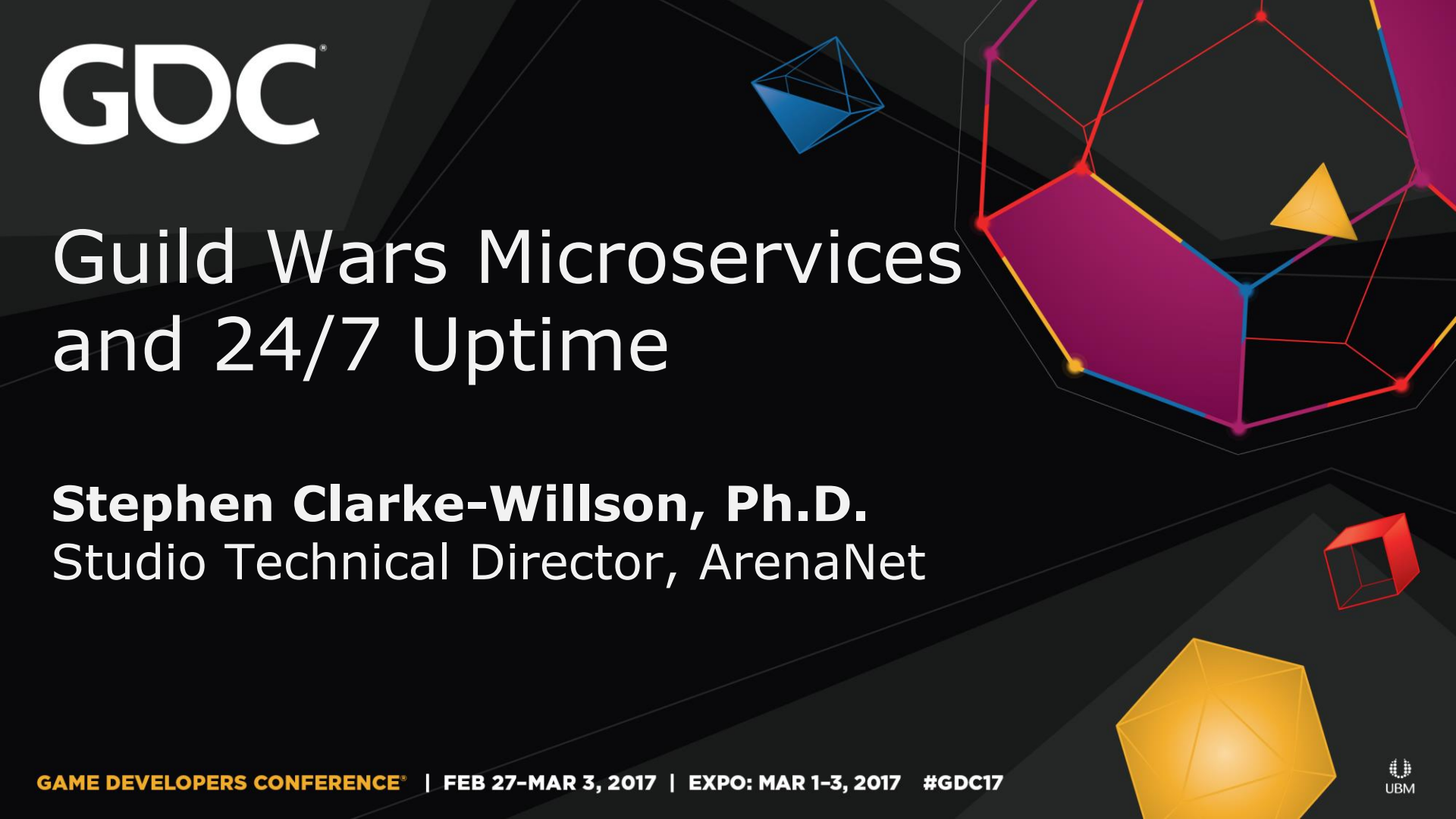

GDC
Guild Wars Microservices and 24/7 Uptime
Stephen Clarke-Willson, Ph.D.
Studio Technical Director, ArenaNet
GAME DEVELOPERS CONFERENCE® | FEB 27–MAR 3, 2017 | EXPO: MAR 1–3, 2017 #GDC17
UBM

Outline

This talk is in four main parts:

1. Meta
2. Overview of microservices in Guild Wars
3. Orchestration
4. Restartability: Problems and solutions

Part One: The Meta

Introduction (me)

- Game development 25 years
- Mixed roles over time of producer, programmer, executive
- Big list of games at Mobygames (mostly accurate)
- Eleven years at ArenaNet

Guild Wars and Guild Wars 2

- Guild Wars introduced MMO-style game with no monthly fee; launched 2005
- Guild Wars 2 is a larger, open-world game, with multiple game modes, and still no monthly fee; launched 2012

Talk meta

- Originally submitted in 2014 for GDC 2015
- Advisory board feedback: that's cool, but you don't explain how to restart servers
- Me: I can't even explain that to my staff
- Pulled talk to mull that over

Microservices became a thing

- Kubernetes – 2014
- Docker 1.0 – 2014
- DockerCon – 2016 – gave me most of my terminology
- Suddenly I had a path to explaining how we build things

