18 | 自托管构建：如何使用 Tekton 构建镜像？

2023-01-18 王炜 来自北京

《云原生架构与GitOps实战》

课程介绍 >

讲述：王炜

时长 14:53　大小 13.59M

你好，我是王炜。

前两节课，我们分别介绍了如何使用 GitHub Action Workflow 和 GitLab CI 来自动构建镜像，它们配置起来相对简单，作为平台自带的功能，也不需要花费额外的维护成本。并且，它们都和 Git 仓库深度整合，这就让触发流水线变得非常简单了。

对于构建次数较少的团队来说，在免费额度范围内使用它们是一个非常好的选择。但是对于构建次数非常频繁的中大型团队来说，综合考虑费用、可控和定制化等各方面因素，他们可能会考虑使用其他自托管的方案。

这节课，我们就来介绍其中一种自动构建镜像的自托管方案：**使用 Tekton 来自动构建镜像。**Tekton 是一款基于 Kubernetes 的 CI/CD 开源产品，如果你已经有一个 Kubernetes 集群，那么利用 Tekton 直接在 Kubernetes 上构建镜像是一个不错的选择。

我会首先带你了解 Tekton 的基本概念，然后我们仍然以示例应用为例，从零开始为示例应用配置构建镜像的流水线，并结合 GitHub 为 Tekton 配置 Webhook 触发器，实现提交代码之后触发 Tekton 流水线并构建镜像，最后推送到镜像仓库的过程。

在学完这节课之后，你将基本掌握 Tekton 的流水线以及触发器的用法，并具备独立配置它们的能力。

在开始今天的学习之前，你需要具备以下前提条件。

- 在本地安装了 kubectl。
- 已经按照第 16 讲将 kubernetes-example 示例应用代码推送到了自己的 GitHub 仓库中。

准备 Kubernetes 集群

由于我们在实践的过程中需要 Kubernetes 集群的 Loadbalancer 能力，所以，首先你需要准备一个云厂商的 Kubernetes 集群，你可以使用 AWS、阿里云或腾讯云等任何云厂商。这里我以开通腾讯云 TKE 集群为例演示一下，**这部分内容比较基础，如果你已经有云厂商 Kubernetes 集群，或者熟悉开通过程，都可以跳过这个步骤。**

首先，登录腾讯云并在这个页面打开 TKE 控制台，点击“新建”按钮，选择“标准集群”。

在创建集群页面输入“集群名称”，“所在地域”项中选择“中国香港”，集群网络选择“Default”，其他信息保持默认，点击下一步。

接下来进入到 Worker 节点配置阶段。在“机型”一栏中选择一个 2 核 8G 的节点，在“公网带宽”一栏中将带宽调整为 100Mbps，并且按量计费。

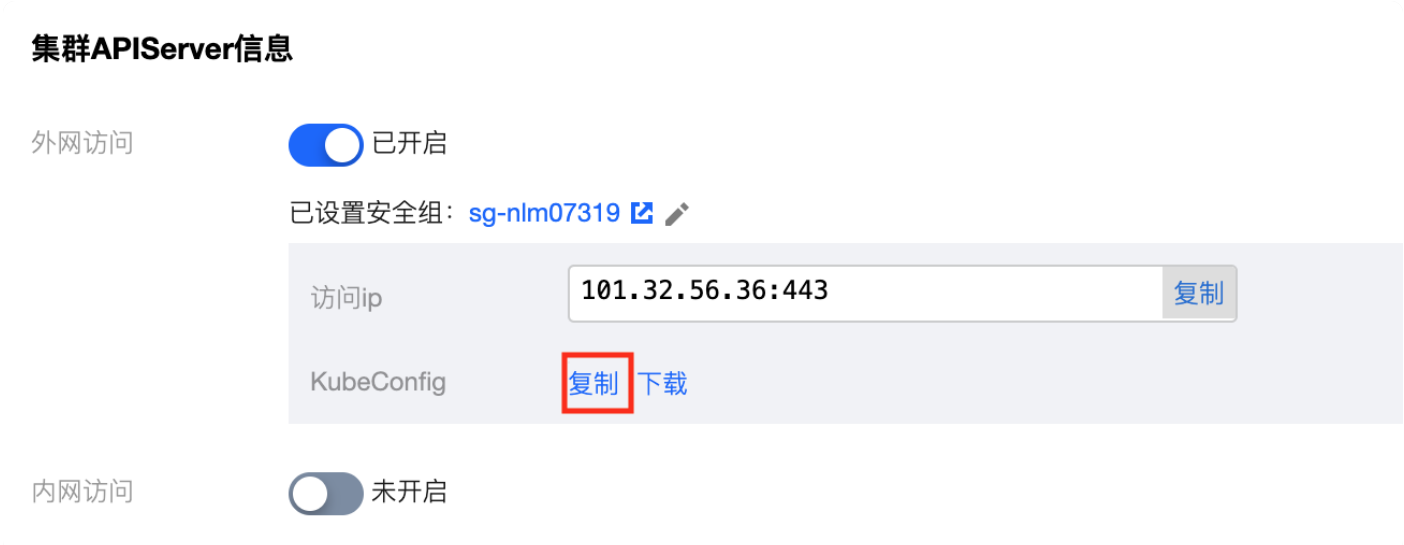
点击“下一步”。后续的页面都保持默认选择，点击“完成”创建 Kubernetes 集群。这里需要等待几分钟直至集群准备就绪。

点击集群名称“docker-build”进入集群详情页，在“集群 APIServer 信息”一栏找到“外网访问”，点击开关来开启集群的外网访问。

在弹出的新窗口中，选择“Default”安全组并选择“按使用流量”计费，访问方式选择“公网 IP”，然后点击“保存”开通集群外网访问。

等待“外网访问”开关转变为启用状态。

接下来，在“Kubeconfig”一栏点击“复制”，复制集群 Kubeconfig 信息。



接下来，将集群证书信息内容写入到本地的 ~/.kube/config 文件内，这是 kubectl 默认读取 kubeconfig 的文件位置。

为了避免覆盖已有的 kubeconfig，首先你需要备份 ~/.kube/config 文件。

```
$ mv ~/.kube/config ~/.kube/config-bak
```


