DevOps for Unreal Engine

🚀 make the builds go nyoom 🚀

REDPOINT

Welcome!

- I'm June (She/Her)
- I make plugins for Unreal Engine
- I do a huge number of Unreal Engine builds across multiple products, platforms and configurations
- Plus I need to run automation and Gauntlet tests for all of this
- Therefore, I need a very good DevOps pipeline for Unreal, so I can actually sustain all of this development as one person

REDPOINT

How this presentation is structured

- This presentation will be an *iterative* journey as we keep introducing more DevOps tools to solve problems and make our lives easier.

- This means: you can adopt as little or as much of this as you want, in the order presented, and get benefits along the way

- Each new tool solves more problems and lets us build and develop our Unreal Engine projects faster!

- Everything here is free and open source

Unreal Engine Scripts

What are you using to build today?

- UnrealBuildTool?
- AutomationTool?
- BuildCookRun?

Problems:

- Complex command lines with lots of flags
- Builds stuff in sequence, no running of jobs in parallel
- No tolerance for transient environmental failures
- Build failed? Have to start the whole thing over again which can cost a lot of time
- How do you run your automation or Gauntlet tests?

REDPOINT

Introducing: Unreal Engine Scripts

- PowerShell scripts that work on Windows and macOS
- Define your builds in JSON
- Specify automation tests and Gauntlet tests in JSON
- Support for different build variants (“distributions”)

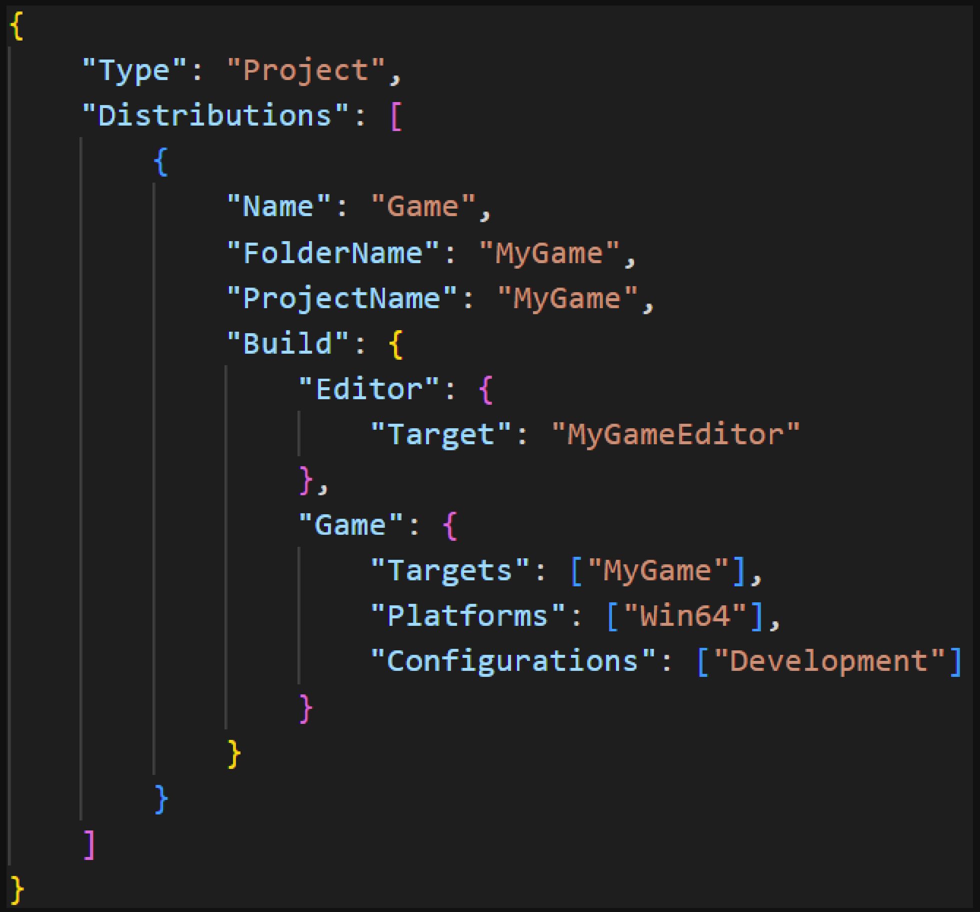
Do a build with:

```
.\BuildScripts\Build.ps1 -Engine 5.1 -Distribution Game -ThenTest -ThenDeploy
```

REDPOINT

What does the JSON look like?

- Create a file called BuildConfig.json in the root of the repository
- Assuming your project is at:
    - MyGame/MyGame.uproject
- Probably the simplest definition you can end up with



```json
{
    "Type": "Project",
    "Distributions": [
        {
            "Name": "Game",
            "FolderName": "MyGame",
            "ProjectName": "MyGame",
            "Build": {
                "Editor": {
                    "Target": "MyGameEditor"
                },
                "Game": {
                    "Targets": ["MyGame"],
                    "Platforms": ["Win64"],
                    "Configurations": ["Development"]
                }
            }
        }
    ]
}
```

Other features of Unreal Engine Scripts

- Project builds support running Gauntlet & custom tests
- Plugins builds support running automation & custom tests
- Build for Client and Server configurations as well
- Can package your plugins for Marketplace Distribution and check compliance with Marketplace rules
- Can deploy games out to Steam for you, and/or run custom deployment scripts
- Format.ps1 to automatically format your C++ code with clang-format
- Deals with a *ton* of environmental issues like file locks, mutexes, page file errors and other nonsense that can cause temporary failures, and will automatically retry for you

REDPOINT™

Where can I get it?

https://src.redpoint.games/redpointgames/unreal-engine-scripts

More instructions on how to install and use it in your project can be found on that page.

Benefits so far

- Simplified build, test and deployment of your games and plugins
- More reliable builds locally and on our build servers

(don't worry, this list will get longer as we go through the presentation)

Will this build things in parallel yet?

