A two-part technique for efficiently scaling build and test automation

Josh Nixdorf
Software Engineer – Electronic Arts

GAME DEVELOPERS CONFERENCE® 2012 MARCH 5-9, 2012 WWW.GDCONF.COM

Why is scaling automation difficult and why do I care?

- QA effort is best spent checking for authenticity and fun, not looking for software bugs.
- Developers could write unit tests or run extensive at-desk testing, but their time is best spent writing and tuning features.
- Developer-side automated builds and tests help ensure that everyone's time is focused on what they do best and helps maintain work-life balance

Note that this is not a reason to not right unit tests, that's still a good idea, but there is a balance

Ultimately all of these things help to save money and time.

At EAC we automate everything, almost nothing that is seen outside the dev team was created at someone's desk. As a courtesy we don't check-in and then leave. Which leads to our first problem. People want to know when can I go home after my check-in? (We see few check-ins after 4:00pm) The check-in to test result time has a big impact on when check-ins stop happening. Seems like an easy problem to fix, just keep things fast…

Presentation

1. How to scale?
2. Problems with Section 1
3. Solutions to Section 2
    1. Technique 1: Virtual machines
    2. Technique 2: Virtual drives
4. Our Implementation

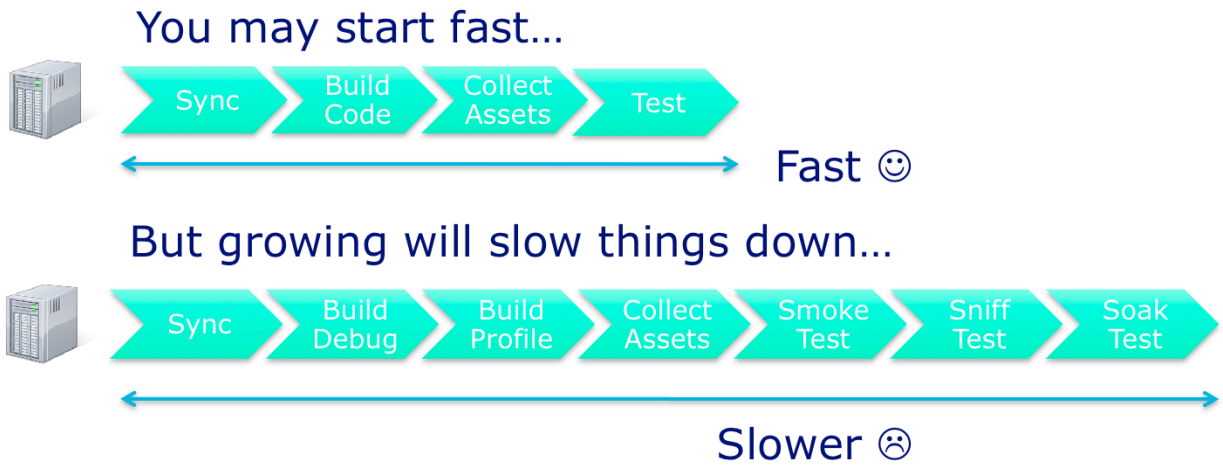
Context: How big is a project?

- Average project (~15 projects total)
  - 1-2 active branches
  - 100+GB/branch
  - 10-25 million LOC (includes libraries)
  - 2-3 Platforms (PS3, Xbox360, PC)
  - ~4 build configurations per platform
  - Executable size: ~150MB
  - Game size: ~8 GB (full DVD)
  - Average clean compile time: 15-30 min/config
    - Without grid building: 1 hour+/config
    - Without grid or pre-built libraries: 10 hours+/config
  - 40 commits/day (early production), 100+commits/day (peak production)
  - **Old Commit-to-test-result duration: ~60-120 minutes**

I am not bragging about these numbers, I don't brag about them, I complain about them.

These are just for the central managed teams at EAC in Burnaby. Volumes/time differ in other parts of the company. In general, ours are actually pretty good as we iterate on the same titles year after year, so we tend to invest a lot in our tools and processes.

