Progressive Deployments mit Flagger

Einleitung

- Einführung zu Deploymentstrategien und progressiven Deployments
- Voraussetzungen
- Tooling

Progressive Delivery

- Warum progressive Delivery
  - Schnelleres kontrolliertes ausliefern von Code
  - Reduzieren des Risikos fehlerhafte Versionen Live zu setzen
  - Begrenzung des Work-In-Progress, häufigere Auslieferung
  - Bessere Customer Experience

- Voraussetzung für effektives DevOps
- Entfall der Planung/Diskussion wann Features bereitgestellt werden
- Endkunde kann in die Entwicklung durch Tests einbezogen werden

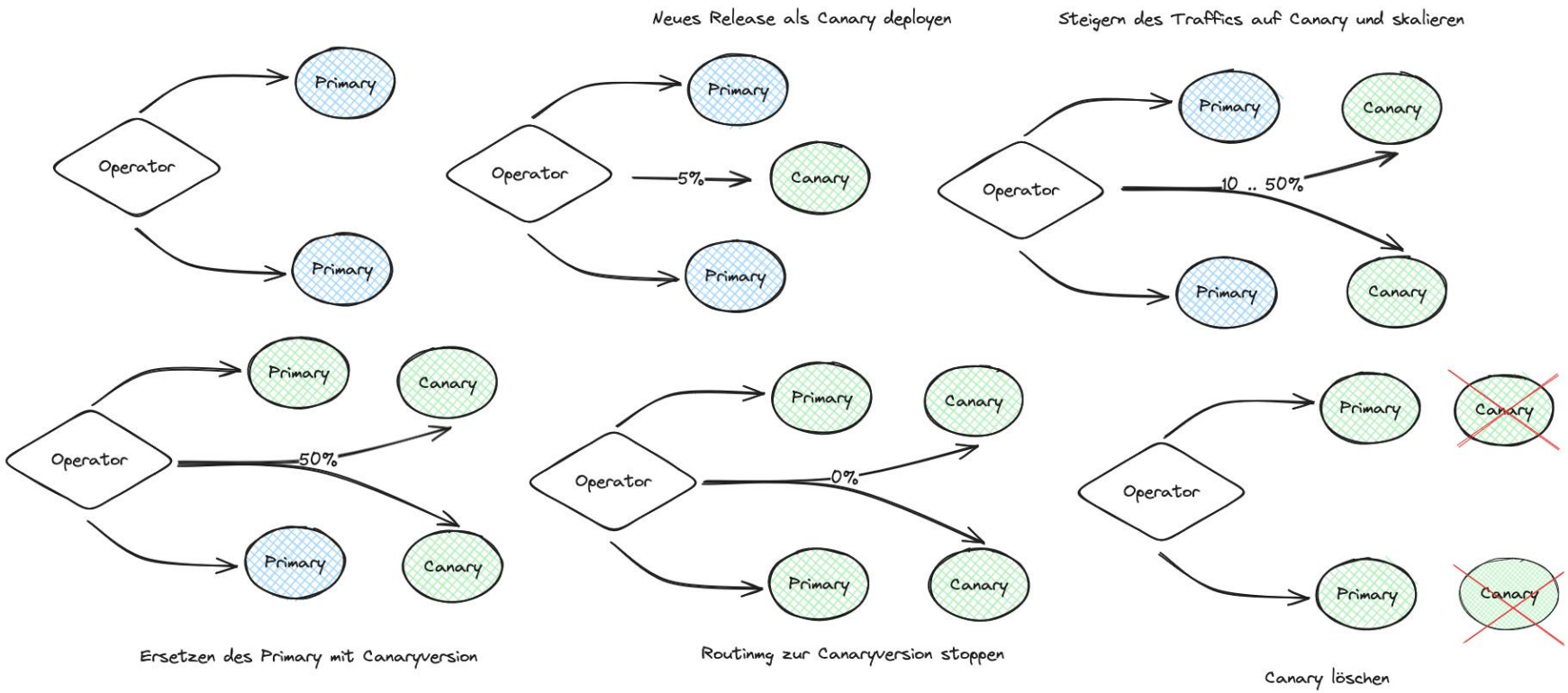
Deploymentstrategien

- Es wird immer ein Canary deployed, welches den zu testenden neuen Stand enthält
- Parallel dazu läuft immer der Primary mit dem aktuellen Stand
- Für progressive Delivery mehrere Strategien
  - Canary Release
  - A/B-Testing
  - Blue/Green-Deployment
  - Feature-Flag Deployment