- Not quite, but it will in a moment
- Powered by BuildGraph
- But we currently only have one “node” (our local machine)
- To build in parallel, we need a build server and multiple build machines

GitLab

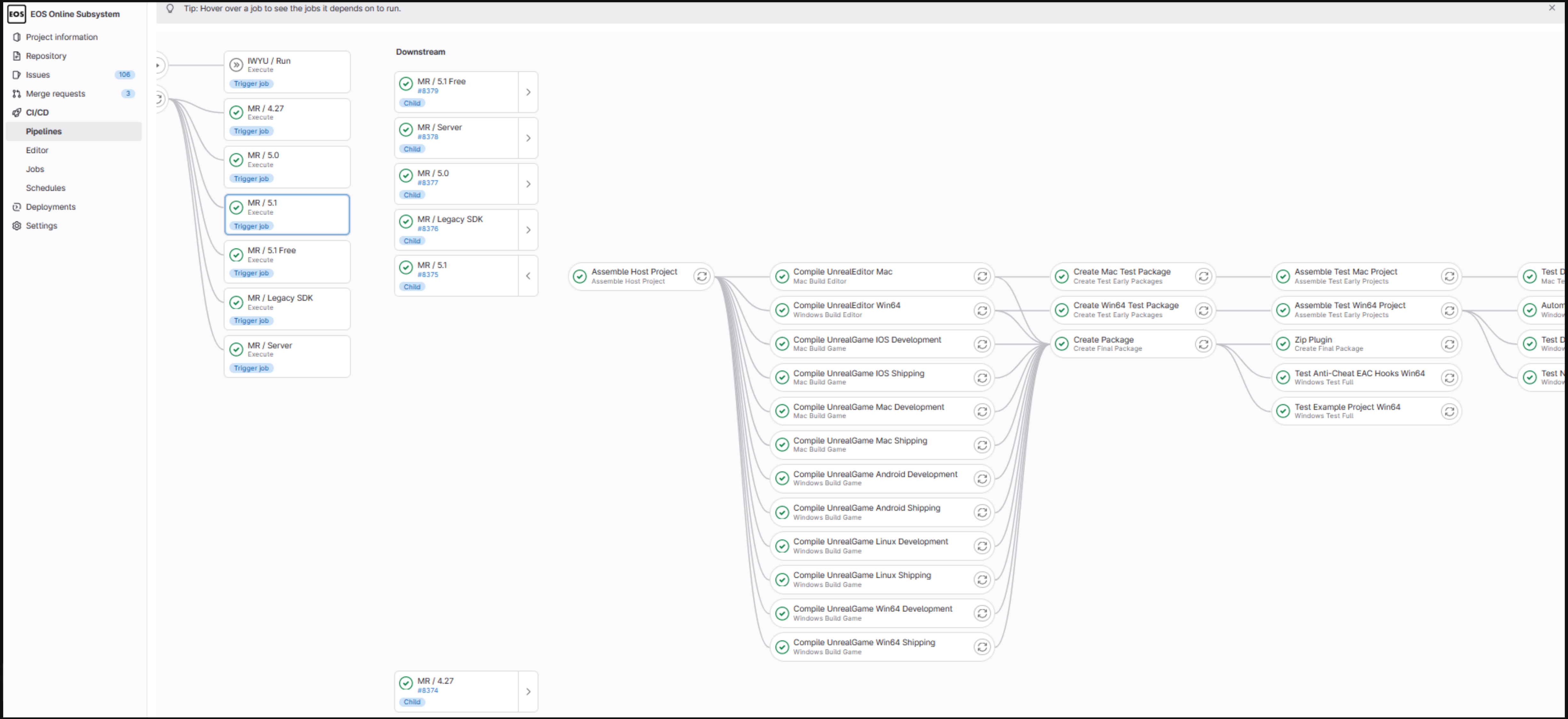
Technically any build server will do

- You don’t specifically need to use GitLab
- But GitLab is the only build server that Unreal Engine Scripts currently has out-of-the-box support for
- Unreal Engine Scripts has to know how to *translate* the BuildGraph graph into build jobs on the build server
- We use GitLab so that’s the support we’ve written, but you could add support for Jenkins or TeamCity or any build server that supports dynamic build pipelines, with just a little bit of PowerShell

GitLab is open source though

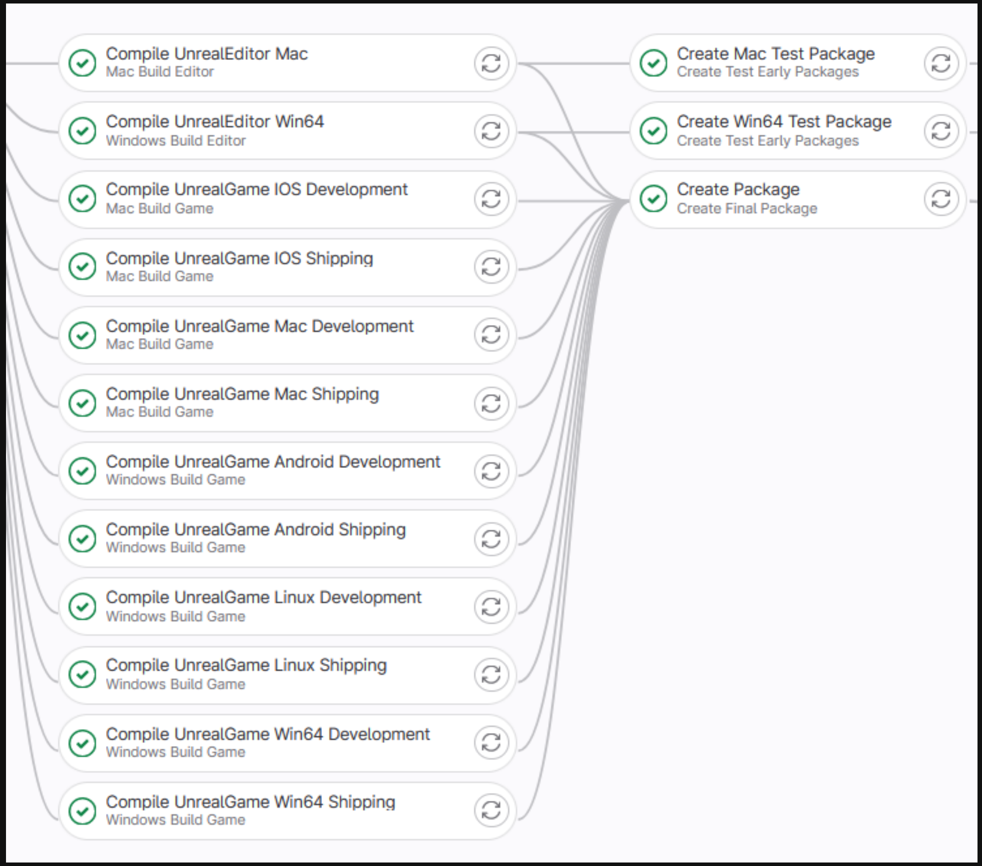
- This used to be an easier sell before they heavily limited their SaaS offering
- If you have 5 or fewer people on your team, Free SaaS offering might work for you
- Above 5 people it gets expensive fast though, so I would recommend self-hosting GitLab if you can
- I self-host in Kubernetes, but there are much more simple deployment options available where you just need a Linux box somewhere
- Self-host instructions for Ubuntu: https://about.gitlab.com/install/#ubuntu

What does this get us?



