OFFICIAL CONFERENCE RECAP

Aspire Conf 2026

Book of News

Everything announced, demonstrated, and discussed at Aspire Conf 2026 — the premier conference for cloud-native development.

13	7h	10+
SESSIONS	CONTENT	ANNOUNCEMENTS

Executive Summary

AspireConf 2026 — the first-ever dedicated conference, held on March 23rd, 2026 — was a landmark event that signaled Aspire's transformation from an inner-loop developer tool into a full-spectrum platform for building, observing, and deploying distributed applications and AI agents. Thirteen sessions packed into a single day covered everything from beginner onboarding to enterprise-scale AI orchestration, and the recurring theme was unmistakable: Aspire 13.x has arrived at the model. The deployment story in particular landed with tremendous force — Mitch Denny's live Kubernetes-to-Raspberry-Pi demo and Norm Johanson's unscripted live AWS deployment were the two jaw-dropping moments of the day, both illustrating that 'deploy anywhere' is no longer aspirational marketing copy but working code.

The AI story was equally electric. From the opening keynote (Damian Edwards, David Fowler, Maddy Montaquila) setting the stage for agentic, distributed systems, through two dedicated AI sessions — 'Aspire to Be Agentic' (cloud architect Remi + Seth on polyglot multi-agent systems) and 'Coding Agents Need Aspire Too' (Pierce + Mike on VS Code + Copilot + MCP integration) — to Luke Parker's session on using Aspire as the shared observability surface for both human developers and AI agents in OpenCode, the conference made a compelling case: Aspire is the observability and orchestration backbone that makes AI agents trustworthy. The Windows 365 customer spotlight (a pre-recorded video from the team in China) demonstrated AI agents orchestrating fixes across 35+ enterprise repositories at scale.

Polyglot support emerged as the conference's third major arc. Chris Ays showed Go, Python, JavaScript/Vite, Spring Boot, and C# all running under a single TypeScript AppHost; Josh Goldberg gave a deep dive on TypeScript's type system in the Aspire ecosystem; and the theme of 'one AppHost, many languages' was woven into sessions on agentic systems, testing (Andres's IoT Arduino water sensor pipeline), and deployment. The community session — led by Engineering Manager Jose Perez Rodriguez and Adam Ratzman — closed the day by celebrating 600+ PRs, 170+ contributors, and 2,400+ community issues, and painting a welcoming picture for the next wave of contributors. AspireConf 2026 made one thing crystal clear: Aspire has crossed the chasm from 'interesting .NET dev tool' to 'the platform for modern distributed applications, agents, and deployments.'

Top Highlights

01 Aspire 13.2 ships V-Net support for Azure Container Apps — define VPCs, subnets, and NSGs in C# — unblocking enterprise deployments that previously required hand-crafted Bicep

02 Aspire CLI (`aspire deploy`) now powers end-to-end deployment via a pipeline steps graph, replacing the static manifest-file model — Docker Compose integration is GA, Kubernetes generates Helm charts

03 AWS Lambda support reaches GA: debug Lambda functions locally in Visual Studio with a built-in API Gateway emulator and full OTel visibility in the Aspire dashboard

04 AWS CDK deployment integration (preview): `builder.AddAWSCDKEnvironment()` maps Aspire resources to CDK constructs (Redis → ElastiCache, .NET → ECS Fargate), with a live on-stream deployment demo

05 Aspire standalone dashboard works with any OpenTelemetry-compatible application in any language — run as a single Docker image with one URL, no AppHost needed

06 GenAI telemetry in the Aspire dashboard (system prompts, tool definitions, model names, token costs, streaming chunks) makes AI agent behavior fully transparent and debuggable

07 Aspire MCP server lets AI coding agents like OpenCode query live traces and logs programmatically, grounding them in real observability data rather than guessing

08 Windows 365 uses Aspire and AI agents to orchestrate automated fixes across 35+ enterprise repositories with a centralized rollout dashboard

09 TypeScript AppHost is now production-ready, enabling polyglot apps (Go, Python, Spring Boot, Node.js, C#) all orchestrated under a single AppHost with full service discovery

10 Aspire community reaches 600+ merged PRs, 170+ contributors, and 2,400+ shaping issues; repository moves to github.com/microsoft/aspire with a new Community Toolkit for third-party integrations

Sessions

🎤 Keynote

🚀 Getting Started

🤖 AI & Agents

🛠️ DevEx & Tooling

🧪 Testing

🌐 Multi-Language

☁️ Deployment

💼 Customer Story

🤝 Community

🎤 KEYNOTE

Keynote : Come Meet the New Aspire

🎤 David Fowler, Damian Edwards, Maddie Montaquilla

⏱ 1:10:45 | ▶ Watch Session

> *Come meet the new Aspire. Discover how Aspire can transform the way you build and deploy your distributed apps and agents. With code-centric control, orchestrate and observe simple or complex systems with no rewrites, and deploy anywhere. This talk is from #AspireConf on March 23rd, 2026. 🎤 Speakers: Damian Edwards, David Fowler, and Maddy Montaquila 🎬 Watch the other sessions from #AspireConf: https://www.youtube.com/playlist?list=PLSi5JsxQ5oNvRCeQj5v6ZYUe1gwzTSUfR 🖍️ Learn about Aspire: ht*

· Summary

The inaugural Aspire Conf opened with a 70-minute barnburner keynote from the three-person Council of Aspirations - distinguished engineer David Fowler, architect Damian Edwards, and PM Maddie Montaquilla - celebrating the simultaneous launch

of Aspire 13.2, the biggest release in the project's history. The team declared Aspire is now 'an agent-ready, code-first tool to compose, debug, and deploy any distributed app' and walked through what that means: almost 1,000 pull requests, 2.6 million CLI downloads since 13.0, and a months-long 'dry January' infra sprint that repositioned the team to ship much faster going forward.

The headlining feature of 13.2 is a fully functional TypeScript app host - a year of engineering work that lets developers author the app host in either C# or TypeScript with a single Aspire install, no .NET required. Fowler demoed this live with 'A3' (All Aboard Aspire), a polyglot transit app spanning Go (BART), Python (Boston MBTA), C# (NYC MTA), and a Node.js front end, all orchestrated by a single TypeScript app host. The demo showcased secrets management via user-secrets, the new parameters API for safe clone-and-run onboarding, Redis caching wired automatically via the Aspire Redis integration, and the VS Code extension's new inline code-lens resource-state indicators alongside a graph visualizer. The new aspire.config.json unifies the old settings and launch-profiles files into one, with automatic migration.

Deployment took center stage in a second demo block. Damian introduced compute environment integrations - the Docker Compose target ships in 13.2 - where a single 'aspire deploy' command builds containers, tags them, pushes to a local registry, generates a compose YAML file, and brings up the full environment. A new pipeline-summaries feature prints every deployed resource's public URL on completion. The extensible deployment-pipeline architecture was highlighted: teams can inject custom stages (e.g., post to Slack) and toggle resources between run vs. publish mode using ordinary TypeScript if-statements and the isRunMode/isPublishMode API.

The third act covered coding-agent readiness. Aspire 13.2 ships a raft of new CLI commands (`aspire start` for background launch, `aspire describe`, `aspire logs`, `aspire otel spans/logs`, `aspire agent init` for configuring skill files) and a skill file that teaches any coding agent (Copilot, Claude Code, OpenCode) how to use the CLI. With `aspire start --isolated`, multiple sub-agents can each spin up their own app host instance with zero port conflicts, enabling parallel experiments in separate Git work trees. Fowler ran a live Copilot YOLO demo adding a brand-new Rust service (DC Metro data) to the A3 app complete with Dockerfile and app-host wiring. The keynote closed by previewing Java app host support already in a massive PR and teasing 13 more community sessions for the rest of the day.

· Key Takeaways

→ Aspire 13.2 ships TypeScript app host - no .NET installation required for TypeScript/JS teams

→ New aspire.config.json consolidates settings + launch profiles into one file with auto-migration

→ Docker Compose deployment target ships in 13.2 via single `aspire deploy` command

→ Pipeline summaries print all public endpoint URLs on successful deployment completion

→ `aspire start` launches app host in background so agents can continue working concurrently

→ New `aspire agent init` and skill files teach Copilot/Claude Code how to use Aspire CLI

→ `aspire start --isolated` enables port-conflict-free parallel agent work trees

→ OTEL GenAI view in dashboard updated to show prompts, tool definitions, and token usage per span

→ 2.6 million CLI downloads since 13.0; nearly 1,000 PRs merged for 13.2 alone

· Announcements & Features

Aspire 13.2 release — Largest release to date with ~1,000 PRs; shipped simultaneously with the Aspire Conf conference

TypeScript app host — App host can now be authored in TypeScript in addition to C#; single `aspire install` handles everything, no .NET required

aspire.config.json — New unified config file that replaces aspire-settings.json and launchSettings.json; auto-migrated on upgrade

Docker Compose deployment target — First-party compute environment integration; `aspire deploy` builds containers, generates Docker Compose

YAML, and deploys locally or in CI

Pipeline summaries — After `aspire deploy` completes, prints a summary of all deployed resources and their public endpoint URLs

`aspire start` command — Launches app host in the background so coding agents can continue working without being blocked on process output

`aspire describe` / `aspire logs` / `aspire otel` commands — New CLI commands give agents text-based access to all dashboard data including resources, logs, and OTel spans

`aspire agent init` and skill files — Sets up agent configuration and installs a skill file teaching agents how to use the Aspire CLI and workflow

`aspire start --isolated` mode — Each sub-agent gets its own app host instance with no port conflicts, enabling safe parallel work tree experiments

aspire.dev What's New AI summary — 'Open in AI' button on the What's New page sends the full release notes to the user's preferred AI for a quick summary

New full-stack templates — Python, TypeScript, and C# full-stack starter templates added to `aspire new`

Java app host preview — Java app host support was in a massive PR at snap time for 13.2; community demo expected later in the day

· Demos & Live Code

A3 (All Aboard Aspire) polyglot transit app — TypeScript app host orchestrating Go (BART), Python (MBTA), C# (NYC MTA), Node.js front end,

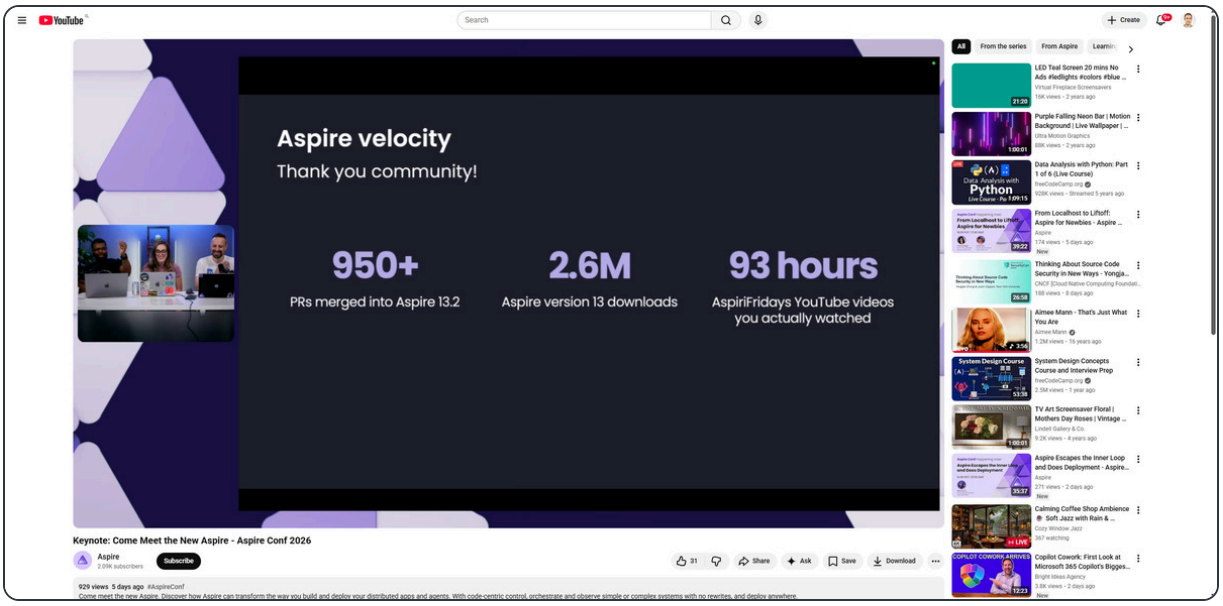
Redis cache, and OpenAI advisor - demonstrated clone-and-run, parameters UI, OTel GenAI traces in dashboard

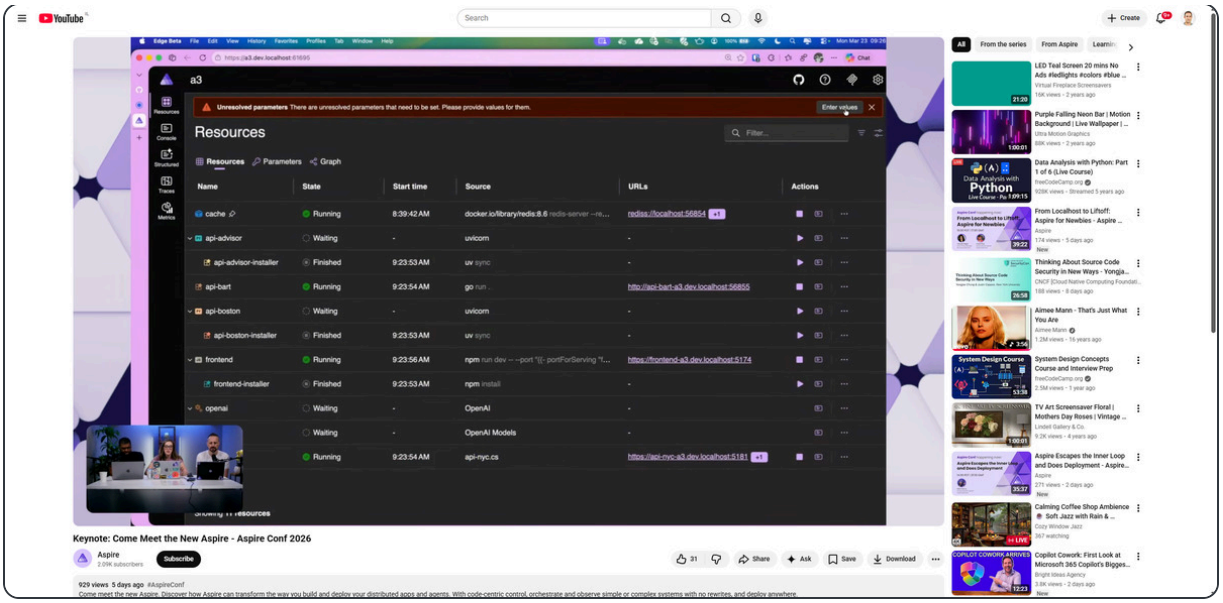
aspire deploy to Docker Compose — Single `aspire deploy` command built containers, generated Docker Compose YAML, and launched a fully-working deployed environment from the same TypeScript app host

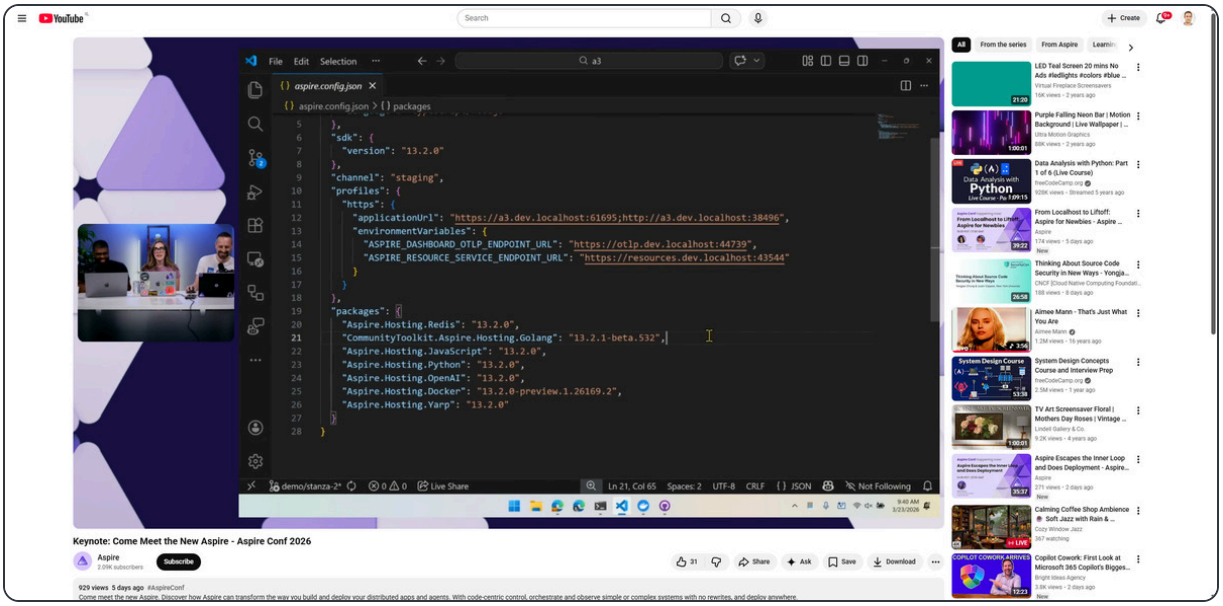
GitHub Copilot YOLO mode adding Rust service — Copilot used `aspire start` (background), `aspire describe`, skill file, and `aspire doc search` to add a new Rust DC Metro API service end-to-end including Dockerfile

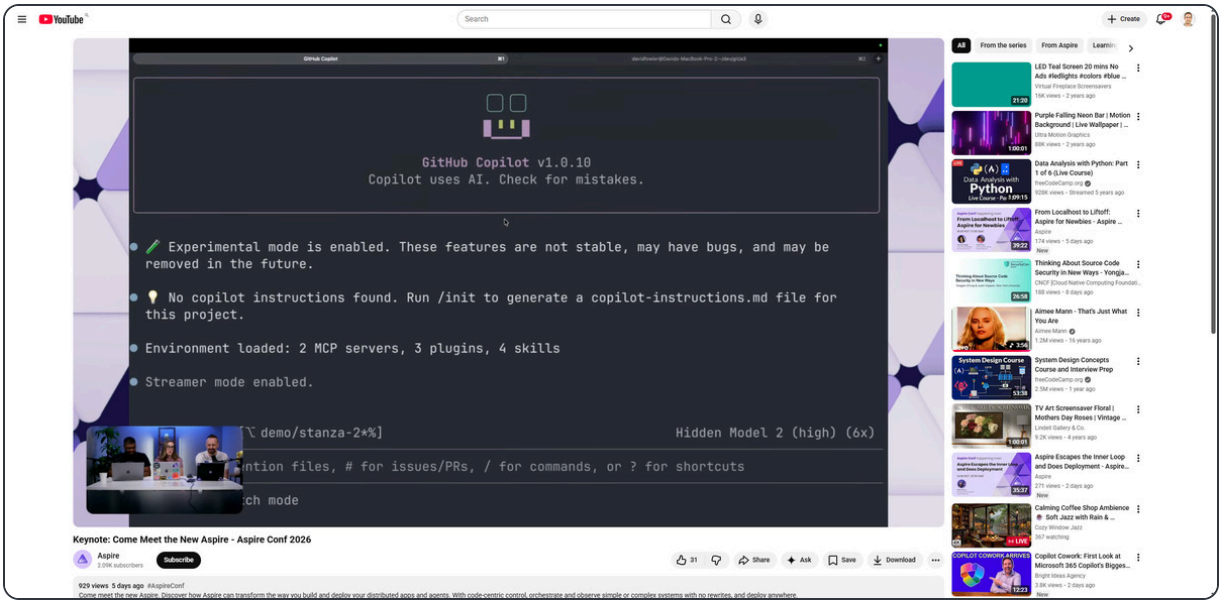
VS Code extension inline code-lens — Resource state shown in-line inside app host TypeScript file while Aspire is running; live status indicators next to each resource definition

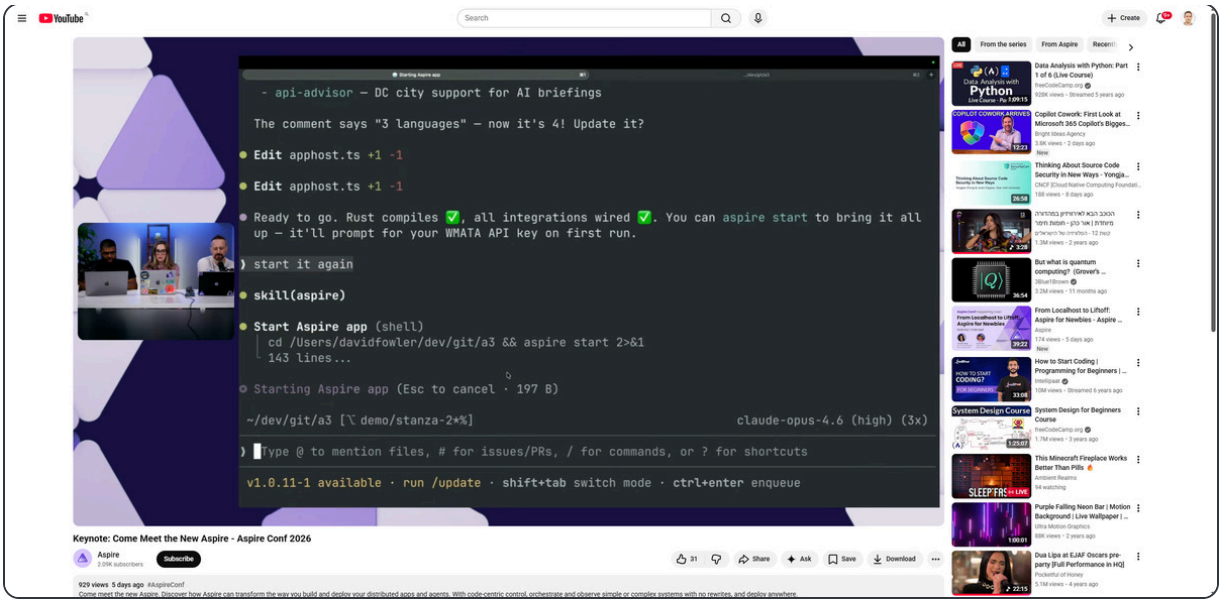
aspire.dev What's New AI summarizer — Clicked 'Open in AI' on aspire.dev/whats-new to send the 1500-line release notes directly to an AI agent for instant summarization




a3
Resources
Keynote: Come Meet the New Aspire - Aspire Conf 2026


aspire.config.json
Keynote: Come Meet the New Aspire - Aspire Conf 2026


GitHub Copilot v1.0.10
Copilot uses AI. Check for mistakes.
Experimental mode is enabled. These features are not stable, may have bugs, and may be removed in the future.
No copilot instructions found. Run /init to generate a copilot-instructions.md file for this project.
Environment loaded: 2 MCP servers, 3 plugins, 4 skills
Streamer mode enabled.
Keynote: Come Meet the New Aspire - Aspire Conf 2026



· Code Samples

TypeScript app host wiring four polyglot services with Redis and OpenAI (A3 demo)

```
import { createBuilder } from '@aspire/js';
const builder = await createBuilder(args);
const cache = builder.addRedis('cache');
const bart  = builder.addExecutable('bart',  { command: 'go run .' });
const mbta  = builder.addExecutable('mbta',  { command: 'python main.py' })
const nyc   = builder.addProject('nyc');
const openai = builder.addExternalService('openai', { url: process.env.OPEN
await builder.build().run();
```

Toggling resources between run and publish mode in TypeScript app host

```
if (!builder.executionContext.isRunMode) {
  const yarp = builder.addContainer('gateway', { image: 'yarp' })
    .withStaticFiles(frontend)
    .withExternalEndpoints()
    .withPort(80);
}
```

Adding Docker Compose compute environment integration via aspire.config.json

```
// aspire.config.json
{
  "packages": {
    "Aspire.Hosting.Docker": "13.2.0"
  }
}
// apphost.ts
builder.addDockerComposeEnvironment('compose');
```

New agent-friendly CLI commands in Aspire 13.2

```
aspire start             # launch app host in background
aspire start --isolated  # port-conflict-free isolated instance
aspire describe          # list all resources and their state
aspire logs --resource myapi    # stream resource logs
aspire otel spans        # export OTel spans as text
aspire agent init        # configure skill files for coding agents
```

🚀 GETTING STARTED

From Localhost to Liftoff : Aspire for Newbies

🎤 David Pine, Claudia

⏱ 39:21 | ▶ Watch Session

> *What happens when you learn Aspire alongside someone seeing it for the first time? David walks Claudia through a live, beginner-friendly tour — from localhost to the cloud. Expect real questions, clear explanations, and a fresh look at how Aspire handles orchestration, observability, and service composition today. Whether you're new or catching up, you'll leave with a simple mental model and a clear way to explain Aspire to your team. This talk is from #AspireConf on March 23rd, 2026. 🎤 Speak*

· Summary

David Pine, a senior software engineer on the Aspire team, and Claudia, a product manager on the developer tools team at Coportei, delivered a ground-up walkthrough of Aspire 13.2 tailored for first-time users. The session doubled as an insider origin

story: Claudia had never used Aspire before, and Pine's explicit job was to win her over live on stream. The pair grounded the session in a real-world testimonial from customer Russ: 'I had someone start on a Monday morning and they were able to contribute code by lunch.' They benchmarked that against the old approach of juggling docker run, virtual-env activation, and per-service README instructions for every new developer.

Pine demoed `aspire new` from scratch in a terminal, selecting the Express + React starter, and walked through the resulting TypeScript app host file. He set a breakpoint directly inside app-host.ts and hit F5 in VS Code, demonstrating that the Aspire debugger steps through the app host into actual service code - a first for any orchestration layer. Key ergonomics highlighted: automatic npm install for Node resources, configuration injection (environment variables auto-wired from resource references so services never need hardcoded connection strings), and the Aspire dashboard's graph view. He then ported the weather starter into a full to-do CRUD app with a PostgreSQL database using builder.addPostgres with WithDataVolume and WithPersistentLifetime, showed the WithCommand extension to add a 'Clear Database' button on the dashboard, and demonstrated how the same DI service can be shared between dev-time commands and E2E tests.

A second, more ambitious demo featured a rock-paper-scissors battle app - officially stolen from Adam (VS Code extension dev) - with an ASP.NET Core minimal API officiator, a React+Vite front end, and five bots written in Python, C#, Go, Rust, and Node, all wired in a single app host. The entire app deploys to Azure Container Apps via `aspire deploy`, and a live link at aka.ms/aspire-rps was shared for viewers to join and battle the bots in real time. The session closed with announcements of official C# and TypeScript API reference documentation now published on aspire.dev, and the project surpassing 150 integrations across the official set, vendor packages (e.g., AWS), and the community toolkit.