然后新建 ~/.kube/config 文件，将刚才复制的 Kubeconfig 内容写入到该文件内。

最后，执行 kubectl get node 来验证 kubectl 与集群的联通性。

复制代码

```
$ kubectl get node
NAME            STATUS   ROLES    AGE     VERSION
172.19.0.107    Ready    <none>   5m21s   v1.22.5-tke.5
```

待 Node 信息返回后，我们的 Kubernetes 集群也就准备好了。

安装组件

准备好云厂商 Kubernetes 集群之后，接下来我们需要安装两个组件，分别是 Tekton 相关的组件以及 Ingress-Nginx。

Tekton

首先，安装 Tekton Operator。

复制代码

```
$ kubectl apply -f https://storage.googleapis.com/tekton-releases/pipeline/late
namespace/tekton-pipelines created
podsecuritypolicy.policy/tekton-pipelines created
......
```

等待 Tekton 所有的 Pod 就绪。

复制代码

```
$ kubectl wait --for=condition=Ready pods --all -n tekton-pipelines --timeout=3
pod/tekton-pipelines-controller-799f9f989b-hxmlx condition met
```

```
pod/tekton-pipelines-webhook-556f9f7476-sgx2n condition met
```

接下来，安装 Tekton Dashboard。

复制代码

```
$ kubectl apply -f https://storage.googleapis.com/tekton-releases/dashboard/lat
serviceaccount/tekton-dashboard created
role.rbac.authorization.k8s.io/tekton-dashboard-info created
......
```

然后，分别安装 Tekton Trigger 和 Tekton Interceptors 组件。

复制代码

```
$ kubectl apply -f https://storage.googleapis.com/tekton-releases/triggers/late
podsecuritypolicy.policy/tekton-triggers created
clusterrole.rbac.authorization.k8s.io/tekton-triggers-admin created
......

$ kubectl apply -f https://storage.googleapis.com/tekton-releases/triggers/late
deployment.apps/tekton-triggers-core-interceptors created
service/tekton-triggers-core-interceptors created
......
```

等待所有 Tekton 的所有组件的 Pod 都处于就绪状态，Tekton 就部署完成了。

复制代码

```
$ kubectl wait --for=condition=Ready pods --all -n tekton-pipelines --timeout=3
pod/tekton-dashboard-5d94c7f687-8t6p2 condition met
pod/tekton-pipelines-controller-799f9f989b-hxmlx condition met
pod/tekton-pipelines-webhook-556f9f7476-sgx2n condition met
pod/tekton-triggers-controller-bffdd47cf-cw7sv condition met
pod/tekton-triggers-core-interceptors-5485b8bd66-n9n2m condition met
pod/tekton-triggers-webhook-79ddd8d6c9-f79tg condition met
```

Ingress-Nginx

安装完 Tekton 之后，我们再来安装 Ingress-Nginx。

复制代码

```
$ kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/kubern
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
serviceaccount/ingress-nginx-admission created
......
```

等待所有 Ingress-Nginx Pod 处于就绪状态，Ingress-Nginx 就部署完成了。

复制代码

```
$ kubectl wait --for=condition=AVAILABLE deployment/ingress-nginx-controller --
deployment.apps/ingress-nginx-controller condition met
```

暴露 Tekton Dashboard

配置好 Tekton 和 Ingress-Nginx 之后，为了方便访问 Tekton Dashboard，我们要通过 Ingress 的方式暴露它，将下列内容保存为 tekton-dashboard.yaml。

复制代码

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-resource
  namespace: tekton-pipelines
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
    - host: tekton.k8s.local
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: tekton-dashboard
                port:
                  number: 9097
```

然后，执行 kubectl apply 将它应用到集群内。

复制代码

```
$ kubectl apply -f tekton-dashboard.yaml
ingress.networking.k8s.io/ingress-resource created
```


接下来，获取 Ingress-Nginx Loadbalancer 的外网 IP 地址。你可以使用 kubectl get service 来获取。

复制代码