What does this get us?

- Run massive amounts of jobs in parallel across all our build servers
- True graph-based builds, so we are never blocking a build job that should be able to start immediately
- Just need to register build agents with GitLab
- And then use Unreal Engine Scripts


Compile UnrealEditor Mac
Mac Build Editor
Compile UnrealEditor Win64
Windows Build Editor
Compile UnrealGame IOS Development
Mac Build Game
Compile UnrealGame IOS Shipping
Mac Build Game
Compile UnrealGame Mac Development
Mac Build Game
Compile UnrealGame Mac Shipping
Mac Build Game
Compile UnrealGame Android Development
Windows Build Game
Compile UnrealGame Android Shipping
Windows Build Game
Compile UnrealGame Linux Development
Windows Build Game
Compile UnrealGame Linux Shipping
Windows Build Game
Compile UnrealGame Win64 Development
Windows Build Game
Compile UnrealGame Win64 Shipping
Windows Build Game
Create Mac Test Package
Create Test Early Packages
Create Win64 Test Package
Create Test Early Packages
Create Package
Create Final Package

Setting up shared storage

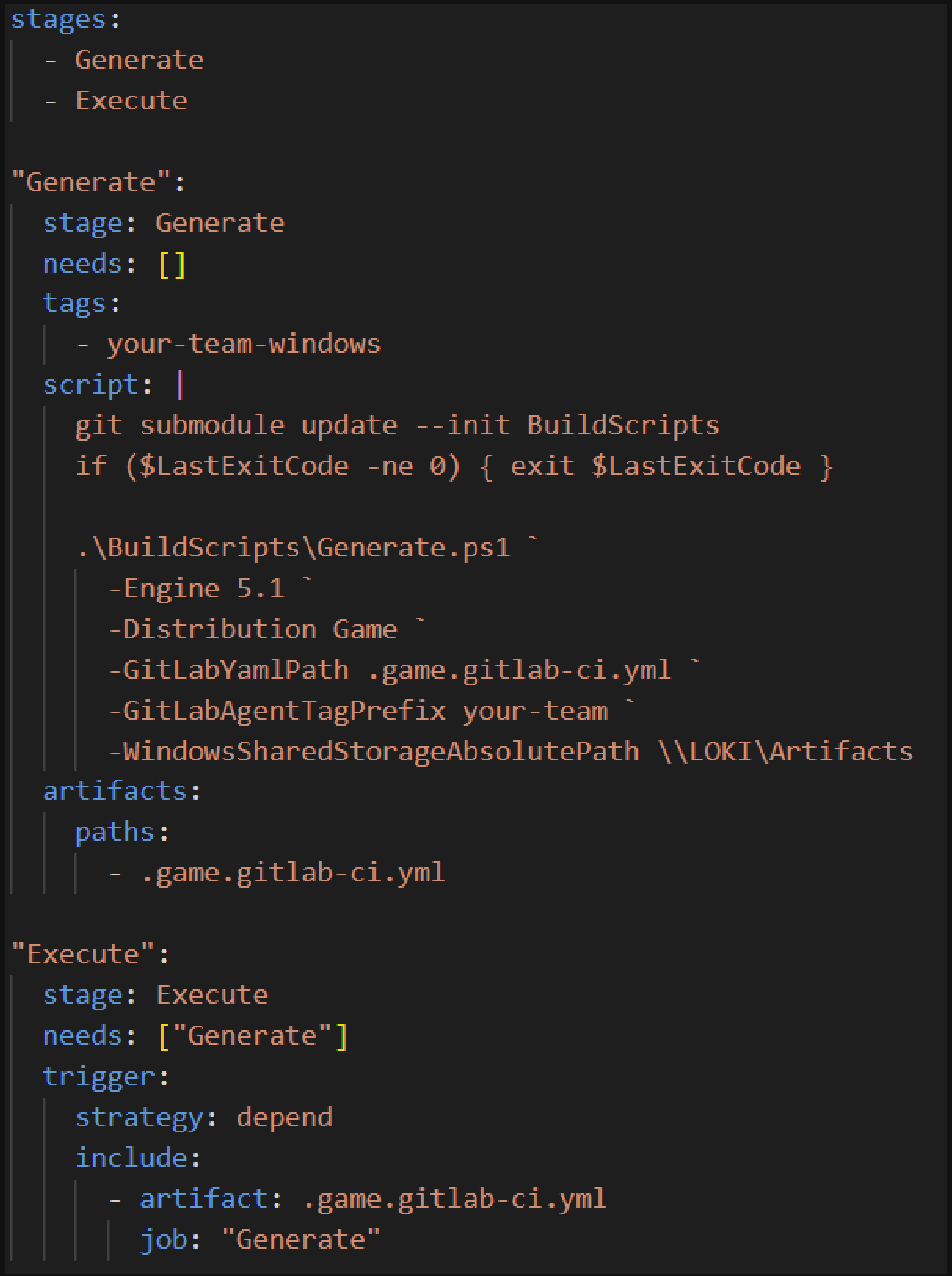
- Before we can use this yet, you’ll need to have some “shared storage”
- This is a normal Windows share on some computer on your network
- Does not matter where it is, as long as all build agents can access it, and it has enough disk space to store build artifacts
- We’ll pretend like you’ve set up a read-write Windows share at \\LOKI\Artifacts for the rest of this section.

How do I use it?

