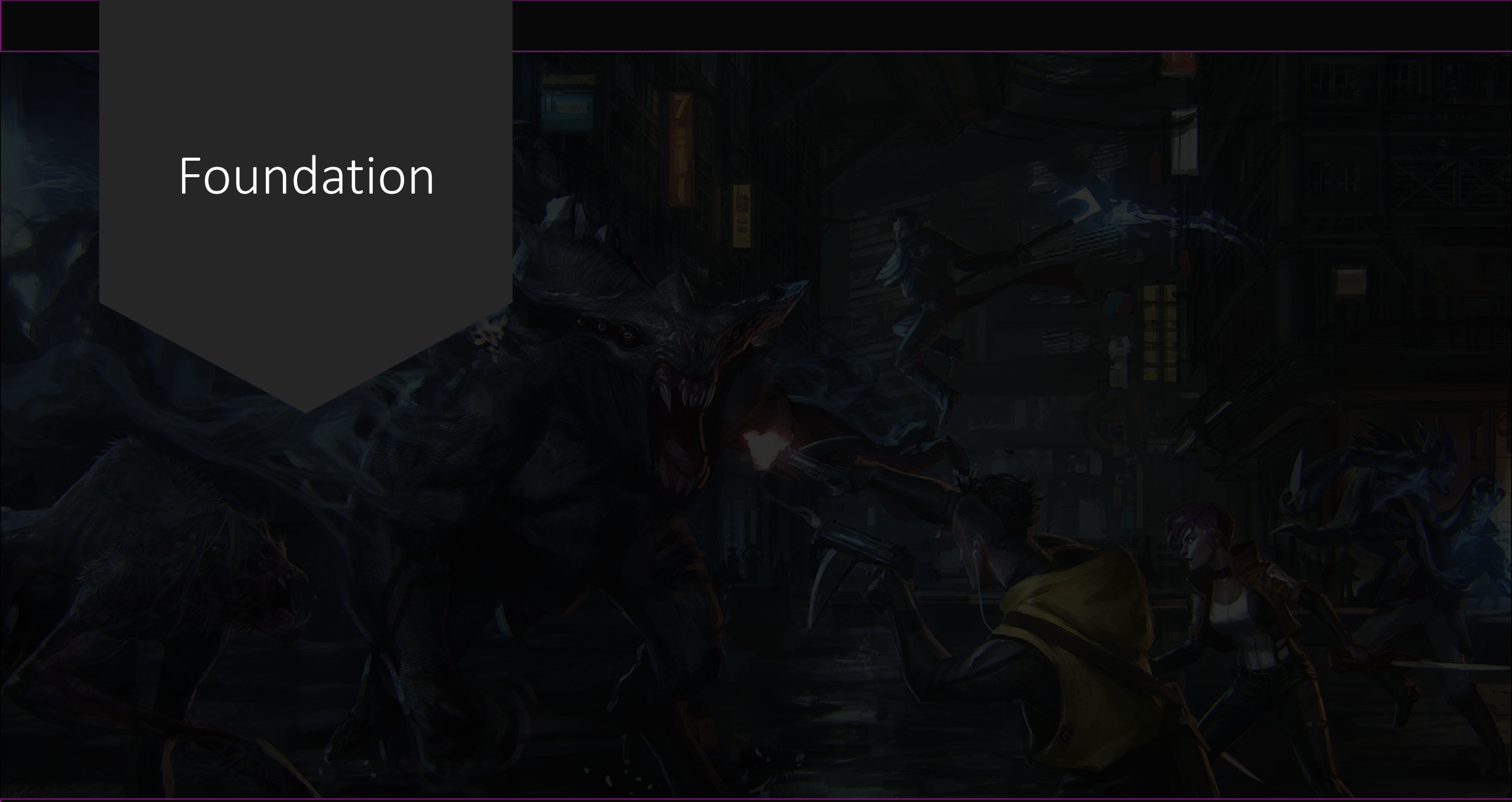
QC Games

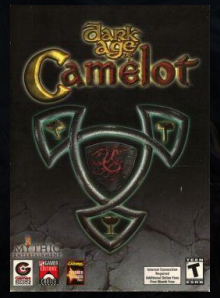
Technology Overview

Foundation



Team History

- The QC Games team has decades of experience building and operating multiplayer experiences from Dark Age of Camelot to DA: Inquisition
  - Developed rich, critically acclaimed multiplayer experiences
  - Supported games as live services
  - Operated games at massive scale



