

The Data Building Pipeline of Overwatch

David Clyde

Senior Software Engineer

Blizzard Entertainment

David Clyde

- Big Huge Games
 - MSSQL + Perforce custom asset pipeline
- Blizzard
 - Three different asset pipelines using various forms of sql and http.
- Generalist
 - Localization
 - Branching
 - Streaming
 - I wrote shaders once... a long time ago

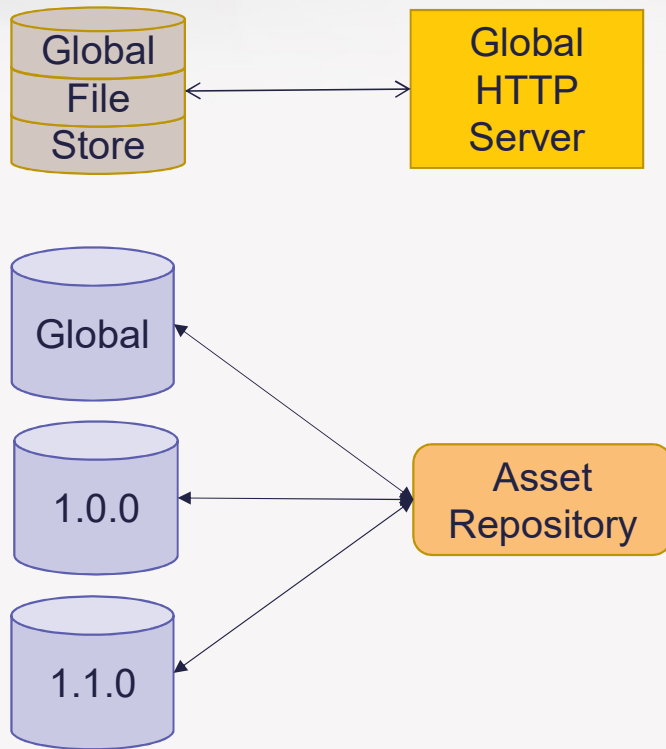
Goals for Overwatch pipeline

- Pull latest build/content in < 5 minutes
- Changes seen instantly in engine
- Local changes fully impact the game immediately
- Share your changes with others before committing them
- Never commit content with a known issue
- Easy to branch with automatic branch integration
- Specialize content for region, platform, and locale alike
- Full and automatic control over what assets go in a build
- Encrypt content at a feature level

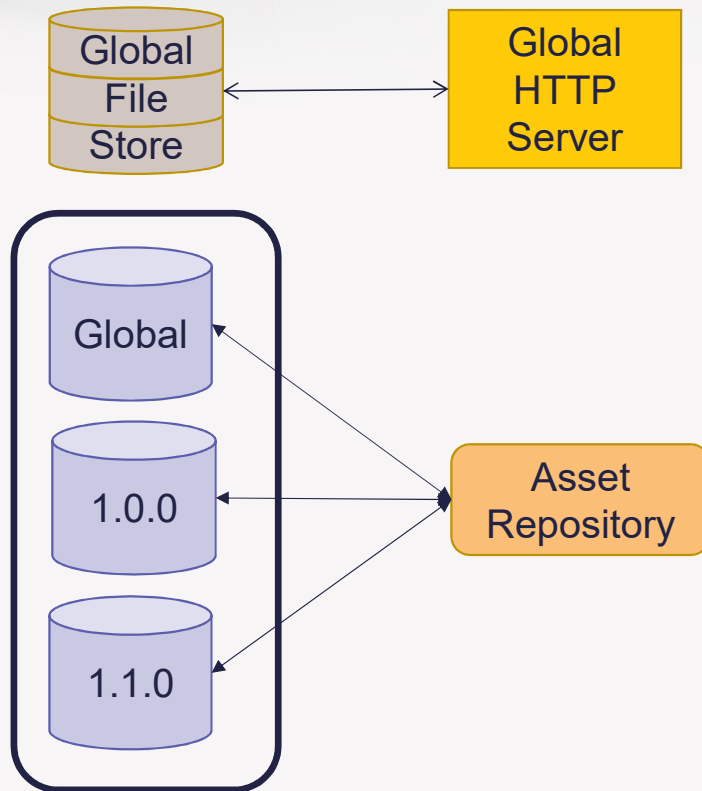
Standard Technologies

- SQL Databases
- HTTP Servers
- Scalable
- Easy to support technologies

Global Asset Services



Global Asset Services



Global Asset Schema

- ASSET_REPO_GLOBAL
 - Users identified by UserID
 - Workspaces identified by WorkspaceID
 - Branches identified by BranchID
 - Commits identified by CommitID
- ASSET_REPO_<BRANCHID>
 - Asset table indexed by GUID
 - Asset History table indexed by GUID and CommitID.

	 COLUMN_NAME	 DATA_TYPE
1	GUID	NUMBER (20, 0)
2	COMMIT_ID	NUMBER (20, 0)
3	FLAGS	NUMBER (20, 0)
4	PAYLOAD	BLOB

Anatomy of an Asset

Oracle database row

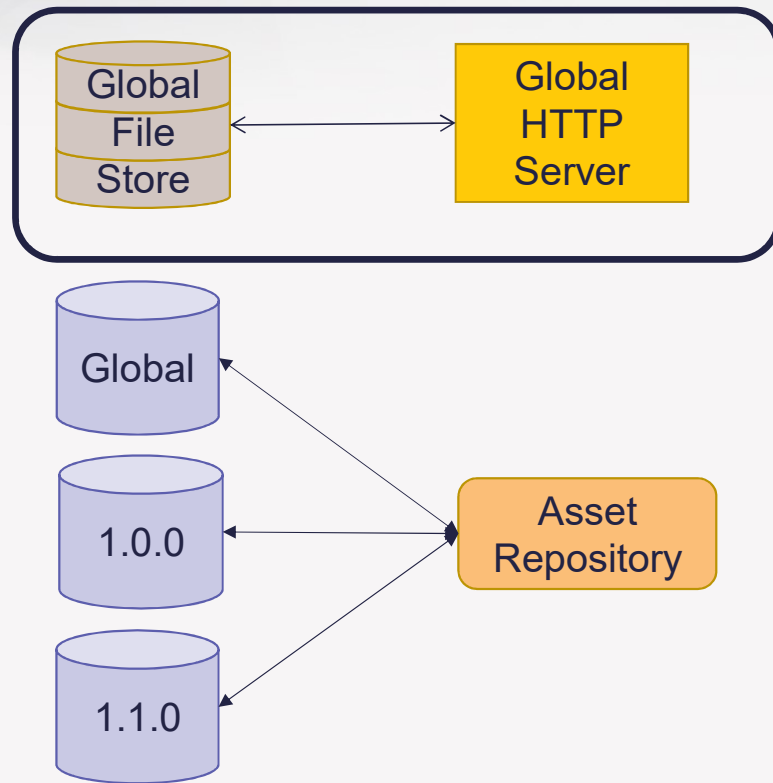
- GUID – Unique identifier within our project
- Commit ID
- Flags (Deleted/Archived/etc)
- Binary Payload (up to 64k compressed)
 - Name
 - Optional file backing SHA1
 - StructuredData

StructuredData

- Custom made binary tagged format.
- Auto generated classes using simple structure definition language.

```
1 class STUExample
2 {
3     s32 m_exampleValue;
4     ref<Model> m_displayModel;
5     ref<Animation> m_idleAnimation;
6
7     scope editor:
8         string m_comment;
9
10    defaults
11    {
12        m_exampleValue = -1;
13    }
14 };
15
16 asset Example
17 {
18     STUExample;
19
20     dependson Model;
21 };
```