- Set up your .gitlab-ci.yml file so the build server calls:

```
.\BuildScripts\Generate.ps1 …
```

- Then tell the build server to start another build using the generated .game.gitlab-ci.yml file
- Uses the exact same BuildConfig.json file as your local builds
- In this example, your Windows build agents should be registered in GitLab with the tag “your-team-windows”



```yaml
stages:
  - Generate
  - Execute

"Generate":
  stage: Generate
  needs: []
  tags:
    - your-team-windows
  script: |
    git submodule update --init BuildScripts
    if ($LastExitCode -ne 0) { exit $LastExitCode }

    .\BuildScripts\Generate.ps1 `
      -Engine 5.1 `
      -Distribution Game `
      -GitLabYamlPath .game.gitlab-ci.yml `
      -GitLabAgentTagPrefix your-team `
      -WindowsSharedStorageAbsolutePath \\LOKI\Artifacts
  artifacts:
    paths:
      - .game.gitlab-ci.yml

"Execute":
  stage: Execute
  needs: ["Generate"]
  trigger:
    strategy: depend
    include:
      - artifact: .game.gitlab-ci.yml
        job: "Generate"
```

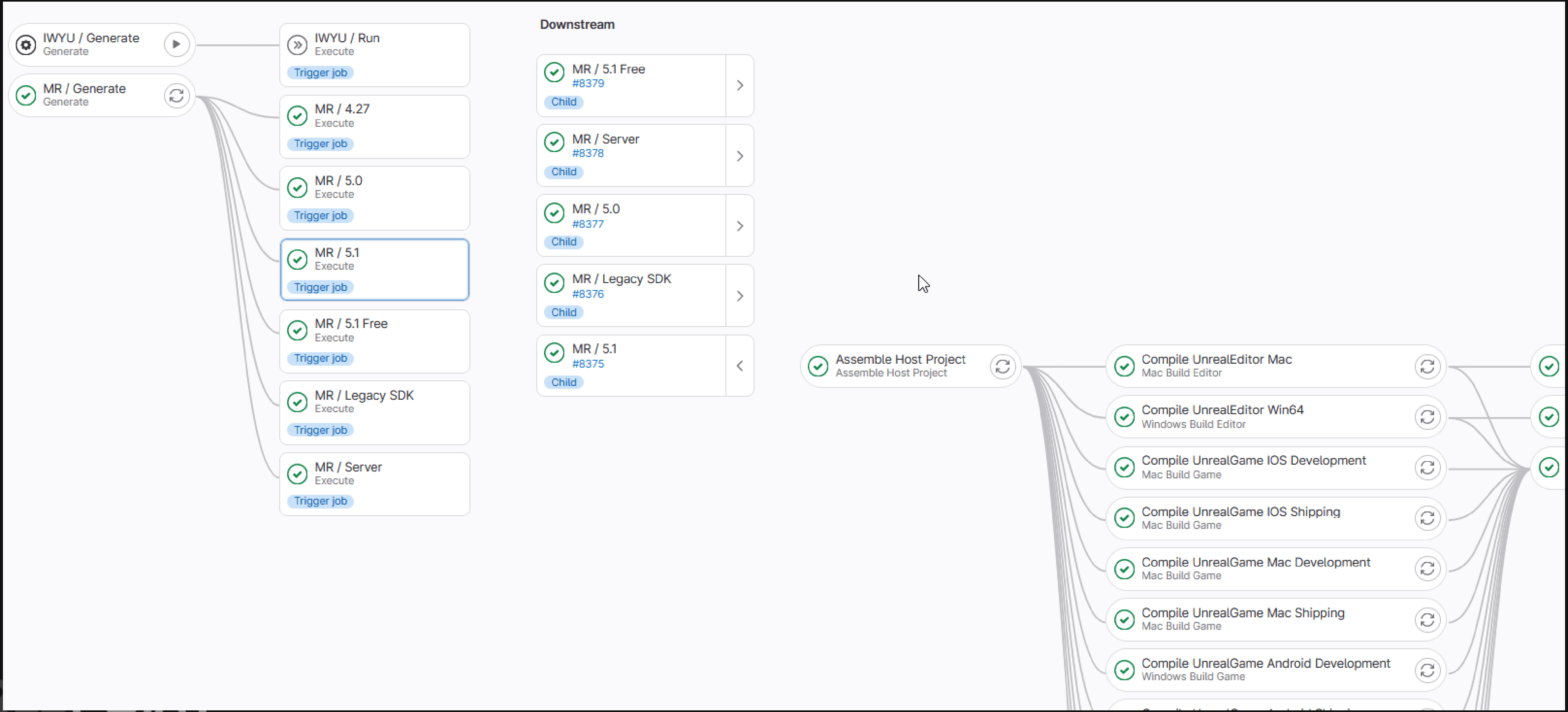
Benefits so far

- Simplified build, test and deployment of your games and plugins
- More reliable builds locally and on our build servers
- Massively parallelised builds via our GitLab build server
- Retry individual build steps via the GitLab UI if they fail