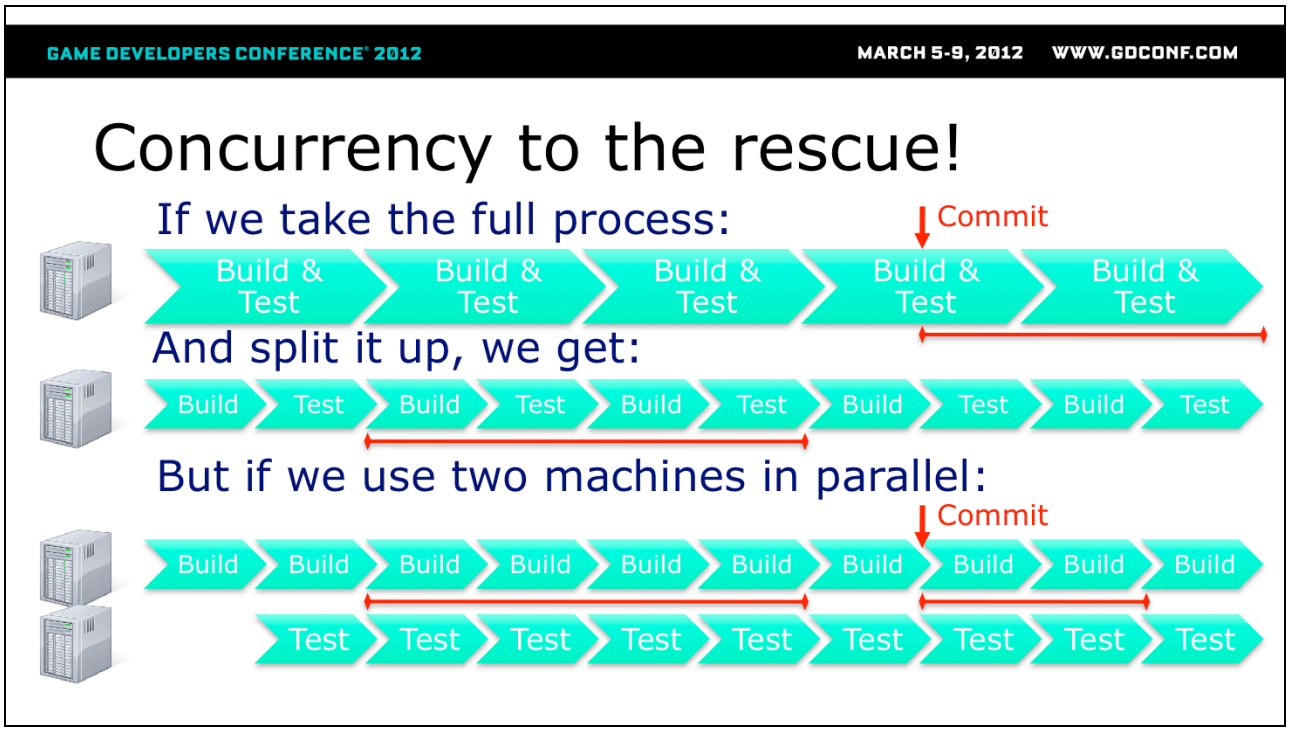
Section 1: How do we go wrong?



Best case = one machine that does all automation processes (usually not practical)

Requires: Few daily check-ins, number of check-ins < ($time_{workday}$/$time_{result\ duration}$), Few builds (Configs/SKUs/ platforms), Few tests

1st plan is to use faster servers to offset the growth. However, this approach gets prohibitively expensive as you grow.



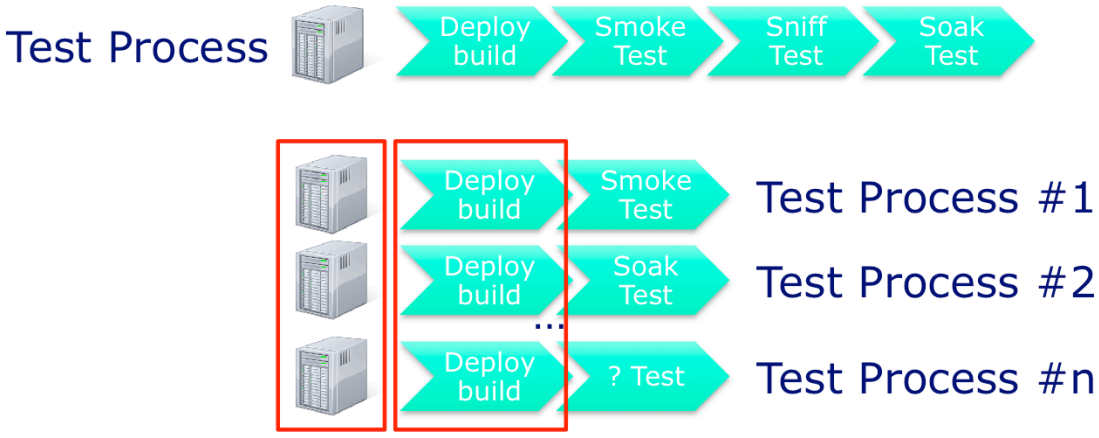
Serial: Most efficient use of hardware (should always be busy) and less hardware required but slow

Parallel: Potentially less efficient use of hardware and more potentially required (Test environments may be cheaper than build environments) but process should complete faster

At this point, we highly parallelized the system and we've reduced the check-in to result time. Plus we've significantly increased the number of iterations that we can do in a day.

Why stop at just two machines?

Take the existing processes and split them further



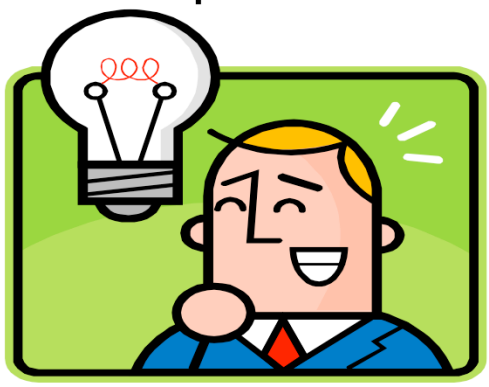
Doing something simple like separating out the tests into its process/environment has a big impact.