Canary Release

- Es wird eine neue Version Deployed
- Geringer Useranteil wird auf die neue Version umgeleitet
- Anzahl Benutzer wird gesteigert
- Switchover auf die neue Version wenn alles erfolgreich
- Alte Version deinstallieren

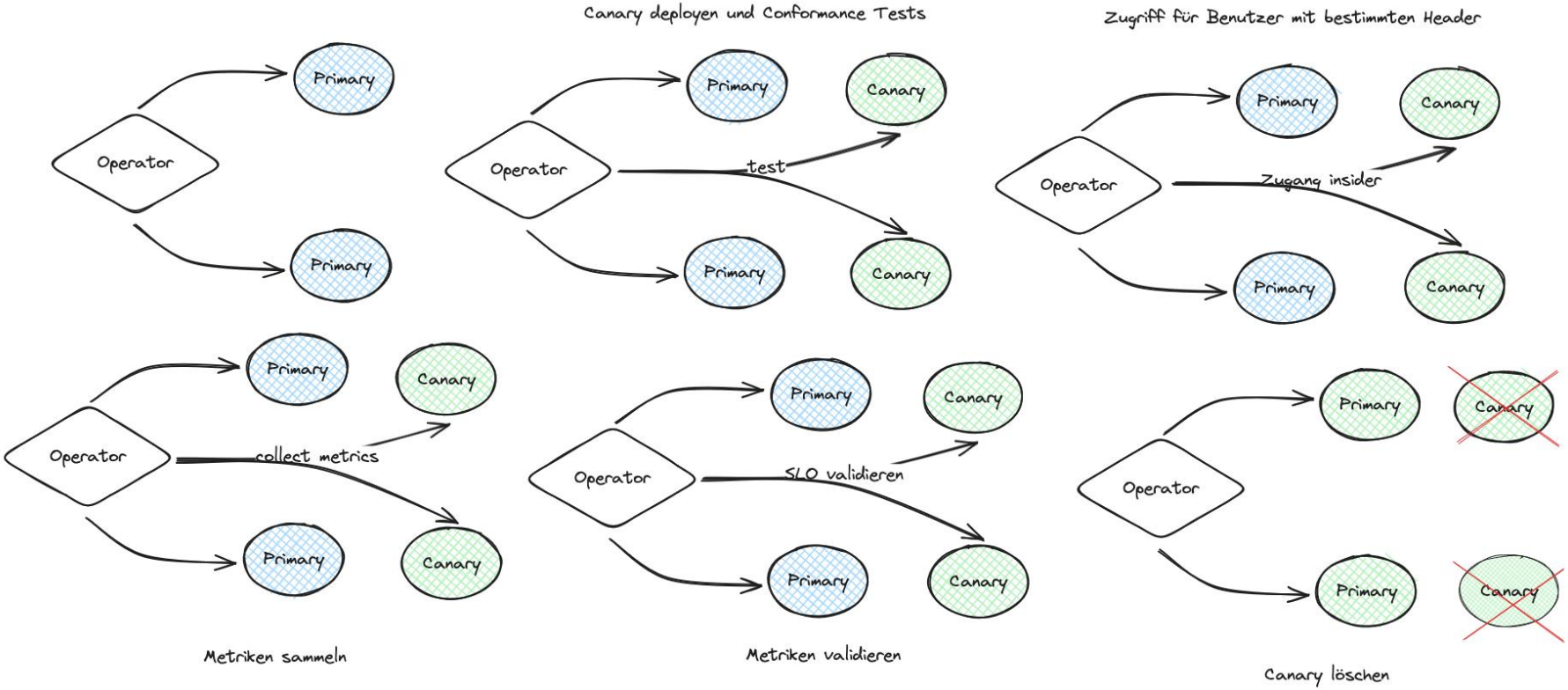
Canary Release



A/B Testing

- Traffic Routing auf die neue Version für eine bestimmte Benutzergruppe
- Durchführen von verschiedenen Tests auf die neue Version
- Verwendung von HTTP-Header oder Cookies
- Nützlich bei Frontend-Applikationen die Session-Affinity benötigen