Credit where credit is due

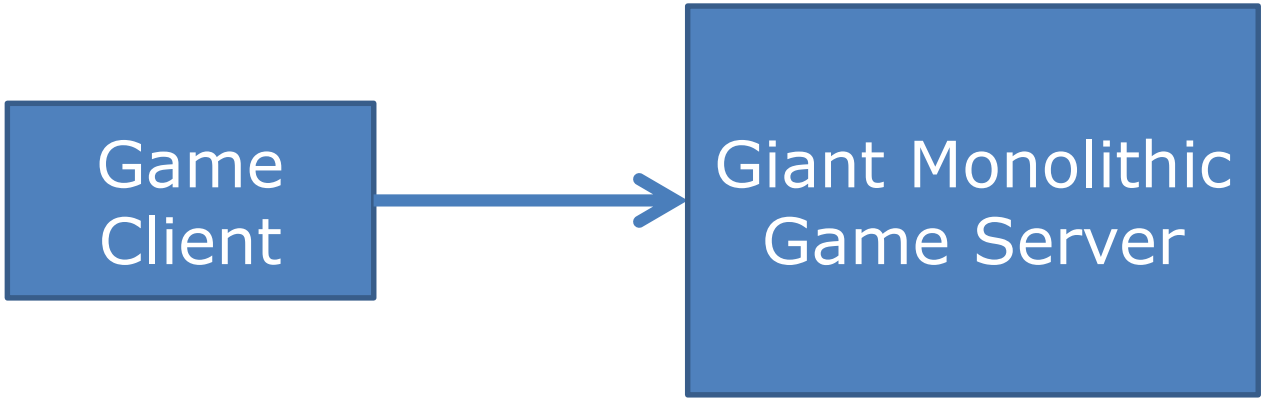
- Mike O'Brien designed and wrote original version of Battle.net at "another company"
- Pat Wyatt wrote most Guild Wars 1 back-end servers as restartable services
- Glen Knowles wrote the Guild Wars 2 microservice infrastructure and many of the services
- Halldor Fannar gave lots of constructive feedback on this talk

Part Two: Overview of Guild Wars Services

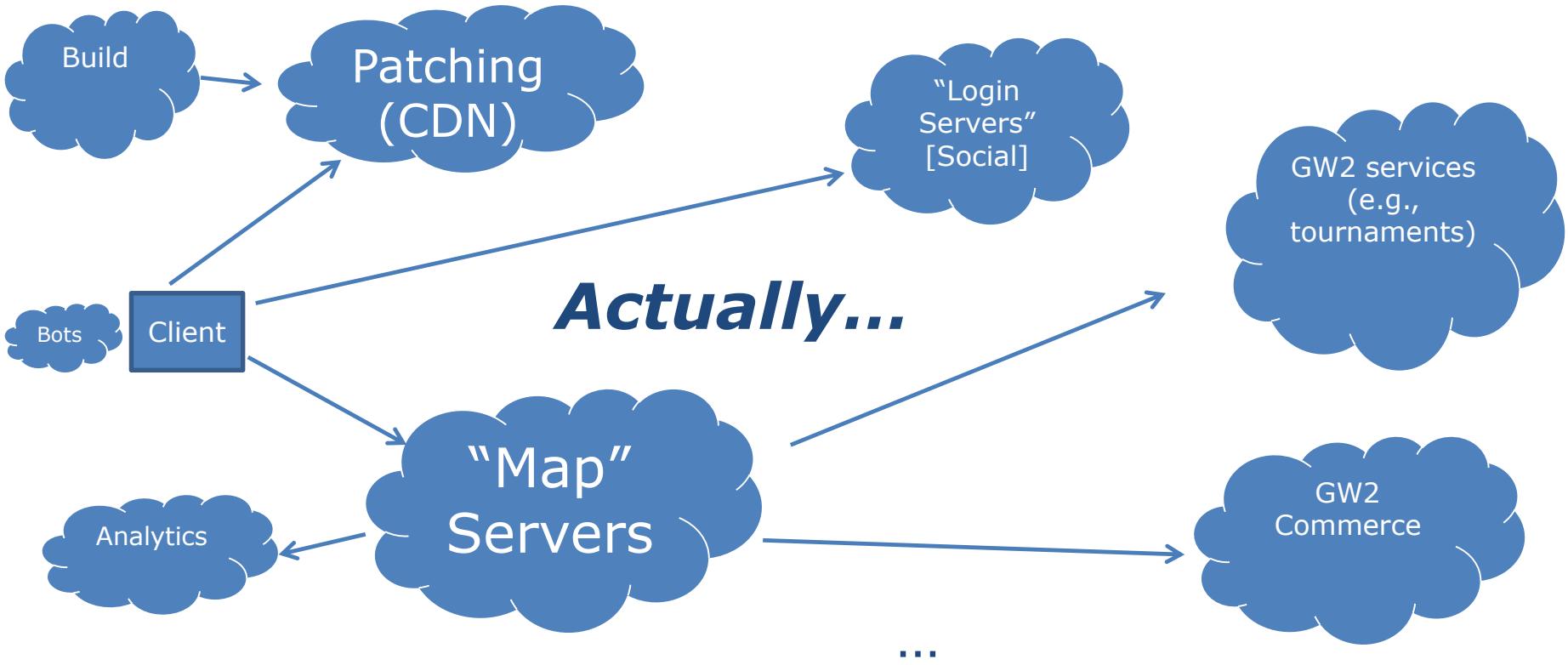
What is a microservice?