· Key Takeaways

→ `aspire new` scaffolds a TypeScript or C# app host with agent skill files and Playwright CLI in one step

→ VS Code debugger can set breakpoints directly in app-host.ts and step into service code

→ Configuration injection auto-wires connection strings as typed environment variables to dependent services

→ builder.addPostgres() with WithDataVolume and WithPersistentLifetime for persistent local DB

→ WithCommand adds interactive dashboard buttons usable in both developer workflow and E2E tests

→ Rock-paper-scissors demo: single app host managing 5 polyglot bots deployed to Azure Container Apps

→ Official C# and TypeScript API reference docs now published on aspire.dev

→ 150+ total integrations milestone across official, vendor, and community toolkit packages

· Announcements & Features

C# and TypeScript API reference docs on aspire.dev — Official reference documentation for both app-host languages now published at aspire.dev; still early, feedback requested

150+ integrations milestone — Aspire now offers over 150 integrations across official team packages, vendor integrations like AWS, and the community toolkit

New GitHub org: Microsoft Aspire — Project moved from .NET GitHub organization to its own dedicated Microsoft Aspire organization

· Demos & Live Code

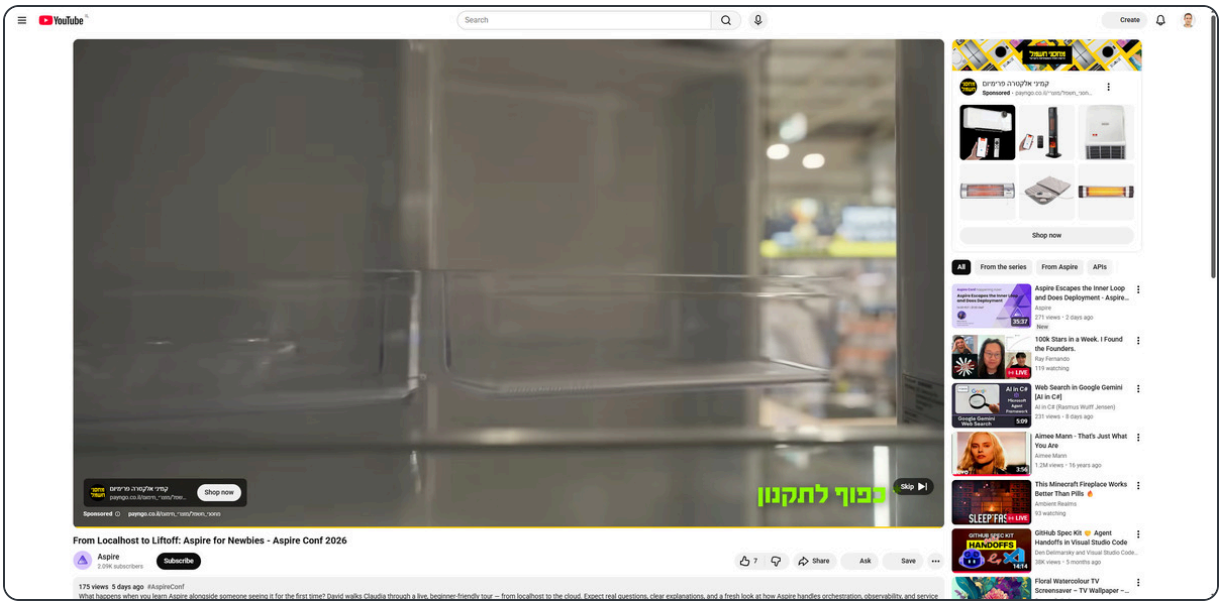
`aspire new` Express + React from scratch — Created a new project, showed TypeScript app host, set breakpoint in app-host.ts, debugged live through weather API endpoint with VS Code debugger

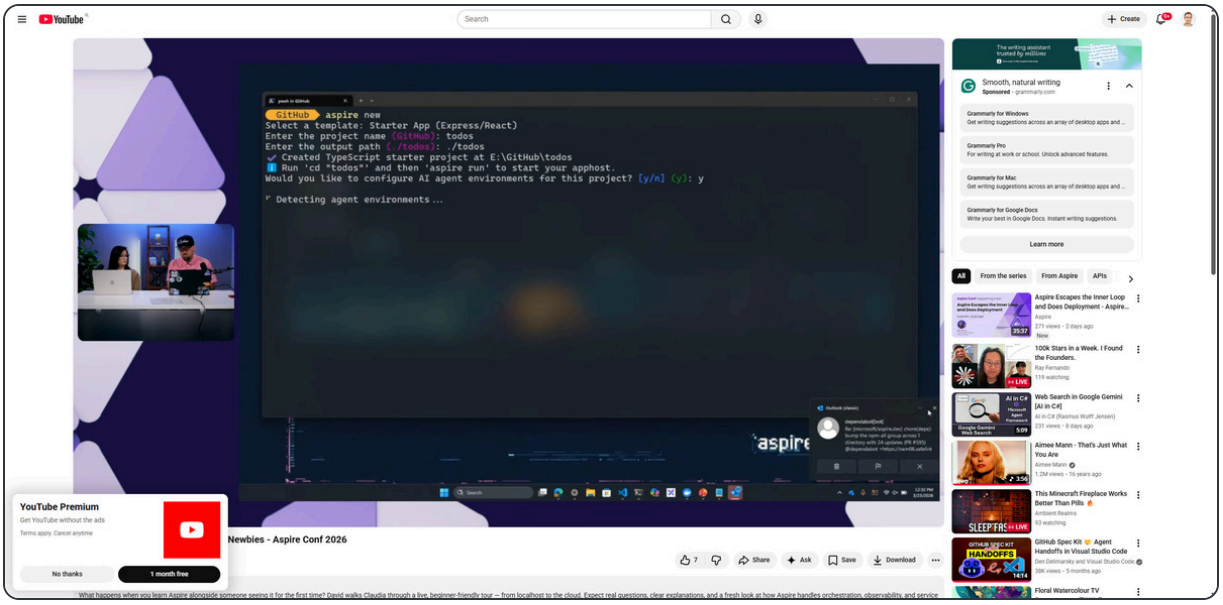
To-do app with PostgreSQL — Extended the starter with addPostgres, WithDataVolume, WithPersistentLifetime, a CRUD Express API, and a 'Clear

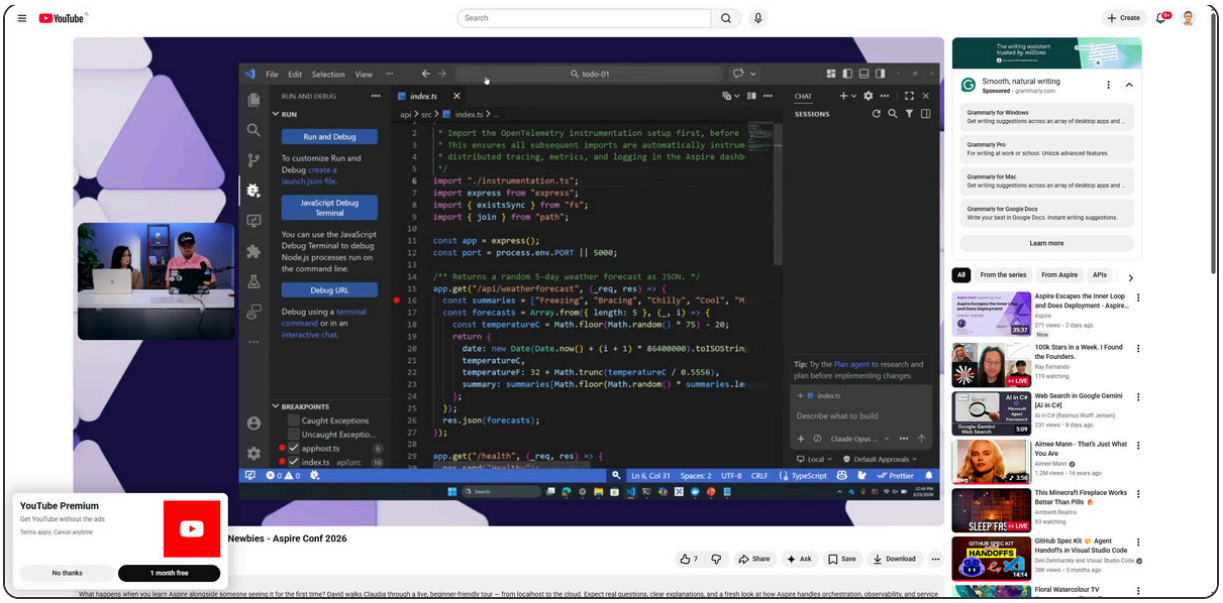
Database' WithCommand button visible on the resource card

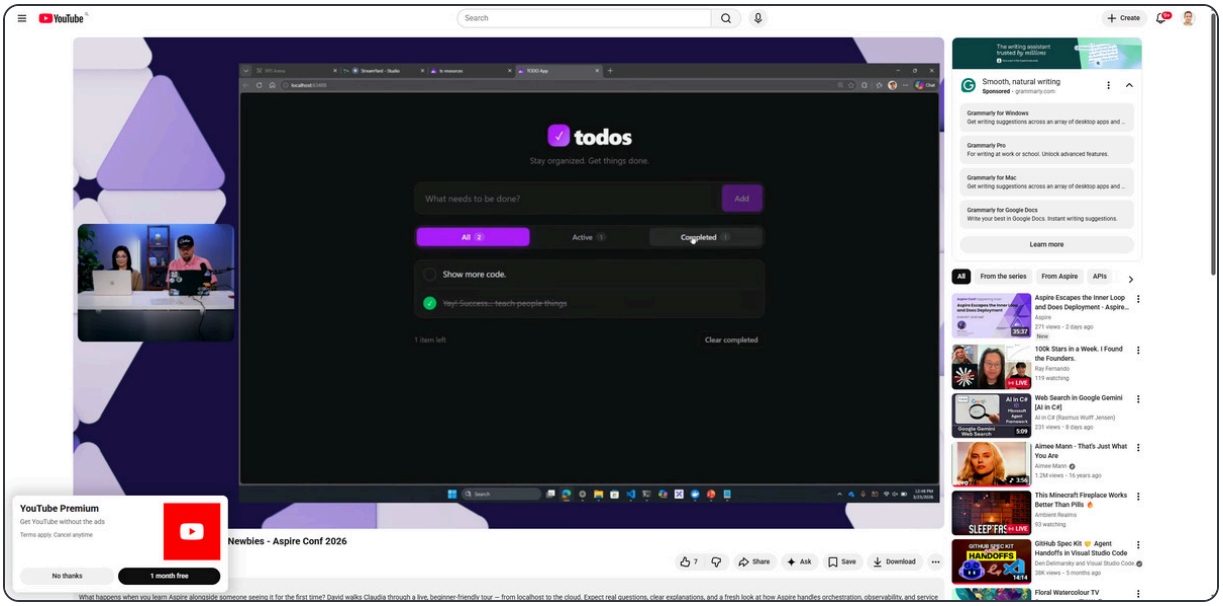
Rock-paper-scissors battle app — Five polyglot bots (Python, C#, Go, Rust, Node) fighting via an ASP.NET Core officiator, deployed live to Azure Container Apps at aka.ms/aspire-rps for viewer participation

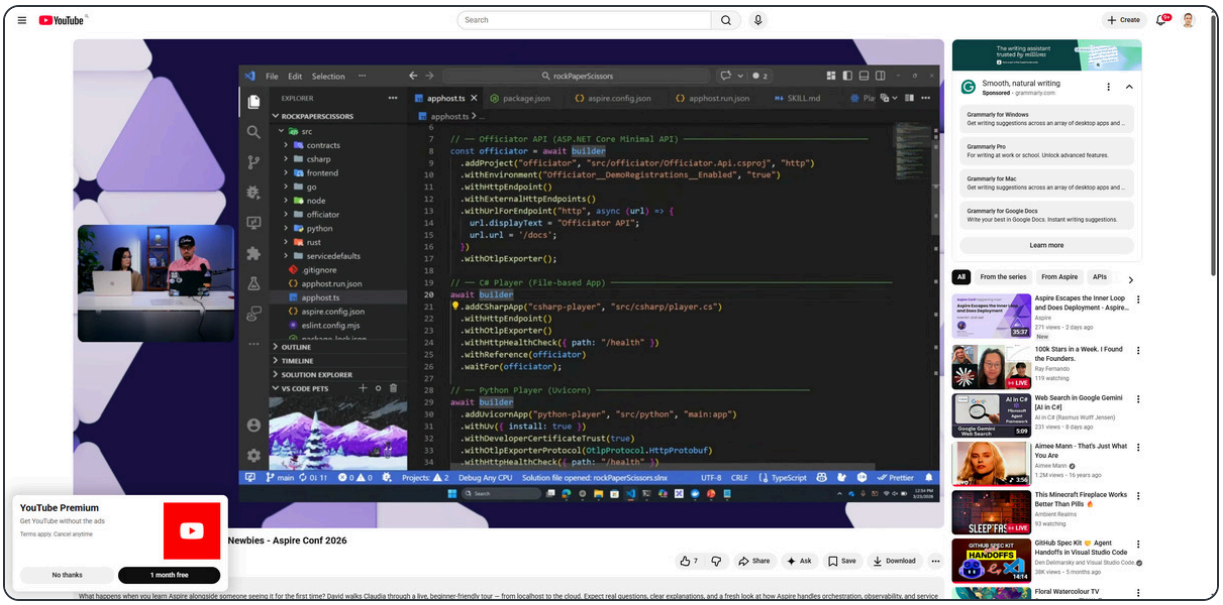
GitHub Copilot adding Postgres to a Node app — Agent called `aspire doc search postgres node javascript`, found the integration docs, ran `aspire add`, and wired up the database - live in session











- Code Samples

TypeScript app host with PostgreSQL database and frontend dependency chain

```
const pg  = builder.addPostgres('db').withDataVolume().withPersistentLifeti
const db  = pg.addDatabase('todos');
const api = builder.addNode('api', { source: './api' }).withReference(db).w
const fe  = builder.addVite('frontend', { source: './frontend' }).waitFor(a
```

Custom dashboard command with confirmation dialog (Clear Database button)

```
resource.withCommand('clear-db', 'Clear Database', async (ctx) => {
  const db = ctx.serviceProvider.getRequiredService();
  await db.executeAsync('TRUNCATE todos');
  return CommandResults.Success();
}, opts => { opts.confirmationMessage = 'This will delete all todos. Contin
```

aspire new flow with agent-init prompt showing skill file setup

```
aspire new
# → choose template (Express + React, Blazer, etc.)
# → project name
# → prompted: 'Configure AI agent environments for this project? [Y/n]'
# → installs Aspire skill file + Playwright CLI skill
cd my-app && code .
```

AI & AGENTS

Aspire to Be Agentic : Designing Distributed Agentic Systems Without the Chaos

🎤 Tomaso, Seth

⏱ 33:07 | ▶ Watch Session

> *Agentic systems are inherently distributed: multiple agents, multiple services, multiple languages. And right from the start, things get messy fast. Silent failures. Unclear execution paths. Too many "What tool did the agent call?" moments. Aspire's polyglot support and built-in observability help you bring order to the chaos. You'll see how much easier your life becomes when building agentic applications with Aspire. This talk is from #AspireConf on March 23rd, 2026. 🎤 Speakers: Seth Juarez*

· Summary

Cloud solution architect Tomaso and Microsoft's Seth - who introduces himself simply as someone who 'makes stuff at Microsoft' - co-presented a no-slides, all-

demos session on using Aspire as the backbone for production-grade distributed agentic systems. Their shared thesis: an agent is just 'an HTTP client glorified with a system prompt and some tools', and once you accept that, the real engineering challenge shifts entirely to observability across multiple networked agent processes - which is exactly Aspire's sweet spot.

Tomaso opened with Microsoft Agent Framework (MAF), showing how the built-in DevUI (a chat frontend using the OpenAI Responses/Conversation protocol) works fine when all agents are in-process but breaks down the moment you distribute them across services. He demoed a not-yet-merged DevUI Aspire integration for MAF that bridges this gap, then presented his flagship demo: an Alpine AI Ski Resort Dashboard with five agents (weather, safety, ski coach, lift traffic, advisor) running as separate ASP.NET Core and Python projects. The advisor agent acts as the single entry point and invokes the other agents as HTTP tools on demand - only calling the ones it needs based on the user's question. Enabling OTEL_INSTRUMENTATION_GENAI_CAPTURE_SENSITIVE_DATA in both .NET and Python clients surfaces rich GenAI telemetry in the Aspire dashboard: parallel tool calls visible in the waterfall view, full prompt/response content, Cosmos DB emulator data-explorer spans, and token usage per model call. A key announced feature is the new Microsoft Foundry Aspire integration on NuGet, co-built with the Foundry team, replacing hand-maintained Bicep files.

Seth then took over with a fundamentally different demo: Zavacore, a fully voice-driven AI shirt-design assistant built with Python backend, React frontend, and Microsoft Foundry Microsoft Foundry policy agents for content moderation. He showed the app capturing a camera image, sending it to OpenAI image generation, and returning a custom shirt design - all driven by voice. The punchline was practical: Seth demonstrated how, when a content moderation block hit the design pipeline in a pre-conference run in Europe, he used GitHub Copilot CLI pointing at the production Aspire traces to diagnose the failure in minutes. The session reinforced that Aspire's observability story is equally valuable for Python and React stacks as it is for .NET.

· Key Takeaways

→ Agents as HTTP clients: distributed agentic systems need observability infrastructure, not just frameworks

→ New Microsoft Foundry Aspire integration on NuGet replaces Bicep-based Foundry project setup

→ Enable OTEL_INSTRUMENTATION_GENAI_CAPTURE_SENSITIVE_DATA to see full prompt/response in Aspire dashboard

→ Parallel multi-agent tool calls are clearly visible in the OTel waterfall trace view

→ DevUI (MAF chat frontend) Aspire integration in progress for distributed agent scenarios

→ Aspire works equally well for Python-only and React-only stacks, not just .NET services

→ Copilot CLI + Aspire production traces = remote debugging workflow for AI application failures

→ Seth moved from MCP to Aspire CLI as primary agent interface after upgrading to 13.2

· Announcements & Features

Microsoft Foundry **Aspire integration** — New NuGet package co-built with the Foundry team; define Foundry projects, deploy hosted and prompt agents from the app host, eliminating manual Bicep files

runAsHostedAgent() API — New app host API to deploy your own agent code to Microsoft Foundry as a hosted compute resource directly from the app host definition

MAF DevUI Aspire integration (preview/PR) — Bridges Microsoft Agent Framework's DevUI chat frontend with distributed Aspire-hosted agents; not yet merged at time of conference

· Demos & Live Code

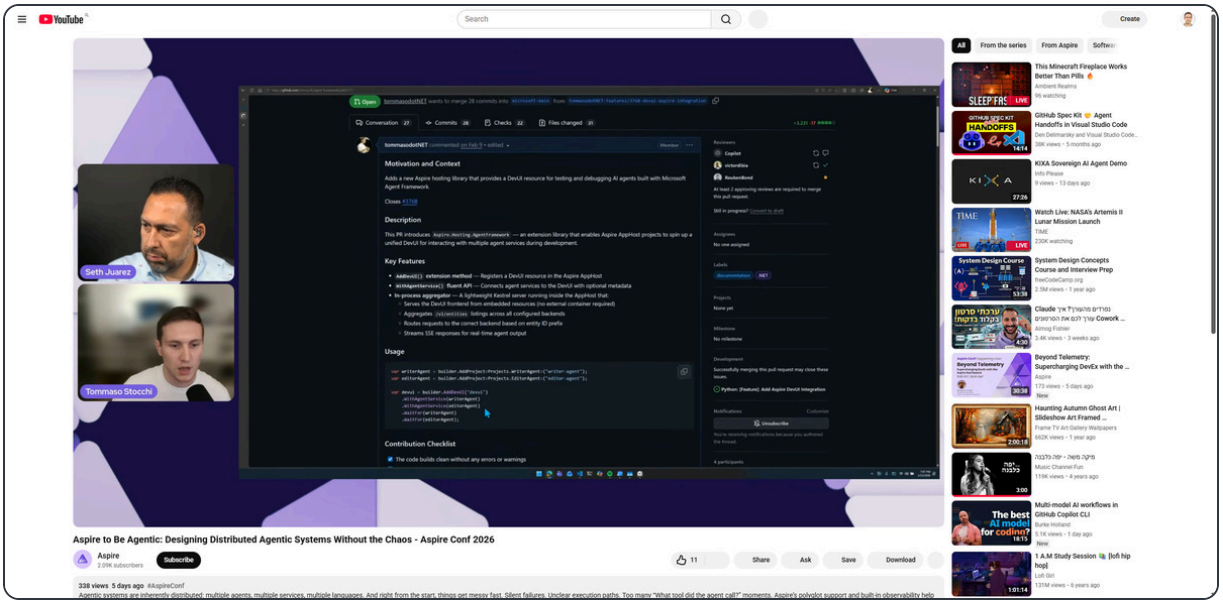
Microsoft Agent Framework DevUI (in-process) — Three-agent pipeline (writer, editor, publisher) using MAF's format-story tool, shown in DevUI with

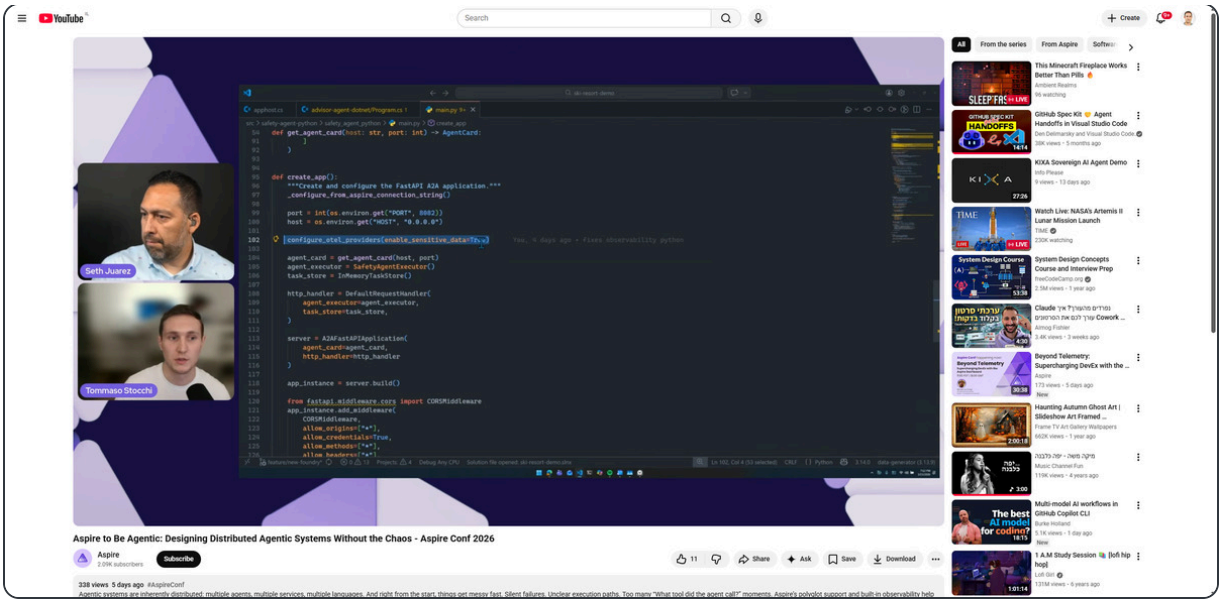
real-time event streaming and OTel traces

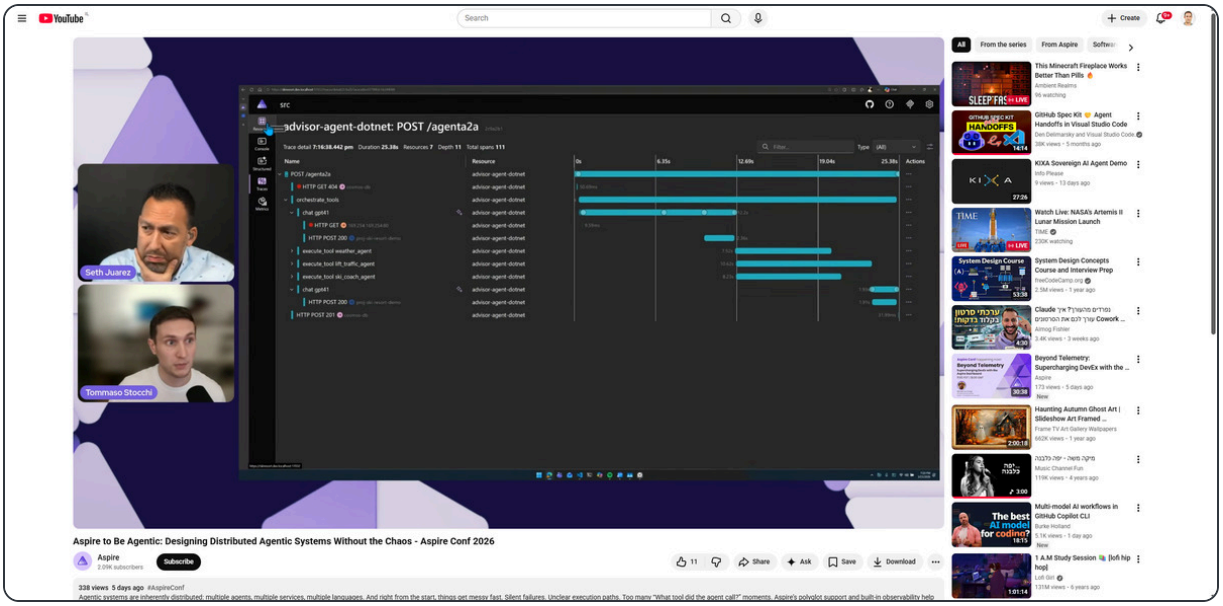
Alpine AI Ski Resort Dashboard — Five distributed agents (weather, safety, ski coach, lift traffic, advisor) as separate .NET/Python APIs; Aspire dashboard showed parallel tool calls, full GenAI prompts, Cosmos DB spans in a single trace waterfall

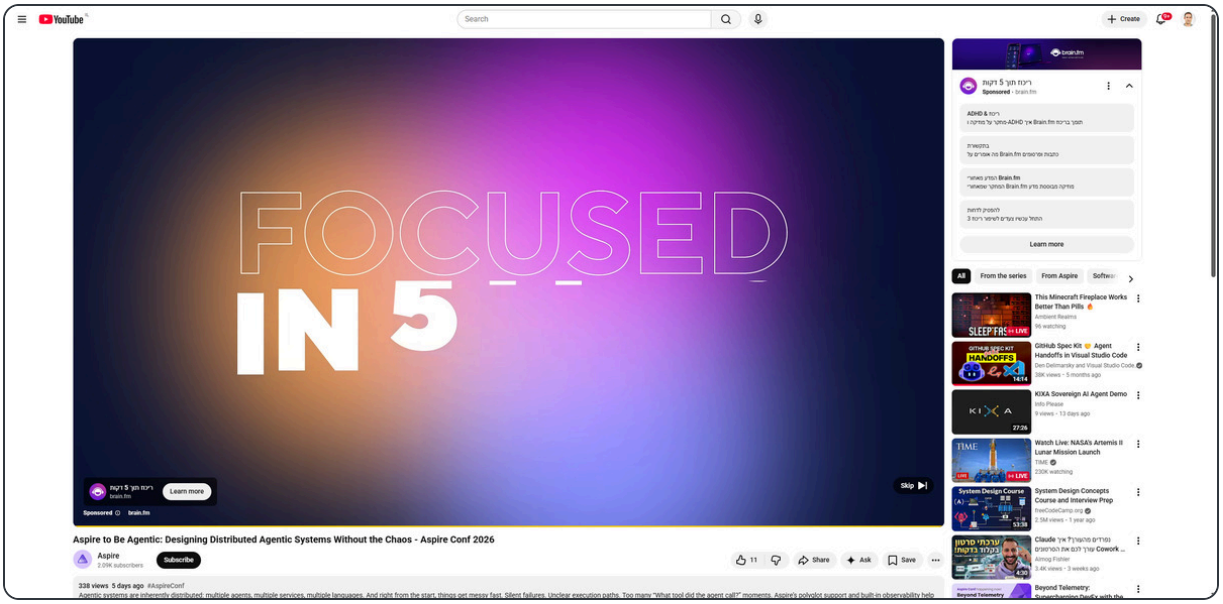
Zavacore voice AI shirt designer — Voice agent (Python + React, no .NET) that captures camera images and generates AI shirt designs via OpenAI + Microsoft Foundry Microsoft Foundry agents; demonstrated content moderation trace visibility

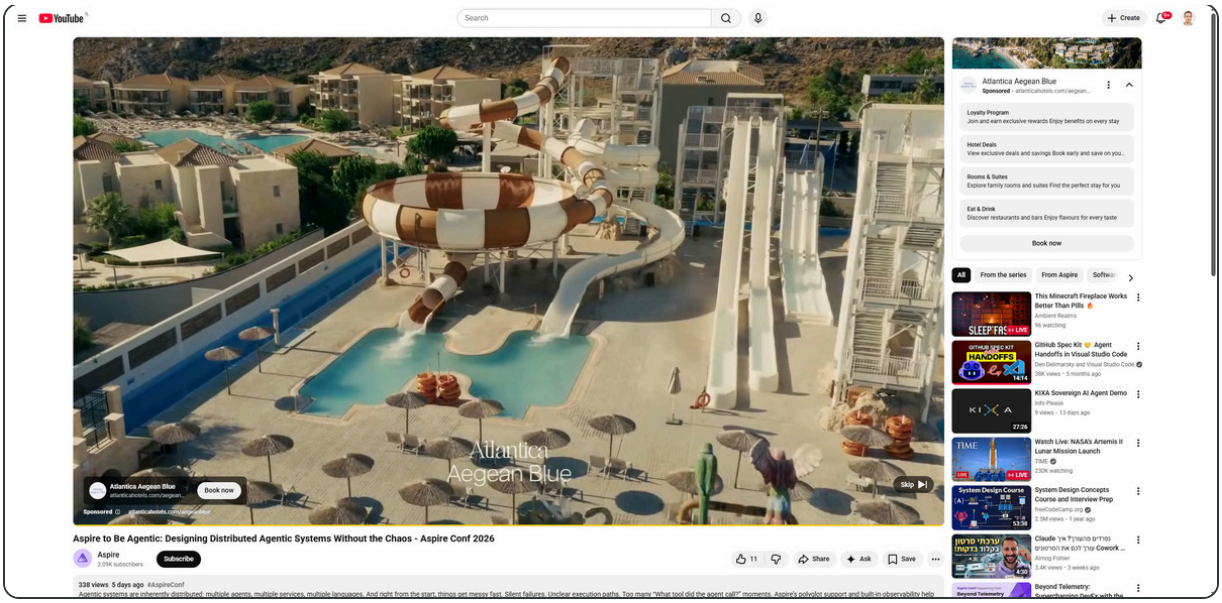
Remote production debug via Copilot CLI + Aspire traces — Seth asked Copilot CLI to inspect production Aspire logs and traces, which identified and explained a content-moderation block on the image-generation API call




Seth Juarez
Tommaso Stocchi
Aspire to Be Agentic: Designing Distributed Agentic Systems Without the Chaos - Aspire Conf 2026
Aspire
2.09K subscribers
Subscribe
338 views 5 days ago #AspireConf


Seth Juarez
Tommaso Stocchi
advisor-agent-dotnet: POST /agenta2a
Aspire to Be Agentic: Designing Distributed Agentic Systems Without the Chaos - Aspire Conf 2026
Aspire
2.09K subscribers
Subscribe
338 views 5 days ago #AspireConf


FOCUSED
IN 5
Learn more
Skip
Aspire to Be Agentic: Designing Distributed Agentic Systems Without the Chaos - Aspire Conf 2026
Aspire
2.09K subscribers
Subscribe
338 views 5 days ago #AspireConf



· Code Samples

Enabling sensitive GenAI OTel data capture in .NET and Python

```
# .NET (appsettings.json or environment)
"OTEL_INSTRUMENTATION_GENAI_CAPTURE_SENSITIVE_DATA": "true"

# Python
import os
os.environ['OTEL_INSTRUMENTATION_GENAI_CAPTURE_SENSITIVE_DATA'] = 'true'

# Results in full prompt/response content visible in Aspire dashboard GenAI
```

Microsoft Foundry Aspire integration in app host

```
var foundry  = builder.AddAzureAIFoundry("foundry");
var project  = foundry.AddProject("my-project");
var model    = project.AddDeployment("gpt-4o", "gpt-4o", "2024-11-20");
// Deploy your own agent code to Foundry as hosted compute:
var agent = builder.AddProject("advisor")
  .RunAsHostedAgent(project)
  .WithReference(model);
```

Distributed multi-agent system with advisor invoking sub-agents as HTTP tools

```
// Each agent is a separate ASP.NET Core / Python project registered in app
var weather  = builder.AddProject("weather");
var safety   = builder.AddProject("safety");
var coach    = builder.AddProject("coach");
var advisor  = builder.AddProject("advisor")
  .WithReference(weather)
  .WithReference(safety)
  .WithReference(coach);
// Advisor invokes sub-agents as HTTP tools - only when needed by the LLM
```

🛠️ DEVEX & TOOLING

Beyond Telemetry : Supercharging DevEx with the Aspire Dashboard

🎤 Michael Cummings

⏱️ 30:38 | ▶ Watch Session

> *Aspire is often positioned as a way to model and run distributed applications, but its biggest impact is how it improves the everyday developer experience. In this talk, we'll focus on small, practical features in Aspire that reduce friction during local development and testing. Rather than adding complexity, these capabilities help you move faster with fewer context switches. Get some concrete ideas for using Aspire to make your app dev more approachable, safer to operate, and easier to work on*

· Summary

Michael Cummings, a principal software engineer on Microsoft's NuGet and VS Marketplace team, delivered one of the most practical sessions of the day: a deep dive into customizing the Aspire dashboard to transform it from a generic resource-viewer into a project-specific developer experience. Cummings's team converted

their entire development environment to Aspire around version 6/7 and ships their service as a container, giving him years of real-world dashboard experience. His thesis: most teams only use about 20% of the dashboard's customization APIs, and the other 80% - mostly a handful of extension methods - can dramatically reduce onboarding friction and make local development faster for everyone on the team.

