Lab 9.4. Create a Trigger to Execute a Pipeline

Overview

In this example, we will demonstrate how to trigger a pipeline when a commit is made. We will be using the same tasks as before, including a **`git clone`** task to clone the repository, a **`build and push`** task using Kaniko, and a **`deploy to cluster`** task using helm. Below are the details of these tasks.

First Task (Git Clone)

This task enables you to clone a GitHub repository by providing its URL and branch name.

Apply the **`git clone`** task from the Tekton Hub using the following command:

```
kubectl apply -f
https://raw.githubusercontent.com/tektoncd/catalog/main/task/git-clone/0.9/g
it-clone.yaml -n tekton-pipelines
```

Second Task (Build and Push Image)

Create this **`task`** using the below manifest and name it as `build-push-docker-image-task.yaml`:

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: build-push-docker-image-task
spec:
  workspaces:
    - name: output
  params:
    - name: app_repo
    - name: container_image
    - name: container_tag
    - name: secret-name
  volumes:
    - name: kaniko-secret
      secret:
        secretName: $(params.secret-name) #name of the docker secret
        items:
          - key: .dockerconfigjson
            path: config.json
  steps:
    - name: build
      image: gcr.io/kaniko-project/executor:debug
      workingDir: "/workspace/output/"
      command: [/kaniko/executor]
      args:
        - --context=./
        - --destination=$(params.container_image):$(params.container_tag)
        - --force
      volumeMounts:
        - name: kaniko-secret
          mountPath: /kaniko/.docker/
```

In this task, we will be passing the **`secret`** name, **`image`** name, and **`tag`** name as parameters. The task will use the Kaniko command to build and push the image to the repository.

To use this task, we need to pre-apply a secret that contains the Docker registry credentials and pass the secret as a volume.

After creating the secret, apply this task using the following command:

```
kubectl apply -f build-push-docker-image-task.yaml -n tekton-pipelines
```

Third Task (Deploy to Cluster)

Create this **`task`** with the below manifest and name it as `deploy-to-cluster-task.yaml`:

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: deploy-to-cluster-task
  labels:
    app.kubernetes.io/version: "0.3"
  annotations:
    tekton.dev/pipelines.minVersion: "0.12.1"
    tekton.dev/categories: Deployment
    tekton.dev/tags: helm
    tekton.dev/platforms: "linux/amd64,linux/s390x,linux/ppc64le,linux/arm64"
spec:
  params:
    - name: charts_dir
      description: The directory in source that contains the helm chart
    - name: service
      description: The helm release version in semantic versioning format
    - name: release_name
      description: The helm release name
      default: <release name>
    - name: release_namespace
      description: The helm release namespace
      default: "default"
    - name: values_file
      description: "The values file to be used"
      default: "values.yaml"
    - name: tag
  workspaces:
    - name: output
  steps:
    - name: upgrade
      image: docker.io/kiwigrid/gcloud-kubectl-helm
      workingDir: /workspace/output
      script: |
        echo current installed helm releases
        helm list --namespace "$(params.release_namespace)"
        helm list -A
        echo installing helm chart...
        helm upgrade --install --wait --values
"$(params.charts_dir)/$(params.values_file)" --set tag="$(params.tag)"
--namespace "$(params.release_namespace)" "$(params.release_name)"
"$(params.charts_dir)" -debug
```

This task requires passing parameters such as `charts_dir`, `release_name`, `release_namespace`, `values_file`, and `tag`. It utilizes the `helm` command to deploy the code into the cluster.

Before applying this task, certain permissions need to be granted for our Service Account.

```
kubectl apply -f deploy-to-cluster-task.yaml -n tekton-pipelines
```

Pipeline

Let us create a `pipeline` using the following manifest and save it as `build-push-and-deploy-pipeline.yaml`:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: build-push-and-deploy-pipeline
```

