Cloud Computing – CLC
Canary deployment with Flagger

created at the university of applied sciences Upper Austria

Data Science and Engineering

FH OÖ, Site Hagenberg

Group project

Submitted by

Martin Hanreich, Cristina Mafra de Sa und Sarah Prandner

1 Contents

2 Introduction

This document aims to provide an easy-to-follow tutorial for the canary deployment strategy assisted by the Kubernetes operator Flagger. Focus is on the encountered obstacles and remaining questions when attempting to work with flagger while following the official Flagger docs. In order to create this guide, a sample application is deployed using Flagger. Furthermore, three more aspects when using flagger are shown: a demo of an automated rollback, monitoring of the rollout and alerting in case of events. In combination with Flagger NGINX is used for traffic routing while the analysis and monitoring is done via Prometheus and Microsoft Teams is used for receiving notifications and alerts regarding the deployment status.

2.1 Tools

Nachfolgend soll ein Überblick über einige der verwendeten Tools gegeben warden. Flagger unterstützt Alternativen für jedes dieser Tools. Hier wird allerdings

2.1.1 Flagger

Flagger is a progressive delivery tool for Kubernetes. It helps with the deployment of an application by providing services, which facilitate the rollout process and its automation. Specifically, Flagger supports the Canary Release, Blue/Green and Blue/Green Mirroring deployment strategies as well as A/B-Testing. Moreover, Flagger works in conjunction with other tools to enable for example effective monitoring and alerting. These include Service Meshes like Istio or Linkerd, Ingress-Controllers like Skipper and monitoring via DataDog.

2.1.2 Helm

Helm is a tool used to manage Kubernetes applications that simplify their deployment through a collection of files, known as Charts. The Charts describe a related set of Kubernetes resources and assist in defining, installing and updating applications regardless of their complexity.

2.1.3 NGINX

More specifically the NGINX Ingress Controller is used. It is responsible for traffic routing and therefore handling external requests from the users.

2.1.4 Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit. Prometheus collects and stores its metrics as time series data, i.e. metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels. Prometheus scrapes metrics from instrumented jobs, either directly or via an intermediary push gateway for short-lived jobs. It stores all scraped samples locally and runs rules over this data to either aggregate and record new time series from existing data or generate alerts.

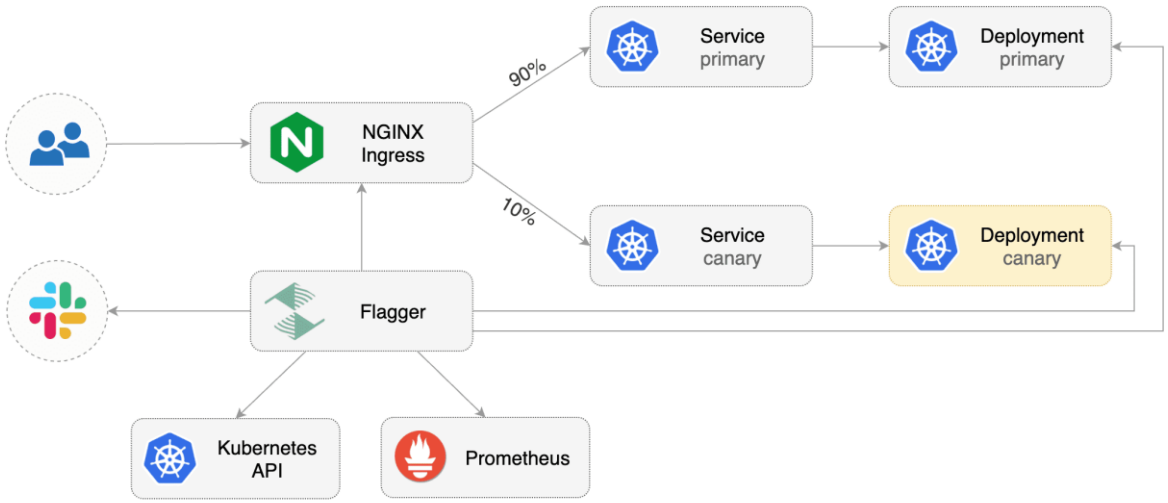
2.1.5 Microsoft Teams

With the help of Prometheus alerting through AlertManager we will be able to send alert messages via Teams. For that we have to configure incoming Webhooks in Teams. This will then generate a webhook URL which will be used to post messages to our channel. To send the third-party tool for Microsoft Teams, an alertmanager.yaml file will be needed. So that our AlertManager is ready for sending alerts using webhook, a Secret object - which is used by Prometheus operator's AlertManager and then applied to Kubernetes - must be created. Last but not least a default Microsoft Teams Message card template has to be defined and a Kubernetes manifest with Helm passing "Incoming Webhook URL" of Microsoft Teams must be generated. This will be finally deployed to Kubernetes and makes it possible to receive alerts in our Microsoft Teams Channel.

A deep look into Flagger and the tools: [Flagger]

2.2 Architecture

Following diagram shows how the previous described components work together for the canary rollout:



The ingress controller NGINX handles the traffic coming from the users and to the current (primary) version or the updated (canary) version. Flagger is the controller who orchestrates the deployment process. The monitoring tools Prometheus that continuously collects data about the current deployment assists Flagger.

2.3 Canary Deployment

Canary deployment is one of the strategies used to deploy changes or release new versions of applications. With this approach, the deployment happens gradually, which means, the changes or the second version of the application is deployed alongside the first version and a load balancer is used to control the traffic of users between versions. At first, the release will affect only a subset of users, which allows the new version to be tested with real data and reduces failure during deployment, also facilitating a faster rollback in case of problems. Once the deployment is working well, the subset of users is increased gradually until all the traffic is completely on the new version.

The figure below shows an example of canary deployment in six steps:

1. The first step shows the infrastructure at start, in which the application's version 1 is running.
2. Then a new instance starts running with the new version and 5% of user traffic is shifted to it, while 95% remains accessing the version 1. At this point it is possible to start the evaluation of the version's performance, collect logs and monitor possible errors.
3. In step 3 the traffic of users is increased gradually to two instances of the application with version 2 running alongside of version 1. At this point is possible to define the acceptance of the version and decide if it will be released completely or if it will be necessary to roll back to version 1.
4. In Step 4, 50% of users are using the new version and one instance with version 1 is upgraded.
5. Step five shows all instances running version 2 and the traffic is shifted back to the previous infrastructure at the step 1, but now with the deployment of the new version complete.
6. Finally, after successfully completing the deployment of version 2 successfully, the new instances are deleted.