Io Build Monitor

REDPOINT

Problem: GitLab UI sucks for very complex builds

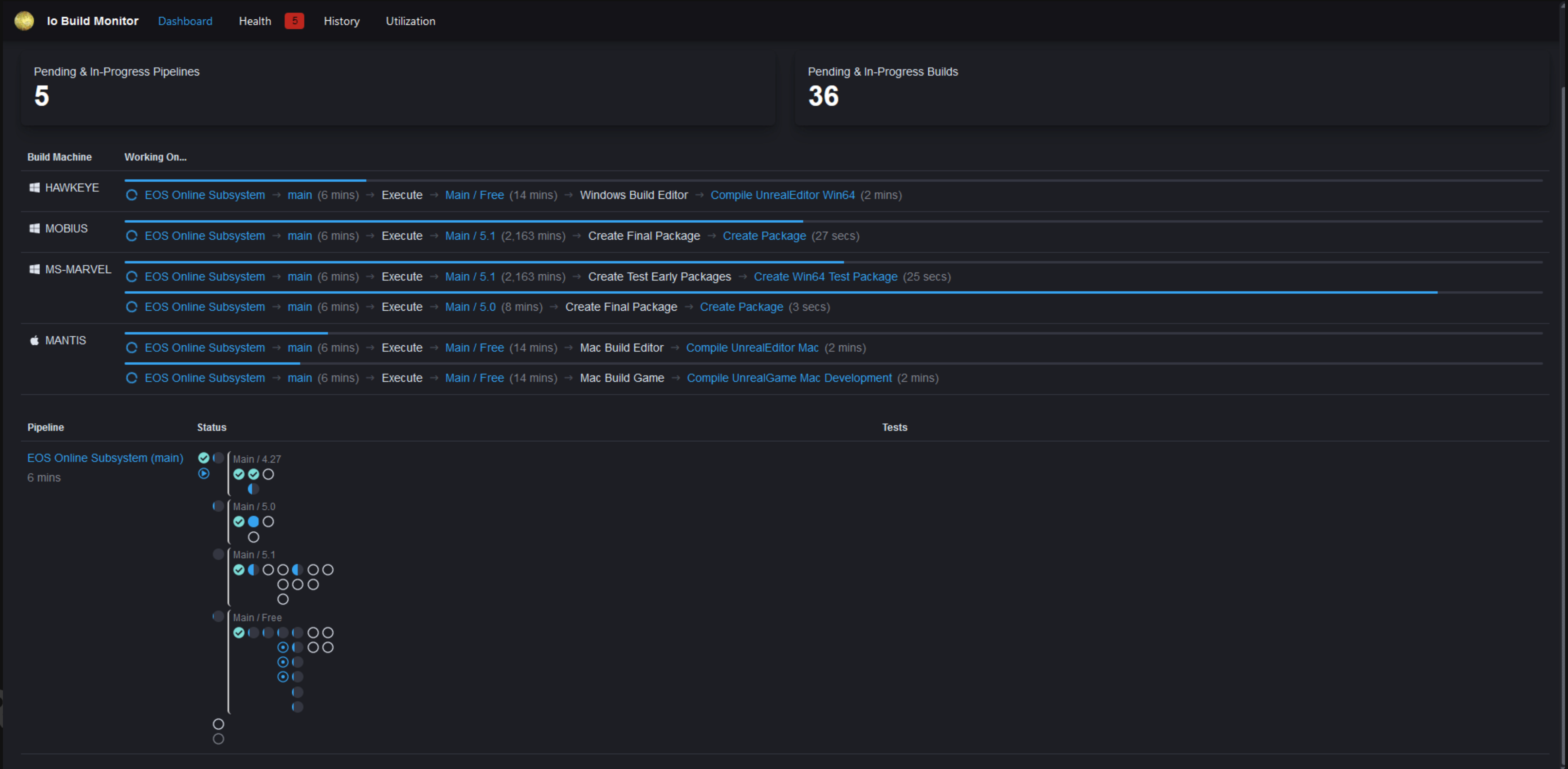


Problem: GitLab UI sucks for very complex builds

- Have to click to expand build pipelines to see individual job failures
- Can’t see what all the build machines are currently working on (no overall view)
- No progress or estimated time for jobs to complete
- No visibility of test progress

REDPOINT

Io is a better build monitor for GitLab



Io Build Monitor Dashboard Health 5 History Utilization

Pending & In-Progress Pipelines
5

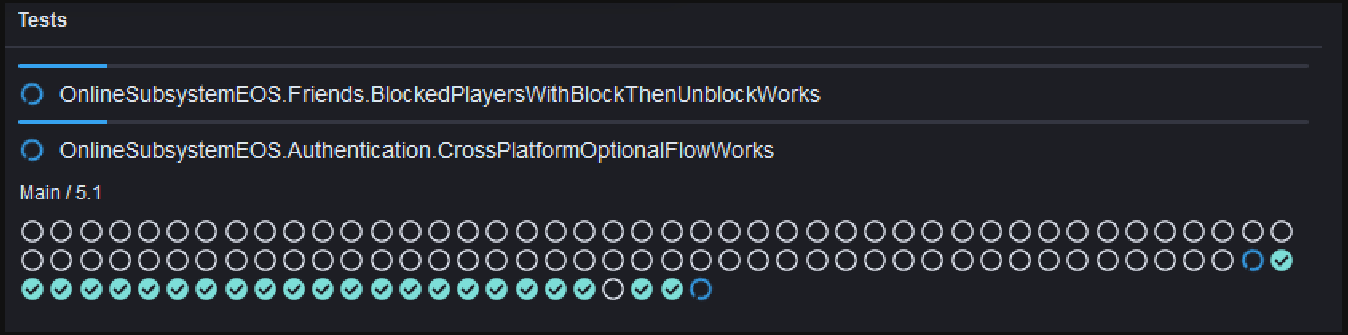
Pending & In-Progress Builds
36

Build Machine	Working On...
HAWKEYE	EOS Online Subsystem → main (6 mins) → Execute → Main / Free (14 mins) → Windows Build Editor → Compile UnrealEditor Win64 (2 mins)
MOBIUS	EOS Online Subsystem → main (6 mins) → Execute → Main / 5.1 (2,163 mins) → Create Final Package → Create Package (27 secs)
MS-MARVEL	EOS Online Subsystem → main (6 mins) → Execute → Main / 5.1 (2,163 mins) → Create Test Early Packages → Create Win64 Test Package (25 secs)
	EOS Online Subsystem → main (6 mins) → Execute → Main / 5.0 (8 mins) → Create Final Package → Create Package (3 secs)
MANTIS	EOS Online Subsystem → main (6 mins) → Execute → Main / Free (14 mins) → Mac Build Editor → Compile UnrealEditor Mac (2 mins)
	EOS Online Subsystem → main (6 mins) → Execute → Main / Free (14 mins) → Mac Build Game → Compile UnrealGame Mac Development (2 mins)

Pipeline	Status	Tests
EOS Online Subsystem (main) 6 mins	Main / 4.27 Main / 5.0 Main / 5.1 Main / Free	

Io is a better build monitor for GitLab

- View progress and ETAs!
- See what all the build servers are currently doing
- Quickly see all downstream builds without having to click through
- View the status of automation tests as they run:



Tests

OnlineSubsystemEOS.Friends.BlockedPlayersWithBlockThenUnblockWorks

OnlineSubsystemEOS.Authentication.CrossPlatformOptionalFlowWorks

Main / 5.1

Where can I get it?

https://src.redpoint.games/redpointgames/io-build-monitor

Some guidance on how to install it is on that page, but you will need to have a Kubernetes cluster and container registry to use the setup scripts that already exist.

You can run it outside Kubernetes if you like; it just needs Redis and PostgreSQL.

Benefits so far

- Simplified build, test and deployment of your games and plugins
- More reliable builds locally and on our build servers
- Massively parallelised builds via our GitLab build server
- Retry individual build steps via the GitLab UI if they fail
- Easily see what all our build servers are doing
- View all our in-progress build jobs, pipelines and automation tests, with time estimates

UEFS

What problems do we still have?

- We can only run one Unreal Engine build per machine at a time, due to the global mutex
- We have to manually install new Unreal Engine versions on all our build agents when we need them
- Installing or updating Unreal on each machine is *slow*, 100,000+ files for it to extract
- Multiple UE versions = lots of disk space used

Is there a better way to package and install Unreal?

- What do we want?
- Single file that contains an Unreal Engine install
- No need to extract, just want to mount the file in-place
- We can do this with VHD disk images!