```
spec:
  params:
    - name: gitrevision-tag
  workspaces:
    - name: shared-data
      description: |
        This workspace will receive the cloned git repo and be passed
        to the next Task for the repo's README.md file to be read.
    - name: shared-data-dep
  tasks:
    - name: fetch-repo
      taskRef:
        name: git-clone
      params:
        - name: url
          value: <your git-repo url>
        - name: revision
          value: $(params.gitrevision-tag)
      workspaces:
        - name: output
          workspace: shared-data
    - name: build-container-image
      runAfter: ["fetch-repo"]
      taskRef:
        name: build-push-docker-image-task
      params:
        - name: app_repo
          value: dir:///workspace/output/
        - name: container_image
          value: <image name>
        - name: container_tag
          value: <image tag>
        - name: secret-name
          value: < name of the secret>
      workspaces:
        - name: output
          workspace: shared-data
    - name: helm-clone
      runAfter: ["build-container-image"]
      taskRef:
        name: git-clone
      params:
        - name: url
          value: "<your repo URL>"
        - name: revision
          value: $(params.gitrevision-tag)
      workspaces:
        - name: output
```

```
          workspace: shared-data-dep
    - name: deploy-to-dev
      runAfter: ["helm-clone"]
      taskRef:
        name: deploy-to-cluster-task
      params:
        - name: charts_dir
          value: /workspace/output/<helm char path>
        - name: release_name
          value: <release name>
        - name: release_namespace
          value: <namespace>
        - name: values_file
          value: values.yaml
        - name: tag
          value: <Tag>
      workspaces:
        - name: output
          workspace: shared-data-dep
```

Apply this pipeline with the following command:

```
kubectl apply -f build-push-and-deploy-pipeline.yaml -n tekton-pipelines
```

Trigger

Let us create a `Trigger` for the pipeline we just created using the following manifests and name it as `build-push-and-deploy-trigger.yaml`:

```
---
# secret
apiVersion: v1
kind: Secret
metadata:
  name: github-secret
type: Opaque
stringData:
  secretToken: "1234567" # secret can be any number

---
# Event listener
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-event-listener
spec:
  serviceAccountName: <Service account name>
  triggers:
    - triggerRef: github-listener-trigger
  resources:
    kubernetesResource:
      serviceType: NodePort

---
# Trigger
apiVersion: triggers.tekton.dev/v1beta1
kind: Trigger
metadata:
  name: github-listener-trigger
spec:
  interceptors:
    - name: "verify-github-payload"
      ref:
        name: "github"
        kind: ClusterInterceptor
      params:
        - name: secretRef
          value:
            secretName: "github-secret"
            secretKey: "secretToken"
        - name: eventTypes
          value:
            - "push" # for GitLab its "Push Hook" and "Tag Push Hook"
    - name: "CEL filter: only when PRs are opened"
      ref:
        name: "cel"
      params:
        - name: "overlays"
          value:
            - key: branch_name
              expression: "body.ref.split('/')[2]" #Here we are splitting the
body.ref as it gives output "ref/head/<branch name or tag>" with "/" to grep only
branch name.
  bindings:
    - ref: binding
  template:
    ref: trigger-template

---
# Trigger Binding
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerBinding
metadata:
  name: binding
spec:
  params:
    - name: gitrevision-tag
      value: $(extensions.branch_name) #the branch name that we split in trigger

---
# Trigger Template
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerTemplate
metadata:
  name: trigger-template
spec:
  params:
    - name: gitrevision-tag
  resourcetemplates:
    - apiVersion: tekton.dev/v1beta1
      kind: PipelineRun
      metadata:
        generateName: build-push-and-deploy-run-
      spec:
        serviceAccountName: <service account name>
        pipelineRef:
          name: build-push-and-deploy-pipeline
        podTemplate:
          securityContext:
            fsGroup: 1001
        params:
          - name: gitrevision-tag
            value: $(tt.params.gitrevision-tag)
        workspaces:
          - name: shared-data #workspace for build pipeline
            volumeClaimTemplate:
              spec:
                accessModes:
                  - ReadWriteOnce
                resources:
                  requests:
                    storage: 1Gi
          - name: shared-data-dep #workspace for helm charts
            volumeClaimTemplate:
              spec:
                accessModes:
                  - ReadWriteOnce
                resources:
                  requests:
                    storage: 1Gi
```

To create a trigger for your Git repository using the manifests for `Secret`, `EventListener`, `Trigger`, `Trigger Binding`, and `Trigger Template`, use the following command to apply the manifest:

```
kubectl create -f build-push-and-deploy-trigger.yaml -n tekton-pipelines
```

After applying this trigger file, verify the EventListener object; its `READY` status should be *True*. Check it with the following commands:

```
kubectl get el -n tekton-pipelines
kubectl get pods -n tekton-pipelines
```

After ensuring that all pods are ready, create an `ingress` to enable communication between GitHub and Tekton for the EventListener. Use the following manifest to write the ingress file and name it `triggers-ingress-resource.yaml`:

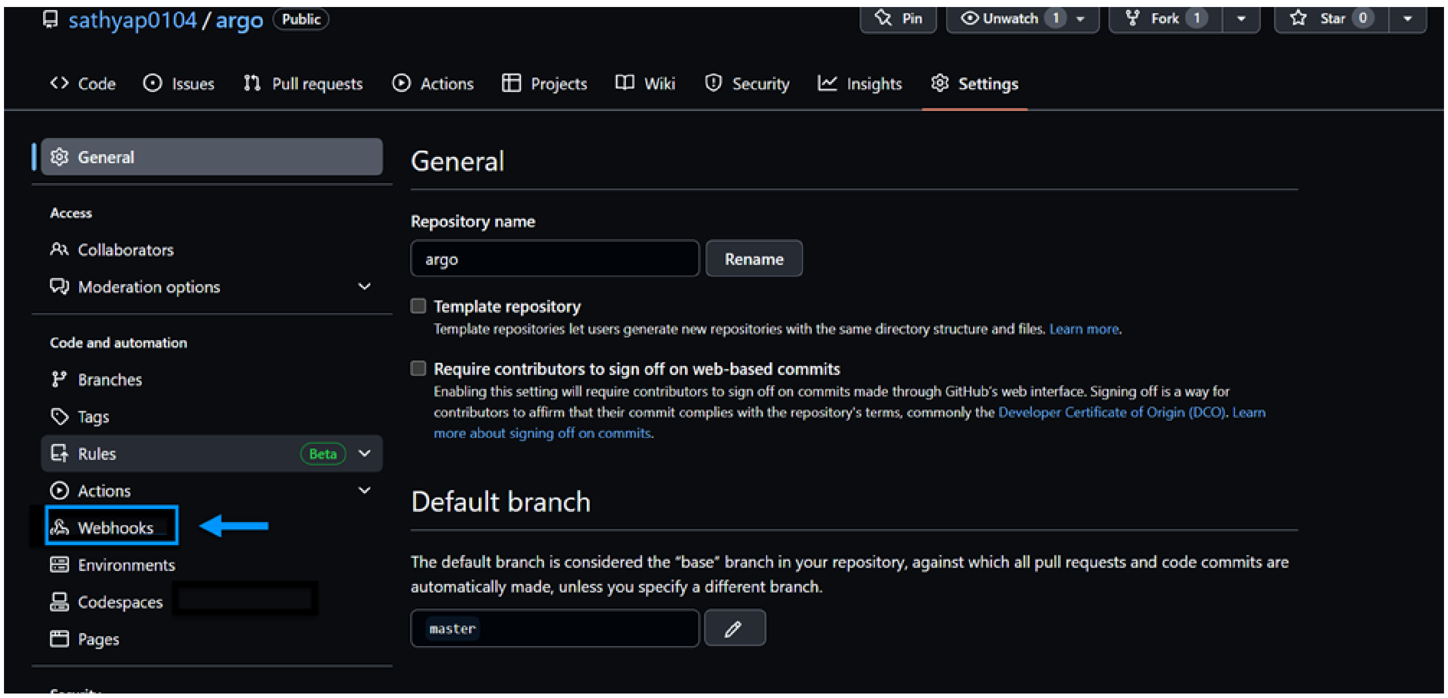
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: triggers-ingress-resource
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
    - host: tekton.<external IP>.nip.io # write the domain name if u have one
      http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: el-github-event-listener
                port:
                  number: 8080
```

We will now apply the ingress resource file and obtain the IP address to configure the webhook. To do this, execute the following command:

```
kubectl apply -f triggers-ingress-resource.yaml -n tekton-pipelines
kubectl get ing -n tekton-pipelines
```

To run the trigger in the cluster, we require some permissions for our service account.
To configure the eventListener with GitHub using a webhook, follow these four steps:

1. Go to your GitHub repository and navigate to the repository settings.



[redacted]

General

Repository name

argo

Template repository
Template repositories let users generate new repositories with the same directory structure and files. Learn more.

