

Cloud Computing – CLC

Canary deployment with Flagger

created at the university of applied sciences Data Science and Engineering

FH OÖ, Site Hagenberg



Group project

Submitted by

Martin Hanreich, Cristina Mafra de Sa und Sarah Prandner

1 Contents

1	Introduction.....	3
1.1	Tools	3
1.1.1	Flagger	3
1.1.2	Helm	3
1.1.3	NGINX	3
1.1.4	Prometheus	3
1.1.5	Microsoft Teams.....	4
1.2	Architecture.....	4
2	Canary Deployment.....	5
3	Setting up	6
3.1	Requirements	6
3.2	Installation.....	6
3.2.1	Helm	6
3.2.2	NGINX	6
3.2.3	Flagger and Prometheus	6
3.3	Preparing Deployment	7
3.3.1	Executing the canary deployment	9
3.4	Automatic rollback	9
3.4.1	Microsoft Teams Alert.....	11
3.5	Prometheus.....	15

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 werden. 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](#)

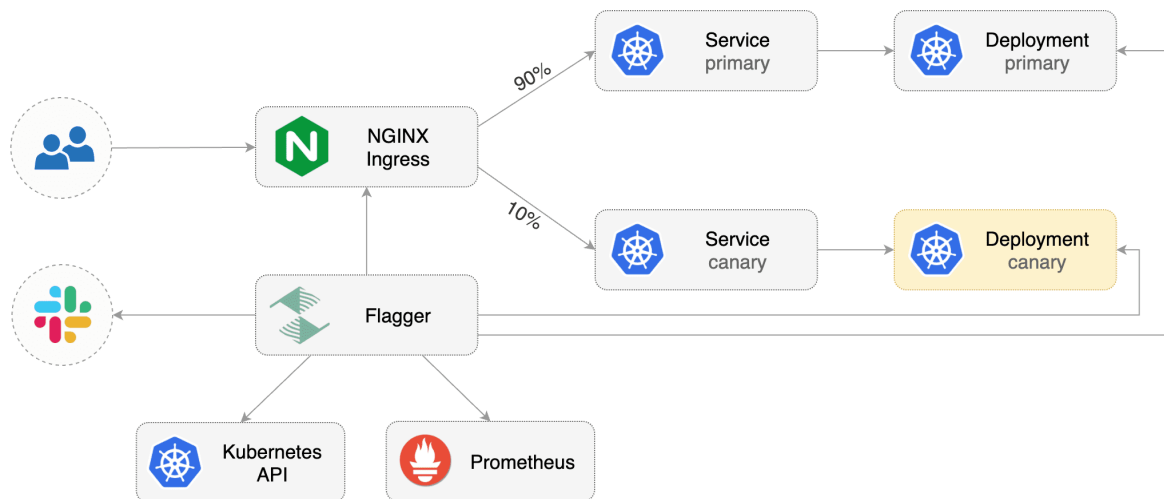
Prometheus is an open-source systems monitoring and alerting toolkit. Prometheus collects and stores its metrics as time series data, i.e. metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels. Prometheus scrapes metrics from instrumented 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](#)

Das Ziel ist es Flagger so zu konfigurieren das in Hinblick auf das Canary-Deployment automatische generierten Benachrichtigungen an MS-Teams gesendet werden. Dazu muss zuerst in Microsoft Teams eine eingehende Webhooks konfiguriert werden. Diese Webhook-URL, wird verwendet, um Nachrichten über Flagger an den gewünschten Kanal zu senden. Wenn man schlussendlich einer Canary-Deployment durch Aktualisieren des Container-Images auslöst, wird eine MS Teams Notification generiert.

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.

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.