The talk was structured as a tour of 'paper cuts' - small API touches with outsized ergonomic payoffs. First: DistributedApplicationOptions.DashboardApplicationName lets you set a custom title with emojis, replacing the auto-derived project name. Second: the WithUrlForEndpoint extension method gives each endpoint a DisplayText label, controls its DisplayLocation (SummaryAndDetails vs. DetailsOnly), sets DisplayOrder for left-to-right URL ordering, and can inject fully custom URLs (e.g., a /dungeon page link) that aren't native service endpoints. Third: WithIconName maps Fluent UI icon names from the fluentui.dev catalog to resource cards, replacing generic container/project/executable icons with semantically meaningful ones. All three were demonstrated live with a Redis-backed dungeon-events web app.

The second half focused on WithCommand - Cummings's favorite feature - which adds interactive buttons to a resource card. He started with basic commands plus confirmation dialogs, then leveled up to CommandOptions for setting description, icon name, and IsHighlighted (which pins the command to the top action bar, giving you three prime slots next to Stop). The crown jewel was the InteractionService (preview, diagnostic ID ASPIRE0001), which exposes PromptInputsAsync to render a rich modal dialog composable from code - dropdown choices, text fields, number inputs - accessible from both the dashboard UI and the Aspire CLI during publish/deploy operations. Cummings closed by showing how the exact same DungeonEventsCacheService injected via DI into WithCommand can also be injected into a DistributedApplicationTestingBuilder E2E test, ensuring developer seeding and test seeding share one implementation - demonstrated with a Verify.Tests HTTP snapshot assertion.

· Key Takeaways

→ DashboardApplicationName sets a custom emoji-friendly title in the Aspire dashboard header

→ WithUrlForEndpoint controls DisplayText, DisplayOrder, and DisplayLocation (summary vs. details-only)

→ WithUrls() bulk-hides noisy URLs to DetailsOnly, dramatically decluttering the resource list view

→ Custom non-endpoint URLs (e.g., /dungeon) can be injected via WithUrlForEndpoint overload

→ WithIconName maps Fluent UI catalog names to resource icons for semantic visual identity

→ WithCommand adds interactive buttons; CommandOptions.IsHighlighted pins to the top action bar

→ InteractionService.PromptInputsAsync renders composable modal dialogs with dropdowns and text/number fields

→ InteractionService is available in both dashboard UI and Aspire CLI publish/deploy flows

→ App-host DI services (e.g., cache seeder) can be shared between WithCommand and E2E tests

· Announcements & Features

InteractionService modal inputs — Preview feature (diagnostic ASPIRE0001) exposing PromptInputsAsync, PromptMessageBoxAsync, and PromptNotificationAsync for rich dashboard dialogs; available in dashboard UI and CLI

Fluent UI icon support in dashboard — WithIconName uses the fluentui.dev icon catalog names to set semantic icons on any resource card

File picker for InteractionService (upcoming PR) — Michael has an open PR adding a file-picker input type to the InteractionService dialog, motivated by VS Marketplace upload workflows

· Demos & Live Code

Custom dashboard title and URL display — Set DashboardApplicationName with emoji, used WithUrlForEndpoint to name endpoints 'Home' and 'Dungeon',

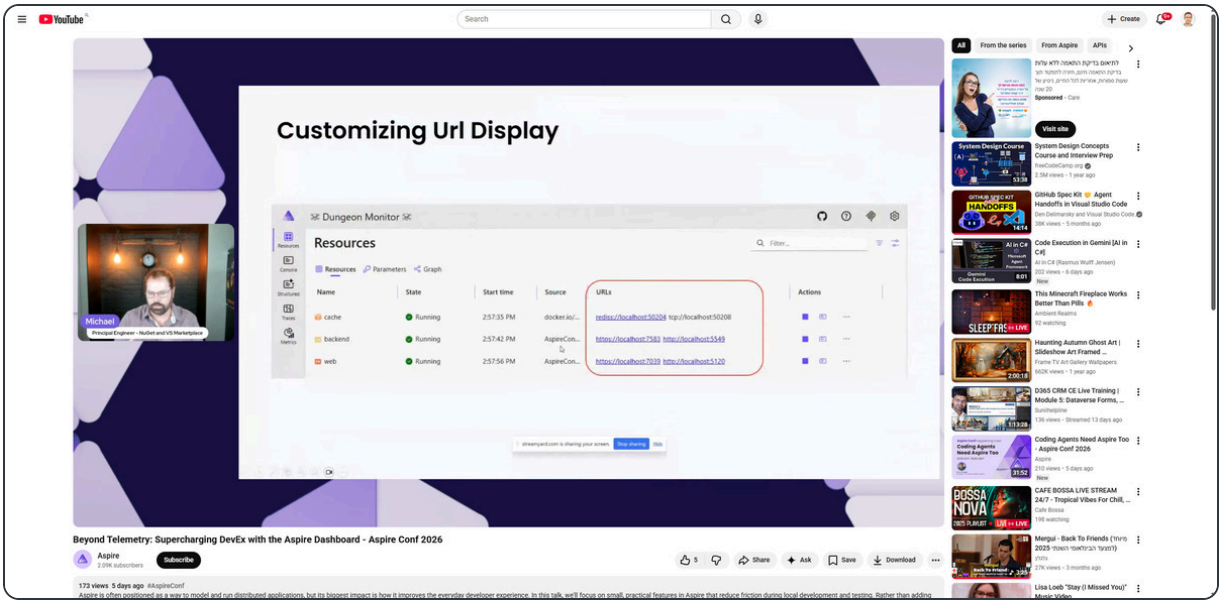
hid HTTP/noise URLs to details-only, and set display order

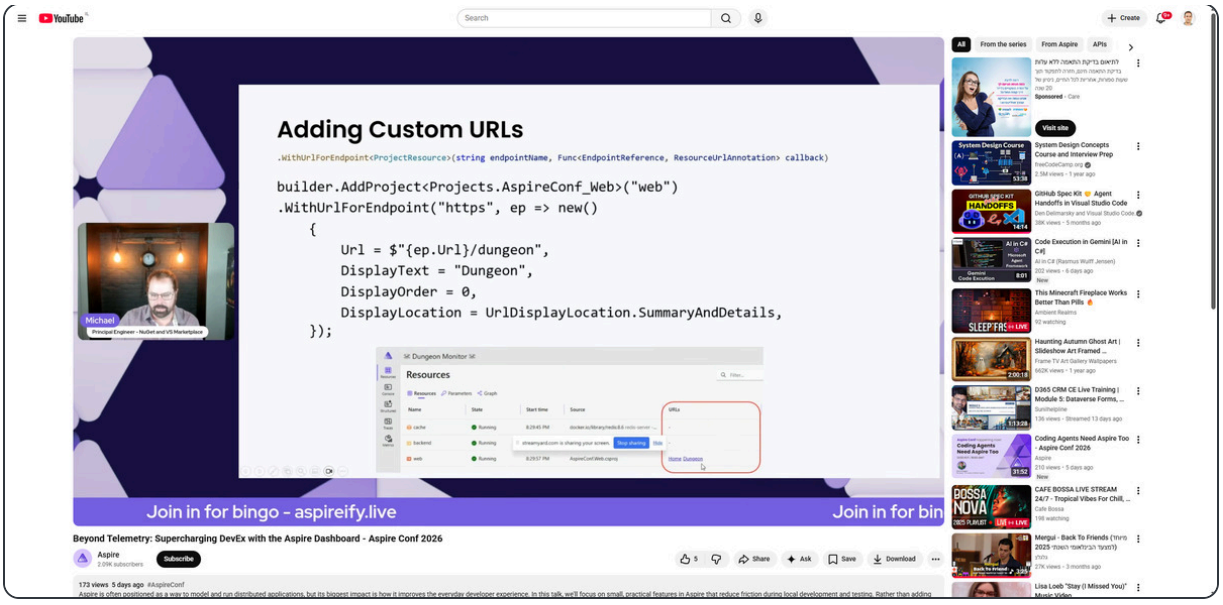
Fluent UI resource icons — Applied WithIconName('ShieldKeyhole', IconVariant.Filled) to resources; icons visible in both resource list and graph view

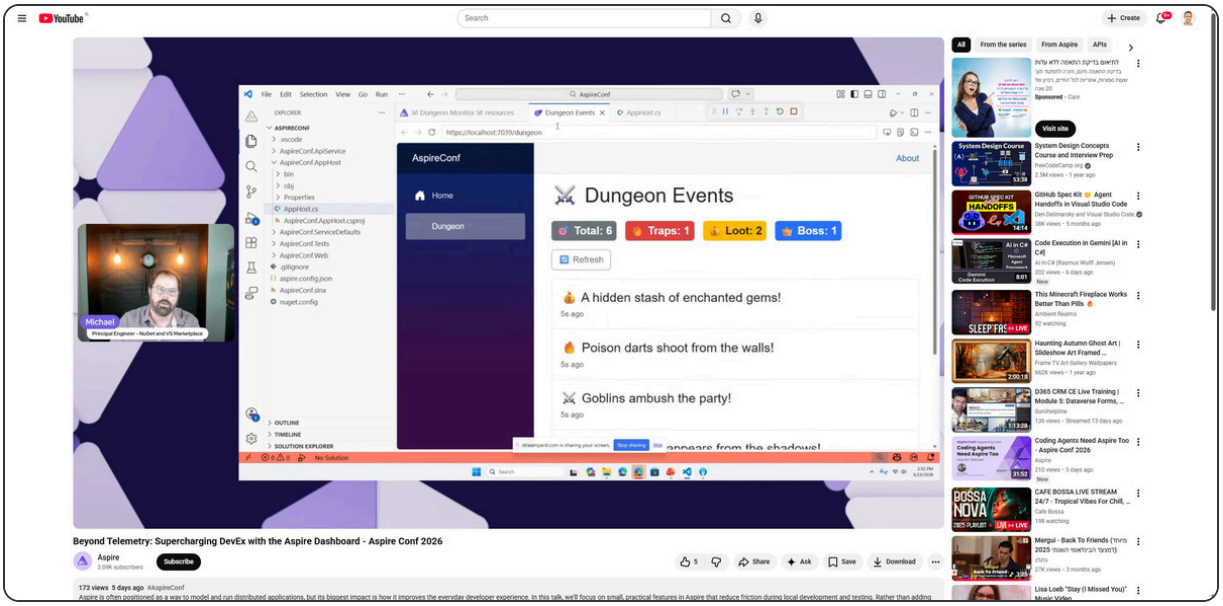
WithCommand seed-data button — Added a 'Seed Dungeon Events' command that pushes data to Redis; added CommandOptions with IsHighlighted to pin it to the action bar with tool-tip description

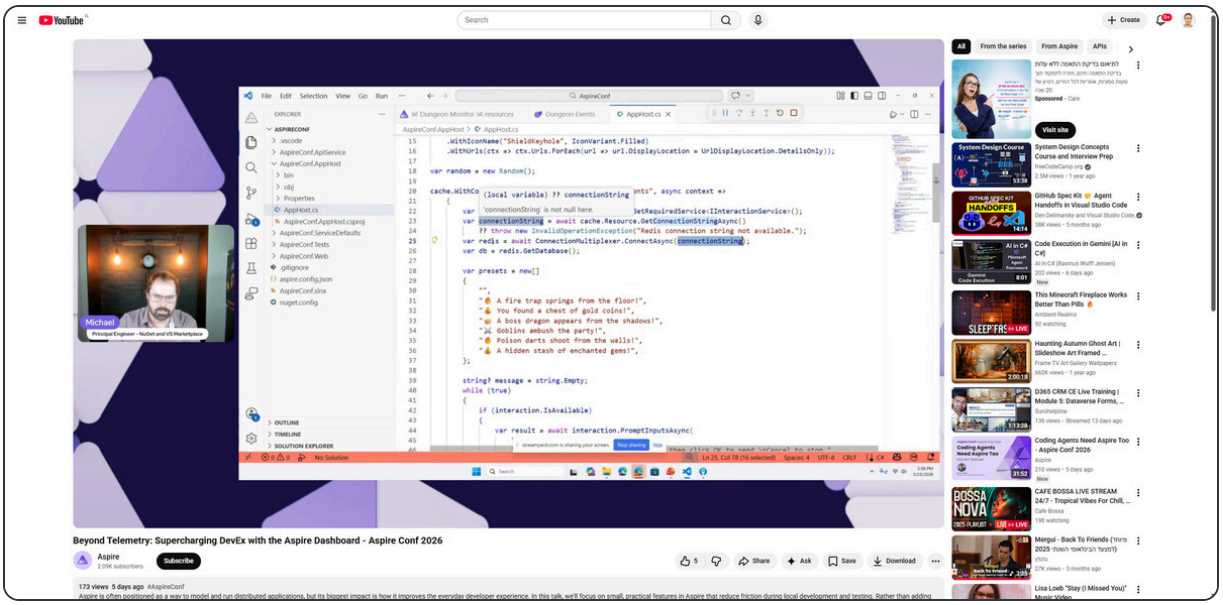
InteractionService PromptInputsAsync modal — Rich modal dialog with a preset dropdown (boss dragon, goblin horde, etc.) and free-text field; selected value pushed to Redis and immediately reflected on the dungeon page

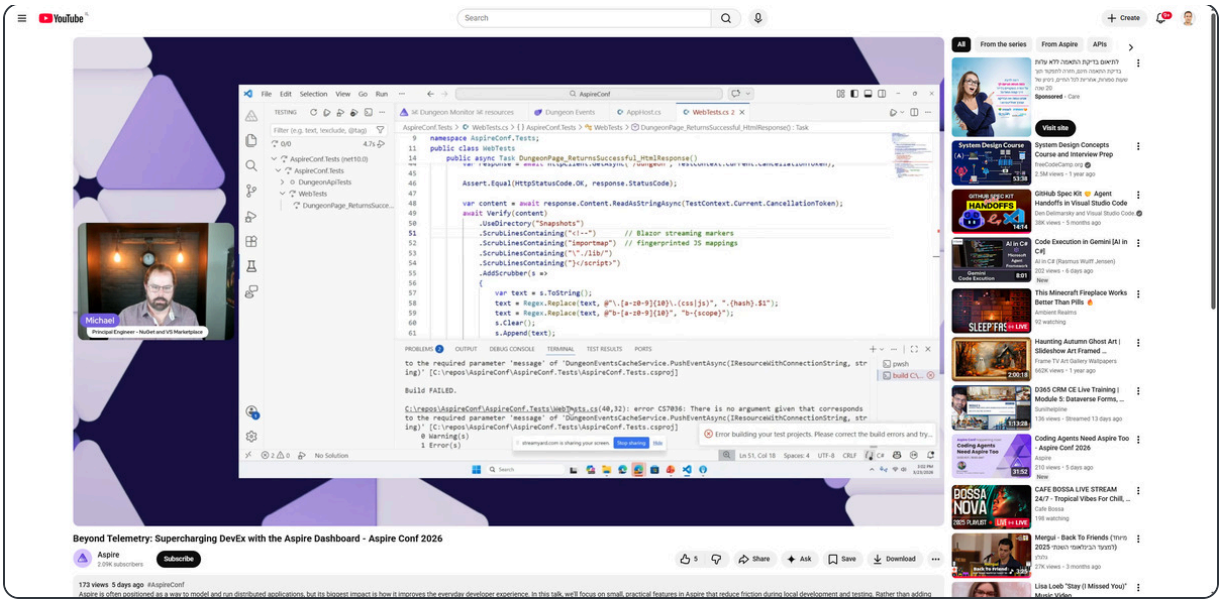
E2E test sharing app-host services — DistributedApplicationTestingBuilder loaded same app host; DungeonEventsCacheService injected via DI into test to seed data; Verify.Tests snapshotted the full HTTP response




Adding Custom URLs
.WithUrlForEndpoint<ProjectResource>(string endpointName, Func<EndpointReference, ResourceUrlAnnotation> callback)
builder.AddProject<Projects.AspireConf_Web>("web")
.WithUrlForEndpoint("https", ep => new()
{
Url = $"{ep.Url}/dungeon",
DisplayText = "Dungeon",
DisplayOrder = 0,
DisplayLocation = UrlDisplayLocation.SummaryAndDetails,
});
Resources
Join in for bingo - aspireify.live
Join in for bin
Beyond Telemetry: Supercharging DevEx with the Aspire Dashboard - Aspire Conf 2026
Aspire
Subscribe


AspireConf
Home
Dungeon
About
Dungeon Events
Total: 6
Traps: 1
Loot: 2
Boss: 1
Refresh
A hidden stash of enchanted gems!
5s ago
Poison darts shoot from the walls!
5s ago
Goblins ambush the party!
5s ago
Beyond Telemetry: Supercharging DevEx with the Aspire Dashboard - Aspire Conf 2026
Aspire
Subscribe


Beyond Telemetry: Supercharging DevEx with the Aspire Dashboard - Aspire Conf 2026
Aspire
Subscribe



· Code Samples

Custom dashboard application title with emoji

```
var builder = DistributedApplication.CreateBuilder(args,
    new DistributedApplicationOptions
    {
        DashboardApplicationName = "🏰 Dungeon DevEx"
    });
```

WithUrlForEndpoint - display text, order, and visibility control

```
webResource
    .WithUrlForEndpoint("https", u ⇒
    {
        u.DisplayText     = "Home";
        u.DisplayOrder    = 1;
        u.DisplayLocation = ResourceUrlAnnotationDisplayLocation.SummaryAnd
    })
    .WithUrls(ctx ⇒
    {
        foreach (var url in ctx.Urls)
            url.DisplayLocation = ResourceUrlAnnotationDisplayLocation.Deta
    });
```

Injecting a custom URL not tied to a service endpoint

```
webResource.WithUrlForEndpoint("https", ep ⇒
    new ResourceUrlAnnotation
    {
        Url             = ep.Url + "/dungeon",
        DisplayText     = "Dungeon",
        DisplayOrder    = 0,
        DisplayLocation = ResourceUrlAnnotationDisplayLocation.SummaryAndDe
    });
```

WithIconName using Fluent UI icon catalog names

```
cacheResource.WithIconName("DataTrending",  IconVariant.Filled);
apiResource  .WithIconName("ShieldKeyhole", IconVariant.Regular);
```

WithCommand with full CommandOptions (highlighted, confirmation, icon)

```
resource.WithCommand(
    "seed-events",
    "Seed Dungeon Events",
    async ctx ⇒
    {
        var svc = ctx.ServiceProvider.GetRequiredService();
        await svc.PushEventsAsync(resource);
        return CommandResults.Success();
    },
    opts ⇒
    {
        opts.IconName            = "ArrowUpload";
        opts.IsHighlighted       = true;
        opts.ConfirmationMessage = "Add dungeon events to Redis cache?";
        opts.Description         = "Seeds test events into the dungeon even
    });
```

InteractionService PromptInputsAsync for rich modal dialogs

```
var interaction = ctx.ServiceProvider.GetRequiredService();
if (interaction.IsAvailable)
{
    var result = await interaction.PromptInputsAsync(
        "Seed Events",
        "Choose a preset or enter a custom event:",
        [
            new InteractionInput { Label = "Preset",       Type = InputType
                                   Choices = ["Boss Dragon", "Goblin Horde"
            new InteractionInput { Label = "Custom Event", Type = InputType
        ]);
    await svc.PushEventsAsync(resource, result.Values["Preset"]);
}
```

Sharing DI services between WithCommand and DistributedApplicationTestingBuilder E2E test

```
// App host - register service
builder.Services.AddSingleton();

// E2E test
var appHost = await DistributedApplicationTestingBuilder
    .CreateAsync();
await using var app = await appHost.BuildAsync();
var svc    = app.Services.GetRequiredService();
await svc.PushEventsAsync(/* dungeon resource */);
var client = app.CreateHttpClient("webfrontend");
var html   = await client.GetStringAsync("/dungeon");
await Verify(html);
```

AI & AGENTS

Coding Agents Need Aspire Too

🎤 Pierce, Maddie Montaquilla

⏱ 31:51 | ▶ Watch Session

> *Your coding agents are only as good as the context they can access. Aspire hands them the keys — your entire app topology, real-time logs and traces, and resource commands like stop and restart — all with zero setup. You'll see how Aspire turns your agents from helpful assistants into full-stack collaborators you can actually trust. This talk is from #AspireConf on March 23rd, 2026. 🎤 Speaker: Pierce Boggan 🎓 Watch the other sessions from #AspireConf: https://www.youtube.com/playlist?list=PL*

· Summary

Pierce, PM lead for VS Code and GitHub Copilot, joined Maddie Montaquilla for what was explicitly billed as a 'user study' - Pierce admitted he'd learned Aspire the night before the conference. The resulting session was one of the most illuminating of the day: a real-time demonstration of how Aspire's new agent-readiness features look to

a complete newcomer wielding GitHub Copilot in full autonomy mode, with Maddie narrating from the perspective of someone who built these features.

Pierce started with a vibe-coded Next.js coloring-book app built overnight - gradients everywhere, margins everywhere, the telltale signs. The experiment: could GitHub Copilot, equipped only with Aspire's skill file and CLI, add OpenTelemetry tracing and metrics integration to his app, then validate its own work? He initiated the agent using VS Code Copilot's 'autopilot' mode - a feature that combines YOLO tool-approval (no manual clicks to allow tool calls) with active model gaslighting to prevent the model from declaring victory prematurely. The agent autonomously called `aspire doc search` to find OTel integration docs, used `aspire start` to launch the app in the background, read back logs and structured traces, restarted the app host as needed, and ultimately surfaced traces and metrics flowing through the Aspire dashboard - all without Pierce writing a single line of code. The Playwright CLI was invoked for headless browser verification, with screenshots saved locally for inspection.

The broader conversation was arguably the session's most valuable content. Pierce articulated four properties that make Aspire uniquely AI-native: (1) the CLI gives agents a well-structured interface with zero configuration overhead; (2) the skill file solves the context problem by teaching agents available commands upfront; (3) `aspire start` + log/trace CLI commands close the validation loop (write code → run → read output → iterate); and (4) Playwright CLI integration closes the browser testing loop. He demoed VS Code's sessions panel - draggable to a floating window, filterable by status - and discussed the `aspire start --isolated` work tree feature enabling three to five simultaneous parallel agent experiments on the same codebase with no port conflicts. Pierce closed with his vision of a self-improving agent loop: use Aspire traces to automatically detect slow endpoints, kick off an agent to fix them, verify with Playwright, and commit - a continuous autonomous improvement cycle.