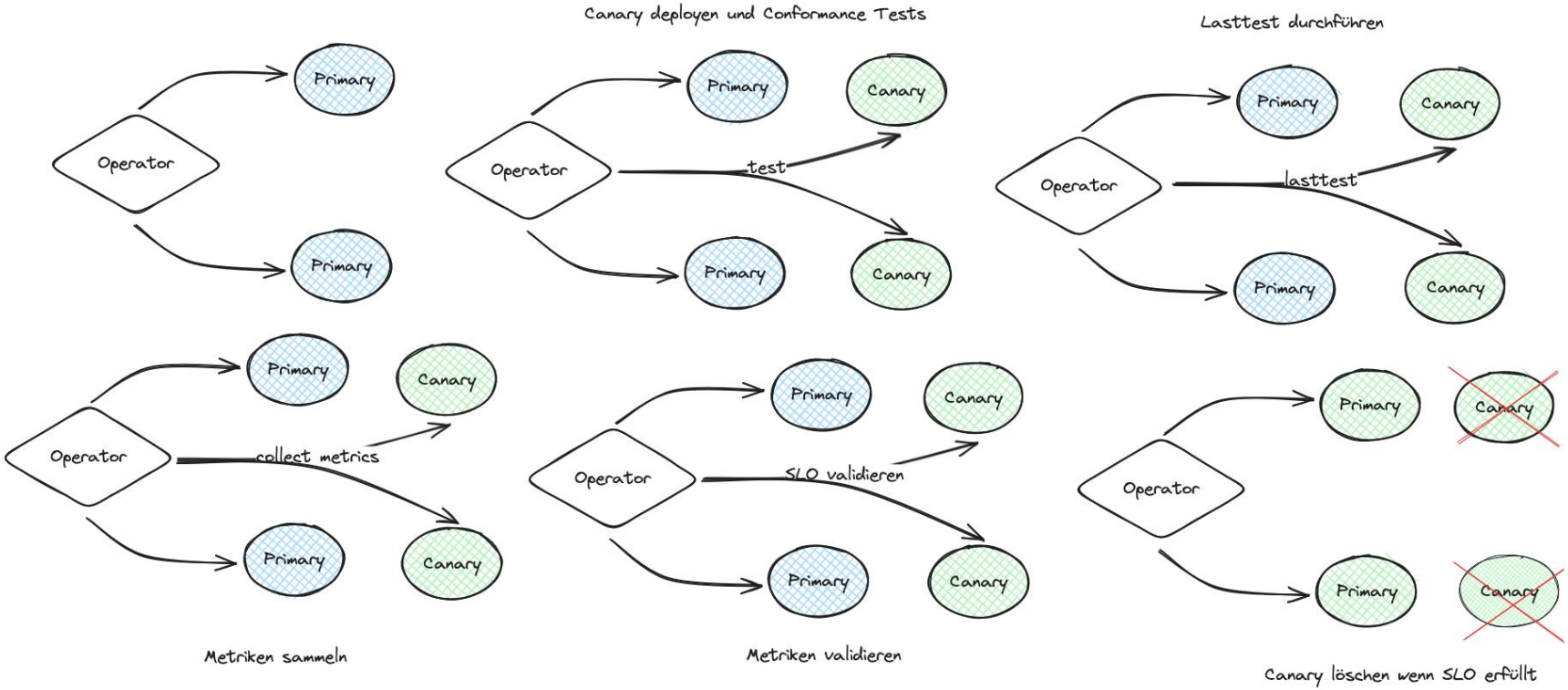
A/B Testing


Operator
Primary
Primary
Canary deployen und Conformance Tests
Operator
Primary
Canary
test
Primary
Canary
Zugriff für Benutzer mit bestimmten Header
Operator
Primary
Canary
Zugang insider
Primary
Canary
Operator
Primary
Canary
collect metrics
Primary
Canary
Metriken sammeln
Operator
Primary
Canary
SLO validieren
Primary
Canary
Metriken validieren
Operator
Primary
Canary
Primary
Canary
Canary löschen