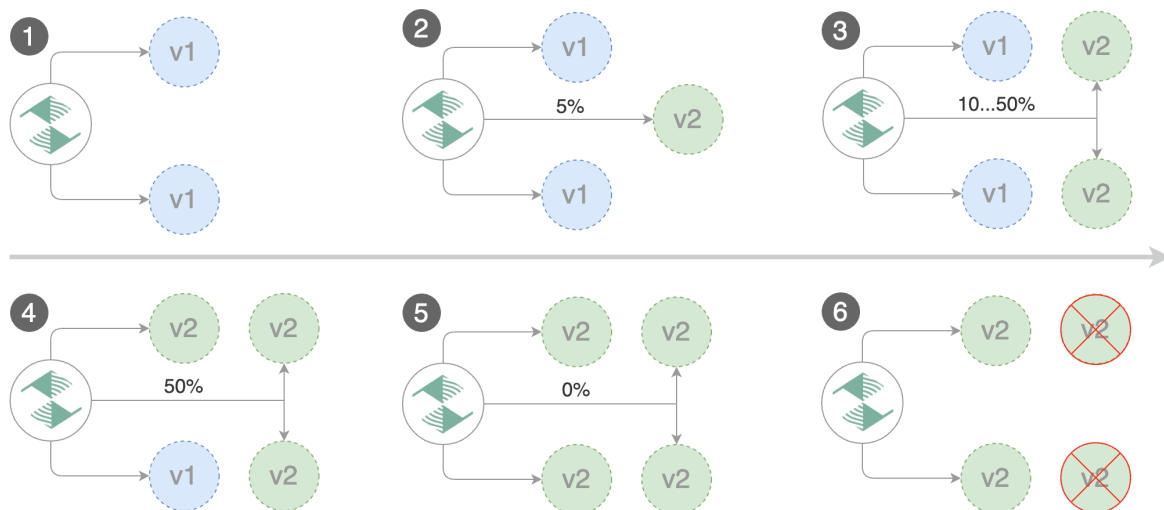


Figure 1 - Steps canary deployment. Image from <https://docs.flagger.app/tutorials/istio-progressive-delivery>

4 Setting up

4.1 Requirements

A Kubernetes cluster v.1.16 or newer is required to install Flagger. The Kubernetes client needs to be installed also

4.2 Installation

Before the setup for the canary deployment can begin a few tools need to be installed first. If it is installed you can enter `kubectl` version in the terminal without an error occurring.

4.2.1 [Helm](#)

There are different methods to install Helm on Kubernetes. The Binary method is the most simple. 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.

```
➤ helm version
```

4.2.2 [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
```

4.2.3 [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

4.3 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.

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 files *podinfo-canary.yaml* and *podinfo-ingress.yaml*. They specify different aspects of the canary deployment itself and the external access via Ingress.

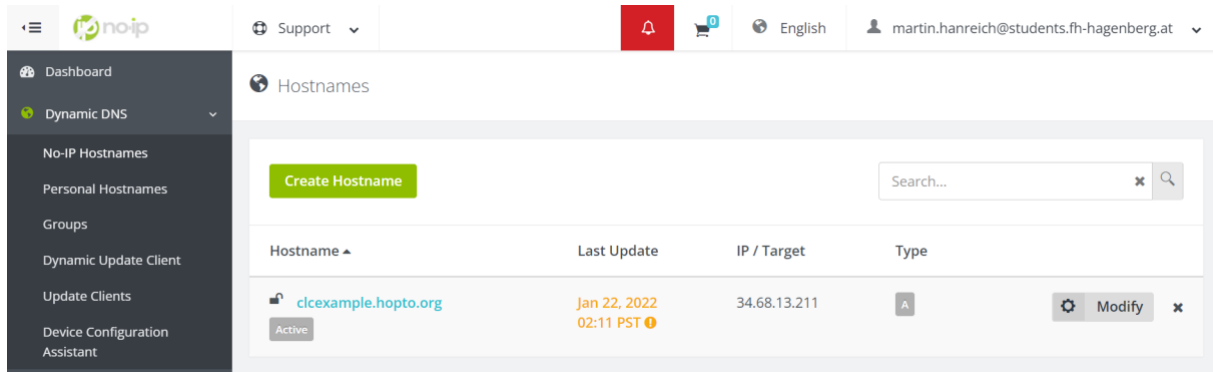
Open these files in a text editor

```
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://clcxample.hopto.org/"

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: podinfo
  namespace: test
  labels:
    app: podinfo
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - host: "clcxample.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.