· Key Takeaways

→ Pierce learned Aspire the previous night and had an AI agent successfully add OTel in < 30 minutes

→ VS Code Copilot 'autopilot' mode = YOLO tool approvals + model gaslighting to prevent premature completion

→ Aspire skill file solves agent context problem - agents discover available commands automatically

→ `aspire doc search` lets agents look up aspire.dev docs on-demand during code generation tasks

→ `aspire start` (background) + CLI log/trace commands complete the agent validation feedback loop

→ Playwright CLI + Aspire = full browser-based feedback loop with headless screenshots and DOM interaction

→ `aspire start --isolated` enables 3-5 parallel agent work trees on same app with zero port conflicts

→ VS Code sessions panel is draggable to a floating window; supports managing multiple simultaneous agent chats

→ Pierce declared Aspire 'the most AI-native dev framework' based on one night of experience

· Announcements & Features

VS Code Copilot 'autopilot' mode — Combines YOLO tool-approval with active model gaslighting to keep the agent iterating until truly done; prevents the common pattern of premature completion

VS Code sessions panel — Draggable floating panel showing all in-progress and historical Copilot chat sessions with filtering; accessible from the VS Code title bar

Aspire CLI output to file — Aspire CLI outputs written to file rather than stdout to prevent polluting agent context windows with noisy build/log output

work tree isolation (`aspire start --isolated`) — Custom Aspire CLI feature enabling each agent work tree to run its own isolated app host with no port conflicts for safe parallel experiments

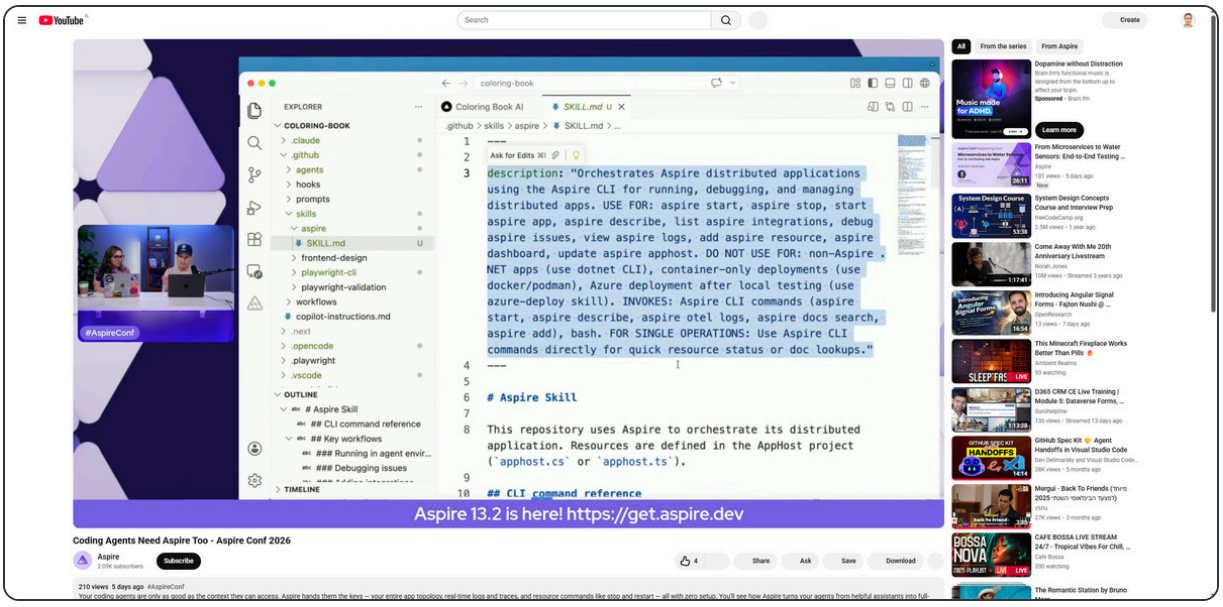
· Demos & Live Code

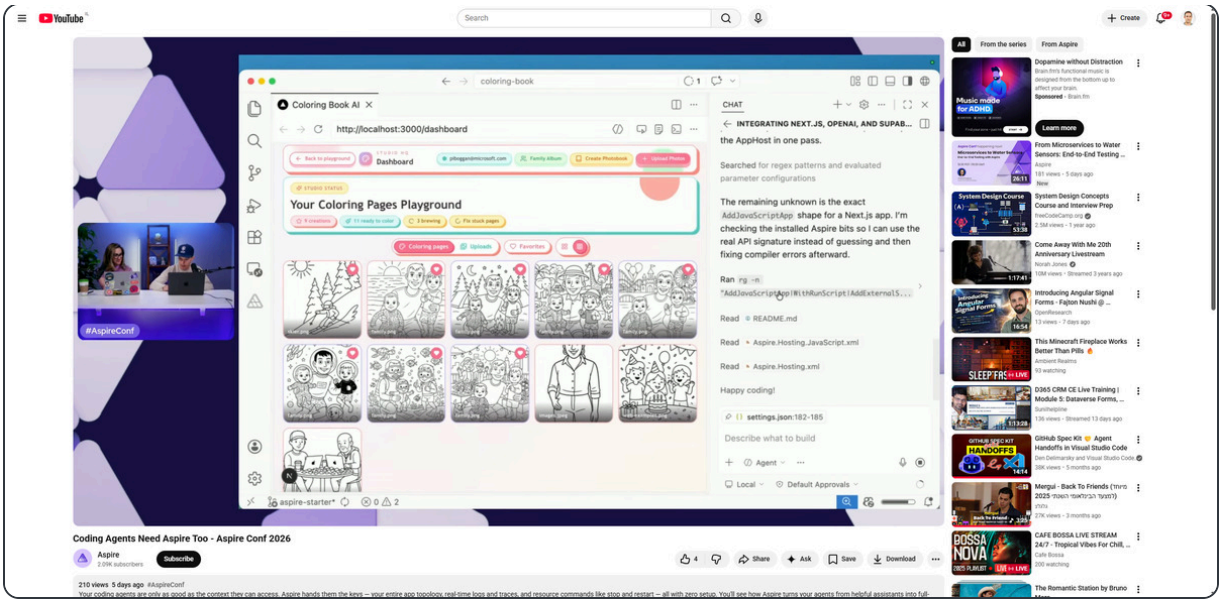
OTel integration added to Next.js app by Copilot agent — From a bare Next.js coloring-book app, Copilot in autopilot mode called aspire doc search, aspire start, read logs, restarted app host, and surfaced structured traces + metrics in the Aspire dashboard

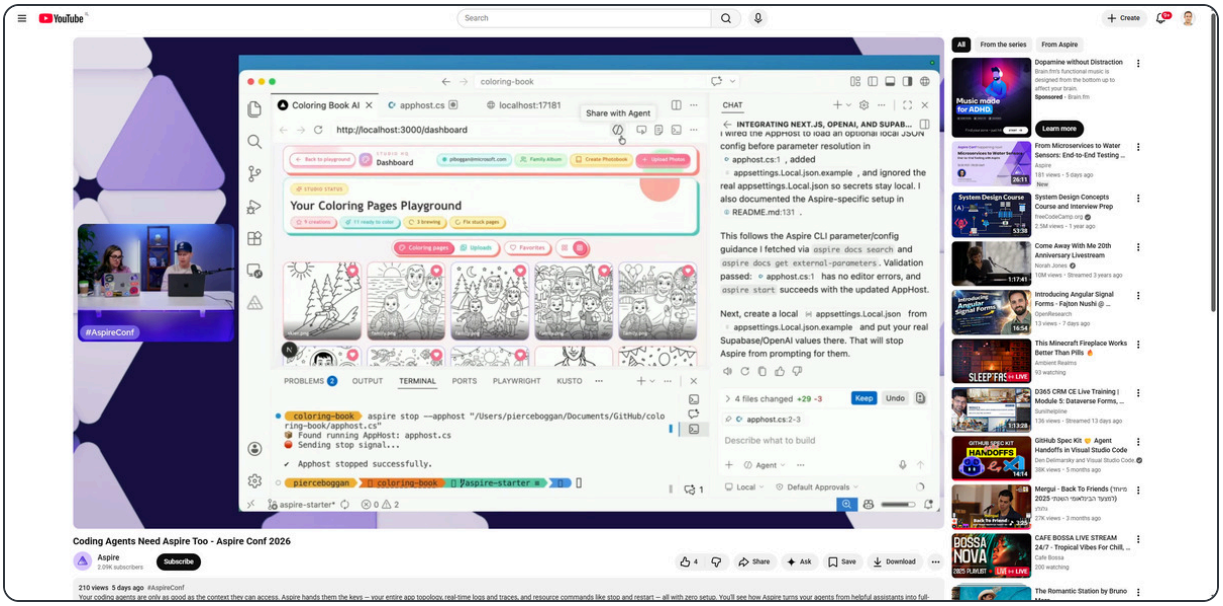
VS Code autopilot mode with Aspire skill — Showed agent dropdown with autopilot option selected; agent kept iterating through errors and restarts without manual prompting, gaslighting itself to continue

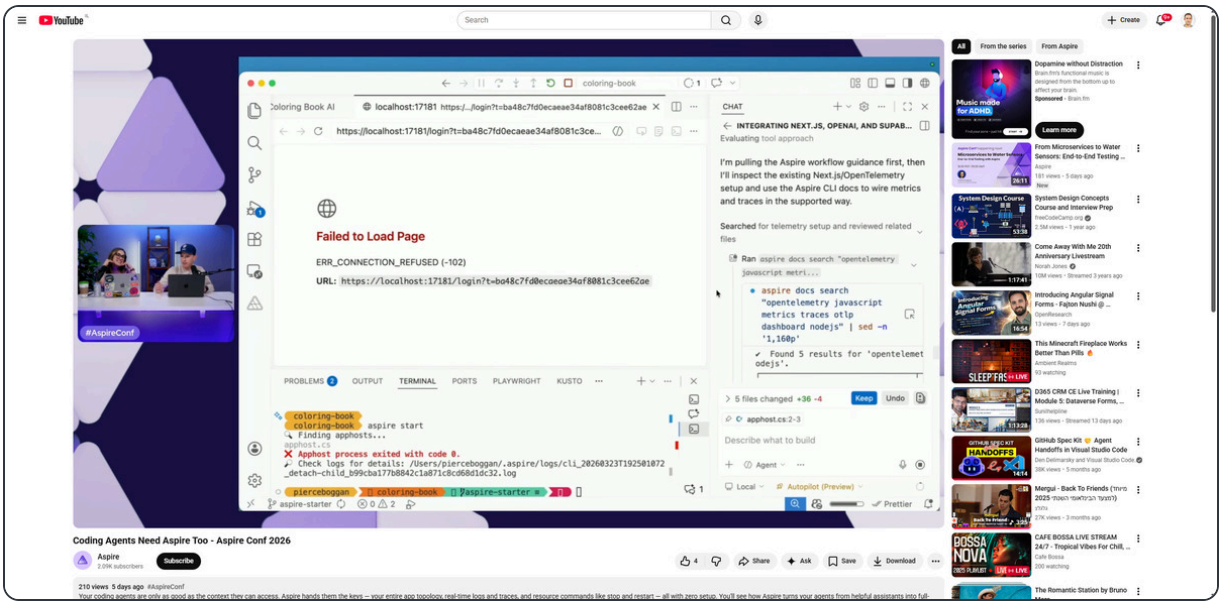
Playwright CLI headless browser verification loop — Agent called Playwright CLI to take headless screenshots of the running app, saved them to the project, and used them to confirm OTel integration was working before completing

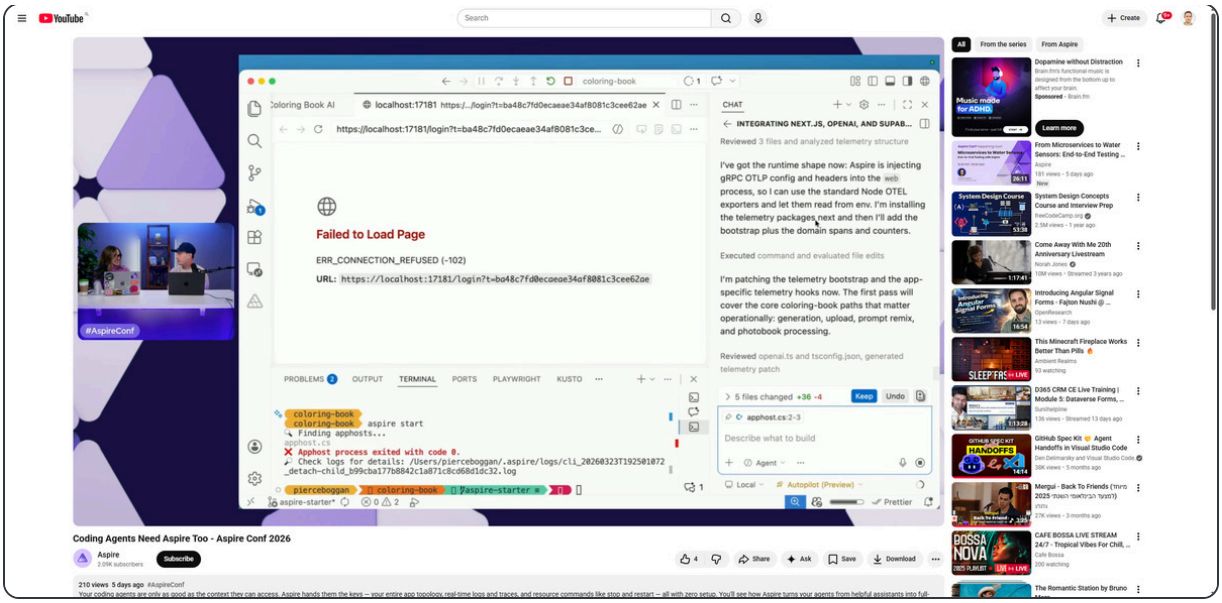
VS Code sessions panel and parallel work trees — Demonstrated dragging the sessions panel to a floating window; showed running multiple Copilot sessions simultaneously with isolated Aspire instances for each work tree




Coding Agents Need Aspire Too - Aspire Conf 2026
Aspire
Subscribe
#AspireConf
Coding Agents Need Aspire Too - Aspire Conf 2026


Coding Agents Need Aspire Too - Aspire Conf 2026
Aspire
Subscribe
#AspireConf
Coding Agents Need Aspire Too - Aspire Conf 2026


Failed to Load Page
ERR_CONNECTION_REFUSED (-102)
Coding Agents Need Aspire Too - Aspire Conf 2026
Aspire
Subscribe
#AspireConf
Coding Agents Need Aspire Too - Aspire Conf 2026



· Code Samples

GitHub Copilot autopilot mode workflow with Aspire (conceptual)

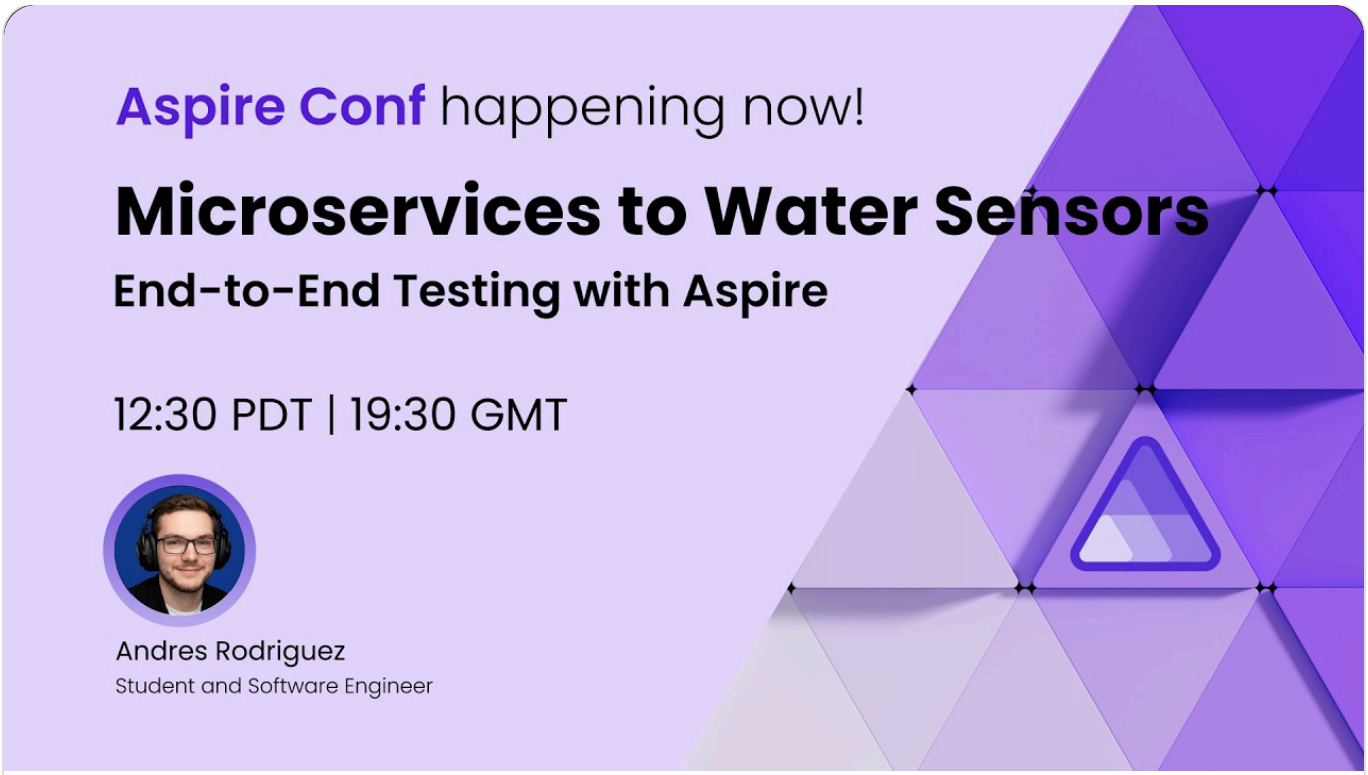
```
# In VS Code Copilot chat:
# 1. Agent type dropdown → select 'autopilot'
# 2. Prompt: 'Add OpenTelemetry integration with traces and metrics'
#
# Agent autonomously calls:
#   aspire doc search 'opentelemetry nextjs'
#   aspire start
#   aspire logs --resource app
#   aspire otel spans
# ...iterates until traces appear in the Aspire dashboard
```

Parallel agent work trees with isolated Aspire instances

```
# Terminal 1 (Agent A working on feature-A git worktree)
aspire start --isolated   # gets unique port range e.g. 7100-7110

# Terminal 2 (Agent B working on feature-B git worktree)
aspire start --isolated   # gets unique port range e.g. 7200-7210

# No port conflicts - each agent has its own dashboard and app host
```

Self-improving agent loop concept enabled by Aspire traces

```
# Conceptual continuous improvement workflow:
# 1. aspire otel spans | grep 'duration > 500ms'  → find slow endpoints
# 2. Copilot: 'These endpoints in aspire traces are slow, go fix them'
# 3. Agent iterates: edit code → aspire start → aspire otel spans → veri
# 4. Playwright CLI takes screenshot to confirm UI still works
# 5. Agent commits fix, moves to next slow endpoint
```



TESTING

From Microservices to Water Sensors : End-to-End Testing with Aspire

🎤 Andres Rodriguez, Maddy Montaquila

⏱ 26:11 | ▶ Watch Session

> *In this session, Andres will walk through how he uses Aspire to test an end-to-end pipeline that spans from cloud microservices to physical IoT devices — specifically an automated irrigation system powered by Arduino sensors and water pumps. You'll learn how Aspire's orchestration and integration testing capabilities can verify not just your web APIs and databases, but the full journey of data from a soil moisture sensor through HTTP endpoints to a real-time dashboard. Whether you're building tr*

· Summary

In one of the most unconventional sessions at AspireConf 2026, student developer Andres Rodriguez demonstrated how transcends its cloud-native roots to orchestrate end-to-end testing for a real-world environmental IoT platform deployed across Puerto Rico. His system ingests data from three categories of physical

sensors — soil moisture and water sensors, air quality/pollution sensors, and rain gauges — routing telemetry through HTTP endpoints to a real-time dashboard. The entire stack, from Azure PostgreSQL Flexible Server and EF Core migrations to a dedicated data seeder, is described in a single Aspire AppHost that serves triple duty: development environment, production deployment manifest, and integration test harness.

Andres walked through how the same AppHost that drives `azd` deployments to Azure Container Apps also bootstraps a complete testing environment with zero manual container management. In production the seeder runs as a Container App Job (fire-and-forget); in testing, passing ASPNETCORE_ENVIRONMENT='testing' switches it to seed lightweight test fixtures instead. The integration tests themselves use `DistributedApplicationTestingBuilder.CreateAsync()` to spin up the entire app host, wait for health checks to propagate through the dependency chain (databases -> seeder → monolith portal), then obtain typed HTTP clients per domain module and fire real HTTP assertions. No Docker Compose. No manual port mapping. No connection string copy-paste.

A highlight of the session was an impromptu live debugging episode: Andres ran `dotnet test` moments before the presentation and a PostgreSQL authentication failure surfaced, triggered by stale Docker volumes holding old credential state. Maddy Montaquila joined on-screen to guide him through `aspire secret list`, `aspire secret delete`, and `aspire ps`/`aspire stop` to nuke the conflicting app hosts — a candid, real-world tutorial on Aspire's secret management CLI. Once secrets were cleared and volumes removed, the tests passed cleanly, illustrating exactly the failure modes Aspire helps you debug.

Andres also previewed a custom deployment script using C# single-file top-level applications and the new `Aspire.Deploy(resourceName)` API. The script reads a list of services affected by the latest Git commit and deploys only those resources — selective partial deployment in roughly 80 lines of code. The talk closed with a tease of adding custom Aspire commands to the integration test suite, a feature he spotted from an earlier dashboard session he had not yet implemented. This session proved Aspire's architecture is solid enough that a solo student with a production IoT platform can leverage the same abstractions as an enterprise team.

· Key Takeaways

- → Single AppHost covers development, production deployment (Azure Container Apps), and integration testing — no separate test config required
- → Test seeder pattern: same seeder switches behavior via ASPNETCORE_ENVIRONMENT='testing', injecting test-specific fixtures without touching production seed logic
- → DistributedApplicationTestingBuilder.CreateAsync() boots the entire stack including PostgreSQL and workers — no manual Docker container management needed
- → Each domain module exposes a typed HTTP client (e.g., SensorHealthHttpClient) making integration tests clean and strongly-typed
- → New Aspire.Deploy(resourceName) API enables selective partial deployments via C# scripts — only deploy services changed in the latest commit
- → Live debugging demo: aspire ps, aspire stop, aspire secret list/delete used to resolve a PostgreSQL volume/credential conflict in real time
- → Custom dashboard commands (drop database, reset state) are extensible into integration test workflows via the testing fixture

· Announcements & Features

Aspire.Deploy(resourceName) API — New programmatic deployment API allowing selective per-resource deployment from C# scripts, enabling CI/CD pipelines to deploy only services changed in the current commit

aspire secret CLI commands — Secret management commands (aspire secret list, aspire secret delete) demonstrated live for managing user secrets and resolving authentication conflicts between dev and test environments

Custom Dashboard Commands in Integration Tests — Custom commands registered in the AppHost can be invoked from within integration test fixtures — not just from the dashboard UI — enabling test-time database resets and state management

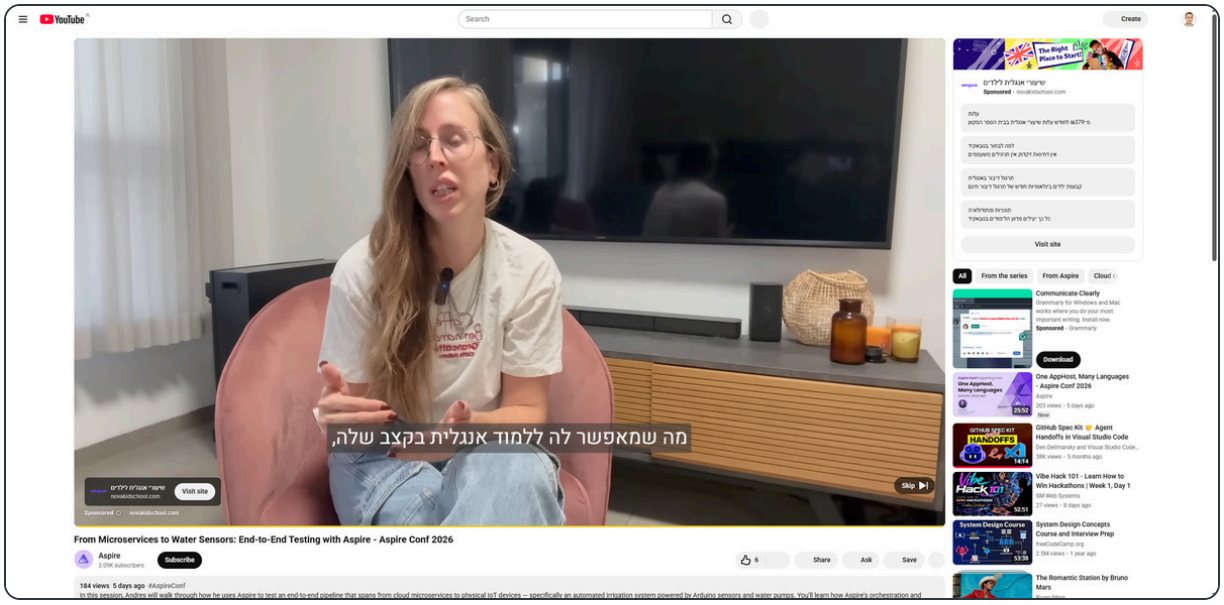
· Demos & Live Code

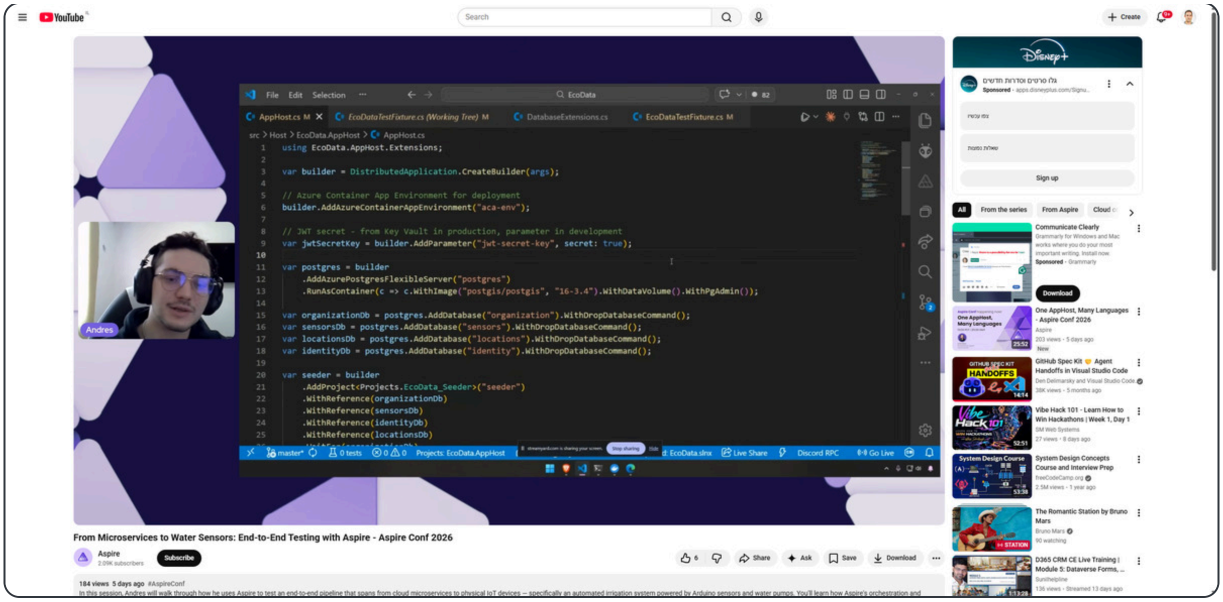
EcoSensor IoT Platform AppHost — Full Azure Container Apps AppHost for a Puerto Rico environmental monitoring platform: Azure PostgreSQL Flexible Server, seeder Container App Job, modular monolith portal, ingestion worker, JWT secrets via Azure Key Vault, and min/max replica scaling configuration

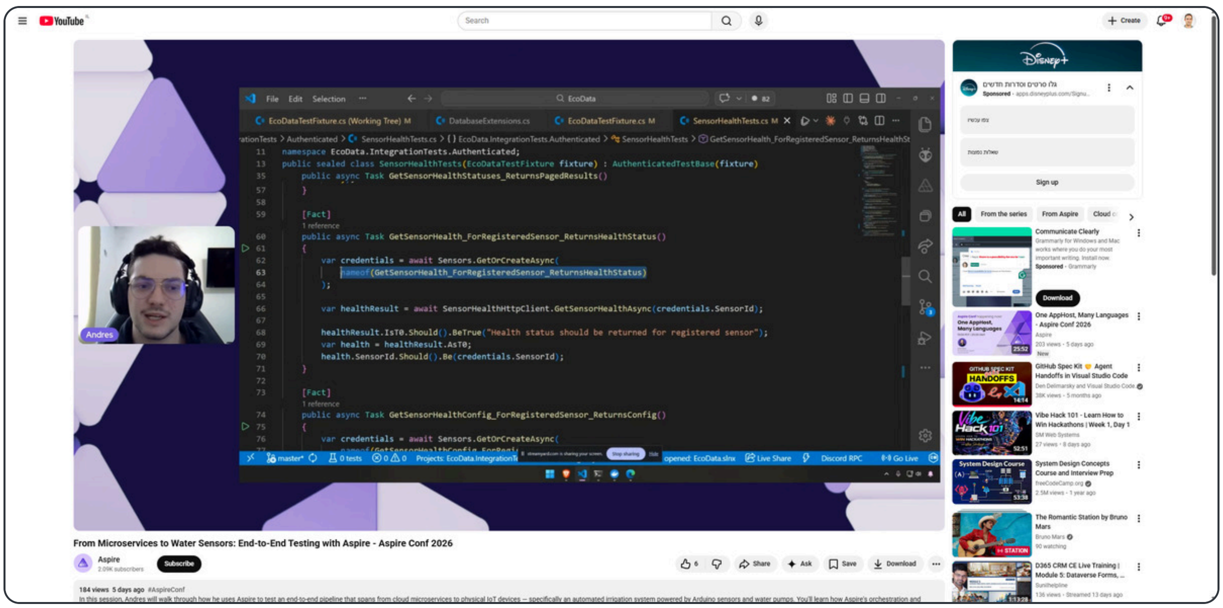
Integration Test Fixture Walkthrough — Live walkthrough of DistributedApplicationTestingBuilder spinning up the full app host, seeding test data, logging in as admin via HTTP, and running sensor health assertions with per-method sensor naming to avoid cross-test data collisions

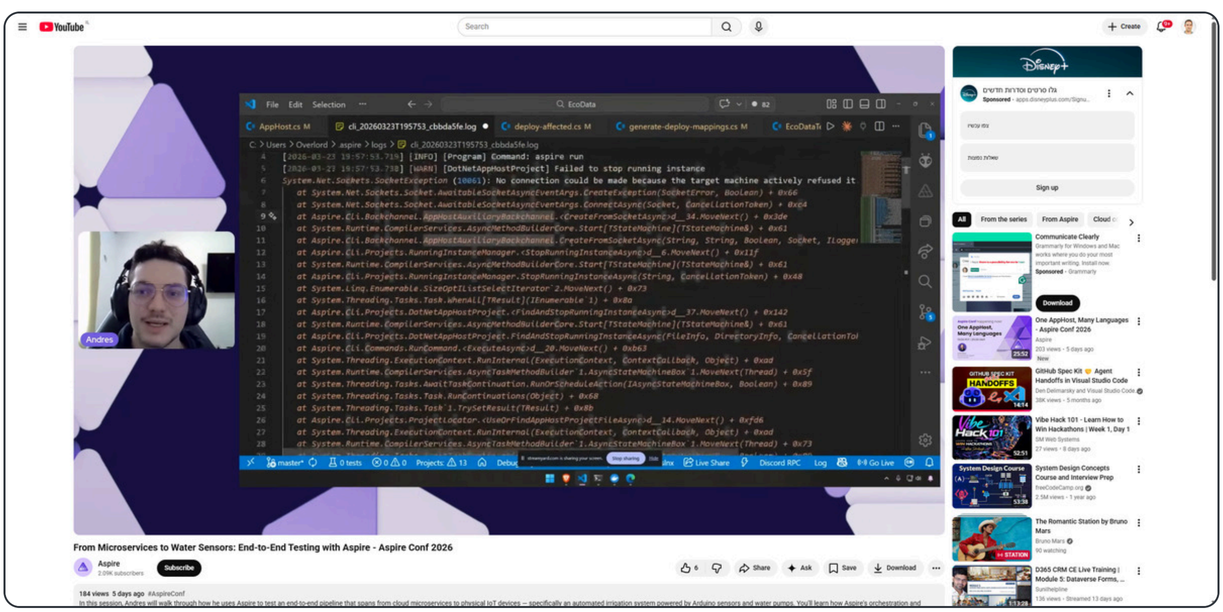
Selective Deployment C# Script — ~80-line C# top-level program using Aspire.Deploy() to read Git-affected services and deploy only changed resources to Azure Container Apps