Blue/Green Deployment

- Kann auch ohne Service/Mesh verwendet werden
- Traffic-Routing mit k8s CNI
- Mit Service-Mesh Traffic-Mirroring möglich, aber nur wenn Requests idempotent sind
  - Requests werden an Canary gespiegelt, aber Response kommt nur vom primary
  - Metriken auf dem Canary werden ausgewertet

Blue/Green Deployment



Überblick Flagger

- Operator zum automatisieren des progressiven Deployments
- Aktuelle Version 1.41.0
- Hat diverse Deploymentstrategien implementiert
- Benötigte Tools:
  - ServiceMesh (istio, linkerd, kuma usw.)
  - Oder IngressController (nginx, traefik, gloo, …)
  - Für Analysen prometheus, influxDB, usw.
  - Für Benachrichtigungen und Alerting ein Messenger (Slack, MS Teams, Discord, Rocket…)
- Kann in GitOps-Pipelines zusammen mit Flux, JenkinsX, ArgoCD etc. verwendet werden

Installation Flagger

- Mehrere Optionen: helm, kustomize
- Installations-Manifests abhängig vom verwendeten ServiceMesh oder Generic bei Verwendung mit IngressControllern
- Installation mit Flux
- Es wird Operator bereitgestellt
- Installation stellt drei Custom Resources bereit
  - Canary
  - MetricsTemplate
  - AlertProvider

Flagger CRDs

- Canary: zentrale Konfiguration des Deploymentprozesses
- MetricsTemplate: Definition von eigenen Metrik-Queries um Daten für Thresholds zu ermitteln
- AlertProvider: Konfiguration von speziellen Benachrichtigungen für einzelne Deployments

Funktion

- Definition des Releaseprozesses über Canary CRD
- Als Ziel Deployment oder Daemonset
- Trigger Anpassungen an Deployment oder entsprechenden ConfigMaps oder Secrets
- Flagger erstellt passende Ressourcen auf Basis der Definition z.B Services, VirtualServices, DestinationRules
- Durchführen von Conformance-Tests oder Triggern von LoadTests über Webhooks
- Notification und Alerting über AlertProvider

Canary CRD

- Definiert den Releaseprozess
- Verschiedene Sektionen zur Definition der Ziele, Thresholds und Aktionen
- Ziele
  - Als Ziele können Deployments oder Daemonsets definiert werden
- Analyse
  - Definition von eigenen Metrik-Queries um Daten für Thresholds zu ermitteln
  - Definition von Webhookaufrufen um load oder Conformacetests zu starten gegen das Canary-Deployment
  - Konfiguration von speziellen Benachrichtigungen für einzelne Deployments

Canary target

```
spec:
    targetRef:
        apiVersion: apps/v1
        kind: Deployment
        name: backend
    progressDeadlineSeconds: 120
    autoscalerRef:
        apiVersion: autoscaling/v2beta2
        kind: HorizontalPodAutoscaler
        name: backend
```

- Target-Spezifikation in Canary
- Optional kann HPA definiert werden
- Timeout für das Ausrollen und die Tests
- Mit dieser spec werden folgende Artefakte angelegt:
  `deployment/<targetRef.name>-primary`
  `hpa/<autoscalerRef.name>-primary`

Canary service


```
spec:
  service:
    name: podinfo
    port: 9898
    portName: http
    appProtocol: http
    targetPort: 9898
    portDiscovery: true
```

- Service-Spezifikation in Canary Resource
- Legt fest wie der Workload innerhalb des Clusters verfügbar ist
- Mit dieser spec werden folgende Artefakte angelegt:
  `<service.name>.<namespace>.svc.cluster.local`
  `hpa/<autoscalerRef.name>-primary`

Canary service

- Die Artefakte werden über das label „app“ zugeordnet
- Mit dieser spec werden folgende ClusterIP-Services angelegt:

  `<service.name>.<namespace>.svc.cluster.local`
  selector `app=<name>-primary`

  `<service.name>-primary.<namespace>.svc.cluster.local`
  selector `app=<name>-primary`