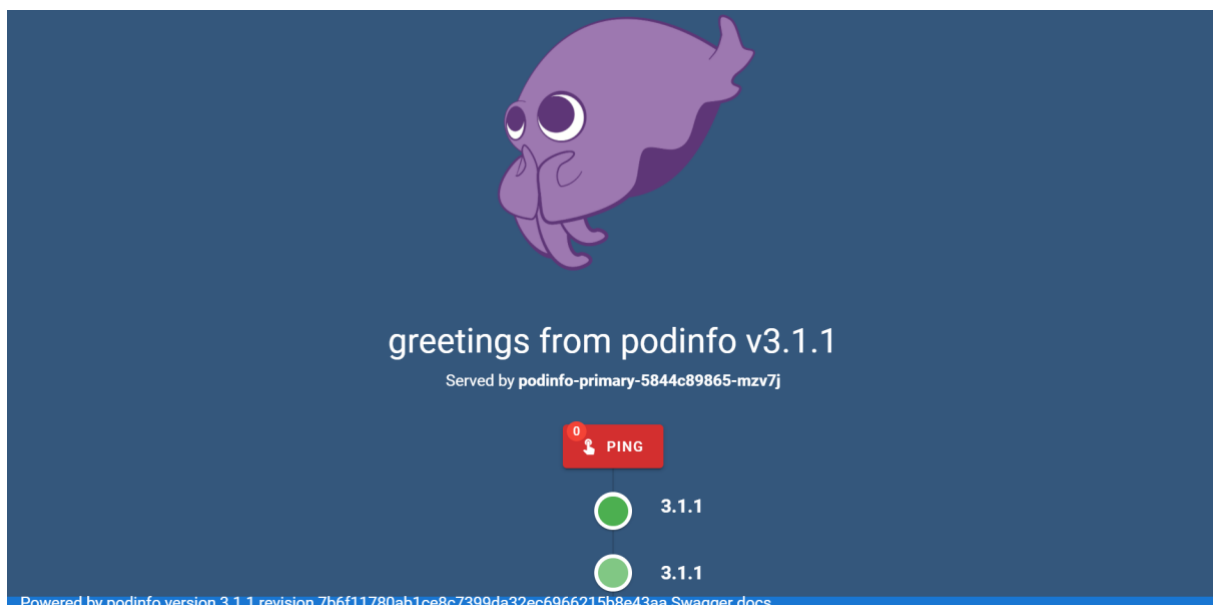
After that, add your domain name to the *podinfo-canary.yaml* and *podinfo-ingress.yaml* files.

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.



4.3.1 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 podinfo=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 podinfo=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, specify 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.

4.4 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*.

```
-
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 wanted metric. For the specification Prometheus own query language called *PromQL* is used.

```
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 files they need to be added to kubernetes with the *apply* command.