This is also a good split point because builds and tests often have a 1 to many relationship. So this split enables the tests to scale without any deeper dependency on the builds than that they exist.

Parallelization is obvious, so why am I even presenting… because parallelization is expensive, in this example we now need 4 times the hardware

Throw more machines at the problem?

- But new problems arise:
  1. Need more hardware
     - Perpetually looking for budget
  2. Redundant work
     - How many times do we copy the same build?
- Solutions:
  1. Virtualize the hardware
  2. Virtualize the build volume

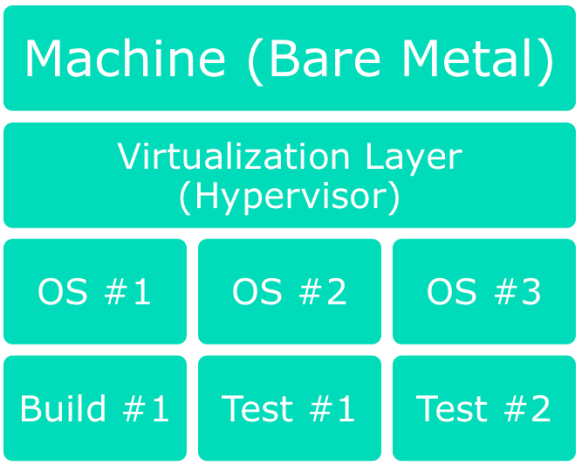


Why not just buy more hardware?

- Physical hardware consumes resources, many of which also don't scale well
  - Space
  - Cooling
  - Electrical
  - Networking
  - **IT**
- Oh yeah, and cost
- Virtualization let's us pretend to have hardware

Also, consider the overall support cost to all groups involved such as IT: Physical repair, removal / install, software installs (OS, compilers and tools)

What is Virtualization?



1. Handy for scaling
2. Easy to create OSs
3. Encapsulated Environments
4. Varying hardware specs

In case you aren't familiar with it…

It's using software to simulate a physical device. In this example a piece of software known as a hypervisor is installed on the bare metal (or in an existing OS) and the hypervisor simulates the existence of a new machine that can have a different hardware configuration (to some limits) and its own distinct OS.

It's particularly useful for scaling, since the only limiting factor is the resources afforded by the actual hardware.

There also really handy for encapsulating environments. You may not need to run 4 OSs at once, but for testing, you may need quick access to 4 differently configured environments, with virtualization you can do that all on the same box.

How do I know if it's right for me?

Analyze steps in the automation processes

Sync	Generate SLN	Compile	Link	Deploy assets	Package	Create disc	Publish
•Disk IO •Network speed •Repository server	•Build system •Dependency checker	•CPU •Grid size	•Available RAM	•Disk IO	•Disk IO •CPU •Grid size	•Disk IO	•Disk IO •Network speed

Using tools like Cacti and PerfMon, we can measure and track our worst bottlenecks

- Need more CPU ☹, but grid offsets CPU ☺
- Disk IO is a hindrance ☹
  - We'll solve this later

In the old days CPU was our biggest problem. When we want it built faster, we put faster CPUs in the machines. Some parts of the company continue to do this, but our server farm had 150 machines, its an expensive investment to keep things this as fast as possible. So we revised our build system and moved to grid based build. As a result the CPU speed is relatively less important as it is now split over many more machiens. We also adapted our compression and packaging step to leverage the grid.

Many of the other steps are serial though, on a build machine, this means that a 2 core machine performs almost as well as a 4+ core machine. For developers though we still want to have as many cores available so that they can still comfortably use their computer while compiling.

The next biggest bottleneck is local disk. Historically we have mitigated this by using 15,000 RPM drives (and have been looking at SSDs). Unfortunately they are often quite small (160GB) so its very difficult to use the same build machine for more than game. Developers don't' get 15k drives/SSDs but they are less impacted as they rarely need to package, create disks, or publish.

