

Technical Presentation

Architecture, AI, and Engineering Behind ObjectDash

26 slides

Contents



Let's Play

Get out your phone and go to:



Scan to play ObjectDash

[Click here to play the game](#)

www.objectdash.com

What you're about to do:

1. **Look** at an AI-generated image
2. **Find** the hidden objects before the timer runs out
3. **Score** points for speed and accuracy
4. **Compete** on today's leaderboard

A brand new set of images is published every day at midnight. Come back tomorrow — the images will be completely different.



Presentation content current as of February 22, 2026.



What is ObjectDash?

ObjectDash is a daily AI-generated picture-guessing game.

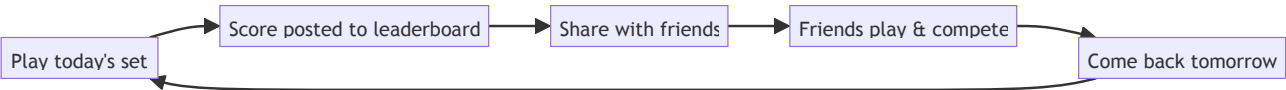
Every day, five brand-new images are generated by AI. Each image hides a set of real objects — your job is to find them before the clock runs out.

How a Game Works

```
Game (1 per day)
└─ Round 1 - 30 seconds → Find the hidden objects
└─ Round 2 - 30 seconds → New image, new objects
└─ Round 3 - 30 seconds
└─ Round 4 - 30 seconds
└─ Round 5 - 30 seconds
    └─ Score → Leaderboard
```

- Every correct selection earns points — **speed matters**
- A perfect round with no wrong clicks is **flawless** — worth a bonus
- Complete all 5 rounds to post your **daily score**
- **Streaks**, achievements, and a global leaderboard keep you coming back

The Viral Loop



💡 Why I Built It

The Original Idea

When **DALL-E 3** was first released, I had an idea: *What if an AI generated a scene with hidden objects, and players had to find them?*

I started sketching it out — but the project I had in mind was too ambitious. I set it aside.

What Changed: Agentic Software Development

Last year, **AI-assisted development went mainstream**. Not just code completion — full agentic workflows that could implement specs, write tests, and iterate on complex features.

Suddenly the project was achievable. I picked it back up.

Why These Specific Technologies?

Three things shaped the tech choices:

Motivation	Technology
I work in Microsoft's partner org with Elastic as a managed partner	Azure + Elastic Cloud
I wanted to demonstrate Azure's serverless scaling capabilities	Azure Functions Flex Consumption
I wanted to show the power of .NET 10 for cloud-native apps	.NET 10 Isolated Worker
I wanted to explore Elastic's newer AI features in a real app	Elastic Agent Builder, ES

The Constraint That Made It Interesting

73 days. One developer. Nights and weekends only.

Everything — backend, frontend, infrastructure, AI pipelines, agents, analytics — built from scratch using **GitHub Copilot** and a structured spec methodology called **Spec Kit**.

📦 Codebase & Azure Functions

Codebase Size

Metric	Value
Total source files	855
Total lines of code	~93,000
C# (.NET backend)	~64,000 lines
TypeScript / TSX (frontend)	~29,000 lines
Bicep (infrastructure as code)	~2,500 lines
Backend tests (.NET / xUnit)	627 tests across 87 files
Frontend tests (Playwright E2E)	153 tests across 15 files
Total automated tests	780
Primary Copilot models	Claude Opus 4.5, Claude Opus 4.6

80 Azure Functions

Trigger Type	Count	Description
HTTP	66	Game API, Admin API, Agent endpoints, Scheduled jobs
Queue	7	Image generation, validation, title gen, Elastic sync
MCP Tool	7	Database queries, KQL analytics, data file access

HTTP Functions by Domain

Domain	Count	Examples
Gameplay	12	StartGame, StartRound, EndRound, RecordSelection
Player Profile	11	GetProfile, CreateProfile, ClaimProfile, GenerateAlias
Image / Admin	14	Generate, ListImages, Delete, Restore, Purge, Search
Daily Sets	7	GetDailySets, AssignDailySet, GetImageForGameplay
AI Agents	11	AgentChat, DeployedAgent x5, ElasticAgent x3
Leaderboards / Stats	3	GetLeaderboards, GetAchievements, GetStatistics
Telemetry / Feedback	5	RecordEvent, RecordBatch, SubmitFeedback, GetFeedback
Infra / Auth	3	HealthCheck, UpdatePlayerRole, SearchPlayers



Repository Stats

By the Numbers

Metric	Value
Total commits	317
Total branches	59
GitHub Copilot branches	17
Copilot-authored PRs merged	10+

Metric	Value
Project duration	73 days (Dec 11 – Feb 22)

GitHub Copilot as a Collaborator

17 of the 59 branches were **created and worked by GitHub Copilot** — not just suggestions in the editor, but full agentic coding sessions: reading specs, creating files, writing tests, opening PRs.

Total branches:	59
└ Human branches:	42
└ Copilot branches:	17 (29% of all branches)
└└ Merged PRs:	10+

Copilot didn't just assist — it was treated as a team member with its own branches and pull requests.

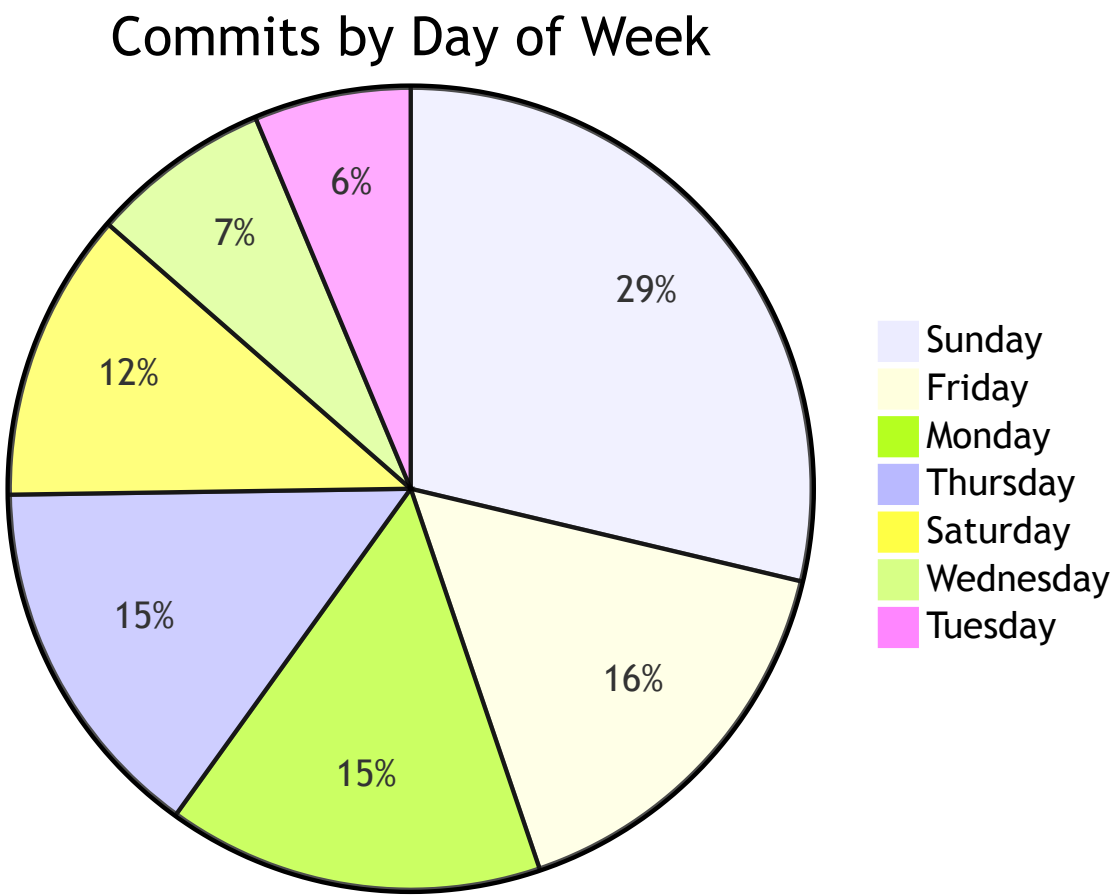
What the Branches Represent

Each branch typically corresponds to one **Spec Kit task set** — a focused implementation of a single user story or technical component. Copilot branches were used when the task was well-defined enough to hand off entirely to the agent.

When Was This Built?

