

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

Kubernetes 原生 CI/CD 工具 Tekton

前言：
课程名称：Kubernetes 原生 CI/CD 工具 Tekton

实验环境：
本章节 Kubernetes 集群环境如下：

| 角色 | IP | 主机名 | 组件 | 硬件 |
|------|----------------|--------------|--|--|
| 控制节点 | 192.168.128.11 | k8s-master01 | apiserver controller-manager scheduler etcd containerd | CPU：2vCPU 硬盘：100G 内存：3GB 开启虚拟化 |
| 工作节点 | 192.168.128.21 | k8s-node01 | kubelet kube-proxy containerd calico coredns | CPU：6vCPU 硬盘：100G 内存：12GB 开启虚拟化 |

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：
微信号：ZhangYanFeng0429
二维码：



版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

1、认识 Tekton

1.1、什么是 Tekton?

Tekton 是一个功能强大且灵活的 Kubernetes 原生开源框架，是由谷歌开源的，功能强大且灵活，开源社区也正在快速的迭代和发展壮大，主要用于创建持续集成和持续部署、持续交付（CI/CD）系统。它支持多云/多集群下进行搭建、测试和部署，可实现滚动部署、蓝/绿部署、金丝雀部署或 GitOps 工作流等高级部署。另外，基于 kubernetes CRD 定义的 pipeline 流水线也是 Tekton 最重要的特征。

扩展：什么是 CRD?

CRD 全称是 Custom Resource Definition，其特点如下：

- 1、CRD 本身是 Kubernetes 的一种资源，允许用户自定义新的资源类型。
- 2、除了 CRD 还需要用户提供一个 Controller，以实现自己的逻辑。
- 3、CRD 允许用户基于已有的 Kubernetes 资源，例如 Deployment、Configmap 等，拓展集群能力。

CRD 本身是一种 Kubernetes 内置的资源类型，即自定义资源的定义，用于描述用户定义的资源是什么样子。

CRD 的相关概念：

- 1、从 Kubernetes 的用户角度来看，所有东西都叫资源 Resource，就是 Yaml 里的字段 Kind 的内容，例如 Service、Deployment 等。

- 2、除了常见内置资源之外，Kubernetes 允许用户自定义资源 Custom Resource，而 CRD 表示自定义资源的定义。

1.2、使用 Tekton 的好处

- 可定制的：Tekton 实体是完全可定制的，从而具有高度的灵活性。平台工程师可以定义非常详细的构建基目录，以供开发人员在各种情况下使用。

- 可重复使用的：Tekton 实体是完全可移植的，因此一旦定义，组织内的任何人都可以使用给定的管道并重用其构造块。这使开发人员可以快速构建复杂的管道，而无需“重新发明轮子”。

- 可扩展的：Tekton Catalog 是 Tekton 社区驱动的存储库。您可以使用 Tekton 目录中的预制组件快速创建新的并展开现有管道。

- 标准化：Tekton 在您的 kubernetes 集群上作为扩展安装并运行，并使用完善的 kubernetes 资源模型。Tekton 工作负载在 kubernetes 容器中执行。

- 缩放性：为了增加工作负载容量，您可以简单地将节点添加到集群。Tekton 与您的群集进行缩放，无需重新定义您的资源分配或对管道的任何其它修改。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

1.3、Tekton 中的组件

- Tekton Pipelines: Tekton 的基础, 定义了一组 CRD, 用于定义 pipeline。
- Tekton Triggers: 允许基于 event 实例化 pipeline。比如: git 的 pr 请求。
- Tekton Cli: 提供命令行工具和 Tekton 交互。
- Tekton Dashboard: 图形化界面展示 pipeline 信息。
- Tekton Catalog: 高质量的、社区贡献的 pipeline 仓库。
- Tekton Hub: 图形化访问 Tekton Catalog。
- Tekton Operator: 在 kubernetes 上安装、移除、更新 Tekton 组件的项目。

2、安装 Tekton

Tekton 官方 Github 地址:

<https://github.com/tektoncd/pipeline/releases>

Tekton 官方参考手册:

<https://tekton.dev/docs/>

Tekton dashboard 地址:

<https://github.com/tektoncd/dashboard/releases>

Tekton 与 kubernetes 版本对照表:

| Required Kubernetes Version |
|--|
| • Starting from the v0.24.x release of Tekton: Kubernetes version 1.18 or later |
| • Starting from the v0.27.x release of Tekton: Kubernetes version 1.19 or later |
| • Starting from the v0.30.x release of Tekton: Kubernetes version 1.20 or later |
| • Starting from the v0.33.x release of Tekton: Kubernetes version 1.21 or later |
| • Starting from the v0.39.x release of Tekton: Kubernetes version 1.22 or later |
| • Starting from the v0.41.x release of Tekton: Kubernetes version 1.23 or later |

我们是 kubernetes 1.23 或以上版本, 这里我们需要安装 Tekton v0.41.x 及以上版本。

目前最新版本 v0.43.0, 在 dockerhub 找不到 “sidecarlogresults” 镜像文件。

Tekton v0.42.0 版本没有使用到 “sidecarlogresults” 镜像文件。这里我们部署一个 Tekton v0.42.0 版本。

(1) 下载 yaml 文件

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

| | | |
|----------------------|--------|--------------|
| ▼ Assets 4 | | |
| release.notags.yaml | 106 KB | Nov 23, 2022 |
| release.yaml | 106 KB | Nov 23, 2022 |
| Source code (zip) | | Nov 23, 2022 |
| Source code (tar.gz) | | Nov 23, 2022 |