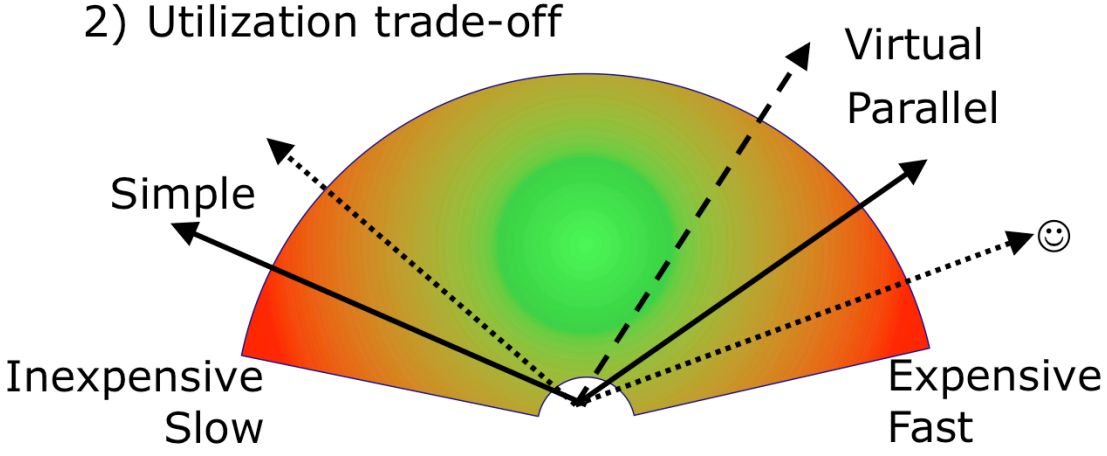
Interesting note: We've found we actually decrease build times up to 10% but syncing into a compressed folder. The processor is fast enough that the compressed folder doesn't appear to add any overhead during the syncing step, but the compressed folders require less disk reads so we see improved performance on SLN generation and build times.

Virtualization is not a silver bullet, you need to decide if virtualization fits for you. In our case we started by analysis of our automated processes. The discoveries made us realize that our hardware was under utilized. This was intentional, as mentioned we are obsessed with speed, so we want to have things leap into action as soon as there is new work.

This made virtualization a great option for us. We had spare capacity available for other processes. Note that this works best if you can load balance things so that they aren't all consuming all of the same resources at the same time. Virtualization may still be valuable for you even if you don't have much spare capacity.

So how is virtualization helpful?

1) Efficient resource utilization
2) Utilization trade-off

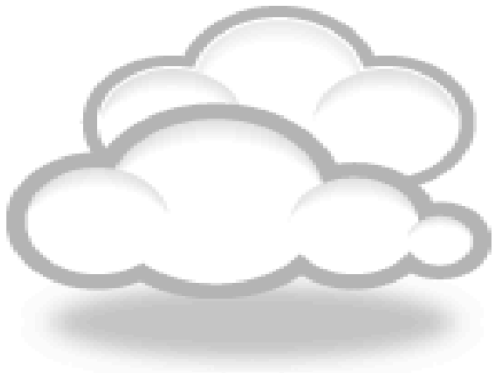


Virtualization allows us to avoid committing fully one way or another. We still have confines but it's a range instead of a point. You choose where you are on the curve.

With the same set of hardware virtualization may take you even further, because you are you can share resources. As long as they aren't over allocated it's like you have machines that you didn't pay for.

One more thing...

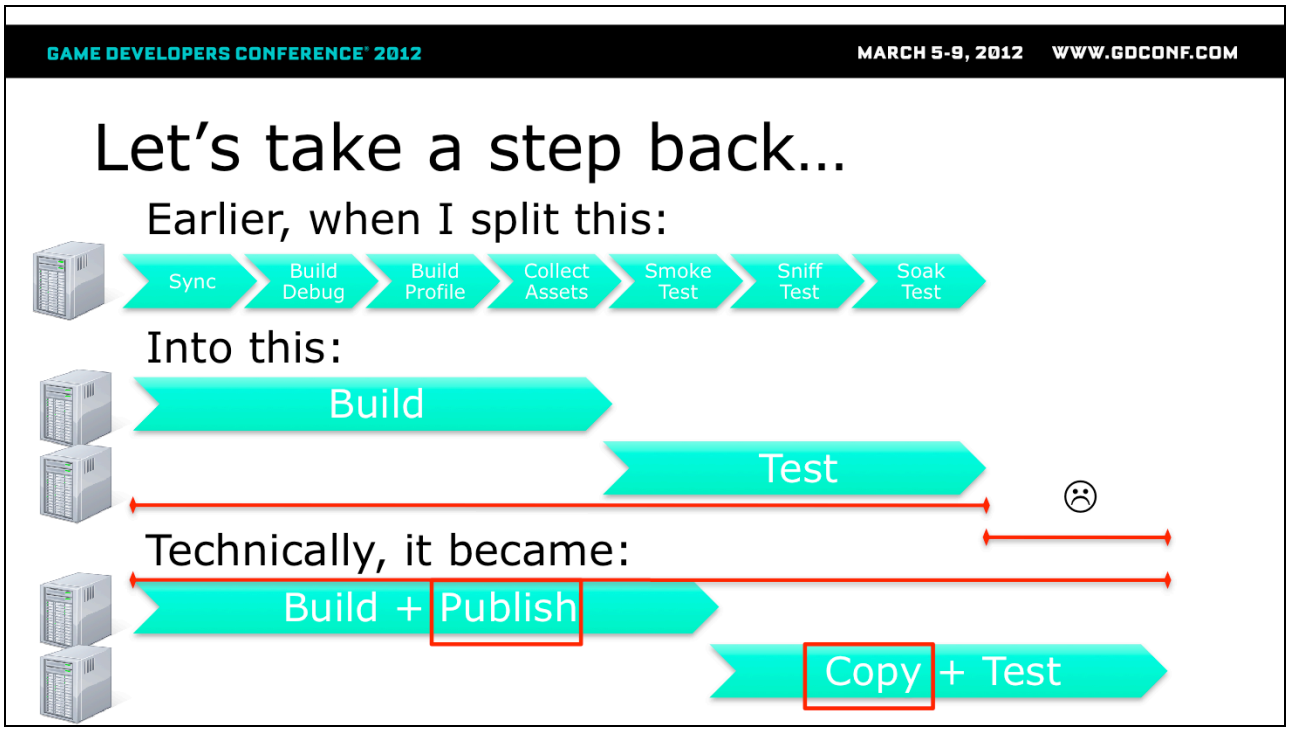
- Virtualization allows us to be 'elastic'
  - Need automation for a new branch?
- Almost like having your own 'cloud'