```
$ kubectl get services --namespace ingress-nginx ingress-nginx-controller --out
43.135.82.249
```

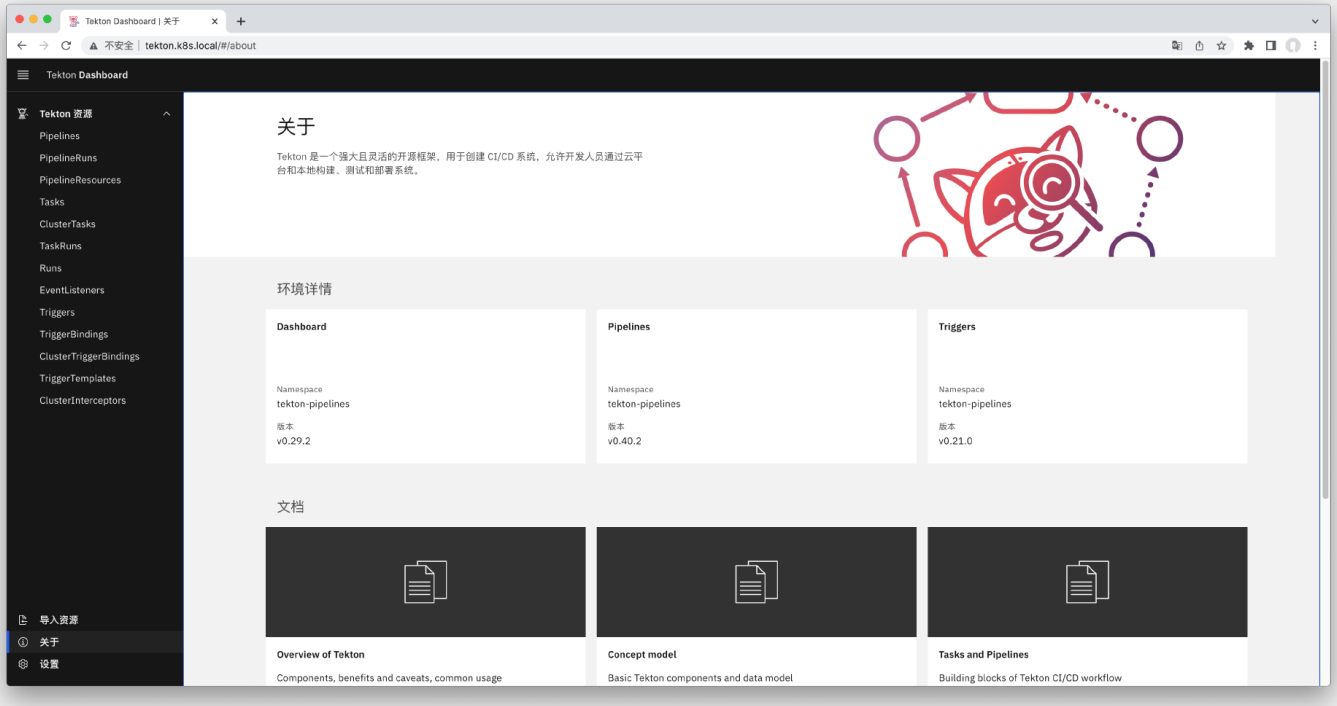
由于之前我在 Ingress 策略中配置的是一个虚拟域名，**所以需要在本地配置 Hosts**。当然你也可以将 Ingress 的 host 修改为实际的域名，并且为域名添加 DNS 解析，也能达到同样的效果。

以 Linux 系统为例，要修改 Hosts，你需要将下面的内容添加到 /etc/hosts 文件内。

复制代码

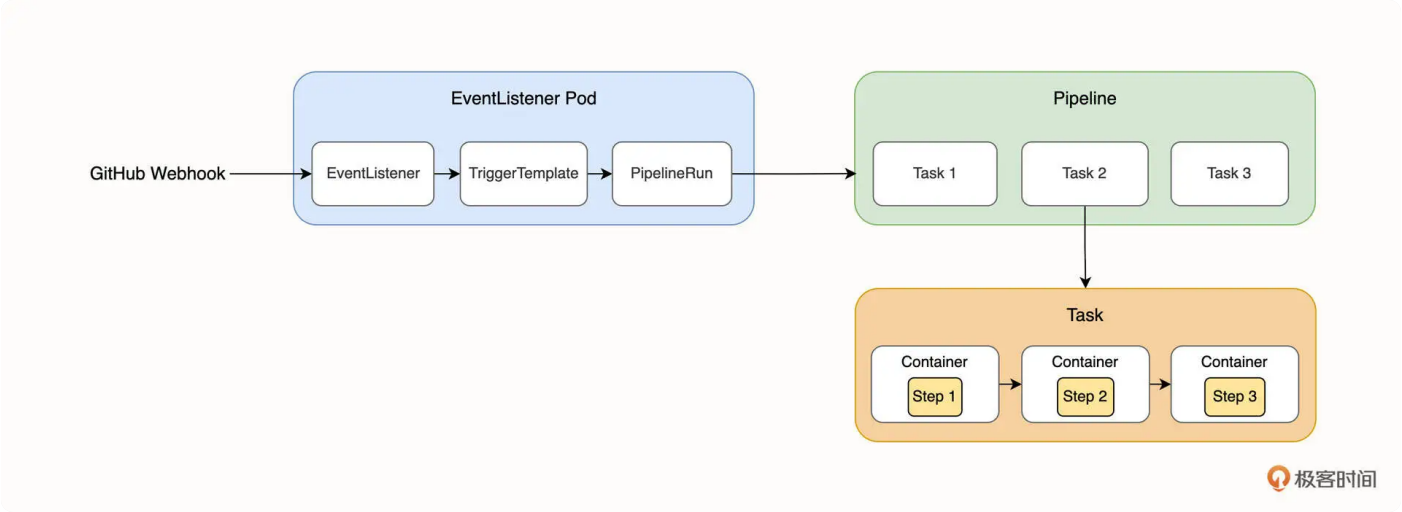
```
43.135.82.249 tekton.k8s.local
```

然后，你就可以通过域名 http://tekton.k8s.local 来访问 Tekton Dashboard 了。



Tekton 简介

在正式创建 Tekton 流水线之前，你需要先了解一些基本概念，你可以结合下面这张图来理解一下。



在上面这张图中，从左往右依次出现了这几个概念：EventListener、TriggerTemplate、PipelineRun、Pipeline、Task 以及 Step，接下来，我就简单介绍一下它们。

EventListener

EventListener 顾名思义是一个事件监听器，它是外部事件的入口。EventListener 通常以 HTTP 的方式对外暴露，在我们这节课的例子中，我们会在 GitHub 创建 WebHook 来调用 Tekton 的 EventListener，使它能接收到仓库推送事件。

TriggerTemplate

当 EventListener 接收到外部事件之后，它会调用 Trigger 也就是触发器，而 TriggerTemplate 是用来定义接收到事件之后需要创建的 Tekton 资源的，例如创建一个 PipelineRun 对象来运行流水线。这节课，我们会使用 TriggerTemplate 来创建 PipelineRun 资源。

Step

Step 是流水线中的一个具体的操作，例如构建和推送镜像操作。Step 接收镜像和需要运行的 Shell 脚本作为参数，Tekton 将会启动镜像并执行 Shell 脚本。

Task

Task 是一组有序的 Step 集合，每一个 Task 都会在独立的 Pod 中运行，Task 中不同的 Step 将在同一个 Pod 不同的容器内运行。

Pipeline

Pipeline 是 Tekton 中的一个核心组件，它是一组 Task 的集合，Task 将组成一组有向无环图（DAG），Pipeline 会按照 DAG 的顺序来执行。

PipelineRun

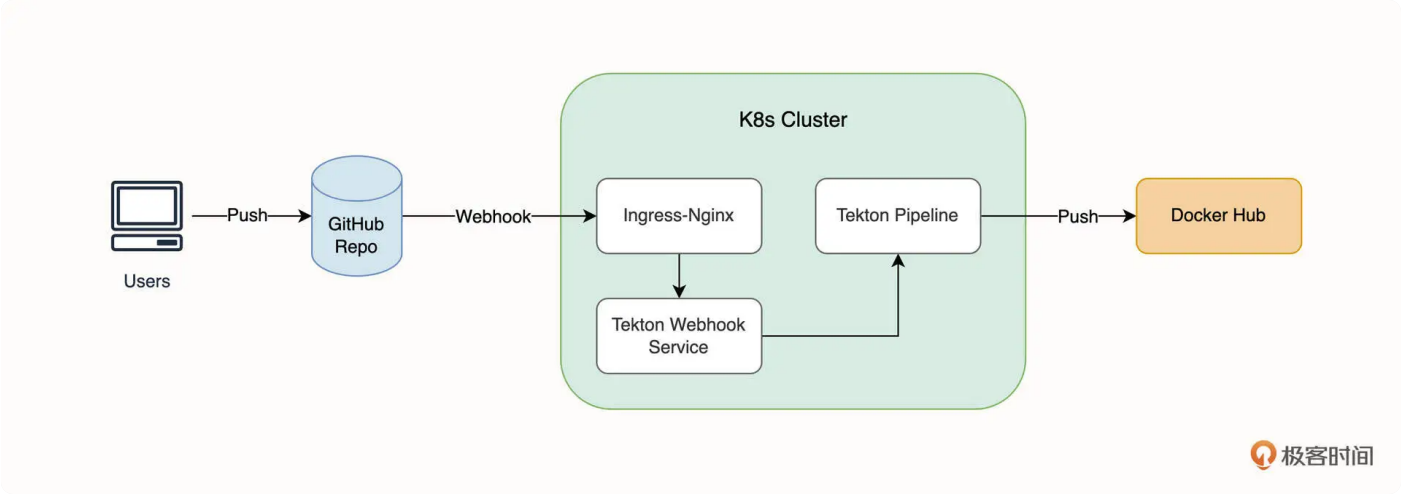
PipelineRun 实际上是 Pipeline 的实例化，它负责为 Pipeline 提供输入参数，并运行 Pipeline。例如，两次不同的镜像构建操作对应的就是两个不同的 PipelineRun 资源。

创建 Tekton Pipeline

可以看出，Tekton 的概念确实比较多，抽象也不好理解。别担心，接下来我们就实际创建一下流水线，在这个过程中不断加深理解。

概述

我们先来看看这节课创建的 Tekton 流水线最终可以实现的效果，如下图所示。



简单来说，当我们向 GitHub 推送代码时，GitHub 将以 HTTP 请求的方式通知集群内的 Tekton 触发器，触发器通过 Ingress-Nginx 对外暴露，当触发器接收到来自 GitHub 的事件推送时，将通过 TriggerTemplate 来创建 PipelineRun 运行 Pipeline，最终实现镜像的自动构建和推送。

创建 Task

好，下面我们正式开始实战。**在创建 Pipeline 之前，我们需要先创建两个 Task，这两个 Task 分别负责“检出代码”还有“构建和推送镜像”。**

首先创建检出代码的 Task。