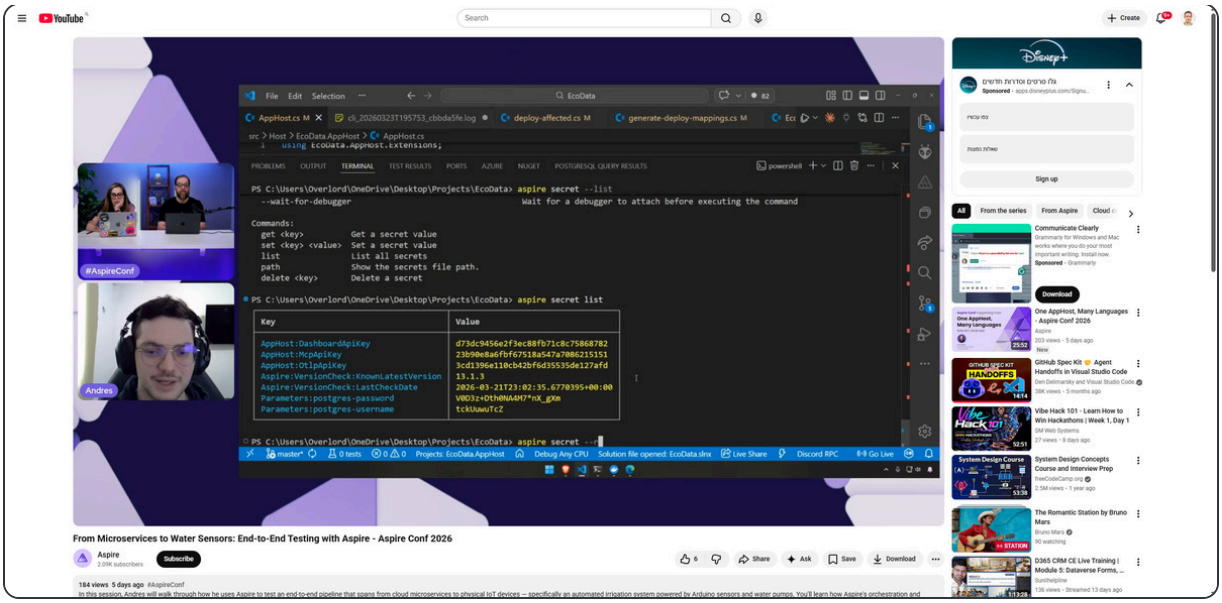
Live Secret/Volume Debugging — Unscripted live debug session resolving a PostgreSQL authentication failure using aspire ps, aspire stop, and aspire secret delete to clear stale Docker volume credentials




From Microservices to Water Sensors: End-to-End Testing with Aspire - Aspire Conf 2026
Aspire
Subscribe
184 views 5 days ago #AspireConf
In this session, Andres will walk through how he uses Aspire to test an end-to-end pipeline that spans from cloud microservices to physical IoT devices — specifically an automated irrigation system powered by Arduino sensors and water pumps. You'll learn how Aspire's orchestration and
Andres
using EcoData.AppHost.Extensions;
var builder = DistributedApplication.CreateBuilder(args);
// Azure Container App Environment for deployment
builder.AddAzureContainerAppEnvironment("aca-env");
// JWT secret - from Key Vault in production, parameter in development
var jwtSecretKey = builder.AddParameter("jwt-secret-key", secret: true);
var postgres = builder
.AddAzurePostgresFlexibleServer("postgres")
.RunAsContainer(c => c.WithImage("postgis/postgis", "16-3.4").WithDataVolume().WithPgAdmin());
var organizationDb = postgres.AddDatabase("organization").WithDropDatabaseCommand();
var sensorsDb = postgres.AddDatabase("sensors").WithDropDatabaseCommand();
var locationsDb = postgres.AddDatabase("locations").WithDropDatabaseCommand();
var identityDb = postgres.AddDatabase("identity").WithDropDatabaseCommand();
var seeder = builder
.AddProject<Projects.EcoData_Seeder>("seeder")
.WithReference(organizationDb)
.WithReference(sensorsDb)
.WithReference(identityDb)
.WithReference(locationsDb)


From Microservices to Water Sensors: End-to-End Testing with Aspire - Aspire Conf 2026
Andres
namespace EcoData.IntegrationTests.Authenticated;
public sealed class SensorHealthTests(EcoDataTestFixture fixture) : AuthenticatedTestBase(fixture)
public async Task GetSensorHealth_ForRegisteredSensor_ReturnsHealthStatus()
var credentials = await Sensors.GetOrCreateAsync(
nameof(GetSensorHealth_ForRegisteredSensor_ReturnsHealthStatus)
var healthResult = await SensorHealthHttpClient.GetSensorHealthAsync(credentials.SensorId);
healthResult.IsT0.Should().BeTrue("Health status should be returned for registered sensor");
var health = healthResult.AsT0;
health.SensorId.Should().Be(credentials.SensorId);
public async Task GetSensorHealthConfig_ForRegisteredSensor_ReturnsConfig()


From Microservices to Water Sensors: End-to-End Testing with Aspire - Aspire Conf 2026
Andres
[INFO] [Program] Command: aspire run
[WARN] [DotNetAppHostProject] Failed to stop running instance
System.Net.Sockets.SocketException (10061): No connection could be made because the target machine actively refused it



· Code Samples

Integration test fixture: create app host, set testing environment, await portal resource readiness

```
var appHost = await DistributedApplicationTestingBuilder
    .CreateAsync();

// Switch seeder to test data mode
appHost.Configuration["ASPNETCORE_ENVIRONMENT"] = "testing";

await using var app = await appHost.BuildAsync();
await app.StartAsync();

// Wait for entire dependency chain: postgres → seeder → portal
await app.WaitForResourceAsync("eco-portal", KnownResourceStates.Running);
```

AppHost: production seeder as Container App Job, dev/test seeder as project

```
var seeder = builder.AddProject("seeder")
    .WithEnvironment("ASPNETCORE_ENVIRONMENT",
        builder.Environment.EnvironmentName)
    .WaitFor(postgresDb);

// In production: run once as a Container App Job
```

```
if (builder.ExecutionContext.IsPublishMode)
    seeder.PublishAsAzureContainerAppJob();
```

Selective deployment using new Aspire.Deploy API in a C# top-level script

```
// C# top-level script: deploy only git-affected services
var affected = GetAffectedServices(); // reads from git diff
var resourceMap = new Dictionary {
    ["EcoPortal"] = "eco-portal",
    ["IngestionWorker"] = "ingestion-worker"
};
foreach (var service in affected)
    if (resourceMap.TryGetValue(service, out var resource))
        await Aspire.Deploy(resource);
```

MULTI-LANGUAGE

One AppHost , Many Languages

🎤 Chris Ays, Maddy Montaquila, David Fowler

⏱ 25:51 | ▶ Watch Session

> *Aspire makes polyglot systems feel like one product by letting you run and wire everything through a single AppHost. Chris will show some popular patterns - like a Go backend + Vite frontend, Python API + JS frontend, Spring Boot with PostgreSQL, and C# API with CosmosDB. You'll see the same repeatable workflow for local dev, service discovery, and config across Python, TypeScript, Go, Java, and .NET, without the usual chaotic repo setup and onboarding. This talk is from #AspireConf on March 23*

· Summary

Chris Ays, Principal Software Engineer in Microsoft's Azure Reliability / endOps organization, delivered an energetic tour of Aspire's polyglot story in version 13.2, with a particular focus on what the new TypeScript AppHost unlocks for teams building multi-language systems. His core thesis: modern distributed systems routinely combine Python AI services, Go microservices, Java Spring Boot backends,

and React/TypeScript frontends — and Aspire 13.2 is the first version where all of these can be orchestrated from a single AppHost with unified service discovery, consistent connection strings, and cross-language OpenTelemetry observability through the Aspire dashboard.

Chris demoed A3 (All Aboard Aspire), a sample app showcasing the TypeScript `distributedApplication` builder API. The API mirrors its C# counterpart: `builder.addRedis('cache')`, service references via `.withReference()`, and lifetime configuration. In VS Code with the Aspire extension, the TypeScript AppHost is fully debuggable — Chris set a breakpoint in `apphost.ts`, launched the debug session via `npx tsx`, and stepped through the builder call stack to inspect resource state (host, port, connection strings) live. Language-specific installers run automatically as Aspire resources: `npm install` for Node/React front-ends, `uv sync` for Python services — all visible as resource lifecycle tasks in the dashboard.

A forward-looking section highlighted what's coming rather than shipping today. Python, Go, and Java app hosts are under active development behind experimental feature flags in `aspire.config.json` (e.g., `polyglot_support_enabled`). Running `aspire config list` reveals these flags. The community toolkit already has Java Spring Boot hosting support via `addSpringApp()` with active PRs to deepen it. The Aspire CLI gained templates for TypeScript including an Express+React scaffold that wires up OpenTelemetry boilerplate, an MCP JSON for agent tooling, a `tsconfig.json`, and publish container files — a complete project in seconds.

The session also surfaced a real pain point: OpenTelemetry instrumentation in non-.NET runtimes is still a manual setup step. While .NET gets service defaults (OTel, health checks, service discovery) for free, TypeScript and Python require explicit package installation and wiring. David Fowler, dropping in toward session's end, mentioned the team is exploring an 'OTel my app' agent skill that can wire up telemetry for any language automatically. The session closed with a live `aspire new express-react` scaffolding demo that generated a full TypeScript+Vite frontend+Express backend app host in under a minute — demonstrating how far Aspire's polyglot ergonomics have come.

· Key Takeaways

→ Aspire 13.2 ships the TypeScript AppHost (apphost.ts) as a stable feature — same distributedApplication builder pattern as C#, fully debuggable with VS Code

breakpoints via npx tsx

→ aspire.config.json centralizes app host language, SDK version, package references, and experimental feature flags (e.g., polyglot_support_enabled)

→ Language-specific dependency installers (npm install for Node, uv sync for Python) run automatically as Aspire resource lifecycle tasks visible in the dashboard

→ Experimental Python, Go, and Java app hosts available behind feature flags — aspire config list shows all available experimental settings

→ aspire new gains TypeScript templates: empty TypeScript host, Express+React full-stack scaffold with MCP JSON, tsconfig.json, OTel boilerplate, and container publish files

→ Java Spring Boot hosting available via Aspire Community Toolkit (addSpringApp()), with active PRs to enhance capabilities

→ Cross-language OpenTelemetry still requires manual setup for non-.NET runtimes; a future 'OTel my app' agent skill is being considered to automate this

· Announcements & Features

TypeScript AppHost (Aspire 13.2 stable) — TypeScript app host with distributedApplication builder API, full VS Code debugging support via npx tsx, and automatic installer task management for npm and uv dependencies

aspire.config.json Polyglot Configuration — Centralized configuration file supporting multi-language app hosts, SDK version pinning, package references, and experimental feature flags for Python/Go/Java hosts

aspire new TypeScript Templates — New project scaffolding templates for TypeScript: empty TypeScript host and Express+React full-stack app with OTel wiring, MCP JSON for agent skills, tsconfig.json, and container publish configuration

Experimental Python/Go/Java AppHosts — Python, Go, and Java app hosts under active development, accessible via feature flags in aspire.config.json —

highly experimental, not yet production-ready

aspire config list — CLI command listing all available Aspire config settings including experimental feature flags, enabling developers to opt into preview polyglot features

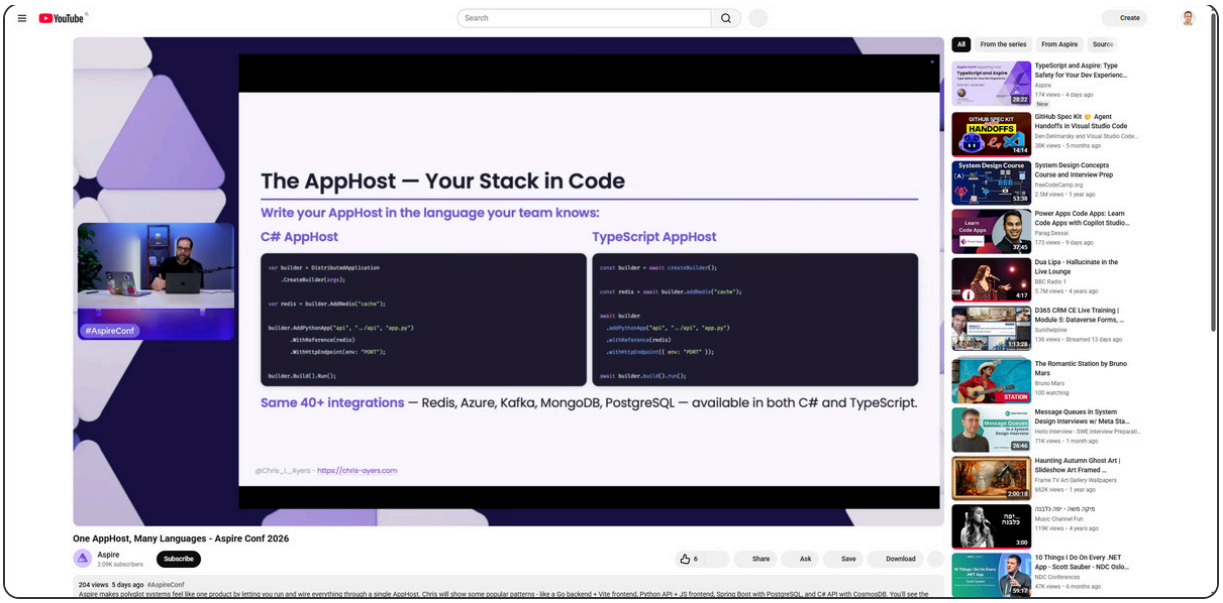
· Demos & Live Code

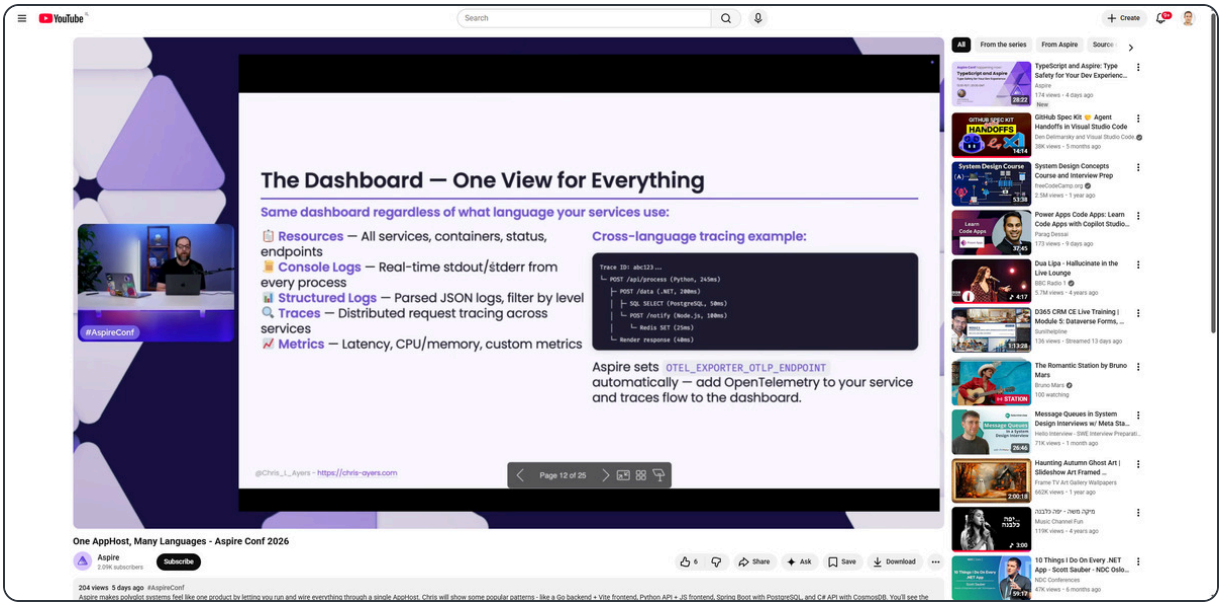
A3 (All Aboard Aspire) TypeScript AppHost Debug Session — Live VS Code debugging of apphost.ts — breakpoint set in TypeScript app host builder code, stepped through call stack via npx tsx debugger to inspect Unicorn app resource host/port/connection strings at runtime

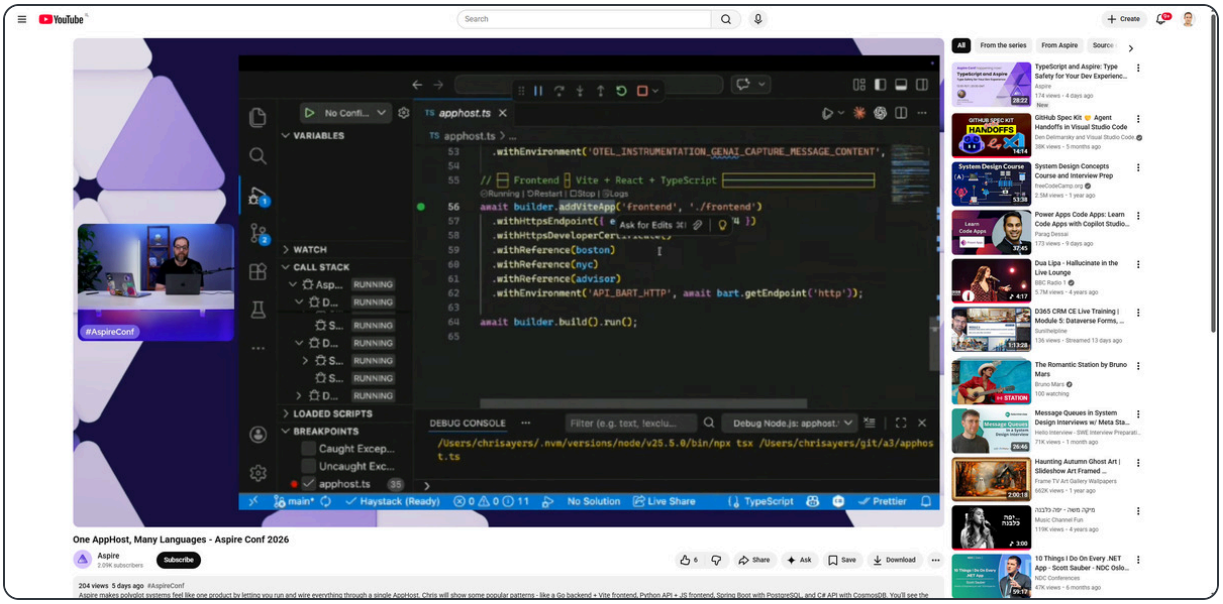
Cross-Language Dashboard (Python, Node, Redis) — Aspire dashboard showing resources from Python (uv sync installer task), Node (npm install installer task), and Redis side-by-side with console logs and resource lifecycle events per language

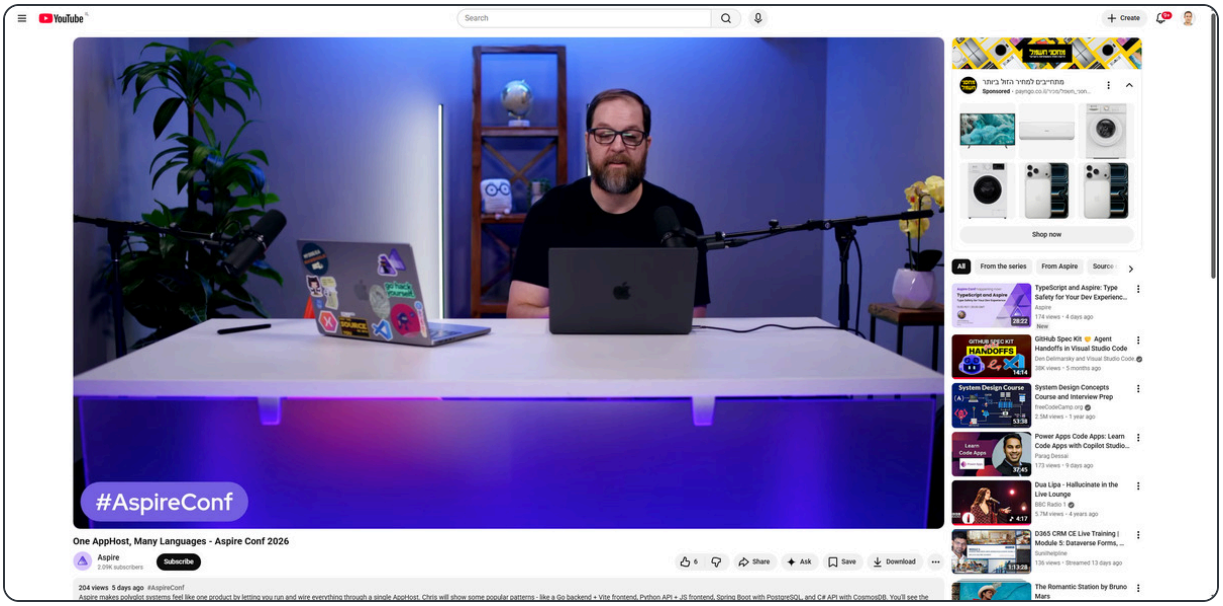
aspire new express-react Scaffolding — Live scaffolding of Express+React TypeScript app host via aspire new, generating apphost.ts, MCP JSON for agent skills, tsconfig, frontend directory, and publish container files — complete in under a minute

Java Spring Boot via Community Toolkit — Code demo showing addSpringApp() from the Aspire Community Toolkit for orchestrating Java Spring Boot services with PostgreSQL dependencies in a TypeScript or .NET app host


The AppHost — Your Stack in Code
Write your AppHost in the language your team knows:
C# AppHost
TypeScript AppHost
Same 40+ integrations — Redis, Azure, Kafka, MongoDB, PostgreSQL — available in both C# and TypeScript.
#AspireConf
One AppHost, Many Languages - Aspire Conf 2026


The Dashboard — One View for Everything
Same dashboard regardless of what language your services use:
Resources — All services, containers, status, endpoints
Console Logs — Real-time stdout/stderr from every process
Structured Logs — Parsed JSON logs, filter by level
Traces — Distributed request tracing across services
Metrics — Latency, CPU/memory, custom metrics
Cross-language tracing example:
Aspire sets OTEL_EXPORTER_OTLP_ENDPOINT automatically — add OpenTelemetry to your service and traces flow to the dashboard.
#AspireConf
One AppHost, Many Languages - Aspire Conf 2026


#AspireConf
One AppHost, Many Languages - Aspire Conf 2026



· Code Samples

TypeScript AppHost (apphost.ts): adding Redis cache and wiring services with distributedApplication builder

```
import { DistributedApplication } from '@microsoft/dotnet-aspire';

const builder = await DistributedApplication.createBuilder();

const cache = builder.addRedis('cache');

const api = builder.addNpmApp('api', '../api')
  .withReference(cache)
```

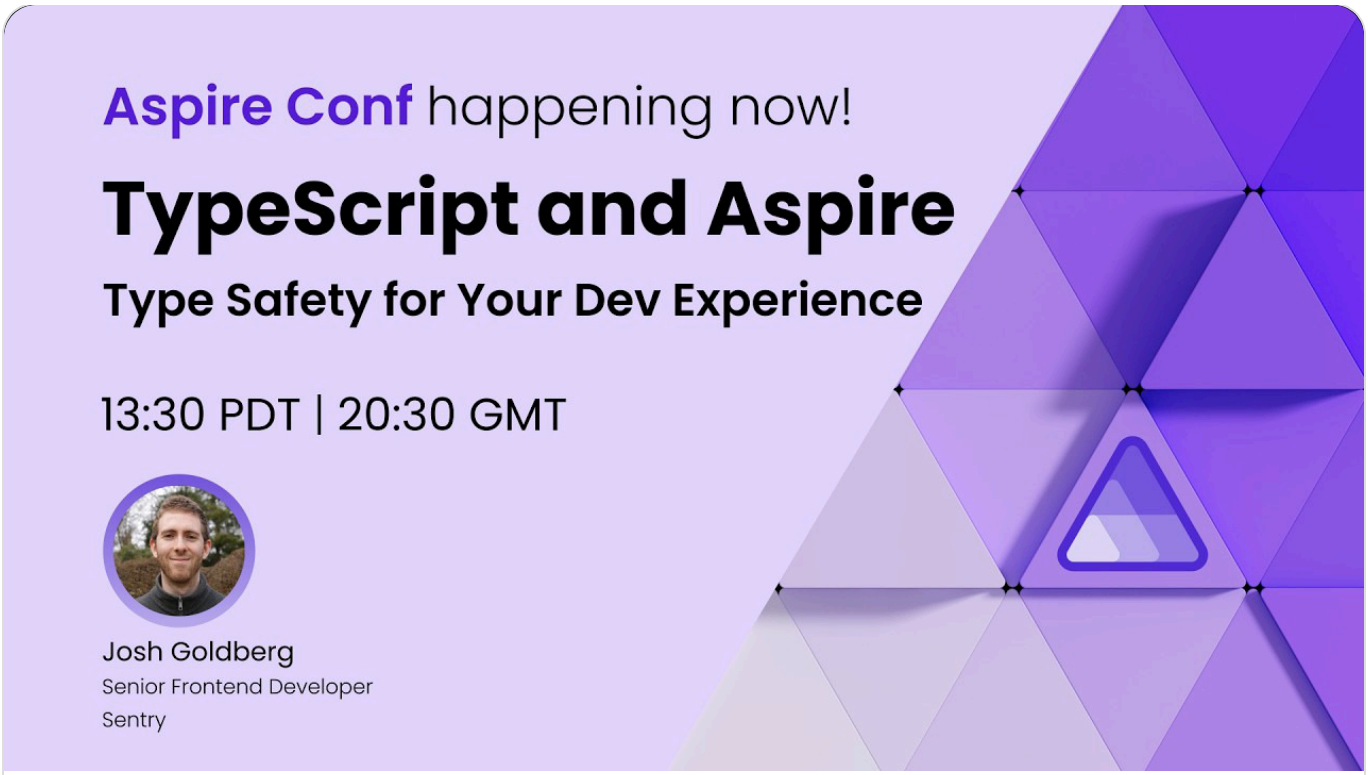
```
  .withLifetime('persistent');

builder.addViteApp('frontend', '../frontend')
  .withReference(api);

await builder.build().run();
```

aspire.config.json: TypeScript app host with polyglot feature flag and SDK version pinning

```
{
  "appHost": {
    "language": "typescript",
    "path": "./apphost.ts"
  },
  "sdk": {
    "version": "13.2.0"
  },
  "features": {
    "polyglot_support_enabled": true
  },
  "packages": {
    "@microsoft/dotnet-aspire": "13.2.0"
  }
}
```

Convention-based connection string consumption in Node.js from Aspire environment injection

```
// Aspire injects connection strings as environment variables
// following ConnectionStrings__{resourceName} convention
const redisConnectionString = process.env['ConnectionStrings__cache'];
const dbConnectionString = process.env['ConnectionStrings__postgres'];
const apiEndpoint = process.env['services__api__http__0'];
```



🛠️ DEVEX & TOOLING

TypeScript and Aspire : Type Safety for Your Dev Experience

🎤 Josh Goldberg, Maddy Montaquila, David Fowler, Chris Ays

⏱️ 28:21 | ▶ Watch Session

> *Types aren't just for your server-side C#: they're a huge benefit in frontend and full-stack logic too! Let's dive into all the wonderfully fully-typed libraries and utilities in a freshly installed Aspire app. We'll cover the basics of how types work in TypeScript compared to traditional languages like C#, how they simultaneously catch bugs and help you write features in your code, and uncover some seriously nifty features of the TypeScript type system along the way. This talk is from #AspireC*

· Summary

Josh Goldberg — author of O'Reilly's Learning TypeScript, Sentry engineer, and co-founder of the TypeScript conference Squiggleconf — took the stage at AspireConf 2026 to make the case that TypeScript's type system is not just JavaScript's safety net: it's a first-class developer experience layer that Aspire's TypeScript AppHost

now exploits to extraordinary effect. The session opened with an architectural argument: 20 years ago, web clients were trivially simple while servers held all the complexity. Today's frontend carries equal or greater complexity — data models, network request types, user session objects, business logic, unit tests — yet JavaScript was never designed for this scale. TypeScript is the fix, and Aspire 13.2 makes it a first-class citizen.