也可以执行如下命令下载：

```
wget
https://storage.googleapis.com/tekton-releases/pipeline/previous/v0.4
2.0/release.yaml
```

(2) 查看需要修改的镜像

过滤查看需要修改的镜像

```
[root@k8s-master01 ~]# cat release.yaml | grep "pipeline/cmd/"
```

```
[root@k8s-master01 ~]# cat release.yaml | grep "pipeline/cmd/"
image: gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/controller:v0.42.0@sha256:1fa50403c071b760984e23e26d0e60d2f7e470208ef2eb73581ec556bacdad95
"image: gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/kubeconfigwriter:v0.42.0@sha256:672d416c97c15d20102749c6e86195683d037bd6c8
787560c9c87ade8b610071", "git-image", "gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/git-init:v0.42.0@sha256:211b0822659b2030a9e12b1c0b47faab2187a63a24ed9d21044
520f967674130", "entrypoint-image", "gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/entrypoint:v0.42.0@sha256:77e43d0fc9f7e7bdfa31dc16082b08dace95ce81c91a86c08df
a2f947212ce72", "nop-image", "gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/nop:v0.42.0@sha256:bd1fcc45d40a8ef1621789856caa2f54d7a884f19af921105feafae0131648c5"
, "imagedigest-exporter-image", "gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/imagedigestexporter:v0.42.0@sha256:370d5a0e39577f784f1376fac082230b9a44950c01fe2
190692a8a5a810adc6", "pr-image", "gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/pullrequest-init:v0.42.0@sha256:e00d578d40d57a5124bee5107cb3358763874588a7fe2522
ebc7bb979280d06e", "workingdirinit-image", "gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/workingdirinit:v0.42.0@sha256:60a39c629448ac2845c4781513ef44c2f2fbc6e
b321d70a016002b5fa7b2379",
image: gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/resolvers:v0.42.0@sha256:aa7d21d45f0b1c411823d6a943e668c820f9cf52f1549d188ed89e992f6e0
image: gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/webhook:v0.42.0@sha256:90989eeb6e0ba9c481bfaba3b1bcc70725baa50404c8f6ce9d22cc601e63dc
[root@k8s-master01 ~]#
```

需要修改的镜像文件如下（目前这些镜像再 dockerhub 上可以找到，并且一直在更新，只有最新版本的 tekton 需要的“sidecarlogresults”镜像文件目前 dockerhub 还没有）：

```
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/controller
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/resolvers
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/webhook
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/kubeconfigwriter
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/git-init
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/entrypoint
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/nop
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/imagedigestexporter
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/pullrequest-init
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/workingdirinit
```

(3) 修改 release.yaml 文件

写 shell 脚本进行批量替换

```
[root@k8s-master01 ~]# vi image.txt
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/controller
r
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/resolvers
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/webhook
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/kubeconfigwriter
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/git-init
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/entrypoint
t
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/nop
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/imagdigestexporter
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/pullrequest-init
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/workingdirinit

[root@k8s-master01 ~]# vi sed-image.sh
#!/bin/bash
for image_name in `cat image.txt | awk -F '/' '{print $7}'`
do
    sed -i
"s#gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/${image_name}#gcritokton/pipeline-${image_name}#g" release.yaml
done
```

简单说明：

这里我读取 image.txt，过滤出镜像名称。Dockerhub 上面的镜像格式都是 pipeline-镜像名称，然后我们替换即可。

```
# 替换
[root@k8s-master01 ~]# sh sed-image.sh
[root@k8s-master01 ~]# cat release.yaml | grep
"gcritokton/pipeline"

[root@k8s-master01 ~]# cat release.yaml | grep "gcritokton/pipeline"
image: gcritokton/pipeline-controller:v0.42.0@sha256:1fa50403c071b768984e23e26d0e68d2f7e470284ef2eb73581ec556bacdad95
"-kubeconfig-writer-image", "gcritokton/pipeline-kubeconfigwriter:v0.42.0@sha256:672df16c97c15d20102749c0e86195683d037bd6c8787560c9c07ade8b610071", "-git-image", "gcritokton/pipeline-git-init:v0.42.0@sha256:21b0822659b2030a9e12b1c0b47faab2187a63a24ed9d21044520f967674138", "-entrypoint-image", "gcritokton/pipeline-entrypoint:v0.42.0@sha256:77c73d0f097f7bdf41d1c10802b00ace05c03091a0c006f2f507212c72", "-nop-image", "gcritokton/pipeline-nop:v0.42.0@sha256:bd1fcc54d0baef1621789856ca2f5007a884f19af921105fcafa013164865", "-imagdigest-exporter-image", "gcritokton/pipeline-imagdigestexporter:v0.42.0@sha256:370d5a0e39577f784f1376fac0822218b9a40980c01fc2198692a0a5a010adcc", "-pr-image", "gcritokton/pipeline-pullrequest-init:v0.42.0@sha256:e00d578d40d57a5124bee5107cb3358763874588a7fa2522ebc7bb979280d06e", "-workingdirinit-image", "gcritokton/pipeline-workingdirinit:v0.42.0@sha256:68a39c629448ac2845c4781513ef44c2f2fbc6b6b321d70a016802b5fa7b2379".
image: gcritokton/pipeline-resolvers:v0.42.0@sha256:aa77d21d45f0bc1c411823d6a943e668c820f9cf52f1549d188ed89e992f6e0
image: gcritokton/pipeline-webhook:v0.42.0@sha256:90989eeb6e0ba9c481b1faba3b01bcc70725baa50484c8f6ce9d22cc601e63dc
[root@k8s-master01 ~]#
```

```
# 然后进入 release.yaml 将 sha256 这些都删除掉，最后修改如下：
[root@k8s-master01 ~]# cat release.yaml | grep
"gcritokton/pipeline"
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
[root@k8s-master01 ~]# cat release.yaml | grep "gcr.io/tekton/pipeline"
image: gcr.io/tekton/pipeline-controller:v0.42.0
  - "kubefig-writer-image", "gcr.io/tekton/pipeline-kubefig-writer:v0.42.0", "-git-image", "gcr.io/tekton/pipeline-git-init:v0.42.0", "-entrypoint-image", "gcr.io/tekton/pipeline-entrypoint:v0.42.0", "-nop-image", "gcr.io/tekton/pipeline-nop:v0.42.0", "-imagedigest-exporter-image", "gcr.io/tekton/pipeline-imagedigestexporter:v0.42.0", "-pr-image", "gcr.io/tekton/pipeline-pullrequest-init:v0.42.0", "-workingdir-init-image", "gcr.io/tekton/pipeline-workingdir-init:v0.42.0",
image: gcr.io/tekton/pipeline-resolvers:v0.42.0
image: gcr.io/tekton/pipeline-webhook:v0.42.0
```