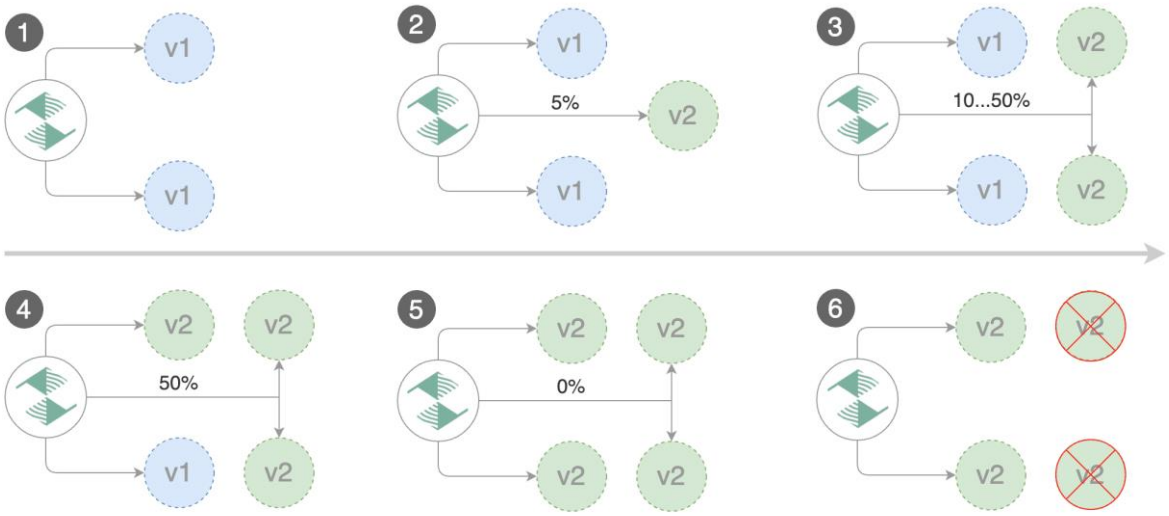


Figure 1 - Steps canary deployment. Image from https://docs.flagger.app/tutorials/istio-progressive-delivery

3 Tutorial-Get Canary Development running quickly

3.1 Setting up

3.1.1 Requirements

A Kubernetes cluster v.1.16 or newer is required to install Flagger. The Kubernetes client needs to be installed also, so it can be used within the command line. If it is installed you can enter 'kubectl version' in the terminal without an error occurring.

3.1.2 Installation

Before the setup for the canary deployment can begin, a few tools need to be installed first. At the very start the package manager Helm is required since it is used to get all the other tools.

3.1.3 Helm

There are different methods to install Helm on Kubernetes. The Binary method is the most simple one. After downloading the desired version of Helm from the link https://github.com/helm/helm/releases and it is unzipped, move the executable file to the bin directory and validate it by executing the command below. If you are using Windows, it will be easier to use the command *helm* setting the PATH in the System variables. Enter following command to verify that Helm can be used within the command line.

- *helm version*

3.1.4 NGINX

To install Nginx, first create a namespace in Kubernetes called ingress-nginx using the command below:

- *kubectl create ns ingress-nginx*

Then use Helm to install the Nginx Ingress controller:

- *helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx*
- *helm upgrade -i ingress-nginx ingress-nginx/ingress-nginx --namespace ingress-nginx --set controller.metrics.enabled=true --set controller.podAnnotations."prometheus\.io/scrape"=true --set controller.podAnnotations."prometheus\.io/port"=10254*

3.1.5 Flagger and Prometheus

Use the command below to Install Flagger and Prometheus in the same namespace as the ingress controller:

- *helm repo add flagger https://flagger.app*

- *helm upgrade -i flagger flagger/flagger --namespace ingress-nginx --set prometheus.install=true --set meshProvider=nginx*

After installing these tools go to the website of your kubernetes cluster provider like Google Cloud Platform or Microsoft Azure and check if all the pods are created without errors.

3.2 Preparing Deployment

After all necessary tools are installed; the preparation for the deployment can begin. First, get a local copy of the github repo and navigate with the command line to the files directory, which is a folder contained in the repo.

In this folder, execute the command:

- *Podinfo\kubectl apply –k .*

With this command, a deployment file and a horizontal pod autoscaler, which are in the podinfo folder, are added with the help of *Kustomize*.

In the *files* folder are the two more important files for setting up the deployment: *podinfo-canary.yaml* and *podinfo-ingress.yaml.* They specifiy different aspects of the canary deployment itself and the external access via Ingress.

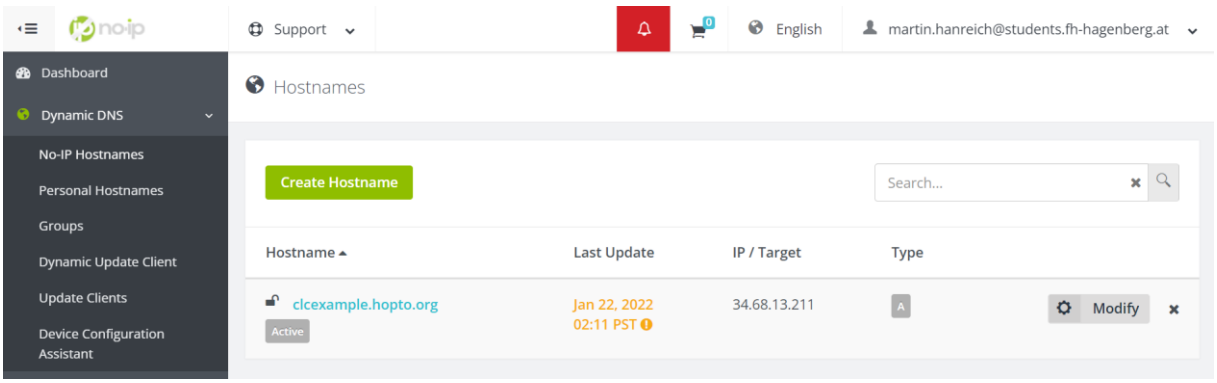
Open these files in a text editor because the host needs to be specified.

```
metrics:
- name: request-success-rate
  # minimum req success rate (non 5xx responses)
  # percentage (0-100)
  thresholdRange:
    min: 99
  interval: 1m
# testing (optional)
webhooks:
  - name: acceptance-test
    type: pre-rollout
    url: http://flagger-loadtester.test/
    timeout: 30s
    metadata:
      type: bash
      cmd: "curl -sd 'test' http://podinfo-canary/token | grep token"
  - name: load-test
    url: http://flagger-loadtester.test/
    timeout: 5s
    metadata:
      cmd: "hey -z 1m -q 10 -c 2 http://clcexample.hopto.org/"
```