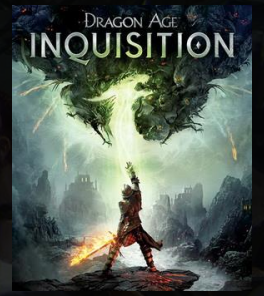
DAOC
Released: Oct 2001

Warhammer
Released: Sep 2008

SW:Galaxies
Released: Jun 2003

Mass Effect
Released: Nov 2007

SW:TOR
Released: Dec 2011



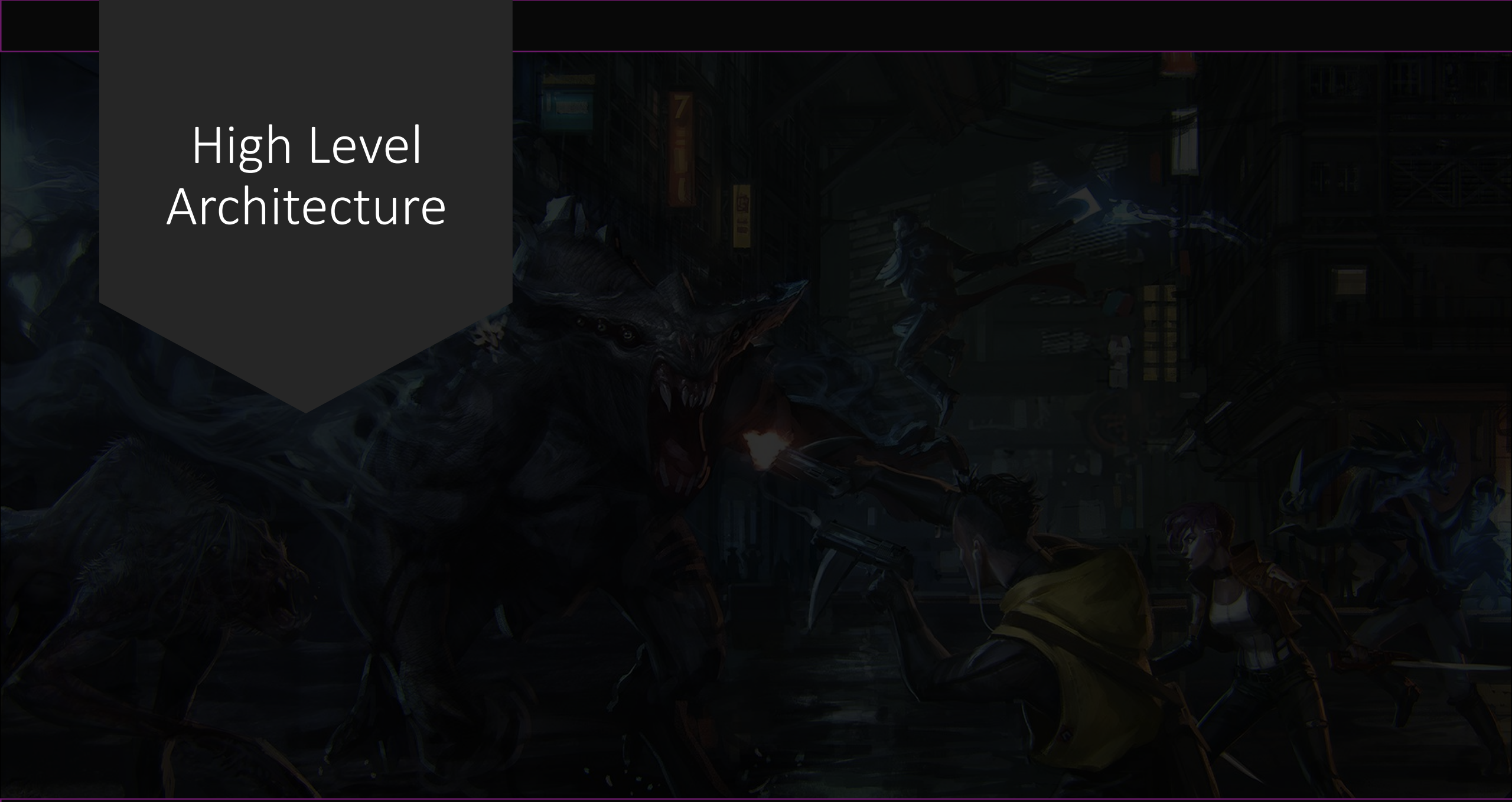
DA: Inquisition
Released: Nov 2014

- Built Breach from the ground up with an engineering team of ~10 over 3 years

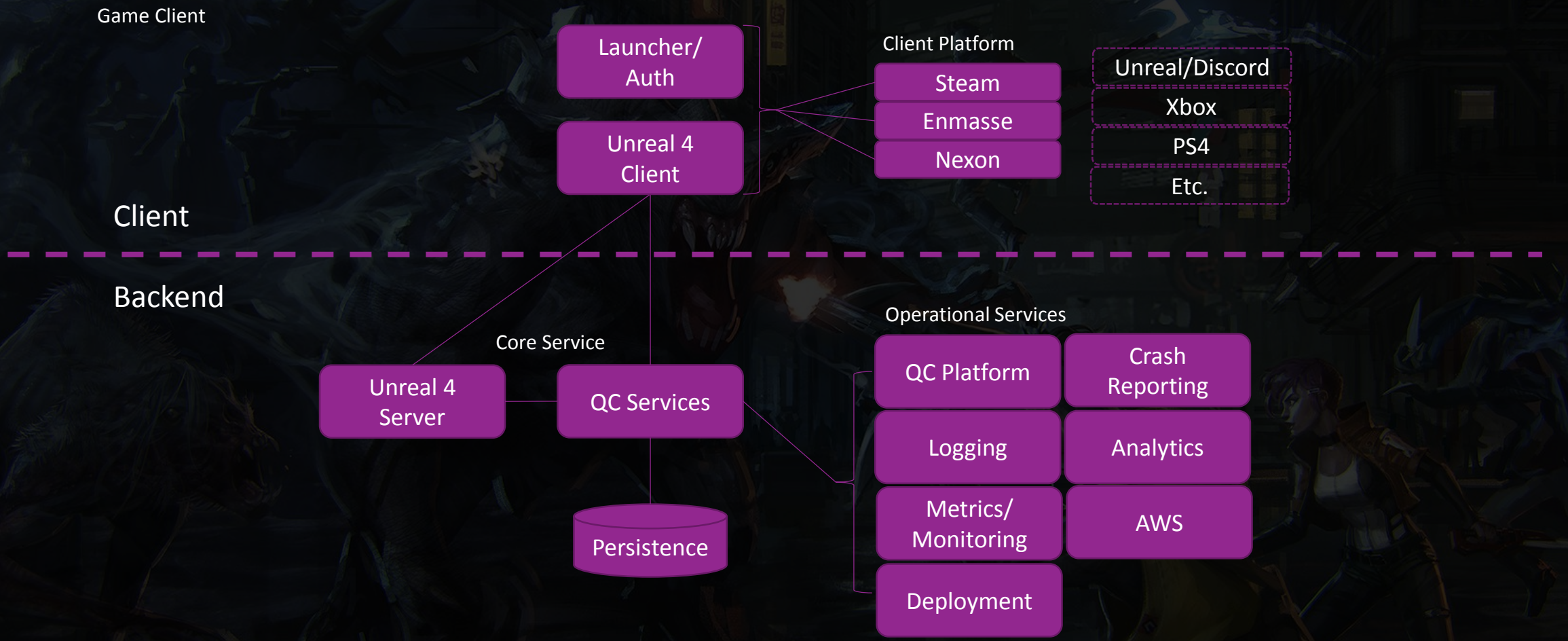
Technology Stack Goals

- The team set out with the following goals for QC Games' technology
  - Ease of use and robustness: from writing code and developing content to live operations
    - Focus on developer friendly workflows to maximize the team's velocity
  - Create a tech stack that can be enhanced and maintained for the life of the service
    - QC Games' technology needed to support constant updating post launch, which is critical for live service games
  - Platform Agnostic
    - All technology decisions from game engine to service architecture were made to support a number of a potential client platforms
    - The service stack can support multiple client platforms simultaneously enabling cross play and cross progression seamlessly

