Highly Scalable Yocto Project® Build Automation

Paul Barker, Konsulko Group

Yocto Project Dev Day *Virtual*, North America, 2020

Introduction

About This Talk

- **Introduction**
- **Automating Yocto Project Builds**
- **Scaling Up & Scaling Out**
- **A low-cost example setup**
  - Including public sstate cache & download mirror
- **Summary & Future Work**

About Me

- Involved in Yocto Project since 2013
- Work across the whole embedded stack
- Principal Engineer @ Konsulko Group

pbarker@konsulko.com

https://twitter.com/pbarker_dev

About Konsulko Group

- Embedded Linux & Open Source Software Consultancy
- Globally distributed team of community and industry veterans
- Contributors to the Linux kernel, U-Boot, Yocto Project, OpenEmbedded, Automotive Grade Linux (AGL) and many more projects

https://konsulko.com

Caveat Emptor

- **Opinions are mine**
  - Not my employers
  - Not those of the Yocto Project as a whole

- **Gather performance data & test thoroughly before spending lots of money on hardware/services**

Automating Yocto Project Builds

Simple Setup

- **Dedicated CI build environment**
- **CI build script**
  - Update layers
  - Run the build
  - Copy artifacts to shared storage
- **Some form of trigger**
  - When new commits are pushed to your layers
  - On a schedule

Other Requirements

- **Capture build logs**
- **Manual command to clean working directory and/or sstate cache**
  - Sometimes needed when debugging issues

Yocto Project Autobuilder

- Used and maintained by Yocto Project itself
- Based on buildbot
- Highly customisable

Other Solutions

- Buildbot
- GitLab CI
- Jenkins
- … The list is endless

Scripting the Build

- **CI configuration should be as simple as possible**
  - Ideally just runs one command
- **Ensure you can run the same script locally**
  - Makes debug much easier
  - Allows you to build & release manually if your CI system breaks

Pulling in Other Layers

- **Many possibilities**
  - Git submodules
  - Repo
  - oe-layersetup
  - kas
- **Depends on your preference, team and workflow**

Scaling Up & Scaling Out

Re-examining the Simple Setup

- **Let’s break the setup into components:**
  - Build environment
  - sstate cache
  - Download cache
  - Artifact storage
  - Log storage
  - Management interface

Scaling Up

- **Just get a bigger build machine…**
  - We can also increase efficiency

- **NVMe > SATA SSD > SATA HDD**
  - Check the write endurance on SSDs
- **Server/Workstation Hardware > Desktop > Laptop**
  - I’ve seen CI on an old laptop and people wondering why it’s slow

CPU and RAM

- **Don’t just blindly assume more expensive is better**
- **Check for single threaded bottlenecks**
  - If they dominate build time go for high clock frequency
  - If not, go for high core count
- **Profile your RAM usage during a build**
- **Use the fastest supported RAM**

Other Factors

- **Use a dedicated machine not a VM**
- **Containers are ok as the I/O overhead is low**
  - Look closely at the docs if you’re using docker
- **Separate the management interface**
  - Smaller controller machine or VM

Preparing to Scale Out

- **Centralised NAS or other storage solution**
  - sstate cache
  - Download mirror
  - Artifact storage

- **Multiple build machines**

Other Benefits of Scaling Out

- Developer machines can use central sstate cache
- Maintaining a download mirror is important anyway
  - Helps with license compliance
  - Protects you if upstream sources disappear
- Can reduce single points of failure

Local Scale Out

- **High-speed networking**
  - 10 Gbps between build machines & NAS is recommended
- **Reliable network**
  - Very low likelihood of transfer errors
- **Serve and update caches over NFS**

Global Scale Out

- Limited network speeds
- Packet drops & errors will happen
  - May see errors accessing the sstate cache
- Serve caches over HTTP(S)
- Update caches over SSH or an API

Distributed Cache & Artifacts Storage

- **You can build your own storage cluster**
  - Using Ceph, Glusterfs, etc
  - Minio is also an option with an S3 compatible API

- **Or you can use a cloud service**
  - Amazon S3, Azure, Google Cloud or similar
  - BackBlaze B2 is a low-cost option
  - Avoid Dropbox, OneDrive, Google Drive, etc as they're not designed for this

