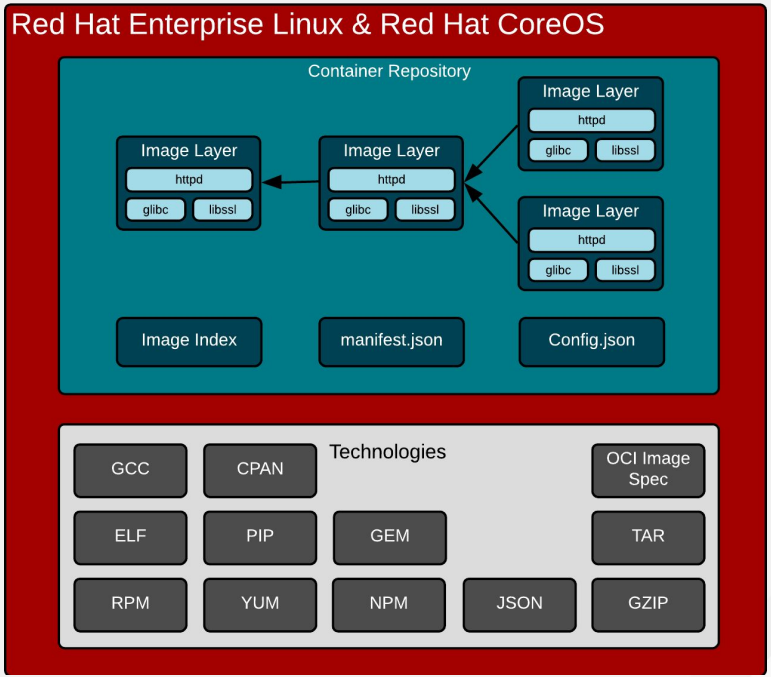

CONTAINER IMAGES
redhat.

CONTAINER IMAGE

Open source code/libraries, in a Linux distribution, in a tarball

Even base images are made up of layers:

- Libraries (glibc, libssl)
- Binaries (httpd)
- Packages (rpms)
- Dependency Management (yum)
- Repositories (rhel7)
- Image Layer & Tags (rhel7:7.5-404)
- At scale, across teams of developers and CI/CD systems, consider all of the necessary technology

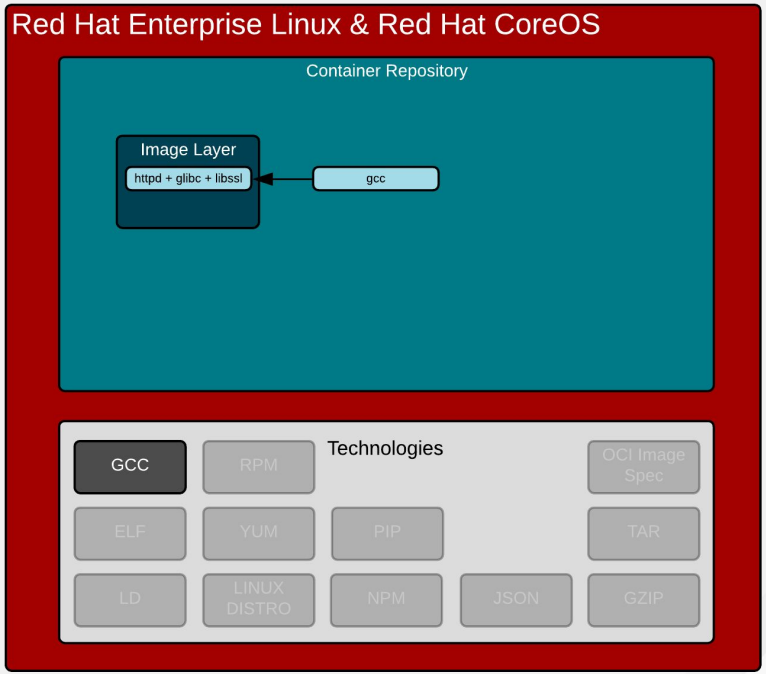


IT ALL STARTS WITH COMPILING

Statically linking everything into the binary

Starting with the basics:

- Programs rely on libraries
- Especially things like SSL - difficult to reimplement in for example PHP
- Math libraries are also common
- Libraries can be compiled into binaries - called static linking
- Example: C code + glibc + gcc = program