Require contributors to sign off on web-based commits
Enabling this setting will require contributors to sign off on commits made through GitHub's web interface. Signing off is a way for contributors to affirm that their commit complies with the repository's terms, commonly the Developer Certificate of Origin (DCO). Learn more about signing off on commits.

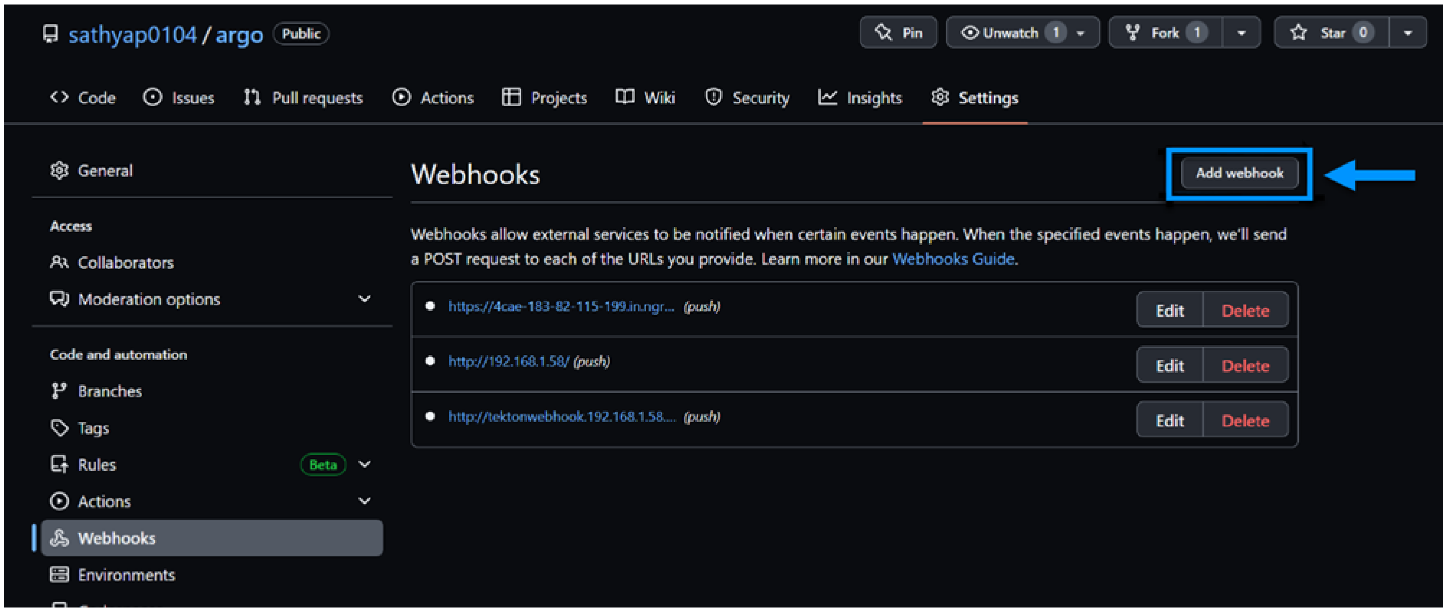
Default branch

The default branch is considered the "base" branch in your repository, against which all pull requests and code commits are automatically made, unless you specify a different branch.

```
master
```