  `<service.name>-canary.<namespace>.svc.cluster.local`
  selector `app=<name>`
- Es können auch URI matches und Rewrite-Rules angegeben werden
- Für istio können noch CORS, gateways und traffic policies definiert werden

Canary status

- Der Status kann über die Resource abgefragt werden

  `kubectl get canaries -A`

  selector `app=<name>-primary`

```
NAMESPACE NAME      STATUS      WEIGHT LASTTRANSITIONTIME
test      podinfo   Progressing 15   2019-06-30T14:05:07Z
prod      frontend  Succeeded   0    2019-06-30T16:15:07Z
prod      backend   Failed      0    2019-06-30T17:05:07Z
```

Canary analysis

- In diesem Abschnitt wird die Deploymentstrategie definiert
- Welche Metriken verwendet werden
- Welche Webhooks für Tests
- Welche Alert-Endpunkte
- Analysis wird periodisch ausgeführt, bis alle Bedingungen erfüllt sind
- Im Fehlerfall wird Canary zurückgerollt
- Canary kann auch mit setzen von suspend pausiert werden

Build-in Metrics

- 2 Buildin Metriken: reques-success-rate und request-duration
- Implementiert mit Prometheus-Queries, für jedes Mesh und Ingress verfügbar

```
metrics:
- name: request-success-rate
  interval: 1m
  # minimum req success rate (non 5xx responses)
  # percentage (0-100)
  thresholdRange: min: 99
- name: request-duration
  interval: 1m
  # maximum req duration P99
  # milliseconds
  thresholdRange: max: 500
```

Custom Metrics

- Definition von Custom-Metrics mit MetricsTemplate Ressource
- Verschiedene Typen prometheus, datadog oder andere
- Definition der Datenquelle und der Abfrage

```
metrics:
- name: request-success-rate
  interval: 1m
  # minimum req success rate (non 5xx responses)
  # percentage (0-100)
  thresholdRange: min: 99
- name: request-duration
  interval: 1m
  # maximum req duration P99
  # milliseconds
  thresholdRange: max: 500
```

Custom Metrics


```
apiVersion: flagger.app/v1beta1
kind: MetricTemplate
metadata:
  name: latency
  namespace: istio-system
spec:
  provider:
    type: prometheus
    address: http://flagger-prometheus.istio-system:9090
  query: |
   histogram_quantile(
     0.99,
     sum(
        rate(
          istio_request_duration_milliseconds_bucket{
            reporter="destination",
            destination_workload_namespace="{{ namespace }}",
            destination_workload=~"{{ target }}"
          }[{{ interval }}]
          )
      ) by (le)
   )
```

Webhooks

- Erweiterung der Metrik-Analyse mit Webhooks
- Wenn 200er als Antwort, dann erfolgreich
- Verschiedene Typen die Ausführungszeitpunkt der Hooks bestimmen
- Beispiel pre-rollout

```
webhooks:
- name: conformance-test
  type: pre-rollout
  timeout: 15s
  url: http://flagger-loadtester.prod.svc.cluster.local/
  metadata:
    type: bash
    cmd: "curl -sd 'test' http://frontend-canary.prod:9898/token | grep token"
```

Webhooks Manual gating

- Mit Webhooks kann auch manuelle Bestätigung realisiert werden
- Dazu flagger loadttester verwenden mit Gating im confirm-rollout
- Entweder komplett anhalten mit `gate/halt`
- Canary wird auf State Waiting gesetzt
- Wieder freigeben mit `gate/approve`
- Oder als Check mit `gate/check`
- Dann im Container `gate/open` oder `gate/close` absetzen zum starten oder anhalten

Webhooks Loadtesting

- Mit Webhooks Loadtest mittels Flagger loadtesttool durchgeführt werden
- Dazu werden Requests an Komponente geschickt
- Wenn zu langsame oder keine Antwort schlägt Stage fehl

Demo

Aufbau Demo

- K8s-Cluster mit istio ServiceMesh
- Kleine Testanwendung mit Frontend und Backend
- Verschiedene Canaries, mit CanaryRelease und A/B-Testing

Thorsten Wussow

SLIX Gesellschaft für Computersysteme mbH
Nandlstädter Weg 6
84072 Au i. D. Hallertau

phone +49 1733208013
mail thorsten.wussow@slix.de
web www.slix.de