Emulation, Virtualization, Containerization

EaaSI Training Module #4

During This Module

- What is the difference between emulation, virtualization, and containerization?
- Why does the difference matter for long-term digital preservation?
- How, why, and when does the EaaSI platform make use of each?

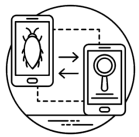
But first…have you ever wished you had a fresh, new computer - even though your current one isn't technically "broken"?

Maybe you want to...

- Run an application not compatible with your operating system

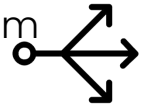
- Isolate a program from your other files

- Test how your application/script/file will behave on someone else's computer



- Troubleshoot corrupt, buggy, or just unexpected software behavior

- Host multiple web sites from the same server

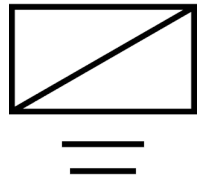


- Present a sparkling clean desktop when you have to screen-share in that video call later

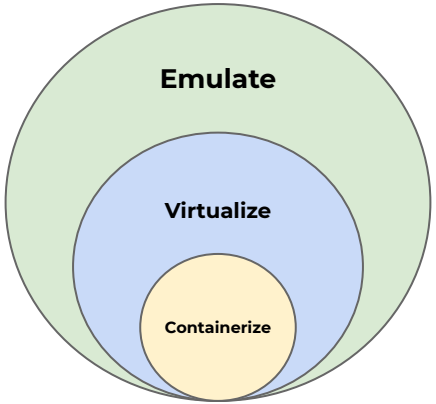
...but physical computers are heavy and expensive...

...and it's not very efficient to have two (or more) machines around just to do one thing each!

Abstracting a Computer



- Emulation, virtualization, and containerization are all methods to make components of a physical machine abstract or virtual
- Helps make software compatible and portable across systems; or, can help you duplicate and isolate systems on a single *real* computer (“host”)
- Choosing a method depends on the level of compatibility and complexity of your target software (“guest”)



Emulation

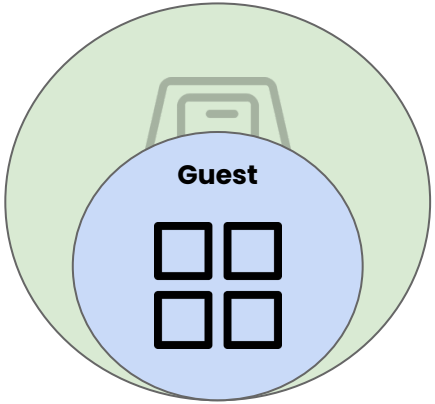
- The entire “guest” system is recreated by software - including hardware
- The “guest” is fundamentally *incompatible* with the “host” (without emulation)
- For example: a current PC **can not** run a Commodore 64 program without full emulation
- Handy for long-term access and backwards compatibility

Contemporary Examples of Emulation:

- *Mini vMac (early Macintoshes)*
- *VICE (Commodore)*
- *Hatari (Atari)*

Virtualization

- The “guest” system makes at least some use of real “host” hardware
- The “guest” is fundamentally *compatible* with the “host”
- For example: an Intel iMac **can** run Windows 10, it just doesn’t come pre-installed
- Handy for efficiency, cross-compatibility

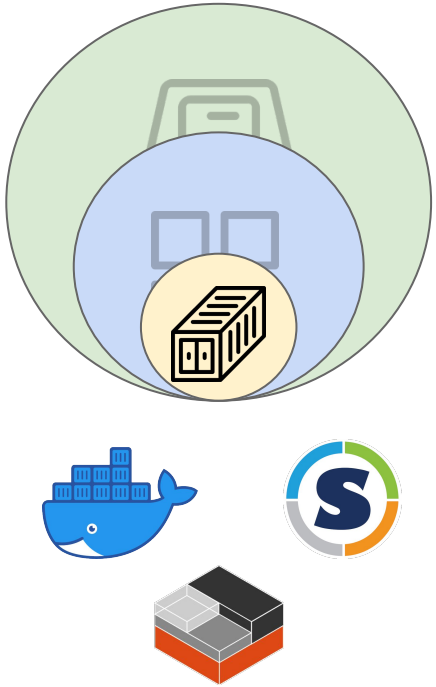


Contemporary Examples of Virtualization:

- *Parallels Desktop*
- *VirtualBox*
- *VMWare*

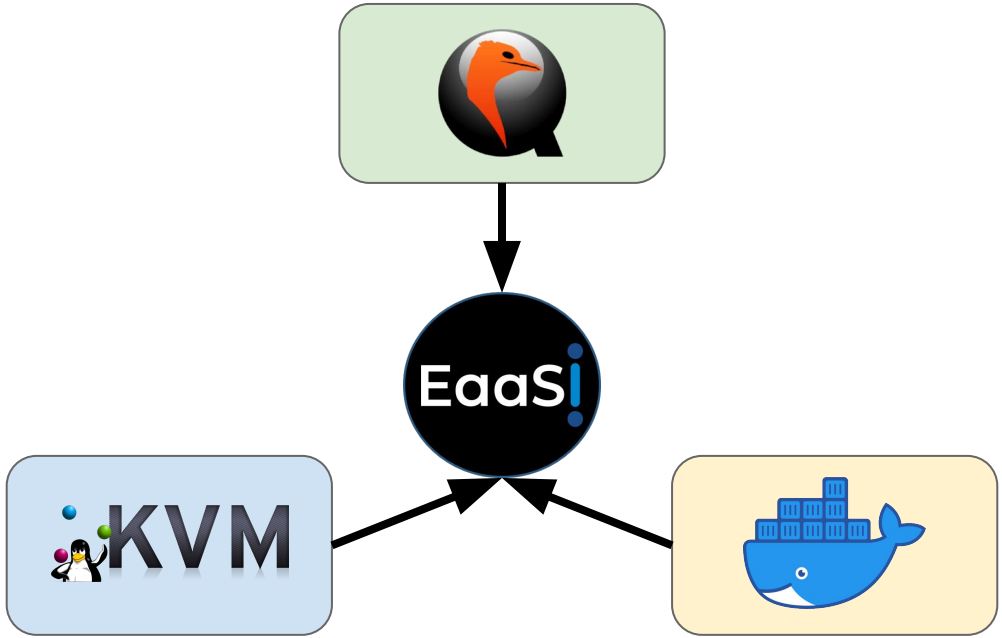
Containerization

- A single application is packaged with only the bare minimum of operating system components necessary to run it
- Emulation and virtualization recreate entire “guest” systems; containers don’t even try
- Also fundamentally requires *compatibility* between the software in the container and “host” hardware
- Handy for isolating, distributing applications



Contemporary Examples of Containerization::