Adding Webhook: Going to Webhook Settings

2. In the left menu bar, select *Webhooks* and click on *Add webhook*.



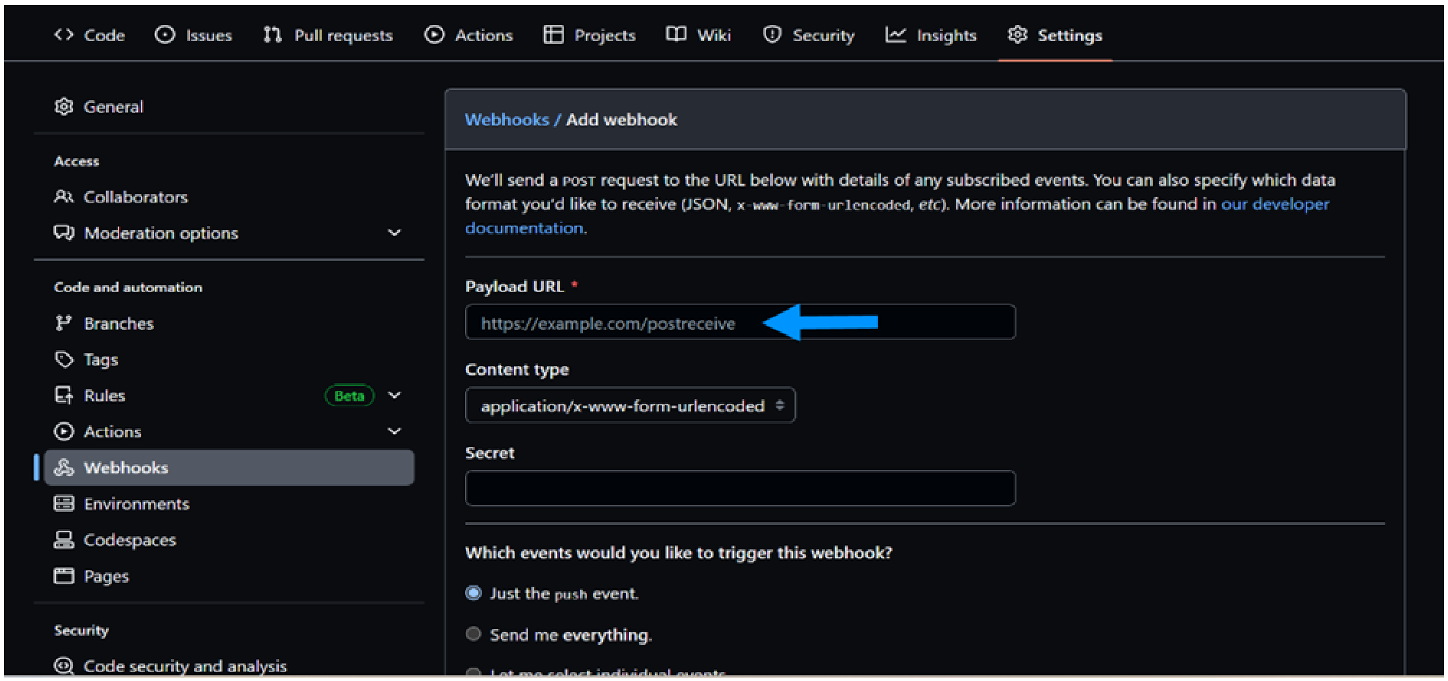
Webhooks

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

- https://4cae-183-82-115-199.in.ngr... (push)
- http://192.168.1.58/ (push)
- http://tektonwebhook.192.168.1.58.... (push)

Adding Webhook: Creating New Webhook

3. In the *Payload URL* field, enter the external IP address or domain name of your ingress, followed by a forward slash (/).



Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, *etc*). More information can be found in our developer documentation.

