POC - Canary Deployment

Assumption:

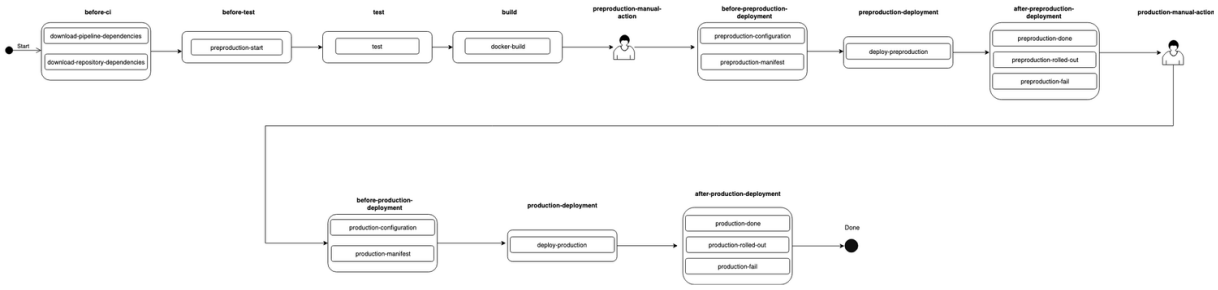
1. We need to deploy the application to production to test the functionality with live traffic as we would be creating the service and ingress in the Production stage
2. Deployment of the canary will not deploy any ingress or service manifest. Therefore on new service deployment, these will need to be created by a full production release.
3. While using canary deployment for the existing pipeline we need to manually remove the label **env: production** from the **service definition** for the first time as we need to redirect the live traffic to the canary deployment. Once the pipeline is deployed using canary from the next time the traffic will be redirected to the canary deployment by default, then it would be developers' choice to whether delete the canary deployment after monitoring before deploying to production or the user can delete the canary deployment after the deployment to production.
4. For POC purpose we have tested the canary deployment in the pre-prod setup as we didn't want to disturb the production setup. But the actual canary will be deployed in the production setup. But the document has changes needed for the production setup.
5. The approach must be to use the same deployment file so as to have a single source for all deployments in all the environments. The previous approach as per the link was not considered as we were adding new manifest files and config for canary deployment.

This document will explain all the necessary steps and implementation details required to complete the POC. All the code changes are pushed in **canary-deploy** branch in the Minerva repository.

1. We will add four new stages before the **production-manual-action** stage in the current pipeline for the canary deployment process.

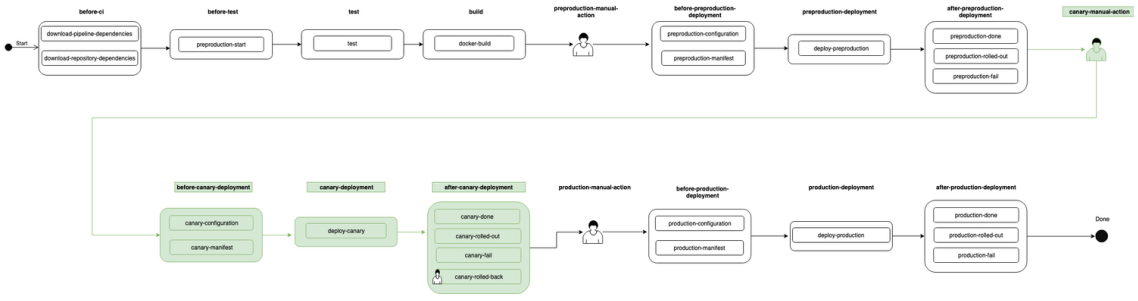
2. The implementation of the new stages in the current flow is explained as showed in the diagrams below:

    a. Current Flow



    b. Canary Flow

        i. The canary stages are added in green color to the current flow.



        ii. The additional canary stages will do the following operations mentioned below:

1. **canary-manual-action**: This will be a manual step where a user approval would be needed to go to the next stage which is before-canary-deployment.

