Uni Web Topics Presentation

Presentation on Cloud Native Development

Felicitas Pojtinger

2021-11-19

Introduction

Contributing

These study materials are heavily based on professor Heuzeroth's "Spezielle Themen für Web-Anwendungen" lecture at HdM Stuttgart.

Found an error or have a suggestion? Please open an issue on GitHub (github.com/pojntfx/uni-webtopics-notes):

Figure 1: QR code to source repository

If you like the study materials, a GitHub star is always appreciated. :)

License

Figure 2: AGPL-3.0 license badge

SPDX-License-Identifier: AGPL-3.0

Overview

Overview

- What is DevOps?
- Which parts of the software lifecycle does it cover?
  - Development
  - Distribution (I will focus on this today)
  - Operation
- What is “cloud native”?
- Why are “traditional” distribution methods still relevant?

Development

Development

- DevOps also includes development!
- Modern development should not be bound to any client attributes (it should not matter if the client is a RISC-V Linux machine, a locked-down Windows workstation or an Android phone)
- Development should be possible from any platform, for any platform
- The only truly cross-platform application framework is the web
- PWAs make it possible for web apps to have all the features native apps have
- PWAs work offline by default
- Why not make our development environments PWAs?
- Virtual machines and user-friendly hypervisors and containers make it possible to run the editor's backend locally too

Distribution

Basic Distribution Principles

- Binaries
  - Compiled forms of software
  - On Linux: ELF binaries, PE binaries on Windows and MACH-O binaries on macOS
  - Binaries can be statically or dynamically linked
    - Statically linked: Since the Linux ABIs are stable, one can depend on them not changing - this allows not linking against any specific C library and makes the resulting binary portable across distributions. It also allows including all external dependencies into the binary, effectively making it a “single-file” distribution method
    - Dynamically linked: Thanks to dlopen and package management, dynamic linking can also be used. Most of the time (especially on non-Linux OSes), at least the C library and external dependencies (i.e. SQLite) thus need to be available in LD_LIBRARY_PATH at runtime; if they are not, the application can’t continue. This makes the binaries non-portable across distributions; for example, if a binary is built on a Debian 11 host, it most probably won’t run on a Debian 10 host due to the different versions of the GNU C library used. This does however

Packaging Overview

- What is a package?
    - Includes the binary, assets, metadata and signature
    - Is self-describing
    - Mostly some form of archive (i.e. RPM, `.tar.gz`) in combination with a metadata file and signature
- What is a package manager?
    - Can install, remove and update packages
    - Mostly two components: Low-level tool to install and remove package files (dpkg on Debian, rpm on Fedora) and a high-level tool to search, download, install and resolve dependencies (apt on Debian, dnf on Fedora)
    - Can resolve and install runtime and build-time dependencies (i.e. dependency on C library, SQLite, SDL2, headers for cURL etc.)
    - Can check GPG signatures of
- Repository

Distribution to RedHat Enterprise Linux

- RHEL is a very popular distribution and serves as the upstream of many other distros (CentOS, Rocky Linux etc.)
- Fedora Linux is its upstream
- Is based on the RPM package format and the DNF package manager
- Commercial
- Very long support cycles (at least ten years per major release)
- RPM package format: Demo
- DNF package manager and repositories: Demo

Distribution to Debian GNU/Linux

- Debian is another very popular distribution that also serves as the upstream of many other distros (Ubuntu, Linux Mint, Pop!_OS etc.)
- Is based on the DEB package format and the APT package manager
- Community-Driven, completely Free Software
- 5 years support per major release
- DEB package format: Demo
- APT package manager and repositories: Demo

Distribution to Linux (universal)

- Flatpak is a universal package format for Linux (“apps for Linux”)
- Sandboxed
- Standalone/runtime-based (does not depend on OS-provided libraries)
- Runs on all Linux distributions, even older major releases: Allows the user to install modern software on stable systems like Debian
- Permission system: Camera, File Access, Terminals
- Portals: Allow interaction with host system (i.e. screen sharing, file access etc.)
- Includes a universal repository system
- Demo: Setting up Flatpak and installing “Video Downloader”
- No concept of source packages; a single YAML/JSON manifest (reverse FQDN) is the source package and serves as the build system too
- Tight integration with GNOME Builder

In Comparison: Comparison to Distribution with Android, Windows and macOS

- As we are in a web context I will only take a short look at proprietary platform's distribution mechanism
- The intention here is to show why these platforms are not viable for secure usage
- Android
  - APKs can be published to a "app store" which is comparable to a repository, i.e. F-Droid, Huawei AppGallery, Samsung Galaxy, Amazon App Store or Google Play.
  - APKs are ZIP files with a Manifest, Dalvik Bytecode, Resources and native libraries
  - Updates are handled by the app store
  - Permissions are handled by the operating system
- Windows
  - MSI packages or self-extracting installers are used
  - No concept of source or binary packages, no reproducibility - most

Distribution to Kubernetes/the Cloud

- Native
  - scp to server
  - Configure systemd Unit
  - Get status and logs using systemctl and journalctl
- Docker
  - Like VMs, but the host kernel is shared (isolation taking place by using CGroups)
  - “Apps for servers”
  - Workflow
    - Define Dockerfile
    - Create OCI image by using docker build
    - Push using docker tag and docker push
    - Pull using docker pull
    - Start using docker run
    - Get logs using docker logs

Pipelines

- bagop & bagccgop
  - Tools to build for a lot of platforms
  - Examples in Go, but these kind of tools exist for almost all programming languages
  - Cloud environments and Kubernetes clusters are heterogeneous: They can have many different CPU architectures (x86_64, arm64, riscv64, ppc64 etc.)
  - Portability is the key factor; your apps should not assume their target platforms and use standard build systems, which makes this much easier
  - bagop and bagccgop try to build for all possible targets (i.e. Linux on riscv64, FreeBSD on x86_64 etc.) by default and then allows you to disable platforms which one can't compile for
  - bagop works for pure Go, is fast and builds for ~40 targets by default
  - Demo: Build a Hello World Go app with bagop
  - C libraries further complicate the situation: These require dynamically linked libraries and headers at compile/runtime, which