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: podinfo
  namespace: test
  labels:
    app: podinfo
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
    - host: "clcexample.hopto.org"
      http:
        paths:
          - pathType: Prefix
            path: "/"
            backend:
              service:
                name: podinfo
                port:
                  number: 80
```

The domains in the red rectangles need to be replaced with one's own domain. A domain can be bought for example from Google.

For testing purposes, there is a free alternative. When registering on noip.com, a custom domain name can be assigned to an IP address.



The IP address needs to be the address of the NGINX Load Balancer. To get this address enter the following command in the command line.

➢ *kubectl get service ingress-nginx-controller –namespace=ingress-nginx*

```
C:\Users\P41914\Desktop\Docker\CLC\Flagger>kubectl get service ingress-nginx-controller --namespace=ingress-nginx
NAME                       TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)                      AGE
ingress-nginx-controller   LoadBalancer   10.48.0.180   34.68.13.211   80:32635/TCP,443:32219/TCP   15d
```

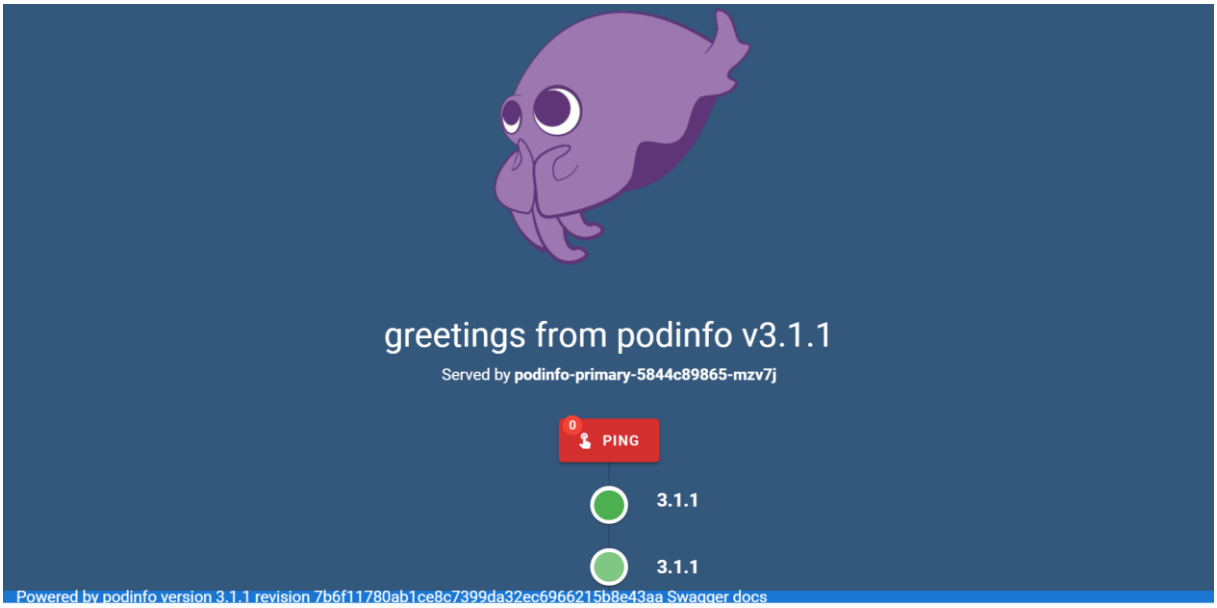
Enter this address as your target for your domain at the *noip* website.

After that, add your domain name to the *podinfo-canary.yaml* and *podinfo-ingress-yaml* files.

Now the files need to be applied. For that make sure you are in the files folder and enter following commands.

- *kubectl apply –f ./podinfo-ingress.yaml*
- *kubectl apply –f ./podinfo-canary.yaml*

Now you can enter the name of your domain in an internet browser and see the following image.



3.3 Executing the canary deployment

After the preparation steps the actual deployment can start. For that, a sample docker image provided by flagger is used. It can later be replaced by a custom image.

To trigger the canary deployment enter following command.

- *kubectl –n test set image deployment/podinfo podinfod=stefanprodan/podinfo:3.1.1*

This command sets the image specified in deployment to another version. Flagger detects this change and automatically begins with the canary deployment.

```
C:\Users\P41914\Desktop\Docker\CLC\Flagger>kubectl -n test set image deployment/podinfo podinfod=stefanprodan/podinfo:3.1.1
deployment.apps/podinfo image updated
```

Note: If the message ‘image update’ does not appear it probably indicates that the version is already the primary version. Therefore, specifiy another image version for example podinfo:3.1.2.

You can monitor the progress with following command.

- *Kubectl –n test get canaries*

By adding –watch or under Windows using a while loop the progress can continuously be monitored.

4 Exploring possibilities with Flagger

4.1 Automatic rollback

One of the main requirements for a safe deployment is the quick rollback in case problems occur. The configuration for that is done in the podinfo-canary-yaml as well as for other variables, which can be adjusted there.

```
analysis:
  # schedule interval (default 60s)
  interval: 10s
  # max number of failed metric checks before rollback
  threshold: 10
  # max traffic percentage routed to canary
  # percentage (0-100)
  maxWeight: 50
  # canary increment step
  # percentage (0-100)
  stepWeight: 5
  # NGINX Prometheus checks
  metrics:
  - name: request-success-rate
    # minimum req success rate (non 5xx responses)
    # percentage (0-100)
    thresholdRange:
      min: 99
    interval: 1m
  # testing (optional)
  webhooks:
    - name: acceptance-test
      type: pre-rollout
      url: http://flagger-loadtester.test/
      timeout: 30s
      metadata:
        type: bash
        cmd: "curl -sd 'test' http://podinfo-canary/token | grep token"
    - name: load-test
      url: http://flagger-loadtester.test/
      timeout: 5s
      metadata:
        cmd: "hey -z 1m -q 10 -c 2 http://clcexample.hopto.org/"
```

By default, the metric being monitored to check whether to rollback is the *request-success-rate*.

However, there is the possibility to add further metrics via Prometheus.

For that, a file has to be created which specifies the desired metric. For the specification Prometheus own query language called *PromQL* is used.

However, there are many predefined metrics for Prometheus available, which one can find online.

Take for example the following metric *latency*, which can be used to check for unwanted delay.