Payload URL *

https://example.com/postreceive

Content type

application/x-www-form-urlencoded

Secret

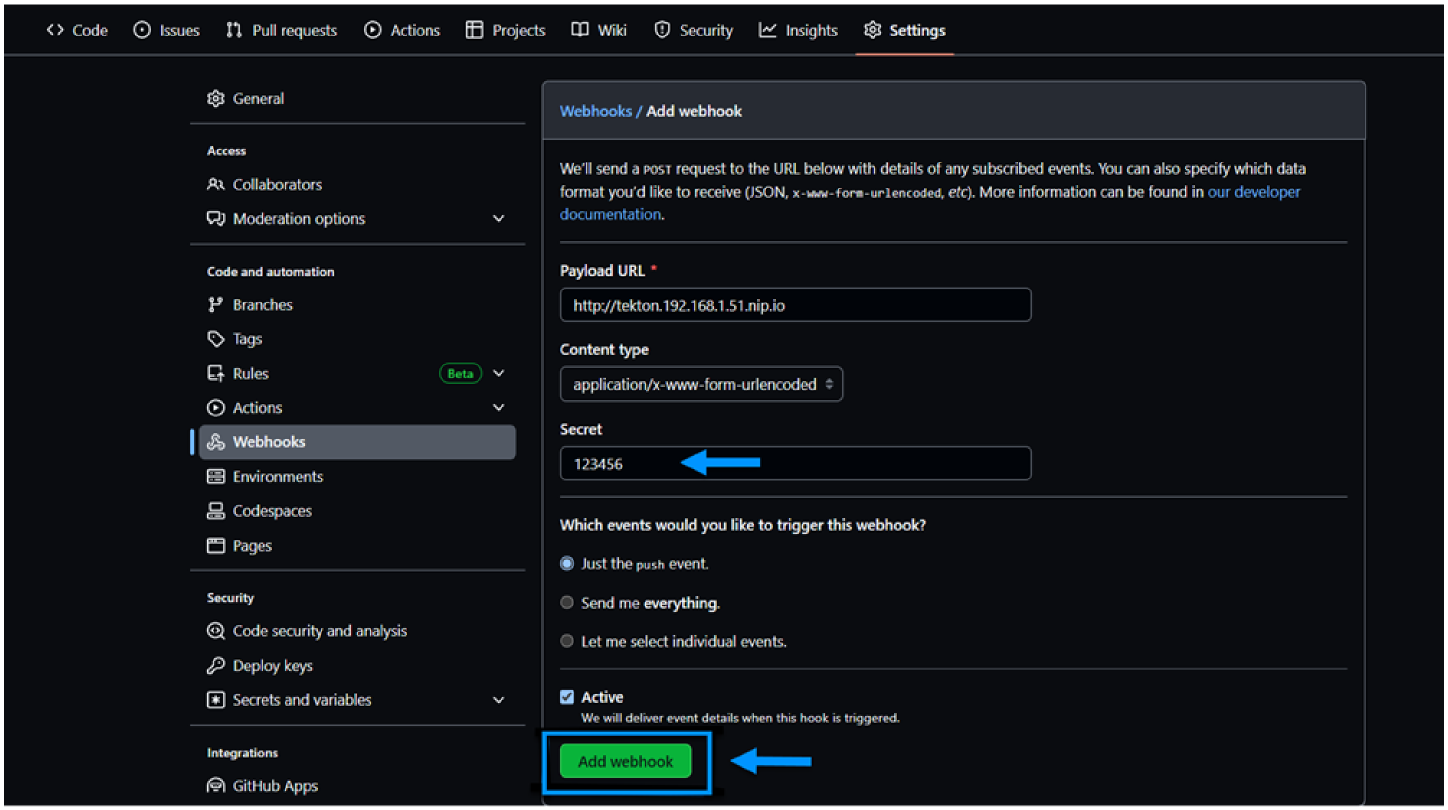
Which events would you like to trigger this webhook?

Just the push event.

Send me everything.

Adding Webhook: Configuring the Webhook

4. In the *Secret* field, enter the secret token that was passed in the `trigger` file. In this example, the secret token is '1234567'. Finally, click the *Add Webhook* button at the bottom of the page.



Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, *etc*). More information can be found in our developer documentation.

Payload URL *

http://tekton.192.168.1.51.nip.io

Content type

application/x-www-form-urlencoded

Secret

123456

Which events would you like to trigger this webhook?

Just the push event.

Send me everything.

Let me select individual events.