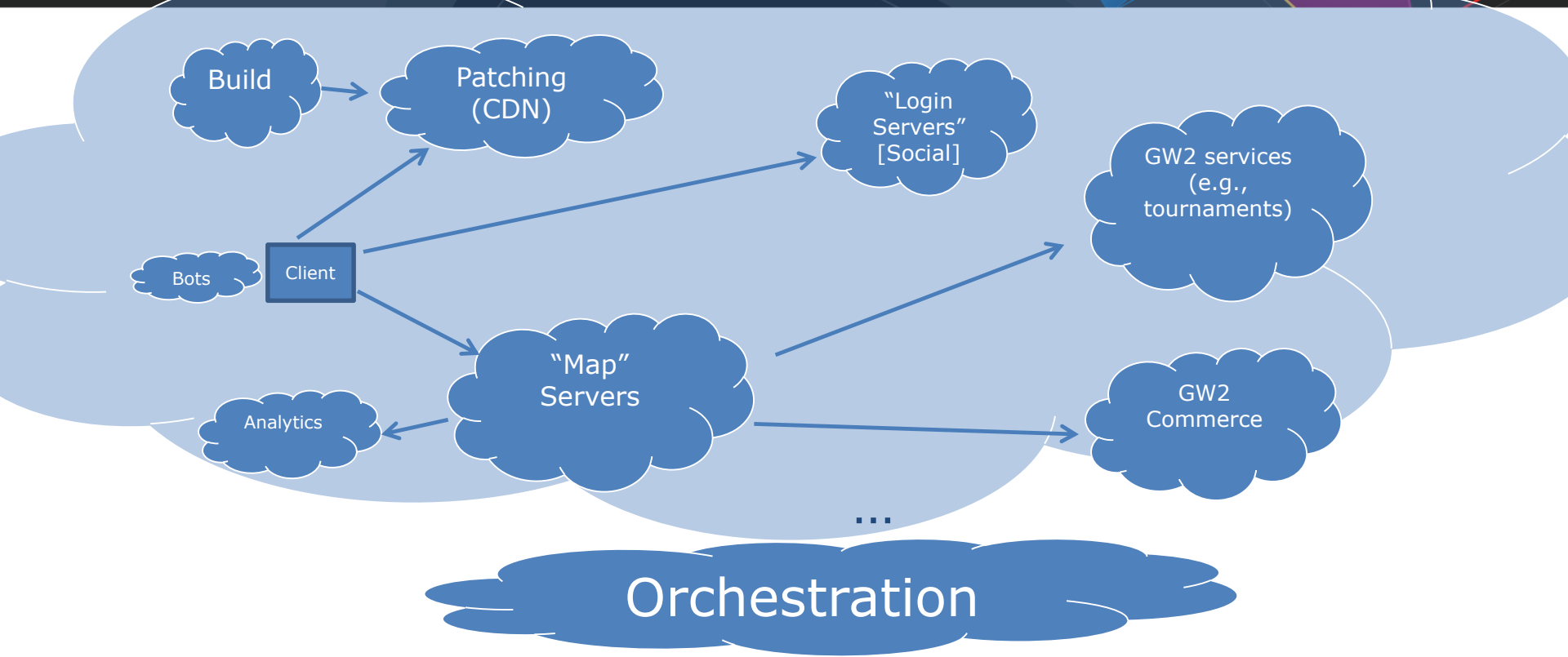
- Microservices are the evolution of the original Unix utility concept of “doing one thing well”.
- Microservices are like well factored subroutines … except asynchronous and separately restartable.
- Originally enabled by “Chroot” file system isolation in Unix; an alternative to virtualization.
- *Most* of our microservices have process-level isolation; restartability is much harder without that.

Player's view of a game server:



SIMPLE! ***(Too Simple)***


Build
Patching (CDN)
"Login Servers" [Social]
GW2 services (e.g., tournaments)
Bots
Client
Actually…
"Map" Servers
Analytics
GW2 Commerce
…


Build
Patching (CDN)
"Login Servers" [Social]
GW2 services (e.g., tournaments)
Bots
Client
"Map" Servers
Analytics
GW2 Commerce
...
Orchestration

About 80 *types* of microservices

AccessKeySrv	CliGate	**ForwardSrv**	ListSrv	PresenceSrv	SsoAcctSrv
AccountSrv	ContentSrv	GameGate	LogQuerySrv	PvpGate	StreamSrv.dll
AdSrv	CryptSrv	GemStoreSrv	MailSrv	PvpQuerySrv	TournSearchSrv
AnProxySrv.dll	DataSrv	**GroupSrv**	MetricsSrv	PvpSrv	TournSrv
ArenaSrv (host)	DbNameSrv.dll	GwSrv.dll	NameSrv	PvpStatSrv	TradeSrv
Auth2fSrv	DbSqlSrv.dll	IndexSrv	OAuthSrv	ReflectionSrv	WalletProxySrv
AuthSrv	DbSrv	IpCheckSrv	PhoneSrv	RevenueSrv	WalletSrv
AuthSrv.dll	DeliverySrv	ItemDataSrv	PitchingSrv	**RouteSrv**	WebGate
BillingSrv	**DllSrv.dll**	ItemSearchSrv	PlayerlessSrv	SessGateSrv	
BrokerSrv	ErrLogSrv	JsSrv (V8 host)	P-Gate	SiteSrv.dll	+ ~12 Web Services
CensusSrv	EventLogSrv	LadderSrv	P-Proxy	SmsSrv	
ChannelSrv	ExchangeSrv	LeaderboardSrv	P-Srv	SpawnSrv (host)	