High Level Architecture



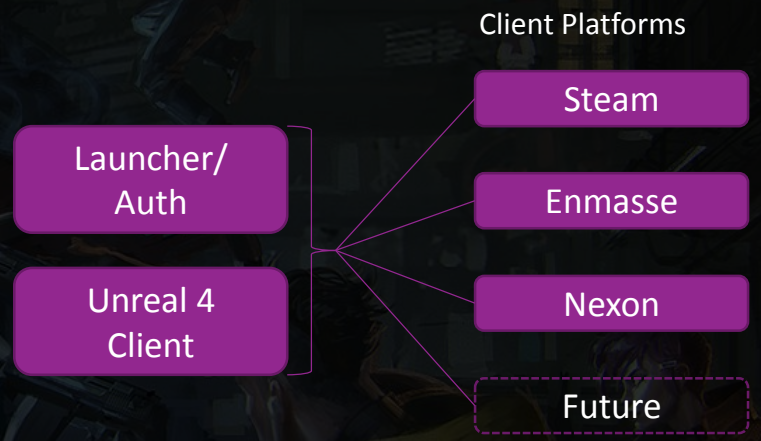
Breach High Level Architecture



Client/Server Architecture

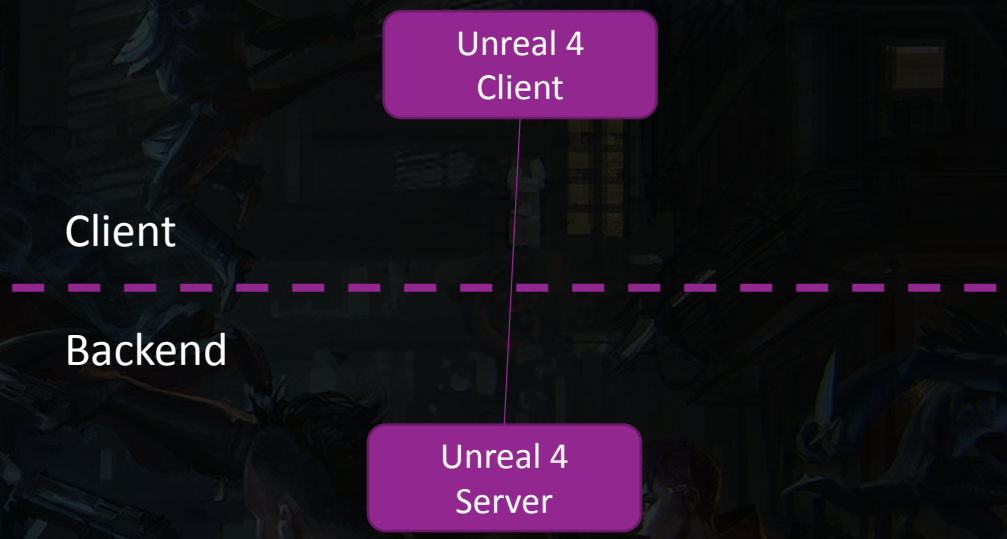
Client Platform

- Integrate with each distribution platform to distribute/patch the client binaries and content
- Custom authentication integration per platform to establish ownership and identity
- Platform identity is used to “link” platform accounts with an internal account to support platform agnostic cross play and cross progression
- Future platforms will be integrated in this way (Epic, Discord, PS4, etc.)



Game Simulation Architecture

- Fully native Unreal 4 client/server implementation for real-time game simulation
- Well established out-of-the-box development workflows
- Documented and supported by the Unreal community and Epic
- The QC Games team builds custom systems on top of the core Unreal workflows
- Using native Unreal network protocols for real-time client/server communication



Core Breach Requirements

- The following core requirements drove the architecture of Breach's client and server
  - Online Co-op/PvP action RPG combat where positioning, timing, animation frames, and aim matter.
  - Real weapon and projectile collision checks determine if hits occur, to feel like a true action game.
  - Server authoritative for a fair game.
  - Must feel responsive, similar to a local offline game, even though it's a server/client game.
  - Allow designers to generate 100s of abilities and combat features quickly and efficiently.

Development Toolbox

- The Breach Gameplay team began with a very robust toolbox of designer facing features that they have used to build up more complex combat behaviors
  - A timeline system to choreograph a series of events/animations/collision checks/etc while some ability is executing
  - Re-use that same timeline system for executing buffs/debuffs
  - A fully featured data-driven gameplay RPG stats system.
  - Re-usable systems for targeting and querying gameplay state
  - Action game staples: Beams, AOEs, auras, projectiles
  - User friendly tools to edit and debug combat data
- All of these systems are incredibly data-driven and QC Games' designers can build and iterate abilities, creatures, AIs, level objectives very quickly, before we begin building any final art for them

Client Prediction

- In order for the game to feel responsive, Breach's combat system runs nearly all the same logic on the client and server in parallel
  - The client predicts everything
  - When an ability is requested it will play animations and spawn projectiles immediately without waiting for the server
  - When projectiles hit it will play explosions and flinches without the servers authorization
- This can cause mispredictions and the Breach Gameplay team has architected numerous techniques to reduce the side-effects
  - All combat systems are created with the idea that everything will run on both the client and server, which means that we usually get the same outcome
  - Use expected client latency to offset the playback of timelines on the client/server so that they better line up
  - Detect predicted actions and abort them within a certain time window if no matching server action arrives
  - A gameplay state debugger that allows us to analyze how state changes on the client and server frame by frame, to debug and resolve issues involving misprediction
  - Most mispredictions are actually quite benign and are difficult for a player to ever notice – these can safely be ignored
  - In a few rare cases, mispredictions are particularly awful even if they happen extremely rarely, and the system accepts round trip latency instead. Knocking a character back through the air is one example.