```yaml
apiVersion: flagger.app/v1beta1
kind: MetricTemplate
metadata:
  name: latency
  namespace: test
spec:
  provider:
    type: prometheus
    address: http://flagger-prometheus.ingress-nginx:9090
  query: |
    histogram_quantile(0.99,
      sum(
        rate(
          http_request_duration_seconds_bucket{
            kubernetes_namespace="{{ namespace }}",
            kubernetes_pod_name=~"{{ target }}-[0-9a-zA-Z]+(-[0-9a-zA-Z]+)"
          }[1m]
        )
      ) by (le)
    )
```

After the creation of the metric files they need to be added to kubernetes with the *apply* –f command.

To actually use the defined metrics they need to be referenced inside the podinfo-ingress.yaml file.

```yaml
    maxWeight: 50
    # canary increment step
    # percentage (0-100)
    stepWeight: 5
    # NGINX Prometheus checks
    metrics:
    - name: request-success-rate
      # minimum req success rate (non 5xx responses)
      # percentage (0-100)
      thresholdRange:
        min: 99
      interval: 1m
    - name: "latency"
      templateRef:
        name: latency
      thresholdRange:
        max: 0.5
      interval: 1m
```

There another entry in the *metrics* section is created where the *name* refers to the name given to the metric in the previous created metric yaml file.