How many times have you heard (or said) "And we'll need all of automation for the new branch". We used to dread those words. Not anymore though.

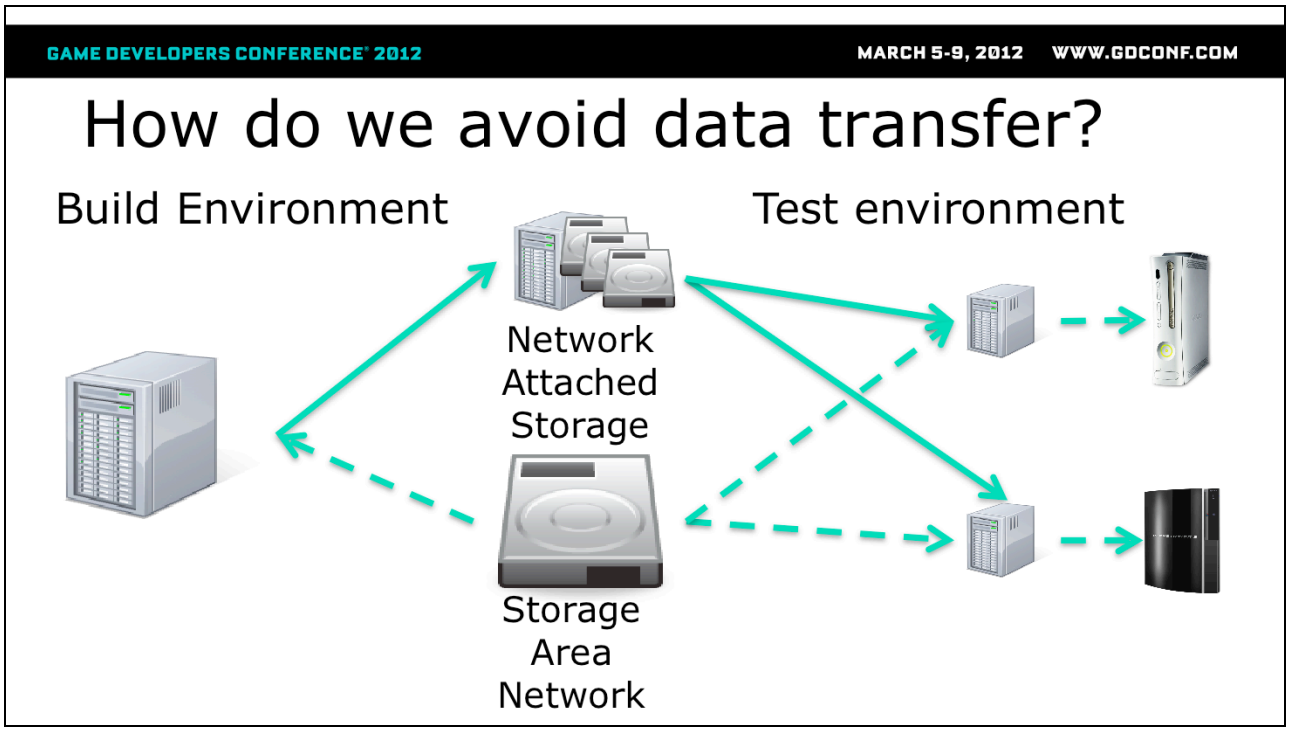
This ability to scale performance and the ability to encapsulate environments allows us to handle large rapid growth. We can adapt to most large changes, and if they become permanent that's when we can make the case to buy more hardware to reduce to durations again.

The major benefit of virtualization is that standing up new hardware means slowing things down a little rather than buying new hardware, which is especially useful for temporary changes (demo/PR/conference branches). Example, we can have 30 fast machines or 60 slows ones and we can switch between those choices very quickly.



So now we had a dial we could turn to adjust things, but could we make them faster?