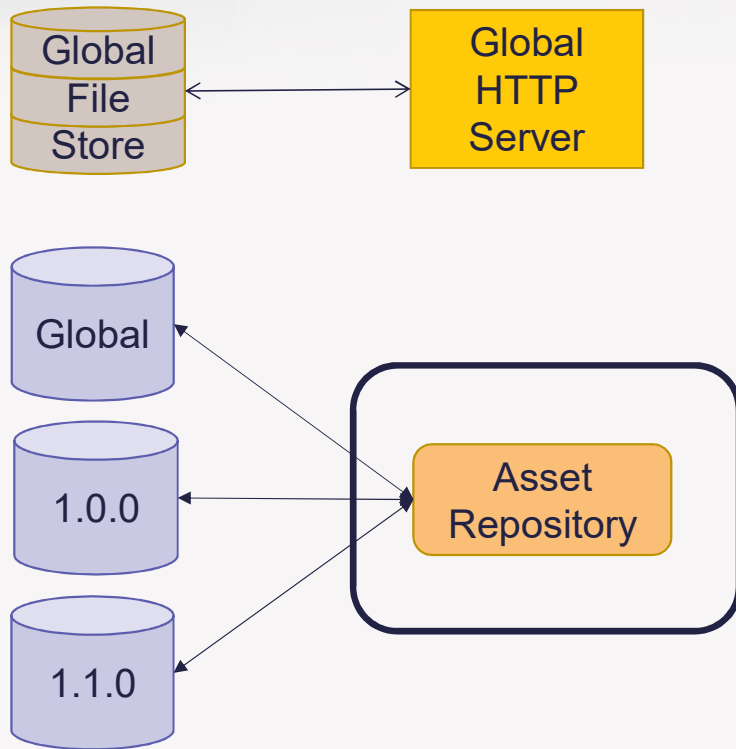
Global Asset Services



Asset File Store

- [https://<server>/<Sha1\[-4:2\]>/<Sha1\[-2:2\]>/<Sha1>](https://<server>/<Sha1[-4:2]>/<Sha1[-2:2]>/<Sha1>)
Content addressable!
- Bucketed by last two bytes of the file Sha1
- Using Sha1 for uniqueness content hash
- Currently have 1.7 TB in unique all time data
- 900 GB snapshot of assets @ head revision
- CentOS running Nginx with custom python validator
- 64 GB ram for OS file cache to fit our common work set
- Hardware specs can vary on team size and needs

Global Asset Services

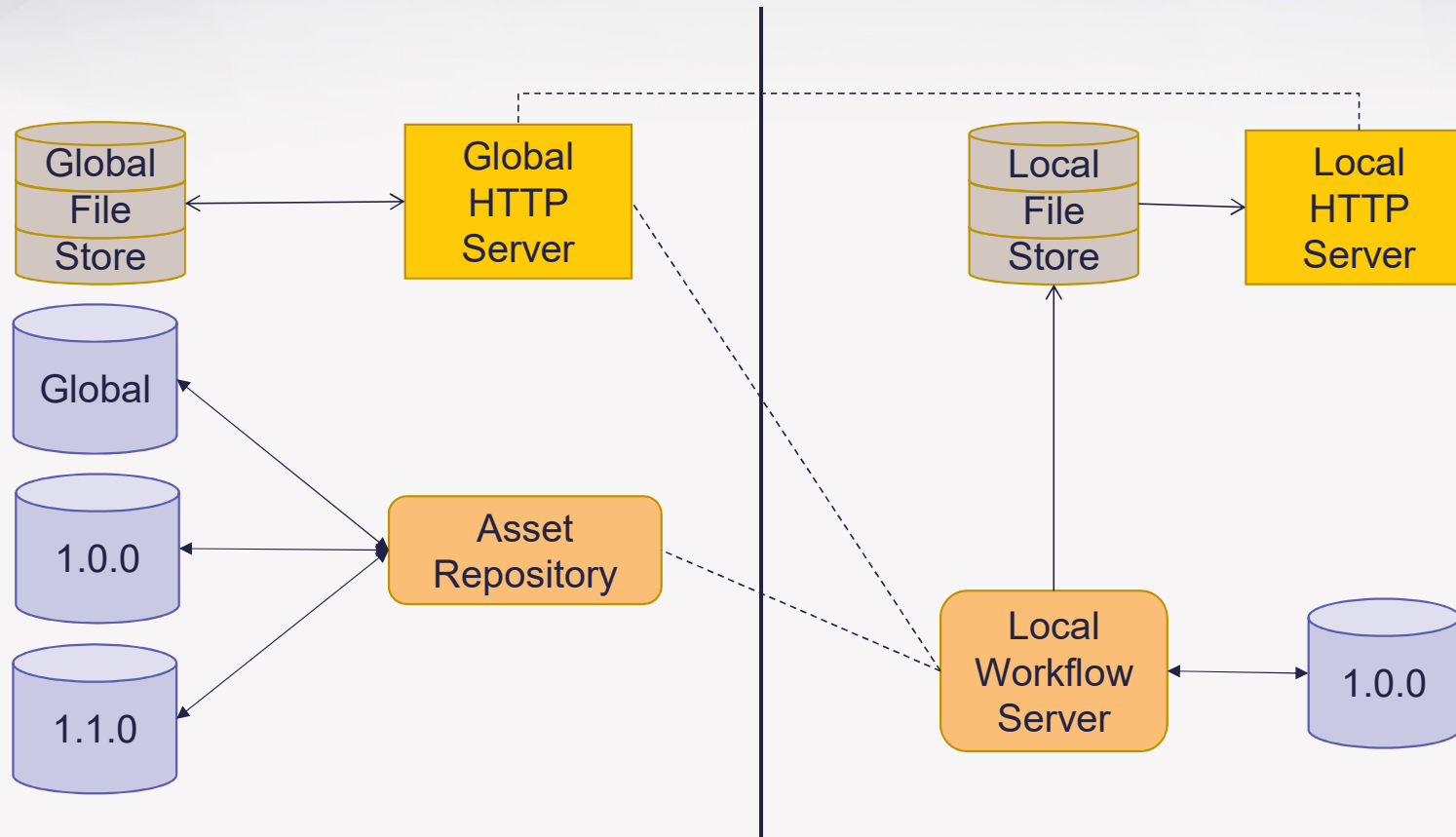


Asset Repository Server

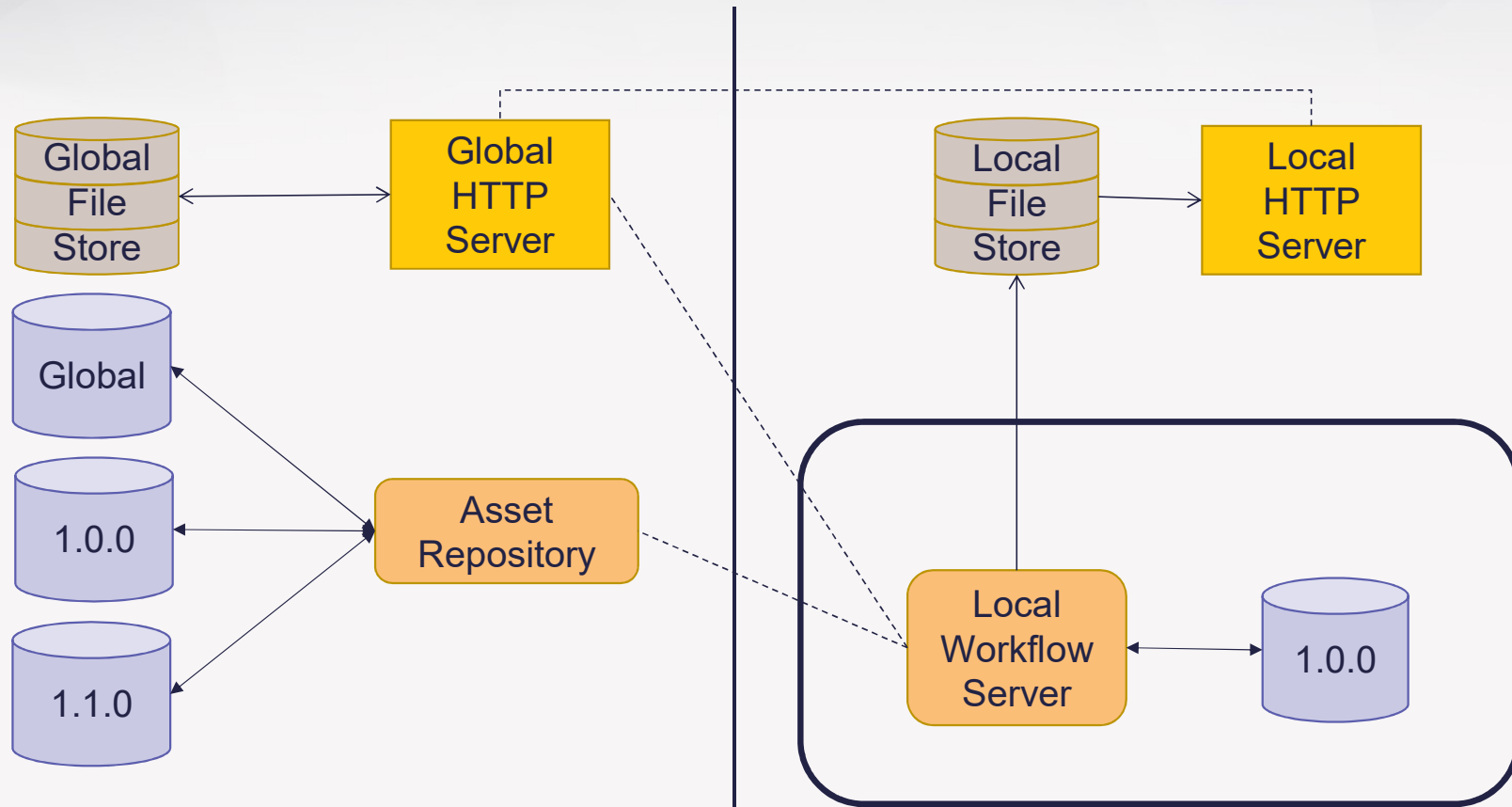
- Creates all base GUIDs
- Manages exclusive checkouts
- Handles commits
 - Integration to downstream branches
- History queries