2. **before-canary-deployment:** This stage will first create new configmaps and secrets for the canary deployment in the **production namespace**. It will also create new manifest files for k8's resources other than the service and ingress as we would be using the same service and ingress as that of **production.**

3. **canary-deployment:** This stage will deploy the new k8's file created in the previous stage in the **production namespace.**

4. **after-canary-deployment:** This stage is very similar to the other after-deployment stage where we send notifications to the slack, but we have added a new job named: **canary-rolled-back** which would be our rollback strategy if the canary deployment fails.

   NOTE: The **canary-rolled-back** job will be a manual step where the user input is needed if in case we need to delete the canary-deployment. This is a new functionality that we have added to the current pipeline where we can delete a deployment if needed.

3. The four stages would be named as below:

   a. **canary-manual-action**
      This would be a manual action after the **after-preproduction-deployment** stage in the current pipeline. This stage would be followed by the stages mentioned below.

   b. **before-canary-deployment**
      This stage would contain two jobs named:
      i. canary-configuration
      ii. canary-manifest

   c. **canary-deployment**
      This stage would contain a job named:
      i. deploy-canary

   d. **after-canary-deployment**
      This stage would contain four jobs named:
      i. canary-done
      ii. canary-fail
      iii. canary-rollback
      iv. canary-rollout

      Note: All the above stages are added considering the current flow to maintain the best practices followed in the current pipeline.

4. PFB the changes that would be required in the current pipeline for the additional stages to be added for Canary deployment.

   a. Add the above stages in the required order in the current flow

      The stages mentioned above would be added to the required order in the stages.yaml file.

```
stages:
- before-ci
- before-test
- test
- after-test
- before-build
- build
- after-build
- before-deployment
- deployment
```

```
- after-deployment
- preproduction-manual-action
- before-preproduction-deployment
- preproduction-deployment
- after-preproduction-deployment
- canary-manual-action
- before-canary-deployment
- canary-deployment
- after-canary-deployment
- production-manual-action
- before-production-deployment
- production-deployment
- after-production-deployment
- after-ci
```

b. Changes required for the canary-manual-action stage:

   i. The following content would be added for the manual action in the **.gitlab-ci-general.yaml.**

```
#
# Canary notification jobs
#
canary-start:
  extends: .releaselog-start
  stage: canary-manual-action
  when: manual
  allow_failure: false
  variables:
    NOTIFICATION_ENV: canary
```

   Note: The environment would be considered canary.

c. Changes required for the **before-canary-deployment** stage canary-configuration job:

   i. The following content would be added for the **canary-configuration** job in the **.gitlab-ci-general.yaml.**

```
############################################################
# GENERATING and VALIDATING MANIFEST
############################################################
canary-configuration:
  extends: .configuration-deployment
  stage: before-canary-deployment
  tags:
  - shared
  - production
```

```
  variables:
    KUBE_CLUSTER: preproduction
    ENV: canary
    CONFIG_PATH: "minerva/services/$CI_PROJECT_NAME/config
/production"
```

Note: The environment considered is canary as we would be using different config maps and secrets as compared to production. This will be helpful in scenarios where we need to change configmaps or secrets and if we consider using same configmaps and secrets we would bring down the stable deployment as we would change them to test in canary.

d. Changes required for the **before-canary-deployment** for the canary-manifest job:

   i. The following content would be added for the **canary-manifest** job in the **.gitlab-ci-general.yaml.**

```
canary-manifest:
  extends: .canary-generate-and-validate-manifest
  stage: before-canary-deployment
  variables:
    ENV: canary
    KUBE_CLUSTER: production
  tags:
  - shared
  - production
```

   Here we would be creating a new job named canary-generate-and-validate-manifest in the **job-template.yaml** which will be having other MANIFEST_PATH value which would be pointing to the k8's templated needed for canary deployment.

   ii. We would be adding the following code to the **job-template.yaml**

```
.canary-generate-and-validate-manifest:
extends: .deploy-template
stage: before-deployment
variables:
OUTPUT_PATH: deploy/_output/$ENV/manifest
FILEPATH: deploy/_output/$ENV/manifest
MANIFEST_PATH: minerva/services/$CI_PROJECT_NAME/manifest
KUBELIZE_CONFIG_PATH: minerva/services/$CI_PROJECT_NAME
/config/kubelize
only:
refs:
- master
- /^release.+/
artifacts:
paths:
- $OUTPUT_PATH
expire_in: 7 days
```

```
script:
- ${BIN_RENDERER_CANARY}
- ${BIN_APPLIER} validate
```

Here in this job, we have used ${BIN_RENDERER_CANARY} instead of ${BIN_RENDERER} as we just need to render the deployment files (as off now for POC purpose for Gerbata application) and no service or ingress files.