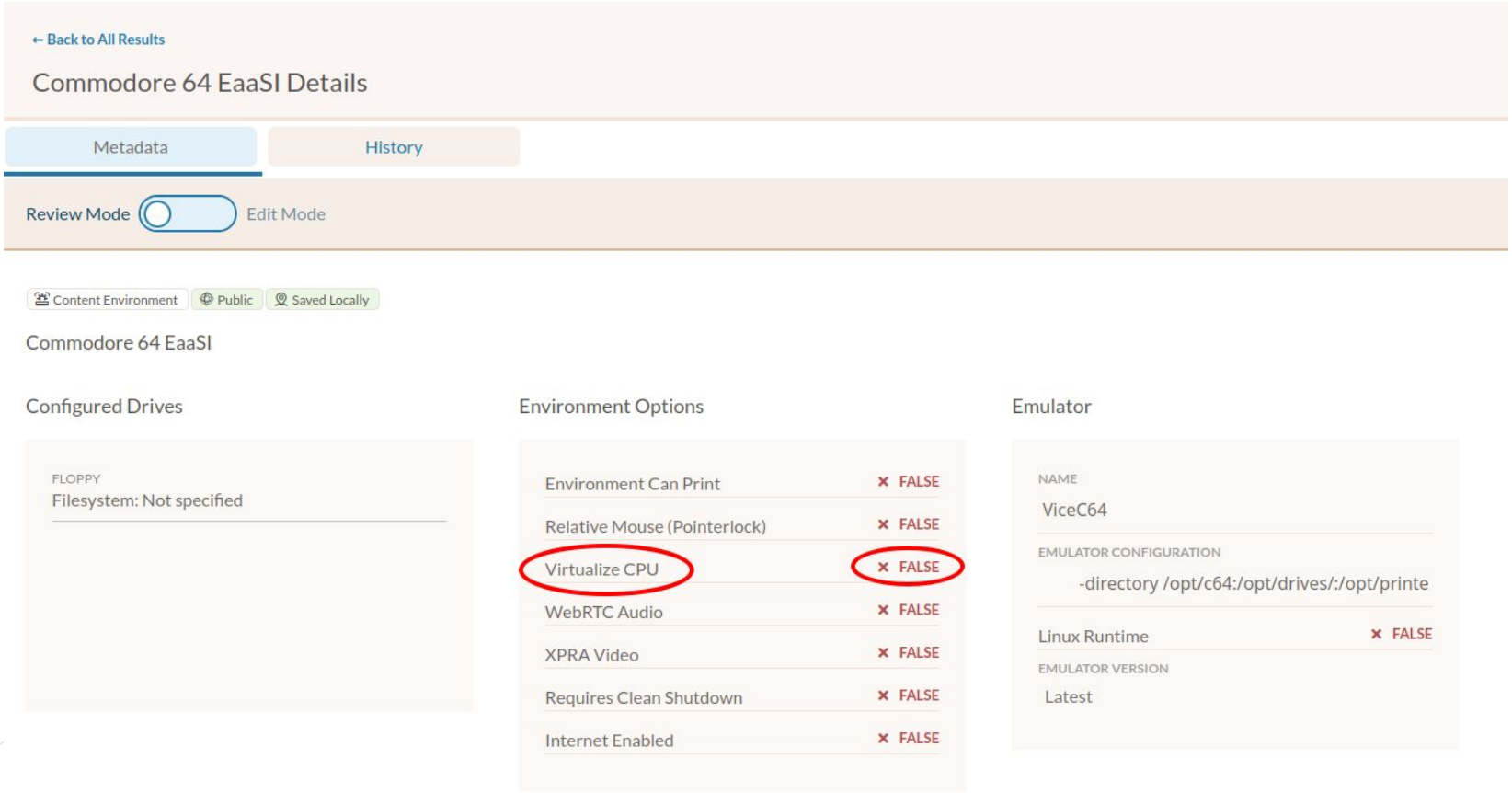
- *Docker*
- *LXC*
- *Singularity*



