Percentage-based Canary for Microservices

https://tutuplapak.atlassian.net/browse/DPT-412

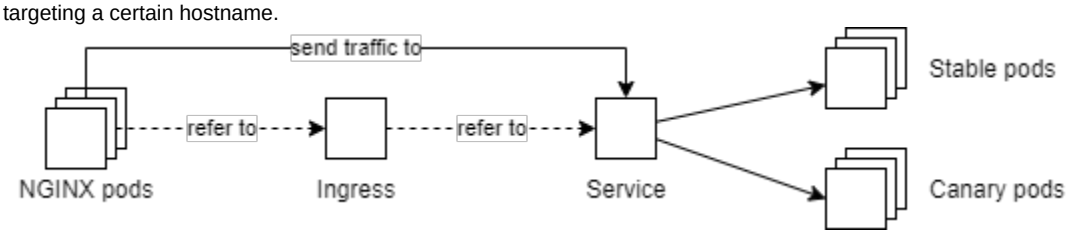
> This document is still a draft. Comments, suggestions, and questions are welcome.

Table of contents

Current approach

Currently, each microservice has two Kubernetes deployments in production: one that hosts stable code and one that hosts a new code (we call it “canary”). The canary deployment will spawn a single pod.

Those two deployments are then exposed through a single Kubernetes service, which will then exposed through a single Kubernetes ingress targeting a certain hostname.



Traffic is distributed evenly by the Kubernetes service to all pods behind the service. Hence, we can only control the amount of traffic going to canary pods by adjusting the **proportion of numbers between canary pods and stable pods**. For example, a microservice with 3 stable pods and 1 canary pod will have 25% of the traffic going to the canary pod.

Due to how the traffic is distributed, **we can’t adjust the portion of traffic going to canary pods lower** – e.g. to 10% – **without increasing the number of stable pods** – e.g. to 9 pods.

After the stable pods use the new code, the canary pods keep serving traffic. On the next code deploy, the canary pod will use the newer code.

It is possible that a canary deployment goes wrong and needs to be rolled back. Currently, the deployment pipeline provides a “canary-rollback” job, which will **delete canary pods**.

For reference, a common deployment pipeline can be seen here.

Requirements

Service owners need to be able to control the amount of traffic that will go to canary pods regardless of the proportion of numbers between canary pods and stable pods.

For example, even if a microservice has 3 stable pods and 1 canary pod, we want to able to route only 10% of total traffic to the canary pod.

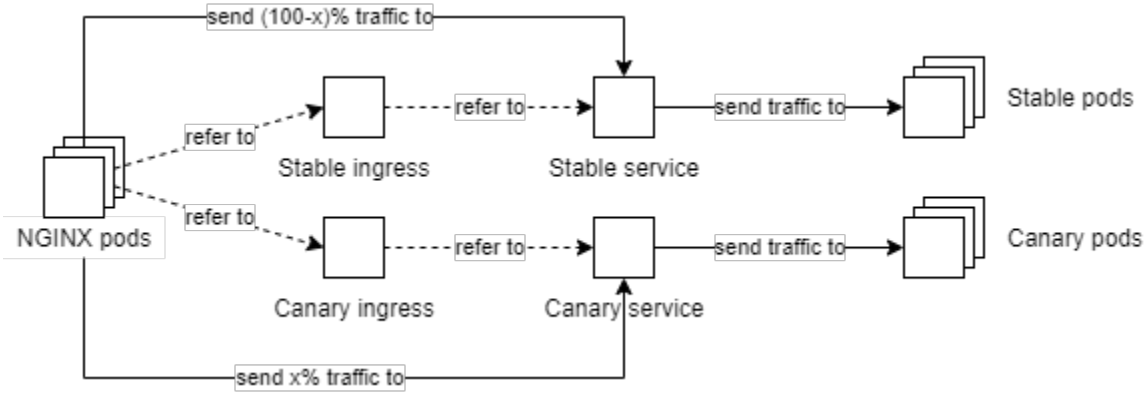
Constraint

We keep using NGINX ingress installed managed by Kubernetes & Load Balancer team to do canary to minimize changes while providing a functional percentage-based canary feature for our microservices.

Another kind of solutions like using service mesh, while fancy, will bring more complexities to be handled.

New approach

The existing NGINX ingress that we use to expose microservices now is able to handle canary traffic distribution. This is done by having two Kubernetes ingress objects for a microservice: stable ingress and canary ingress. The stable ingress will expose the stable service, which only selects stable pods. The canary ingress will expose the canary service, which only selects canary pods.



The percentage of traffic will then be configured in the canary ingress.

It is possible that a canary deployment goes wrong and needs to be rolled back. Currently, the deployment pipeline provides a “canary-rollback” job, which will **delete canary ingress, service, and pods**.