```
$ kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhan
task.tekton.dev/git-clone created
```

这个 Task 是 Tekton 官方提供的插件，它和 GitHub Action 的 checkout 插件有一点类似，主要作用是检出代码。在这里，我们不需要理解它具体的细节。

然后，创建构建和推送镜像的 Task。

```
$ kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhan
task.tekton.dev/docker-socket configured
```

简单介绍一个这个 Task，关键内容如下。

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: docker-socket
spec:
  workspaces:
    - name: source
  params:
    - name: image
      description: Reference of the image docker will produce.
    ......
  steps:
    - name: docker-build
      image: docker:stable
      env:
        ......
        - name: IMAGE
          value: $(params.image)
```

```
        - name: DOCKER_PASSWORD
          valueFrom:
            secretKeyRef:
              name: registry-auth
              key: password
        - name: DOCKER_USERNAME
          valueFrom:
            secretKeyRef:
              name: registry-auth
              key: username
      workingDir: $(workspaces.source.path)
      script: |
        cd $SUBDIRECTORY
        docker login $REGISTRY_URL -u $DOCKER_USERNAME -p $DOCKER_PASSWORD
        if [ "${REGISTRY_URL}" = "docker.io" ] ; then
          docker build --no-cache -f $CONTEXT/$DOCKERFILE_PATH -t $DOCKER_USERN
          docker push $DOCKER_USERNAME/$IMAGE:$TAG
          exit
        fi
        docker build --no-cache -f $CONTEXT/$DOCKERFILE_PATH -t $REGISTRY_URL/$
        docker push $REGISTRY_URL/$REGISTRY_MIRROR/$IMAGE:$TAG
      volumeMounts: # 共享 docker.socket
        - mountPath: /var/run/
          name: dind-socket
  sidecars: #sidecar 提供 docker daemon
    - image: docker:dind
      ......
```


spec.params 字段用来定义变量，并最终由 PipelineRun 提供具体的值。

spec.steps 字段用来定义具体的执行步骤，例如，我们在这里使用 docker:stable 镜像创建了容器，并将 spec.params 定义的变量以 ENV 的方式传递到容器内部，其中 DOCKER_PASSWORD 和 DOCKER_USERNAME 两个变量来源于 Secret，我们将在后续创建。

spce.steps[0].script 字段定义了具体执行的命令，这里执行了 docker login 登录到 Docker Hub，并且使用了 docker build 和 docker push 来构建和推送镜像。我们对 Docker Hub 和其他镜像仓库做了区分，以便使用不同的 TAG 命名规则。

spce.sidecars 字段为容器提供 Docker daemon，它使用的镜像是 docker:dind。

仔细回想一下上节课的内容你会发现，**这个 Task 定义的具体行为和 GitLab CI 定义的流水线非常类似**，它们都是指定一个镜像，然后运行一段脚本，并且都是用 DiND 的方式来构建和推

送镜像的。

创建 Pipeline

创建完 Task 之后，由于它们实现的具体功能是独立的，所以我们需要将他们联系起来。也就是说，我们希望 Pipeline 先克隆代码，再构建和推送镜像。所以，下面我们需要创建 Pipeline 来引用这两个 Task。

复制代码

```
$ kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhan
pipeline.tekton.dev/github-trigger-pipeline created
```