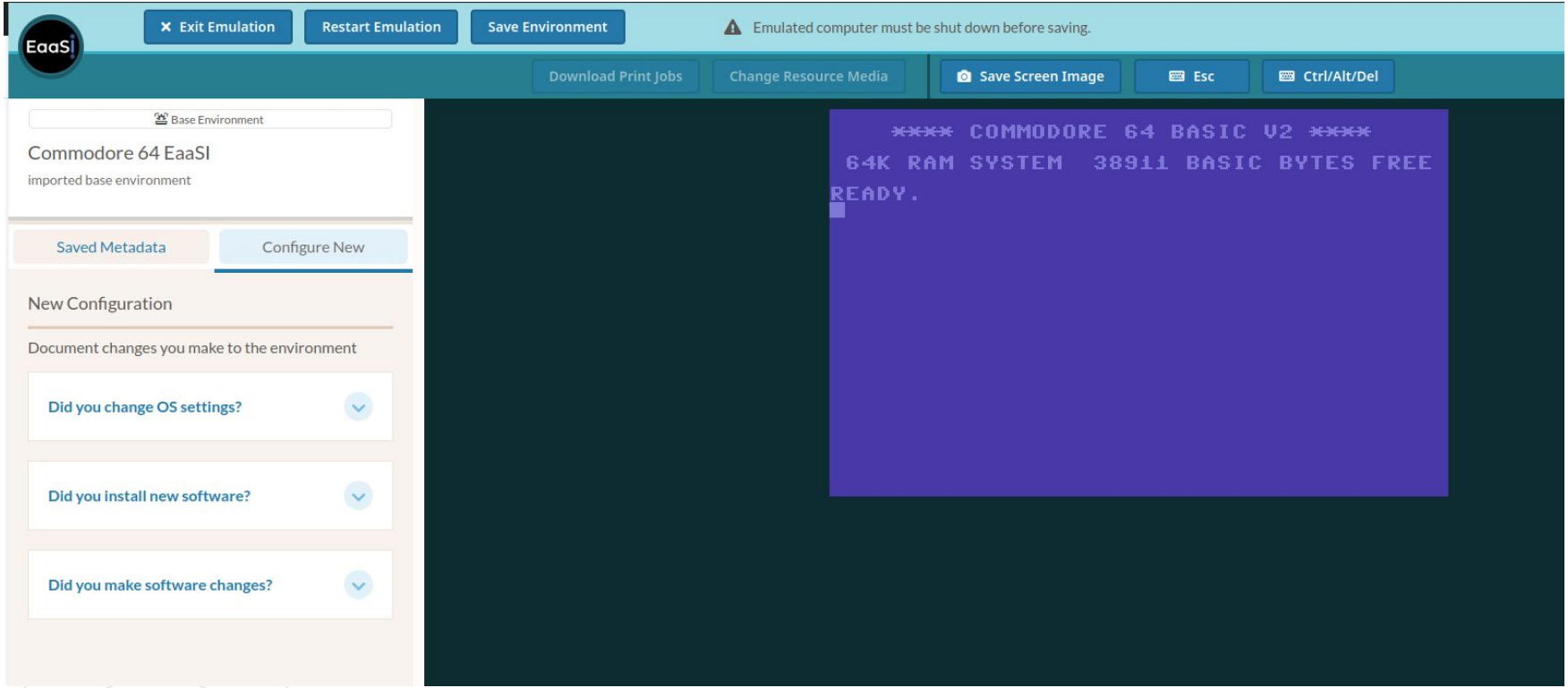
EaaSI takes advantage of all three techniques - for different reasons!