Service Architecture

Service Goals

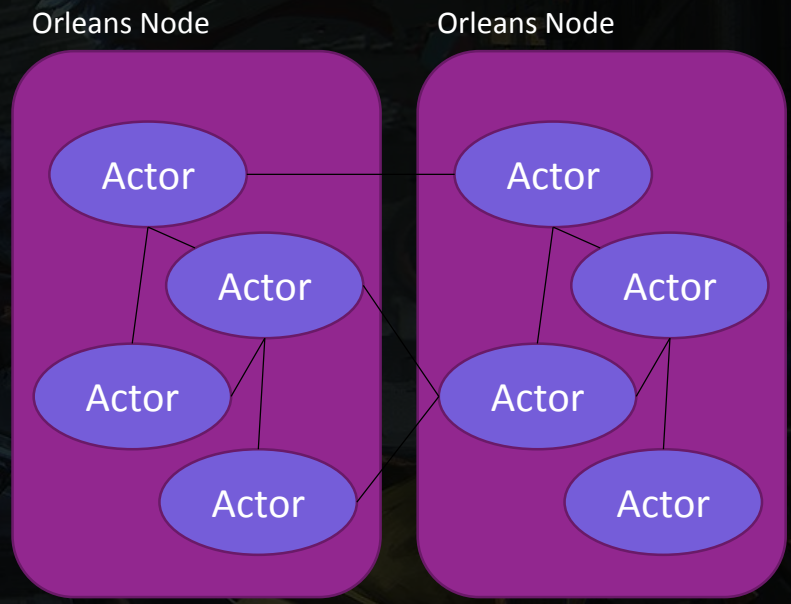
- Provide latency tolerant services to the game server and client
- Highly scalable framework for massive scale
- Single global shard for all platforms to support cross play and cross progression
- Programmer friendly model for developing highly distributed and scalable services

Service Architecture

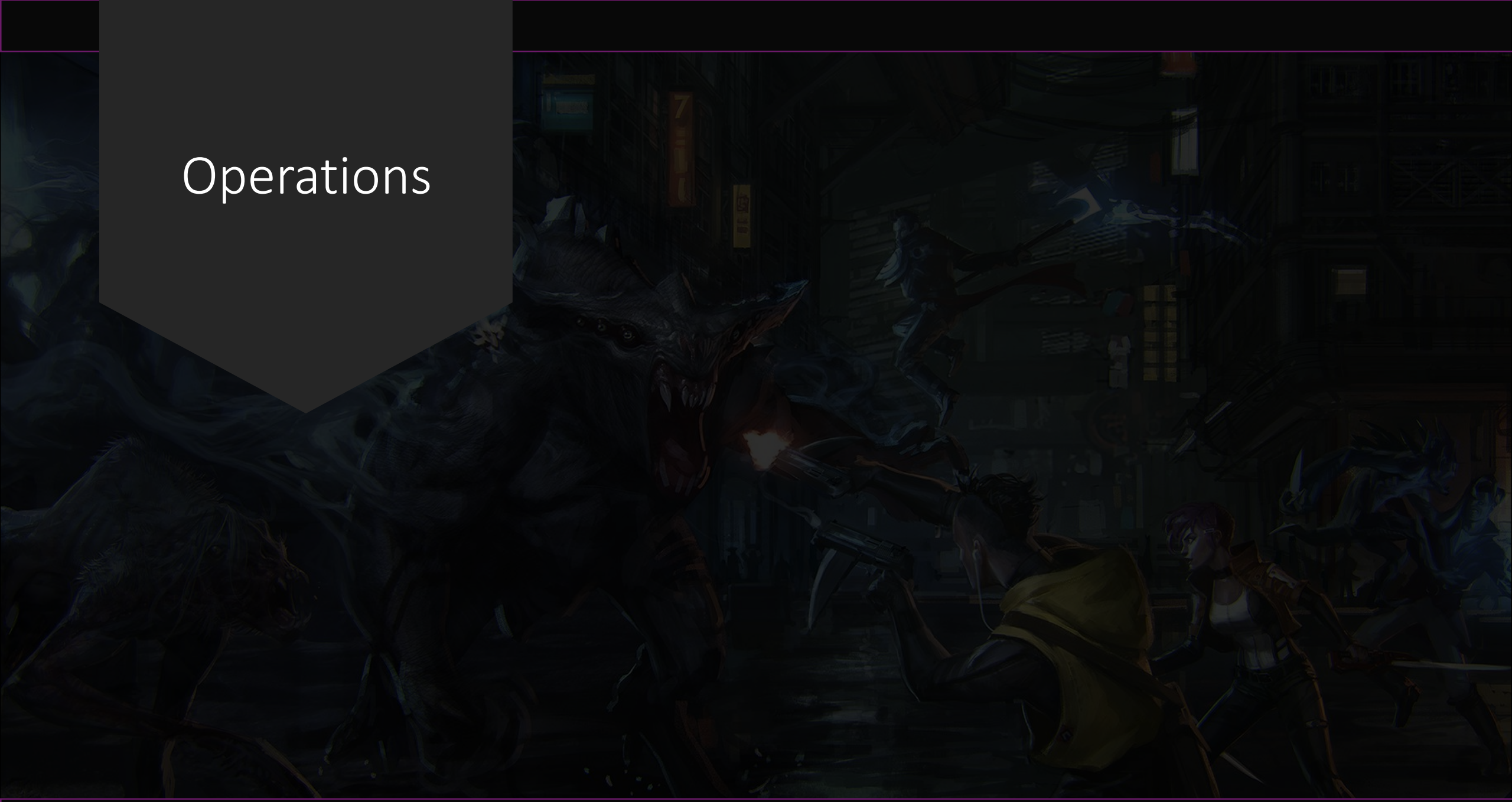
- SignalR communication provides bi-directional communication with the client and game servers
- Horizontally scalable ASP.NET front-end provides security and identity services
- Project Orleans based back-end provides horizontal scale and a programming model for building highly distributed services
- MongoDB used for persistence of all account data