The centerpiece technical revelation was that the Aspire TypeScript SDK auto-generates a 21,500-line TypeScript declaration file exposing every possible AppHost operation with full type annotations. When you Cmd+Click into `addViteApp` in VS Code, you land in this generated file and see precisely typed interfaces like `VolumeOptions`, `AddViteAppOptions` with `runScriptName?: string`, and every other configuration surface. IntelliSense becomes the documentation: as you type in `apphost.ts`, the editor shows parameter names, types, and JSDoc inline. Josh demonstrated live how a wrong type assignment (`runScriptName: false` instead of a string) produces an immediate red squiggly with a precise error message.

Josh then went deep on TypeScript's unique structural typing model, focusing on type narrowing — a feature largely absent from languages like C# in its pure form. He showed how union types (`string | number`) combined with `typeof` guards allow the TypeScript compiler to track the exact type of a value at each code point. He traced a real production bug: a `count` variable never assigned in a `catch` block becomes `undefined`, and `undefined * 2` yields `NaN` — caught statically by TypeScript's control-flow analysis. He also introduced Zod, demonstrating how `z.infer` derives TypeScript types from runtime schema definitions — eliminating duplicate interface declarations for API responses.

The session closed with typed linting — Josh's professional focus area that ships in Aspire's TypeScript dev templates today. `typescript-eslint` enables type-aware lint rules combining ESLint's syntax awareness with TypeScript's type graph. Josh walked through a cascade: a lint rule detects `await onClick` (awaiting a void-returning function), then cascades through automated fixes — remove the await, remove async, remove the now-unnecessary state — collapsing a React component to about two-thirds its original size. He also demoed the `no-floating-promises` rule catching a missing `await` in an Aspire `apphost.ts` init file — the exact subtle async bug that breaks distributed app startup silently.

· Key Takeaways

→ Aspire TypeScript SDK auto-generates a 21,500+ line TypeScript declaration file — every AppHost API (addViteApp, addRedis, VolumeOptions, etc.) is fully typed with IntelliSense and JSDoc, eliminating the need for external documentation lookups

→ Type narrowing is TypeScript's killer feature: the compiler performs control-flow analysis through if/catch/typeof guards to track the exact type of every value at each code point, catching entire classes of runtime bugs statically

→ Union types (string | number) are idiomatic TypeScript and, combined with narrowing, replace verbose null checks and separate error/loading state variables — one state variable can encode both loading and error states

→ JSON.parse returns `any` (unsafe) — replace with Zod's schema.parse() and z.infer for fully type-safe API response deserialization from a single schema definition

→ typescript-eslint type-aware lint rules ship by default in aspire new TypeScript templates: no-floating-promises, logical-assignment-operators, and others detect async mistakes and enable cascading auto-fixes

→ Typed linting cascade demo: detecting await onClick (void) and auto-fixing to remove await, async keyword, derived state — shrinking a React component by approximately one-third

→ The `any` type in TypeScript is the 'YOLO' escape hatch — prefer `unknown` which requires explicit narrowing before use, catching shape mismatches that any silently swallows

· Announcements & Features

Aspire TypeScript SDK Auto-Generated Declarations (21,500+ lines) — Complete TypeScript declaration file shipping with the Aspire TypeScript SDK, covering every AppHost API including VolumeOptions, AddViteAppOptions, WithEndpointOptions, and all builder methods — navigable via Cmd+Click in VS Code

typescript-eslint in Aspire TypeScript Templates — Type-aware linting via typescript-eslint ships by default in aspire new TypeScript project templates,

including no-floating-promises, logical-assignment-operators, and other type-aware lint rules out of the box

Squiggleconf Moving to Spring 2027 at Boston Aquarium IMAX — The TypeScript tooling conference co-founded by Josh Goldberg is moving from fall to spring 2027, to be held at the Boston Aquarium IMAX theater

· Demos & Live Code

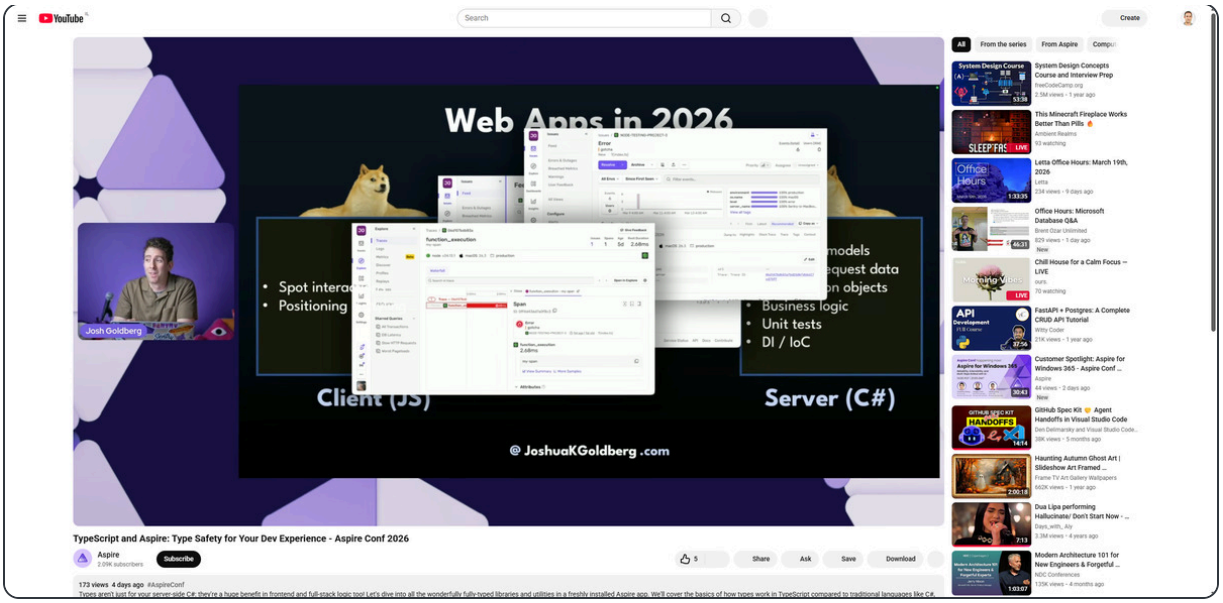
apphost.ts IntelliSense and Type Error Demo — Live VS Code demo: Cmd+Click into addViteApp to navigate the 21,500-line auto-generated Aspire TypeScript declarations, showing VolumeOptions interface; demonstrated immediate type error for boolean assignment to runScriptName (string field)

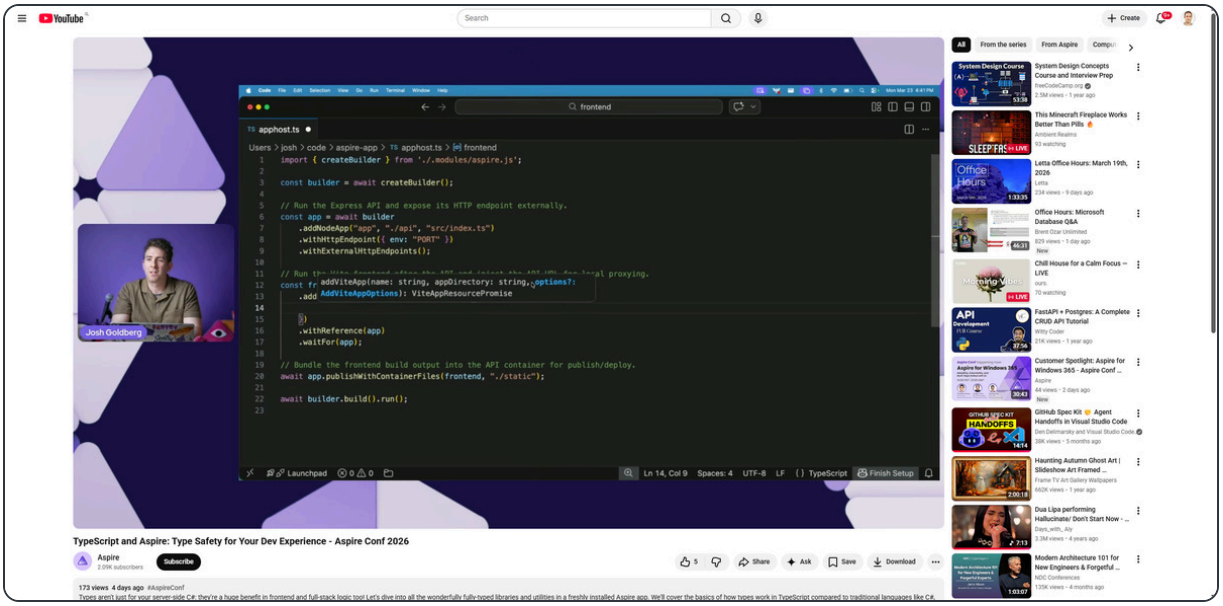
Type Narrowing with Union Types in app.tsx — Live refactor: collapsed separate loading/error useState variables into a single boolean|string union, demonstrating TypeScript narrowing inside if(typeof loading === 'string') block — hover confirms value is narrowed to string inside the branch

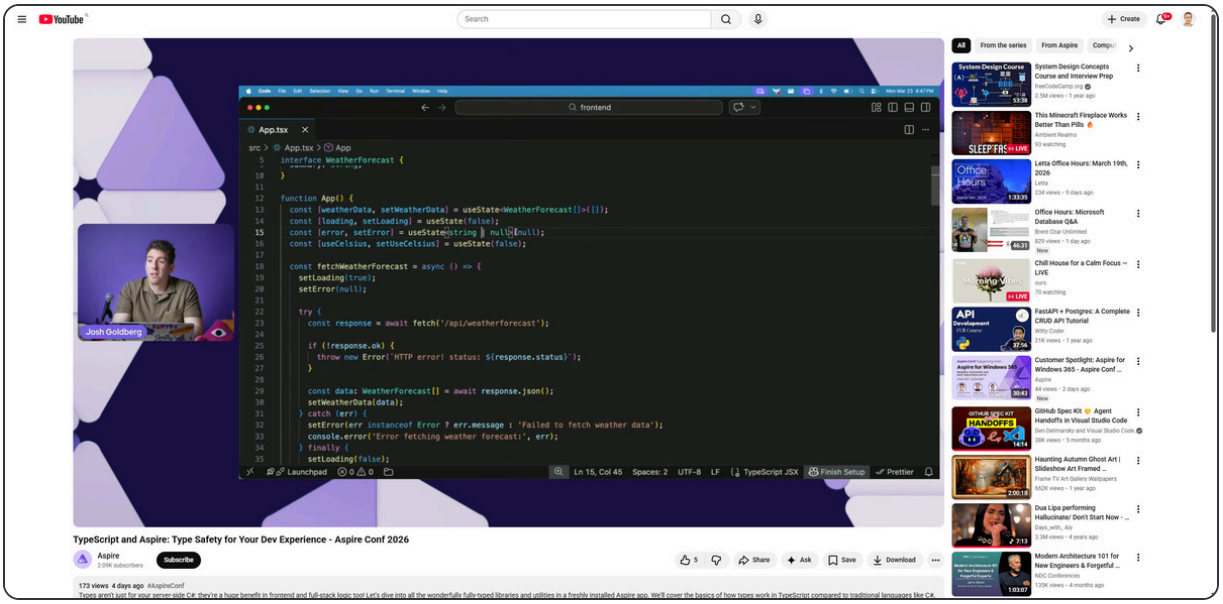
Zod Runtime Schema Validation — Replacing JSON.parse (returns any) with weatherForecastSchema.parse() and z.infer, eliminating the duplicate WeatherForecast interface for type-safe API response deserialization

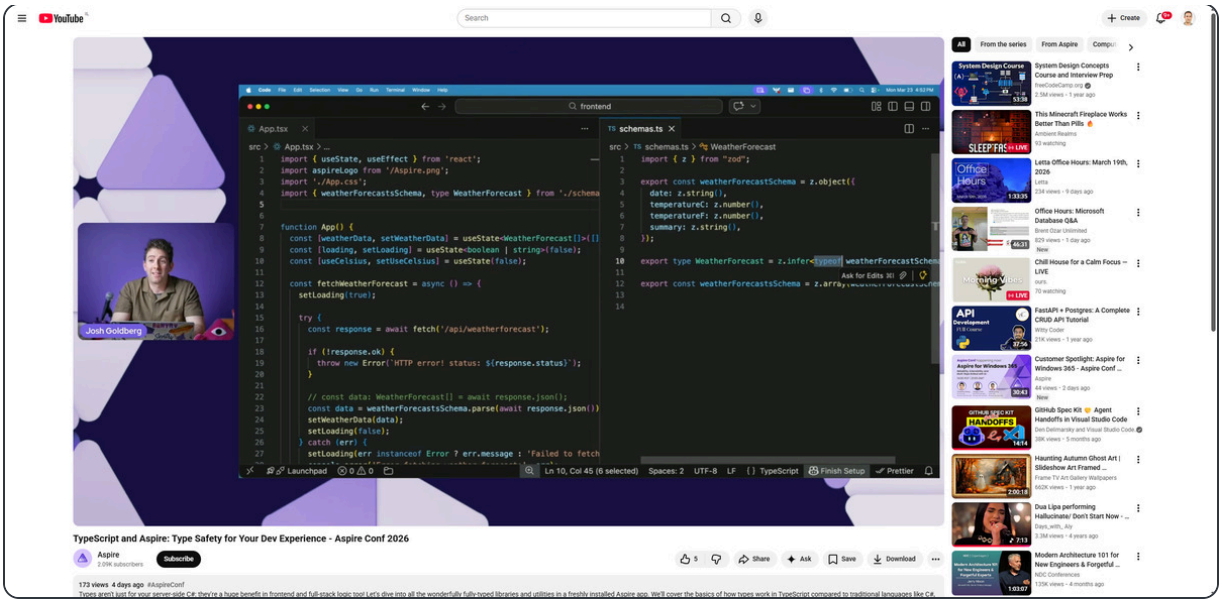
Type-Aware Lint Cascade — Starting from @typescript-eslint/no-floating-promises on await onClick, cascading through auto-fixes: remove await → remove async → inline curry function → remove useState → remove disabled=false, reducing component code by ~33%

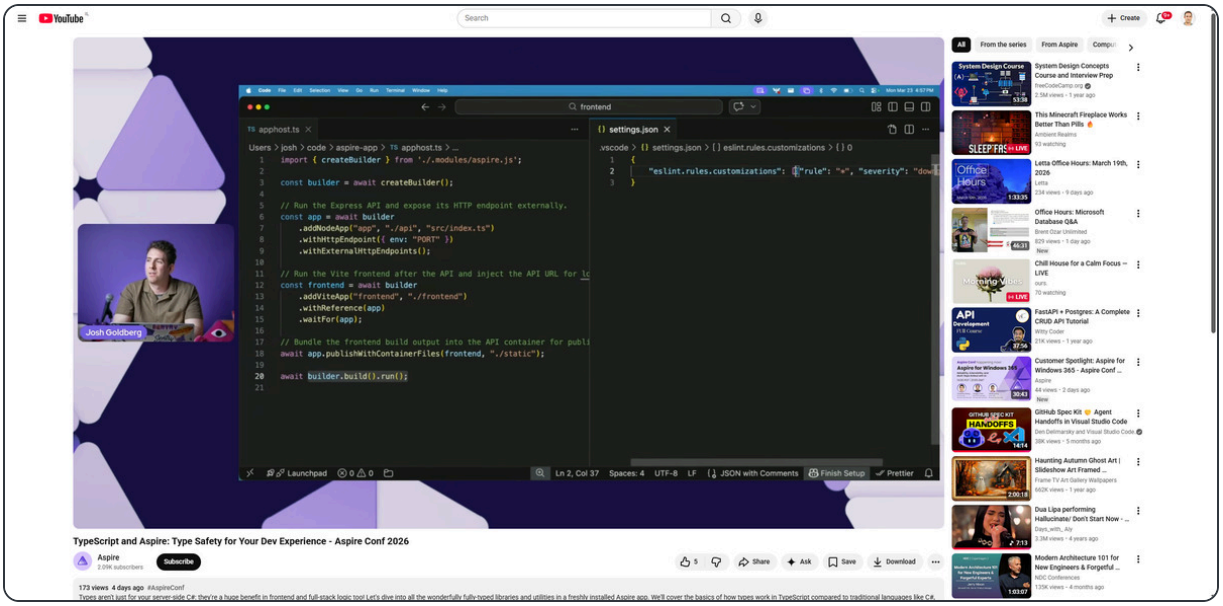
no-floating-promises in apphost.ts — Demo of the no-floating-promises lint rule catching a missing await on an async init call in apphost.ts — the exact async mistake that silently breaks Aspire distributed app startup


Web Apps in 2026
Client (JS)
Server (C#)
@ JoshuaKGoldberg .com
Josh Goldberg
TypeScript and Aspire: Type Safety for Your Dev Experience - Aspire Conf 2026
Aspire
Subscribe


apphost.ts
import { createBuilder } from './.modules/aspire.js';
const builder = await createBuilder();
// Run the Express API and expose its HTTP endpoint externally.
const app = await builder
.addNodeApp("app", "./api", "src/index.ts")
.withHttpEndpoint({ env: "PORT" })
.withExternalHttpEndpoints();
.withReference(app)
.waitFor(app);
// Bundle the frontend build output into the API container for publish/deploy.
await app.publishWithContainerFiles(frontend, "./static");
await builder.build().run();
Josh Goldberg
TypeScript and Aspire: Type Safety for Your Dev Experience - Aspire Conf 2026
Aspire
Subscribe


App.tsx
interface WeatherForecast {
function App() {
const [weatherData, setWeatherData] = useState<WeatherForecast[]>([]);
const [loading, setLoading] = useState(false);
const [error, setError] = useState<string | null>(null);
const [useCelsius, setUseCelsius] = useState(false);
const fetchWeatherForecast = async () => {
setLoading(true);
setError(null);
try {
const response = await fetch('/api/weatherforecast');
if (!response.ok) {
throw new Error(`HTTP error! status: ${response.status}`);
}
const data: WeatherForecast[] = await response.json();
setWeatherData(data);
} catch (err) {
setError(err instanceof Error ? err.message : 'Failed to fetch weather data');
console.error('Error fetching weather forecast:', err);
} finally {
setLoading(false);
Josh Goldberg
TypeScript and Aspire: Type Safety for Your Dev Experience - Aspire Conf 2026
Aspire
Subscribe





· Code Samples

Type narrowing with union types — TypeScript infers string type inside typeof guard

```
let value: string | number = Math.random() > 0.5 ? 'aspire-conf' : 1337;

if (typeof value === 'string') {
  // TypeScript narrows: value is string here
  console.log(value.toUpperCase()); // OK
}
// value.toUpperCase() here → TS error: number has no toUpperCase
```

Zod schema replacing JSON.parse (any) with type-safe runtime validation and z.infer type derivation

```
import { z } from 'zod';

const weatherForecastSchema = z.array(z.object({
  date: z.string(),
  temperatureC: z.number(),
  summary: z.string().optional(),
}));

// Single source of truth: type derived from schema, not duplicated
type WeatherForecast = z.infer;

// Runtime validation — ZodError thrown if shape doesn't match
const data = weatherForecastSchema.parse(await response.json());
```

typescript-eslint no-floating-promises: catching missing await in apphost.ts async init

```
// BAD: missing await on async init — silently breaks app startup
initializeApp(); // eslint: @typescript-eslint/no-floating-promises

// GOOD: quick-fix adds the await operator
await initializeApp();
```

any vs unknown: unsafe escape hatch vs type-safe alternative requiring explicit narrowing

```
// any — unsafe, TypeScript trusts you completely
const unsafe: any = JSON.parse(data);
unsafe.nonexistentProp; // TypeScript: fine (it's any — YOLO)

// unknown — safe, must narrow before use
const safe: unknown = JSON.parse(data);
// safe.nonexistentProp; // TS error: Object is of type 'unknown'
// Use Zod or typeof guards to narrow before accessing properties
```

💼 CUSTOMER STORY

Customer Spotlight : Aspire for Windows 365

🎤 Yongyu Chen, Chuanbo Zhang, Jisheng Xing, Maddy Montaquila

⏱ 30:43 | ▶ Watch Session

> *Customer Spotlight: Aspire for Windows 365 - Reliability, Extensibility, and Multi-Repo Rollout with AI We'll share how Windows 365 doubled Aspire adoption while improving reliability by driving E2E CloudTest success and systematically removing the top onboarding blockers, including key reliability fixes for Azure Functions and Cosmos DB. We'll cover the concrete work that made runs consistently "green," plus the repeatable onboarding patterns used to move services onto Aspire and CloudTest at*

· Summary

In the most enterprise-scale story of AspireConf 2026, the Windows 365 WCX (Windows Cloud Experience) engineering systems team — presenting via pre-recorded video from China — detailed how they turned into the

orchestration backbone for an AI-driven, multi-repository automated rollout system spanning dozens of microservices. Presented by senior engineering manager Yongyu Chen, architect Chuanbo Zhang, and developer Jisheng Xing, the talk chronicled a three-phase adoption journey from December 2025 to March 2026 that doubled Aspire adoption across the Windows 365 service fleet — growing from 15 to 39 repositories and from 200 to 600+ E2E automation test cases.

Phase 1 (Functionality) established the trusted platform baseline. The team built an internal Aspire F5 SDK — a NuGet package shipped to all Windows 365 service teams — that standardized health endpoints for web APIs and Azure Functions, provided shared CosmosDB and Azure Service Bus emulators via an 'Email Hub' shared library, and enabled the F5-to-run experience. Phase 2 (Reliability) tackled the hardest onboarding blockers: Azure Functions failing to start under `dotnet test` (fixed by standardizing .NET 9 via `global.json` and installing Azure Functions Core Tools via MSI); Azure Functions host isolation conflicts in Cloud Test parallel agents (fixed with explicit function-style startup patterns); and CosmosDB emulator instability (fixed with write-based readiness health checks). CI setup time collapsed from 30 minutes to 8 minutes.

Phase 3 (AI-driven Scale) is where the story becomes extraordinary. Unable to manually update 39+ repositories to a new quality baseline, the team built a GitHub Copilot SDK-based agent fleet running on top of Aspire as the control plane. Aspire's eventing model provided coordination primitives without a separate message broker. Four specialized agents form the worker tier: the Local Fix Agent (applies code/config fixes, opens PRs), the Cloud Fix Agent (monitors CloudTest CI results, applies additional fixes), the Learning Agent (silently observes fix patterns and updates shared skill prompts so the system improves each run), and the Godzilla Agent (safety layer validating all changes against approved boundaries and reverting out-of-scope modifications). The orchestrator uses adaptive concurrency — more parallelism when runs succeed, less when they fail.

The Aspire extensibility deep-dive was technically impressive. Custom resource types wrap containers with lazy endpoint references (resolved at launch-time for composable service discovery). Custom dashboard commands let operators pause or resume a long-running AI rollout directly from the Aspire UI. A shared `AppHostFixture` base class removes startup race conditions and boilerplate from every team's E2E tests. Most striking: a custom Aspire InteractionService extension monitors Azure Container Registry login expiry and surfaces a 'Login Now' banner in

the Aspire dashboard — replacing 20 minutes of confused log archaeology. Result: a repo that once took half a day of expert manual onboarding now receives a ready-to-merge PR with a consistent green baseline in 2-8 hours of automated agent work.

· Key Takeaways

→ Windows 365 grew from 15 to 39 Aspire-onboarded repos and 200 to 600+ E2E automation cases in 3 months (Dec 2025-Mar 2026) using a three-phase adoption strategy culminating in AI-driven multi-repo rollout

→ Aspire acts as the AI agent control plane: eventing model and coordination primitives replace a separate message broker for coordinating four specialized agents without direct agent-to-agent coupling

→ Four-agent architecture: Local Fix Agent (applies fixes, opens PRs), Cloud Fix Agent (monitors CloudTest CI), Learning Agent (silently evolves shared skill prompts), Godzilla Agent (validates safety boundaries, reverts out-of-scope changes)

→ CI setup time reduced from 30 minutes to 8 minutes: .NET 9 SDK standardization via global.json, Azure Functions Core Tools MSI install, and write-based CosmosDB readiness health checks

→ Two-tier metric/check engine: attributed classes with CheckAsync (what's wrong) and FixAsync (remediation steps for Copilot) — assembly scanning auto-registers them, adding a new check requires writing one class

→ Custom Aspire dashboard ACR login expiry banner via InteractionService: background service monitors ACR auth status and surfaces actionable 'Login Now' alert — eliminates ~20 minutes of log debugging per incident

→ Repo onboarding went from half-day manual expert effort to 2-8 hours of automated agent work producing a ready-to-merge PR with a consistent green quality baseline

· Announcements & Features

Aspire as AI Agent Orchestrator (Production at Scale) — Windows 365 production proof that Aspire's eventing model, coordination primitives, and dashboard can orchestrate multi-agent AI workflows at scale — 39 repos, 4 agent types, adaptive concurrency, LLM model fallback

Aspire F5 Internal SDK (NuGet) — Internal NuGet package shipping standardized health endpoints for web APIs and Azure Functions, shared CosmosDB/Service Bus emulators via Email Hub, service defaults, and AppHostFixture base class to all Windows 365 service teams

AppHostFixture Base Class Pattern — Standardized E2E test fixture managing full Aspire lifecycle: start distributed app, wait for all resource health checks to pass, run test suite, clean up — eliminates startup race conditions and per-team boilerplate

Aspire Dashboard ACR Login Banner (InteractionService) — Custom dashboard extension via Aspire InteractionService: background service monitors Azure Container Registry login expiry and surfaces a Login Now alert banner, auto-dismissed on auth restoration

Adaptive Concurrency Agent Orchestration — Orchestrator applies adaptive concurrency across agent pools: increases parallelism when fix passes succeed, reduces when they fail; includes LLM model fallback when primary model hits token or rate limits

· Demos & Live Code

Multi-Repo Rollout Dashboard — Centralized rollout dashboard showing status of 35 Windows 365 repos across multiple onboarding scenarios with drill-down from scenario overview to per-repo check results to specific failure reasons, plus rollout history for audit

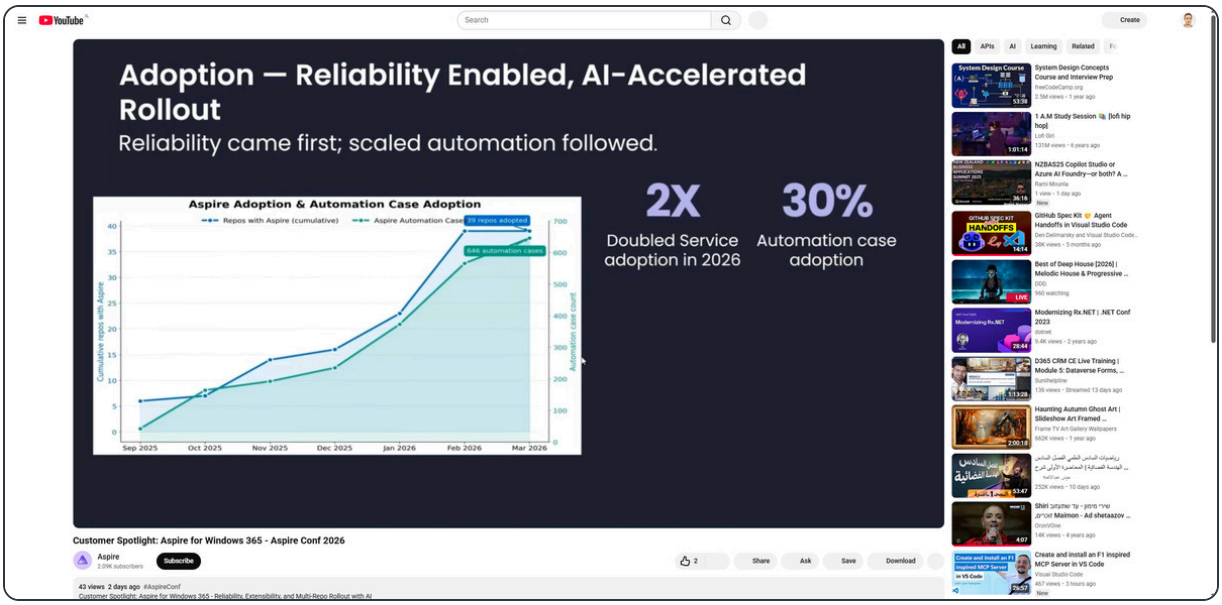
Local Fix Copilot Agent Trigger — Demo selecting failing repos and scenarios in the rollout UI, triggering the Local Fix Copilot Agent to apply fixes in parallel across all selected repositories with real-time progress tracking and automatic PR generation

