CI/CD for Yocto Project® Maintainers with Kubernetes and Tekton Pipelines

Trevor Gamblin, Wind River Systems
Tim Orling, Intel Corporation

Yocto Project Virtual Summit Europe, October 29-30, 2020
Introduction

• **Trevor:**
  • Userspace/Infrastructure Developer at Wind River

• **Tim:**
  • Yocto Project Architect at Intel

• **Both meta-python maintainers**
• **Very excited about k8s and Tekton**
Agenda

• State of Yocto CI/CD
• K8S and Tekton
• Pipelines!
• Shared State!
• Cool Tools!
• Future Work
Disclaimer

• Yocto experts, not k8s experts (suggestions welcome!)

• Focusing on single-node developer setup (for now)
State of Yocto CI/CD

The Yocto Autobuilder and Other Pursuits
The Yocto Autobuilder

• Based on the Buildbot framework
• Bare metal (could be containerized)
• Official Yocto Autobuilder has restricted access (and that’s OK)
• Incredibly useful - but maybe overkill for average developer workflow
Other Current Solutions

• Homegrown Autobuilder instances
• Jenkins (security issues!)
• TravisCI
• GitLabCI
• Others?
Kubernetes and Tekton

Containers All The Way Down
What is Kubernetes?

• Container orchestrator maintained by CNCF, built for resiliency and scalability

• Application-agnostic - use it to build and deploy what you want

• Supports various container runtimes

• https://kubernetes.io/
Why Kubernetes?

• It’s everywhere, highly-scalable, with lots of existing use cases, extensions, services

• Provides building blocks for the user to adapt, rather than a specific, tailored solution

• Yocto is doing more containerized builds – why not use k8s for that?
What is Tekton?

• CI/CD framework for Kubernetes

• Adds:
  - Tasks
  - Pipelines
  - TaskRuns and PipelineRuns
  - EventListeners
  - TriggerTemplates, TriggerBindings

• https://tekton.dev/
Why Tekton?

• Tekton is very customizable, relatively easy to understand

• More modern, designed around contemporary technologies

• Has a rich dashboard, CLI
Our Setup

• Single-node cluster
• HP Z640, E5-2630v4 x2, 64GB RAM
• Fedora 32 Server Edition
• kubeadm, not Minikube
End Goal – Pipelines on a Dashboard
How Do We Get There?

• https://github.com/threexc/yocto-tekton

• Setup instructions, example pipelines (we’ll get to those shortly)

• Already in use in the wild - created to help maintain the meta-python layer
A Pipeline for Poky
Concept

• Identify delta between master and master-next branches

• Get a list of corresponding recipes, build with bitbake

• Report results and any unmappable patches

• https://github.com/threexc/yocto-tekton/tree/main/poky
Pipeline Implementation Examples

```yaml
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: poky-setup-workspace
  namespace: tekton-pipelines
spec:
  steps:
  - name: create-workspace
    image: registry.hub.docker.com/threeeex/yocto-builder
    workingDir: /workspace
    script:
      #!/bin/bash -xe
      if [ ! -d poky ]; then
        git clone git://git.yoctoproject.org/poky
      fi
    volumeMounts:
    - name: build
      mountPath: /workspace

  - name: update-workspace
    image: registry.hub.docker.com/threeeex/yocto-builder
    workingDir: /workspace
    script:
      #!/bin/bash -xe
      (cd poky && git checkout master-next && git pull --rebase)
    volumeMounts:
    - name: build
      mountPath: /workspace

apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: poky-build-pipeline
  namespace: tekton-pipelines
spec:
  tasks:
  - name: poky-setup-workspace
    taskRef:
      name: poky-setup-workspace
      taskRef:
        - name: poky-build
          runAfter:
          - poky-setup-workspace
          taskRef:
            name: poky-build
```
A Pipeline for meta-python
Concept

• Same idea as the poky pipeline (identify recipes, build, report), for meta-python
• Makes maintaining meta-python way easier!
• Will eventually include containerized image tests (e.g. meta-python-ptest-image)
• https://github.com/threexc/yocto-tekton/tree/main/meta-python
Workflow Example (meta-python)

1. cronjob.yaml
2. eventlistener.yaml
3. triggertemplate.yaml
4. pipeline-run.yaml
5. pipeline.yaml
6. log-task.yaml
7. build-task.yaml
8. setup-workspace.yaml

Service account configuration:
- serviceaccount.yaml

The workflow starts with the cronjob.yaml file, which triggers an event listener. The event listener then triggers the trigger template. The pipeline-run.yaml file is triggered either automatically or manually. The pipeline.yaml file is generated based on the trigger. The pipeline is composed of task stages: log-task, build-task, and setup-workspace.
Pipeline Implementation Examples

```yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: meta-python-build-pipeline
  namespace: tekton-pipelines
spec:
tasks:
  - name: meta-python-setup-workspace
    taskRef:
      name: meta-python-setup-workspace
  - name: meta-python-build
    runAfter:
      - meta-python-setup-workspace
    taskRef:
      name: meta-python-build
  - name: meta-python-log
    when:
      - input: "$(tasks.meta-python-build.results.built)"
        operator: in
        values: ["yes"]
    taskRef:
      name: meta-python-log
```

```yaml
apiVersion: tekton.dev/v1alpha1
kind: PipelineRun
metadata:
creationTimestamp: null
generateName: meta-python-build-pipeline-run
namespace: tekton-pipelines
spec:
pipelineRef:
  name: meta-python-build-pipeline
  timeout: 1h0m0s
podTemplate:
  volumes:
    - name: build
      hostPath:
        path: /tekton/data
  - name: log
    hostPath:
      path: /tekton/data/logs
status: {}
```
Implementation Examples - Continued

```yaml
apiVersion: triggers.tekton.dev/v1alpha1
kind: TriggerTemplate
metadata:
  name: meta-python-build-template
namespace: tekton-pipelines
spec:
  resourceTemplates:
    - apiVersion: tekton.dev/v1beta1
      kind: PipelineRun
      metadata:
        generateName: meta-python-build-pipeline-run-
      spec:
        pipelineRef:
          name: meta-python-build-pipeline
        timeout: "3h"
      podTemplate:
        volumes:
          - name: build
            hostPath:
              path: /tekton/data
          - name: log
            hostPath:
              path: /tekton/data/logs
```

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: meta-python-cronjob
namespace: tekton-pipelines
spec:
  schedule: "0 */12 * * *
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: meta-python-nettools
              image: threeexec/nettools
              args:
                - /bin/bash
          - curl -X POST http://el-meta-python-listener.tekton-pipelines.svc.cluster.local:8080
          restartPolicy: OnFailure
```
Automatic Shared State

Pay no attention to the man behind the curtain...
Shared State for Pipelines

• Everything you’ve seen actually relies on another pipeline for building sstate!

• Similar fundamentals - pipeline runs once per day, runs a poky build that is then used as the SSTATE_MIRROR to reduce build time

• https://github.com/threexc/yocto-tekton/tree/main/sstate/automated
Useful Tools

Improving the Workflow
Useful Tools

- K9s: https://github.com/derailed/k9s
- Helm: https://github.com/helm/helm
- Tekton Catalog: https://github.com/tektoncd/catalog
meta-python + k9s Demo

Demo Video Link
Future Work

• Better pipelines/less hard-coding (haven’t used TriggerBindings, Secrets, ConfigMaps, etc. at all)

• Proper ingress (i.e. do more than NodePort)

• Take it to the cloud - AWS/GCP/etc.

• Run built image using libvirt/kubevirt?

• Patches from patchwork using pwclient
Thanks for your time!
Questions?