Earlier we introduced these new steps to enable parallelization, publish and copy. As we've seen that was a good move, but it did potentially slow things down. I haven't mentioned yet how much time they added. That's because we don't actually have them in our process anymore, or at least we don't pay a time penalty for them anymore.

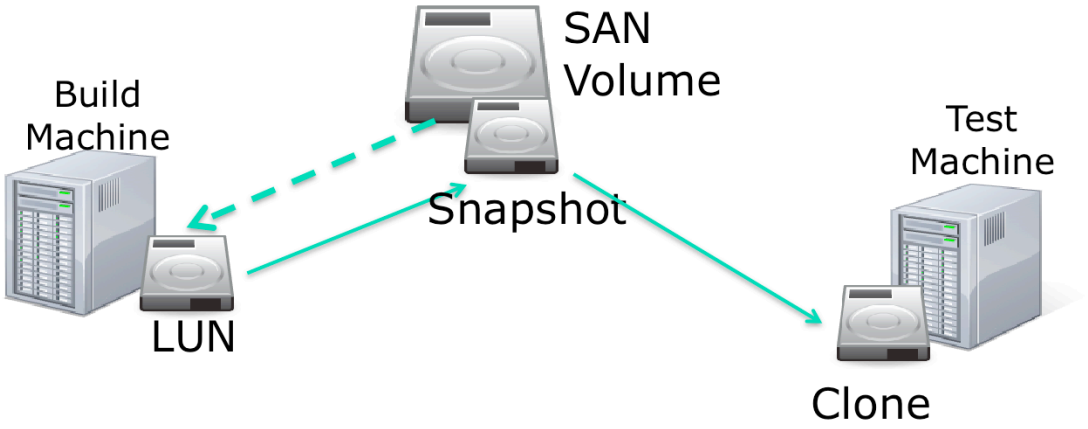


By just having the all of the data live on the network in the first place we avoid needing to copy it around. Granted this requires a storage area network and a revised infrastructure that allows this.

This is not technically streaming though it's helpful to think of it that way, but the point is true the data all resides on the network.

Note: that combined with streaming to the console we were able to eliminate all copying from our automation pipeline

How can that even work?



Since these aren't copies (they are deltas) they don't have any creation time

The SAN or filer, is basically a giant collection or disks. We can carve off virtual disks, called LUNs from the collection of storage. Those LUNs can then be assigned to any machines on the network. The machine then mounts that disk as a local disk using the iSCSI protocol (supported by all major OSs). To be clear, to the OS this disk is indistinguishable from a physical disk inside the device, this isn't just a mapped network drive.

That LUN clone will contain the entire source tree and everything that is build when the build machines does its thing. Once the build is ready to move on the build machine triggers the filer to take a 'snapshot' of that volume. The snapshot then becomes a point-in-time copy of the LUN as it was at that particular moment. What's interesting here is that the snapshot is instantaneous and, at least initially, takes up no space. This is because its really an indirection layer that the filer is now managing on that LUN. All news changes to the LUN will be written somewhere else and the filer will resolve all look-ups appropriately whether a machine is looking at the LUN of the snapshot.

In order to preserve the integrity of the snapshot data, snapshots can't actually be used directly. A snapshot can however be cloned and that clone is available for use. Same as a snapshot, a clone point in time copy of the original LUN except that the clone is writable. Any changes made to the clone start to incur storage penalties.

Also the same as a snapshot a clone has no creation time overhead because its just an indirection layer that the filer is handling for us. What is tricky though is that each build is now its own drive which needs to be dynamically mounted and unmounted from the test environment. This can be a bit tricky but suffice it to say that automating diskpart isn't too difficult and you need to disable system restore on the test machines.

Its also important to note that you can have as many clones as your system has space for.

How are deltas not magic?