(4) 创建并检查 Tekton

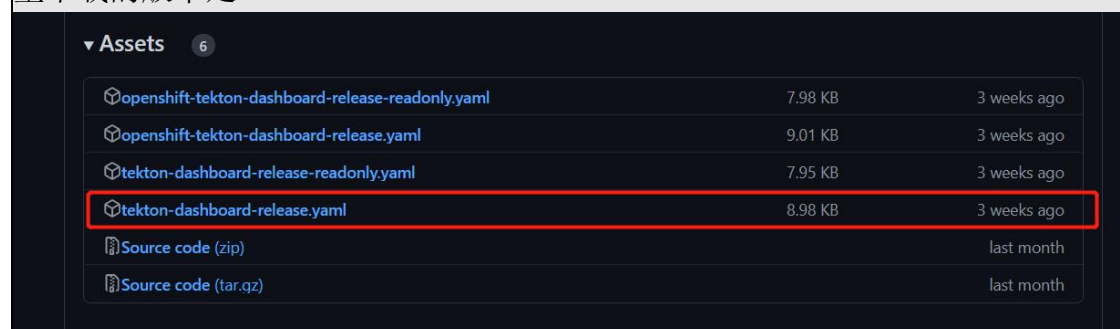
```
[root@k8s-master01 ~]# kubectl apply -f release.yaml

[root@k8s-master01 ~]# kubectl get pods -n tekton-pipelines
NAME                                     READY   STATUS    RESTARTS   AGE
tekton-pipelines-controller-8498b45899-m5htr   1/1     Running   0          77s
tekton-pipelines-webhook-696fcd55f-j8qdg       1/1     Running   0          77s
```

(5) 安装 dashboard（可选）

下载 dashboard yaml 文件

访问“<https://github.com/tektoncd/dashboard/releases>”进行下载，这里下载的版本是 v0.30.0



或者使用如下方式下载：

wget

<https://github.com/tektoncd/dashboard/releases/download/v0.30.0/tekton-dashboard-release.yaml>

修改 tekton-dashboard-release.yaml 文件

```
[root@k8s-master01 ~]# ll tekton-dashboard-release.yaml
[root@k8s-master01 ~]# ll tekton-dashboard-release.yaml
-rw-r--r-- 1 root root 9194 Jan  7 05:04 tekton-dashboard-release.yaml

[root@k8s-master01 ~]# vi tekton-dashboard-release.yaml
修改镜像：
image:
registry.cn-hangzhou.aliyuncs.com/hb-chen/tektoncd-dashboard:v0.30.0
```

安装

```
[root@k8s-master01 ~]# kubectl apply -f
tekton-dashboard-release.yaml
[root@k8s-master01 ~]# kubectl get pods -n tekton-pipelines
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
[root@k8s-master01 ~]# kubectl get pods -n tekton-pipelines
```

| NAME | READY | STATUS | RESTARTS | AGE |
|--|-------|---------|----------|-----|
| tekton-dashboard-7ff9696dfd-mx5kh | 1/1 | Running | 0 | 37s |
| tekton-pipelines-controller-8498b45899-m5htr | 1/1 | Running | 0 | 17m |
| tekton-pipelines-webhook-696fdcd55f-j8qdg | 1/1 | Running | 0 | 17m |