After having added all the desired metrics in the *podinfo-ingress.yaml* file the changes are applied also via the *kubectl apply* command.

When restarting the deployment again these new metrics are contributing to failure counters which determine if a rollback is performed.

To actually test if the rollback is working the podinfo images which is being used here can produce and error or delay on command.

For that, enter in the command line following command.

- *curl <YOUR_HOSTNAME>/status/500 --watch*

This command continuously triggers errors, which eventually leads to the rollback when starting a new deployment.

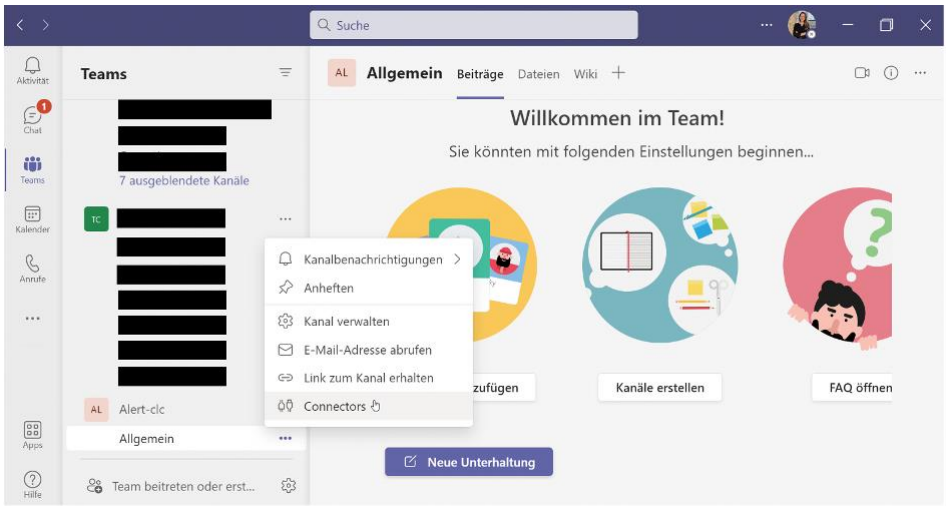
With a slight adaption of this command, a delay is triggered.

- *curl <YOUR_HOSTNAME>/delay/2 --watch*

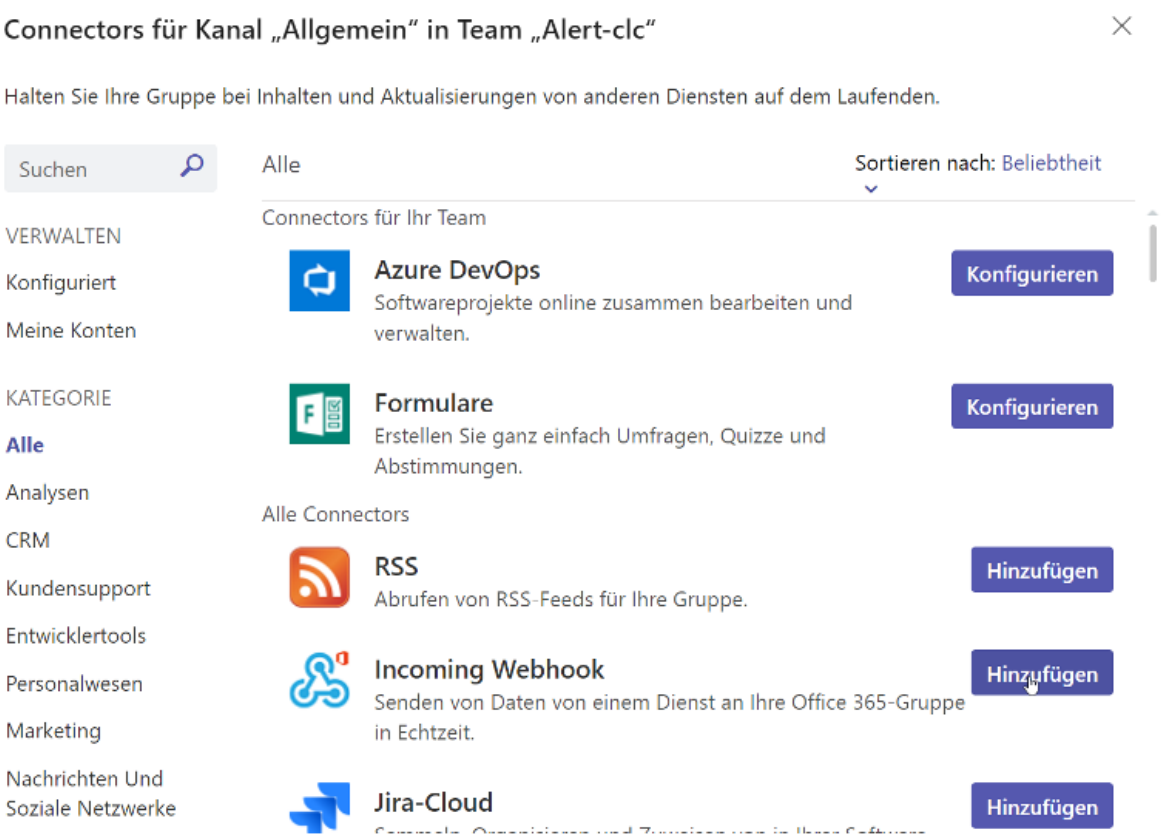
If only this command is executed and the deployment is not finished then this means the additional *latency* metric works.

4.2 Microsoft Teams Alert

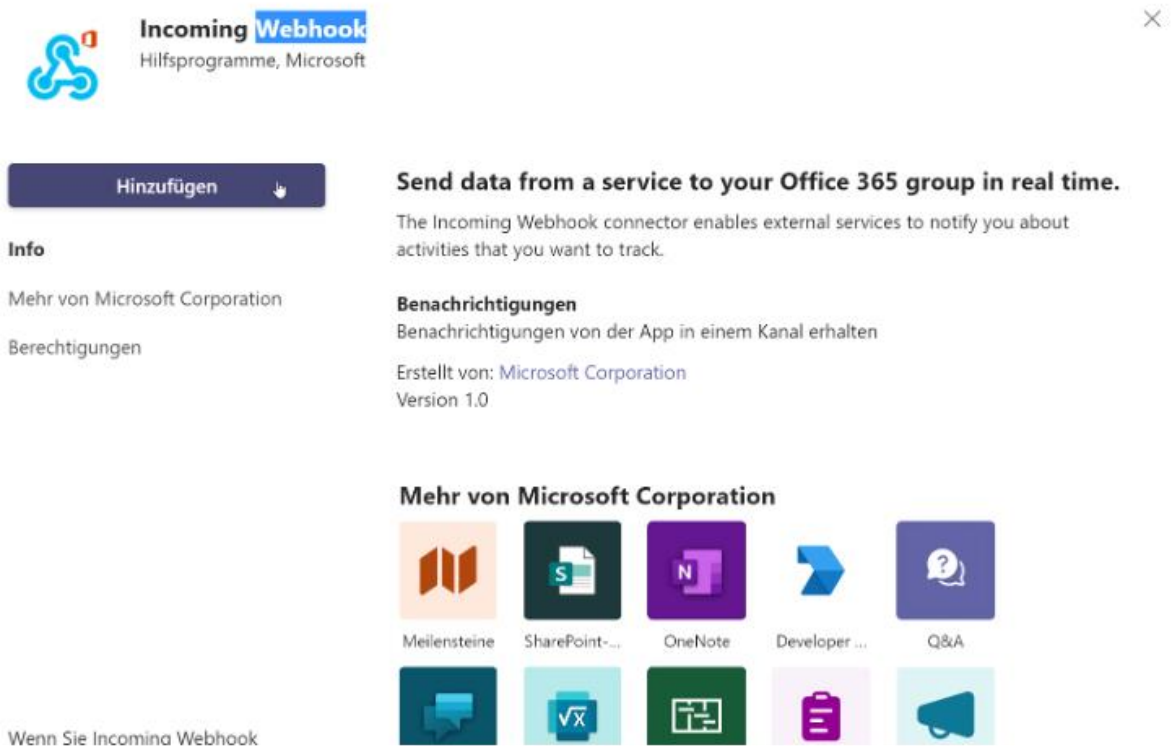
Go to the channel, where you want to receive alerts. Click on the ... on the right side of the channel name and select Connectors from the dropdown list.



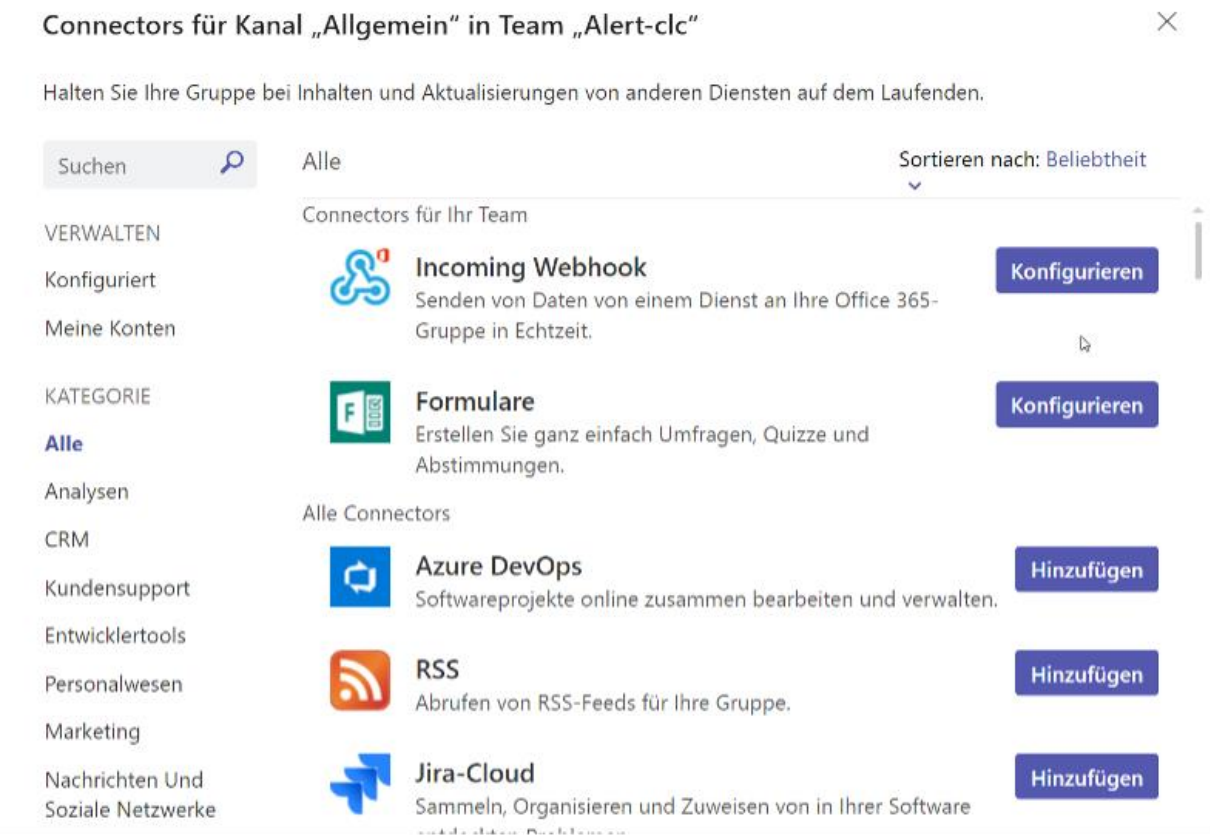
A window will then open. Add "Incoming Webhook".



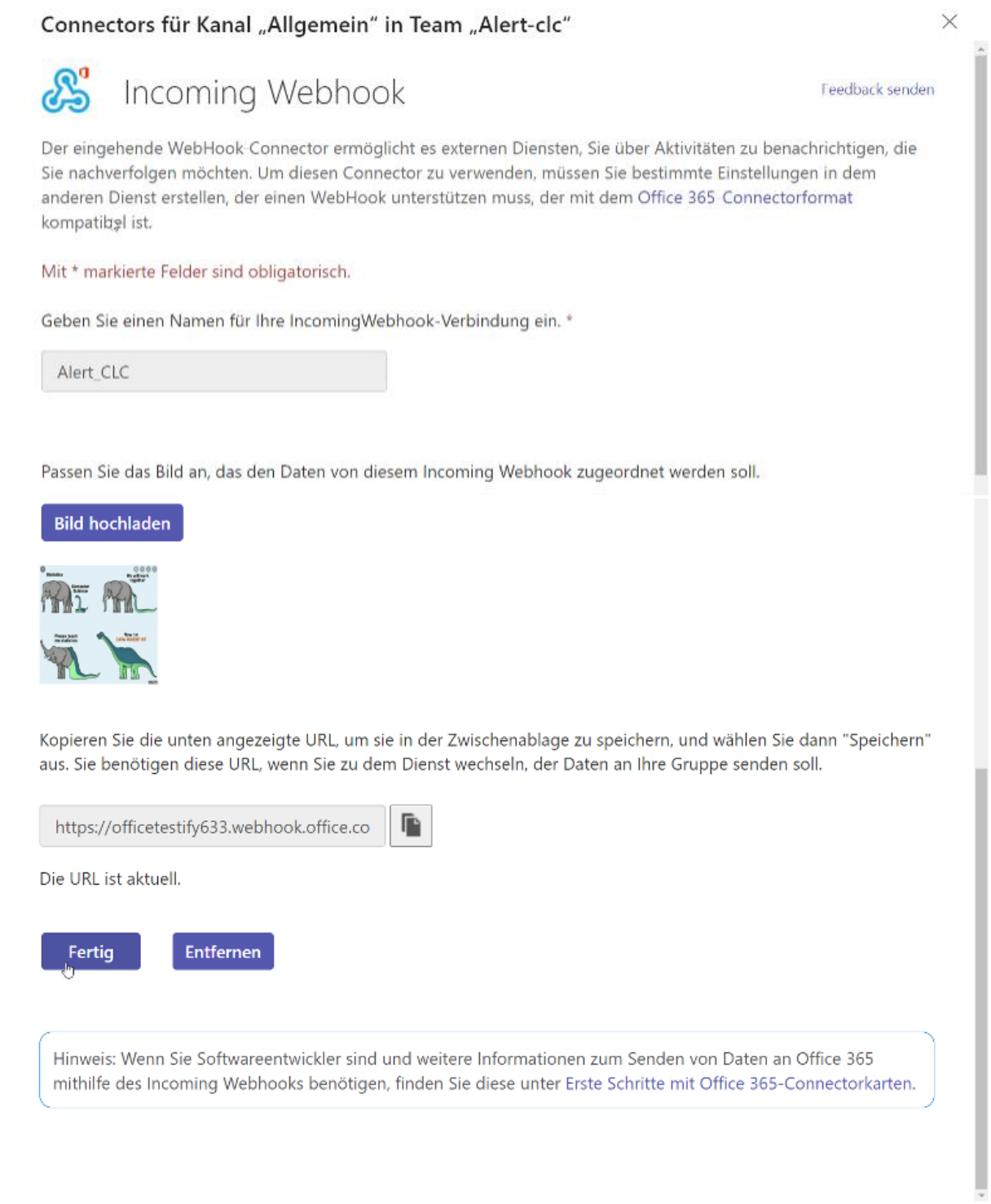
Now the connector "Incoming Webhook" opens where you have to select "Add" one more time.