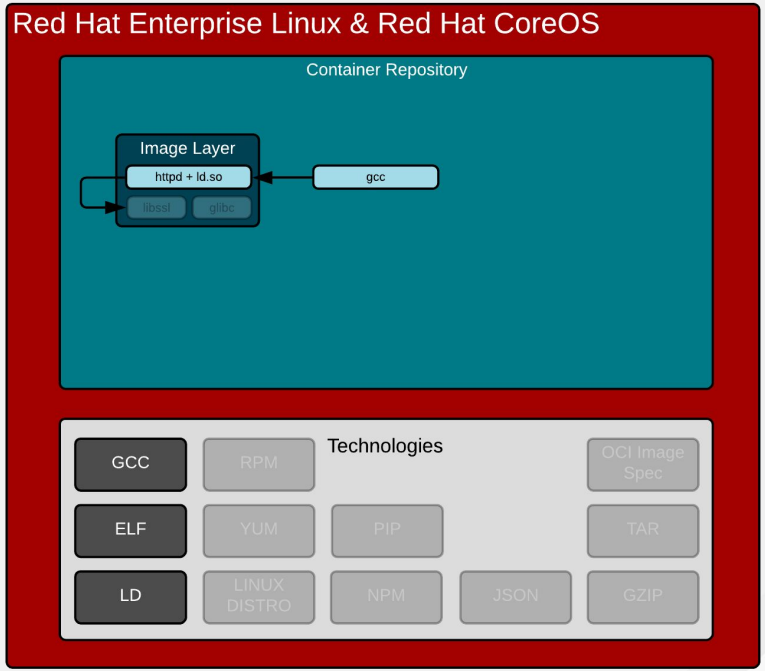


LEADS TO DEPENDENCIES

Dynamically linking libraries into the binary

Getting more advanced:

- This is convenient because programs can now share libraries
- Requires a dynamic linker
- Requires the kernel to understand where to find this linker at runtime
- Not terribly different than interpreters (hence the operating system is called an interpretive layer)

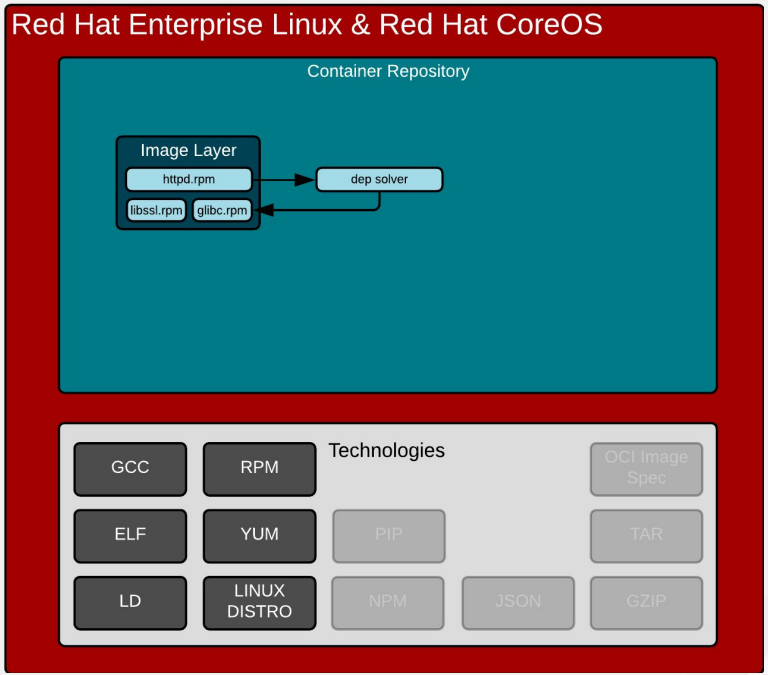


PACKAGING & DEPENDENCIES

RPM and Yum were invented a long time ago

Dependencies need resolvers:

- Humans have to create the dependency tree when packaging
- Computers have to resolve the dependency tree at install time (container image build)
- This is essentially what a Linux distribution does sans the installer (container image)

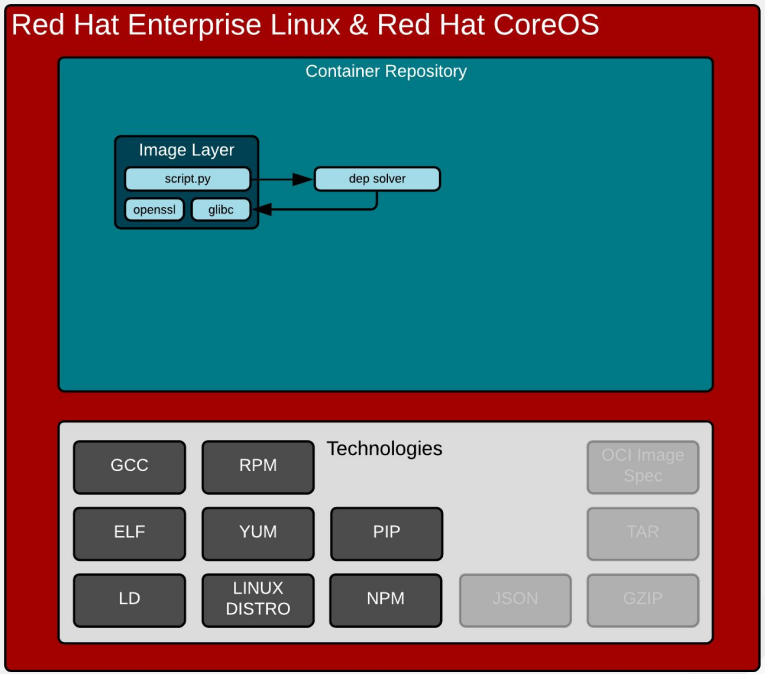


PACKAGING & DEPENDENCIES

Interpreters have to handle the same problems

Dependencies need resolvers:

