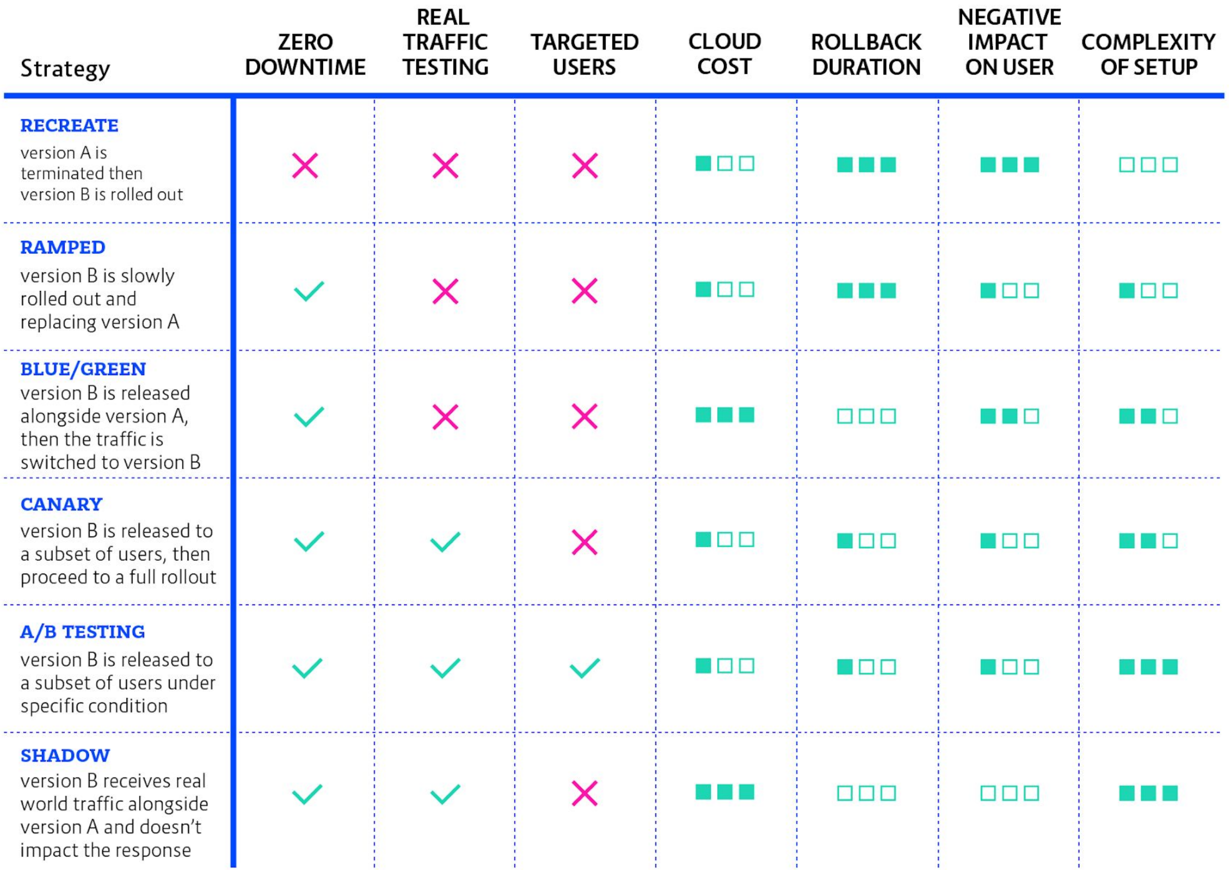
Cons:

- 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

Decision diagram



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	✓	✓	✗	■■■	□□□	□□□	■■■

Thanks!

Etienne Tremel
etienne.tremel@container-solutions.com

@etiennetremel

Container Solutions

info@container-solutions.com
container-solutions.com

Next

Hands on Kubernetes deployment strategies:
github.com/ContainerSolutions/k8s-deployment-strategies

Blog post about strategies:
container-solutions.com/kubernetes-deployment-strategies
thenewstack.io/deployment-strategies