Hosts

- ArenaSrv.exe – hosts services that end in .dll
- SpawnSrv.exe – hosts processes
- JsSrv.exe – V8 (Javascript) host for web services

- Hosts listen on a single port and delegate connections

A Microservice Cluster

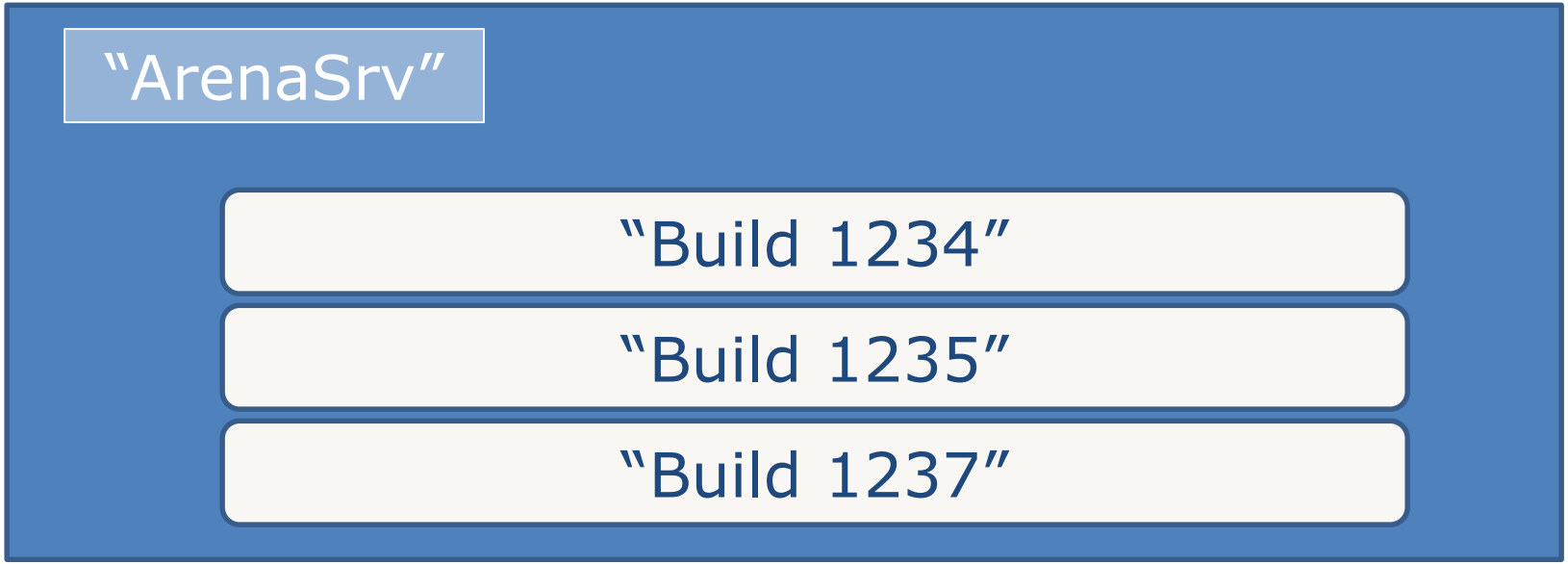
- PvpGate
- PvpQuerySrv
- PvpSrv
- PvpStatSrv
- TournSrv
- TournSearchSrv

The “Macroservice” (The Game)