To configure the connector now you have to select the pre-selected channel again by Clicking on the ... on the right side of Channel name via the connector. Again select Connectors from the dropdown list. Next click on the "Configure" button next to “Incoming Webhook”.



After that, you need to specify a name as well as an image for the IncomingWebhook connection. When you have completed these steps, a URL will be created. Copy it to the clipboard and click on "Done". You will need this URL when you switch to the service that will send data to your groups.



Configure Alertmanager via Flagger

Flagger can be now configured to send notifications to Microsoft Teams with the following command line:

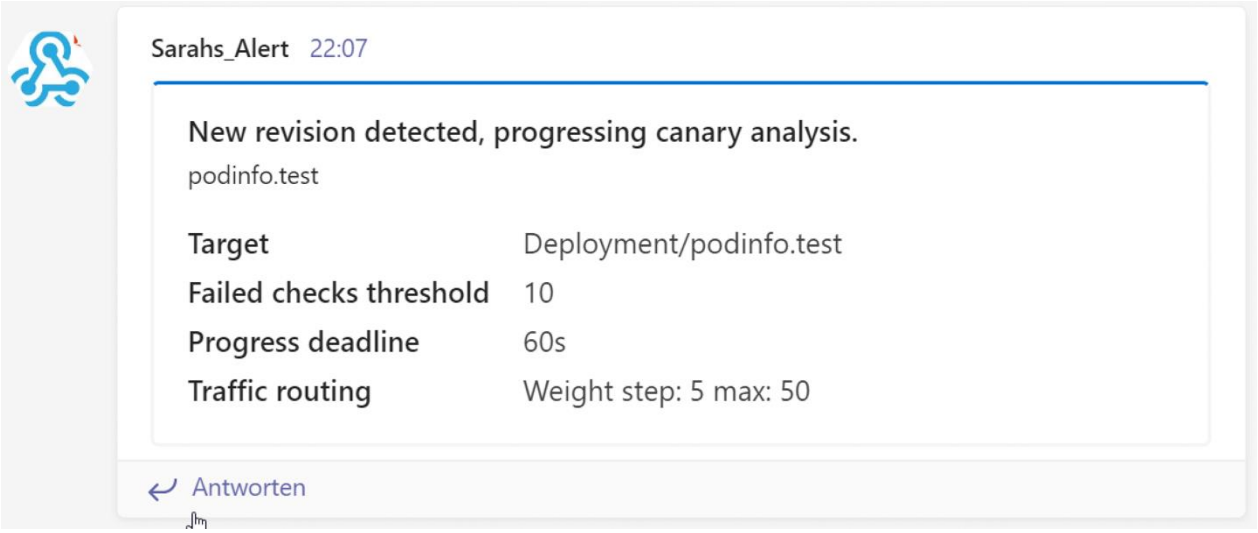
- *helm upgrade -i flagger flagger/flagger \ --set* msteams.url=https://outlook.office.com/webhook/YOUR/TEAMS/WEBHOOK \ --set msteams.proxy.url=my-http-proxy.com # optional http/s proxy

Therefore you have to insert the generated URL from MS Teams and optionally add the proxy URL as followed:

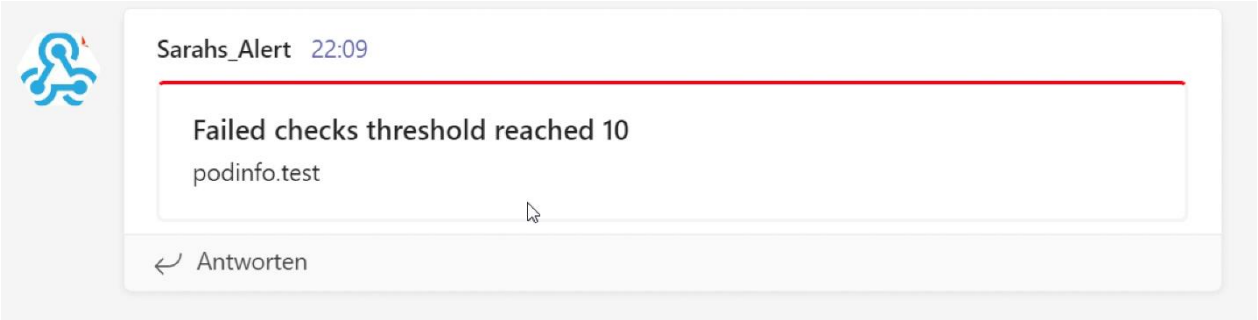
- *helm upgrade -i flagger flagger/flagger \ --set* msteams.url=https://officetestify633.webhook.office.com/webhookb2/63a11652-d1bd-4447-ae9d-927dee285a25@eb8e7f5a-975d-46b9-af52-d98412275d2b/IncomingWebhook/74ca1e25f2714b25a24cd4ac54d848f8/0db59b53-d85f-46d5-8595-28388336a0bd \ --set msteams.proxy.url= sarah.onthewifi.com