这里我也简单介绍一下这个 Pipeline。

复制代码

```yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: github-trigger-pipeline
spec:
  workspaces:
    - name: pipeline-pvc
    ......
  params:
    - name: subdirectory # 为每一个 Pipeline 配置一个 workspace，防止并发错误
      type: string
      default: ""
    - name: git_url
    ......
  tasks:
    - name: clone
      taskRef:
        name: git-clone
      workspaces:
        - name: output
          workspace: pipeline-pvc
        - name: ssh-directory
          workspace: git-credentials
      params:
        - name: subdirectory
          value: $(params.subdirectory)
        - name: url
          value: $(params.git_url)
    - name: build-and-push-frontend
      taskRef:
```

```
      name: docker-socket
    runAfter:
      - clone
    workspaces:
      - name: source
        workspace: pipeline-pvc
    params:
      - name: image
        value: "frontend"
      ......
  - name: build-and-push-backend
    taskRef:
      name: docker-socket
    runAfter:
      - clone
    workspaces:
      - name: source
        workspace: pipeline-pvc
    params:
      - name: image
        value: "backend"
      ......
```


首先，spec.workspaces 定义了一个工作空间。还记得我们提到的每一个 Task 都会在独立的 Pod 中运行吗？那么不同的 Task 如何共享上下文呢？答案就是 workspaces。实际上它是一个 PVC 持久化卷，这个 PVC 将会在 Pod 之间复用，这就让下游 Task 可以读取到上游 Task 写入的数据（比如克隆的代码）。

spce.params 定义了 Pipeline 的参数，参数的传递顺序是：PipelineRun->Pipeline->Task。

spce.tasks 定义了 Pipeline 引用的 Task，例如在这里分别引用了 git-clone 和 docker-socket 两个 Task，并且都指定了同一个 workspaces pipeline-pvc，然后指定了 params 向 Task 传递了参数值。

在 build-and-push-frontend 和 build-and-push-backend Task 中，都指定了 runAfter 字段，它的含义是等待 clone Task 执行完毕后再运行。

所以，Pipeline 对 Task 的引用就形成了一个有向无环图（DAG），在这个 Pipeline 中，首先会检出源码，然后以并行的方式同时构建前后端镜像。

创建 EventListener

创建完 Pipeline 之后，工作流实际上就已经定义好了。但是我们并不希望手动来运行它，我们希望通过 GitHub 来自动触发它。所以，接下来需要创建 EventListener 来获得一个能够监听外部事件的服务。


```
$ kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhan
eventlistener.triggers.tekton.dev/github-listener created
```

EventListener 的具体作用是：接收来自 GitHub 的 Webhook 调用，并将 Webhook 的参数和 TriggerTemplate 定义的参数对应起来，以便将参数值从 Webhook 一直传递到 PipelineRun。

暴露 EventListener

在 EventListener 创建完成之后，Tekton 将会拉起一个 Deployment 用来处理 Webhook 请求，你可以通过 kubectl get deployment 命令来查看。

```
$ kubectl get deployment
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
el-github-listener   1/1     1            1           22m
```

同时，Tekton 也会为 el-github-listener Deployment 创建 Service，以便接受来自外部的 HTTP 请求。

```
$ kubectl get service
NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
el-github-listener   ClusterIP      172.16.253.54   <none>        8080/TCP,90
```

为了能够让 GitHub 将事件推送到 Tekton 中，我们需要暴露 el-github-listener Service。我使用了 Ingress-Nginx 来对外暴露它。

```
$ kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhan
ingress.networking.k8s.io/ingress-resource created
```

这个 Ingress 的内容比较简单，具体如下。


```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-resource
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
    - http:
        paths:
          - path: /hooks
            pathType: Exact
            backend:
              service:
                name: el-github-listener
                port:
                  number: 8080
```

在 Ingress 策略中，我们没有使用 host 定义域名，而使用了 path 来匹配路径。这样，Tekton 接收外部 Webhook 的入口也就是 Ingress-Nginx 的负载均衡器 IP 地址了，具体的地址为 http://43.135.82.249/hooks。

创建 TriggerTemplate

不过 EventListener 并不能独立工作，它还需要一个助手，那就是 TriggerTemplate。TriggerTemplate 是真正控制 Pipeline 启动的组件，它负责创建 PipelineRun。

我们可以通过下面的命令来创建 TriggerTemplate。


```
$ kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhan
triggertemplate.triggers.tekton.dev/github-template created
```

创建 Service Account 和 PVC

下一步，由于触发器并没有具体的执行用户，所以我们还需要为触发器配置权限，也就是创建 Service Account。同时，我们也可以一并创建用于共享 Task 之间上下文的 PVC 。

复制代码

```
$ kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhan
serviceaccount/tekton-build-sa created
clusterrolebinding.rbac.authorization.k8s.io/tekton-clusterrole-binding created
persistentvolumeclaim/pipeline-pvc created
role.rbac.authorization.k8s.io/tekton-triggers-github-minimal created
rolebinding.rbac.authorization.k8s.io/tekton-triggers-github-binding created
clusterrole.rbac.authorization.k8s.io/tekton-triggers-github-clusterrole create
clusterrolebinding.rbac.authorization.k8s.io/tekton-triggers-github-clusterbind
```

设置 Secret

最后，我们还需要为 Tekton 提供一些凭据信息，例如 Docker Hub Token、GitHub Webhook Secret 以及用于检出私有仓库的私钥信息。

将下面的内容保存为 secret.yaml，并修改相应的内容。

复制代码

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: registry-auth
  annotations:
    tekton.dev/docker-0: https://docker.io
type: kubernetes.io/basic-auth
stringData:
  username: "" # docker username
  password: "" # docker hub token

---
# github webhook token secret
apiVersion: v1
kind: Secret
metadata:
  name: github-secret
type: Opaque
stringData:
  secretToken: "webhooksecret"
---
apiVersion: v1
kind: Secret
metadata:
```