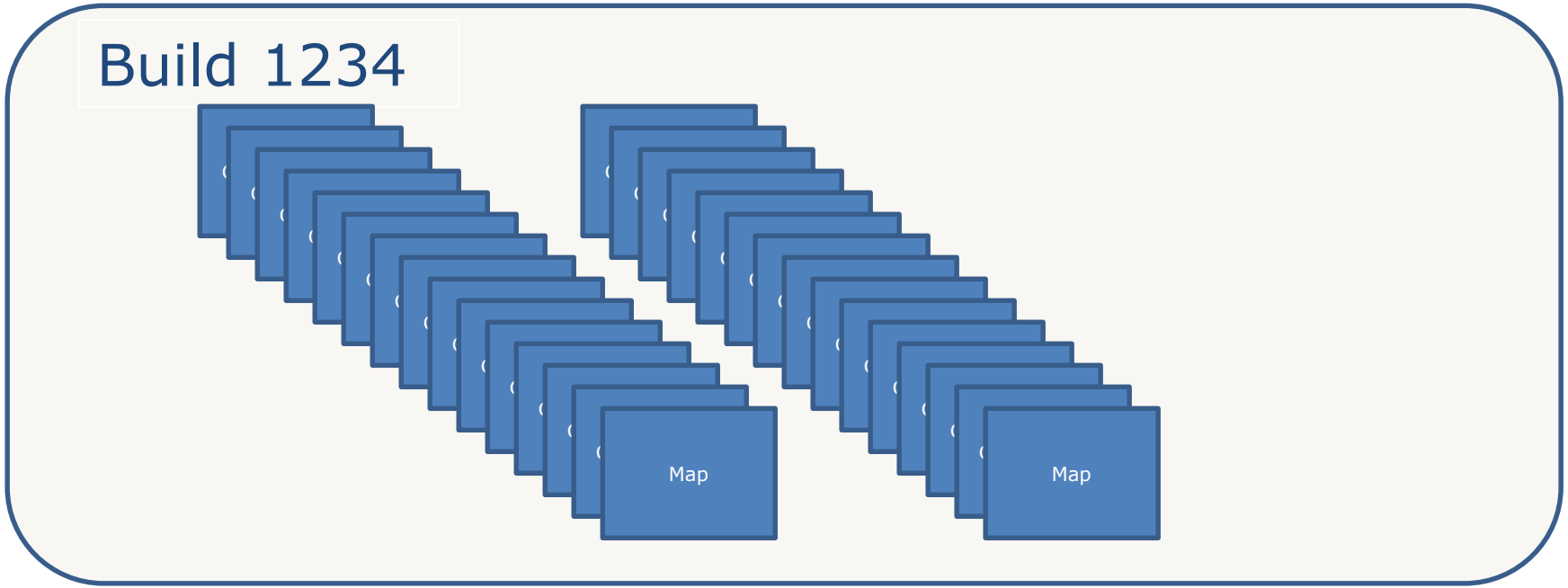
The “macroservice” that runs the game maps has nearly the same features as a microservice, including dynamic configuration and deployment and multiple simultaneous versions, but it’s a big piece of software; and individual maps (except World V World) are not restartable.

Similar principles guided it’s development but it is decidedly not “micro”. The game can run multiple builds at once.

Concurrent Builds



Inside a game build



Maps are time-sliced amongst a blade’s cores.

SpawnSrv.exe

- Can run multiple builds of microservices
- Starts up the new build
- Routes new traffic to it
- Ensures it runs without crashing
- Shuts down the old build
- A ”build” consists of services that talk to each other.

Routing (aka return address)

- Old way:
  - IP (from config file) + Port + “ProgramId” + Build + Map + PlayerId
  - PlayerId was a transitory value unique to a specific map server.
- New way: (Implicit) ProgramId + Player (or Group) Guid.

Part Three: Orchestration

Wikipedia:

"**Orchestration** is the automated arrangement, coordination, and management of computer systems, middleware, and services."

Orchestration at ArenaNet

- Allocation of services to hardware
- Permissions
- Routing
- Isolation

(BTW, we don’t use virtualization.)

Deploying Configurations

- There are two main configuration files with big lists of "server groups" which point to hardware.
- "Hardware addressing" is (still) by IP.
- Some hardware has multiple IPs and therefore looks like multiple machines.
- We got rid of Guild Wars 1 virtualization by assigning processes to IP addresses and letting the OS schedule. We know a lot about how the OS schedules things. ☺ VMs just added another layer of complexity to scheduling.

Blessing of hardware stability

- We own all of our equipment
- It’s made by a reputable vendor and has been stable for four+ years
- We control the datacenter infrastructure
- Stability gives us Predictability
- Predictability guided our deployment strategy.

Configurations

- A section naming service groups (“clusters”) and their services
- A section listing permissions
- A section mapping groups to IPs
- Configurations are dynamically updatable … but not automatically; humans make the changes

Simplest configuration:

127.0.0.1: all

all: (a list of all the services)