REDPOINT™

Building an Unreal Engine package

- Get UE installed via the launcher on one machine
- Use UEFS to make a packaged version of it:

```
uefs build --dir "C:\Program Files\Epic Games\UE_5.1" --pkg ue-5.1.vhd
```

- Then we can mount that package wherever we want with UEFS:

```
uefs mount --dir "C:\UnrealEngine\5.1" --pkg ue-5.1.vhd
```

Wait, does that mean... ?

```
uefs mount --dir "C:\UnrealEngine\5.1_A" --pkg ue-5.1.vhd
uefs mount --dir "C:\UnrealEngine\5.1_B" --pkg ue-5.1.vhd
uefs mount --dir "C:\UnrealEngine\5.1_C" --pkg ue-5.1.vhd
```

- 3 fresh copies of Unreal Engine 5.1
- UAT mutex is *per engine installation*
- Which means we can run a build job in each of these folders, in parallel
- Solved problem: We can only run one Unreal Engine build per machine at a time, due to the global mutex

REDPOINT

How do we get these packages around?

- Lots of build servers? Still need a way to sync these VHDs to every machine
- Could just put them on a network share
  - But underlying disk reads would be slow (going over network)
  - And we’d need to hard-code a path to our specific server in all our build jobs
  - This solution does not scale well

Can we use a container registry?

- Container registries are used by Docker/Kubernetes to version container images
- e.g. Docker Hub is a container registry
- Can we store our VHDs in there and get a versioned tag?
- Yes, turns out you can store whatever you like in container registries (doesn't have to be a container image)

Push a versioned ref to your container registry

```
uefs hash --pkg ue-5.1.vhd
(copy ue-5.1.vhd and ue-5.1.vhd.digest to \\LOKI\UnrealEngine)
uefs push --pkg \\LOKI\UnrealEngine\ue-5.1.vhd --tag
registry.yourteam.com/team/unrealengine:5.1 --ref \\LOKI\UnrealEngine\ue-5.1.vhd
```

- Hash the package locally first (necessary for versioning)
- Push a reference saying "the file is at \\LOKI\UnrealEngine\ue-5.1.vhd" into the registry
- Could we store the actual VHD in the registry? Yes, but this is slower for local network scenarios because the pull would happen over HTTP/S instead of a network share
- So typically you will push refs instead of directly storing the VHD in the container registry

Push macOS versions into the container registry

- We can also build packages for macOS using UEFS:

```
uefs build --dir "/Users/Shared/Epic Games/UE_5.1" --pkg ue-5.1.sparseimage
```

- And push them in much the same way

- When the image is pulled, UEFS is smart enough to pick either the VHD or sparse image based on the current operating system

- So you can use the same tag "registry.yourteam.com/team/unrealengine:5.1" on all build jobs on all build agent operating systems and get the right result

REDPOINT

How do we get packages from the registry to the build agents?

- Can pre-pull with "uefs pull"
- But actually not necessary
- Unreal Engine Scripts support using an engine by UEFS tag, and will *automatically pull and mount the package* for the build job as needed:

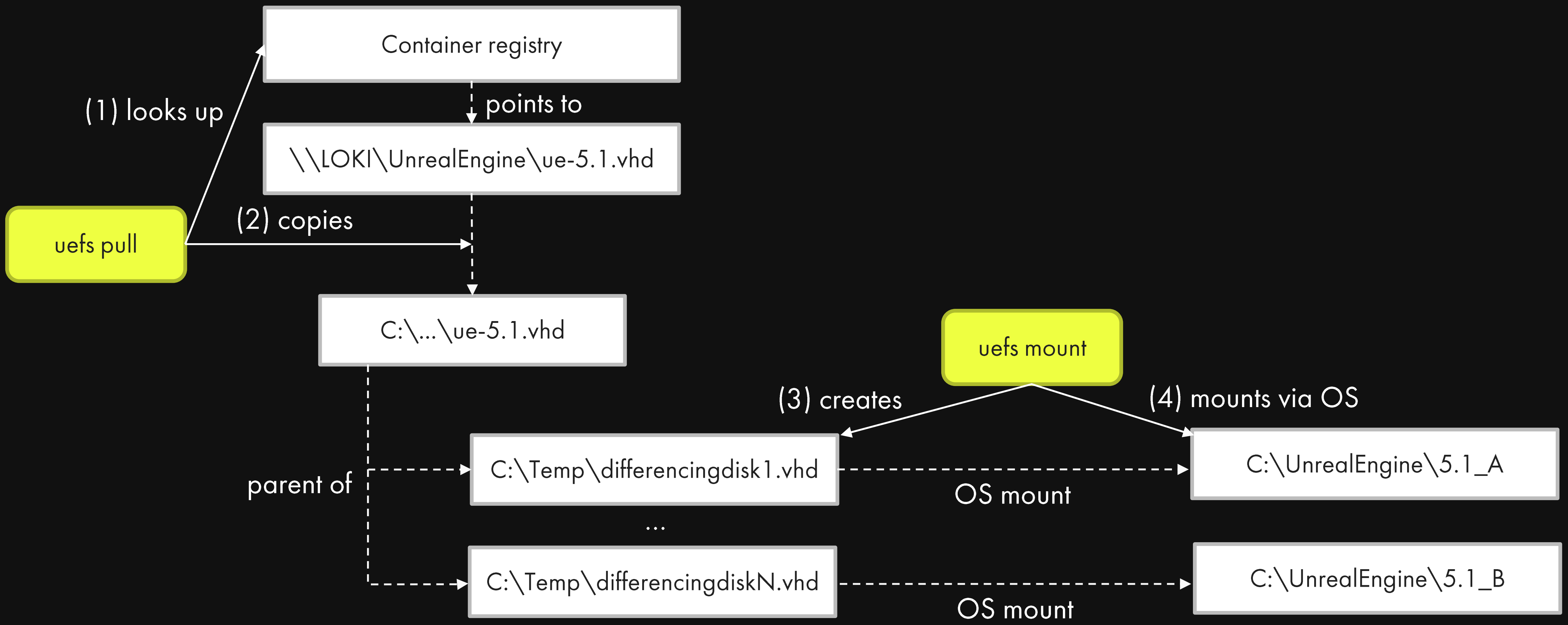
```
.\BuildScripts\Build.ps1 -Engine uefs:registry.yourteam.com/team/unrealengine:5.1 -
Distribution Game -ThenTest -ThenDeploy
```