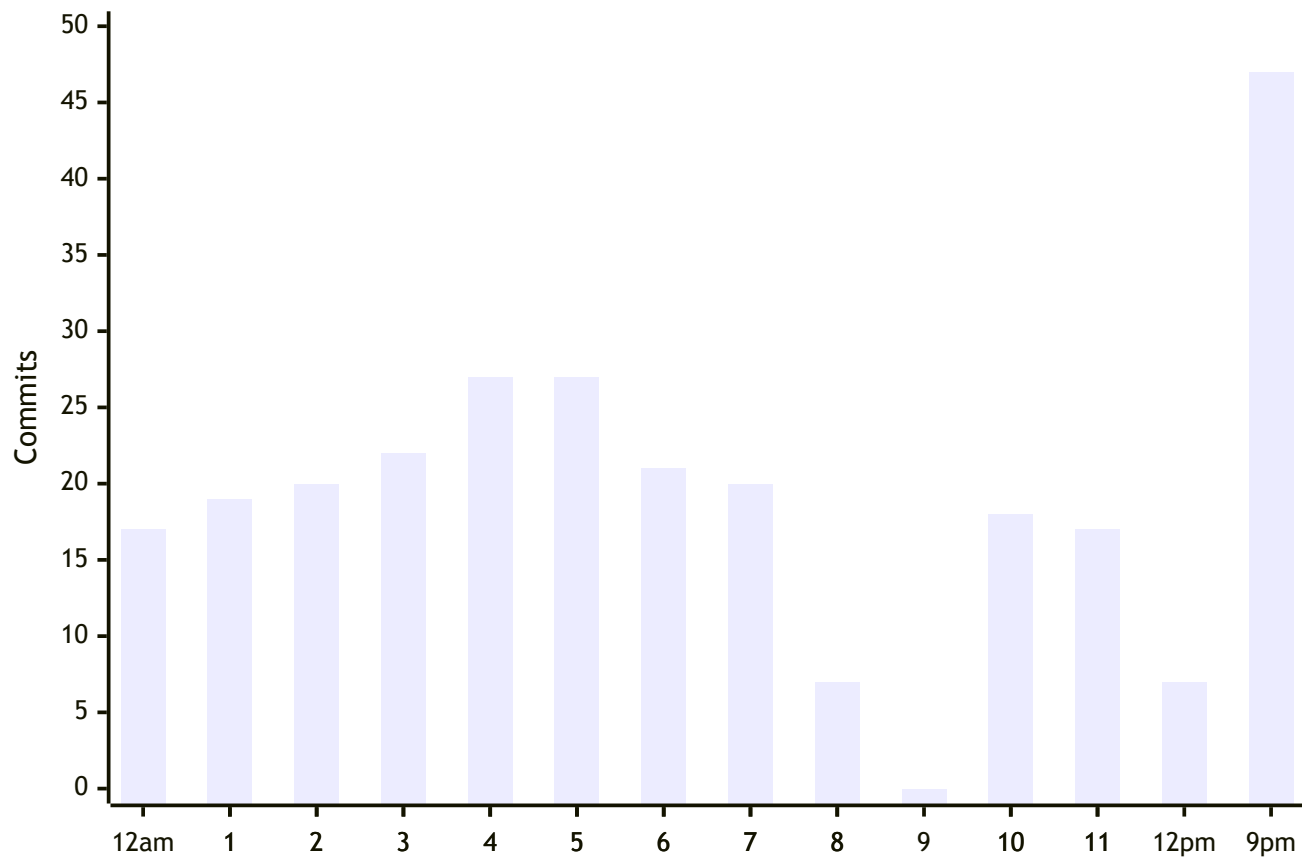
73 days · 317 commits · nights and weekends only

Commits by Day of Week



Commits by Hour of Day (Eastern Time)

Commits by Hour of Day (UTC-5)



What the Data Says

Observation	Detail
Sunday is #1	91 commits — 29% of all activity
9 PM is peak hour	47 commits — nearly 15% of all activity
40% on weekends	Saturday + Sunday = 128 / 317 commits
22% late night	10 PM – 2 AM = 70 commits
Only 8% before noon	This project runs on evening energy

This wasn't built during a hackathon or on company time. It was built at night, on weekends, one spec at a time.



What is Spec Kit?

Spec Kit is a structured feature development methodology designed for AI-assisted coding.

The idea: give GitHub Copilot (or any coding agent) a well-structured specification, and it can implement the feature end-to-end with minimal hand-holding.

A Spec is a Package of Artifacts

```
specs/  
└─ 015-tyche-chat-agent/  
    ├── spec.md           ← What we're building and why (user stories)  
    ├── plan.md           ← Technical approach and design decisions  
    ├── data-model.md     ← Schema and entity definitions  
    ├── tasks.md          ← Ordered task breakdown (what to build, step by step)  
    └── contracts/        ← API contracts, interfaces, expected behaviors
```

The Workflow



Why It Works

- **Specs are machine-readable** — Copilot can read the full spec before writing a line of code
- **Tasks are atomic** — each task is small enough for one agentic session
- **Context window-friendly** — the spec fits in a prompt; no hallucination about requirements
- **Human stays in control** — you write the spec, Copilot implements it

Spec Kit adds a small amount of up-front work — but the payoff is measurable. You'll see the velocity data on the next slides.

Note: Not every feature needed a spec. Many smaller features — bug fixes, UI tweaks, configuration changes — were implemented directly with GitHub Copilot without formal specs. Spec Kit was reserved for large features that spanned frontend, backend, and database changes.













Spec Inventory

17 specs · 93 user stories · ~1,189 tasks · all complete 

Stats

Metric	Value
Total specs	17 (001 – 017)
Total user stories	93
Total spec tasks	~1,189
Avg tasks per spec	~74
Specs completed	17 / 17 

All 17 Specs

#	Feature	Tasks	Completed
001	Image Generation Pipeline	121	 Dec 2025
002	AI Guessing Game (core)	271	 Jan 2026
003	Round Feedback	63	 Jan 25
004	Game Narrative Agent	67	 Jan 26
005	GPT Image Generator	42	 Jan 28
006	Player Profile Page	86	 Jan 29
007	Elasticsearch Round Sync	45	 Jan 30
008	.NET 10 Upgrade	43	 Jan 31
009	Production Deployment Infra	57	 Feb 5
010	UI Layout Refresh	106	 Feb 7
011	Practice Daily Sets	78	 Feb 8
012	Player PIN Login	41	 Feb 8

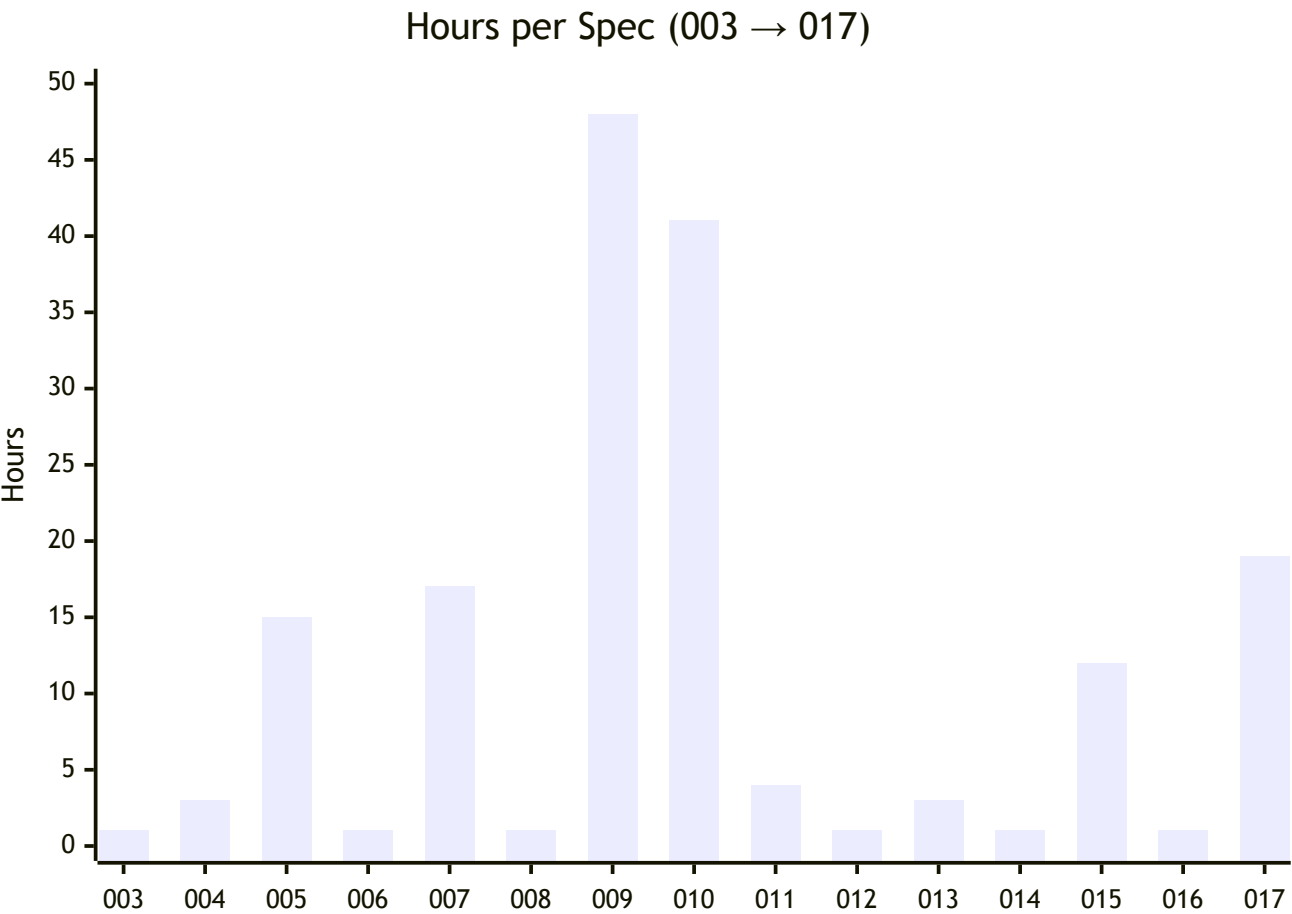
#	Feature	Tasks	Completed
013	SWA DailySet Images via CDN	54	✅ Feb 8
014	MCP Server Tools	22	✅ Feb 10
015	Tyche Chat Agent	52	✅ Feb 12
016	Deployed Agent	41	✅ Feb 14
017	Elastic Agent Chat	—	✅ Feb 22