修改 svc 为 nodeport 并访问测试

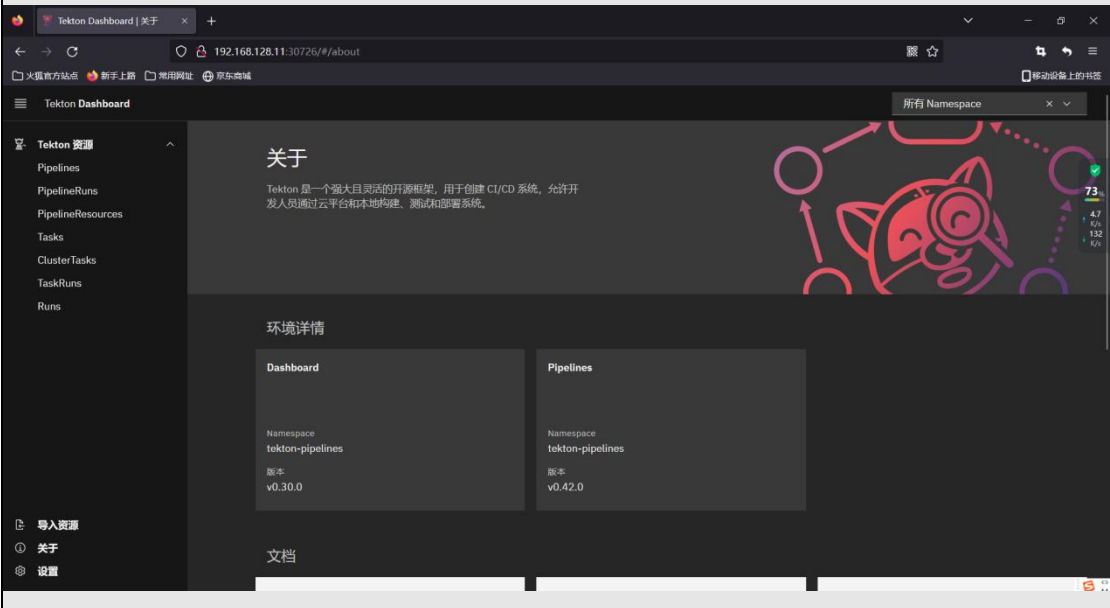
```
[root@k8s-master01 ~]# kubectl edit svc tekton-dashboard -n tekton-pipelines
type: NodePort

[root@k8s-master01 ~]# kubectl get svc -n tekton-pipelines
```

```
[root@k8s-master01 ~]# kubectl get svc -n tekton-pipelines
```

| NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|-----------------------------|-----------|---------------|-------------|------------------------------------|------|
| tekton-dashboard | NodePort | 10.10.64.22 | <none> | 9097:30726/TCP | 110s |
| tekton-pipelines-controller | ClusterIP | 10.10.44.155 | <none> | 9090/TCP,8008/TCP,8080/TCP | 18m |
| tekton-pipelines-webhook | ClusterIP | 10.10.149.253 | <none> | 9090/TCP,8008/TCP,443/TCP,8080/TCP | 18m |

浏览器输入: <http://192.168.128.11:30726>, 访问:

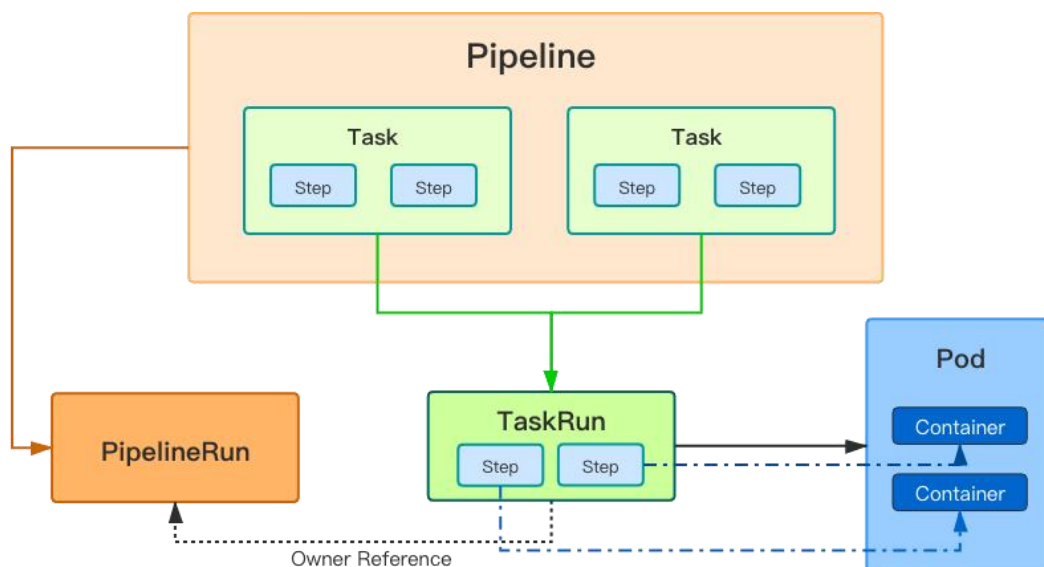


3、Tekton 的四个基本概念

Tekton 最主要的四个概念为: Task、TaskRun、Pipeline 以及 PipelineRun。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



- Task

Task 为构建任务，是 Tekton 中不可分割的最小单位，正如同 Pod 在 Kubernetes 中的概念一样。在 Task 中，可以有多个 Step，每个 Step 由一个 Container 来执行。

- Pipeline

Pipeline 由一个或多个 Task 组成。在 Pipeline 中，用户可以定义这些 Task 的执行顺序以及依赖关系。

- PipelineRun

PipelineRun 是 Pipeline 的实际执行产物，当用户定义好 Pipeline 后，可以通过创建 PipelineRun 的方式来执行流水线，并生成一条流水线记录。

- TaskRun

PipelineRun 被创建出来后，会对应 Pipeline 里面的 Task 创建各自的 TaskRun。一个 TaskRun 控制一个 Pod，Task 中的 Step 对应 Pod 中的 Container。当然，TaskRun 也可以单独被创建。