Easy to take down, halts above operations but all local ops continue just fine...

Local Asset Services



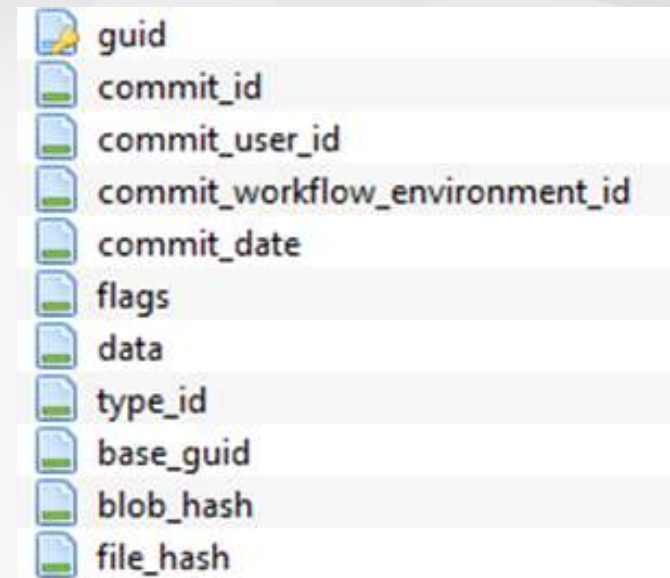
Local Asset Services



Local Asset Schema

The local asset schema stores a view of the asset repository. No history information and each branch is stored in its own database.

snapshot_assets	– Global view of assets at time of pull.
comitted_assets	– Assets committed after last pulling
local_assets	– Any assets you have checked out locally.
asset_names	– Separate table for quick lookups and queries.
local_operations	– Metadata on local changes.



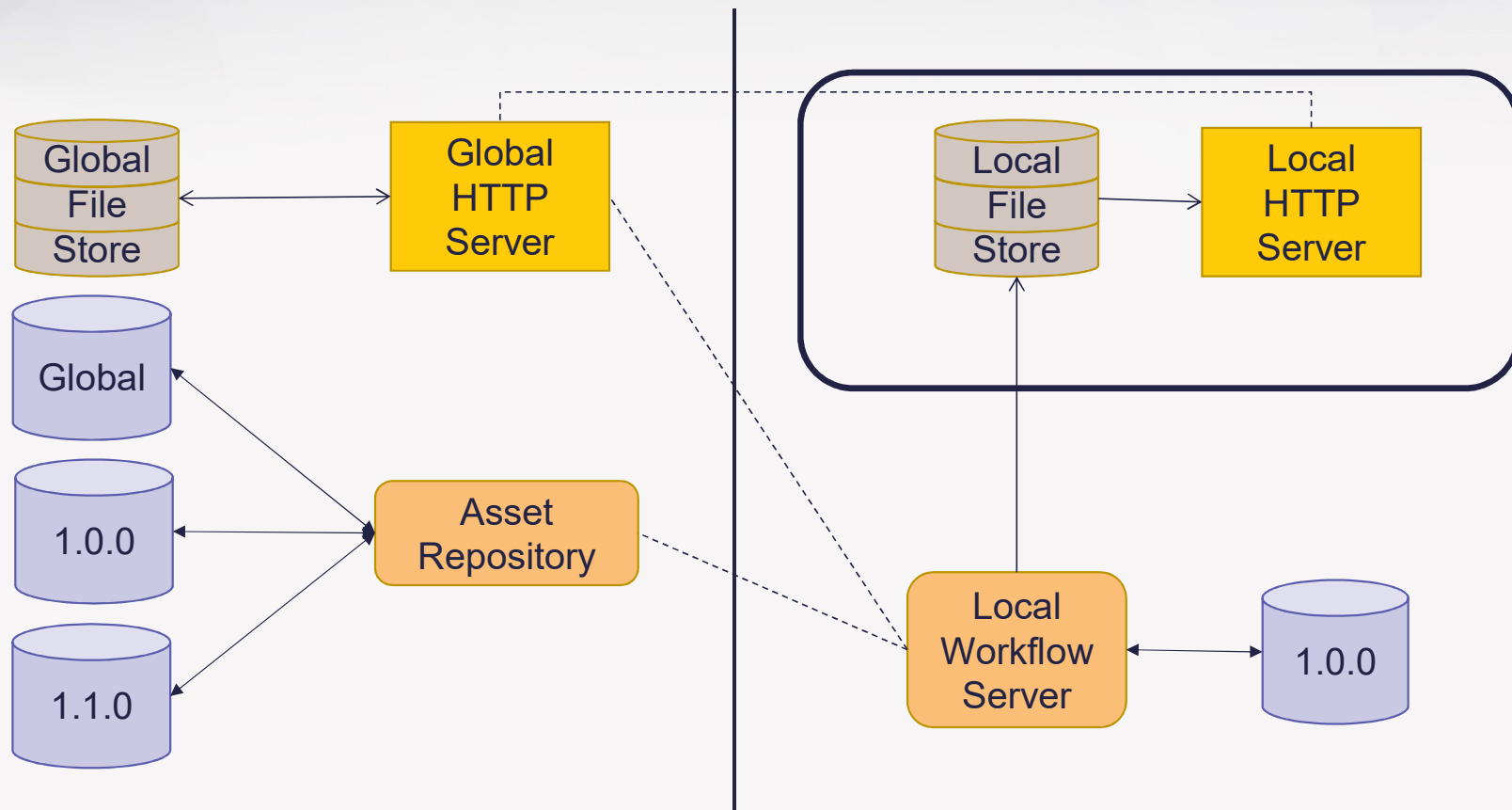
guid
commit_id
commit_user_id
commit_workflow_environment_id
commit_date
flags
data
type_id
base_guid
blob_hash
file_hash

Local Workflow Server

Server that manages our local asset schema. It is used by tools to fetch assets and store working copies. All asset queries by tools go through it.

- Abstracts local queries vs global queries
- Ensures proper transactions with Asset Repository Server and global asset file store
- Layers our three asset tables to work as one conceptual view of the assets
- Reference cache