Project Orleans

- An actor model framework used to develop highly distributable objects across a cluster of nodes
- Allows for trivial horizontal scale of the entire service
- All “Actors” communicate asynchronously in a location transparent way
  - It doesn’t matter if the component I am talking to is in the same process or on another host in another data center
- All services are authored using this framework including:
  - Platform authentication, commerce, chat, friends, matchmaking, account persistence and versioning, game server host management and scaling, and many more



Operations

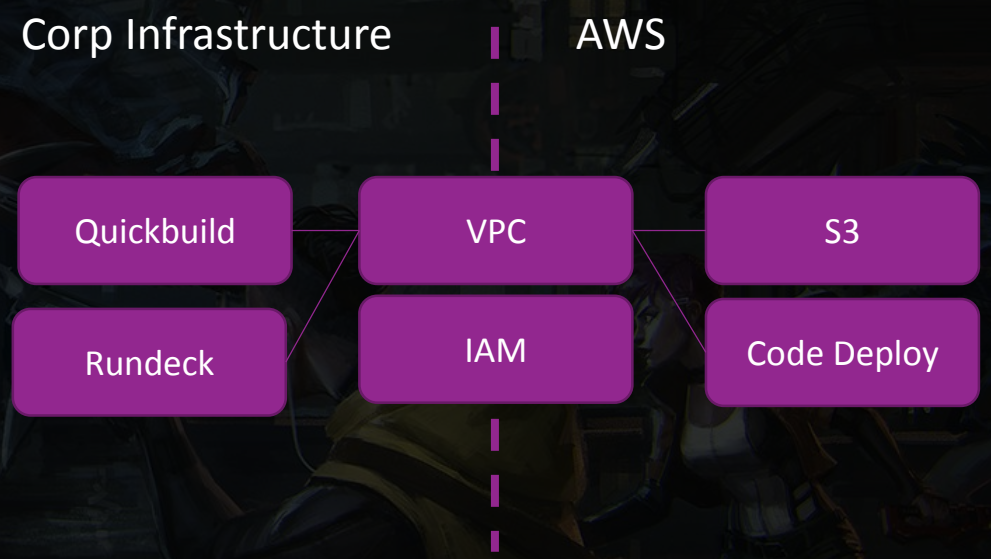


Easy Setup and Operation

- Goals
  - Operations is a constant task, so, as with every aspect of QC Games' stack, we wanted to ensure operations workflows are robust and easy to use
    - Easy scalability is paramount
    - Deployments need to be quick and fault tolerant
    - Environment setup should be easy and repeatable
    - Alerting is critical for a small ops team
    - Monitoring and logging needs to be easily accessible for diagnosing problems
    - Thoroughly test operations processes by running the development environments using the same setup as production

Deployment Infrastructure

- Quickbuild and Rundeck used to automate every aspect of deployment/operations
  - QC Games' Operations has a single operation to deploy a build to the cluster and restart the service
- VPC used to connect development with AWS
- IAM used to control access to AWS environments
- S3 used for build storage
- Code Deploy used to manage pushing builds to the entire cluster