The following data is also added to the variables section of the **job-template.yaml** for the location of a new script.

```
BIN_RENDERER_CANARY: $BIN_PATH/renderer-canary.sh
```

The new script is added to the following location: *minerva/shared/bin/renderer-canary.sh*. PFB the script:

```
#!/bin/bash
# this bash script will be used to run the kubelize
command
# and add necessary variables to the command so it will
render the file using
# specified kubelize config and template
#

source "${BIN_PATH}/.common.sh"

echo -e "${GREEN}Rendering File using kubelize${NN}"

mkdir -p $OUTPUT_PATH
for yml in $(find $MANIFEST_PATH -type f -a -name
"deployment.yml" -o -name "deployment.yaml" | sed -e
"s#${MANIFEST_PATH}/##g"); do
  svc=$(grep -Eo "^[^/]+" <<< $yml);
  echo -e "Generating file for ${yml}"
  kubelize genfile --overwrite -c ${KUBELIZE_CONFIG_PATH}
-s ${svc} -e ${ENV} ${MANIFEST_PATH}/${yml} ${OUTPUT_PATH}
/${svc}/;
done

echo -e "Rendering File completed"
```

Note: We need to add an additional filter to the find command if the application has other k8's resources other than deployment like stateful sets, etc

e. Changes required for the **canary-deployment** stage:

   i. The following content would be added for the **deploy-canary** job in the **.gitlab-ci-general.yaml.**

```
deploy-canary:
  extends: .deploy-template
  stage: canary-deployment
  variables:
    ENV: canary
    FILEPATH: deploy/_output/$ENV/manifest
    KUBE_CLUSTER: production
  tags:
  - shared
  - production
  environment:
    name: canary
  artifacts:
    paths:
    - ${FILEPATH}/.rollout_check.list
```

f. Changes required for **after-canary-deployment** stage canary-done job:

i. The following content would be added for the **canary-done** job in the **.gitlab-ci-general.yaml.**

```
canary-done:
  extends: .releaselog-done
  stage: after-canary-deployment
  variables:
    NOTIFICATION_ENV: canary
```

g. Changes required for **after-canary-deployment** stage canary-fail job:

i. The following content would be added for the canary-fail job in the **.gitlab-ci-general.yaml.**

```
canary-fail:
  extends: .releaselog-failed
  stage: after-canary-deployment
  dependencies:
  - canary-start
  - download-pipeline-dependencies
  variables:
    NOTIFICATION_ENV: canary
```

h. Changes required for **after-canary-deployment** stage canary-rolled-back job:

i. The following content would be added for the **canary-rolled-back** job in the **.gitlab-ci-general.yaml.**

```
#
# Canary Rollback job
#

canary-rolled-back:
  extends: .rollback-template
  stage: after-canary-deployment
  variables:
    ENV: canary
    FILEPATH: deploy/_output/$ENV/manifest
    KUBE_CLUSTER: production
  tags:
  - shared
  - production
  environment:
    name: canary
  artifacts:
    paths:
    - ${FILEPATH}/.rollout_check.list
```

ii. We also need to add a new job in the job-template **.yaml.** PFB the content:

```
.rollback-template:
  stage: after-deployment
  when: manual
  image: ${CI_REGISTRY}/sre/gudang/deployment:3.3.0
  variables:
    FILEPATH: deploy/_output/
    KUBE_CLUSTER: $ENV
    KUBECONFIG_PATH: minerva/services/$CI_PROJECT_NAME
/config/kubeconfig.yaml
    KUBE_NAMESPACE: $CI_PROJECT_NAME
  only:
    refs:
    - master
    - /^release.+/
  before_script:
  - source ${SH_SET_TOKEN}
  - export FILEPATH=${VAR_FILEPATH:-$FILEPATH}
  - export KUBECONFIG_PATH=${VAR_KUBECONFIG_PATH:
-$KUBECONFIG_PATH}
  - export KUBE_CLUSTER=${VAR_KUBE_CLUSTER:-$KUBE_CLUSTER}
  - export KUBE_NAMESPACE=${VAR_KUBE_NAMESPACE:
-$KUBE_NAMESPACE}
  - export VERSION=${CI_COMMIT_TAG:-$CI_COMMIT_SHA}
  - export TIMESTAMP=`date +"%Y-%m-%dT%H:%M:%SZ"`
```

```
  script:
  - ${BIN_APPLIER} delete
```

