MILEPOST GCC

Docker image

Lodi Luca 877862

MILEPOST GCC

- Milepost-gcc is a compiler (based on gcc) that can automatically tune the compilation based on some **features extracted from the code**, to adapt to different architectures and objectives.
- The extracted **features** can be analyzed by other tools, since they are **exported as textual fles** during the compilation.

MILEPOST GCC

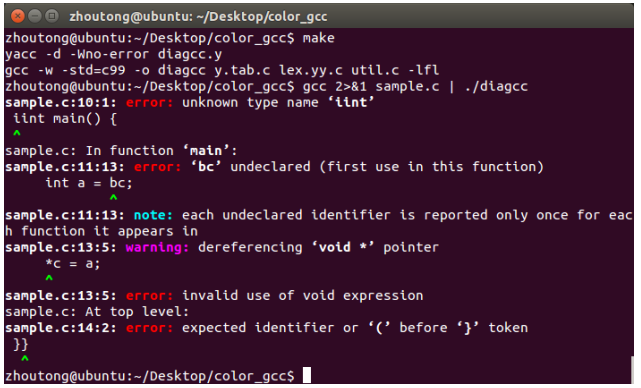
- As for many research projects, this tool's development and maintenance was **discontinued in 2010** (v2.1.1)
- This tool ceased to be used for compilation, instead is used just for **code feature extraction** only, inside another compiler: **ctuning-cc** (spiritual continuation of milepost gcc), moved into the 'Collective Knowledge' (**ck**) project since 2015

The problem

1) Milepost's **code feature extraction** core is **still useful for and used in** many machine learning based code compilation / **optimization projects**.

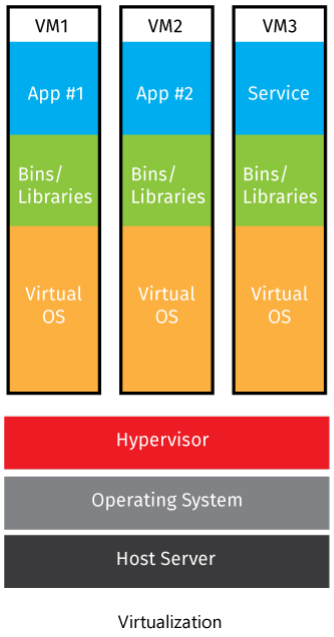
2) Due to its **age (>7years)** it is **extremely hard to compile** and use under new OS and environments.

3) **Days can be wasted** in recreating Milepost's **execution environment**.

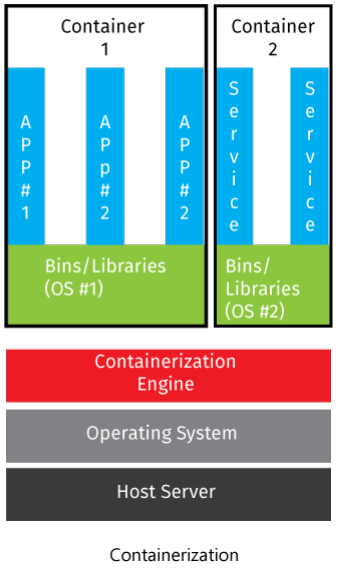


The solution

Containerization (para-virtualization) can easily **recreate and 'ship' its execution environment** within a milepost container (**image**), that can be executed **without the need** of any otherwise **cumbersome VM**.



Virtualization



Containerization

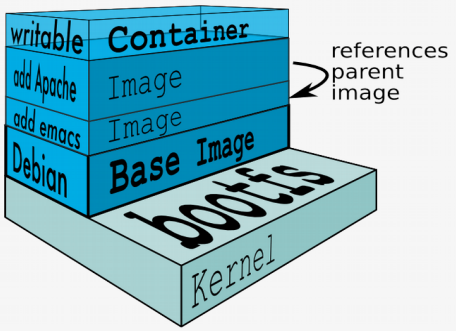
Docker can easily **ship images** executable under linux and windows, with **no setup time,** since it exploit the underlying OS's kernel / libraries.

Docker

- Let **build** an image **from a recipe** (**Dockerfile**), on top of a **base image** (e.g: an OS)

- The image can be **shipped** (distributed) from a **Docker Hub repository**

- Any computer can **run the image** (sw and environment) once **pulled** from the Docker Hub.
- Docker images are **composed of layers**, each layer can be built shipped and pulled **independently**, exploiting a **powerful hierarchical cache** system.



Challenges (image's requirements)

- It should **hide** the environment setup's **complexity**
- It should be **lightweight** and with **minimum dependencies** (trade-of)
- It should **ease** as much as possible the code feature **extraction process**.
- It should be **easy to run**.

Taken steps

- Selected Ubuntu 16.04 as **base image** for compatibility reasons [~47MB layer]
- Install some **dependencies** for compilation (build) process [~137MB layer]
- Build the **Milepost GCC 2.1.1** (inside Ctuning-cc 2.5) with all the dependencies, following the setup process suggested in the CK project. [~2.4GB layer]
- Setup confguration of the tool in the image [~100KB layer]

  This image can work, but is around 3GB of size !

- **Analysis** of the **main size inefciency:**
  - The build did not clean **object fles** after the compilation [~1.6GB waste]
  - Compilation OS **libraries / binaries**, useless for execution [~500MB waste]
  - Each **layer store the fle diference w.r.t. the previous layer** (a shell command is a layer), so **installing in one layer and uninstalling in a diferent layer store a fle twice**. (but enable caching of intermediate layers)
- Exploration of docker **multi-stage build**: selectively copy just some fles from a previous layer to a new image ... unsuccessful since it was almost impossible to identify all the required fles. (**additive process**)

Taken steps

- Exploration of the **--squash build fag**: compress all the layers into one, saving twice the size of fles installed and then deleted in diferent layers. Successful in producing a smaller working image.
- **Iterative hierarchical deletion process** to select the unnecessary fles in the image (.obj fles, OS lib / bin, CK repo, ...)

  Now the image size is ~552MB on disk.
- Creation of a **script** that can **automatically extract the features** from the .c fles in a folder, collecting them in a ‘features’ folder. The script is executed once the image is run.

  Final image, ~552MB, 1 layer
- Creation of a **script (alias)** to **simplify the usage of the image**: pull the image, mount working dir in the image, run image extracting the features, and copy back outside the image.
- **Extensive testing** on diferent **OS / machines**.
- **Documentation** and **publishing** in a public repo.

Final result

- Well documented public **GitHub repo** with:
  - **Dockerfle** recipe (for future extension)
  - Test **sources** for the image
  - **Alias script**
  - Usage **guide**

  https://github.com/lulogit/milepost-feature-extractor-docker
- Public **Docker Hub repo** from which to **pull the compressed image** for execution (**~208MB**)

  https://hub.docker.com/r/lullo/milepost-feature-extractor/