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

```
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.

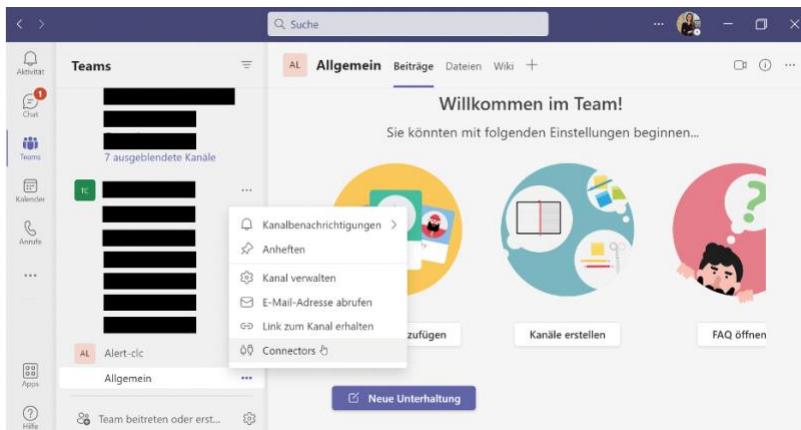
When restarting the deployment again these new metrics are contributing to failure counters which determine if a rollback is performed.

4.5 Microsoft Teams Alert

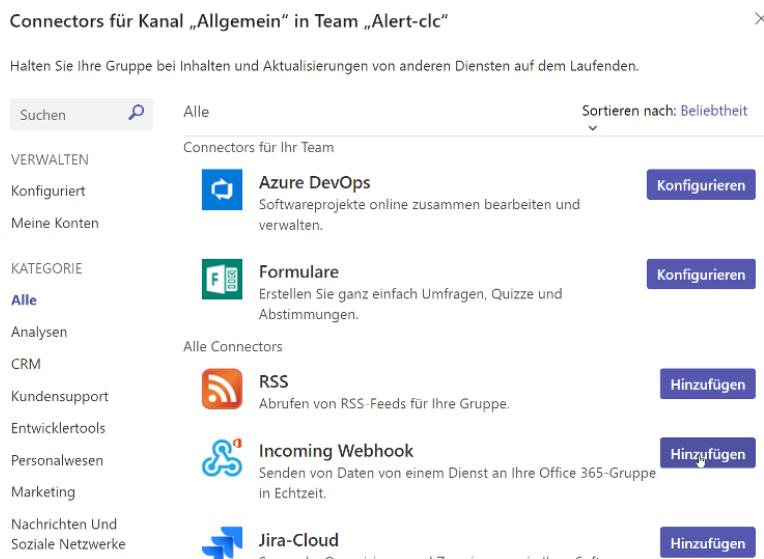
The goal now is to configure Flagger to send automatically generated notifications to MS teams regarding Canary deployment.

MS-Teams

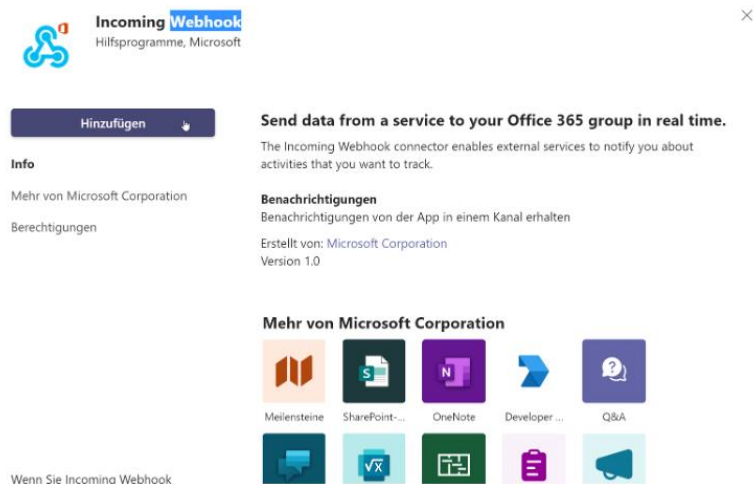
Go to the channel, where you want to receive alerts. Click on the ... on the right side of the channel name and select Connectors from the dropdown list.



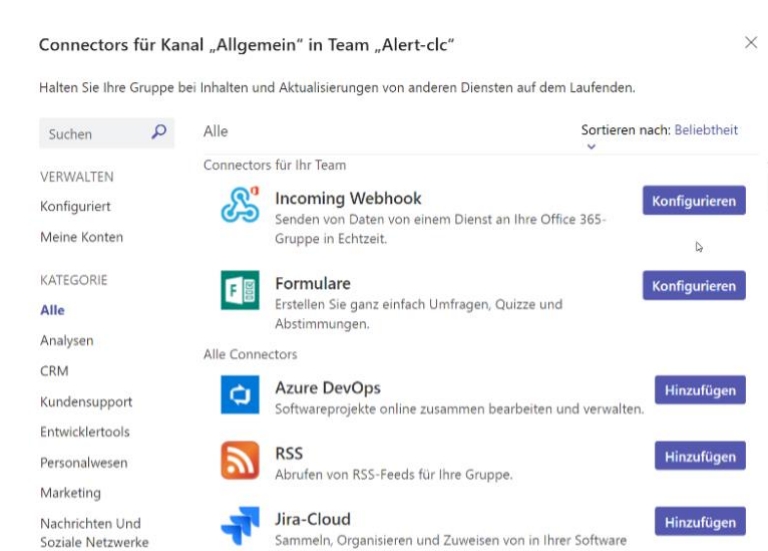
A window will then open. Add "Incoming Webhook".



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.

Connectors für Kanal „Allgemein“ in Team „Alert-clc“

 Incoming Webhook Feedback senden

Der eingehende WebHook Connector ermöglicht es externen Diensten, Sie über Aktivitäten zu benachrichtigen, die Sie nachverfolgen möchten. Um diesen Connector zu verwenden, müssen Sie bestimmte Einstellungen in dem anderen Dienst erstellen, der einen WebHook unterstützen muss, der mit dem Office 365 Connectorformat kompatibel ist.

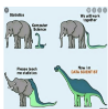
Mit * markierte Felder sind obligatorisch.

Geben Sie einen Namen für Ihre IncomingWebhook-Verbindung ein. *


Alert_CLC