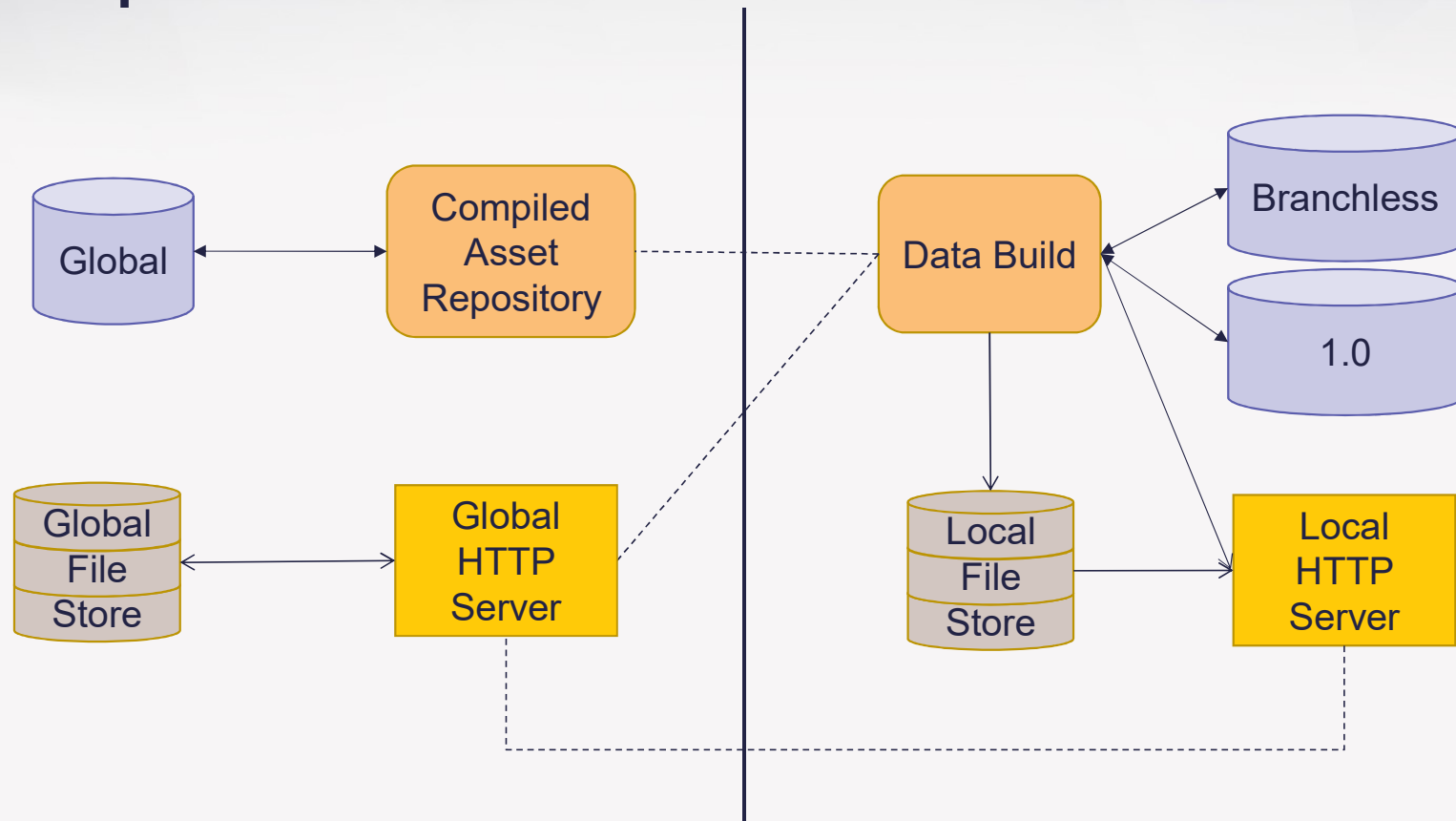
Local Asset Services

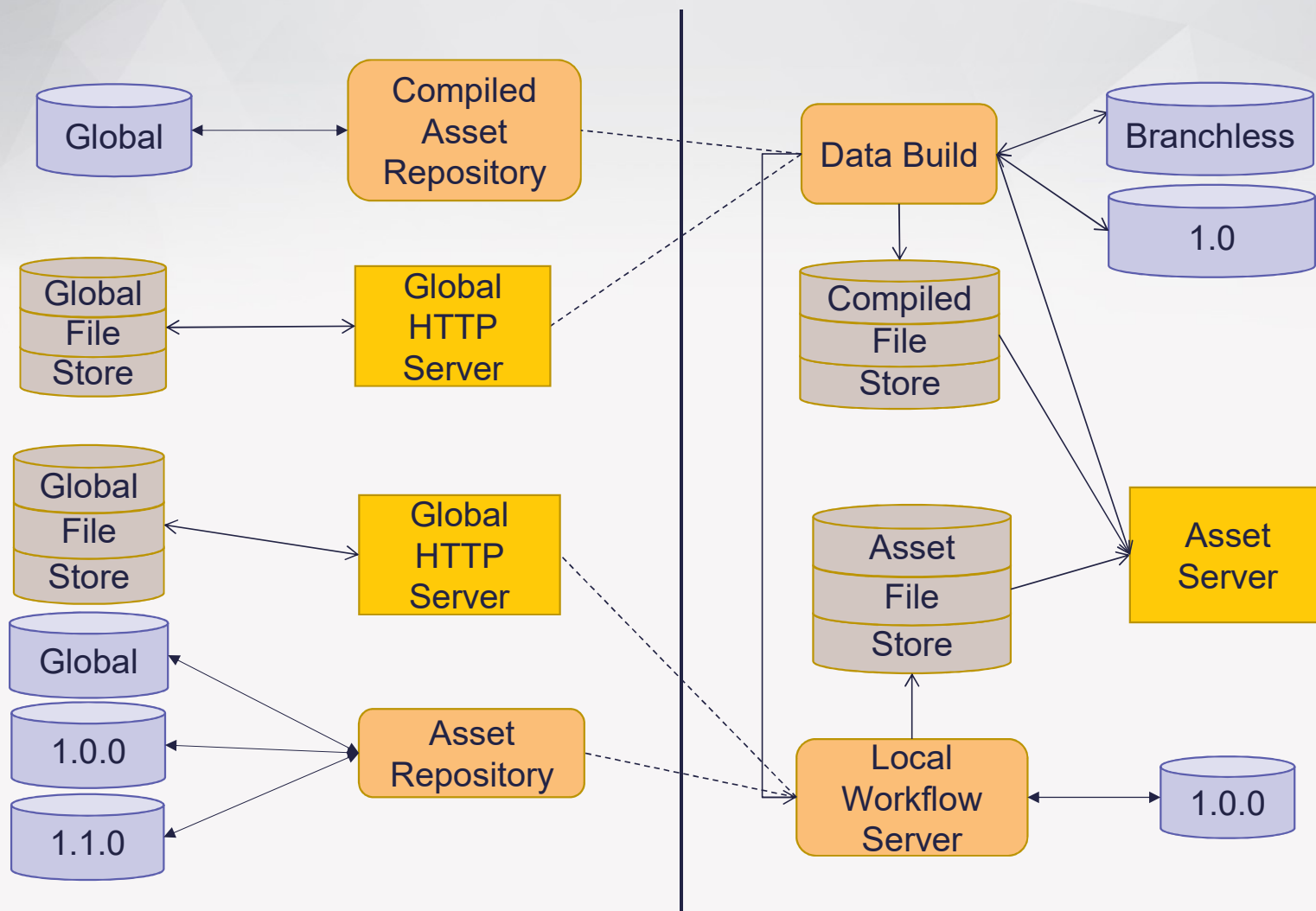


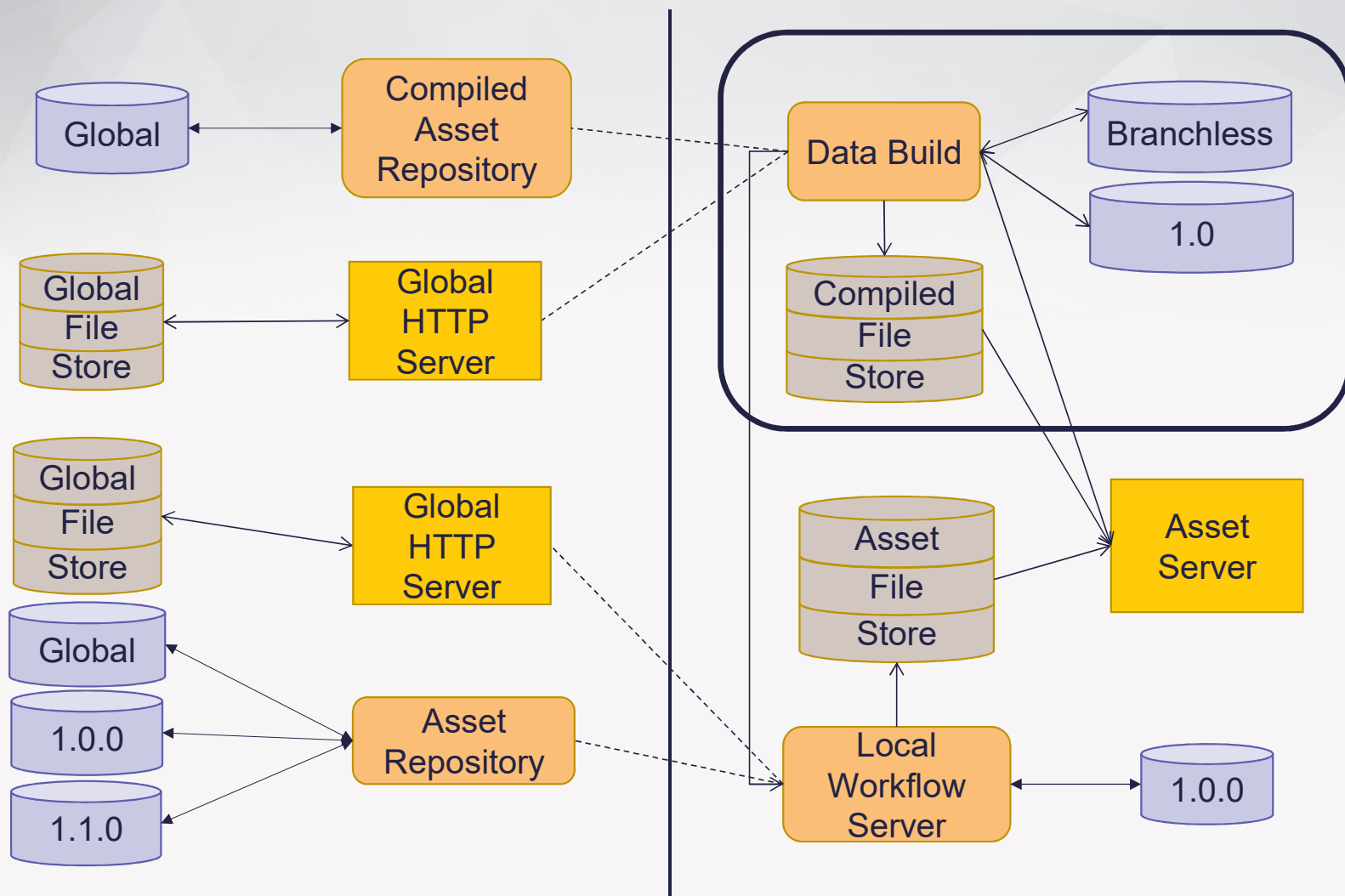
Local File Store and AssetServer

- Asset Local File Store is essentially identical to our global asset file store except it resides locally on your disk
- AssetServer is a locally running http server that can serve up files from the asset local file store. If files don't exist locally they will be redirected to the asset global file store. Only truly needs to serve up files unique to the local machine

Compiled Asset Services







DataBuild

DataBuild is our executable process that compiles all of our assets using a Builder specified per asset type. Going from type to type it builds all of our assets.

- Extracts references to other assets
- Compiles assets and outputs a database of the compiled assets and their references to other compiled assets

DataBuild Schemas

Branch schema is a snapshot of content for that branch.

Branchless is simply a mapping of inputs to outputs, it can contain multiple versions of the same asset.

- branch.asset_references
- branch.compiled_assets
- branch.compiled_asset_references
- branchless.compiled_assets
- branchless.compiled_references

And other supporting tables

DataBuild --sourcepull

“Sync all assets to head”

- Tells Local Workflow Server to pull up to date assets.
- Incremental in nature, pulls changes since last changelist pulled
- Fresh pull from scratch is currently 1.4 million records taking up to 10 minutes to process and write to disk even on an SSD
- 600mb resulting db file

DataBuild --targetplatform all --environment all

“Build those assets for all situations”

- Overwatch is on PC, XBox One, and Playstation 4. Some content is compiled differently on different platforms
- Can build data with “all” “future” “shipping” environment mindsets. Some builders remove anything not relevant. No Sombra in the build!
- Builders may return a different CompilerID per platform and environment combination they support
- Each type is built once per unique CompilerID
- If a type returns the same CompilerID for all combinations it will only be built once and those results will be used for all manifests

DataBuild --platform all --region all --locale all

“Build all variations of those assets”

- We actively build data for 3 platforms, 2 regions, and 13 locales. Each combination results in a manifest unique to that combination
- Asset types can opt in to specialization for each of these. Compiled results are saved namespaced to that specialization using `localization_id` field in the schema
- 6 of our types support region specialization
- 13 of our types support platform specialization
- 16 of our types support locale localization

DataBuild --compile all