Check MS Teams Notification

If we run the Canary deployment now it shows the following information in MS-Teams.



If the deployment fails, the following error message appears.



Problems:

Unfortunately, the original command as documented did not work for me. For this reason, an adjustment was necessary as followed:

- *helm upgrade -n ingress-nginx -i flagger flagger/flagger* \ --set msteams.url=https://officetestify633.webhook.office.com/webhookb2/63a11652-d1bd-4447-ae9d-927dee285a25@eb8e7f5a-975d-46b9-af52-d98412275d2b/IncomingWebhook/74ca1e25f2714b25a24cd4ac54d848f8/0db59b53-d85f-46d5-8595-28388336a0bd \ --set msteams.proxy.url= sarah.onthewifi.com

4.3 Prometheus

To install Prometheus, download through the link https://prometheus.io/download/ and run the command below:

- *tar xvfz prometheus-*.tar.gz*
- *cd prometheus-**

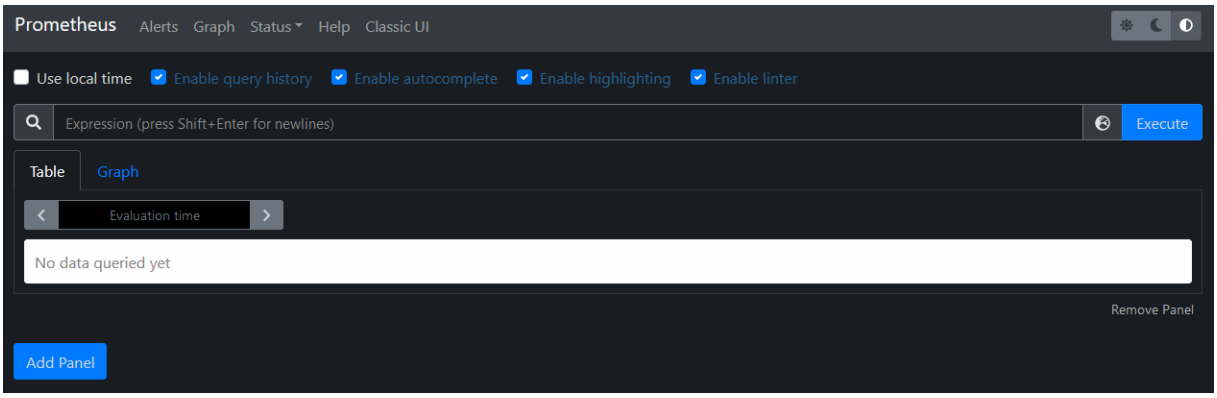
Because Prometheus collects metrics from targets, the parameters in filed prometheus.yaml should contain the information about which targets it is going to collect metrics. Below is an example of the prometheus.yaml file showed at the website https://prometheus.io/docs/prometheus/latest/getting_started/, in which Prometheus is set up to monitor itself.