Passen Sie das Bild an, das den Daten von diesem Incoming Webhook zugeordnet werden soll.

Bild hochladen



Kopieren Sie die unten angezeigte URL, um sie in der Zwischenablage zu speichern, und wählen Sie dann "Speichern" aus. Sie benötigen diese URL, wenn Sie zu dem Dienst wechseln, der Daten an Ihre Gruppe senden soll.

<https://officetestify633.webhook.office.co> 

Die URL ist aktuell.

Fertig **Entfernen**

Hinweis: Wenn Sie Softwareentwickler sind und weitere Informationen zum Senden von Daten an Office 365 mithilfe des Incoming Webhooks benötigen, finden Sie diese unter [Erste Schritte mit Office 365-Connectorkarten](#).

Configure Alertmanager via Flagger

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

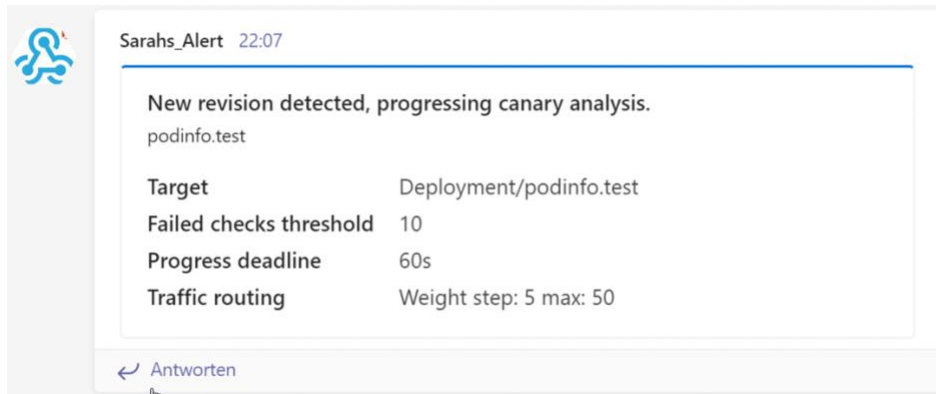
```
➤ helm upgrade -i flagger flagger/flagger \ --set msteams.url=https://outlook.office.com/webhook/YOUR/TEAMS/WEBHOOK \ --set msteams.proxy.url=my-http-proxy.com # optional http/s proxy
```

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

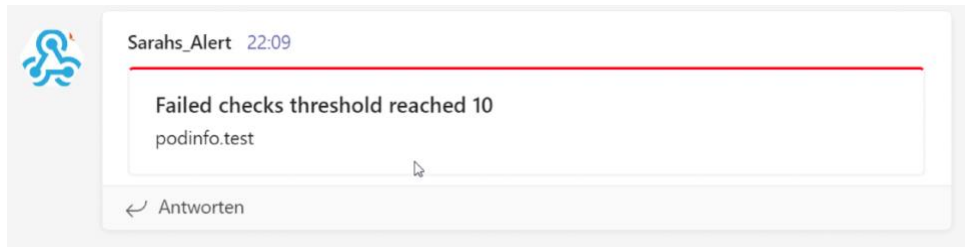
```
➤ helm upgrade -i flagger flagger/flagger \ --set msteams.url=https://officetestify633.webhook.office.com/webhookb2/63a11652-d1bd-4447-ae9d-927dee285a25@eb8e7f5a-975d-46b9-af52-d98412275d2b/IncomingWebhook/74ca1e25f2714b25a24cd4ac54d848f8/0db59b53-d85f-46d5-8595-28388336a0bd \ --set msteams.proxy.url=sarah.onthewifi.com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