```
  name: git-credentials
data:
  id_rsa: LS0tLS......
  known_hosts: Z2l0aHViLm......
  config: SG9zd......
```


解释一下，你主要需要修改的是下面这几个字段。

- 将 stringData.username 替换为你的 Docker Hub 的用户名。
- 将 stringData.password 替换为你的 Docker Hub Token，如果你还没有创建好，可以在上一节课找到相应内容。
- 将 data.id_rsa 替换为你本地 ~/.ssh/id_rsa 文件的 base64 编码内容，这将会为 Tekton 提供检出私有仓库的权限，你可以使用 `$ cat ~/.ssh/id_rsa | base64` 命令来获取。
- 将 data.known_hosts 替换为你本地 ~/.ssh/known_hosts 文件的 base64 编码内容，你可以通过 `$ cat ~/.ssh/known_hosts | grep "github" | base64` 命令来获取。
- 将 data.config 替换为你本地 ~/.ssh/config 文件的 base64 编码内容，你可以通过 `$ cat ~/.ssh/config | base64` 命令来获取。

然后，运行 kuebctl apply，同时将这 3 个 Secret 应用到集群内。

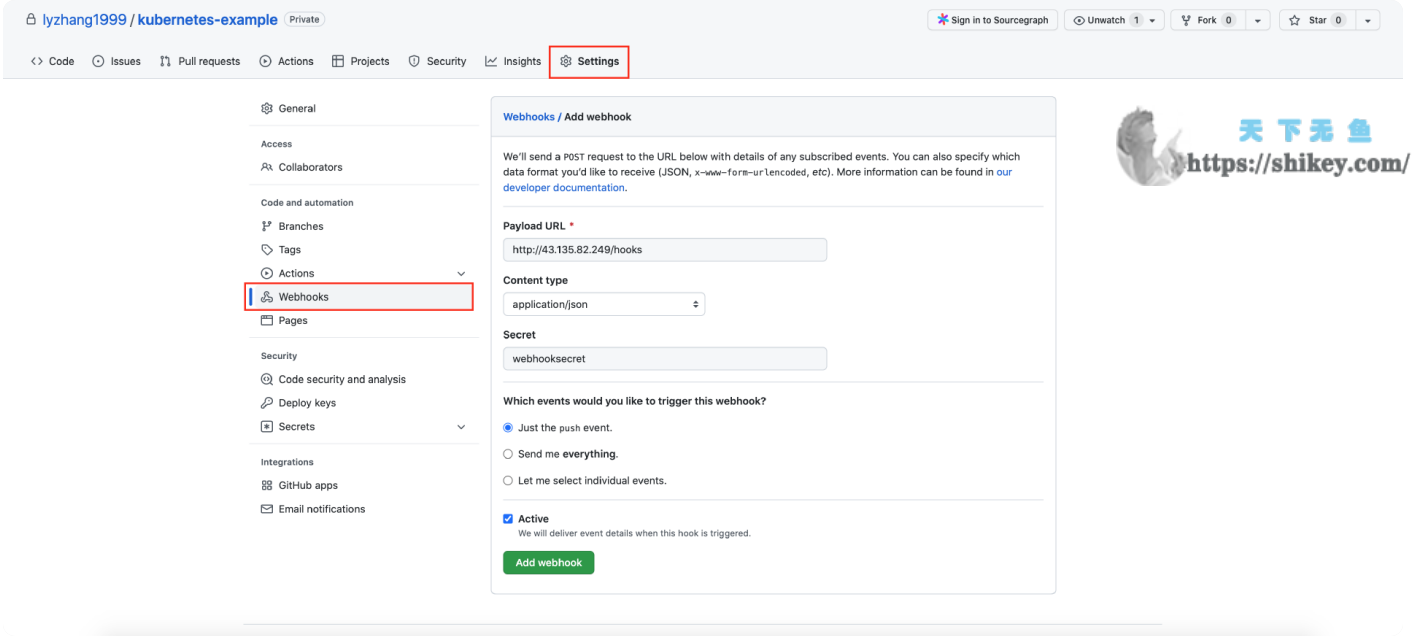
复制代码