Note: This would be a manual step in which we need to trigger the rollback after we observe that the canary pods are not performing as per the requirement after monitoring them in DD dashboard.

iii. For the rollback, we need to add a new operation to the existing **applier.sh.** PFB the updated **applier.sh**

```
#!/bin/bash
#
# This is a bash script to validate and apply kubernetes
manifest files to a
# specific kubernetes cluster
# some variable needs to be exported as Environment
variable for this script
#
# requred variable are ENV and FILEPATH
#
# usage:
#  ./applier.sh [apply|validate/delete]
#
# apply will have artifact with content list of manifest
with type
# deployment, daemonset, and statefulset. The artifact
will be stored at
# ${FILEPATH}/.rollout_check.list, the list is
representing any manifest that
# can be monitored by kubectl rollout status
#

source "${BIN_PATH}/.common.sh"

build_command() {
  CMD="kubectl"
  if [ -n "$KUBECONFIG_PATH" ]; then CMD="$CMD --
kubeconfig $KUBECONFIG_PATH"; fi
  if [ -n "$KUBE_CLUSTER" ]; then CMD="$CMD --cluster
$KUBE_CLUSTER"; fi
  if [ -n "$TOKEN" ]; then CMD="$CMD --token $TOKEN"; fi
  if [ -n "$KUBE_NAMESPACE" ]; then CMD="$CMD --namespace
$KUBE_NAMESPACE"; fi
  if [ "$1" == "validating"  ]; then CMD="$CMD --dry-
run=true"; fi
  if [ "$1" == "deleting"  ]; then CMD="$CMD delete -f
$2"; else CMD="$CMD apply -f $2"; fi
}

if [ -z "$ENV" ] || [ -z "$FILEPATH" ]; then
```

```
  printf "${RED}\$ENV or \$FILEPATH has not been set!${NN}
\n"
  echo 'exiting...'
  exit 1
else
  printf "${GREEN}\$ENV${NN} has ben set to
${YELLOW}$ENV${NN}\n"
  printf "${GREEN}\$FILEPATH${NN} has ben set to
${YELLOW}$FILEPATH${NN}\n"
  if [ -n "$KUBECONFIG_PATH" ];then
    printf "${GREEN}\$KUBECONFIG_PATH${NN} has ben set to
${YELLOW}$KUBECONFIG_PATH${NN}\n";
  fi
fi

file_list() {
  find "$FILEPATH" -type f -name '*.yaml' -o -name '*.
yml' -o -name '*.json'
}

format_output() {
  while read -a line; do

  if [ "$1" == "validating" ]; then
    COLOR=$GREEN;
  else
    if echo "$line" | grep -Ei
"deploy|daemonset|statefulset" > /dev/null ; then
      echo "$line" >> ${FILEPATH}/.rollout_check.list
    fi
    COLOR=$YELLOW;
  fi

  printf "${COLOR}($1)${NN}%42s ~> %-42s${YELLOW}${line
[*]:1}${NN}\n" "$(trim_left $2 40)" "$(trim_left ${line
[0]} 40)"
  done
}

trim_left() {
  echo $1 | awk -v len=$2 '{ if (length($0) > len) print
"..." substr($1, (length($0)-(len-3))); else print; }'
}

trim_right() {
  echo $1 | awk -v len=$2 '{ if (length($0) > len) print
substr($1, 1, len-3) "..." ; else print; }'
}

execute() {
```

```
  for i in $(file_list); do
    build_command $1 $i
    eval "$CMD" 2>/tmp/applier-error | format_output $1 $i
    if [ ${PIPESTATUS[0]} -ne 0 ]; then
      printf "${RED}FAILED when $1${NN} ${YELLOW}$i${NN}
${RED}($(</tmp/applier-error)${NN}\n"
      exit 1
    fi
  done
}

case $1 in
  apply )
    echo 'applying manifest'
    execute applying
    ;;
  validate )
    echo 'validating manifest'
    execute validating
    ;;
  delete )
    echo 'deleting manifest'
    execute deleting
    ;;
  * )
    echo "please give the action command for the script"
esac
```