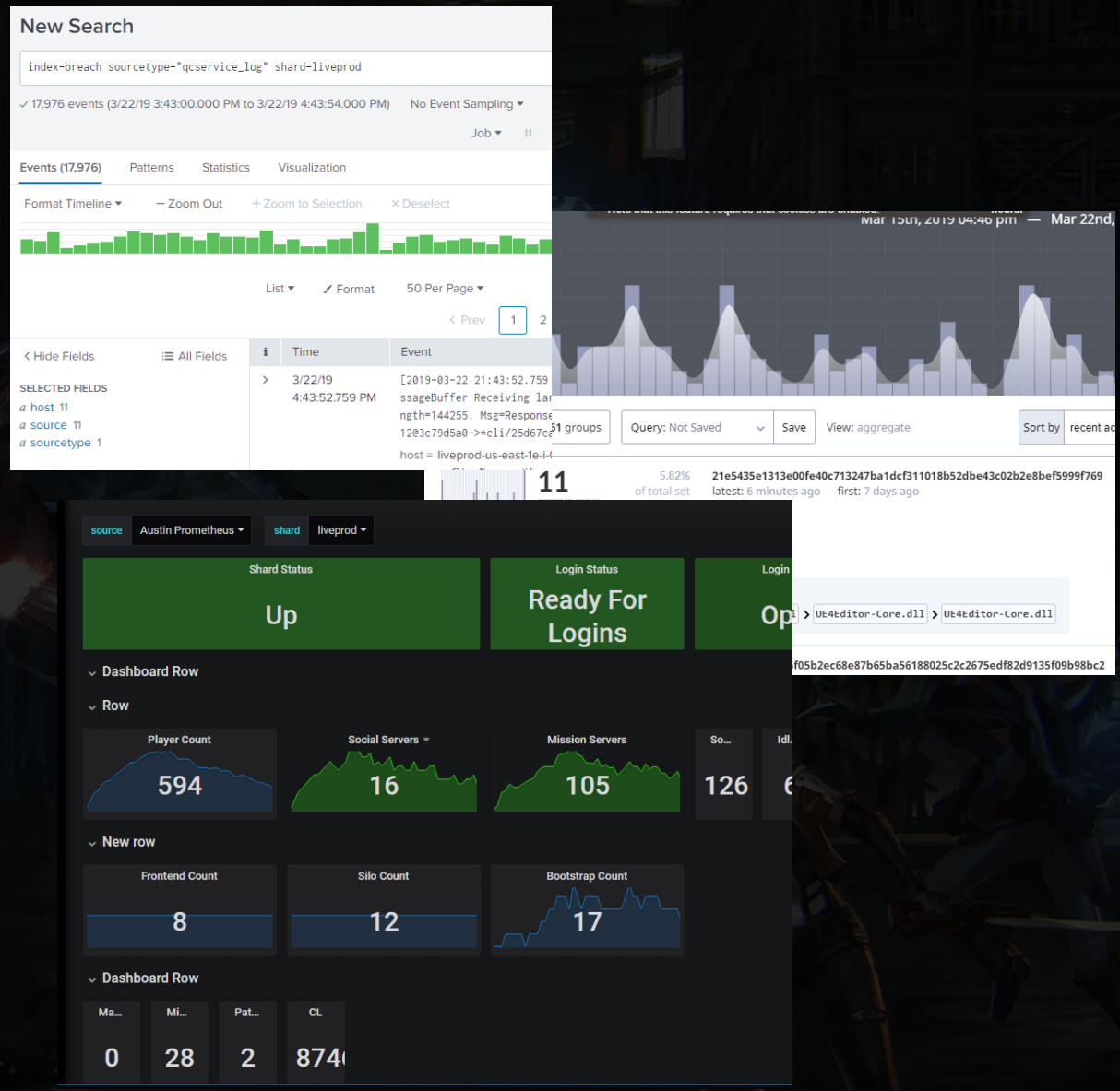


Runtime Infrastructure

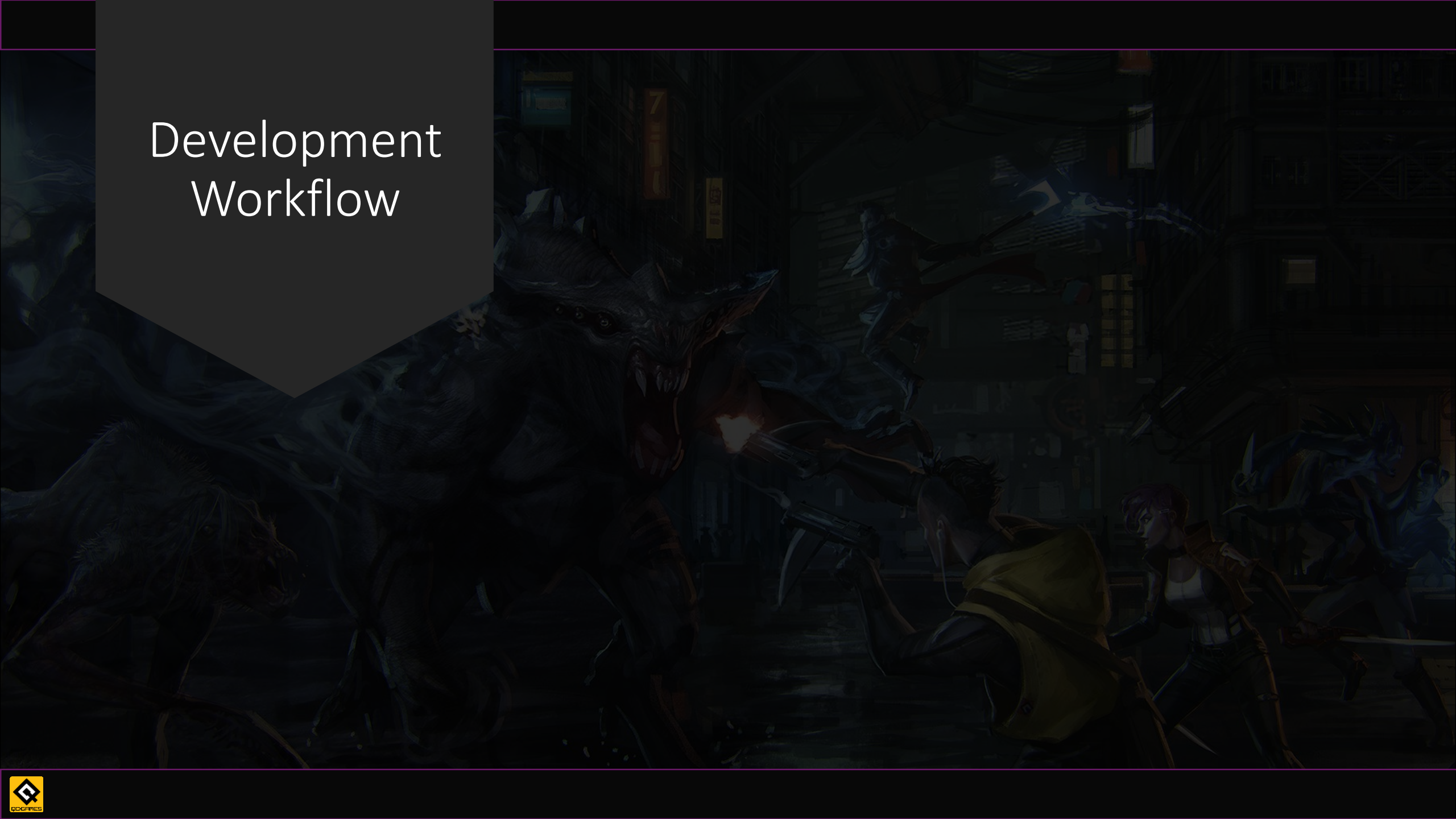
- AWS Load-Balancers used to distribute incoming client connections
- Auto-scale groups used to easily scale components up and down
- Game server scaling is done automatically by the service based on player load
- Using a managed shared and replicated MongoDB cluster for all persistence
- This entire setup can be stamped out using a “shard creation” script to ensure a reproducible and repeatable environment

Key Operational Tools

- Splunk
  - Centralized logging that collects logs from every host in the cluster
  - Powerful search engine for diagnosing issues
- Backtrace
  - Client crash reporting service
  - Critical to find high frequency crashes so we can improve the stability of the game
- Prometheus/Grafana
  - Dashboards for monitoring the service
- Zabbix
  - Watches key metrics for signs of issues and notifies PagerDuty
- PagerDuty
  - Notifies all on-call operations personnel when there is a critical issue
- DeltaDNA
  - Analytics platform used to store all telemetry
  - Dashboard provide quick visibility into KPIs
  - Combined with Tableau and various other analytics tools for deep understanding of how the business is performing



Development Workflow



Productivity is Key

- Goals
  - Easy setup for artists, designers, and engineers
  - Simple daily workflow for getting your environment up to date
  - Easy download and play for fast iteration
  - Surface issues as quickly as possible through CI/CD

Workflow Tools

- Unreal 4
  - World class tools for creating content and systems
  - Data-driven systems for maximum flexibility and design creativity
- QC Now
  - Simple tool to update your working environment or download and play the latest builds
- Continuous Integration
  - Quickbuild used for all build automation
  - Always building code and content to find issues fast
- Continuous Deployment
  - Unattended deployment of nightly builds to development environments in AWS
  - Simple process to start a new full build for testing