The first two specs (001, 002) are the foundation — they took weeks. Everything after was built on top of a working, tested codebase. Watch what happens to velocity.

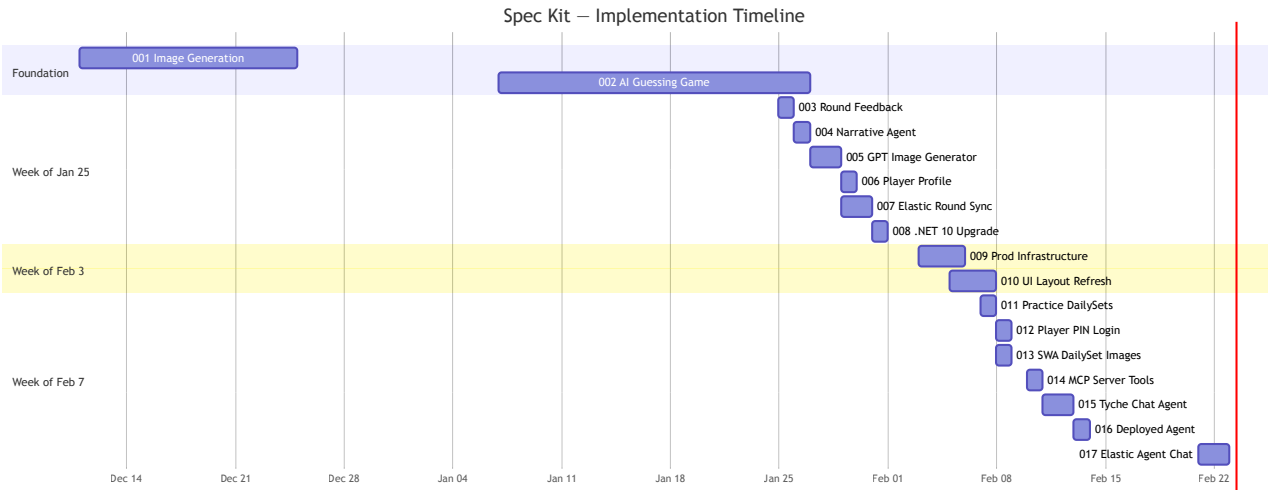
🚀 Implementation Velocity

Hours from first commit to last commit per spec (specs 003–017)

Velocity Chart



Timeline



What the Data Shows

Period	Specs	Observation
Dec – Jan	001–002	Foundation built — weeks, not hours
Jan 25–31	003–008	6 features in 7 days — avg ~6 hrs each
Feb 7–14	011–016	6 features in 7 days — avg ~3 hrs each (2× faster)

Standouts: - Spec 012 (Player PIN Login) and Spec 008 (.NET 10 Upgrade) — **under 1 hour** end-to-end - Spec 017 (Elastic Agent Chat) — full-stack SSE streaming chat with 46 tests, done in **~19 hours** - Spec 009 & 010 (Infra + UI redesign) — expected outliers at 2–3 days each

The pattern is clear: the more mature the codebase, the faster each spec ships. Spec Kit + GitHub Copilot creates a compounding acceleration effect.

Tech Stack

Layer	Technology
Frontend	React 18 + TypeScript + Vite
Backend	.NET 10, Azure Functions V4 (Isolated Worker)
Database	Azure SQL Database (EF Core + EF Migrations)
Analytics	Elastic Serverless (Elastic Cloud on Azure)
AI — Image Gen	GPT Image 1.5 (Microsoft Foundry)
AI — Vision	GPT-4o Vision (Microsoft Foundry)
AI — Narrative	GPT-4o-mini (Microsoft Foundry)
AI — Agents	Microsoft Agent Framework + Foundry Agent Service
Hosting	Azure Static Web Apps + Azure Functions Flex Consumption
CDN	Azure Front Door
Auth	Azure Managed Identity (DefaultAzureCredential)
Observability	Azure Application Insights
IaC	Azure Bicep
CI/CD	GitHub Actions

Why These Choices?

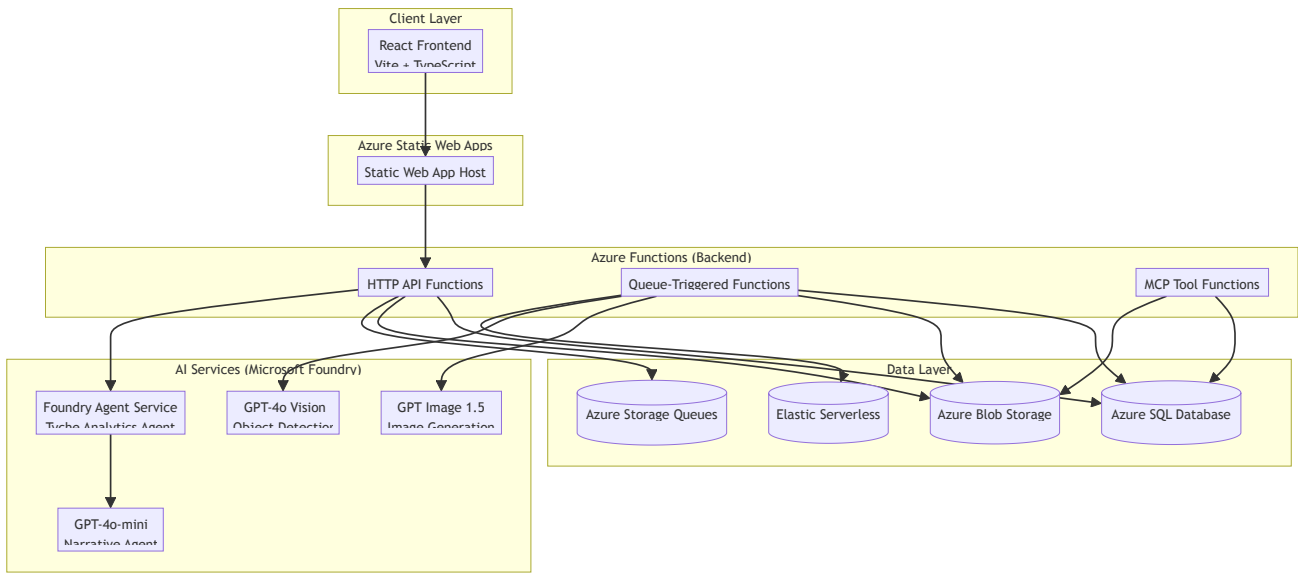
Azure Functions Flex Consumption — serverless, scales to zero, cost-efficient for a game with daily traffic spikes at midnight when new images go live.

.NET 10 — the latest LTS-bound release, with the Isolated Worker model giving full control over the Functions host pipeline, middleware, and DI.

Elastic Serverless on Azure — no-ops Elasticsearch for analytics-scale gameplay data. Synced asynchronously so it never slows down gameplay.

