LA-UR-20-25968

bueno

Benchmarking, Performance Analysis, & Provenance

Presented by Jacob Dickens
Mentors: Samuel Gutierrez (HPC-ENV) and Howard Pritchard (HPC-ENV)

Los Alamos
NATIONAL LABORATORY
EST. 1943

1. **Overview**
    a. Project and Goals
    b. Services
2. **Provenance Mechanisms**
    a. Containers
    b. Docker
    c. Charliecloud
3. **Benchmarking Challenges**
    a. Code Variety
    b. Storage
4. **Going Forward**
    a. Automation/Reproducibility
    b. Status

Overview - Project and Goals

bueno is a benchmarking system providing users with tools to automate application performance analysis

- **Reproducibility** & **automated testing**

A framework that is extendable; replacing tedium of:

- **Data storage/analysis**
- **Program compilation/execution**
- **Environmental discovery/setup**

bueno documentation can be found at: https://lanl.github.io/bueno/html/intro.html

Overview - Services

bueno's command line interface allows for two categories of service:

- **Build** - create container images with metadata annotations for later analysis and configuration reconstruction
- **Run** - execute bueno scripts and coordinate container image activation via Charliecloud or host pass-through (i.e., a non-container option)

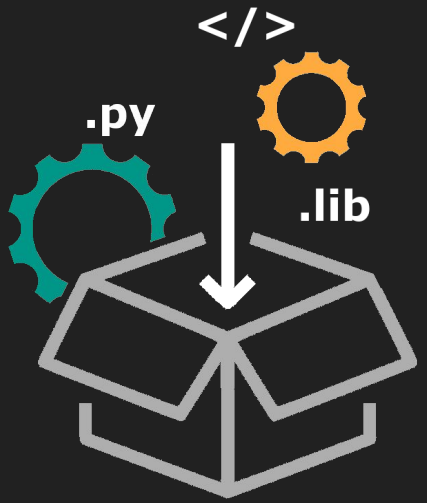
bueno's collection of Python modules aid benchmarking

- **Provides data logging**
- **Records of experiment generation settings**
- **Exposes user-programmable pre- and post-experiment actions**

Provenance Mechanisms - Containers

Containers are self-contained environments that include software packages.

- **Lightweight** - doesn't require reserved OS environment
  - More efficient use of server space
  - Negligible performance-impacting overhead
- **Secure** - strongest default form of application isolation
- **Standard** - current industry standard, broad portability



Lange, John, et al. "Palacios and kitten: New high performance operating systems for scalable virtualized and native supercomputing." 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS). IEEE, 2010.

Providence Mechanisms - Docker

Automate creation and management of services within containers, addressing potential problems with application dependencies:

- **Conflicts** - Different (potentially conflicting) versions can be run in separate docker containers
- **Missing** - All dependencies are packaged with the application
- **Environment** - Migrating to new distributions is straightforward so long as Docker is supported

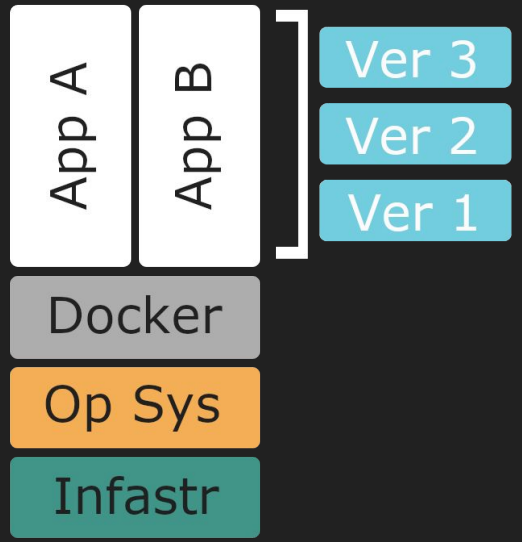
Merkel, Dirk. (2014). Docker: Lightweight Linux Containers.... Retrieved from https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment

Providence Mechanisms - Docker (continued)

- Replaces additional or specialized hardware
- Reduces installation & configuration time
- Simplifies image management
  - Eases repeatability

Docker images are made of collections of layers

- Added when the modifying the image
- Reused for later builds or rolled back as built-in version control



Koutoupis, Peter (2018). Containers. Retrieved from https://www.linuxjournal.com/content/everything-you-need-know-about-linux-containers-part-ii-working-linux-containers-lxc

Providence Mechanisms - Charliecloud

bueno uses Charliecloud, a similar system to Docker

- User-Defined Software Stacks (sets of containers)
- Aids dependency management
- Portable and internally consistent

Charliecloud stands out from Docker by allowing all of this without needing root privileges at build and runtime. An issue that makes Docker a security risk in practice.

Kniep, Christian (2018). The State of HPC Containers. Retrieved from https://www.stackhpc.com/the-state-of-hpc-containers.html

Benchmarking Challenges - Code Variety

Code and code frameworks can vary widely from project to project, but also between individual versions of the same project.

E.g., code that uses different versions of key computational kernels

Collecting data for comparative studies (e.g., post-mortem analysis) is a must

Benchmarking Challenges - Storage

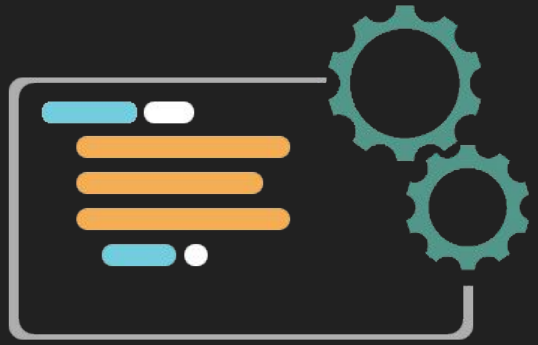
The test results/configuration retained for later use in studies require storage space with each new recorded test and the initial registry of container images.

- **Information** - A potentially large amount of information is needed to guarantee repeatability
- **Memory** - The less information you record in each automated test, the less memory you will need to have set aside for that purpose

Going Forward - Automation/Reproducibility

Our highest priority with bueno:

- Facilitate the reliable automation of application testing
- Support distribution of test settings as well as the benchmark results
  - Improve reproducibility between researchers



Going Forward - Status

The core bueno framework is mostly complete along with public APIs, minor (user-facing) changes are expected moving forward. There is currently support for five applications of varying complexity:

- **Micro-Benchmarks**
- **Proxy & Real Applications**

bueno's source code is available on Github at https://github.com/lanl/bueno

Questions?

If you are interested in bueno supporting your application please let us know

Jacob Dickens
jacobpd@newmexicoconsortium.org

Los Alamos
NATIONAL LABORATORY
EST. 1943