- Humans have to create the dependency tree when packaging
- Computers have to resolve the dependency tree at install time (container image build)
- Python, Ruby, Node.js, and most other interpreted languages rely on C libraries for difficult tasks (ex. SSL)

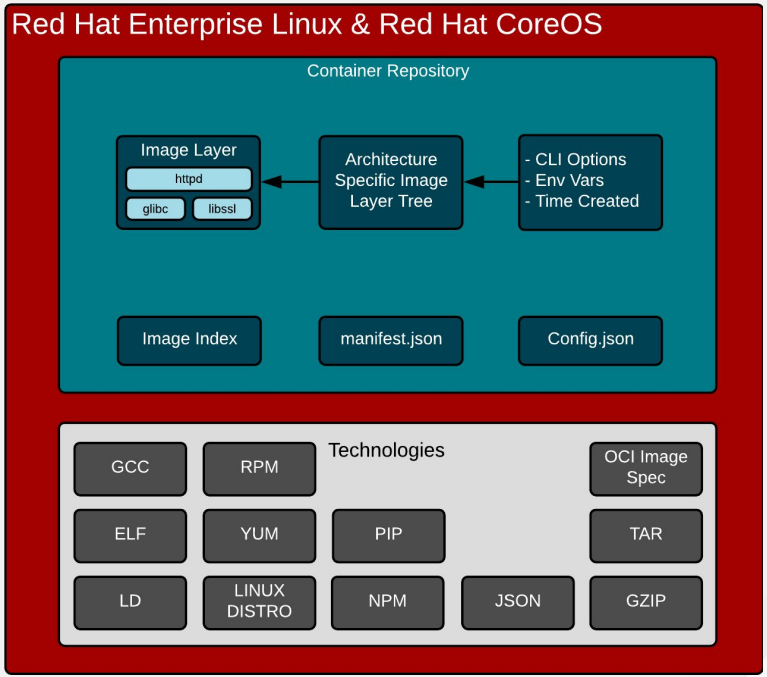


CONTAINER IMAGE PARTS

Governed by the OCI image specification standard

Lots of payload media types:

- Image Index/Manifest.json - provide index of image layers
- Image layers provide change sets - adds/deletes of files
- Config.json provides command line options, environment variables, time created, and much more
- Not actually single images, really repositories of image layers

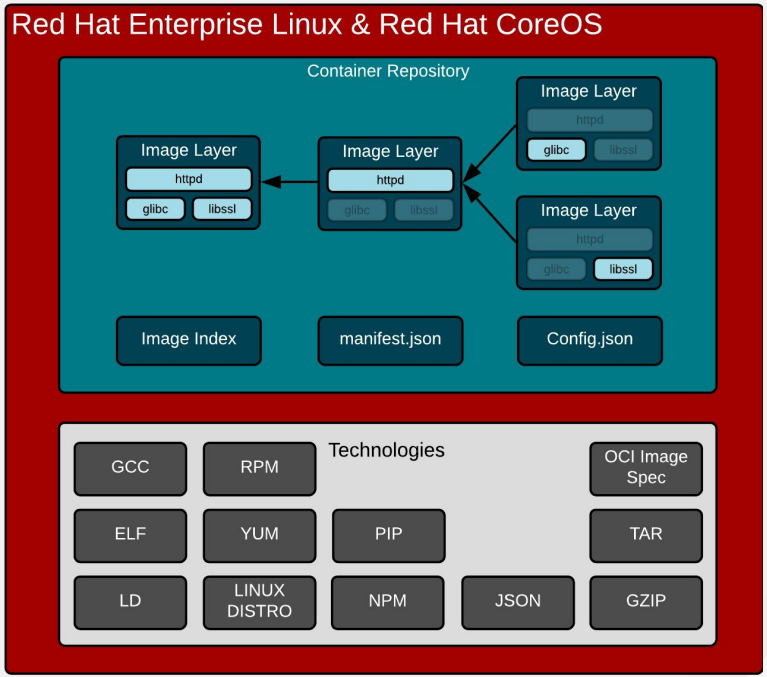


LAYERS ARE CHANGE SETS

Each layer has adds/deletes

Each image layer is a permutation in time:

- Different files can be added, updated or deleted with each change set
- Still relies on package management for dependency resolution
- Still relies on dynamic linking at runtime