“Time to get to work”

- StructuredDataBuilder handles 80% of our types
- Derived StructuredDataBuilders handle another 15%
- Engine team custom writes last 5%
- 203 different types of which we build 170 and process 180
- Compiled asset results are stored in the Compiled File Store
- Results stored in both of DataBuild’s databases

Outputs

Compiler Hash:

- asset_type
- compiler_id
- localization_id
- databuild_version

compiled_assets	
	compiler_hash
	build_type
	source_guid
	source_hash
	compiled_guid
	compiled_type
	compiled_md5
	compiled_size
	compiled_flags

Outputs

Source Hash:

- Asset Hash
 - Structured Data Payload MD5
 - Additional File Sha1
 - Deleted/Archived Flags
- Dependency Asset Hashes
- Uniquely identifies all content inputs to this asset build.

compiled_assets	
	compiler_hash
	build_type
	source_guid
	source_hash
	compiled_guid
	compiled_type
	compiled_md5
	compiled_size
	compiled_flags

DataBuild Manifest

- The end result of all of our building
 - Essentially just a list of GUID, size, and MD5 for every compiled asset
- One manifest per configuration
- Built by simply selecting all sqlite results for each type filtered by the compiler_id relevant for that specific configuration
- Also builds Packages and Bundles

DataBuild Times

- Dev build
 - 350 thousand assets
 - 2 configurations
 - 12 hour build on 8 core machine
- Full retail build
 - 3 million assets across all specialized platforms/regions/locales
 - 65 configurations each with their own manifest
 - No idea how long it takes on an 8 core machine... why would I ever do that?

DataBuild --distribute

“Don’t do work others can do”

- Already make heavy use of multi-threading, still takes us 12 hours
- Over 10 thousand materials generate over 22 thousand unique shaders up to 5 seconds each. Need more CPU-cores
- Using a distributed build system we distribute work to other machines
- Only useful for CPU bound types
- Using 30 workers gets us from 12 hours to 4 hours

What do we have now?

10 minute source asset pull

900gb of asset file data

1gb of executables/dlls as part of a build

8gb of compiled asset data

4 hours of data building time

Turns out this isn't doable in 5 minutes. So at this point we just stopped doing things... it turns out it is much faster not to do things.

Pulling a build

“Don't actually pull the build.”

- Well at least not all of it.
- Clients, servers, tools, can all fetch their own data no need to download compiled assets
- DataBuild can stream data on demand, no need to download source assets
- Don't need every feature, split builds up into folders for each feature download it and go. On average download 150mb of 1gb build executables/libraries
- In order to run the build with local data you must activate the build with DataBuild