EaaSI Emulation

- Environments require an emulator to recreate hardware
- No emulation, no Environments
- Indispensable to the platform and our program of work - that's why it's "**Emulation**-as-a-Service Infrastructure" :)

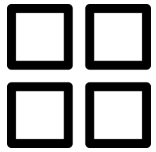


ex. Commodore 64 Environment in EaaSI - virtualization is incompatible and not available

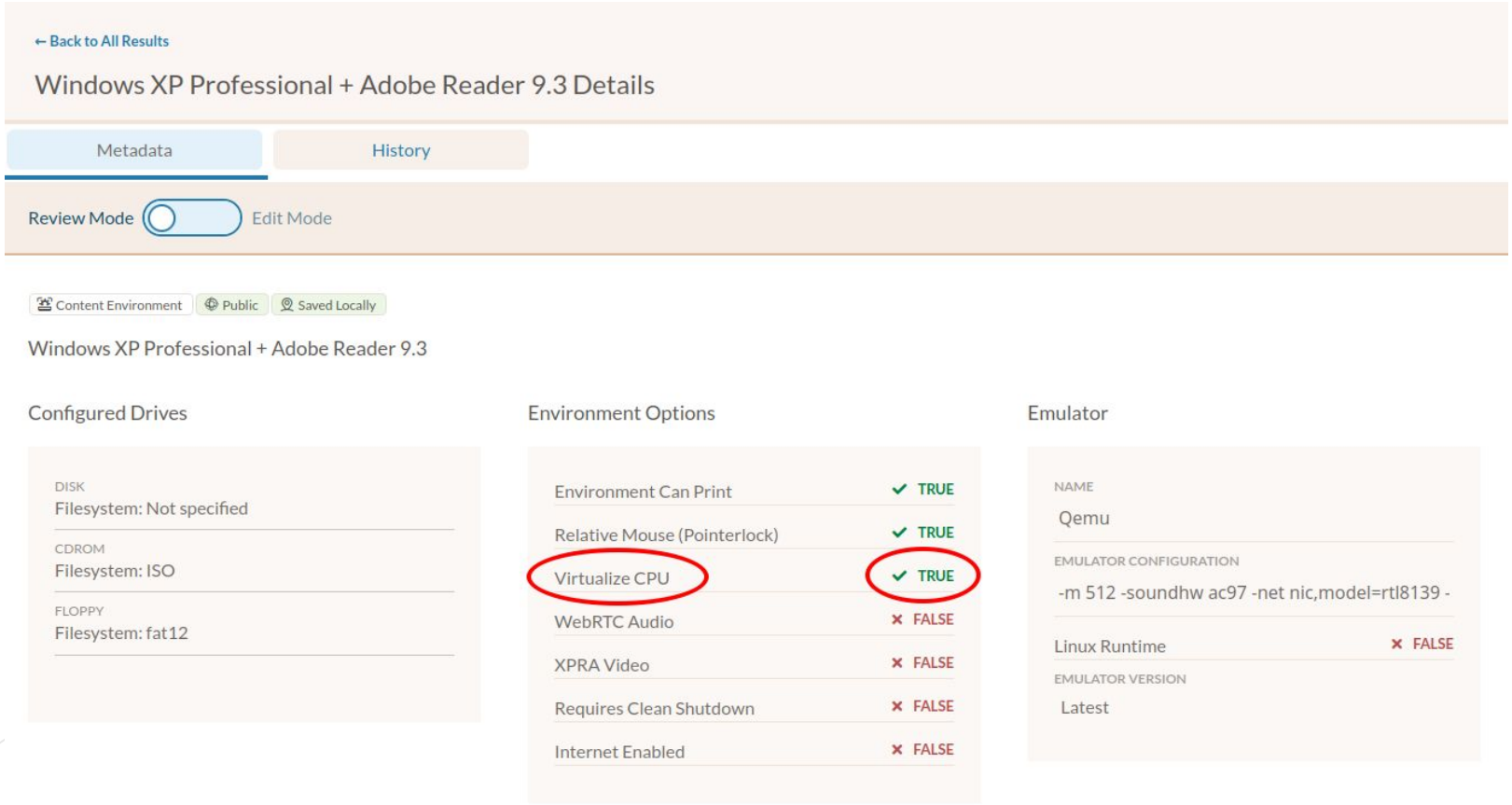