综上所述: Pipeline 由多个 Task 组成，每次执行对应生成一条 PipelineRun，其控制的 TaskRun 将创建实际运行的 Pod。

下面以一个简单例子来展示这些概念：

(1) 查看在 Kubernetes 集群中新增了哪些 Tekton 的 CRD

```
[root@k8s-master01 ~]# kubectl get crd | grep tekton
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
[root@k8s-master01 ~]# kubectl get crd | grep tekton
clustertasks.tekton.dev          2023-01-07T09:57:04Z
customruns.tekton.dev           2023-01-07T09:57:04Z
extensions.dashboard.tekton.dev 2023-01-07T10:13:52Z
pipelineresources.tekton.dev    2023-01-07T09:57:04Z
pipelineruns.tekton.dev         2023-01-07T09:57:04Z
pipelines.tekton.dev            2023-01-07T09:57:04Z
resolutionrequests.resolution.tekton.dev 2023-01-07T09:57:04Z
runs.tekton.dev                 2023-01-07T09:57:04Z
taskruns.tekton.dev             2023-01-07T09:57:04Z
tasks.tekton.dev                2023-01-07T09:57:04Z
```

(2) 创建一个最简单的 Task，里面仅有一个 Step。在一个 nginx 镜像中执行一条命令。

```
[root@k8s-master01 ~]# vi task-example.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: task-example
spec:
  steps:
  - name: nginx
    image: nginx:latest

# 创建 Task
[root@k8s-master01 ~]# kubectl apply -f task-example.yaml
task.tekton.dev/task-example created

# 查看创建的 Task
[root@k8s-master01 ~]# kubectl get task
NAME          AGE
task-example  4s
```

(3) 创建一个 Pipeline，里面引用刚才创建的 Task

```
[root@k8s-master01 ~]# vi pipeline-example.yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: pipeline-example
spec:
  tasks:
  - name: task-1
    taskRef:
      name: task-example
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

关键点说明：

```
spec:
  tasks:
  - name: task-1  #自定义名称
    taskRef:  #关联哪个 task
      name: task-example  #要关联的 task 名字
```

```
[root@k8s-master01 ~]# kubectl apply -f pipeline-example.yaml
pipeline.tekton.dev/pipeline-example created
```

```
[root@k8s-master01 ~]# kubectl get pipeline
NAME                AGE
pipeline-example    7s
```

(4)在 Pipeline 存在的前提下,就可以通过创建 PipelineRun 来运行 Pipeline。

```
[root@k8s-master01 ~]# vi pipelinerun-example.yaml
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: pipelinerun-example
spec:
  pipelineRef:
    name: pipeline-example
```

关键点说明：

```
spec:
  pipelineRef:  #关联哪个 pipeline
    name: pipeline-example  #要关联的 pipeline 名字
```

```
[root@k8s-master01 ~]# kubectl apply -f pipelinerun-example.yaml
pipelinerun.tekton.dev/pipelinerun-example created
```

```
[root@k8s-master01 ~]# kubectl get pipelinerun
```

```
[root@k8s-master01 ~]# kubectl get pipelinerun
NAME                SUCCEEDED REASON   STARTTIME   COMPLETIONTIME
pipelinerun-example Unknown    Running   32s
```

查看 pods

```
[root@k8s-master01 ~]# kubectl get pods
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

| [root@k8s-master01 ~]# kubectl get pods | | | | |
|---|-------|---------|----------|-----|
| NAME | READY | STATUS | RESTARTS | AGE |
| pipelinerun-example-task-1-pod | 1/1 | Running | 0 | 50s |
| Pod 的名字就是：pipelinerun 的名字-pipeline 中的 task 名字-pod | | | | |

这样就完成了一个最简单的 Tekton 流水线案例。每一个 PipelineRun 的创建，都会遵循 Pipeline 中的顺序规则去启动 Task 的 Pod。下面引入另外一个概念 PipelineResource 来完成一个稍微复杂的例子，也是 DevOps 中最常见的场景：从代码仓库拉取镜像、进行代码构建、并最终将构建好的镜像推往镜像仓库。

4、PipelineResource

PipelineResource 代表着一系列的资源，主要承担作为 Task 的输入或者输出的作用。它有以下几种类型：

- Git：代表一个 Git 仓库，包含了需要被构建的源代码。将 Git 资源作为 Task 的 Input，会自动 clone 此 Git 仓库。
- pullRequest：表示来自配置的 url（通常是一个 Git 仓库）的 pull request 事件。将 pull request 资源作为 Task 的 Input，将自动下载 pull request 相关元数据的文件，如 base/head commit、comments 以及 labels。
- Image：代表镜像仓库中的镜像，通常作为 Task 的 Output，用于生成镜像。
- Cluster：表示一个除了当前集群外的 Kubernetes 集群。可以使用 Cluster 资源在不同的集群上部署应用。
- Dstorage：表示 blob 存储，它包含一个对象或目录。将 Storage 资源作为 Task 的 Input 将自动下载存储内容，并允许 Task 执行操作。目前仅支持 GCS。
- cloud event：会在 TaskRun 执行完成后发送事件信息（包含整个 TaskRun）到指定的 URI 地址，在与第三方通信的时候十分有用。