Developers can run the entire stack on their development machines

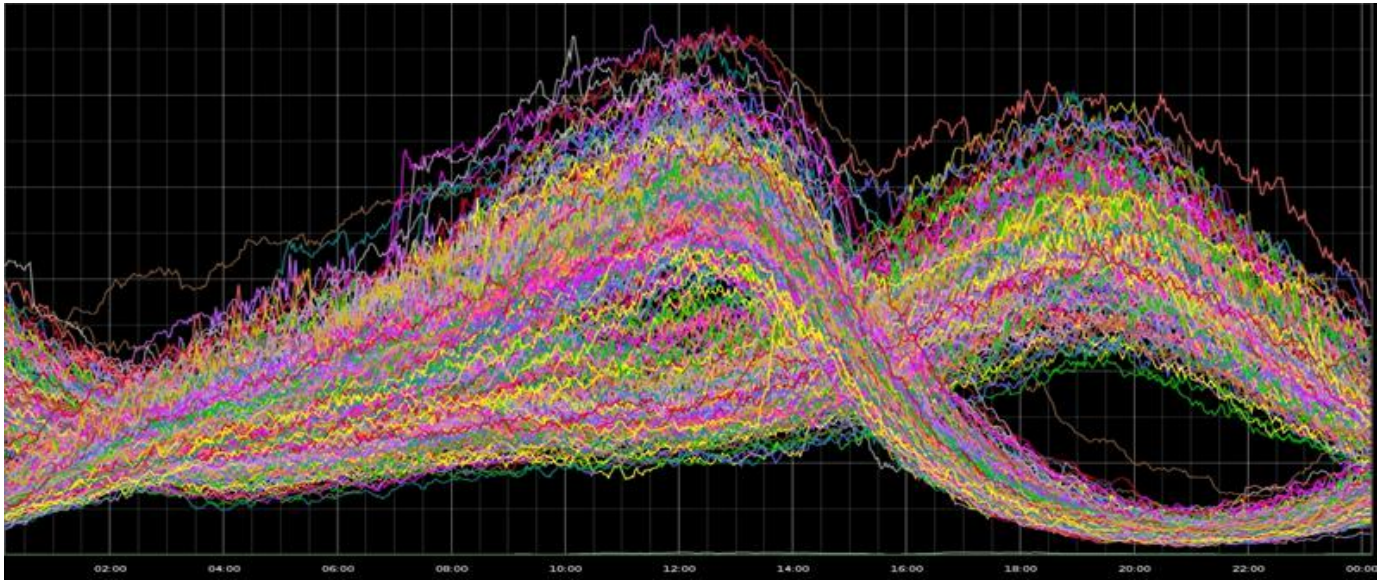
Configurations aid routing

- Routing is a combination of information in the configuration and self-registration
- In a perfect world, there would only be self-registration (and advertising over DNS)
- That’s not our world (yet)
  - Game (“Map”) servers self-register
  - Many services self-register with “RouteSrv”

Monitoring

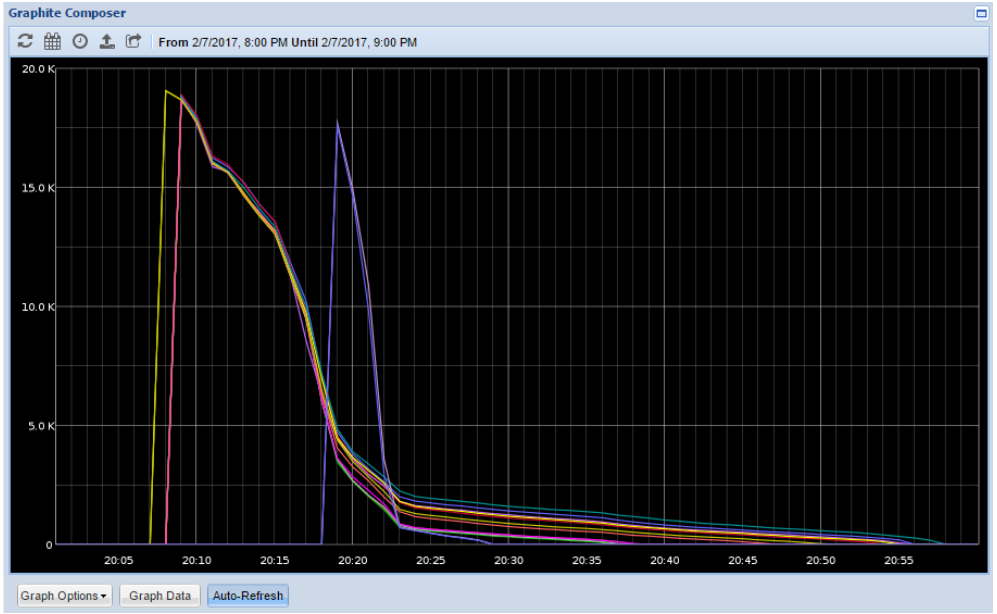
- Monitoring is the “end game” for orchestration
- We gather about a million time-sequenced data values per minute across both Guild Wars games
- Most sample code you will find in books or the web demonstrates use of logging; save it for when it matters
- AtomicAdd(&someVal, 1) is super cheap and aggregates well
- A separate microservice collects and aggregates perf data
- Perf data is visualized via Graphite (https://graphiteapp.org/)

Using visualizations:



This graph suggest removing dependency on hardware continuity

Using visualizations:

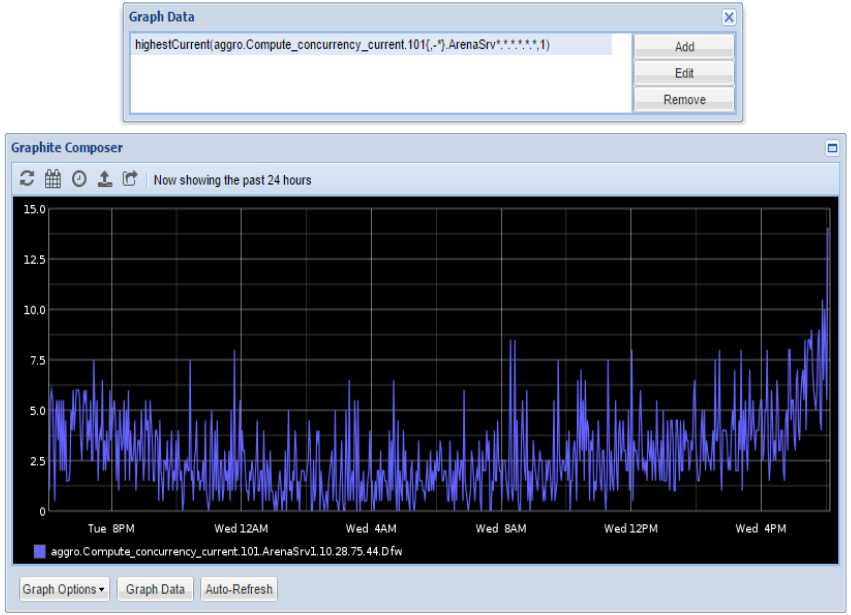


Watching a build dist (living world patch – 18K files – the night before release …)