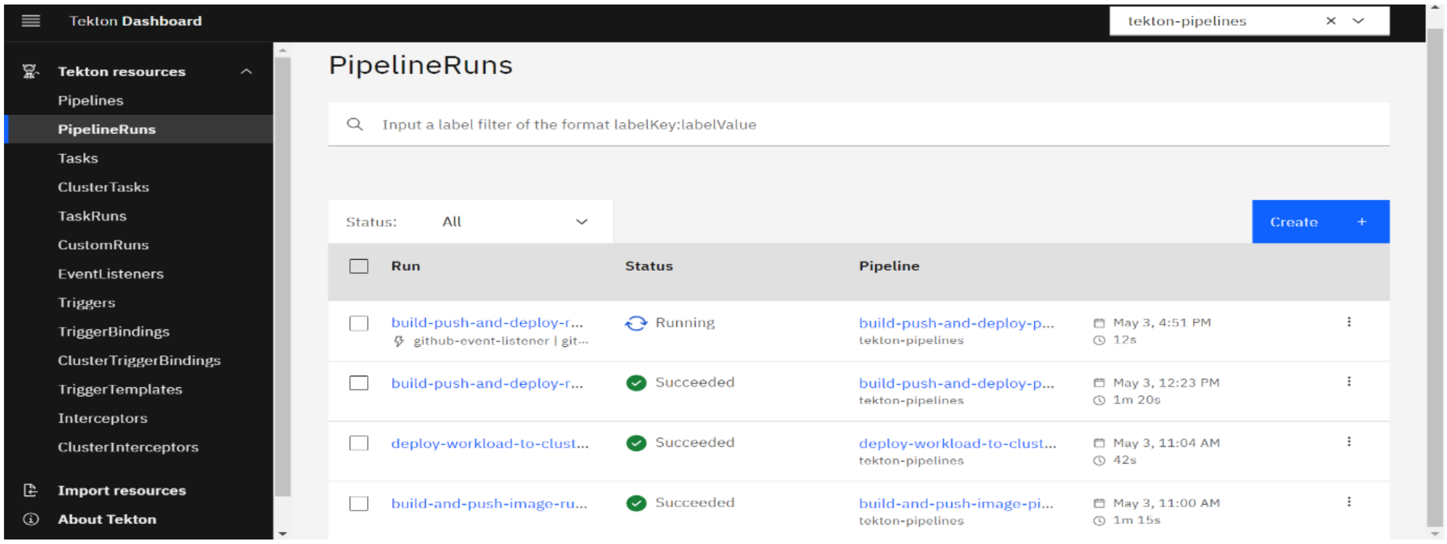
Active

We will deliver event details when this hook is triggered.

Add webhook

Adding Webhook: Final Step

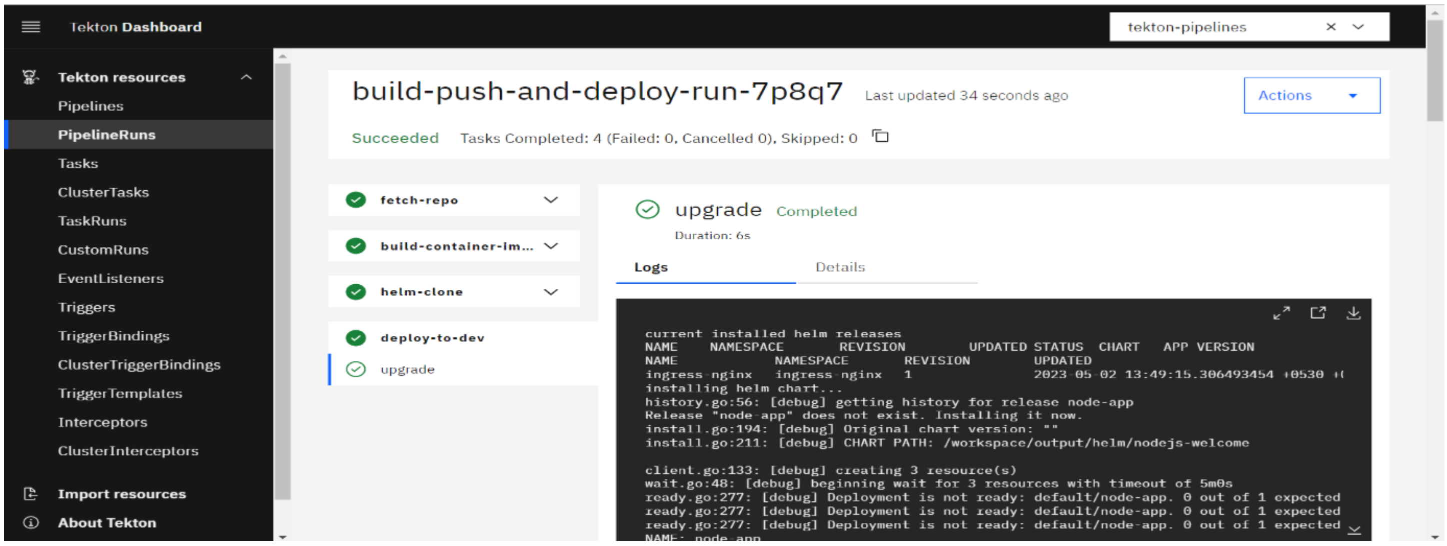
You are all set! Now, each time there is a push or commit in the repository, the pipeline will be triggered.