Azure Static Web Apps — built-in GitHub Actions integration, global CDN, and first-class support for linked backends (Functions).

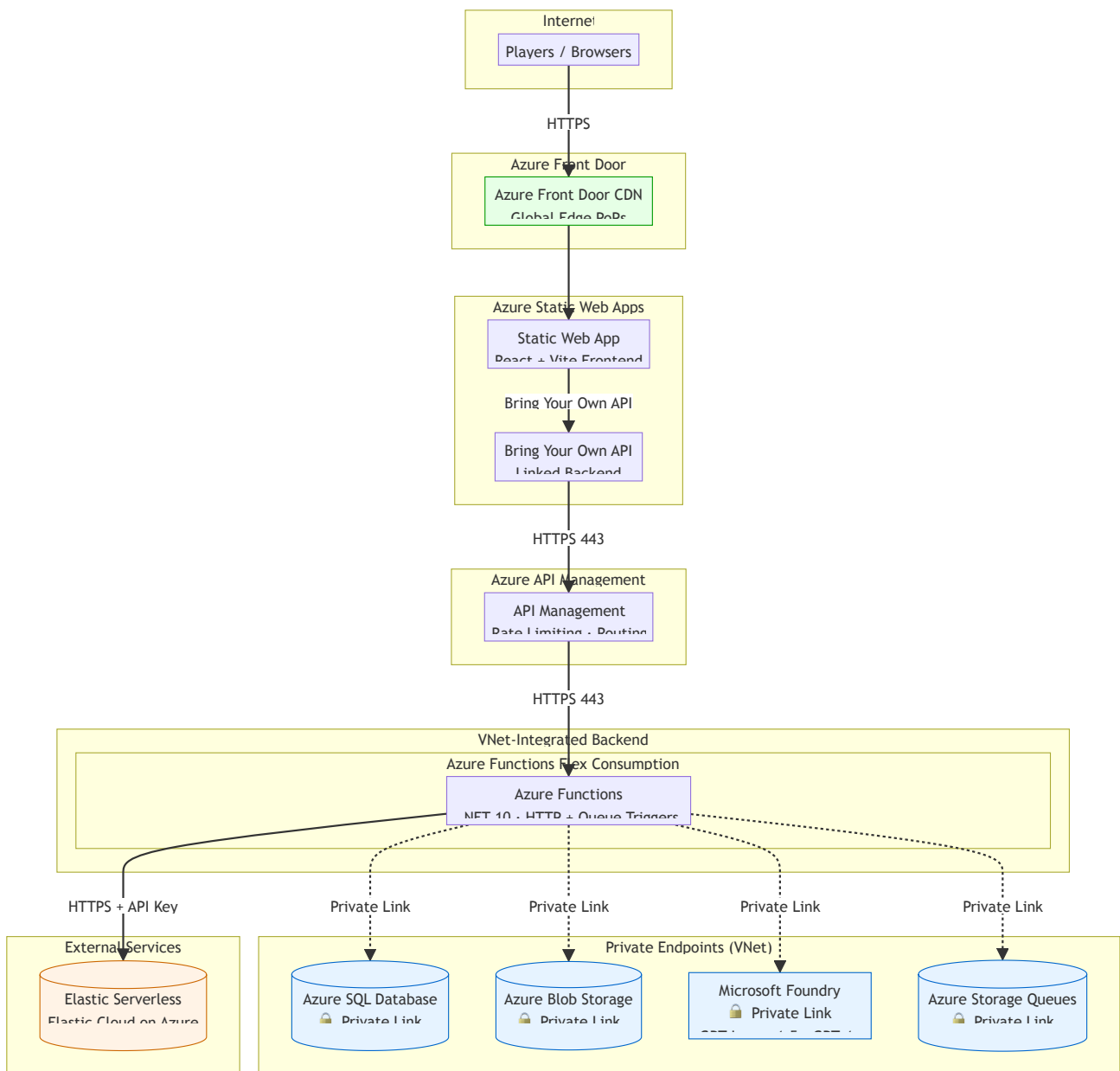
🏗️ System Architecture



Data Flow Summary

Trigger	Path
Player action	React → SWA → HTTP Function → SQL
Image generation	Admin → HTTP Function → Queue → GPT Image 1.5 → Blob → SQL
Elastic sync	End of round → Queue → Elastic Serverless
Agent query	Chat UI → HTTP Function → Foundry Agent Service → MCP Tools → SQL / Logs

🌐 Production Network Architecture



Network Security

Service	Access Model
Azure SQL	Private Link only — not internet-exposed
Azure Blob Storage	Private Link (control plane) · public read for images container
Microsoft Foundry	Private Link only
Storage Queues	Private Link only
Azure Functions	Public port 443 · VNet for outbound private connectivity
Elastic Serverless	Elastic Cloud on Azure · HTTPS + API Key

All internal Azure services communicate over the VNet. The internet never touches SQL, Foundry, or the queue storage directly.

Coming soon: Elastic Serverless will support Private Link, which will allow it to join the VNet topology above — eliminating the public HTTPS connection entirely.

Elastic Deployment Options on Azure

Elastic offers three deployment models on Azure — all production-capable, with different tradeoffs on control, billing, and network isolation:

Model	How It Works	Key Benefits
Azure Native ISV Service	Deployed directly through the Azure Portal, just like a first-party Azure service. Billing flows through Azure.	Enterprise customers can use Azure Commitment (MACC) dollars to pay for Elastic. Azure logs and metrics can be automatically forwarded to Elastic for unified observability. Managed by Microsoft + Elastic jointly.
Elastic Cloud on Azure	Elastic's managed Cloud offering, running in an Elastic-owned Azure subscription. Private Link available.	When Private Link is enabled, data is guaranteed to never leave Azure as it transits into Elastic — even though it runs in Elastic's subscription. Fully managed, no infrastructure to operate.
Self-Managed on Azure	Elasticsearch deployed by you on Azure VMs, AKS, or containers.	Maximum control over configuration, networking, and upgrades. Full VNet integration. Requires your own operational overhead.

ObjectDash uses **Elastic Cloud on Azure** (Elastic's managed offering). As Private Link support becomes generally available for Serverless, the architecture will be updated to eliminate the public HTTPS path.

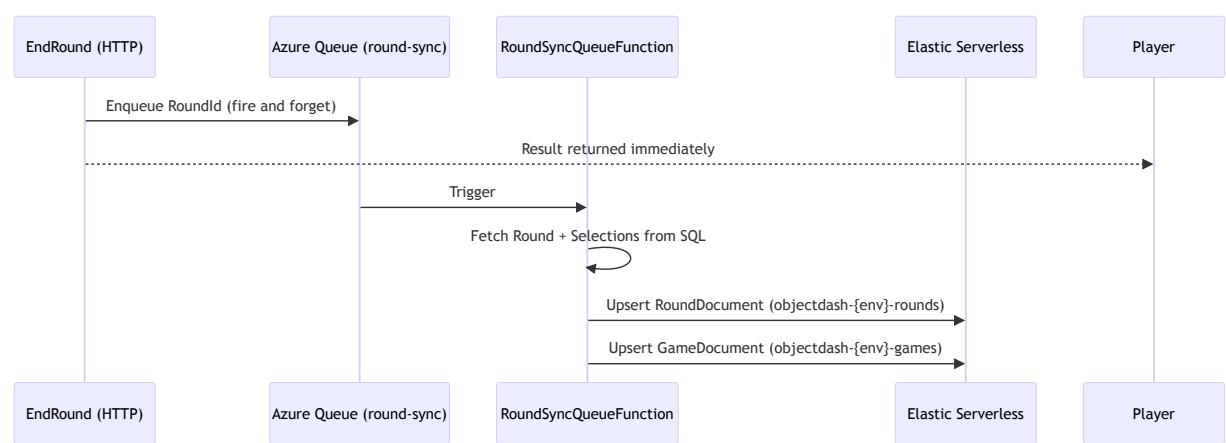
Elastic Integration

ObjectDash uses two data stores: **Azure SQL** for the authoritative game record, and **Elastic Serverless** for analytics-scale gameplay data. They serve different workloads.

Why Two Stores?

	Azure SQL	Elastic
What's stored	Players, games, rounds, leaderboards	All rounds + selections + feedback (denormalized)
Growth rate	Slow — 1 player row per player	Fast — 5+ round docs per player per day
Query pattern	Point lookups, relational joins	Aggregations, full-text, cohort analysis
Typical question	"Get player X's game today"	"Average flawless rate by image this month"

The Sync Pipeline



The player never waits for Elastic. The sync is async, retryable, and idempotent.