i. Changes required for **after-canary-deployment** stage canary-rolled-out job:

   i. The following content would be added for the **canary-rolled-out** job in the **.gitlab-ci-general.yaml.**

```
canary-rolled-out:
  extends: .releaselog-rolled-out
  stage: after-canary-deployment
  variables:
    ENV: canary
    FILEPATH: deploy/_output/$ENV/manifest
  tags:
  - shared
  - production
```

j. We also need to add a new set of the environment variable to the CI-CD pipeline for canary but these must be similar to the production setup.

Variable Name	Value
TOKEN_CANARY	Kubernetes token for the pipeline to authenticate and communicate to the kubernetes cluster, during the test we are using preproduction cluster token for gerbata taken from **TOKEN_PRODUCTION**

VAULT_ADDRESS_CANARY	Vault address to get the secret, as we are testing canary in preproduction we use the value from **VAULT_ADDRESS_PRODUCTION**
VAULT_PASSWORD_CANARY	Taken from **VAULT_PASSWORD_CANARY**
CONSUL_URL_CANARY	It is added to quick fix the kubectl binary inside the gudang/deployment:3.0.0 because it expects a consul server to be running during boot (it doesn't throw an error but add some cryptic message). We will fix it in the gudang/deployment image later.

k. PFB the changes needed in the application folder. We have done the POC on **gerbata** application. PFB the changes needed in the application folder in minera:

i. PFB the changes needed in **minerva/services/gerbata/config/kubelize** file:

```
- name: canary
  replica: "1"
```

ii. Also in the **minerva/services/gerbata/config/kubelize** file we need to increase the value of **quotaRequestsCpu**, **quotaLimitsCpu, quotaRequestsMem** and **quotaLimitsMem** attributes in the production environment by the resources required by **one pod** which would be deployed in the canary deployment in the **production namespace**.
eg:
Suppose, one pod needs 1 CPU and 500 MB then we need to increase the value of **quotaRequestsCpu** and **quotaLimitsCpu** by 1 and the value of **quotaRequestsMem** and **quotaLimitsMem** by 0.5 as we would be needing these resources for the deployment of the canary pod.

iii. In **minerva/services/gerbata/config/kubeconfig.yaml** file we need to add a new cluster with the name **canary** and add the cluster details as that of the production cluster. Not added the code due to security concerns.

iv. We are able to get the k8's metrics for the canary deployment by changing the env to deployment as mentioned in point **(l)** below, but we need to get have some custom metrics pushed so that we are able to get the metric for the traffic distribution to the canary deployment.

1. In order to get this, we need to provide an additional environmental variable named **DD_APP_ENV** in the **minerva/services/gerbata/manifest/deployment.yaml.** PFB the changes needed:

```
        env:
        - name: DD_AGENT_HOST
          valueFrom:
            fieldRef:
              fieldPath: status.hostIP
        - name: DD_APP_ENV
          value: "{{.environment.name}}"
```

2. On the application side, for this example Gerbata is using Ruby on Rails and following this How to Send Custom Metrics to DogStatsD for Kubernetes for setting up the dd client at the application. To add custom metrics for canary environment you can add this changes on connection.rb

```
def tags_hash(hash)
  arr = hash.map{ |k,v| "#{k}:#{v}" }
```

```
  arr << "deployment_env:#{ENV["DD_APP_ENV"]}"
end
```

v. There are certain issues with the configmap creation and its usage in the application deployment for the canary.