Using visualizations:

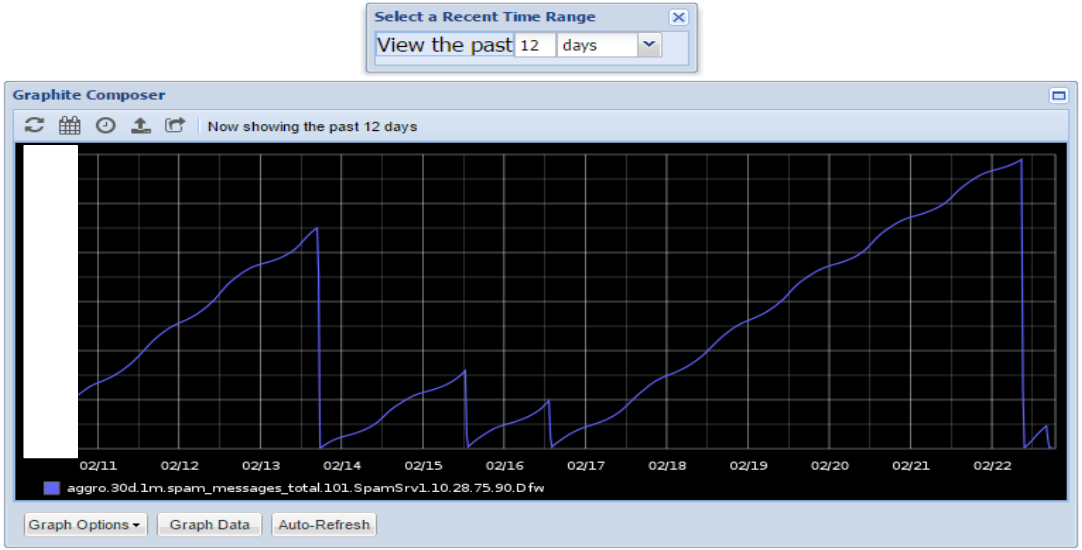
Number of cores / blade in use …

Using visualizations (aggregation):



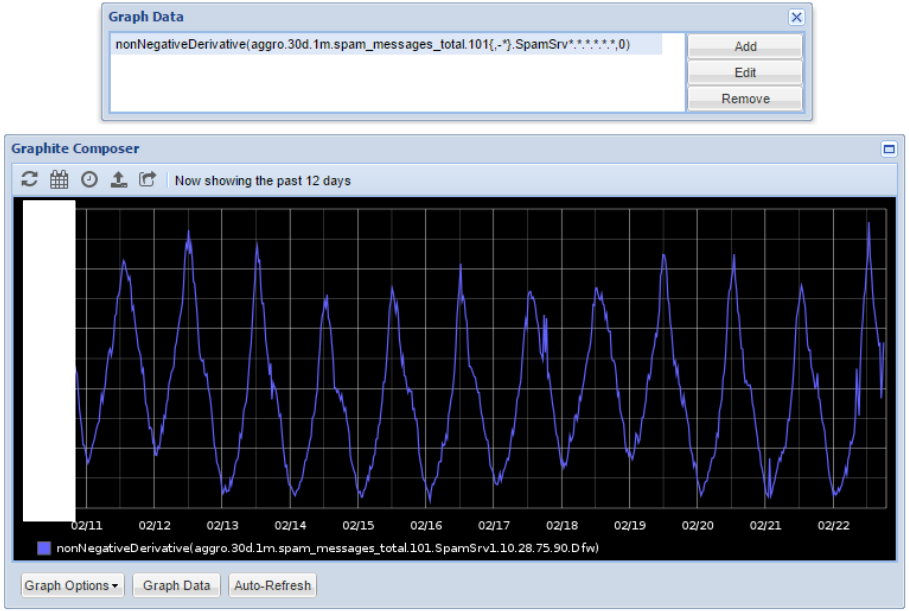
Max number of cores / blade in use …

Using visualizations:



Total spam reported by players (since server restart)

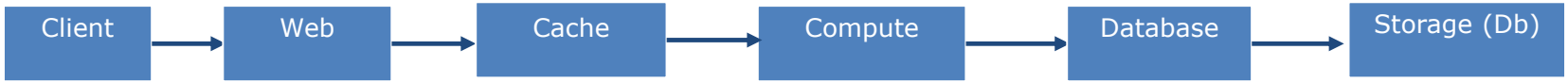
Using visualizations:



Spam reported by players transformed to non-negative derivative (shows rate; hides restarts)

Part Four: Restartability

“Simple” n-tier architecture



- Each layer optionally caches state from the upstream storage or service
- State recreated from upstream service

Why stateful services?