What Gets Indexed

RoundDocument — one per completed round

```
roundId · gameId · playerId · score · elapsedMs · isFlawless · endCause
+ image{} · feedback{} · selections[] (all nested)
```

GameDocument — one per completed game

```
gameId · playerId · dailySetId · totalScore · roundsCompleted · completionTimeMs
```

Azure Infrastructure

Everything is Infrastructure as Code — ~2,500 lines of Bicep across 11 modules.

What Gets Deployed

Category	Azure Resources
AI	Microsoft Foundry account + project · 4 model deployments (GPT Image 1.5, GPT-4o, GPT-4o-mini, DALL-E)
API Gateway	API Management service · API definition · policies · backend routing
Compute	Azure Functions (Flex Consumption) · App Service Plan
Hosting	Static Web App · custom domain binding
Database	SQL Server · SQL Database · Transparent Data Encryption (TDE)
Storage	Storage Account · blob containers · 4 queues + 4 poison queues
Networking	Virtual Network · 4 Private Endpoints · 4 Private DNS Zones · 4 VNet links
Observability	Log Analytics Workspace · Application Insights · Portal Dashboard

Storage Queues

Queue	Purpose
generate-images	Triggers image generation via GPT Image 1.5
validate-image	Triggers GPT-4o Vision validation pass
generate-titles	Triggers GPT-4o-mini title generation
round-sync	Triggers async Elastic sync after each round
*-poison	Dead-letter queues for each of the above

Identity — No Passwords

10 RBAC role assignments — all scoped to the Function App's Managed Identity

Blob Data Contributor	→ Storage Account
Queue Data Contributor	→ Storage Account
Storage Account Contributor	→ Storage Account
OpenAI User	→ AI Foundry
Cognitive Services User	→ AI Foundry
AI Developer	→ AI Foundry
AI Foundry Role	→ AI Foundry project
AI Developer (project)	→ AI Foundry project
Monitoring Reader	→ App Insights
Log Analytics Reader	→ Log Analytics Workspace

No connection strings. No API keys. No secrets rotation. Every Azure service connection uses `DefaultAzureCredential` — Managed Identity in production.

Security

Security isn't an afterthought — it's baked into the infrastructure and code from day one.

Network Security

All internal Azure services are isolated behind a VNet and communicate only via **Private Link**. The public internet cannot reach SQL, Storage, or AI Foundry directly.

Service	Network Exposure
Azure SQL Database	Private Link only — zero internet exposure
Azure Blob Storage	Private Link for writes · public read for the <code>images</code> container only
Microsoft Foundry (AI)	Private Link only
Azure Storage Queues	Private Link only
Azure Functions	Public HTTPS port 443 for incoming requests only

Identity — Zero Passwords

The Function App has a **system-assigned Managed Identity**. Every connection to an Azure service uses `DefaultAzureCredential` — which resolves to the Managed Identity in production.

```
// All Azure clients are initialized the same way - no credentials in config
var client = new AzureOpenAIClient(
    new Uri(endpoint),
    new DefaultAzureCredential()); // ← Managed Identity in production
```

What this means: - No connection strings in `appsettings.json` - No API keys for SQL, Storage, Foundry, or App Insights - No secrets to rotate, leak, or accidentally commit - Access is controlled via 10 RBAC role assignments (see Infrastructure)

The only explicit secret in the system is the Elastic Cloud API key — required because Elastic is an external service outside Azure's identity boundary.

Data Access Controls

MCP SQL Tool — the AI agent's database access is strictly controlled:

```
// Only these 5 tables are accessible - ever
private static readonly HashSet<string> AllowedTables = new(StringComparer.OrdinalIgnoreCase)
{
    "Players", "Images", "ImageTitles", "Achievements", "LeaderboardEntries"
};
// SELECT only - INSERT/UPDATE/DELETE/DROP are rejected at validation
```

SQL Transparent Data Encryption (TDE) — enabled on the database at rest.

Gameplay Anti-Tamper

Attack Vector	Defense
Clock manipulation	<code>ServerEndsAt</code> set by the server — client clock is ignored
Multiple sessions	Session lock prevents concurrent active rounds per player
Score editing	Score is always recalculated server-side from DB selections
Automated clicking	Anti-cheat analyzes selection timing patterns per round

Image Generation Pipeline

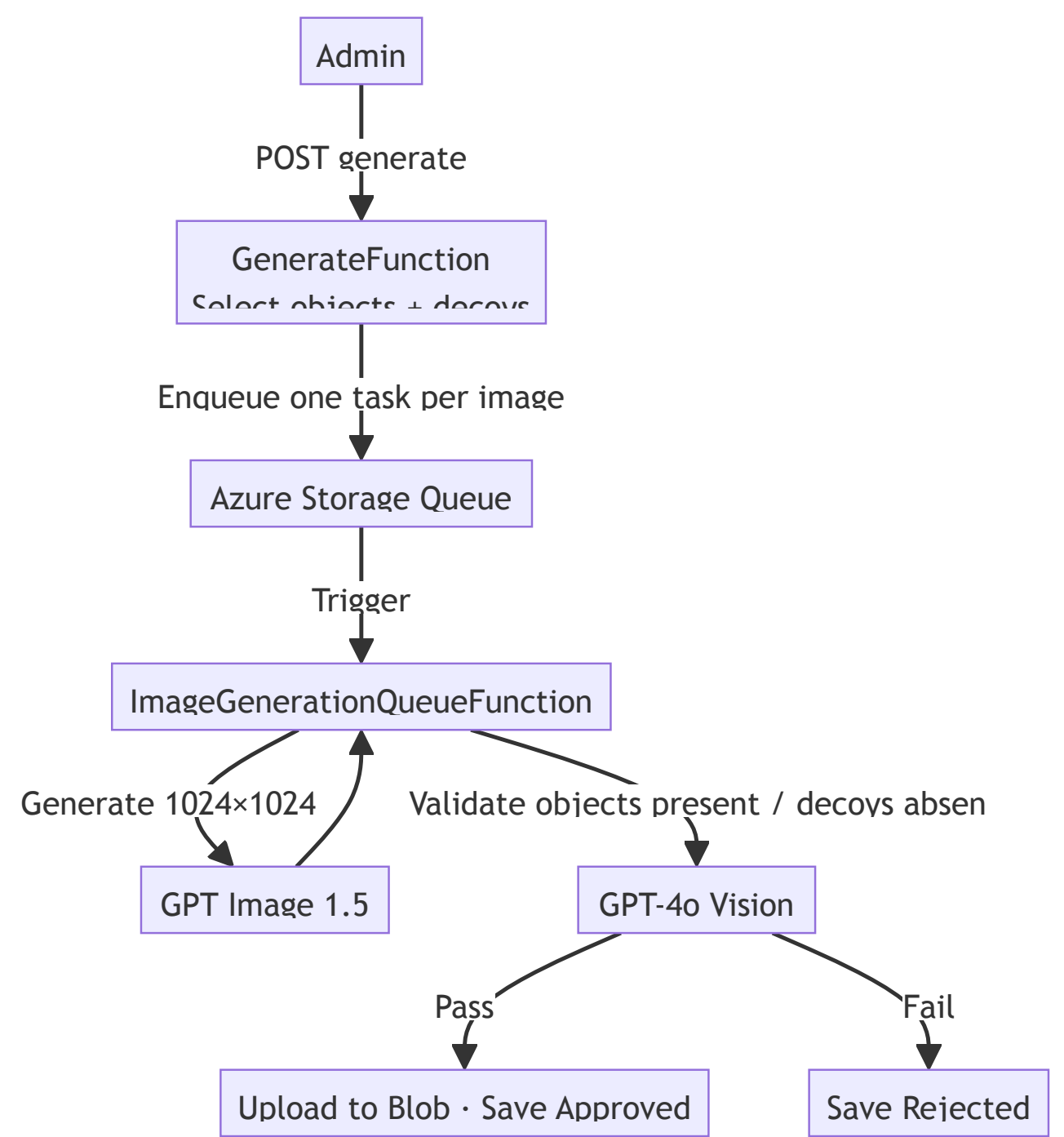
Every image in ObjectDash is **fully AI-generated** using three AI models working in sequence.

Three AI Calls Per Image

Step	Model	What It Does
Generate	GPT Image 1.5	Creates a 1024×1024 image from a structured prompt
Validate	GPT-4o Vision	Confirms the correct objects are present and decoys are absent
Title	GPT-4o-mini	Generates a creative title and alt-text for admin display

Objects and decoys are selected from a **curated catalog of hundreds of recognizable items** with difficulty ratings — so every game is the right level of challenging.

The Pipeline



Each image task is processed independently by the queue function — all images in a batch generate in parallel, automatically scaled by Azure Functions.