Seed Databases

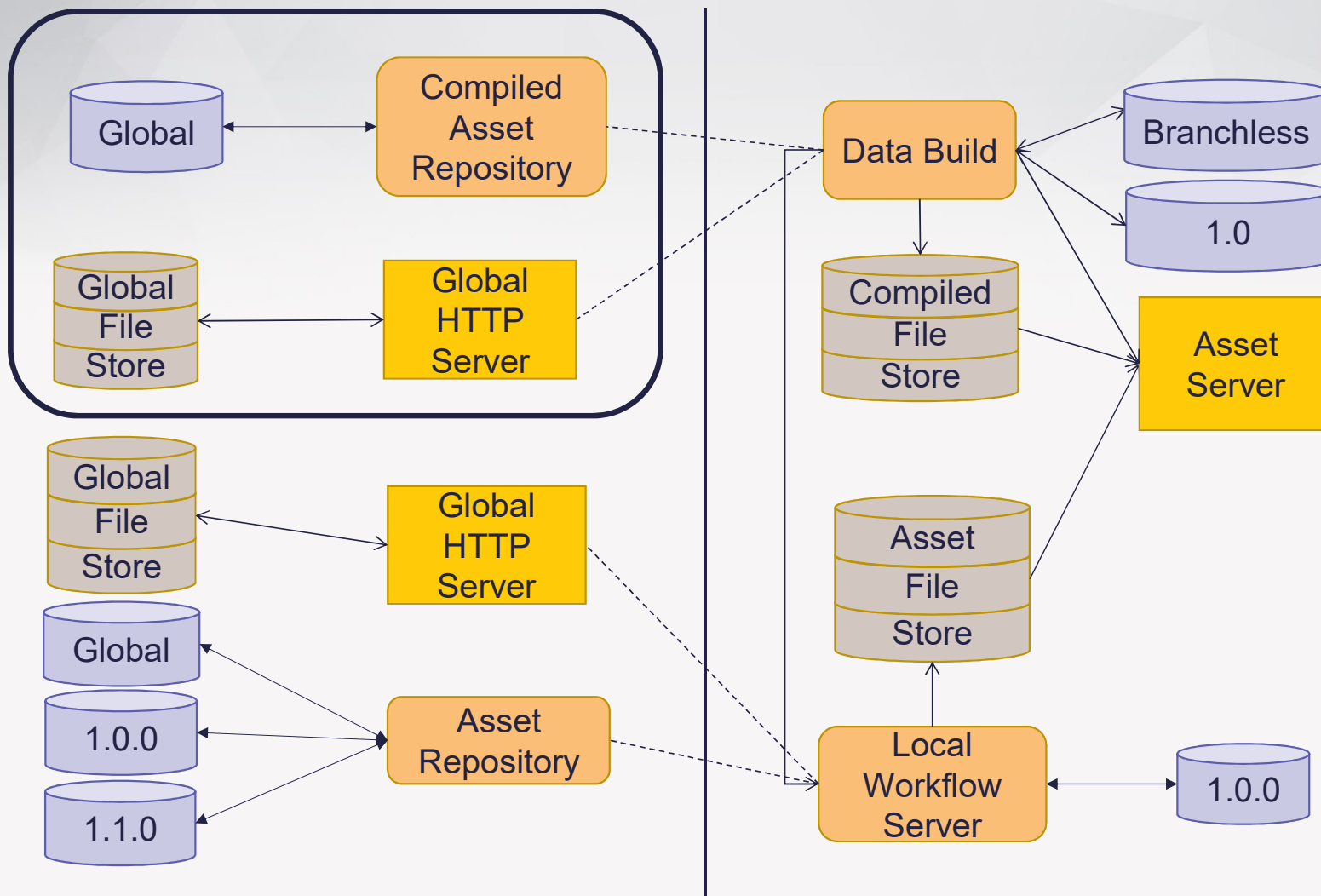
"Don't pull from scratch"

- This applies to our “sourcepull.” If you have never pulled before on that machine for that specific branch we download seed databases that we make every morning.
- ~600 mb download, single sqlite file
- Your pulls are now incremental even the first time you have ever pulled on that machine. This saves us approximately 10 minutes if you have never pulled a branch before

DataBuild --incremental

"Don't build what you have already built"

- Needs to be 100% reliable
- Through knowledge of our dependencies. For each asset fetch its full tree of dependencies. Sort and hash.
- Multi-tiered
 - Check happens at type level first
 - Per asset check if we already have a result in the branch db.
 - See if our "branchless" db knows about it. This copy is independent of branch or configuration. Simply maps input hash to output records.



DataBuild --compiledpull

"Don't build what others have built"

- "Compiled Asset Repository" is essentially a global version of our DataBuild "branchless" database
- SQL view of Compiled Asset Store
 - 25 million files across 1.6 TB for last 60 days + reserved builds
- Only difference is that results here include `localization_id` in the index
- At start of build we query all new records and store them in our branchless db
- Difficult to get perfect. Deliver incremental relevant data, don't waste time for user downloading results

Beneficial Side Effects

"Don't download any asset files"

- By not building the assets we never pulled down the files needed to build them. All file fetching is done for a builder but only if invoked
- Because we have database records we know about every compiled asset already without actually syncing them either
- Complete manifest of every asset in the game without any of the work

Where are we at now?

“5 minute build pull!”

- 30 second build download
- 2-30 second asset pull
- 30-60 second compiled asset pull (async though with following)
- 2-3 minute hashing of content, and building local asset changes
 - Building local changes is distributed for non trivial types
- 1 minute manifest and package build

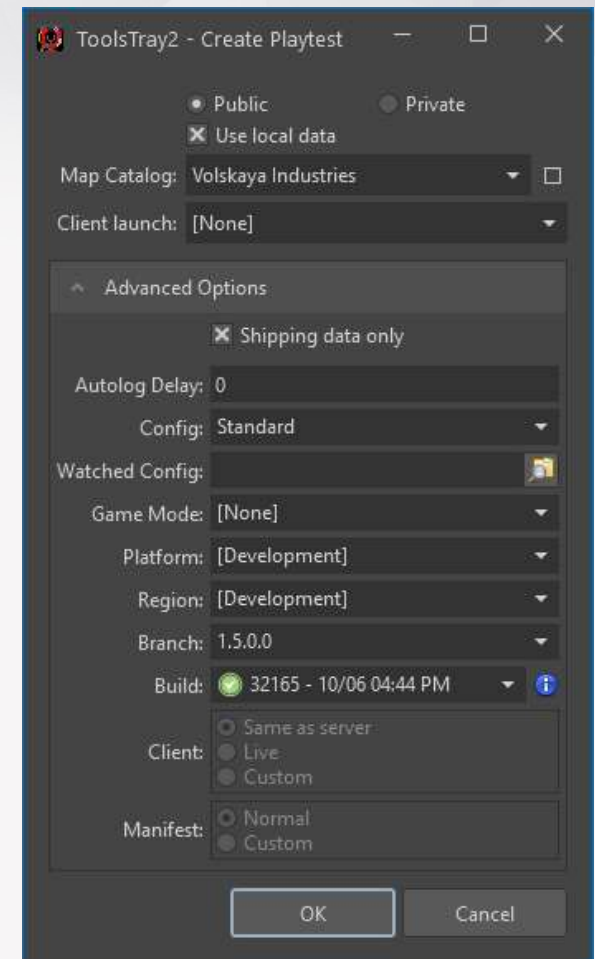
Client Loading

"Don't download compiled assets you don't need"

- Core resource system is async, no blocking load in retail
- Always loads from an archive container, if not present we download it and write it into the archive container
 - Blizzard uses CASC for archive containers now
 - TACT logical layer in Retail builds
 - Development builds simply use HTTP to download the file from an AssetServer
- All on demand, only fetched if you load it. If you only work in one map you only download one map
- Works exactly the same way on consoles! HTTP is standard across many platforms

Playtests with local data

- Only three things needed to run.
Executable, Manifest, HTTP Server
- Tools Tray uploads your manifest and spins up servers who broadcast their existence
- Other users use ToolsTray to join a playtest, downloads the same client and manifest and points at host's AssetServer



Future Improvements!

"Lets see how many of these were done after I wrote this"

- Lazy Build – When an asset changes don't immediately build all dependencies. Let engine request them
- Job/dependency job system – Eliminate fork and join, joins are wasteful if you have more work to do
- Eliminate single points of failure – Run server clusters

Other mentions

- Jesse Blomberg – LocalWorkflowServer
- Philip Orwig – Asset Repository Server
- Rowan Hamilton – Obligatory Lead callout
- Evan Calder – Tool Integration/LWS work
- Luke Mordarski – Distributed Building/Package Manager
- Even Braudaway – Reports/Compiled Asset Repository

Questions

- I don't know what goes here, this part is up to you.

dclyde@blizzard.com

@DaytimeCoder

Bonus Slides

- The opposite of not doing something
- Doing more things

Anatomy of a GUID

64 Bit identifier

- 0xF000000000000000 – 4 bits reserved for engine usage
- 0x0FFF000000000000 – 12 bits used as a Type mask.
- 0x0000F00000000000 – 4 bits to indicate the platform the asset was authored for.
- 0x00000F8000000000 – 5 bits to indicate the region the asset was authored for.
- 0x0000001F00000000 – 5 bits to indicate the locale the asset was authored for.
- 0x00000000FFFFFFFF – 32 bits for a unique identifier

Default platform, region, and locale are "Development" with a value of 0.