以上为 Tekton 目前支持的六大 PipelineResource 类型，具体的配置及使用方法可以参考官方文档。

也可以参考下方官方的 github 地址：

<https://github.com/tektoncd/pipeline/blob/master/docs/resources.md>

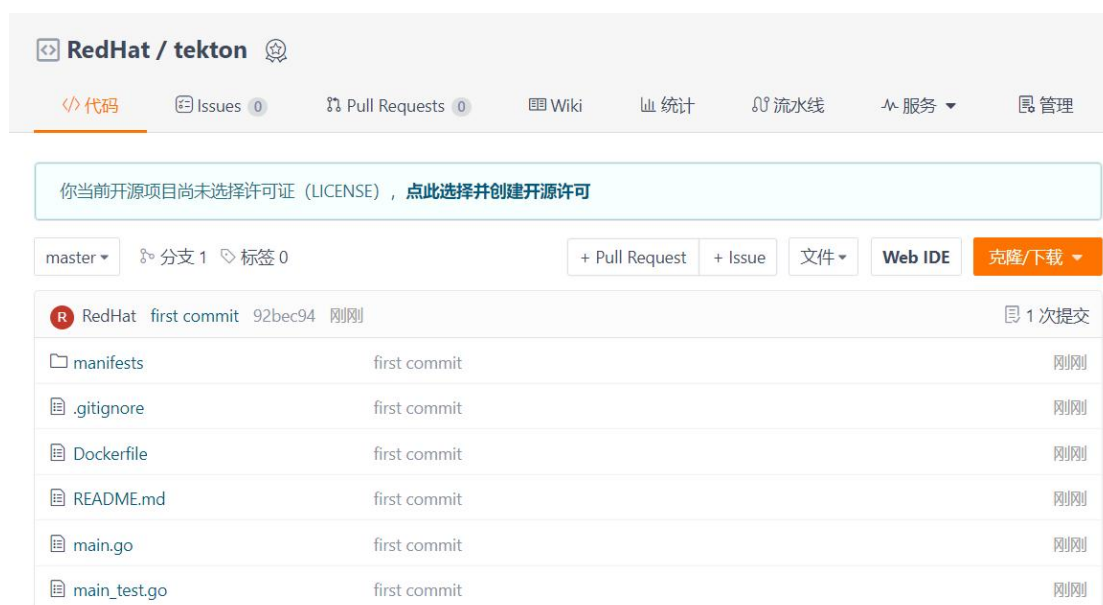
5、Tekton 实现 CI/CD 流水线

在这里我们使用一个简单的 Golang 应用。

源码包为：“项目源码包/tekton-demo.zip”，首先准备代码仓库，这里我使用的是码云。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



5.1、单元测试

首先第一个任务就是 Clone 应用程序代码，对代码进行测试，检查代码是否正确。

(1) 创建 task 任务

```
[root@k8s-master01 ~]# vi task-test.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: test
spec:
  resources:
    inputs:
      - name: repo
        type: git
  steps:
    - name: run-test
      image: golang:1.14-alpine
      workingDir: /workspace/repo
      command: ["go"]
      args: ["test"]

[root@k8s-master01 ~]# kubectl apply -f task-test.yaml
task.tekton.dev/test created
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
[root@k8s-master01 ~]# kubectl get task
NAME    AGE
test    25s
```

上面 yaml 文件，其中 resources 定义了我们的任务中定义的步骤所需的输入内容，这里我们的步骤需要 Clone 一个 Git 仓库作为 go test 命令的输入。

Tekton 内置了一种 git 资源类型，它会自动将代码仓库 Clone 到 /workspace/\$input_name 目录中，由于我们这里输入被命名成 repo，所以代码会被 Clone 到 /workspace/repo 目录下面。

然后下面的 steps 就是来定义执行运行测试命令的步骤，这里我们直接在代码的根目录中运行 go test 命令即可。

(2) 创建 PipelineResource

上面完成了一个 Task 任务，但是该任务并不会立即执行，我们必须创建一个 TaskRun 引用它并提供所有必需输入的数据才行。这里我们就需要将 git 代码库作为输入，我们必须先创建一个 PipelineResource 对象来定义输入信息

```
[root@k8s-master01 ~]# vi git-input.yaml
apiVersion: tekton.dev/v1alpha1
kind: PipelineResource
metadata:
  name: git-input
spec:
  type: git
  params:
    - name: url    #仓库地址
      value: https://gitee.com/redhat_feng/tekton.git
    - name: revision #分支
      value: master

[root@k8s-master01 ~]# kubectl apply -f git-input.yaml
pipelineresource.tekton.dev/git-input created

[root@k8s-master01 ~]# kubectl get PipelineResource
NAME          AGE
git-input     8s
```

(3) 创建 TaskRun 对象

```
[root@k8s-master01 ~]# vi taskrun-test.yaml
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: testrun
spec:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
taskRef:
  name: test
resources:
  inputs:
    - name: repo
      resourceRef:
        name: git-input

[root@k8s-master01 ~]# kubectl apply -f taskrun-test.yaml
taskrun.tekton.dev/testrun created

[root@k8s-master01 ~]# kubectl get taskrun
```

| NAME | SUCCEEDED | REASON | STARTTIME | COMPLETIONTIME |
|---------|-----------|-----------|-----------|----------------|
| testrun | True | Succeeded | 2m14s | 65s |

```
[root@k8s-master01 ~]# kubectl get pods
```

| NAME | READY | STATUS | RESTARTS | AGE |
|-------------|-------|-----------|----------|-------|
| testrun-pod | 0/2 | Completed | 0 | 2m29s |