AI Code: GPT Image 1.5

File: backend/AiGame.Functions/Services/GptImage15Generator.cs

Image Generation

```
public class GptImage15Generator : IImageGenerator
{
    public const string ModelName = "gpt-image-1.5";

    public GptImage15Generator(IConfiguration configuration, IBlobStorageService blobStorage)
    {
        var endpointString = configuration["AzureOpenAI:GptImage15Endpoint"];
        var deploymentName = configuration["AzureOpenAI:GptImage15Deployment"];

        // Managed Identity — no secrets in config
        var azureOpenAIClient = new AzureOpenAIClient(
            new Uri(endpointString),
            new DefaultAzureCredential());

        _client = azureOpenAIClient.GetImageClient(deploymentName);
    }

    public async Task<string> GenerateImageAsync(string prompt, CancellationToken cancellationToken = default)
    {
        var options = new ImageGenerationOptions
        {
            Size = GeneratedImageSize.W1024xH1024
        };

        // GPT Image 1.5 returns raw binary — upload directly to Blob Storage
        var result = await _client.GenerateImageAsync(prompt, options, cancellationToken);
        var fileName = $"gpt-image-{Ulid.NewUlid()}.png";

        return await _blobStorage.UploadImageAsync(
            result.Value.ImageBytes.ToArray(), fileName, cancellationToken);
    }
}
```

Batch Generation — Queue-Based Parallelism

```
// BatchGenerationService.cs — enqueue one task per image
for (int i = 0; i < count; i++)
{
    var objects = await _objectSelector.SelectRandomObjectsAsync(objectsCount);
    var decoys = await _decoySelector.SelectDecoysAsync(objects, decoysCount);

    await queueClient.SendMessageAsync(BinaryData.FromString(
        JsonSerializer.Serialize(new { JobId, Objects = objects, Decoys = decoys, Index = i })));
}
```

Each image task lands in an Azure Storage Queue. The `ImageGenerationQueueFunction` picks them up independently — all images generate in parallel, automatically scaled by Azure Functions.

Server-Authoritative Gameplay

The timer is a core mechanic — and it can't be faked. All timing logic runs on the server.

Round Start: Time is Set Server-Side

```
// RoundService.cs
public async Task<Round> StartRoundAsync(Guid gameId, string imageId, int timeLimitSeconds, ...)
{
    // Prevent concurrent rounds via session lock
    var activeLock = await _sessionLockService.GetActiveLockAsync(game.PlayerId);
    if (activeLock != null)
        throw new InvalidOperationException("Player already has an active round in progress");

    var now = DateTimeOffset.UtcNow;
    var round = new Round
    {
        ServerStartedAt = now,
        ServerEndsAt     = now.AddSeconds(timeLimitSeconds), // ← server owns the clock
        TimeLimitSec     = timeLimitSeconds,
        Status            = RoundStatus.Active
    };

    // Lock prevents another device/tab from starting a second round
    await _sessionLockService.CreateLockAsync(game.PlayerId, round.RoundId, timeLimitSeconds + 60);
    return await _roundRepository.CreateAsync(round);
}
```

Round End: Score Calculated + Anti-Cheat Runs

```
public async Task<Round> EndRoundAsync(Guid roundId, EndCause endCause, ...)
{
    var selections = await _selectionRepository.GetByRoundIdAsync(roundId);

    // Score is always recalculated on the server – client score is never trusted
    var scoreBreakdown = await _scoringService.CalculateScoreAsync(...);

    // Anti-cheat: flag suspicious timing or selection patterns
    round.IsSuspicious = await _antiCheatService.AnalyzeRoundAsync(round, selections);

    // Achievements checked on every round completion
    await _achievementService.DetectAndAwardAchievementsAsync(game.PlayerId, round, game);

    return await _roundRepository.UpdateAsync(round);
}
```

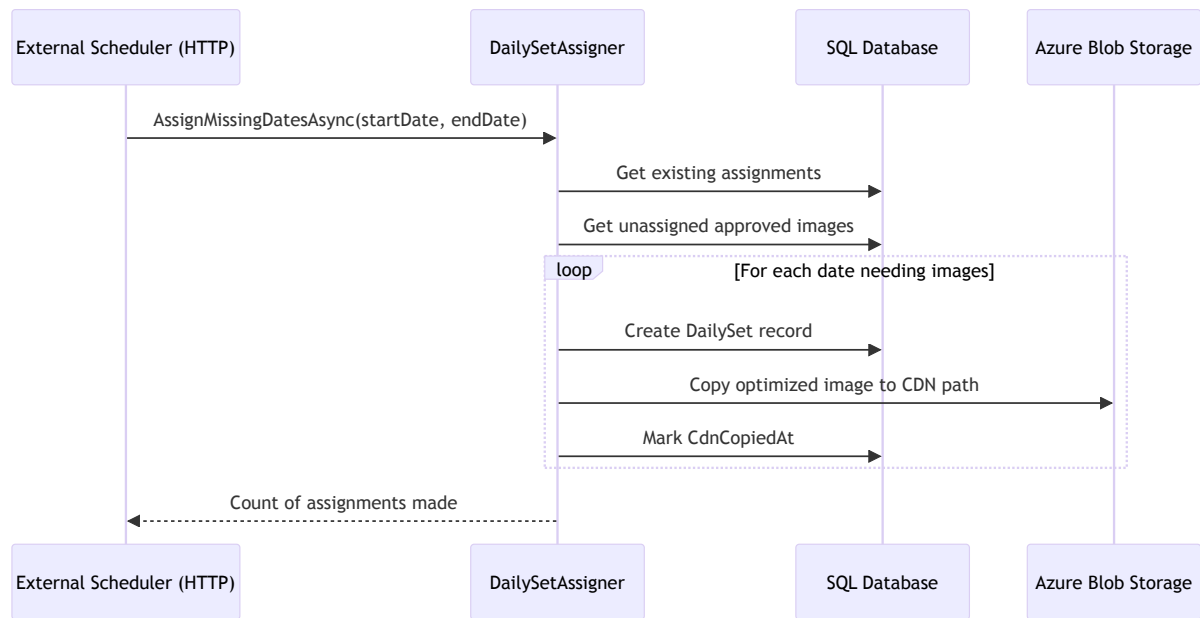
Why Server-Authoritative?

Attack Vector	How It's Blocked
Clock manipulation	<code>ServerEndsAt</code> set by server, never trusted from client
Multiple simultaneous sessions	Session lock prevents concurrent active rounds
Client-side score editing	Score recalculated from DB selections on server
Replay / automated clicking	Anti-cheat analyzes selection timing patterns

17 Daily Set System

One of the defining features of ObjectDash: **everyone plays the same images on the same day**. That's what makes the leaderboard meaningful.

How Daily Sets Work



The CDN Path Convention

```

Azure Blob Storage (public read)
├─ dailysset/
│   └─ 2026-02-22/
│       └─ standard/
│           └─ {imageId}_web.webp ← served directly to players via CDN
  
```

Azure Front Door serves these images at the edge — **no Function invocations at render time**. The image URL in the game is a direct CDN path, not a proxied API call.

Daily Set Assigner Code

```

// DailySetAssigner.cs
foreach (var (date, slotsNeeded) in datesToFill)
{
    for (int slot = 0; slot < slotsNeeded; slot++)
    {
        var image = availableImages[imageIndex++];
        var dateStr = date.ToString("yyyy-MM-dd");

        _dbContext.DailySets.Add(new DailySet
        {
            Date = DateOnly.FromDateTime(date),
            GameType = gameType,
            ImageId = image.ImageId,
        });

        // Copy to CDN path so Front Door can serve it immediately
        await _blobService.CopyBlobAsync(
            webSourceUrl,
            $"dailysset/{dateStr}/{gameType}/{image.ImageId}_web.webp");
    }
}
  
```

Meet Tyche

Tyche is ObjectDash's internal analytics agent — named after the Greek goddess of fortune and chance.

Who Uses Tyche?

Audience	What They Ask
Operators	“Show me all images rejected today and why”
Product Managers	“What’s the average flawless rate per image this week?”
Marketing	“Which objects are players getting wrong the most?”

These are the kinds of questions that would normally require a dashboard, a SQL query, or a data team. Tyche answers them in plain English.