LUN

0000	0101
1101	1010
1001	0111

Snapshot

0001	0101
1101	1011
1001	0110

Snapshot Delta

0001	
	1011
	0110

Snapshot

0001	0101
1101	1011
1001	0110

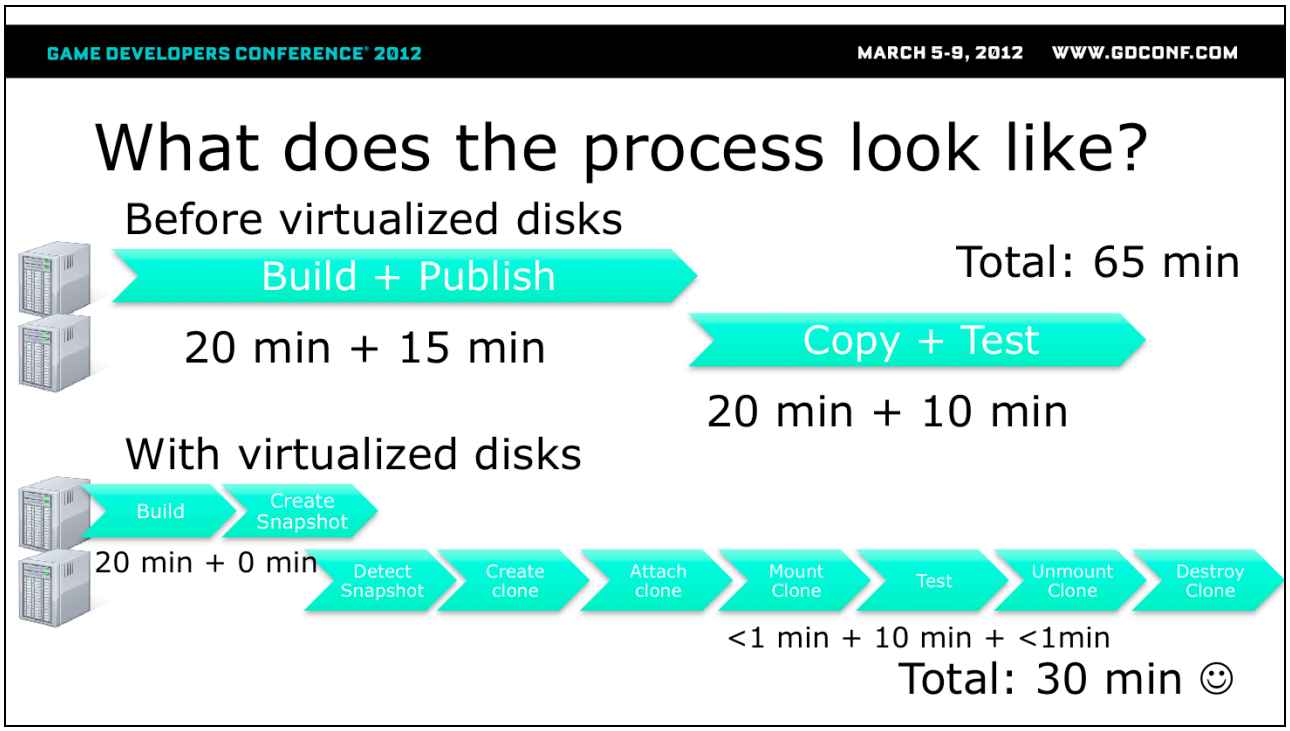
Clone

0000	0101
1101	1010
1001	0111

Clone Delta

0000	
	1010
	0111

The filer is able to keep track of all the changes as they are coming in so it's able to redirect them to the appropriate places



Build Process happens as usual, but instead of a publish of the build, we create a snapshot, which is just a call to a script, which established an RSH connection to the filer and runs the snapshot create command.

The test process runs a script which asks for the list of snapshots on the filer. Using that it detects the new snapshot instead of a published build, the test machine the runs a script which makes an RSH connection to the filer to run the clone creation command using the snapshot as a parameter it also configures the clone so that the test machine can use it (attach)

The iSCSI connection on the machine will automatically recognize the clone and attach it, a script is used to have diskpart mount the clone into the file system

The tests are run as usual. Once the test is over, another script has diskpart detach the clone from the machine, then another script established an RSH connection to the filer and runs the clone destroy command.

Sounds easy but tracking the LUNs, snapshots and clones can be a pain.

What kind of hardware does that?

- NetApp FAS3170 Active-Active Failover Cluster
  - 512GB Flash Cache per controller
  - Dual 10GigE Ethernet per controller
  - 144x450GB SAS disks
  - 64.8TB total raw storage
  - ~70k peak IOPS (good spinning disk is 0.4k IOPS)
    - **350 IOPS x 200 machines = 70k IOPS**
    - This also fixes our disk IO issues since we can peak higher than 350 IOPS (measured at 1k on a LUN)

This is the device that we use to run 140 servers, we tested (and prototyped) with many smaller devices and had success.

Our choice was based on the level of IO operations per second that the device that can handle and the storage capacity that we would get. We tried to find a device where all of our processes could hit their peak IO load simultaneously without crashing the device. We profiled our process and determined that our peak IO writes were about 350/second. With an estimated max of 70,000IOPS (we couldn't generate enough load to verify it). 70000/350 = 200 which was more than enough machines. If we take the 200 x 190GB disc space (150 source + 40 OS) we use up about 40TB of storage which also fits into the spec of the filer.

We actually saw performance increase just by moving the data off of local disk and on to the network, our new blades don't even have local disks

Pitfall #1: Do you know how to admin your filer?

- The longer clones live, the more deltas they store, the more space they take up
  - Forgetting to delete a clone will fill up the volume
  - Filers generally won't clean clones for you
  - Filling up the volume means no new snapshots/clones can be created
- Snapshots also contain clones
  - This creates a dependency nightmare
  - Snapshots become dependant on future snapshots
  - Snapshots with dependencies can't be deleted gracefully

Since a clone is technically a LUN in the volume, any snapshot will contain clones as well. In the default filer configuration, creating a snapshot while a clone is alive will create a dependency on the clone. This creates a forwards dependency on snapshots. Snapshots with dependencies can't be deleted. However, as long as at some point a snapshot is taken while no clones are active (say first thing in the morning, or after any period where the tests finish before the next build) this dependency free snapshot will be created and cleaning can occur from that snapshot backwards. Alternatively, (took us six months to try) there is a poorly named/documented option where the volume can be configured so that snapshots do not contain clones and this dependency is broken.