- "Most performance problems are solved with more RAM" – Bill Joy
- Local state is a highly optimized cache
- Generally non-authoritative

Naïve stateful recovery

- Store all state on disk or in a database for restart
- For instance, a tournament server could store the state of the tournament in case of restart
- … but what if you don’t know how much time passed between restarts?
- Data on disk goes stale quickly; tournament state may have changed

Authoritative Stateful Service

- A service that bosses other services around
- Generally by allocating resources
- Must have accurate view of reality
- Non-authoritative services can lag reality; authoritative services must be accurate

Authoritative Restart

- Storing (semi-)transient data on disk or in a database rarely reflects reality, because reality changes quickly
- So cache useful state locally in RAM and requery from **clients** on restart
- Not significantly different from n-tier case, except backwards!
- Possibly apocryphal story about Uber's catastrophic recovery plan.

Guild Wars 1 tournament state

- The Heroes' Ascent tournament ladder
  - A continuous global tournament between teams of 8 players
  - Each map knows who is in it and where they stand in the tournament
  - When the tournament server restarts, it queries each map to re-derive the state of the tournament
  - No tournament state is saved *by the tournament server* across restarts!

Guild Wars 2 World v World

- World vs. World vs. World maps have game state that persists for a week (or whatever you configure)
- The game state is stored in the load balancer *and* in the maps
- The load balancer can restart; and the maps can restart (e.g., when a new game build comes out); and as long as both don't restart at the same time the state is maintained
- (The load balancer secretly keeps an emergency backup of the state but it is ***rarely*** used – maybe once every few years.)

Guild Wars 1 Observer Mode

- "Stream server" used hybrid solution
- Allowed delayed playback while recording
- Server stored (possibly incomplete) recordings on disk
- Did crazy dance with game servers to continuing recording and/or playback after stream server restart
- Very hard; way easier (if we had the RAM) to keep the authoritative recording on the game server until completed

Transaction Restartability

- Restarting a service means it is off-line for 10 to 20 seconds
- It’s a pain to move the state to a sibling service for such a short outage
- So a strategy for retrying transactions is important for such a brief outage

Rules for retrying transactions

- Only retry on time-out – all other errors should be reported as errors and the transaction abandoned
- You should only *automatically* retry a few times (4-6), about every 30 seconds
- Most problems resolve themselves in a couple of minutes or not at all; might as well give up (which generally helps rather than hinders load issues, if that was the problem)
- Given application knowledge, you might make your own retry rules for specific transactions (e.g., game client patching retries forever)
- Random exponential retry is a waste of effort at the application level and just introduces uncertainty

The oddity of "reconnect before disconnect"

- You will want to write code that assumes that disconnects always occur before reconnects (e.g., caused by a server restart)
- And it will be true on your developer machine
- In reality, you can only trust the most recent connection from a specific machine/service

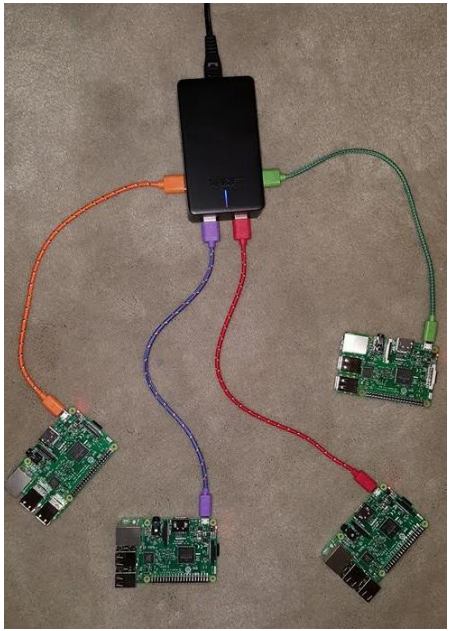
What goes wrong in the DC

- Client A is connected to Server B
- Client A (which is also typically a server for someone else) disconnects for an odd reason (OS crash, yanked cable, … ?)
- Server B does not get an immediate TCP Close; instead the connection times out after ***n*** seconds
- Client A restarts/reconnects to Server B before the old connection times out on Server B
- Server B thinks it has two connections! And only one of them works. And it is the second, newer connection. Hence "reconnect before disconnect." The fix is to only use the most recent connection, no matter what *you think* is in-flight on the old one.

Microservices and Orchestration: Fun you can have at home

- It’s a great time to be learning to program
- Not only can you “learn by doing” with free client software (Unity) but you can also do the same with server software

- Build your own datacenter cluster for $230.00
- Each Pi3 is $50.00, plus the power supply ($30.00)
- You can run Docker and other free software to host/orchestrate your microservices
- Four Pi's provides enough complexity to challenge you



4 PI's are enough to learn a lot, e.g.:

- Restarting all services at once (cold start)
  - IP addresses and DHCP
- Deploy your services automatically
- Image your OS and environment
- Monitor health (esp if you put them on the Live internet!)
- Backup / Restore!

Summary

- ArenaNet has been doing Microservices for fifteen years, except we didn’t call them that
- Restartability was engineered into every server (except the game map server) from day one
- Adding it later is *hard* (WvWvW case)
- You can build your own datacenter for $230.00 and have fun with microservices!

Contact info: stephen@arena.net
Remember to fill out your evaluations
Thanks for listening