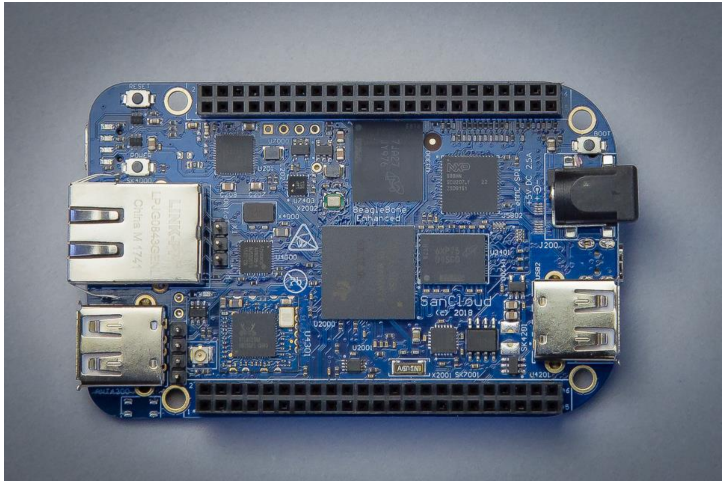
A Low-Cost Example

Use Case

www.sancloud.co.uk

- **meta-sancloud BSP layer**
  - SanCloud BeagleBone Enhanced (BBE)
  - https://github.com/SanCloudLtd/meta-sancloud

- **Supports multiple distros**
  - Poky
  - Arago
  - AGL



High Level Design

- Uses kas to set up layers and build configuration
- GitLab CI is used to trigger builds and collect logs
- Dedicated build servers running GitLab Runner
- Cache, mirrors & artifacts stored in BackBlaze B2
- CloudFlare used to eliminate bandwidth costs

kas

- **Records build configuration in a YAML file**
  - Source repository and refspec for each layer
  - Local patches to apply
  - MACHINE, DISTRO and bitbake targets to build
  - Content of local.conf
- **Simple command line usage:**
  - `kas build kas/bbe-poky.yml`
  - `kas shell kas/bbe-poky.yml` for custom commands

GitLab CI

- Continuous Integration system integrated with GitLab
- Configuration stored in YAML file in the Git repository
- Variables and secret values set in the web interface
- Builds triggered automatically
  - On git push
  - On a schedule (nightly builds)

Build Agent

- CPU: Ryzen 7 3700X (8c/16t)
- RAM: 64 GB DDR4 ECC
- Storage: 2x 1 TB NVMe drives in a RAID1 pair
- Internet Connection: 1 Gbps symmetric

- Rented from Hetzner (Germany)

GitLab Runner configuration

- **GitLab runner has very minimal configuration**
  - Limit concurrent jobs (default is unlimited)
  - Select the Docker job executor
  - Register with GitLab server
- **Set docker image in Gitlab CI YAML file**
  - CROPS images are perfect for this
  - We also use custom images with additional tools installed

BackBlaze B2

- **Cloud storage solution from BackBlaze**
  - Their main product is cloud backup
- **Cheap: $0.005 per GB per month**
  - Pay only for used storage space
  - Allows costs to be capped
- **No upload cost**
- **Downloads cost $0.01 per GB but we can avoid that...**

CloudFlare

- Bandwidth Alliance between CloudFlare & BackBlaze
- Downloads from BackBlaze B2 are free via CloudFlare
- The free tier of CloudFlare is sufficient for this
- Requires a dedicated domain
- Make sure you disable Browser Integrity Check

Uploading to BackBlaze B2

- **Use rclone**
  - Like rsync but for cloud storage
- **BackBlaze API key stored as a GitLab CI variable**
  - Not stored in the git repository
  - Only repository admins can view/edit this

Monthly Costs

- **Build Agents: approx. €60 per month each**
- **BackBlaze B2: approx. $2.00 per month**
- **No long term commitments**
  - Can scale up or down as required

Example references

- GitLab CI config
- Kas build configurations:
  - Poky
  - Arago (based on poky config with changes)
- Bitbake inc files:
  - Enable download mirror
  - Enable sstate mirror

Summary & Future Work

Summary

- **A scalable automated build system can be deployed cheaply**
- **Requires some sysadmin work in setup & maintenance**
- **Many choices available for all components**
  - You can start with a low cost solution and replace components as needed

Future Work

- Extend kas capabilities
- Improve handling of sstate download failures in bitbake
- Provide better linkage between CI logs in GitLab and artifacts stored on BackBlaze
- Automatically expire old sstate caches

Thanks for your time

yocto PROJECT

THE LINUX FOUNDATION

yocto PROJECT

THE LINUX FOUNDATION