```
$ kubectl apply -f secret.yaml
secret/registry-auth created
secret/github-secret created
secret/git-credentials created
```

创建 GitHub Webhook

到这里，Tekton 的配置就已经完成了。接下来还剩最后一步：创建 GitHub Webhook。

打开你在 GitHub 创建的 kubernetes-example 仓库，进入“Settings”页面，点击左侧的“Webhooks”菜单，在右侧的页面中按照下图进行配置。



输入 Webhook 地址，也就是“ Ingress-Nginx 网关的负载均衡器地址 + hooks 路径”，并且将“Content type”配置为“application/json”。

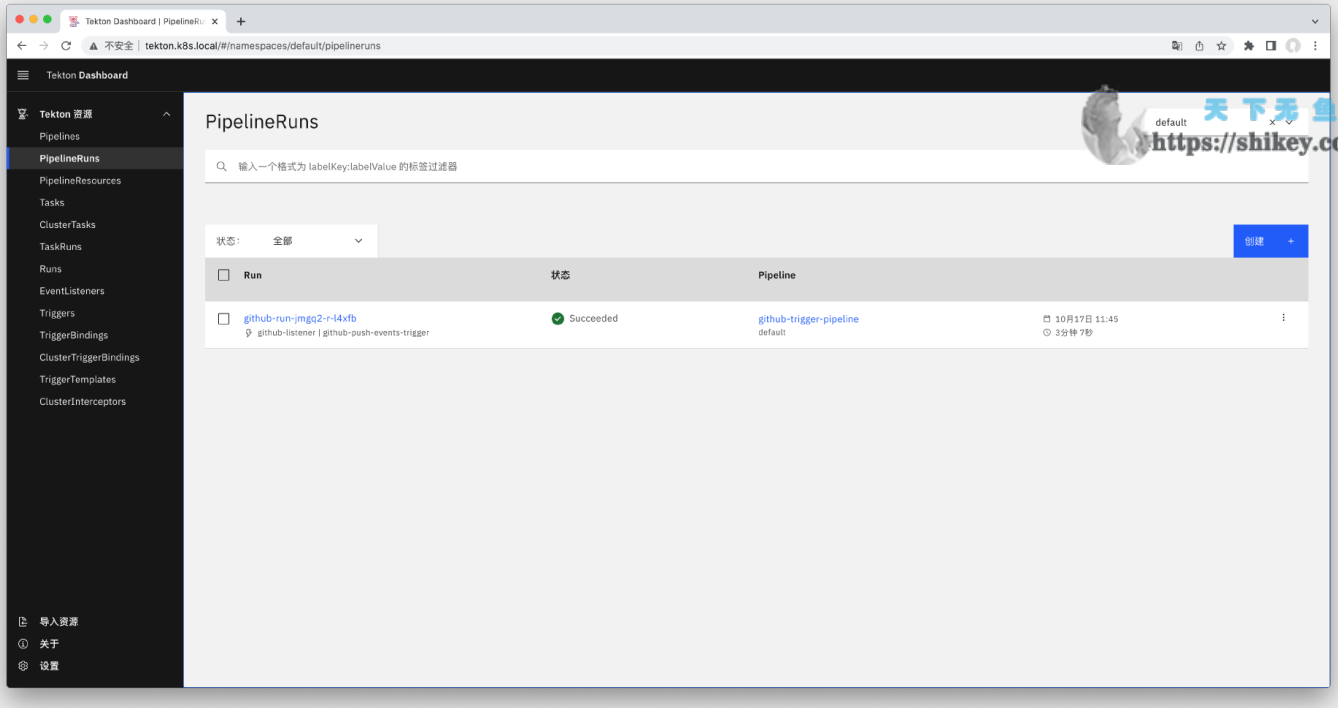
点击“Add webhook”创建。

触发 Pipeline

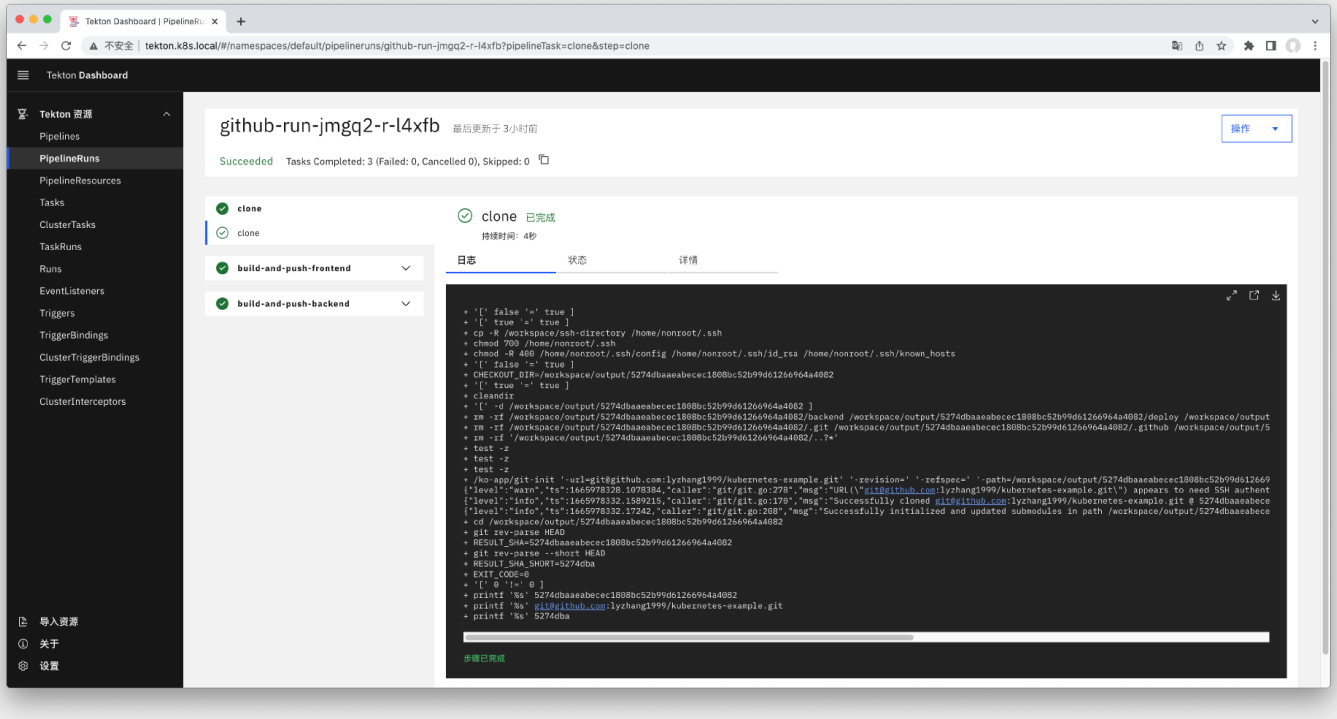
到这里，所有的准备工作就已经完成了。现在我们向仓库提交一个空的 commit 来触发 Pipeline。