Commodore 64 Environment, running thanks to emulation

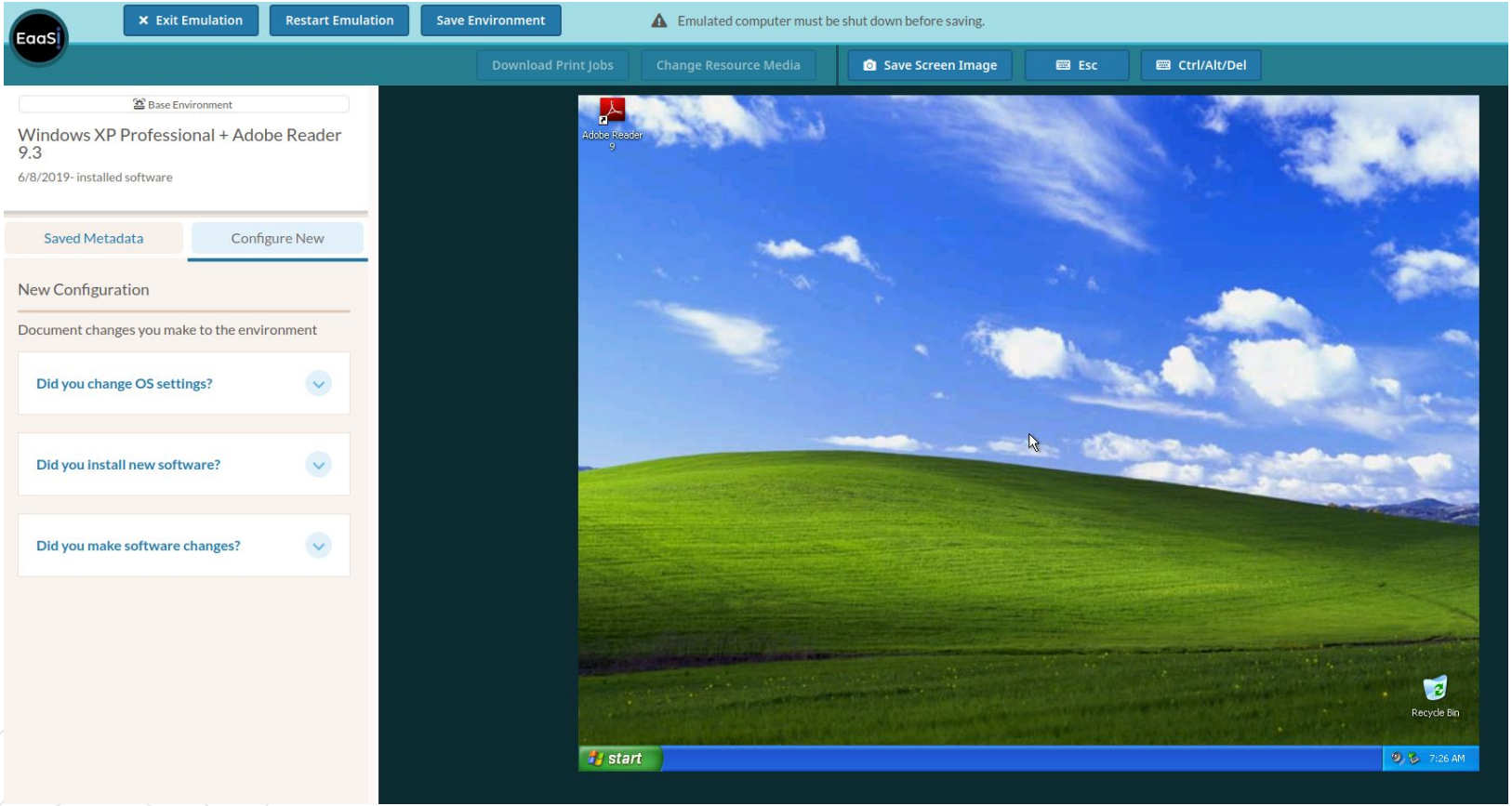
EaaSI Virtualization



- *Some* Environments can take advantage of virtualization
- Improves performance; the Environment will probably be more responsive, run programs more quickly
- But, only Environments running certain guest operating systems and software are compatible (must be KVM-compatible)
- EaaSI server must also be configured properly