Ex:

- 0x0DE0000000000001 – "Welcome to Overwatch"
- 0x0DE0100000000001 – "Welcome to Overwatch on PS4"
- 0x0DE0100100000001 – "Welcome to Overwatch on PS4" <In German>

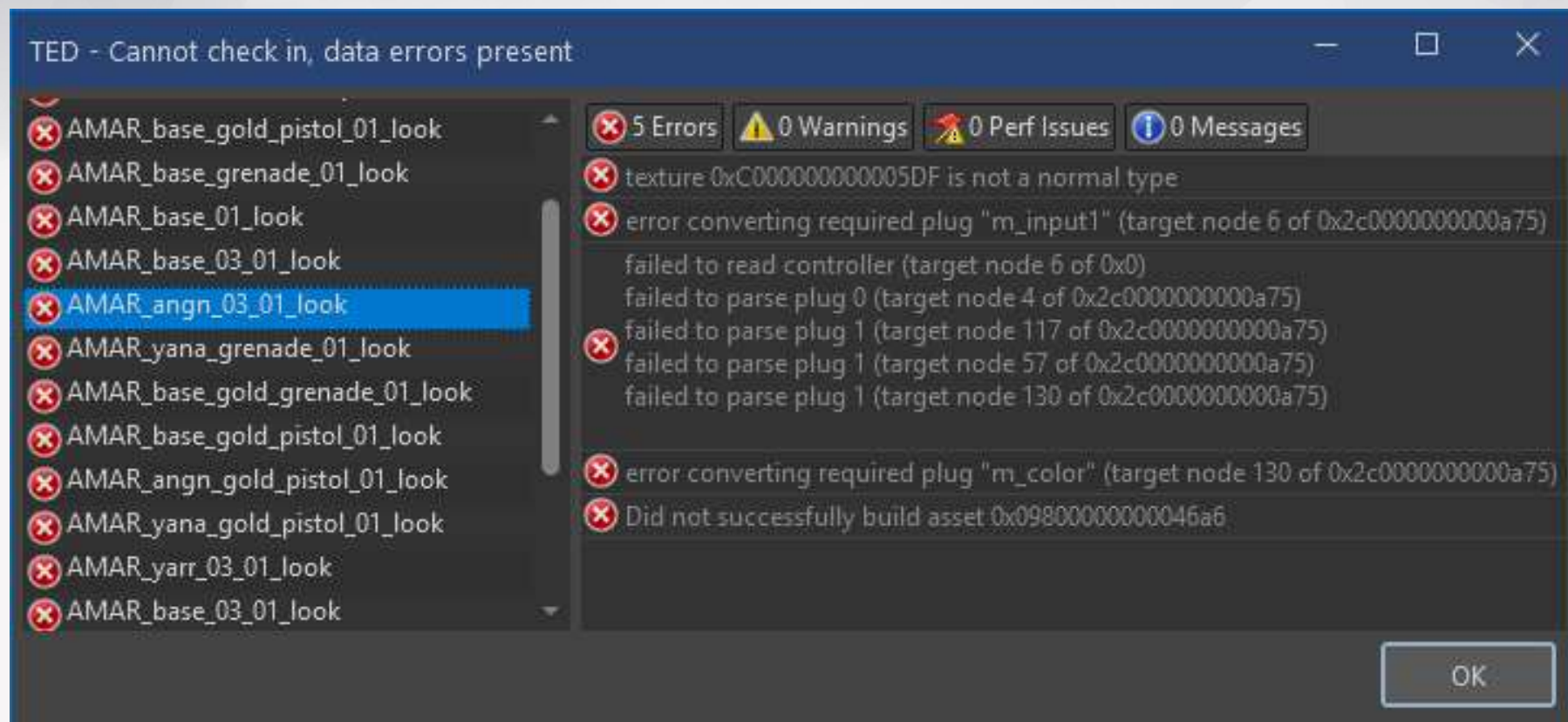
Packages

“Don’t ship our secrets”

- Collection of compiled assets necessary for a feature
- Largely auto generated by builders
- Can specify when they ship and with what encryption
- Built from DataBuild’s compiled asset reference tree
- Strict control over what assets ship as settings propagate to all references
- Runtime presence and status queries
 - If not all heroes are downloaded only open tutorial
 - If you cannot decrypt seasonal content yet don’t show any of it.

Content Validation

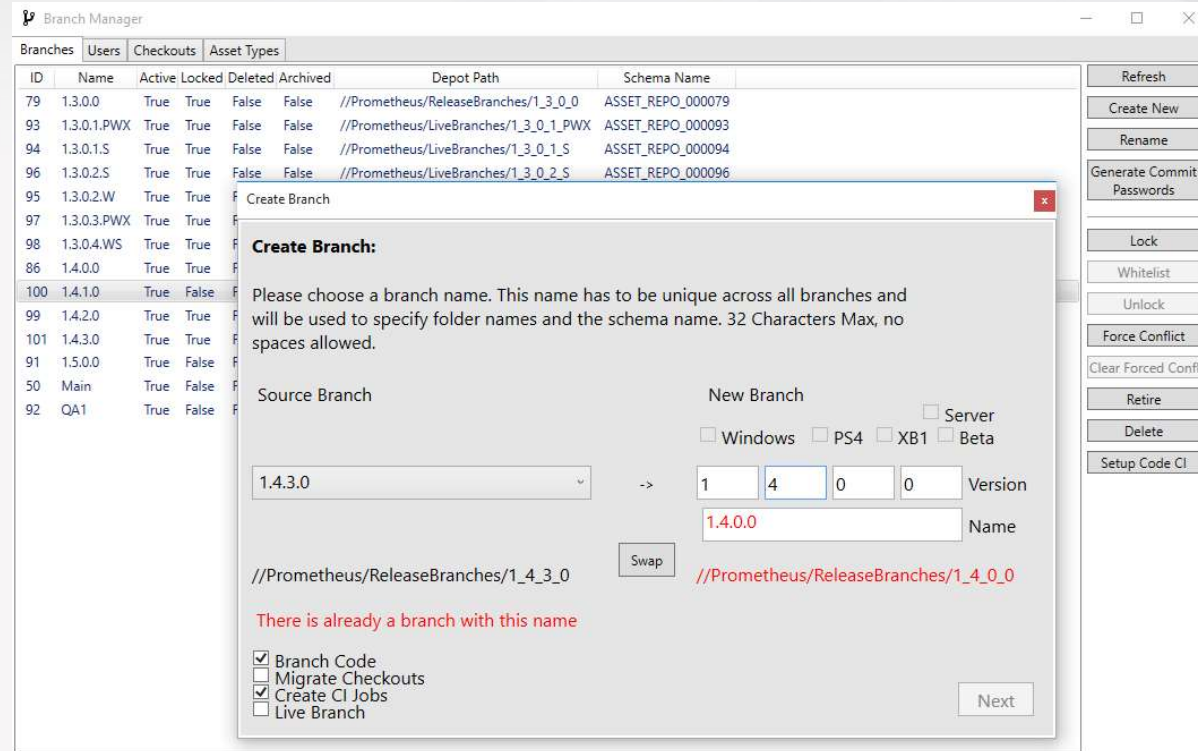
- Builders are encouraged to validate data. Even if a transform isn't necessary maybe check that the texture in the normal map slot isn't a 3D texture, or make sure our icons aren't 4k.
- Warn about any failed validation
- Error on any failures to build
- Prevent users from committing with warnings or errors
- Run Data Continuous Integration building content in all configurations and look for errors and warnings.



Branch Manager

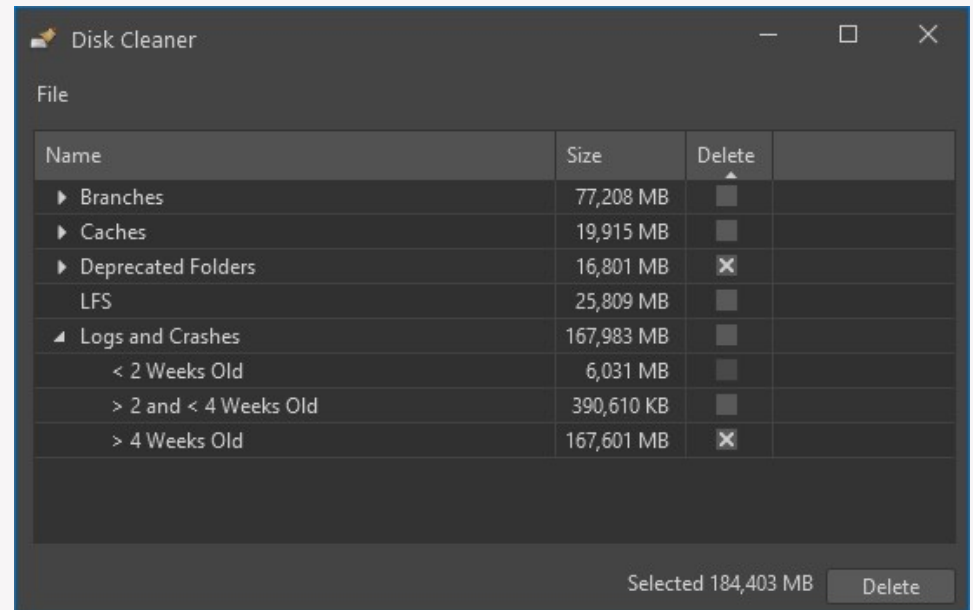
Tool used to manage and create branches, which has been extended to open DB operations to producers. Including:

- Branch Creation
 - CI Jobs
 - Validation of settings
 - Integration setup
- User Creation
- Lock Branches



Disk Cleaner

- Tool helps users cleanup space.
- Combats bloat
- Cleans up old branches
- Cleans up log folders

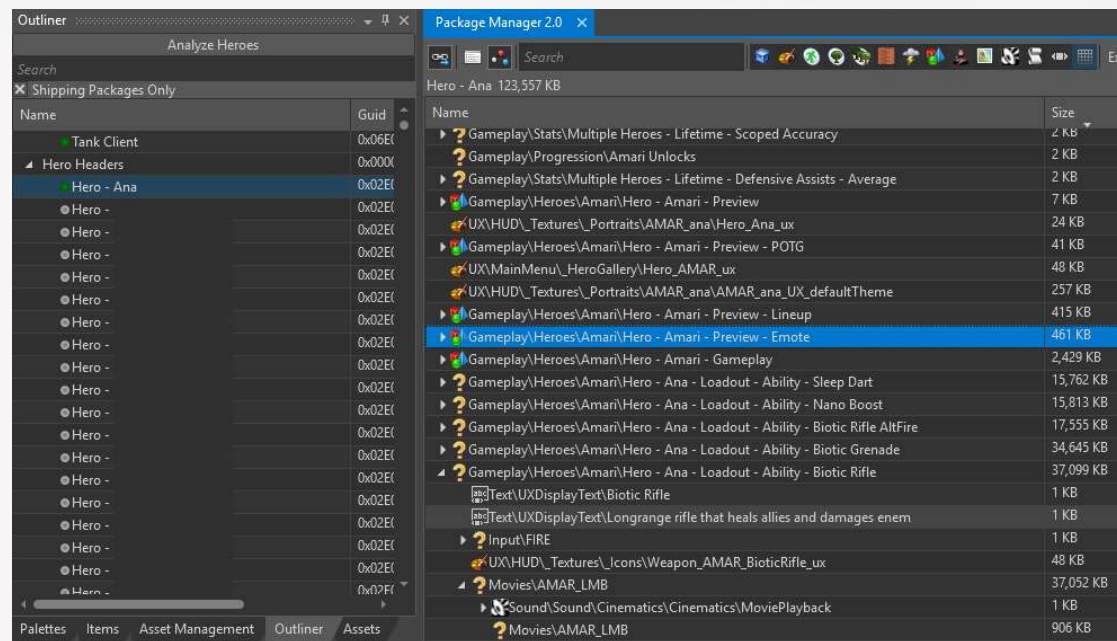


Package Manager

Simply put Package Manager is a visual tool to view what is in a package.

View:

- Assets by size
- Assets by type
- Assets by reference



Maintenance

- Backups
- Seed Database creation
- Compiled Asset Repository Cleanup
- Local database pruning

Data CI

- Builds all data continuously as it is committed.
 - Validates users didn't commit any errors.
 - Provides results to Compiled Asset Repository for others to pull down including builds and devs alike.
- Will email you if you broke something.

DataBuild CI - 1_0_4_0 - Windows;PS4;XB1: Failed	
Job Duration	1 hours 7 minutes
Computer	
Changelist	170028
Return Code	1321
Jenkins Job	
DataBuild Errors	
Too many errors, results truncated	
[INFO] Starting build on 0x025000000000090b Gameplay\Progression\Sprays\Spray - [WARN] spray small thumbnail 0xC00000000007B54 is larger than 128x128 [INFO] using an outro state with a length of 0 seconds, this is a performance penalty [INFO] using an outro state with a length of 0 seconds, this is a performance penalty [WARN] spray small thumbnail 0xC00000000007B54 is larger than 128x128 [INFO] using an outro state with a length of 0 seconds, this is a performance penalty [INFO] using an outro state with a length of 0 seconds, this is a performance penalty [INFO] Successfully built 0x025000000000090b [INFO] Last committed by: rehamilton	

Data Compare

- Optimized Release executable
- 3 different memory initializations
- Doesn't use ResourceCAS
- Incremental/Distributed
- Shuffles build order
- Diffs output manifests
- Catches:
 - Uninitialized memory in the final compiled assets
 - Uninitialized memory in builder
 - Static values injecting stateful values into compiled assets

Package Reports – Load Times

Load Times – Tracks loading trends across multiple builds.

- Loads all of our maps several times.
- Loads all of our various hero/skin/weapon combinations.

	Build	Load Time (s)	Misc Calls	Misc Call Time (s)	Total Reads	Total Read Time (s)
	09/07/16 - 31552	6.018176	16721	0.768578	4128	11.198144
	09/07/16 - 31551	7.394101	16788	0.147569	4197	8.045791
	09/07/16 - 31532	4.975178	17021	0.697969	4206	8.696474

Package Reports - Sizes

Evaluates built packages for each hero/skin/weapon combination and each map. Breaks down compiled assets by type to identify texture, voice, sound, etc budgets per hero, map, whatever. In addition to the trend it includes what files were new that build to account for increases in size.



Build	TotalLoadedSize ↕	Texture ↕		Model ↕		Animation ↕	
10/11/16 - 32270	61,466	24,608	23%	8,087	8%	5,156	5%
10/10/16 - 32232	61,462	24,608	23%	8,087	8%	5,156	5%
10/07/16 - 32189	61,460	24,608	23%	8,087	8%	5,156	5%
10/06/16 - 32131	61,520	24,608	23%	8,087	8%	5,156	5%
10/04/16 - 32064	65,918	26,342	24%	8,217	8%	6,224	6%

Blacklist

- Generates a list of strings from our heroes internal/external names. (Also maps, abilities, skins, etc)
- Searches our asset manifest, executable, and asset data itself looking for said strings.
- Compares what encryption they should have vs what encryption they do have.
- Generates a report of anything in "too early" of an encryption.
- LOTS of false positives, devs like to scare me a lot.

Win_SPWin_RCN_LenUS_EExt.psv (link)				
Blacklist	Line#	Guid	EncKey	Match
	26489	0x0D0000000000EDF	0.5.1 DJ And Roadhog	
	39931	0x0D0000000001772	0.5.1 DJ And Roadhog	
	40044	0x0D0000000001E2C	0.5.1 DJ And Roadhog	
	46059	0x07C000000005F325	1.0.2	
	34875	0x07C000000005F345	0.5.1 DJ And Roadhog	
	45932	0x0D00000000003218	1.0.2	
	2435	0x0300000000001A48	ClientBaseKey	
	2525	0x0C00000000007803	ClientBaseKey	
	4038	0x0A5000000000179C	1.0.3	
	5027	0x0C00000000007B5E	1.0 Beta	
	7678	0x0C00000000005B3F	1.0 Beta	
	36257	0x0C00000000004390	1.3 Gamescom Content	
	36257	0x0C00000000004390	1.3 Gamescom Content	
	36257	0x0C00000000004390	1.3 Gamescom Content	
	2515	0x0C000000000077F9	ClientBaseKey	
	2515	0x0C000000000077F9	ClientBaseKey	
	49580	0x07C0000000040962	1.0.3	
	29	0x0C000000000001ED	ClientBaseKey	
	30	0x0C000000000001EE	ClientBaseKey	
	31	0x0C0000000000021E	ClientBaseKey	
	32	0x0C00000000000EE9	ClientBaseKey	
	2330	0x0580000000000011	ClientBaseKey	
Witch	24816	0x07C0000000043126	0.3.2	Sound\SoundWEMF
Witch	24817	0x07C0000000043127	0.3.2	Sound\SoundWEMF
Witch	24819	0x07C0000000043129	0.3.2	Sound\SoundWEMF
Witch	24820	0x07C000000004312A	0.3.2	Sound\SoundWEMF