Limitation

- Weight can only be defined in integer, from 1% to 100%. (ref)

Changes in pipeline side

Kubernetes manifests of a microservice are stored in Minerva, grouped per component (e.g. `api`, `background`, …). Each component contains one or more Kubernetes manifest to be applied to the cluster. This doesn’t include `ConfigMap` generated from configuration inside `service/<name>/config/<env>` directories and `Secret` generated from Vault.

```
minerva
 services
     <service name>
         config
            kubeconfig.yaml
            kubelize
            preproduction
               configmap
            production
                configmap
         manifest
             <component name>
                deployment.yml
                ingress.yml
                resourcequota.yaml
                service.yml
             <component name>
                 cronjob.yml
                 deployment.yml
```

On deploying canary pods, the pipeline will deploy only `deployment.yml` files.

The new approach will require us to deploy `ingress.yml` and `service.yml` also.

Changes in service owner side

kubelize configuration

Changes in `config/kubelize` file in Minerva:

```
projectName: sample-examples
services:
- name: web
  port: "8888"
  language: go
  environments:
  [... redacted ...]
  canaryWeight: 10
  minCpu: "1"
  maxCpu: "2"
  minMem: "0.5"
  maxMem: "2"
  minReady: 60
  [... redacted ...]
```

Changes:

- Add `canaryWeight` in `config/kubelize` file and set to the percentage of traffic that we want to redirect to canary pods

Kubernetes Ingress

Changes in `manifest/<name>/ingress.yml` file in Minerva:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: {{.config.projectName}}-{{.service.name}}-{{.environment.name}}
  namespace: {{.config.projectName}}
  annotations:
    kubernetes.io/ingress.class: nginx-internal

    {{- if eq .environment.name "canary" }}
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "{{.service.
canaryWeight}}"
    {{- end }}

    # If needed, set additional NGINX annotations here.
    # Ref: https://kubernetes.github.io/ingress-nginx/user-guide/nginx-
configuration/annotations/
    #
    # Example:
    # - If your service requires more than 1MB for upload, add:
    #     nginx.ingress.kubernetes.io/proxy-body-size: <desired limit>
    #   The default value is 1m (http://nginx.org/en/docs/http
/ngx_http_core_module.html#client_max_body_size).
  labels:
    project: {{.config.projectName}}
    service: {{.service.name}}
    env: {{.environment.name}}
spec:
  rules:
  - host: {{.config.projectName}}.{{or .variable.DNS_ENV .environment.
name}}.bl-cloud.internal
    http:
      paths:
      - path: /
        backend:
          serviceName: {{.config.projectName}}-{{.service.name}}-{{.
environment.name}}
          servicePort: {{.service.port}}
```

Changes:

- Add `nginx.ingress.kubernetes.io/canary` annotations
- Change the `host` declaration to optionally use `DNS_ENV` environment variable
  - This is needed so we can use `production` as DNS host on `canary` environment
  - `DNS_ENV` will be provided in the pipeline

Kubernetes Service

Changes in `manifest/<name>/service.yml` file in Minerva:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    project: {{.config.projectName}}
    service: {{.service.name}}
    env: {{.environment.name}}
  name: {{.config.projectName}}-{{.service.name}}-{{.environment.name}}
  namespace: {{.config.projectName}}
spec:
  selector:
    project: {{.config.projectName}}
    service: {{.service.name}}
    env: {{.environment.name}}
  ports:
  - name: http
    protocol: TCP
    port: {{.service.port}}
    targetPort: {{.service.port}}
  type: ClusterIP
```

Changes:

- Add `env` in `.spec.selector`

Test results

The approach above is tested using Alfred, an internal Deployment Tooling service.

- Changes in the pipeline side: https://gitlab.cloud.bukalapak.io/infra/gitops/minerva/-/compare/master...percentage-based-canary
- Changes in Minerva for Alfred: https://gitlab.cloud.bukalapak.io/infra/gitops/minerva/-/commit/e406fceee1047eb213e45c91bfd5ad030f8d61a7
- Changes in Alfred repository: https://gitlab.cloud.bukalapak.io/bukalapak/alfred/-/commit/d309a35d1f72f0e96d27d14cd8153a1564f084b0

We define the weight of canary traffic to be 25%.

- Ingress log: https://kibana-logs.prod.bukalapak.io/goto/5f9396249265f3ac65cf4a5630ee5159
- Application log: https://kibana-logs.prod.bukalapak.io/goto/d82da95ace0b9063c94b8d2df15f24d9
- Datadog metric in NGINX side: https://app.datadoghq.com/notebook/203221/metrics-from-nginx-ingress-in-percentage-based-canary

Findings

- From the ingress log, we can see that canary traffic is indicated in `upstream_name_alternative` field.
- We can't analyze canary metrics from the ingress side, as NGINX attributes all traffic (including canary traffic) in the same `alfred-api-production` label in Datadog.

Discussions

- Suggestions from `@ Sugandi`:
  - Do not define traffic weight in `kubelize`; instead, do it in the pipeline level

    - Still adjustable by service owners, i.e. in `.gitlab-ci.yml`
- By doing that, we can gradually increase canary traffic (e.g. 1% 5% 10% before full rollout)
    - The pipeline will be longer but will result in fewer change errors rollout to production