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
```

4.6 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 file `prometheus.yml` should contain the information about which targets it is going to collect metrics. Below is an example of the `prometheus.yml` 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'

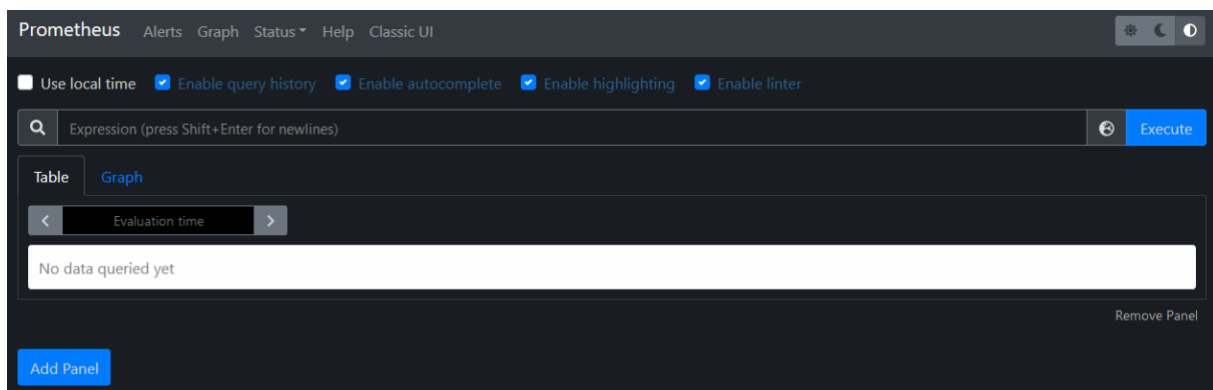
    # Override the global default and scrape targets from this job every 5
    # seconds.
    scrape_interval: 5s

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

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

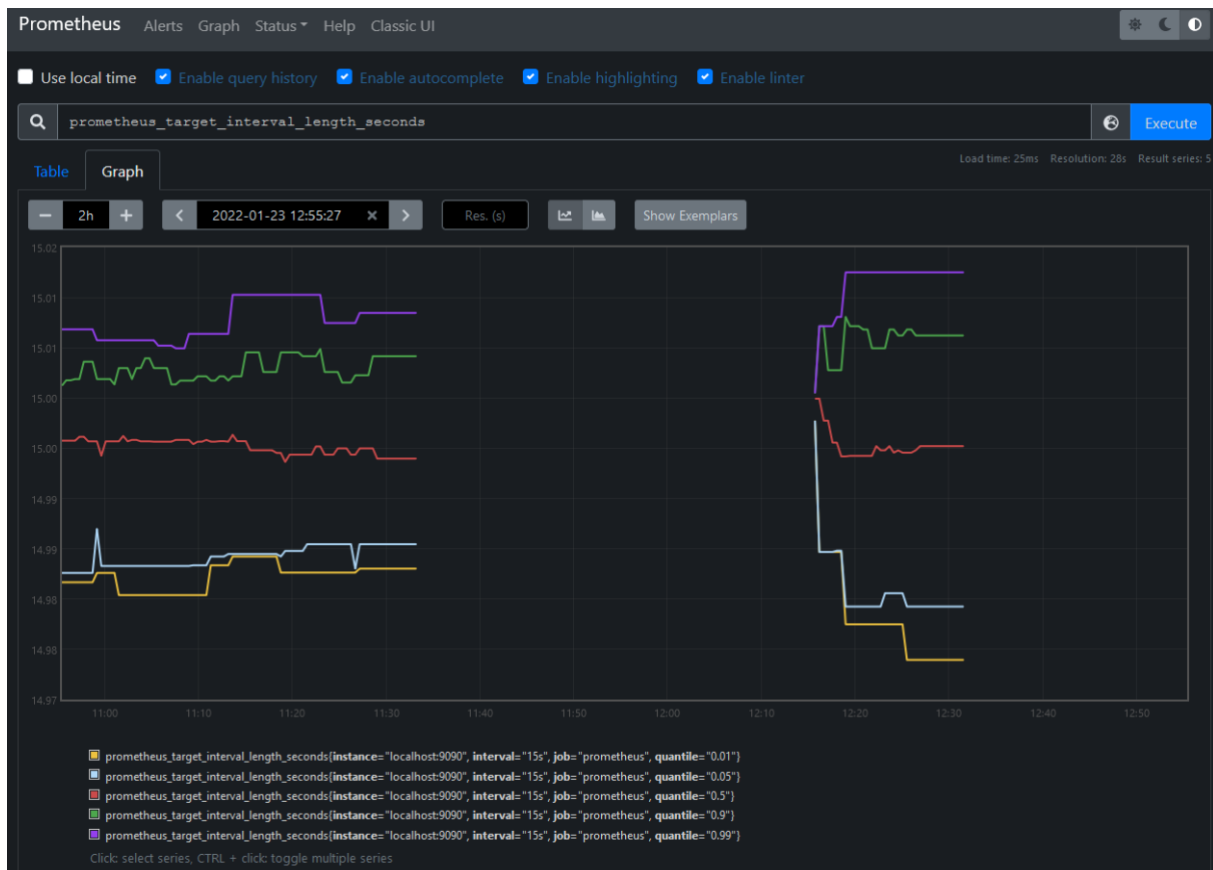
```
➤ ./prometheus --config.file=./prometheus.yml
```

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 be seen at the Panel. It is possible to see these 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

Load time: 41ms Resolution: 14s Result series: 11

Table Graph

Evaluation time

<code>prometheus_http_requests_total(code="200", handler="/-/ready", instance="localhost:9090", job="prometheus")</code>	2
<code>prometheus_http_requests_total(code="200", handler="/api/v1/label/name/values", instance="localhost:9090", job="prometheus")</code>	4
<code>prometheus_http_requests_total(code="200", handler="/api/v1/metadata", instance="localhost:9090", job="prometheus")</code>	1
<code>prometheus_http_requests_total(code="200", handler="/api/v1/query", instance="localhost:9090", job="prometheus")</code>	3
<code>prometheus_http_requests_total(code="200", handler="/api/v1/query_range", instance="localhost:9090", job="prometheus")</code>	30
<code>prometheus_http_requests_total(code="200", handler="/api/v1/targets", instance="localhost:9090", job="prometheus")</code>	2
<code>prometheus_http_requests_total(code="200", handler="/graph", instance="localhost:9090", job="prometheus")</code>	2
<code>prometheus_http_requests_total(code="200", handler="/metrics", instance="localhost:9090", job="prometheus")</code>	27
<code>prometheus_http_requests_total(code="200", handler="/static/filepath", instance="localhost:9090", job="prometheus")</code>	1
<code>prometheus_http_requests_total(code="200", handler="/targets", instance="localhost:9090", job="prometheus")</code>	1
<code>prometheus_http_requests_total(code="304", handler="/static/filepath", instance="localhost:9090", job="prometheus")</code>	6