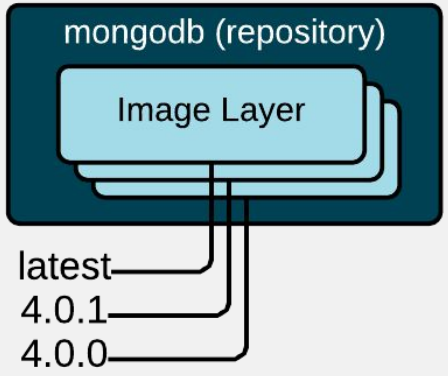


LAYERS ARE CHANGE SETS

Some layers are given a human readable name

Each image layer is a permutation in time:

- Different files can be added, updated or deleted with each change set
- Still relies on package management for dependency resolution
- Still relies on dynamic linking at runtime



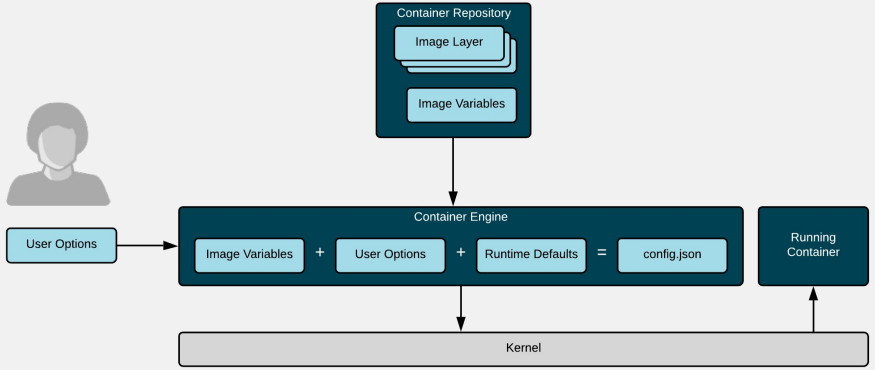
Layers and Tags

CONTAINER IMAGES & USER OPTIONS

Come with default binaries to start, environment variables, etc

Each image layer is a permutation in time:

- Different files can be added, updated or deleted with each change set
- Still relies on package management for dependency resolution
- Still relies on dynamic linking at runtime

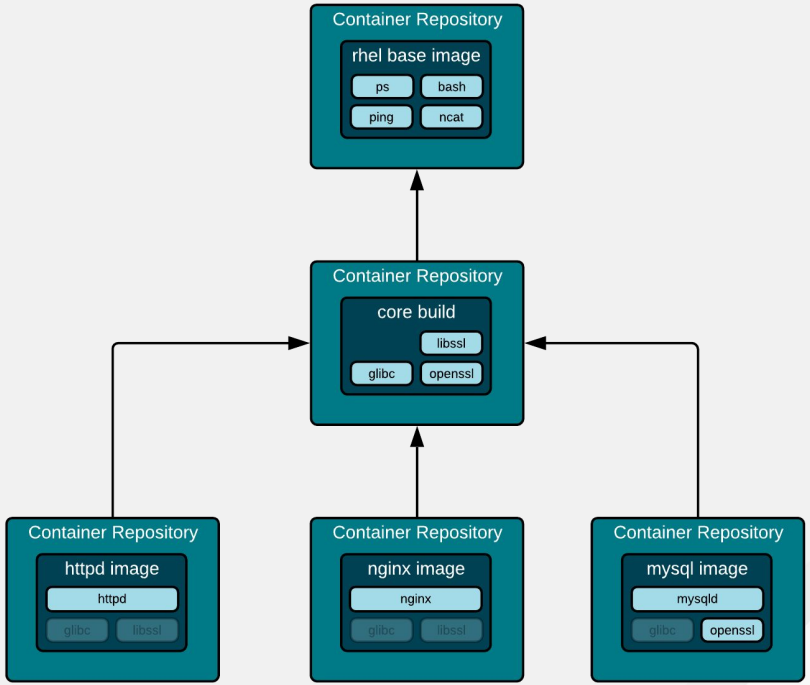


INTER REPOSITORY DEPENDENCIES

Think through this problem as well

You have to build this dependency tree yourself:

- DRY - Do not repeat yourself. Very similar to functions and coding
- OpenShift BuildConfigs and DeploymentConfigs can help
- Letting every development team embed their own libraries takes you back to the 90s



CONTAINER IMAGE

Open source code/libraries, in a Linux distribution, in a tarball

Even base images are made up of layers:

- Libraries (glibc, libssl)
- Binaries (httpd)
- Packages (rpms)
- Dependency Management (yum)
- Repositories (rhel7)
- Image Layer & Tags (rhel7:7.5-404)
- At scale, across teams of developers and CI/CD systems, consider all of the necessary technology