PipelineRuns

Run	Status	Pipeline	
build-push-and-deploy-r... github-event-listener | git...	Running	build-push-and-deploy-p... tekton-pipelines	May 3, 4:51 PM 12s
build-push-and-deploy-r...	Succeeded	build-push-and-deploy-p... tekton-pipelines	May 3, 12:23 PM 1m 20s
deploy-workload-to-clust...	Succeeded	deploy-workload-to-clust... tekton-pipelines	May 3, 11:04 AM 42s
build-and-push-image-ru...	Succeeded	build-and-push-image-pi... tekton-pipelines	May 3, 11:00 AM 1m 15s

Screenshot of Triggering Build, Push and Deploy Pipeline



build-push-and-deploy-run-7p8q7 Last updated 34 seconds ago

Succeeded Tasks Completed: 4 (Failed: 0, Cancelled 0), Skipped: 0

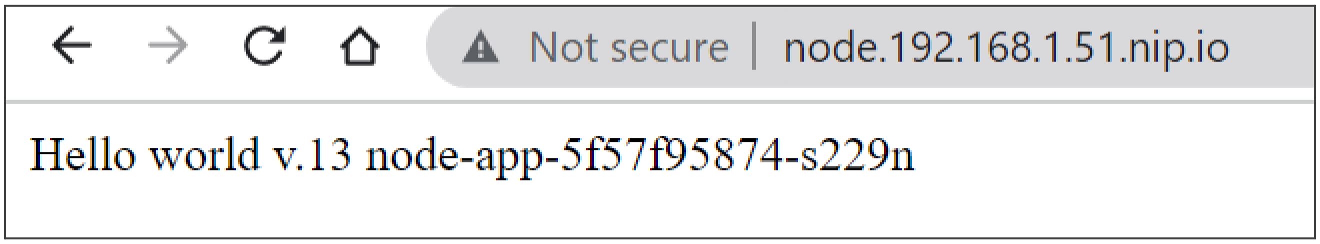
upgrade Completed

Duration: 6s

```
current installed helm releases
NAME    NAMESPACE       REVISION        UPDATED STATUS  CHART   APP VERSION
NAME            NAMESPACE       REVISION        UPDATED
ingress-nginx   ingress-nginx   1               2023-05-02 13:49:15.306493454 +0530 +(
installing helm chart...
history.go:56: [debug] getting history for release node-app
Release "node-app" does not exist. Installing it now.
install.go:194: [debug] Original chart version: ""
install.go:211: [debug] CHART PATH: /workspace/output/helm/nodejs-welcome

client.go:133: [debug] creating 3 resource(s)
wait.go:48: [debug] beginning wait for 3 resources with timeout of 5m0s
ready.go:277: [debug] Deployment is not ready: default/node-app. 0 out of 1 expected
ready.go:277: [debug] Deployment is not ready: default/node-app. 0 out of 1 expected
ready.go:277: [debug] Deployment is not ready: default/node-app. 0 out of 1 expected
NAME: node-app
```

Screenshot of Successfully Triggered Build, Push and Deploy Pipeline



Hello world v.13 node-app-5f57f95874-s229n

Example Node js App Output