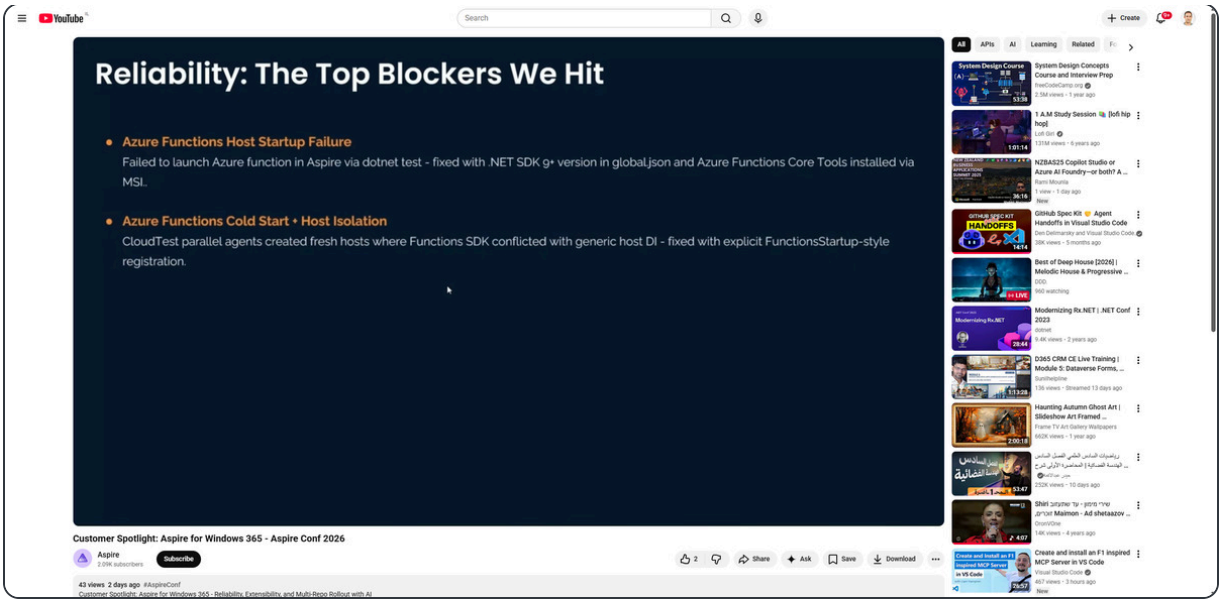
Four-Agent Architecture Walkthrough — End-to-end architecture diagram: Aspire orchestrator host, four-agent worker tier connected via MCP tools,

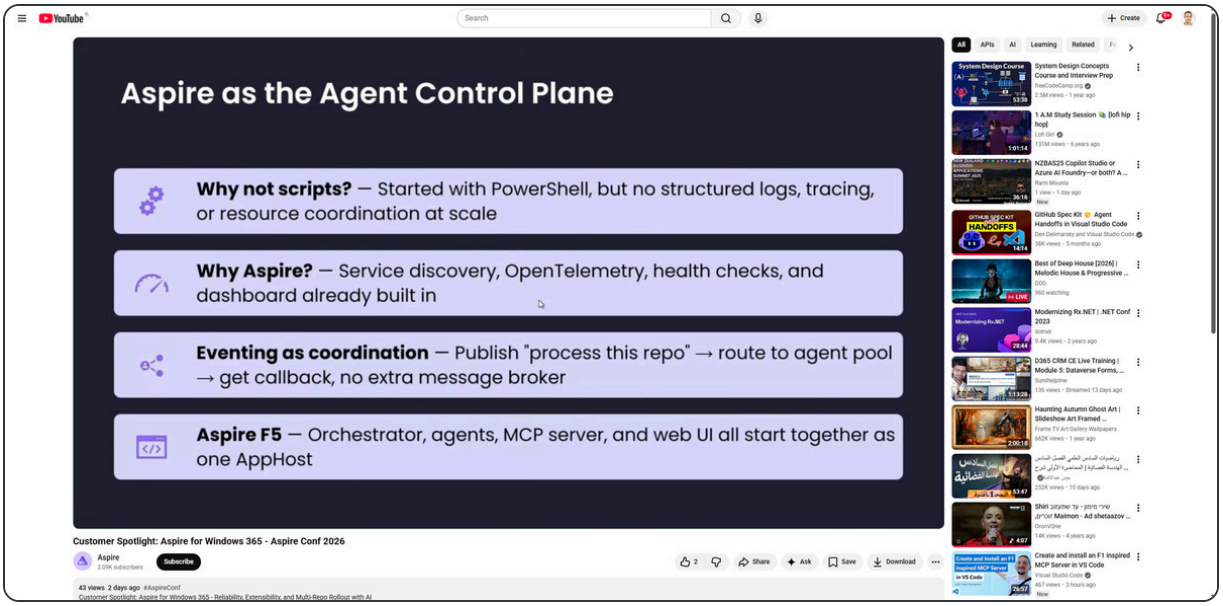
adaptive concurrency loop with LLM model fallback, learning agent updating AI skill prompts from each completed pass

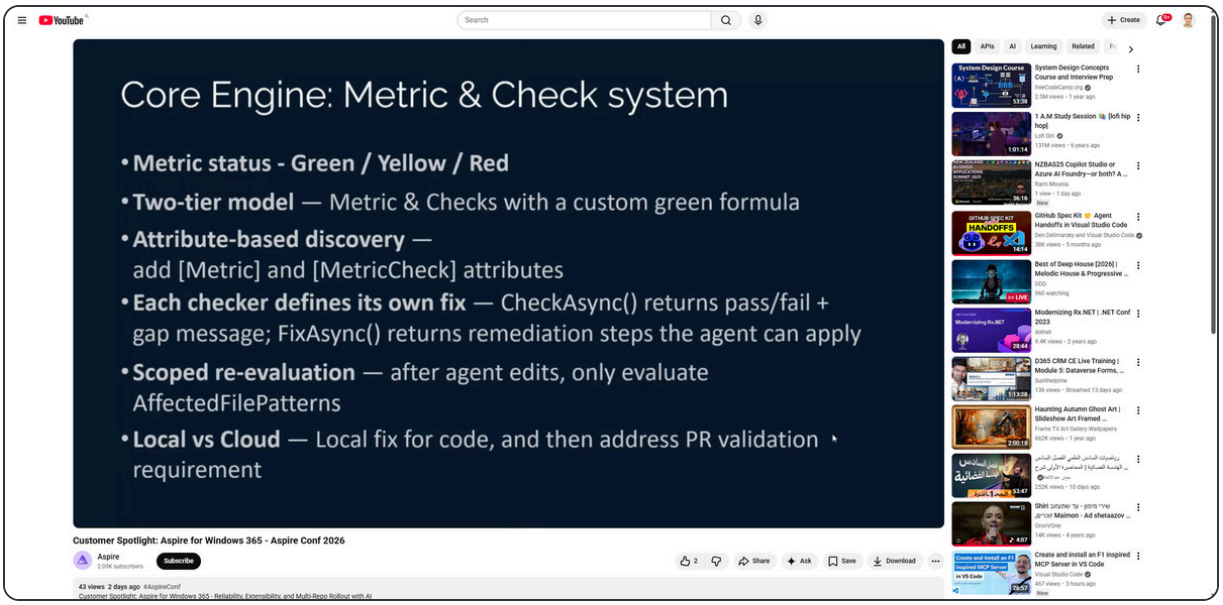
Custom MicroserviceResource Type — Code walkthrough of a custom Aspire resource type wrapping container resources with lazy endpoint reference (ReferenceExpression) enabling composable service discovery — single AddMicroservice() call registers container, endpoint, OTel, and health check

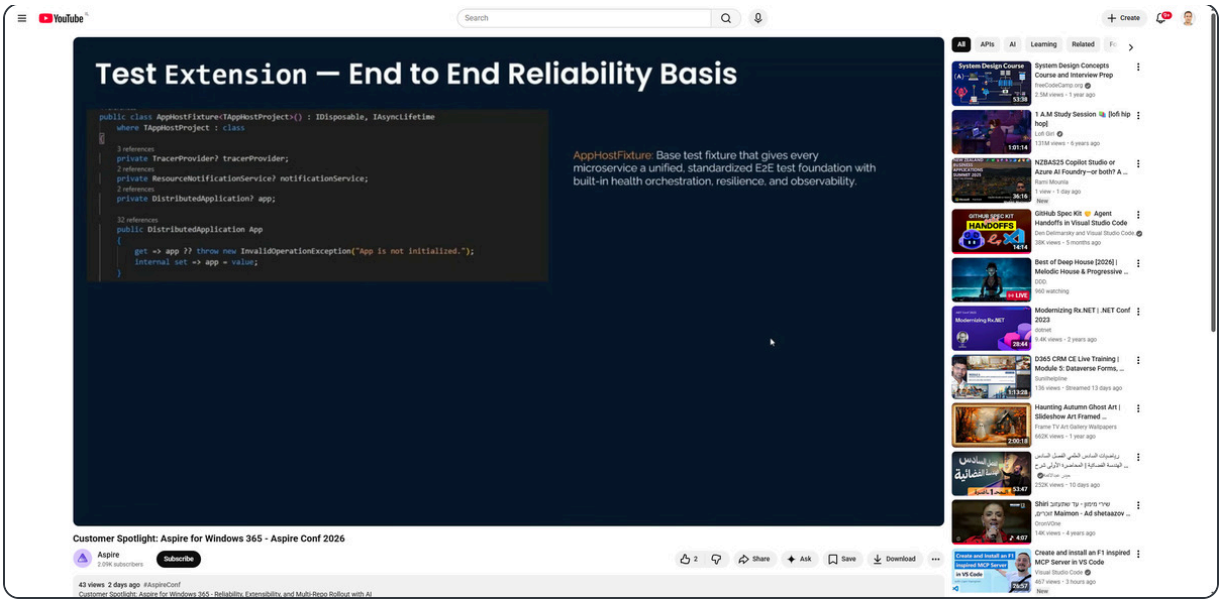
ACR Login Expiry Dashboard Banner — Demo of Aspire dashboard showing expired ACR authentication alert banner with Login Now button, automatically dismissed when the InteractionService background monitor detects restored authentication




Reliability: The Top Blockers We Hit
Azure Functions Host Startup Failure
Failed to launch Azure function in Aspire via dotnet test - fixed with .NET SDK 9+ version in global.json and Azure Functions Core Tools installed via MSI.
Azure Functions Cold Start + Host Isolation
CloudTest parallel agents created fresh hosts where Functions SDK conflicted with generic host DI - fixed with explicit FunctionsStartup-style registration.
Customer Spotlight: Aspire for Windows 365 - Aspire Conf 2026


Aspire as the Agent Control Plane
Why not scripts? — Started with PowerShell, but no structured logs, tracing, or resource coordination at scale
Why Aspire? — Service discovery, OpenTelemetry, health checks, and dashboard already built in
Eventing as coordination — Publish "process this repo" → route to agent pool → get callback, no extra message broker
Aspire F5 — Orchestrator, agents, MCP server, and web UI all start together as one AppHost
Customer Spotlight: Aspire for Windows 365 - Aspire Conf 2026


Core Engine: Metric & Check system
Metric status - Green / Yellow / Red
Two-tier model — Metric & Checks with a custom green formula
Attribute-based discovery — add [Metric] and [MetricCheck] attributes
Each checker defines its own fix — CheckAsync() returns pass/fail + gap message; FixAsync() returns remediation steps the agent can apply
Scoped re-evaluation — after agent edits, only evaluate AffectedFilePatterns
Local vs Cloud — Local fix for code, and then address PR validation requirement
Customer Spotlight: Aspire for Windows 365 - Aspire Conf 2026



· Code Samples

Custom MicroserviceResource with lazy endpoint reference for composable Windows 365 service discovery

```
public class MicroserviceResource(string name) : ContainerResource(name) {

public static IResourceBuilder AddMicroservice(
    this IDistributedApplicationBuilder builder,
    string name, string image)
{
    // One call: container + endpoint + OTel + health check
    return builder.AddResource(new MicroserviceResource(name))
        .WithImage(image)
        .WithEndpoint(8080, scheme: "http")
        .WithOpenTelemetry()
        .WithHealthCheck();
}

// Lazy endpoint ref — resolved at launch-time, enabling composable service
public ReferenceExpression GetServiceUrl(string endpointName)
    ⇒ ReferenceExpression.Create($"{this.GetEndpoint(endpointName)}");
```

AppHostFixture base class for standardized Windows 365 E2E testing

```
public abstract class AppHostFixture : IAsyncLifetime
{
    protected DistributedApplication? App { get; private set; }

    public async Task InitializeAsync()
    {
        var appHost = await DistributedApplicationTestingBuilder
            .CreateAsync();
        App = await appHost.BuildAsync();
        await App.StartAsync();
        // Wait until ALL declared resources pass health checks
        await App.WaitForAllResourcesHealthyAsync();
    }

    public async Task DisposeAsync() => await App!.StopAsync();
}
```

Two-tier metric/check engine: attributed class auto-discovered via assembly scanning

```
[Metric("aspire-onboarding")]
public class AppHostContainsSubCheck : IMetricCheck
{
    // CheckAsync: what's wrong?
    public Task CheckAsync(RepoContext ctx)
        => Task.FromResult(ctx.HasAppHostProject
            ? CheckResult.Pass()
            : CheckResult.Fail("No AppHost project found"));

    // FixAsync: remediation steps Copilot will execute
    public Task FixAsync(RepoContext ctx)
        => Task.FromResult(RemediationSteps.ForFile(
            "AppHost.csproj",
            "Add shared infra ProjectReference"));
}
```

ACR login expiry banner via Aspire InteractionService as a hosted background service

```
// Register in AppHost
builder.Services.AddHostedService();

class AcrLoginMonitor(IInteractionService interactions) : BackgroundService
{
    protected override async Task ExecuteAsync(CancellationToken ct)
    {
        while (!ct.IsCancellationRequested)
        {
            if (!await IsAcrLoginValidAsync())
                await interactions.ShowAlertAsync(
                    "ACR login expired",
                    confirmText: "Login Now",
                    onConfirm: RunAcrLoginAsync);
            await Task.Delay(TimeSpan.FromMinutes(5), ct);
        }
    }
}
```

DEPLOYMENT

Aspire Escapes the Inner Loop and Does Deployment

🎤 Mitch Denny, Maddy Montaquila, David Fowler

⏱ 35:37 | ▶ Watch Session

> *Everyone knows that Aspire makes your development inner loop awesome, but did you know that it can also be used to streamline your deployments - all the way to production! Recent releases of Aspire have significantly improved Aspire's end-to-end deployment capabilities and now is a great time to get across how they work and how it can be adapted to suit your specific environment. This talk is from #AspireConf on March 23rd, 2026. 🎤 Speaker: Mitch Denny 🎬 Watch the other sessions from #Aspire*

· Summary

In one of AspireConf 2026's most technically dense sessions, Mitch Denny — calling in from Melbourne, Australia — delivered a comprehensive tour of Aspire's deployment story, tracing its evolution from a static manifest-file era all the way

through to a first-class, code-centric pipeline model. Mitch framed the history in three distinct 'eras': the Aspire 8.0 era where a JSON manifest file described resources abstractly for external tools like AZD; the Aspire 9.2 era that previewed the Aspire CLI with early deployment intent; and now the 'pipeline era' arriving with Aspire 13.0 and 13.2, where the AppHost model drives deployments directly with full fidelity. The key insight: by eliminating the 'lossy round-trip' through a static manifest, Aspire can now carry the full richness of the application topology all the way to production.

Mitch demonstrated the new deployment model live. Starting with a simple multi-user text-based web application, he added a Docker Compose environment with a single call — `builder.AddDockerComposeEnvironment()` — and then ran `aspire deploy` from the terminal. Aspire launched the AppHost, analyzed the resource graph, and emitted a Docker Compose file, all in seconds. He also showed how developers can customize generated compute resources using callback APIs such as `PublishAsDockerComposeService()`, which exposes a rich object model to tweak things like Linux capabilities (`NET_ADMIN`), custom network configs, and more. The exact same pattern applies to the Kubernetes environment via `PublishAsKubernetesService()`, complete with persistent volume claims and custom Kubernetes objects.

The Kubernetes demo was a showstopper: Mitch deployed his app to an eight-node Raspberry Pi cluster running in his garage, cross-compiling for ARM64 and generating a Helm chart via `aspire publish`. He pointed out that the Aspire team now automatically deploys to a Kubernetes cluster on every PR, treating it as a real deployment target in their own CI pipeline. The session also covered Azure targets: Azure Container Apps (the most mature, now with V-Net support and scheduled jobs), Azure App Service, and the brand-new Azure Virtual Network integration arriving in Aspire 13.2.

The V-Net feature — contributed primarily by 'Eric' on the Aspire team — was called out as 'bananas' by Maddy Montaquila. Using `builder.AddAzureVirtualNetwork()`, developers can define CIDRs, subnets, and network security groups (NSGs) inline in C#, allowing Azure Container Apps to allocate IPs within corporate-controlled virtual networks — a long-requested enterprise feature that previously required hand-crafted Bicep. Mitch capped the session by explaining the underlying pipeline steps primitive: every environment integration (Docker, Kubernetes, Azure) registers a graph of steps, and `aspire deploy` executes them in order, building a dependency

graph automatically. Mitch ran long — earning him the 'going over time' Aspire Bingo square — and promised a dedicated deep-dive stream on pipeline steps.

· Key Takeaways

→ Aspire 13.2 GAs the Docker Compose hosting integration, enabling one-line deployment with `builder.AddDockerComposeEnvironment()`

→ Aspire now generates Helm charts for Kubernetes via `aspire publish`, with support for persistent volume claims and custom Kubernetes objects

→ New Azure Virtual Network (V-Net) support in 13.2: define VPCs, subnets, and NSGs in C# for Azure Container Apps deployments

→ Pipeline steps are the underlying primitive powering the Aspire CLI deploy command — every integration registers a graph of steps

→ Multiple deployment environments can coexist in a single AppHost, enabling simultaneous multi-target deployments

→ Aspire now CI/CD-tests Kubernetes deployment on every PR to ensure end-to-end deployment quality

→ Azure Container Apps job/scheduled job support was added in Aspire 13.0

→ The deployment model shifted from 'manifest + external tool' to 'AppHost-driven pipeline' for full-fidelity, no-rewrite deployments

· Announcements & Features

Aspire Hosting Docker Integration GA — The Docker Compose hosting integration (`Aspire.Hosting.Docker`) is officially GA in Aspire 13.2, enabling `builder.AddDockerComposeEnvironment()` for direct compose-based deployment.

Azure Virtual Network Support — Aspire 13.2 ships Azure VNet integration: define virtual networks, subnets with CIDR notation, and NSGs directly in C# AppHost code, wiring up ACA container environments to corporate networks.

Pipeline Steps API — A new lower-level 'pipeline steps' API powers the Aspire CLI. Steps form a dependency graph; `aspire deploy` builds and executes that

graph. Integrations register their own steps, making the system extensible.

Kubernetes CI Integration — The Aspire team now runs end-to-end Kubernetes deployment tests on every PR, indicating Kubernetes is becoming a first-class target.

· Demos & Live Code

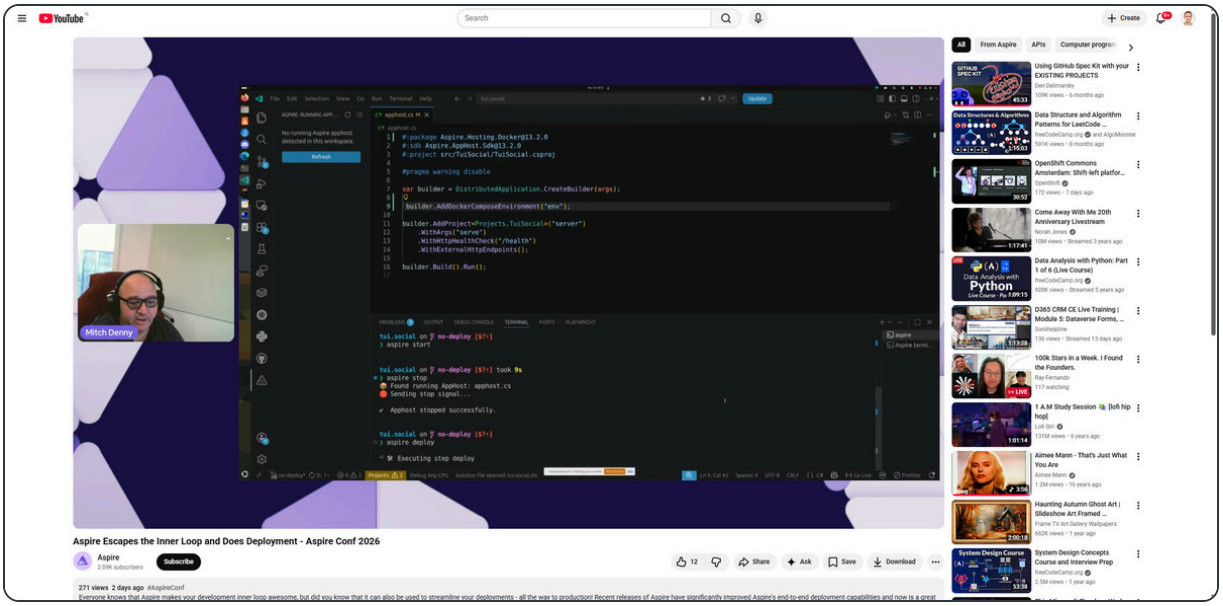
Docker Compose one-line deployment — Mitch added `builder.AddDockerComposeEnvironment()` to a simple multi-user web app and ran `aspire deploy`, which generated a Docker Compose file and deployed it in seconds.

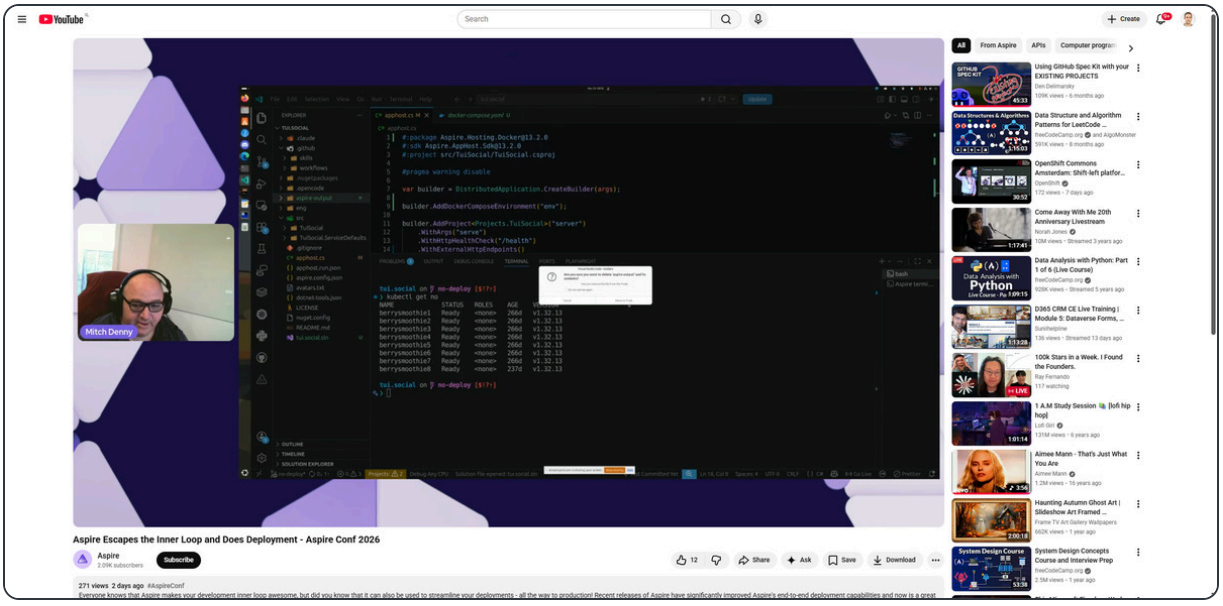
Kubernetes deployment to home Raspberry Pi cluster — Switched branch to Kubernetes config, ran `aspire publish` to generate a Helm chart, built the ARM64 container image, pushed to local cluster registry, and deployed via `helm install` to an 8-node Pi cluster.

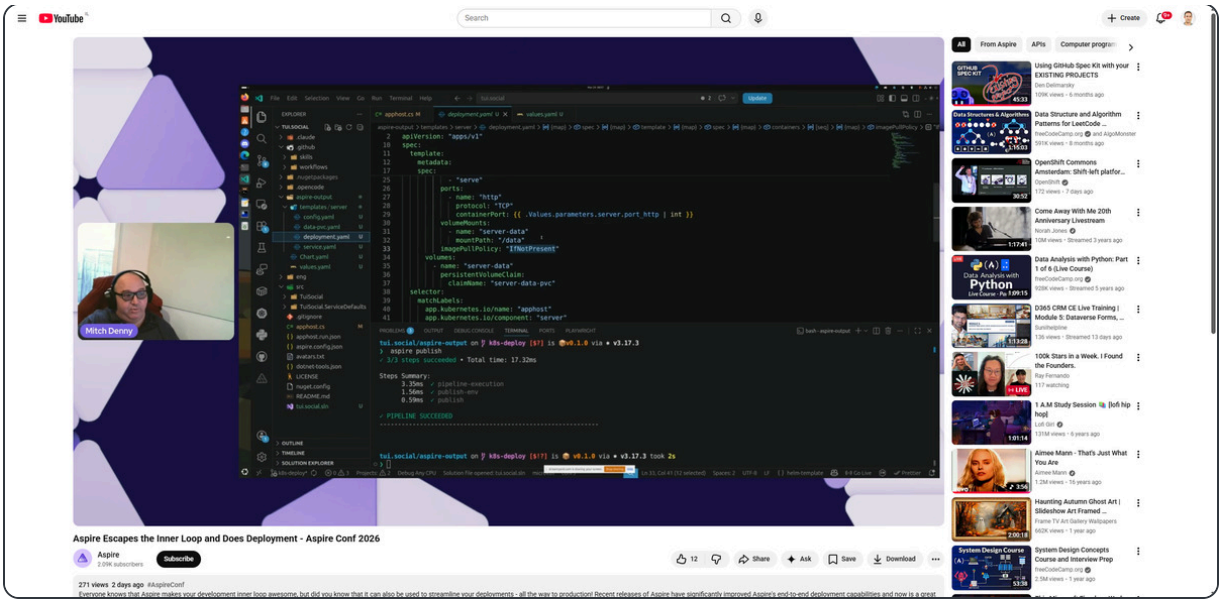
Azure Virtual Network configuration — Added `builder.AddAzureVirtualNetwork('vnet', cidr: '10.100.0.0/16')`, created a subnet, attached it as a delegated subnet to an ACA environment, and added an NSG to allow only load-balancer traffic on port 443.

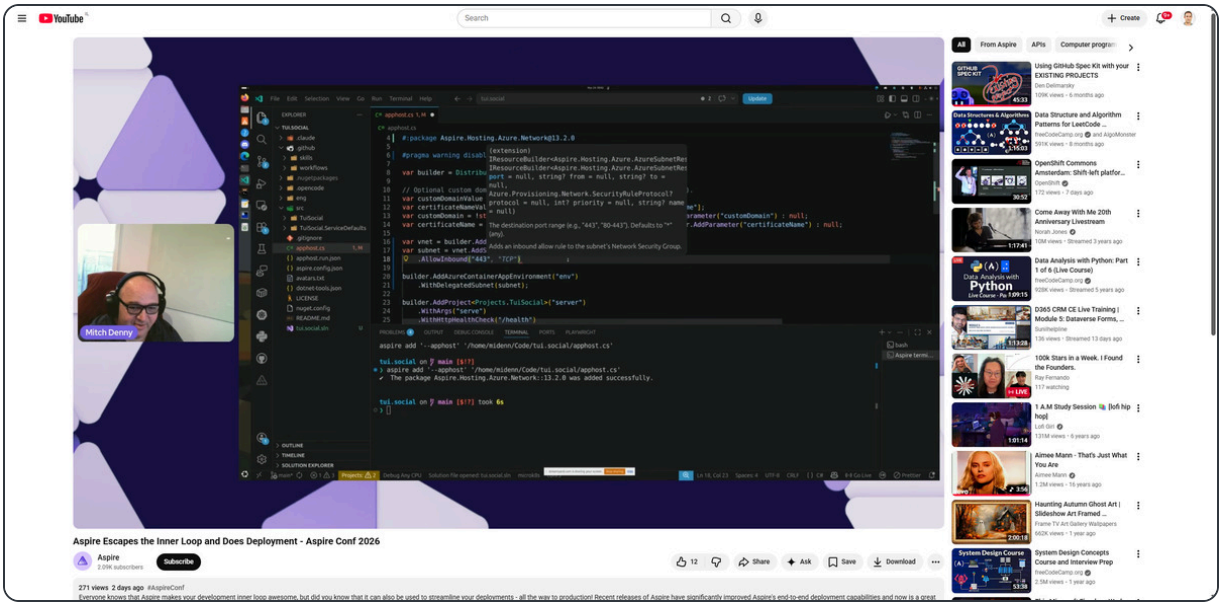
Persistent Volume Claims in Kubernetes — Used `PublishAsKubernetesService()` callback to define a PersistentVolumeClaim with storage limits, volume mounts, and a specific storage class for a distributed block store.