(4) 查看容器的日志信息来了解任务的执行结果信息

```
[root@k8s-master01 ~]# kubectl logs testrun-pod --all-containers
```

```
[root@k8s-master01 ~]# kubectl logs testrun-pod --all-containers
2023/01/09 02:42:06 Entrypoint initialization
{"level":"info","ts":1673232192.3143857,"caller":"git/git.go:176","msg":"Successfully cloned https://gitee.com/redhat_feng/tekton.git @ 92bec94b9b749e5abfd5278377b5cbb8a845fa9 (grafted, HEAD, origin/master) in path /workspace/repo"}
{"level":"info","ts":1673232192.3329346,"caller":"git/git.go:215","msg":"Successfully initialized and updated submodules in path /workspace/repo"}
PASS
ok      _/workspace/repo      0.002s
[root@k8s-master01 ~]#
```

我们可以看到我们的测试已经通过了。

5.2、Docker Hub 配置

为了能够构建 Docker 镜像，一般来说我们需要使用 Docker 来进行，我们这里是容器，所以可以使用 Docker In Docker 模式，但是这种模式安全性不高，除了这种方式之外，我们还可以使用 Google 推出的 Kaniko 工具来进行构建，该工具可以在 Kubernetes 集群中构建 Docker 镜像而无需依赖 Docker 守护进程。

我们需要创建 Docker Hub 的登录凭证，方便后续将镜像推送到镜像仓库。登录凭证可以保存到 Kubernetes 的 Secret 资源对象中。

(1) 创建 Secret

```
[root@k8s-master01 ~]# vi dockerhub-secret.yaml
apiVersion: v1
kind: Secret
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
metadata:
  name: docker-auth
  annotations:
    tekton.dev/docker-0: https://index.docker.io/v1/
type: kubernetes.io/basic-auth
stringData:
  username: zhangyanfeng0429
  password: zhangyf0429

[root@k8s-master01 ~]# kubectl apply -f dockerhub-secret.yaml
secret/docker-auth created
```

我们这里在 Secret 对象中添加了一个 tekton.dev/docker-0 的 annotation，该注解信息是用来告诉 Tekton 这些认证信息所属的 Docker 镜像仓库。

然后创建一个 ServiceAccount 对象来使用上面的 docker-auth 这个 Secret 对象。

(2) 创建 sa

```
[root@k8s-master01 ~]# vi dockerhub-serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: docker-sa
secrets:
- name: docker-auth

[root@k8s-master01 ~]# kubectl apply -f
dockerhub-serviceaccount.yaml
serviceaccount/docker-sa created
```

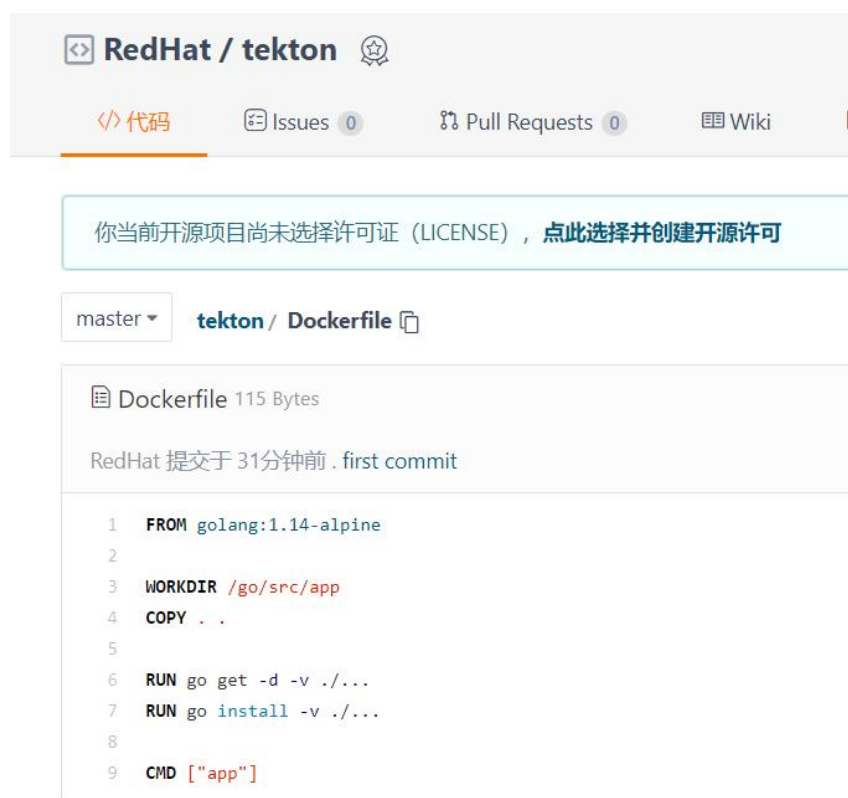
创建完成后，我们就可以在运行 Tekton 的任务或者流水线的时候使用上面的 docker-sa 这个 ServiceAccount 对象来进行 Docker Hub 的登录认证了。

5.3、创建镜像构建任务

现在我们创建一个 Task 任务来构建并推送 Docker 镜像，我们项目的源码根目录下面包含了一个 Dockerfile 文件，这个是构建镜像的基础文件，必须得有。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