What Tyche Can Access

Tyche pulls from **four live data sources** — and it always looks at real data, never a snapshot:

Source	What’s There
Elasticsearch	All games and rounds (scores, selections, feedback, timing)
Azure SQL	Players, images, achievements, leaderboards
Application Insights	Telemetry events, errors, request traces
Reference Data	763 objects catalog · 48,000+ world cities

How Tyche Works

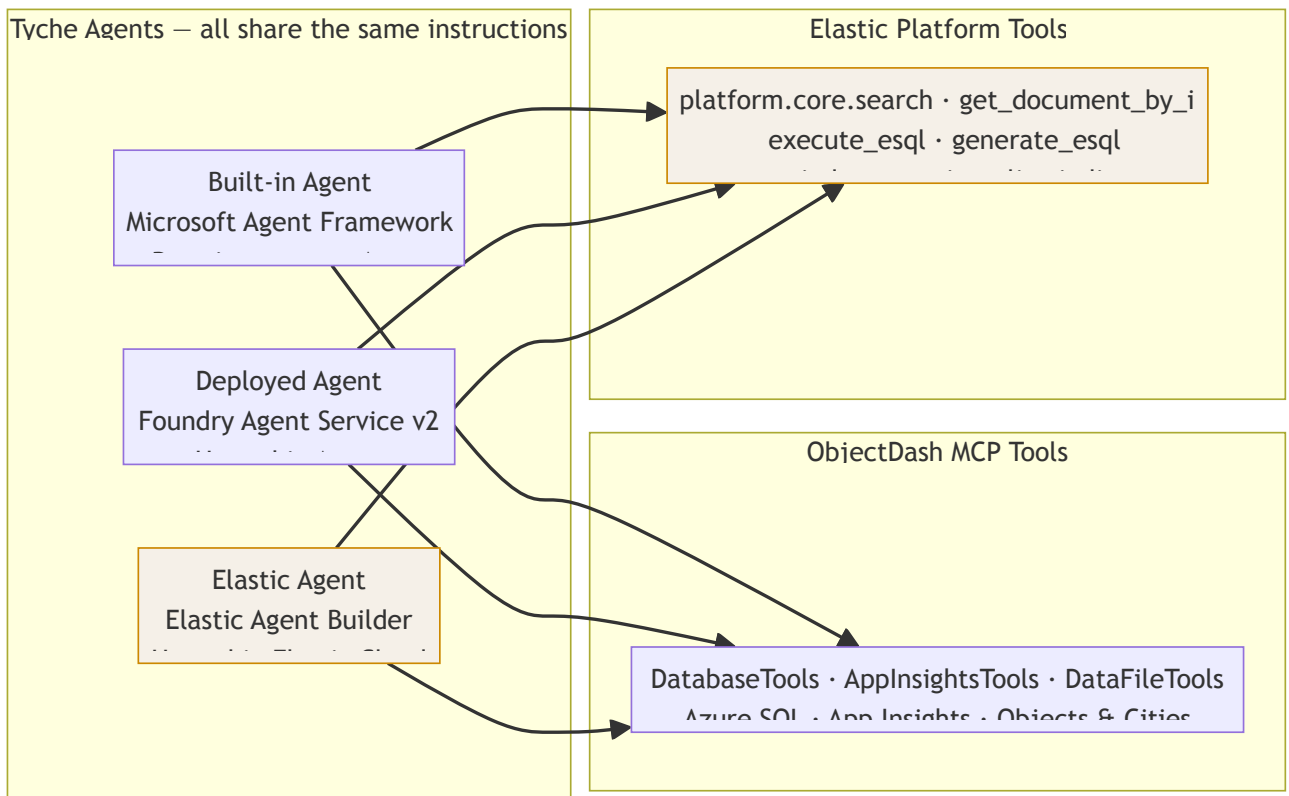
“Answer questions **only by retrieving and combining data** from your tools. Never guess or fabricate data.”
— Tyche system prompt

1. User asks a question in plain English
2. Tyche inspects the schema of the relevant source
3. Tyche executes a query (SQL, ES|QL, KQL, or search)
4. Tyche joins results across sources if needed
5. Tyche renders the answer — **with image thumbnails** when image data is returned

Three Agent Variants

Same instructions. Same tools. Three different hosting models.

ObjectDash implements all three variants — not as alternatives, but as a comparison and exploration of the options available today.



The Three Variants Explained

Built-in Agent — Microsoft Agent Framework (MAF), running **in-process** in the ObjectDash backend. - All code is visible and customizable: tool registration, retry logic, streaming behavior - Thread (conversation) history stored and managed by the Microsoft Agent Service - Best for: rapid iteration, deep testing, full observability in App Insights

Deployed Agent — Foundry Agent Service v2, hosted entirely in Azure. - Agent authored in code or the Azure portal — then deployed to the cloud - The app just calls the service; tool calls happen in the agent runtime, not in the app - Thread management and governance live in the service - Best for: production, decoupling the agent lifecycle from the app release cycle

Elastic Agent — Elastic Agent Builder, configured in Kibana. - Agent is defined and runs in Elastic Cloud — no Azure-side agent runtime - Elastic's `platform.core.*` tools are natively available — ObjectDash uses a subset of these; ObjectDash MCP tools are connected as external tools - No custom Elastic tools were authored for this solution — the built-in platform tools plus the ObjectDash MCP tools cover everything needed - All tool calls stream live to the chat UI as the agent works - Best for: analytics queries that leverage Elasticsearch's native relevance engine

MCP Tool Functions

MCP (Model Context Protocol) is an open standard for connecting AI agents to tools and data sources. ObjectDash benefits from two sets of MCP tools — ones we built, and ones that come with Elastic.

ObjectDash MCP Tools

7 tool functions hosted as **Azure Functions**, using the Azure Functions MCP Bindings. The code lives in this repository at `backend/AiGame.Functions/Functions/McpTools/`.

Group	Functions	Connects To
DatabaseTools	get_database_schema · execute_sql_query	Azure SQL (read-only)
AppInsightsTools	get_appinsights_schema · execute_kql_query	App Insights Log Analytics
DataFileTools	get_data_files_schema · query_objects · query_locations	In-memory CSV catalogs

Elastic Built-in MCP Tools

These come with Elastic — no custom code required. Any agent configured in Elastic Agent Builder gets them automatically.

Tool	What It Does
<code>platform.core.search</code>	Full-text and semantic search across Elasticsearch indexes
<code>platform.core.get_document_by_id</code>	Retrieve a single document by ID and index
<code>platform.core.execute_esql</code>	Run an ES QL query and return tabular results
<code>platform.core.generate_esql</code>	Generate an ES QL query from a natural language description
<code>platform.core.get_index_mapping</code>	Inspect the field mappings for any index
<code>platform.core.list_indices</code>	List all indexes, aliases, and data streams in the cluster

Elastic also provides the capability to **build and register custom tools** within Agent Builder — the ObjectDash MCP tools are connected to the Elastic Agent this way. For this solution, only a subset of the built-in platform tools are configured, and no custom Elastic tools were authored. The ObjectDash MCP tools (built as Azure Functions) provide all the additional data access needed.

How an ObjectDash Tool Function Looks

```
// backend/AiGame.Functions/Functions/McpTools/DatabaseTools.cs
public class DatabaseTools
{
    // Only these tables are accessible - ever
    private static readonly HashSet<string> AllowedTables = new(StringComparer.OrdinalIgnoreCase)
    {
        "Players", "Images", "ImageTitles", "Achievements", "LeaderboardEntries"
    };

    [Function("ExecuteSqlQuery")]
    public async Task<string> ExecuteSqlQuery(
        [McpToolTrigger("execute_sql_query", "Executes a validated read-only SQL query...")]
        ToolInvocationContext context,
        [McpToolProperty("query", "A read-only SQL SELECT query.", isRequired: true)]
        string query)
    {
        // Validate: SELECT only, allowed tables only
        var validationResult = ValidateQuery(query);
        if (!validationResult.IsValid)
            return JsonSerializer.Serialize(new { error = validationResult.Error });

        return await ExecuteQueryAsync(validationResult.SanitizedQuery!);
    }
}
```

Why MCP Matters

The agent doesn't need to know how to query a database — it knows how to call `execute_sql_query`. The tool handles the database. The agent handles the reasoning.

This separation means: - **Any agent** (Azure, Elastic, or future) can use the same tools - **Tools are testable** independently of the agent - **Security is enforced in the tool**, not in the prompt

Agent Platform Comparison

Both platforms are production-ready and capable. ObjectDash is a rare example of a single application that has implemented **both** — giving a direct, real-world basis for comparison.