```
$ git commit --allow-empty -m "Trigger Build"
[main 79ca67e] Trigger Build
$ git push origin main
```

完成推送后，打开 http://tekton.k8s.local/ 进入 Tekton 控制台，点击左侧的“PipelineRun”，你会看到刚才触发的 Pipeline。



点击 PipelineRun 的名称进入详情页面，可以看到 Pipeline 包含 3 个 Task 及其输出的日志信息。



到这里，我们就成功地将 GitHub 和 Tekton Pipeline 结合，并通过 GitHub Webhook 触发了 Pipeline，实现了镜像的自动构建和推送，和上一节课实现的效果一样，每次有新的提交时都会触发构建，**并且每个 commit id 对应一个镜像版本**。

总结

总结一下。在这节课，我们介绍了如何通过 Tekton Pipeline 来自动构建和推送镜像。

Tekton 的概念繁多，并且比较抽象，完全掌握这些概念有一定难度。为了方便理解和记忆，你可以把 Tekton 的概念和 GitHub Action 做类比，例如，Task 有点像 GitHub Action 的插件，git-clone Task 实际上和我们在 GitHub Action 引用的 actions/checkout@v3 插件功能是类似的。

其次，Tekton Pipeline 和 GitHub Action Workflow 也比较类似，Tekton Pipeline 通过引用和组合不同的 Task 形成了一个流水线。而 GitHub Action Workflow 则是通过引用并组合插件来完成一个工作流。

相比较而言，它们之间最大的差异还是在触发器上。GitHub Action Workflow 的触发器内置到了工作流中，只需要在工作流声明什么时候可以被触发即可。而 Tekton 则需要借助 EventListener 和 TriggerTemplate 这两个组件来接收外部 Webhook 事件并触发 Pipeline。所以 Tekton 复杂的配置也主要集中在这两块。

Tekton 虽然配置相对复杂，但它是一次性配置，如果你已有 Kubernetes 集群，那么使用 Tekton 来构建镜像是一个不错的选择，这种方案使得镜像构建成本几乎为零，并且扩展性强。

在前面几节课，我们都是使用 Docker Hub 作为镜像存储仓库的。但实际上，它同样也需要收费，**在镜像存储方面，我们同样也可以使用自托管方案来进一步降低成本。**

在下一节课，我们会详细介绍这部分的内容。

课后题

最后，给你留一道思考题吧。

在一个 Tekton Task 中，Step 是在同一个 Pod 不同的容器里运行的，我们知道 Pod 的容器启动是没有顺序的，那么 Tekton 是如何控制 Step 的运行顺序的呢？

提示：你可以在 Pipeline 运行时，查看 Task 拉起的 Pod Manifest 查找线索（`kubectl get pods $POD_NAME -o yaml`）。

欢迎你给我留言交流讨论，你也可以把这节课分享给更多的朋友一起阅读。我们下节课见。

分享给需要的人，Ta购买本课程，你将得 18 元

生成海报并分享

赞 1　提建议

上一篇　17丨自动构建：如何使用 GitLab CI 构建镜像？

下一篇　19丨自托管构建：如何使用 Harbor 搭建企业级镜像仓库？

精选留言 (8)

写留言

李多
2023-01-26 来自山东

原文：
kubectl apply -f https://storage.googleapis.com/tekton-releases/dashboard/latest/tekton-dashboard-release.yaml
貌似已经 404 无法访问了。

新的 dashboard yaml 在这个路径下：
kubectl apply --filename https://storage.googleapis.com/tekton-releases/dashboard/latest/release.yaml

参照：https://github.com/tektoncd/dashboard

共 1 条评论 >　1

ghostwritten
2023-01-30 来自北京

这篇文章干货很赞。
第二页：
10. 暴露 EventListener
kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhang1999/gitops/m

ain/ci/18/tekton/ingress/github-listener.yaml

11. 创建 TriggerTemplate
kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhang1999/gitops/main/ci/18/tekton/trigger/github-trigger-template.yaml

12. 创建 Service Account 和 PVC
kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhang1999/gitops/main/ci/18/tekton/other/service-account.yaml

13. 设置 Secret
cat ~/.ssh/id_rsa | base64
cat ~/.ssh/known_hosts | grep "github" | base64
cat ~/.ssh/config | base64
多行转一行工具：https://www.lmcjl.com/index/keyword/pinjie.html
kubectl apply -f secret.yaml

14. 创建 GitHub Webhook
在 GitHub 创建的 `kubernetes-example` 仓库，进入“Settings”页面，点击左侧的“Webhooks”菜单，输入 `Webhook` 地址，也就是“ http://外网ip/hooks”，并且将“Content type”配置为“`application/json`”。点击“`Add webhook`”创建。

15. 触发 Pipeline
git commit --allow-empty -m "Trigger Build"
git push origin main


ghostwritten
2023-01-30 来自北京

跟着步骤已全部实现，学到了不少东西。
第一页：
1. 腾讯云创建k8级群：申请标准集群、创建集群网络、设置配额、外网访问、本地kubectl访问集群。
2. 安装 Tekton Operator
kubectl apply -f https://storage.googleapis.com/tekton-releases/pipeline/latest/release.yaml
等待：kubectl wait --for=condition=Ready pods --all -n tekton-pipelines --timeout=300s

3. 安装 Tekton Trigger 和 Tekton Interceptors 组件
kubectl apply -f https://storage.googleapis.com/tekton-releases/triggers/latest/release.yaml
kubectl apply -f https://storage.googleapis.com/tekton-releases/triggers/latest/interceptors.yaml

kubectl wait --for=condition=Ready pods --all -n tekton-pipelines --timeout=300s

4. 安装 Ingress-Nginx
kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.4.0/deploy/static/provider/cloud/deploy.yaml
kubectl wait --for=condition=AVAILABLE deployment/ingress-nginx-controller --all -n ingress-nginx

5. 暴露 Tekton Dashboard
kubectl apply -f tekton-dashboard.yaml
获取外网 IP ： kubectl get services --namespace ingress-nginx ingress-nginx-controller --output jsonpath='{.status.loadBalancer.ingress[0].ip}'
配置43.135.82.249 tekton.k8s.local 通过域名 http://tekton.k8s.local 来访问Tekton Dashboard

6. 创建 Task：检出代码
kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhang1999/gitops/main/ci/18/tekton/task/git-clone.yaml

7. 创建 Task：构建和推送镜像
kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhang1999/gitops/main/ci/18/tekton/task/docker-build.yaml

8. 创建 Pipeline
kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhang1999/gitops/main/ci/18/tekton/pipeline/pipeline.yaml

9. 创建 EventListener
kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhang1999/gitops/main/ci/18/tekton/trigger/github-event-listener.yaml


Amos
2023-01-21 来自江苏

创建 Task和创建 Pipeline两步中各有两个是spec，而不是spce吧？


Amos
2023-01-21 来自江苏

tekton是否支持图形化拖拽的方式编辑task、piplinerun呢？

Waylon
2023-01-20 来自北京

容器的entrypoint启动进程
监控到/tekton/downward/ready文件的创建, 并等待文件内容的写入
执行git-init子进程, 从 git 仓库克隆源码
创建/tekton/tools/0文件

所有的 Step 都是被 /tekton/tools/entrypoints 封装起来执行的。
-wait_file 指定一个文件，通过监听文件句柄，在探测到文件存在时执行被封装的 Step 任务。

-post_file 指定一个文件，在Step任务完成后创建这个文件。
通过文件序列 /tekton/tools/${index} 来对 Step 进行排序。

一步
2023-01-18 来自广东

这里使用云厂商的k8s 集群是不是为了获取公网IP, 方便可以 github webhook 联动？

如果是这样 带公网ip 的云虚拟机是不是也可以完成？

共 2 条评论 >

无名无姓
2023-01-18 来自北京

这个顺序是必须的么？

> 作者回复: 什么顺序呢？