1. **Issue 1:** The name for the configmap is created using the file name of the **minerva/services/gerbata/config /preproduction/gerbata-preproduction-configmap** file. As we are trying to use the same file for creating the configmaps in canary we need to change this logic in the **minerva/shared/bin/configuration-renderer.sh.**

   For the POC purpose, we made some changes in the **configuration-renderer.sh,** file to tackle this. PFB the changes added:

```
  config_name="$(sed -e 's#[^[:alnum:]]#-#g' <<<
${file} | tr '[:upper:]' '[:lower:]')-${ENV}"
```

   So here we basically changed the logic for name created by adding the $ENV at its end. Now to tackle this change we also need to change the configname in the deployment file. PFB the change in the configmap name in the deployment.yaml

```
        envFrom:
        - configMapRef:
            name: gerbata-preproduction-configmap-{{.
environment.name}}
            optional: false
```

   Solution: We need to change the configmap naming convention to something like: {{.config.projectName}}-{{.service.name}}-{{.environment.name}}-config

   This change must also be added to the configmap name in the deployment file.

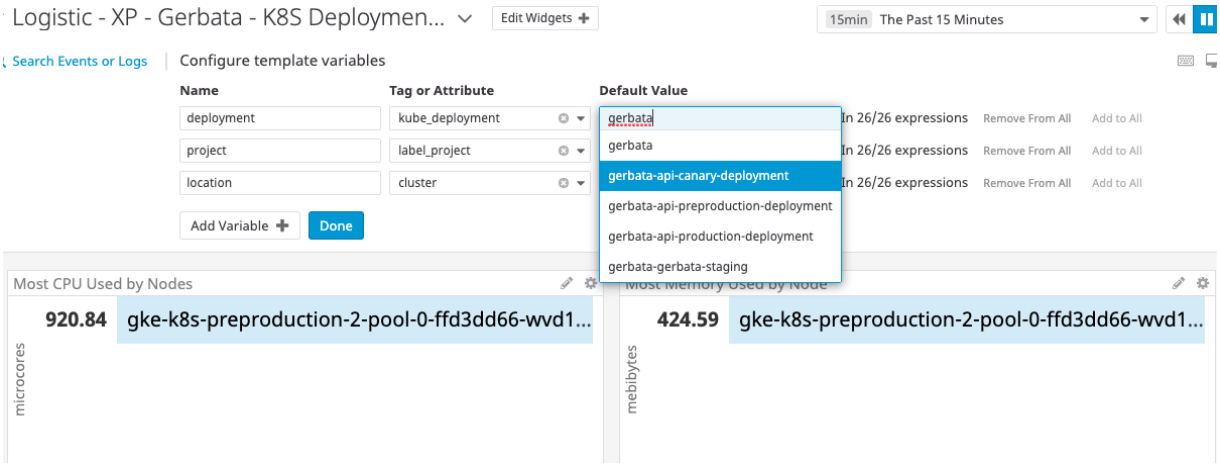
2. We need to have live (production) traffic on the canary pod, but as we would be using the same manifest files for the canary and production deployment we would now have pods with labels env=production and env=canary. But the service in production has selectors with labels as follows:

   Therefore the canary pods won't be serving the production traffic as they are not part of the production service.

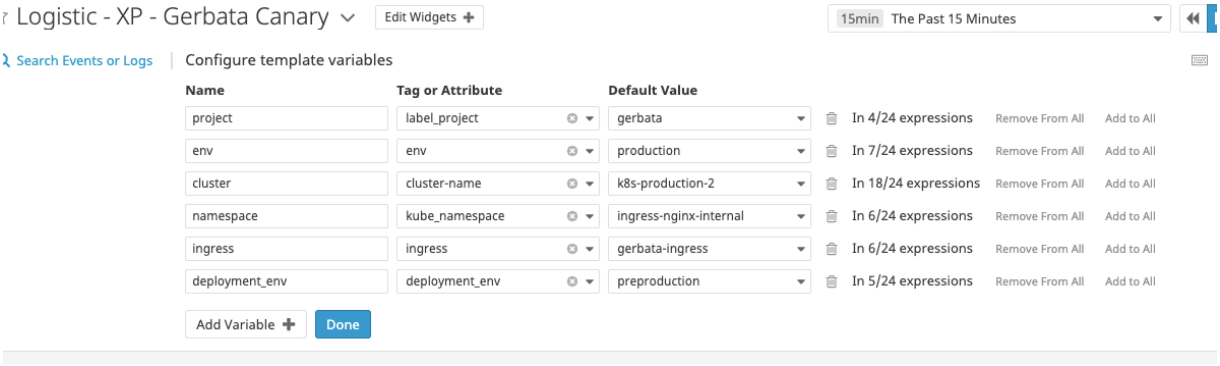
   Solution: We need to remove the **env: {{.environment.name}}** selector label from the service manifest to resolve this.

```
app: {{.service.name}}
project: {{.config.projectName}}
service: {{.service.name}}
type: deployment
env: {{.environment.name}}
```