Elastic Agent Builder vs. Microsoft Foundry Agent Service

Dimension	Elastic Agent Builder	Microsoft Foundry Agent Service
Core strength	Search-driven relevance; native Elasticsearch grounding for data-centric agents	Scalable, governed enterprise agent infrastructure with multi-agent orchestration
Development experience	Fast, low-code setup in Kibana; ready in minutes with preconfigured models	Flexible: portal for simple agents, or custom code (Semantic Kernel, MAF) for full control
Tool integration	Built-in <code>platform.core.*</code> Elasticsearch tools · MCP for external tools	Microsoft Agent Framework + MCP; deep Azure service integration
Model flexibility	Model-agnostic — AWS, Google, Microsoft, and others	Native Azure OpenAI + full Azure AI model catalog
Conversation threads	Managed by Elastic Cloud, integrated into Agent Builder	Managed by Foundry Agent Service; accessible via API and portal
Observability	Built-in monitoring in Elastic Stack; real-time tool call streaming to UI	Azure Portal + App Insights; agent-specific IDs, continuous guardrails
Deployment model	Serverless or hosted in Elastic Cloud (Enterprise plan)	Azure-managed, auto-scaling, CI/CD friendly, pay-as-you-go
Best fit	Data exploration, semantic search, analytics agents grounded in Elasticsearch	Complex multi-agent workflows, business process automation, enterprise governance

The Key Insight

These aren't competing choices — they're complementary.

Use **Elastic** when the answer lives in your search index and relevance is the core value. Use **Microsoft Foundry** when you need multi-agent orchestration, enterprise compliance, or deep Azure integration.

ObjectDash uses both — and the same **Tyche system prompt** runs on both platforms without modification.

Solution Cost

A complete AI-powered game with image generation, three agent platforms, private networking, and enterprise security — for under \$15/day.

Monthly Cost Breakdown — Azure

Service	Monthly Cost (USD)	Fixed / Variable
SQL Database	\$257.04	Fixed
Foundry Models	\$67.30	Variable
AI Image Generation + Titling	\$32.00	Fixed
Virtual Network	\$27.23	Fixed
Microsoft Defender for Cloud	\$14.99	Variable
Foundry Tools	\$10.89	Variable
Static Web App	\$9.17	Fixed
Log Analytics	\$2.60	Variable
Functions	\$1.78	Variable
Storage	\$1.02	Variable
Azure DNS	\$0.41	Fixed
Bandwidth	\$0.20	Variable

Service	Monthly Cost (USD)	Fixed / Variable
Event Grid	\$0.00	Variable

Azure costs computed from real Azure Cost Management daily billing data. SQL cost reflects a 1/3 reserved-capacity reduction.

Monthly Cost — Elastic Cloud

Component	Monthly Cost (USD)
Ingest Compute (1 VCU-hour/day)	\$4.20
Storage (0.15 GB retained)	\$0.01
Elastic Serverless Total	\$4.21

Elastic Serverless pricing: \$0.14/VCU-hour (ingest) + \$0.047/GB/month (storage). Based on ~5 MB/day ingestion with 30-day retention.

Cost Totals

Category	Monthly	Yearly
Fixed (Azure)	\$325.86	\$3,965.65
Variable (Azure)	\$98.79	\$1,200.90
Elastic Serverless	\$4.21	\$51.22
Total	\$428.86	\$5,217.77

Scaling Projections

Variable costs (Azure + Elastic) scale linearly with traffic. Fixed costs stay constant.

Traffic	Monthly Cost	Yearly Cost
1× (today)	\$428.86	\$5,217.77
2×	\$531.85	\$6,470.85
5×	\$840.87	\$10,230.59
10×	\$1,355.87	\$16,496.42
100×	\$10,626.07	\$129,284.19

At 100× traffic the system costs ~\$354/day — still under **one cent per user interaction** at scale.

Assumptions

- **Fixed** = low daily variance (coefficient of variation < 0.3)
- **Variable** = usage-dependent, scales linearly with traffic multiplier
- Azure costs sourced from Azure Cost Management daily billing export
- AI image generation at ~\$1.06/day treated as fixed (batch-scheduled, not user-driven)
- SQL Database reflects a 1/3 reduction from reserved pricing
- Elastic costs assume 5 MB/day ingestion, 30-day retention, 1 VCU-hour/day at base, scaling linearly
- Elastic pricing: ingest at \$0.14/VCU-hour, storage at \$0.047/GB/month (no search or ML VCUs included)

Disclaimer: These figures are estimates only. They were generated with AI assistance based on a small sample of actual usage data and publicly available general pricing. They do not reflect any future optimizations, reserved-instance

commitments, negotiated customer discounts, or promotional credits. Actual costs may vary. Use the Azure Pricing Calculator and Elastic Serverless Calculator for production planning.

Lessons Learned

Building ObjectDash over 73 days taught me as much about process as it did about technology.

Pick the Right Database for the Job

I started with Elasticsearch as my only database, but quickly found it challenging for the traditional RDBMS needs my application had — relational player data, game state, transactions. After talking to an architect at Elastic, I refactored to Azure SQL Database for transactional data and kept Elasticsearch for what it's great at: search and analytics. Game rounds, selections, and scores sync asynchronously into Elastic. That was a significant refactor, but GitHub Copilot and my Spec Kit workflow got me through it.

CORS Never Goes Away

CORS fixes appear at least five separate times across the commit history. In a Static Web App + Azure Functions architecture, CORS coordination between the SWA config and the Functions middleware is an ongoing concern. Every new HTTP method or endpoint surfaced a new gap. A dedicated middleware approach eventually won over per-function handling — but expect to revisit it.

AI Image Generation Requires Constant Tuning

I started the game with 10 objects and 10 decoys per image, but that was too difficult and required too much time. I also started with DALL-E 3, and it didn't do a great job of actually including the objects I wanted — my reject rate was over 25%. Moving to GPT Image 1.5 was a big improvement: my reject rate dropped to about 5%. Game balance can't be designed upfront. The shift to logarithmic speed bonuses and incremental decoy penalties came from watching real players.

Documentation Compounds — Then Needs Pruning

Rapid spec-driven development generates a lot of working documents. At one point I removed 28 stale markdown files in a single cleanup pass. Keeping documentation current is essential — GitHub Copilot relies on it heavily to get context for new features and to avoid breaking things during large refactors. Making sure Copilot writes tests and updates docs with every feature has been one of the most valuable habits in this project.

GitHub Copilot as a Team Member

Copilot authored 29% of branches in this project — not just autocomplete, but full planning, implementation, and PR-based review cycles. It resolved 70 failing unit tests in one session. It also made Entity Framework migrations painless: EF supports schema changes with migrations, and when I needed to add a player PIN column for cross-device profile claiming, I told Copilot to create the migration and seed existing profiles with random PINs. EF migration syntax isn't something I use often enough to recall easily, but with Copilot it was straightforward.

What I'd Explore Next

MCP is the future of analytics in systems with siloed data across multiple databases and backends. It's remarkable how the agent can analyze available tools and join data across systems. Tyche currently has 14 MCP tools, which may already be a lot. I've considered adding MCP servers around the GitHub repo and Azure resources too — but that would mean many more tools. At that point, I'd want separate agents managing tool requests for each service, coordinated through Agent-to-Agent (A2A) communication.

Wrap-Up

Key Takeaways

Theme	What to Remember
AI as colleague	GitHub Copilot authored 29% of branches — agentic, not just autocomplete

Theme	What to Remember
Spec Kit accelerates	6 features/week with 3-hour average cycle times, compounding as codebase matures
laC from day one	~2,500 lines of Bicep; 10 RBAC role assignments; zero passwords
Managed Identity everywhere	<code>DefaultAzureCredential</code> — no secrets to rotate or leak for Azure services
Private by default	SQL, Storage, Foundry, Queues — all behind Private Link, never internet-exposed
MCP is the glue	Same 7 tool functions work across Microsoft Foundry Agent Service and Elastic Agent Builder
Two AI platforms, one codebase	Elastic + Microsoft Foundry aren't either/or — use them for what each does best

Live Demo

Demo	What to Look For
Play the game	Open objectdash.com — is it still going?
Admin: Generate Images	Send a batch to the queue, watch results come back
Built-in Agent (Tyche)	Ask a question — watch it inspect schema, query, join, respond
Deployed Agent	Same question, same tools — agent running in Microsoft Foundry
Elastic Agent	Same question — watch tool calls stream live from Kibana

Suggested Demo Questions for Tyche

“Who are the top 5 players this week? Show me their scores and where they’re from.”

“Show me the three images with the lowest average score. Include thumbnails.”

“Which object is missed most often in round 1? Cross-reference with the catalog to show its difficulty rating.”

“How many games were completed today vs. yesterday?”

Links

Resource	URL
Play the game	objectdash.com
GitHub repo	github.com/michaelsrichter/aigame



objectdash.com
www.objectdash.com

ObjectDash

Huntington, NY

About Us

Privacy Policy

Terms of Service

Cookie Policy

Help

How to Play

Technology

Technical Presentation

Contact

Play

Leaderboard