As mentioned snapshots are dependant on clones, so you've got to ensure your clones are deleted once you are done with them. Forgetting to do that will eventually cause the volume to run out of space and everything to stop (potentially catastrophically depending on your configuration). This situation can't easily be avoided. To mitigate it, at the end of the process we auto-deleted the clone, and at the beginning we try to delete the previous clone in case it snuck by.

Pitfall #2: What's in a name?

- Snapshots are of the entire volume not a single LUN
- The name determines what was ready when the snapshot was taken
  - LUN needs to identify what build this is for
    - game + codeline + build + platform
  - Snapshot needs to identify what changelist this build corresponds to
    - game +codeline + build + platform + changelist
  - Clone needs to identify who is using this snapshot
    - game + codline + build + platform + changelist + server

There may be many LUNs in a volume, so how do you know which one is useful?

Since the snapshot contains the name of the LUN, we know just by the name of the available snapshot which the useful LUN contained within. The clone name builds onto to the same naming scheme so that we can identify the matching snapshot if a clone doesn't get cleaned up.

Would definitely recommend building a separate system to manage these things, as named LUNs is a nuisance on the administration side.

Pitfall #3: Do you know who is watching your disks?

- Disable System Restore!!!
  - Windows is not fond of you using diskpart to add and remove drives that's its busy trying to backup for you
- Keep an eye on your anti-virus/asset management software
  - They may see the activity as suspicious

I can't stress enough that System Restore will mess up everything. Windows does not like volumes to be attached and detached and attached and detached over and over again. What's worse is that it will cause sporadic issues which are surprisingly hard to debug.

Results: Final infrastructure

Servers

- 40 Blade Servers
  - 2x6 core, 24GB RAM, no local storage
  - ESX 5.0, 3:1 VM ratio, runs build VMs
- 16 Old Desktops
  - 2x4 core, 16GB RAM, no local storage
  - ESXi 4, 8:1 VM ratio, runs Test VMs

VMs

- Build VMs (1 template):
  - 2 cores, 4GB RAM
  - Permanent 150GB iSCSi LUN
- Test VMs (1 template):
  - 1 core, 2GB RAM
  - Temporary 150GB iSCSI LUN Clone

The blades should be able to handle 5 VMs of the current spec that we are using. By using 3 we've allowed ourselves room to either add more VMs or increase the performance.

The old desktops for the test machines, are not ideal ;), and we have a current project to migrate them to blades, but since they are really just intended to provide a isolated test environment they don't need to be able to do anything but provide a stable windows host for our test automation platform.

Results: Automation Effort

Build Infrastructure

- ~150 different build configurations
- ~800-1000 builds/day during peak
- 1 engineer to manage part time
- Reduced from 120 blade servers to 40

Test Infrastructure

- ~200 different test suites
- ~1200-1500 test suite runs/day during peak
- 1 engineer to manage part time
- Created 150 test environments using no new hardware
- Prototyped virtualization project without spending any money
  - ESXi is free

With the money that we didn't spend on replacing 80 old blades we can easily buy the filer, as well we reduce our space usage in the data center as well as our carbon footprint. If we were doing this again we would probably have taken measurements of electrical draw, etc on the data center, as we likely have significant monthly savings by not having to power/cool those old blades as well.

Further Work

- Greater parallelization of the build process
  - Reduce the commit-to-test-result time further
- Improve elasticity of the virtual infrastructure
  - Makes branching automation really easy
- One-to-one ratio between check-ins and automation
  - Makes it easy to identify and roll-back commits that break the tests
- Snapshots for developers
  - Easier/cheaper to debug failures that are caught in automation but cannot be reproduced at desk

Greater Parallelization: basically we want to move the sync drive to its on LUN/VM and make clones of that for building, and then use clones of clones for the testing, this should allow us to reduce disc space use, and reduce our VM/project creation time

Elasticity: currently our VMs are created manually and are static, we want to move to dynamic VM creation, were need build/test requirements can be added and VMs are automatically created to adjust for them

Pre check-in verification: this is the ultimate situation, every changelist is checked individually. Last time we tried we hit massive scaling problems, but we hadn't mastered all of the dynamic creation and when we using an array of physical machines. If we can work out the ability to dynamically create and destroy VMs as needed we should be able to take another stab at this.

Snapshots for SEs: there is no reason that a developer couldn't map a clone to their local machine. This would be great for debugging as the clone has both the build and the source tree, can you can basically get access to everything needed for debugging without having to change your current workspace

The end

Key Takeaways

- Highly parallel processes are the only way to scale
- Virtualization is one option for keeping parallelization costs under control
- Snapshots are an excellent way to improve performance and enhance parallelization

Questions

- Thank you