(1) 创建 task 任务

```
[root@k8s-master01 ~]# vi task-build-push.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: build-and-push
spec:
  resources:
    inputs:
      - name: repo
        type: git
  steps:
    - name: build-and-push
      image: aiotceo/kaniko-executor:v1.6.0
      env:
        - name: DOCKER_CONFIG
          value: /tekton/home/.docker
      command:
        - /kaniko/executor
        - --dockerfile=Dockerfile
        - --context=/workspace/repo
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
- --destination=zhangyanfeng0429/tekton-test:latest

[root@k8s-master01 ~]# kubectl apply -f task-build-push.yaml
task.tekton.dev/build-and-push created

[root@k8s-master01 ~]# kubectl get task
NAME          AGE
build-and-push 3s
test          33m
```

这里我们同样将 git 作为输入，也只定义了一个名为 build-and-push 的步骤，Kaniko 会在同一个命令中构建并推送，所以不需要多个步骤了，执行的命令就是/kaniko/executor，通过--dockerfile 指定 Dockerfile 路径，--context 指定构建上下文，就是项目的根目录，然后--destination 参数指定最终我们的镜像名称。

(2) 创建 TaskRun

```
[root@k8s-master01 ~]# vi taskrun-build-push.yaml
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: build-and-push
spec:
  serviceAccountName: docker-sa
  taskRef:
    name: build-and-push
  resources:
    inputs:
      - name: repo
        resourceRef:
          name: git-input

[root@k8s-master01 ~]# kubectl apply -f taskrun-build-push.yaml
taskrun.tekton.dev/build-and-push created
```

注意：这里我们通过 serviceAccountName 属性指定了 Docker 认证信息的 ServiceAccount 对象，然后通过 taskRef 引用我们的任务。

(3) 查看 taskrun 和 pod 状态

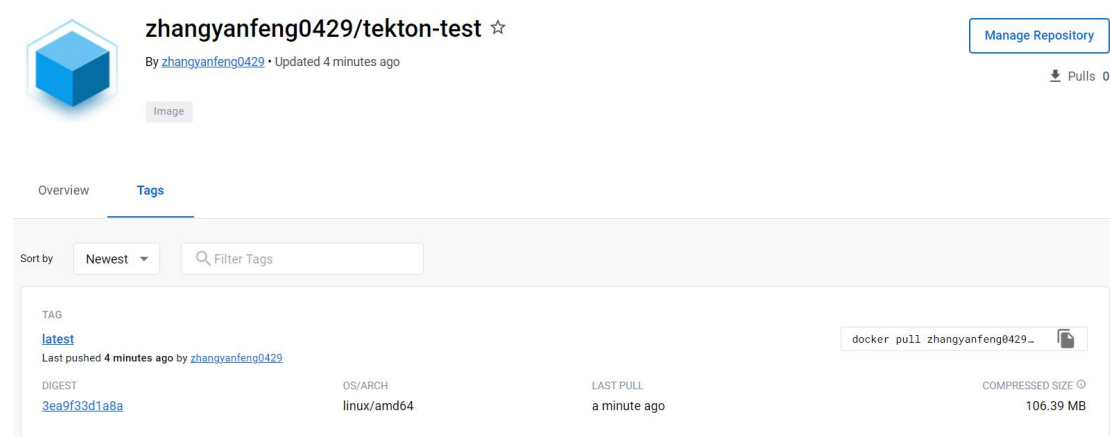
```
[root@k8s-master01 ~]# kubectl get taskrun build-and-push
[root@k8s-master01 ~]# kubectl get taskrun build-and-push
NAME          SUCCEEDED REASON      STARTTIME   COMPLETIONTIME
build-and-push True       Succeeded   92s         23s
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
[root@k8s-master01 ~]# kubectl get pods build-and-push-pod
NAME                READY   STATUS    RESTARTS   AGE
build-and-push-pod  0/2     Completed 0           2m
```

我们可以看到 TaskRun 任务已经执行成功了。我们可以在 Docker Hub 上找到我们的镜像了，如下：



5.4、创建业务容器

对于业务容器的创建，我们基于 kubectl 进行创建即可，Tekton 对于这一环节的支持还不太友好。Tekton 只能创建一些简单的 pod 资源，在生产环境不建议使用 Tekton 在这一环节。

(1) 创建 pod

```
[root@k8s-master01 ~]# vi tekton-test.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tekton-test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tekton-test
  template:
    metadata:
      labels:
        app: tekton-test
    spec:
      containers:
        - image: zhangyanfeng0429/tekton-test:latest
          imagePullPolicy: IfNotPresent
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
name: tekton-test

[root@k8s-master01 ~]# kubectl apply -f tekton-test.yaml
deployment.apps/tekton-test created
```


(2) 查看 pod

```
[root@k8s-master01 test]# kubectl get pods -l app=tekton-test
[root@k8s-master01 ~]# kubectl get pods -l app=tekton-test
NAME                                READY   STATUS    RESTARTS   AGE
tekton-test-678d94bdb7-q76w6        0/1     Completed 2 (31s ago) 33s

[root@k8s-master01 ~]# kubectl logs tekton-test-678d94bdb7-q76w6
Sum: 3
```

因为这个项目比较简单，我们就不需要创建 svc，我们可以看一下源代码：

master ▾ [tekton / main.go](#) 

 main.go 119 Bytes

RedHat 提交于 1小时前 · first commit

```
1 package main
2
3 import "fmt"
4
5 func sum(a, b int) int {
6     return a + b
7 }
8
9 func main() {
10     fmt.Println("Sum: ", sum(1, 2))
11 }
```

就是简单做了一些 1+2。执行完毕之后，pod 就会退出。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**