```
global:
  scrape_interval:     15s # By default, scrape targets every 15 seconds.

  # Attach these labels to any time series or alerts when communicating
with
  # external systems (federation, remote storage, Alertmanager).
  external_labels:
    monitor: 'codelab-monitor'

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries
scraped from this config.
  - job_name: 'prometheus'

    # Override the global default and scrape targets from this job every 5
seconds.
    scrape_interval: 5s

    static_configs:
      - targets: ['localhost:9090']
```

The command below starts Prometheus and forward it to localhost:9090.

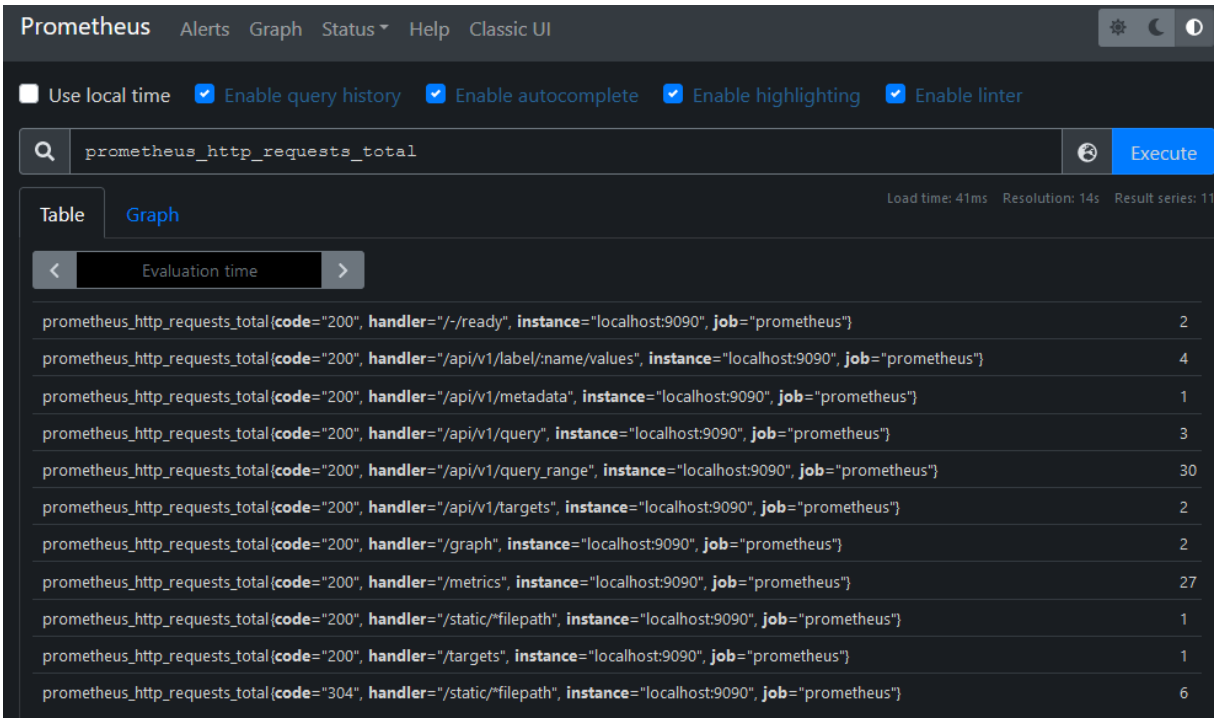
- *./prometheus --config.file=./prometheus.yml*

To access Prometheus, use the link http://localhost:9090. The image below shows the first page.



Once started, you will have to execute what metrics should being seen at the Panel. It is possible to see this metrics in table or graphic format. One metric that Prometheus collected from itself is named *prometheus_target_interval_length_seconds*, which shows the amount of time between target requests to http://localhost:9090. The graphic below shows this metrics divided by different latency percentiles.

Using the expression browser from Prometheus, the image below shows the number of requests to http://localhost:9090.


Prometheus
Alerts
Graph
Status
Help
Classic UI
Use local time
Enable query history
Enable autocomplete
Enable highlighting
Enable linter
prometheus_http_requests_total
Execute
Table
Graph
Load time: 41ms
Resolution: 14s
Result series: 11
Evaluation time
prometheus_http_requests_total{code="200", handler="/-/ready", instance="localhost:9090", job="prometheus"} 2
prometheus_http_requests_total{code="200", handler="/api/v1/label/:name/values", instance="localhost:9090", job="prometheus"} 4
prometheus_http_requests_total{code="200", handler="/api/v1/metadata", instance="localhost:9090", job="prometheus"} 1
prometheus_http_requests_total{code="200", handler="/api/v1/query", instance="localhost:9090", job="prometheus"} 3
prometheus_http_requests_total{code="200", handler="/api/v1/query_range", instance="localhost:9090", job="prometheus"} 30
prometheus_http_requests_total{code="200", handler="/api/v1/targets", instance="localhost:9090", job="prometheus"} 2
prometheus_http_requests_total{code="200", handler="/graph", instance="localhost:9090", job="prometheus"} 2
prometheus_http_requests_total{code="200", handler="/metrics", instance="localhost:9090", job="prometheus"} 27
prometheus_http_requests_total{code="200", handler="/static/*filepath", instance="localhost:9090", job="prometheus"} 1
prometheus_http_requests_total{code="200", handler="/targets", instance="localhost:9090", job="prometheus"} 1
prometheus_http_requests_total{code="304", handler="/static/*filepath", instance="localhost:9090", job="prometheus"} 6

5 Common errors and obstacles

The previous described steps already tried to eliminate many sources of errors. Nevertheless, there are of course quite a few problems left.

Following closing section should therefore give clues to the solutions of some the most often occurring obstacles.

Canary deployment does not progress after the 5-weight milestone with the message '*Halt advancement no values found for nginx metric request-success-rate probably podinfo.test is not receiving traffic: running query failed: no values found*'

This error occurs often and can have many root causes. It occurs because due to some problem in the configuration the LoadTester is unable to do its job.

1. Make sure the Hostname in the podinfo-canary.yaml and podinfo-ingress.yaml is correct

2. Make sure you can reach the podinfo application by entering the Hostname in your web browser. In case it works the 'Greetings from podinfo' message can be seen.

3. Make sure there are no duplicates of pods in different namespaces. For example, the flagger-loadtester should only appear once and should be in the same namespace as the flagger-prometheus and the ingress-nginx-controller pods.

Canary deployment randomly stops progressing after an arbitrary amount of time with the previous error

Maybe the cluster does not have enough resources available. During the deployment and the shifting around there is a strong temporary need for CPU and memory. Go to your Kubernetes cluster provided like Google Cloud or Microsoft Azure, look for error messages, and eventually increase for example the number of nodes.

Changes do not seem to have an effect or a very general error message appears

Every .yaml file that should be used must be added or updated via the kubectl apply –f command. Changing the file and saving is not sufficient. This step is forgotten quite often.