- Works with Generate.ps1 as well, and thus works on GitLab build agents

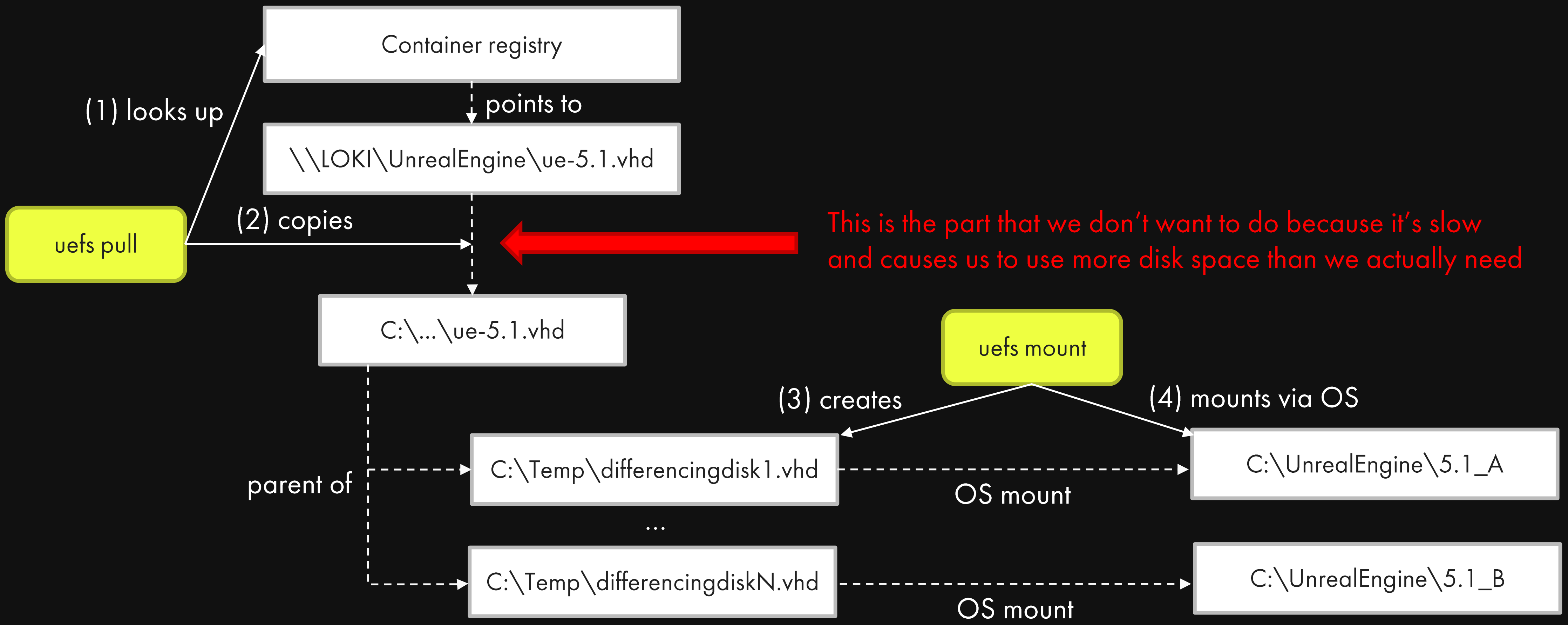
Still need to solve that storage problem though...

- Lots of Unreal Engine packages still means lots of disk space used
- But do we actually *need* all of Unreal Engine?
- Turns out, no, we don't. To launch the editor, you only need about 3gb out of ~100gb locally on disk.
- Let's virtualise the VHD storage itself.

How it works prior to storage virtualisation

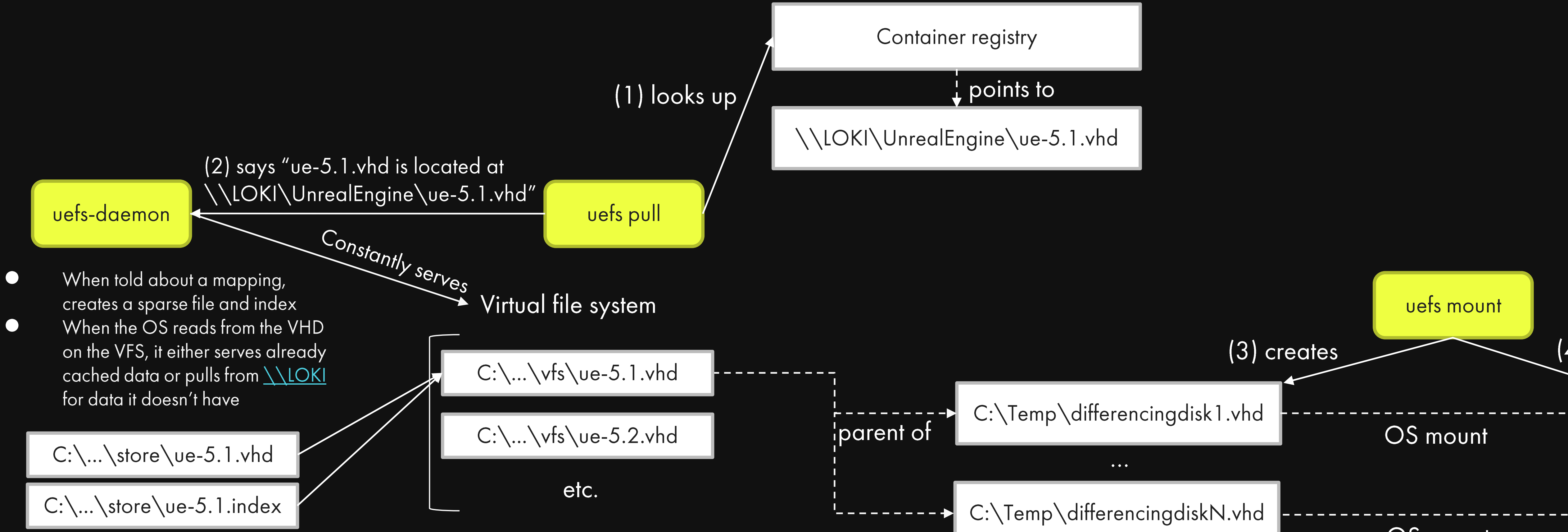


How it works prior to storage virtualisation


Container registry
points to
\\LOKI\UnrealEngine\ue-5.1.vhd
(1) looks up
uefs pull
(2) copies
This is the part that we don't want to do because it's slow and causes us to use more disk space than we actually need
C:\...\ue-5.1.vhd
uefs mount
(3) creates
(4) mounts via OS
parent of
C:\Temp\differencingdisk1.vhd
OS mount
C:\UnrealEngine\5.1_A
...
C:\Temp\differencingdiskN.vhd
OS mount
C:\UnrealEngine\5.1_B

REDPOINT

Virtualise that VHD



- When told about a mapping, creates a sparse file and index
- When the OS reads from the VHD on the VFS, it either serves already cached data or pulls from \\LOKI for data it doesn't have

REDPOINT

Virtualise that VHD

- Data is fetched *once* in 128kb blocks and stored locally
- Pay the “network access” penalty only once for each 128kb block of data in the package
- Near native performance for already cached blocks
    - ~1 min 30 seconds to launch editor from package with no cached blocks
    - ~15 seconds to launch editor from package where editor has launched before
- uefs pull and uefs mount are now *instant*, which makes build jobs start on any machine *instantly*
- We only store the data we’re actually using for each engine install

REDPOINT™

What about source engine builds?