l. For monitoring the kubernetes canary deployment we need to change the parameters in the Datadog Dashboard. Instead of **environment** variable in the variable template, we need to change it to **deployment** with the attribute: **kube_deployment.** Then we can switch the **deployment** variable to observe the pods of respective deployments. The following datadog dashboard can be used for reference: Gerbata Dashboard.

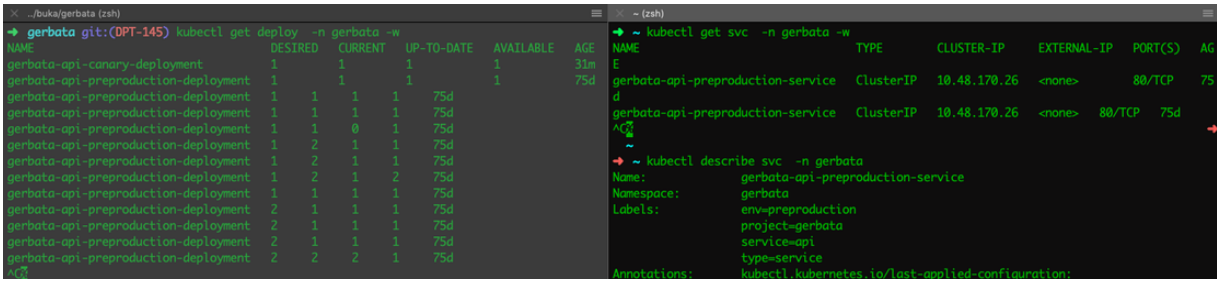


m. For monitoring live traffic to the canary pods, we need to add an additional variable to the existing DD dashboard(specifically the application dashboard where we get the ingress details). Add a new variable named **deployment_version** with the attribute: **deployment_env.** Once done then the traffic towards the canary deployment can also be observed. PFB the screenshot for the same. The change are done for Gerbata Dashboard.



5. We had certain concerns over the following points:
   a. k8's Service IP address changes

      So the concern was over the point when we reapply the service after removing the **env** tag from the label selector. But this scenario was test and the IP address fo the service remained the same as we just apply the service without recreating it. PFB the screenshot for the same.



   b. Traffic distribution on the pods of both the deployments(Canary and Pre-prod). When we apply the service without the **env** tag in label selector, both the pods can be observed behind the service when we describe it. PFB the screenshot for the same

```
➜ gerbata git:(DPT-145) kubectl get pods -n gerbata -o wide
NAME                                                READY   STATUS    RESTARTS   AGE
  IP              NODE                                                NOMINATED NODE
gerbata-api-canary-deployment-864db9bffd-vv7bv      1/1     Running   0          27m
  10.74.0.82    gke-k8s-preproduction-2-pool-0-b62ee906-0411   <none>
gerbata-api-preproduction-deployment-d4b75645f-8zlxj   1/1     Running   0          10m
  10.74.4.252   gke-k8s-preproduction-2-pool-0-de70581e-ks74   <none>
➜ gerbata git:(DPT-145)
```

```
➜ ~ kubectl describe svc  -n gerbata
Name:              gerbata-api-preproduction-service
Namespace:         gerbata
Labels:            env=preproduction
                   project=gerbata
                   service=api
                   type=service
Annotations:       kubectl.kubernetes.io/last-applied-configuration:
                     {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"la
bels":{"env":"preproduction","project":"gerbata","service":"api","type...
Selector:          app=api,project=gerbata,service=api,type=deployment
Type:              ClusterIP
IP:                10.48.170.26
Port:              <unset>  80/TCP
TargetPort:        8869/TCP
Endpoints:         10.74.0.82:8869,10.74.4.252:8869
Session Affinity:  None
Events:            <none>
```

Also during the load test, it was observed that the CPU of both the pods increased during the load test and then it reduced to minimal once the load test was completed. Also, no errors were observed during the load test.