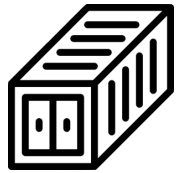


ex. Virtualization enabled on a compatible (Windows XP) EaaSI Environment

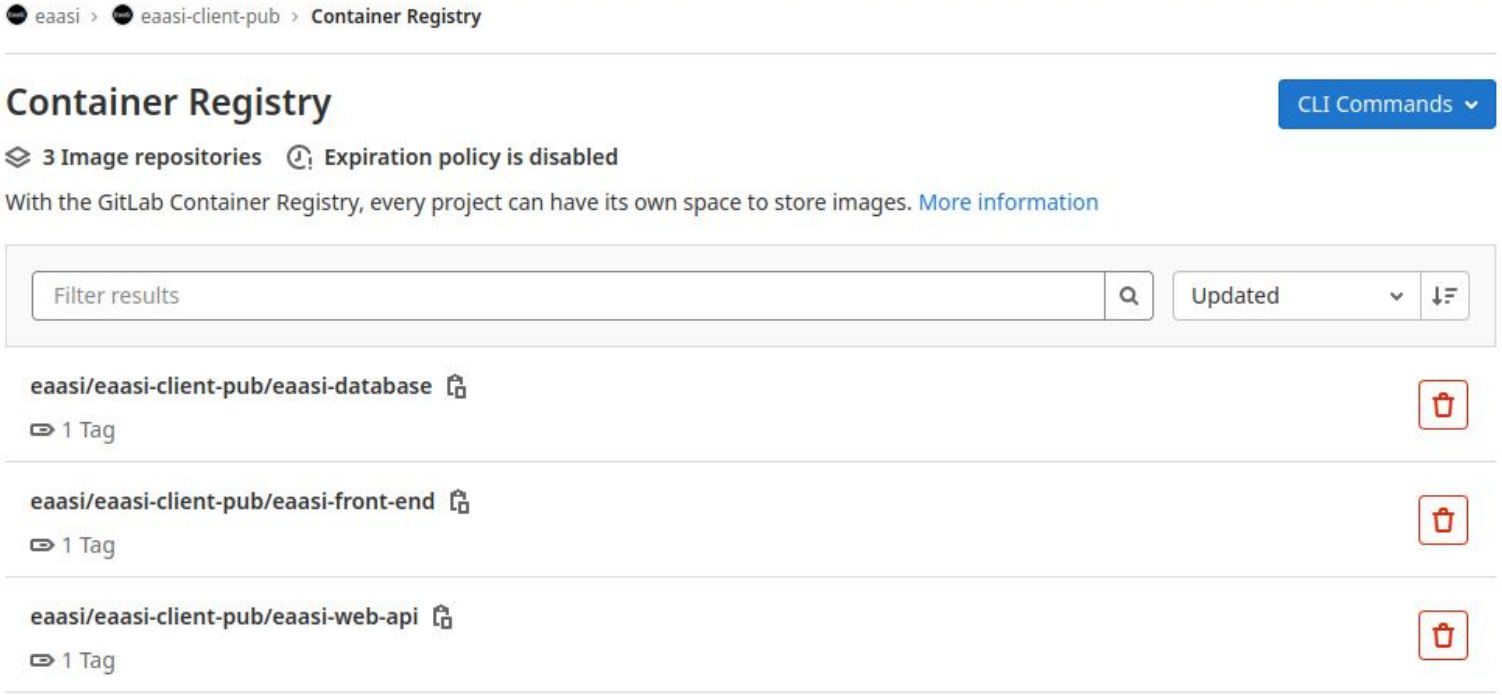


Windows XP + Adobe Reader 9.3 Environment boots and runs more quickly with virtualization than just emulation

EaaSI Containers



- The EaaSI stack itself is deployed and run via containers
- Keeps our strategy modular and flexible - quickly swap out our code when it requires updates
- Stays in line with widely-adopted tools
- EaaSI system administrators can scale up to their needs



GitLab container registry for components that make up the EaaSI Client (screenshot from summer 2020)

All Roads Lead to Emulation

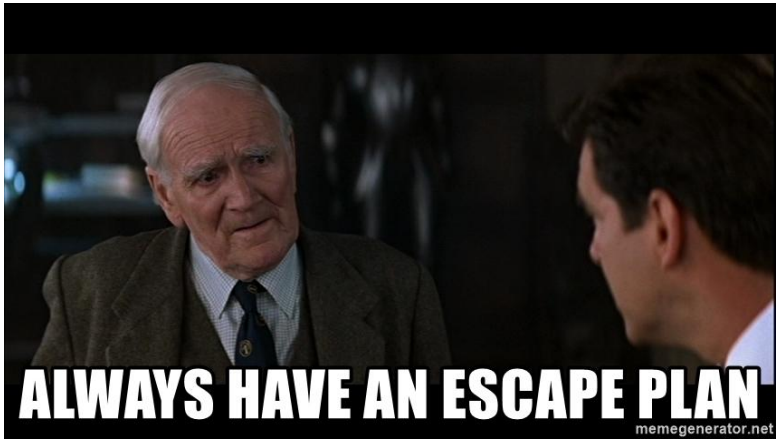
- Virtualization will *always* eventually break - it's inevitable
- Packaging up modern virtual machines as Environments will help to emulate them later
- Time - and software development cycles - are not on our side

What is the servicing timeline for a version (feature update) of Windows 10?

Edition	Servicing timeline Released first half of year (H1)	Servicing timeline Released second half of year (H2)
Windows 10 Enterprise Windows 10 Education Windows 10 IoT Enterprise	18 months from release date	30 months from release date
Windows 10 Pro Windows 10 Pro Education Windows 10 Pro for Workstations Windows 10 Home [2]	18 months from release date	

"Microsoft Lifecycle FAQ", 2021-07-23:
https://docs.microsoft.com/en-us/lifecycle/faq/windows

Exit Strategy



- Currently using Docker but keep containers as platform-agnostic as possible
- Virtualization is a bonus; Environments can **always** fall back to full emulation

Credits

- Training Module written and designed by Ethan Gates, Software Preservation Analyst, Yale University Library
- All photos, screenshots, and videos recorded by Ethan Gates
- Icons sourced from The Noun Project
- EaaSI program of work sponsored by the Alfred P. Sloan Foundation and the Andrew W. Mellon Foundation, hosted by Yale University Library

Yale
Principle Partner

ALFRED P. SLOAN FOUNDATION
Sponsor

THE ANDREW W. MELLON FOUNDATION
Sponsor