- We’ve solved performance and storage problems for pre-built packaged Unreal Engines (like the ones we get from the launcher)
- Can we virtualise source-based engines as well? What does that look like?
- Yes! We can use the same virtual file system infrastructure to virtualise a Git commit.

Mounting engine Git commits

```
uefs mount --git-url git@github.com:EpicGames/UnrealEngine --git-commit 5.1 --
dir "C:\UnrealEngine\5.1-source"
```

- Still has to do an up-front fetch of commit history
- But no need for "git checkout", all files and directories virtualised
- And no need for "GitDependencies", all binary blobs fetched from CDN on demand
- Can mount as many copies of the source engine as you want, with no additional storage cost
- If the commit has been previously fetched, mounting a Git commit is *instant* (no checkout time)

Where can I get it?

https://docs.redpoint.games/uefs/

Docs are a little out of date (you need WinFsp instead of Dokany for virtualised packages).

But overall very simple to install and easy to use.

Lots of active development happening to support future work (which we'll get to in a moment)

Benefits so far

- ...
- Easily see what all our build servers are doing
- View all our in-progress build jobs, pipelines and automation tests, with time estimates
- Version engine packages for Windows and macOS via container registry
- Run multiple Unreal Engine builds at the same time on the same machine, by mounting the same engine multiple times in different directories
- Need to undo changes to an engine? Just unmount and remount to get a fresh copy
- Mount Unreal Engine instantly over the network on Windows, with no performance hit
- Mount engine commits from GitHub without fetching deps or checking out on Windows
- All integrated into Unreal Engine Scripts so it works “out of the box”

REDPOINT™

Future Work: Kubernetes

REDPOINT

State of things right now

- We're in a pretty good place, with performant builds
- Still some environment issues we can't deal with for parallelisation
  - E.g. when Unreal tries to write to stuff in AppData
- Occasionally build servers can go haywire and need fixing/restarting
- Still need to keep all build dependencies up-to-date on each build machine (like VS)
  - Manageable with scripts, but still not great for ensuring each build server has the exact same environment

State of things right now

- Adding new build machines is not fast or trivial
    - Need to add the build machine as a GitLab runner
    - Wait for setup/update scripts to install everything like VS, console kits, etc which can take hours
    - Can't really run builds while this is happening in case a build job get scheduled on a partially configured machine
- Deploying updates to build machines have much of the same problems – can't do builds while things are updating

What we want to have

- Treat the host hardware as just Windows + CPUs/GPUs/memory etc.
- Encapsulate all of our dependencies like Visual Studio into container images
- Do our builds inside containers
- Let Kubernetes do the hard work of scheduling build jobs across all our machines
- Adding a new build machine to the cluster should be simple and fast
- Still need all of our storage virtualisation to work though...

RKM: Redpoint Kubernetes Manager

- Turns out setting up Kubernetes on Windows nodes is hard and complex
- I made a tool called RKM to make setting up Linux+Windows Kubernetes clusters dead simple
- And by simple I mean you download it and then run at an Administrator/root prompt:

```
rkm
```

- Get it from https://src.redpoint.games/redpointgames/rkm
- Need to run it on a Linux machine first, then all the Windows machines after

REDPOINT

The dream 💭

- Instantly start massively parallelised builds in Kubernetes, with any Unreal Engine version and any project from Git with no download or checkout time

- All dependencies (like VS) snapshotted and stored in container images so they can't differ across build machines or randomly break from updates

- Full isolation of environment so it is impossible for build jobs to conflict with each other on the same machine

- Parallelise automation tests and Gauntlet tests across machines via Kubernetes

- Maybe even live test game builds in your browser via Pixel Streaming?

Things that are preventing the dream *(and why this is future work)*

- ❌ Storage virtualisation for VHDs works, but storage virtualisation for Git commits does not due to Windows Container issue #335.
- ❌ You need to specifically run Windows 11 Build 22000. Newer versions of Windows 11 are broken as per Windows Container issue #322.
- ❌ Pulling container images from the registry is still slow (Windows base image is 9GB). Only happens once per build machine which is OK in a bare metal setup, but not viable in a cloud setup. Can’t use storage virtualisation for container images themselves until containerd issue #8206 is solved.

Wrapping Up

REDPOINT

Benefits now & into the future

- Simplified build, test and deployment of your games and plugins
- More reliable builds locally and on our build servers
- Massively parallelised builds via our GitLab build server
- Retry individual build steps via the GitLab UI if they fail
- Version engine packages for Windows and macOS via container registry
- Run multiple Unreal Engine builds at the same time on the same machine, by mounting the same engine multiple times in different directories
- Need to undo changes to an engine? Just unmount and remount to get a fresh copy
- Mount Unreal Engine instantly over the network on Windows, with no performance hit
- Mount engine commits from GitHub without fetching deps or checking out on Windows
- All integrated into Unreal Engine Scripts so it works "out of the box"

Future:

- Versioning dependencies like Visual Studio via container images
- Using containers to ensure isolated build environments
- Simplified build agent management and distributed builds via Kubernetes

REDPOINT

That’s all folks!

- Unreal Engine Scripts: https://src.redpoint.games/redpointgames/unreal-engine-scripts
- GitLab: https://about.gitlab.com/install/#ubuntu
- Io Build Monitor: https://src.redpoint.games/redpointgames/io-build-monitor
- UEFS: https://src.redpoint.games/redpointgames/uefs
- RKM: https://src.redpoint.games/redpointgames/rkm

These slides will be available online at junerhodes.au/history soon.

Bug me with questions on Mastodon: @hq@mastodon.social