The Eras of Aspire Deployment
Aspire 8.0
Aspire 9.2
Aspire 13.2
Aspire 9.0
Aspire 13.0
The Manifest Era
Mitch Denny
Aspire Escapes the Inner Loop and Does Deployment - Aspire Conf 2026


Mitch Denny
Aspire Escapes the Inner Loop and Does Deployment - Aspire Conf 2026


Mitch Denny
Aspire Escapes the Inner Loop and Does Deployment - Aspire Conf 2026





· Code Samples

Add a Docker Compose deployment environment (one line)

```
builder.AddDockerComposeEnvironment("compose");
```

Customize the generated Docker Compose service (add Linux capability)

```
myProject.PublishAsDockerComposeService((service) => {
    service.CapAdd.Add("NET_ADMIN");
});
```

Add a Kubernetes deployment environment

```
builder.AddKubernetesEnvironment("k8s", cluster: myCluster);
```

Customize Kubernetes service with persistent volume claims

```
myProject.PublishAsKubernetesService((resource) => {
    var pvc = new PersistentVolumeClaim("storage") {
        StorageLimit = "10Gi"
    };
    resource.AdditionalResources.Add(pvc);
    resource.Volumes.Add(new VolumeMount { ClaimName = "storage", MountPath
});
```

Add an Azure Virtual Network with subnet and NSG

```
var vnet = builder.AddAzureVirtualNetwork("vnet", "10.100.0.0/16");
var subnet = vnet.AddSubnet("app-subnet", "10.100.1.0/24");
subnet.AddNetworkSecurityGroup(nsg => {
    nsg.AllowInbound(port: 443, source: "AzureLoadBalancer");
});
myAcaEnv.WithDelegatedSubnet(subnet);
```

Build a specific container image as a pipeline step

```
aspire do build-{resource-name}
```

Deploy via Aspire CLI

```
aspire deploy
# or just publish (output artifacts only)
aspire publish
```

DEPLOYMENT

Building and Deploying with Aspire and AWS

🎤 Norm Johanson, Maddy Montaquila

⏱ 30:32 | ▶ Watch Session

> *Learn how to use Aspire with Amazon Web Services to streamline both local development and cloud deployment. This session demonstrates the new support for running and debugging AWS Lambda functions locally within Aspire, enabling a fast inner development loop. Then see how Aspire applications can be deployed to AWS by combining Aspire's orchestration model with AWS Cloud Development Kit (CDK). This talk is from #AspireConf on March 23rd, 2026. 🎤 Speaker: Norm Johanson 🎬 Watch the other sessio*

· Summary

AWS Principal Engineer Norm Johanson brought the cloud giant's perspective to AspireConf 2026, showcasing the rapidly maturing AWS-Aspire integration and delivering one of the conference's most jaw-dropping moments: a live, unscripted

deployment of a full-stack application to AWS that completed successfully on stream. Norm opened by surveying the current AWS feature set for Aspire: automatic provisioning of AWS resources (S3 buckets, DynamoDB tables) via CloudFormation from within the AppHost; DynamoDB Local as an emulator for local development; the newly GA'd Lambda support for local debugging; and the preview deployment/publish pipeline using the AWS CDK.

The Lambda section was a highlight on its own. Norm demonstrated how to register multiple Lambda functions in the AppHost, launch them alongside an API Gateway emulator (a .NET CLI tool installed on demand), and then step-through-debug them from Visual Studio — all without deploying anything to AWS. The emulator surfaces in the Aspire dashboard as a resource, shows event queues, and allows replay. With DynamoDB Local also wired up, developers get a full serverless development loop that is entirely local and observable via OpenTelemetry in the Aspire dashboard. This Lambda integration just reached GA at the time of the conference.

The deployment story centers on a 'marriage' between Aspire and the AWS CDK (Cloud Development Kit), AWS's infrastructure-as-code framework written natively in TypeScript with projections to .NET and other languages. Adding a single NuGet package (`Aspire.Hosting.AWS`) and calling `builder.AddAWSCDKEnvironment()` is all that's needed to get started. Aspire's publish pipeline then runs, building containers and emitting a `cdk-out` folder containing a CloudFormation template, container tar files, and metadata. Running `cdk deploy` against that folder handles the rest. Norm showed how Aspire automatically maps Aspire resource types to appropriate CDK constructs: Redis → Amazon ElastiCache Serverless, .NET web projects → ECS Fargate Express services. Critically, Aspire also wires up all service discovery environment variables and configures the required VPCs, subnets, route tables, and security groups so the services can communicate.

For customization, developers can subclass the CDK Stack class and add their own constructs — the Aspire integration will attach all its generated constructs to that custom stack instead of an empty one. Norm also demonstrated full lifecycle management: because everything is in a single named CloudFormation stack, tearing down the entire deployment is as simple as deleting the stack. The live demo showed the Aspire starter app (Redis cache + .NET backend API + Blazor frontend) deployed to AWS in real time, with environment variables for service discovery wired automatically, and the frontend talking to the backend through ElastiCache. Maddy

Montaquila's reaction — 'You deployed code you wrote for deploying code. That's crazy.' — summed up the moment perfectly.

· Key Takeaways

→ AWS Lambda support for Aspire has reached GA: register Lambda functions in AppHost, debug them locally with an API Gateway emulator from Visual Studio

→ AWS deployment integrates Aspire with the AWS CDK: `builder.AddAWSCDKEnvironment()` maps Aspire resources to CDK constructs (Redis → ElastiCache, .NET projects → ECS Fargate)

→ Live on-stream deployment of a full-stack Aspire app (Redis + API + Blazor) to AWS succeeded without cuts

→ Aspire automatically provisions VPCs, subnets, route tables, and security groups for AWS deployments

→ All AWS features live in a single NuGet package: `Aspire.Hosting.AWS`

→ DynamoDB Local emulator integration allows offline NoSQL development mirroring production DynamoDB

→ Developers can subclass the CDK Stack class to inject custom AWS resources alongside Aspire-managed ones

→ A dedicated AWS channel exists on the Aspire Discord server for community feedback

· Announcements & Features

Lambda Support GA — AWS Lambda support for Aspire has reached general availability. Developers can register Lambda functions in the AppHost, run them alongside an API Gateway emulator, and debug them step-by-step in Visual Studio with full OTel visibility.

AWS CDK Deployment (Preview) — Deploy Aspire apps to AWS using the CDK integration (preview). `builder.AddAWSCDKEnvironment()` triggers a publish pipeline that generates a CloudFormation template and container artifacts deployable with `cdk deploy`.

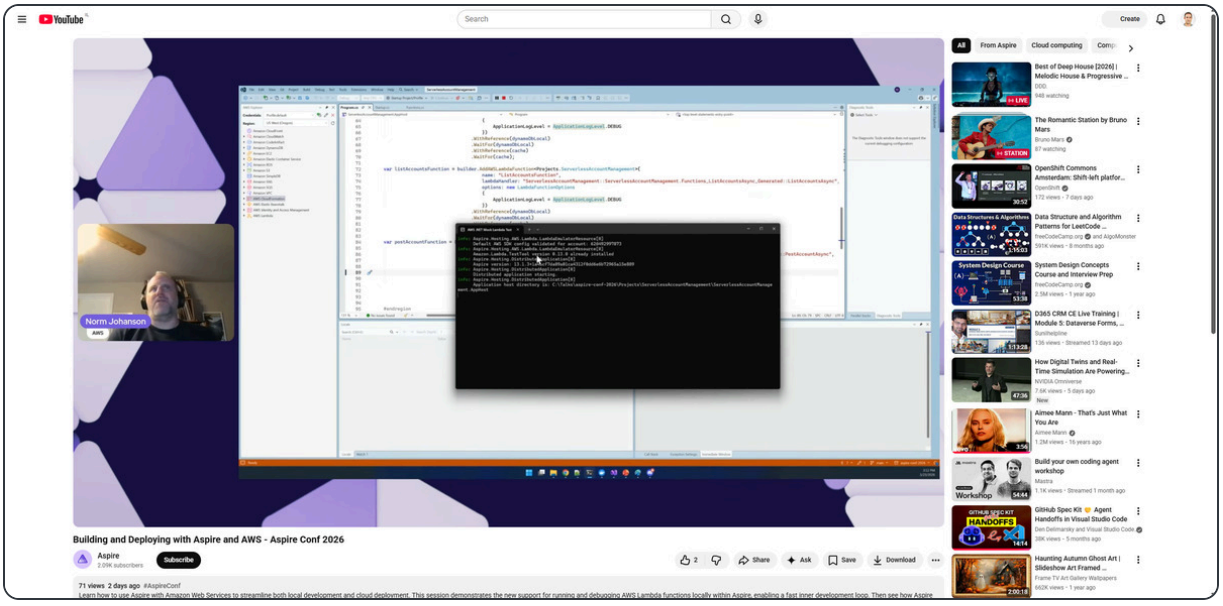
Aspire.Hosting.AWS NuGet Package — All AWS hosting features (S3, DynamoDB, Lambda, CDK deployment) are consolidated in the single `Aspire.Hosting.AWS` NuGet package.

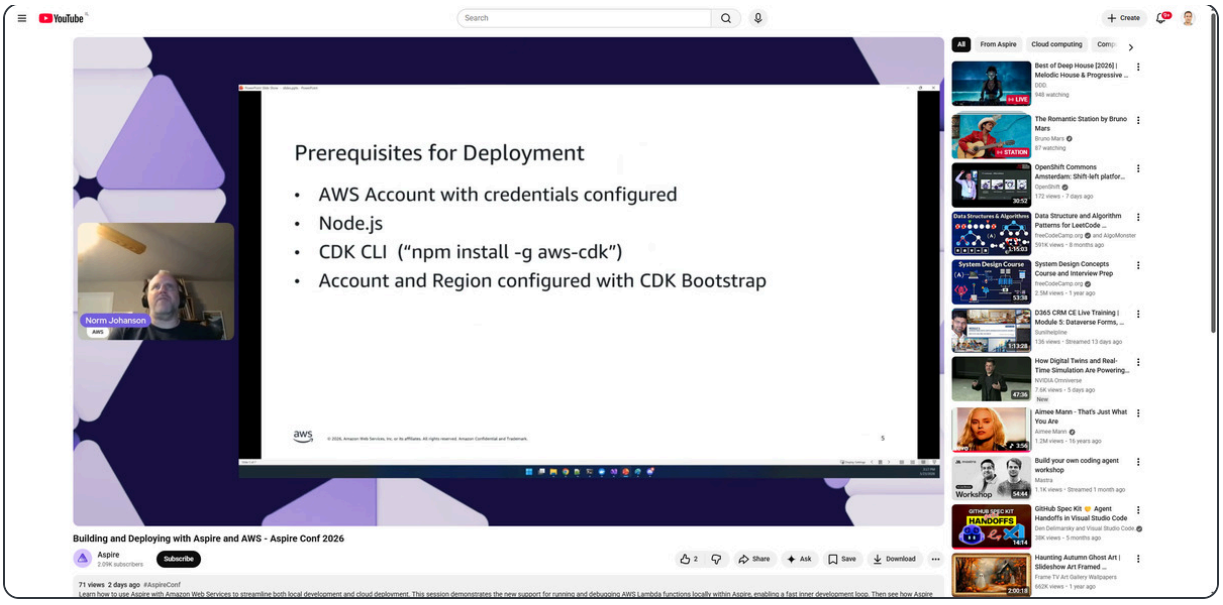
· Demos & Live Code

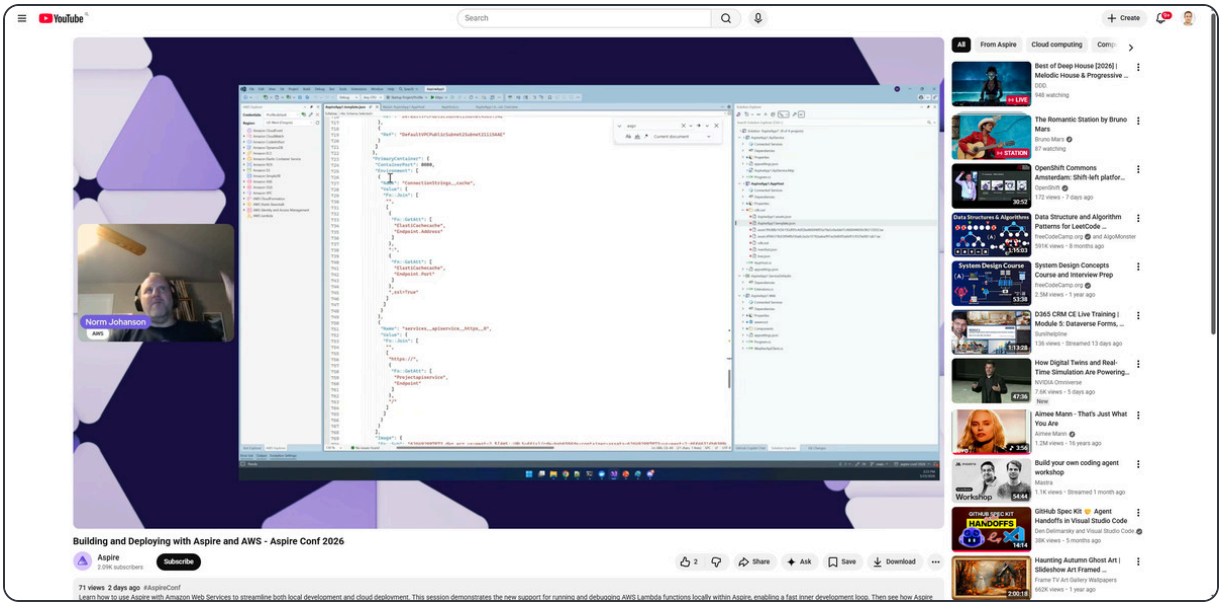
Lambda local debugging with API Gateway emulator — Registered four Lambda CRUD functions in the AppHost with a DynamoDB Local emulator and API Gateway emulator. Set a breakpoint and stepped through a Lambda function invocation in Visual Studio via the REST API endpoint exposed by the emulator.

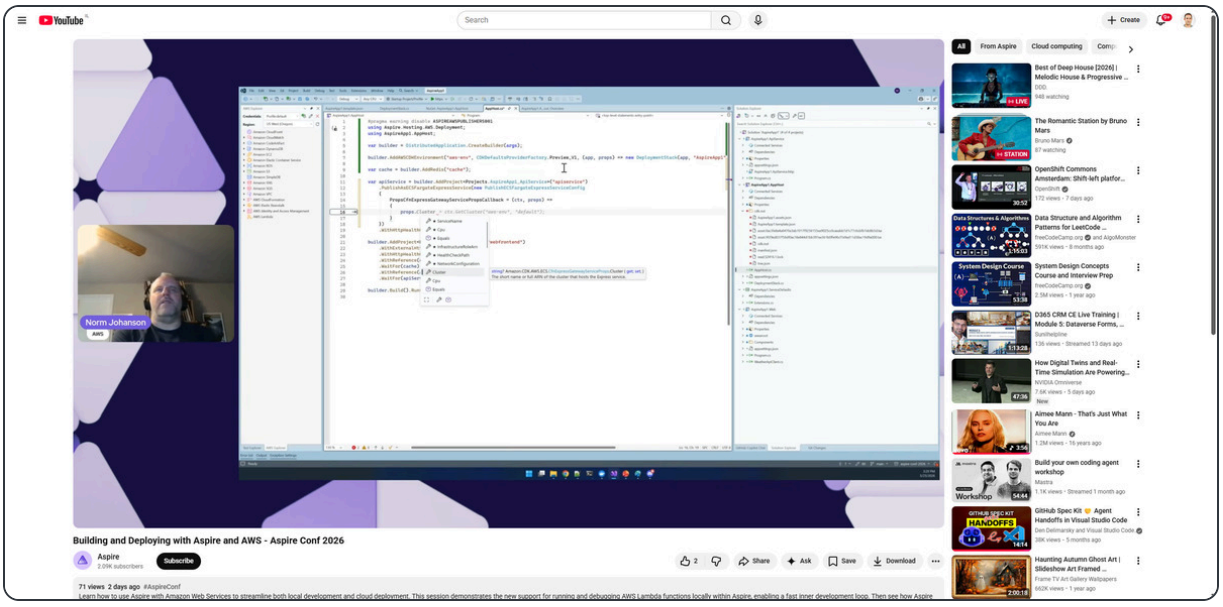
AWS CDK deployment of Aspire starter app — Added `Aspire.Hosting.AWS` to the Aspire starter app, called `builder.AddAWSCDKEnvironment()`, ran `aspire publish` to generate CDK artifacts, then deployed live to AWS. Redis mapped to ElastiCache serverless, web projects to ECS Fargate. VPC and security groups created automatically.

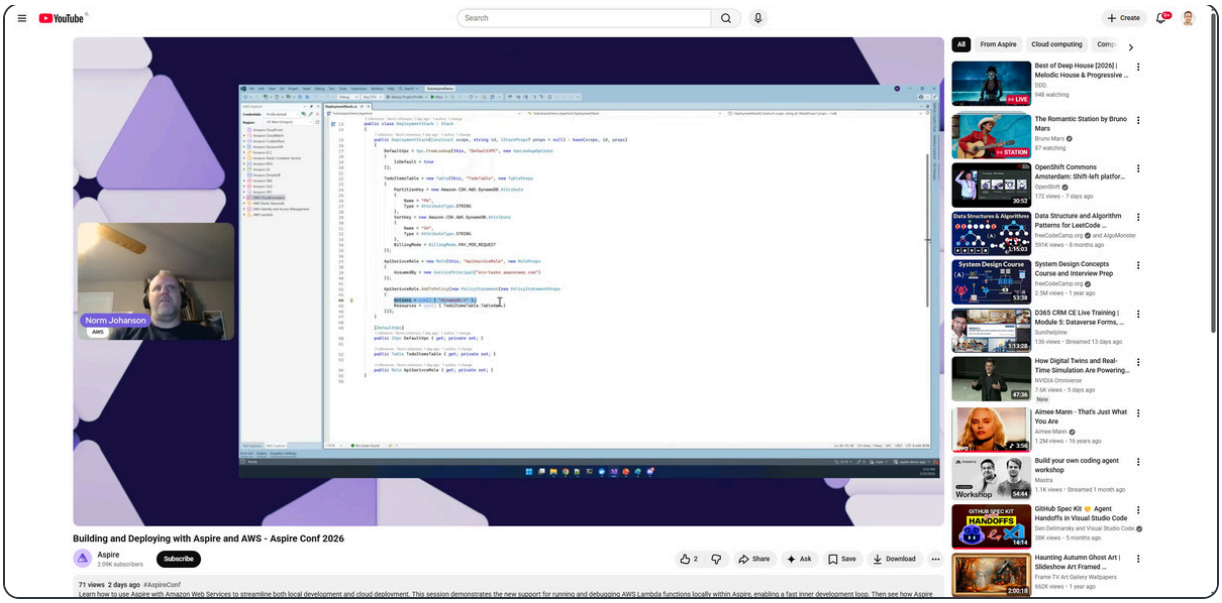
Custom CDK stack with additional AWS resources — Showed how to subclass the CDK Stack class to add custom AWS constructs (e.g., custom S3 bucket or table) and pass that stack to the Aspire CDK environment so custom resources deploy alongside Aspire-managed ones.




Prerequisites for Deployment
• AWS Account with credentials configured
• Node.js
• CDK CLI ("npm install -g aws-cdk")
• Account and Region configured with CDK Bootstrap
Norm Johanson
Building and Deploying with Aspire and AWS - Aspire Conf 2026
Aspire
Subscribe


Norm Johanson
Building and Deploying with Aspire and AWS - Aspire Conf 2026
Aspire
Subscribe


Norm Johanson
Building and Deploying with Aspire and AWS - Aspire Conf 2026
Aspire
Subscribe



· Code Samples

Add AWS CDK deployment environment in AppHost

```
// Add NuGet: Aspire.Hosting.AWS
#pragma warning disable ASPIREHOSTINGAWS001
builder.AddAWSCDKEnvironment("aspire-resources",
    stackName: "my-app-stack");
```

Register Lambda functions in AppHost

```
var lambdaFunctions = builder.AddAWSLambdaFunctions("account-functions")
    .WithReference(dynamoDbLocal)
    .WithReference(redisCache);

builder.AddAWSAPIGatewayEmulator("api-gateway", port: 5050)
    .WithReference(lambdaFunctions);
```

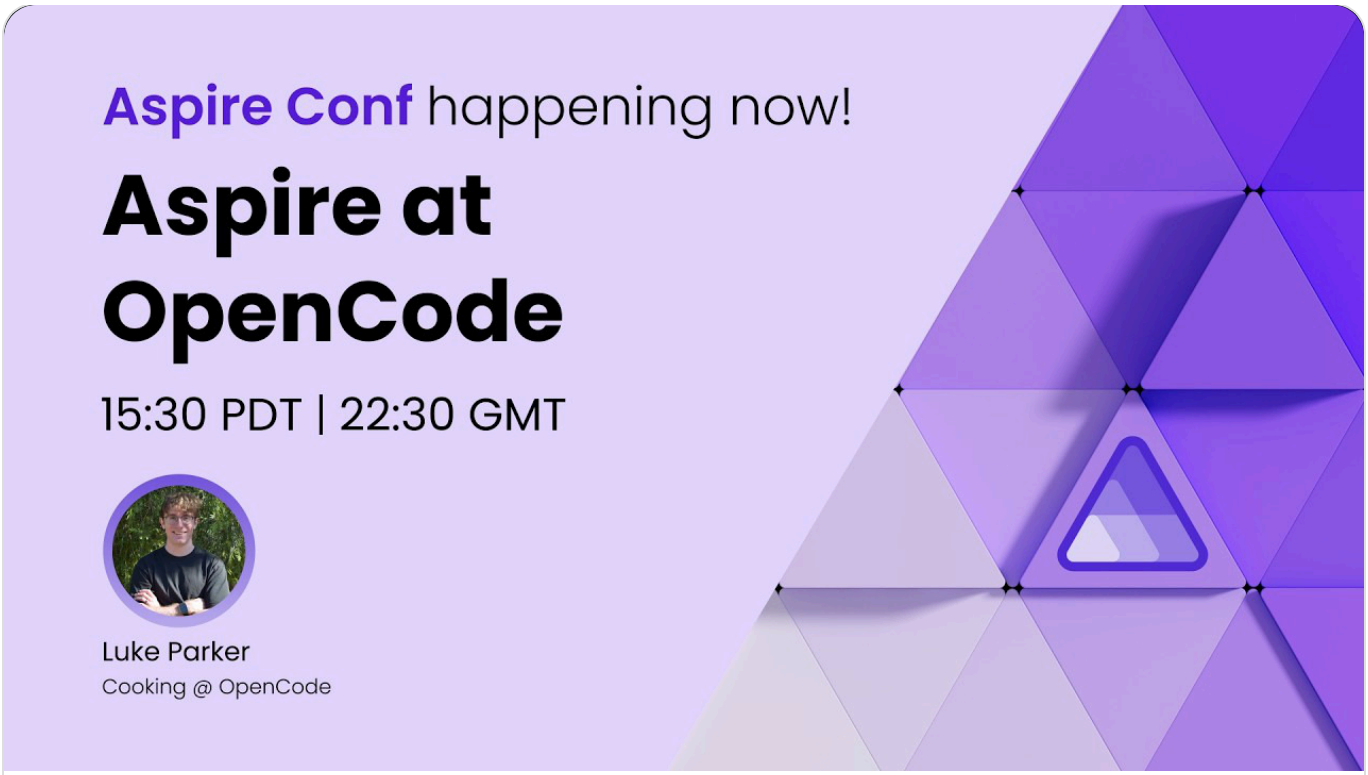
Custom CDK Stack subclass to extend Aspire deployment

```
public class MyAppStack : Stack
{
    public MyAppStack(Construct scope, string id, IStackProps props)
        : base(scope, id, props)
```

```
    {
        var bucket = new Bucket(this, "MyBucket", new BucketProps {
            Versioned = true
        });
    }
}
```

Deploy to AWS using CDK CLI after aspire publish

```
aspire publish
# Then deploy the generated artifacts:
cdk deploy --app cdk-out/
```



AI & AGENTS

Aspire at OpenCode

🎤 Luke Parker, Maddy Montaquila

⏱ 33:10 | ▶ Watch Session

> *OpenCode and Aspire make a strong pair for practical, observable agent workflows. In this demo-driven session, I'll show how Aspire gives both me and an OpenCode agent access to the same OpenTelemetry data, so we can inspect traces, metrics, and runtime behavior from the same human-readable view instead of relying on hidden magic. You'll learn how to use Aspire as a local observability surface for agent-assisted development, how to ground an agent in the same evidence a human would use, and why*

· Summary

Luke Parker from OpenCode brought one of AspireConf 2026's most thought-provoking sessions: not a tour of new Aspire features, but a live demonstration of using Aspire's standalone dashboard as the shared observability surface for both human developers and AI coding agents. OpenCode is an agentic CLI harness — it calls LLMs, reads disk files, runs bash commands, makes HTTP requests, and

touches databases, all to fulfill a developer's intent. Luke's central thesis was radical in its simplicity: if you give an AI agent and its human operator access to the same OpenTelemetry data — the same traces, spans, logs, and GenAI telemetry — the agent produces far more trustworthy and targeted fixes than when it's left to guess or 'vibe'.

Luke revealed that OpenCode is itself a distributed application: a CLI frontend handling UI rendering and input, and a separate worker process handling the bulk of the LLM loop through an event-driven API. He wired the entire system up to the Aspire standalone dashboard — running it from source, not even using the Docker image — by simply setting the OpenTelemetry endpoint URL in the application's environment. What followed was a live tour of OpenCode's internals rendered as spans and traces in the Aspire dashboard. Developers could see system prompts, user messages, tool definitions (the full bash tool definition, inputs, and outputs), LLM model selection (GPT-5 Nano for the title generator, 'Kimmy' for the main chat), streaming chunks, database calls, and the exact timing breakdown of every subsystem. The GenAI-specific trace filter in the dashboard — a dedicated dropdown Luke called 'the magic sparkles button' — made it trivially easy to isolate AI inference spans from the noise.

The session culminated in a live experiment: Luke asked OpenCode to analyze a 3-second `bash` span that should have taken under 500ms, pasting a screenshot of the trace into the chat. The agent — grounded in the actual trace data, span IDs, resource names, and structured logs — correctly identified that a plugin was thread-sleeping for 3 seconds before executing a shell command. Luke also showed the Aspire MCP integration (`aspire list traces`) being called by OpenCode to programmatically query the dashboard's data, creating a feedback loop where the agent inspects its own telemetry to self-diagnose. He made a sharp comparison of model behaviors: Claude Opus tends to jump straight to code changes; GPT-5.2 reads 50,000 files and thinks forever — but both perform better when grounded in concrete OTel data rather than guessing.

Luke's closing message resonated across the entire conference: Aspire's standalone dashboard is language-agnostic — any app that emits OpenTelemetry data (the OTEL standard) can plug in with a single URL, in any language. He urged developers to run the standalone dashboard as a first step even before adopting the AppHost, treating it as the best local OpenTelemetry viewer available. The session was praised as the only standalone dashboard demo of the day and received a standing ovation in chat.

· Key Takeaways

→ OpenCode is itself a distributed system: CLI frontend + worker backend, both observable via Aspire standalone dashboard with one OTel URL

→ Aspire standalone dashboard runs as a single Docker image and ingests OTel from any language/framework — no AppHost required

→ The GenAI filter ('magic sparkles button') in the Aspire dashboard lets developers isolate LLM inference spans with one click

→ System prompts, tool definitions, model selections, streaming chunks, and token costs are all visible in the dashboard's GenAI spans

→ Aspire MCP integration enables AI agents to programmatically query the dashboard (`aspire list traces`) for self-directed diagnostics

→ Agents perform more accurately when grounded in the same OTel trace data the developer sees — prevents 'vibing' and false optimizations

→ The Aspire CLI in 13.2 supports querying hotel (OTel) logs directly from the command line

→ Luke's key pattern: screenshot a trace and paste it into OpenCode — the agent can use visual + structured data for superior context

· Announcements & Features

Aspire Standalone Dashboard as Universal OTel Viewer — The Aspire standalone dashboard can be run as a single Docker image against any application emitting OpenTelemetry data — regardless of language, framework, or whether Aspire AppHost is used. Luke highlighted this as the best local OTel viewer available.

Aspire MCP Server for Agent Integration — The Aspire MCP server exposes dashboard data (traces, logs, resources) to AI agents via the MCP protocol. Agents can call `aspire list traces` to query live telemetry and self-diagnose issues.

Aspire CLI OTel Query Support (13.2) — The Aspire CLI in 13.2 adds support for querying OpenTelemetry data (logs, traces) from the command line, enabling

scripted access to dashboard data without the browser UI.

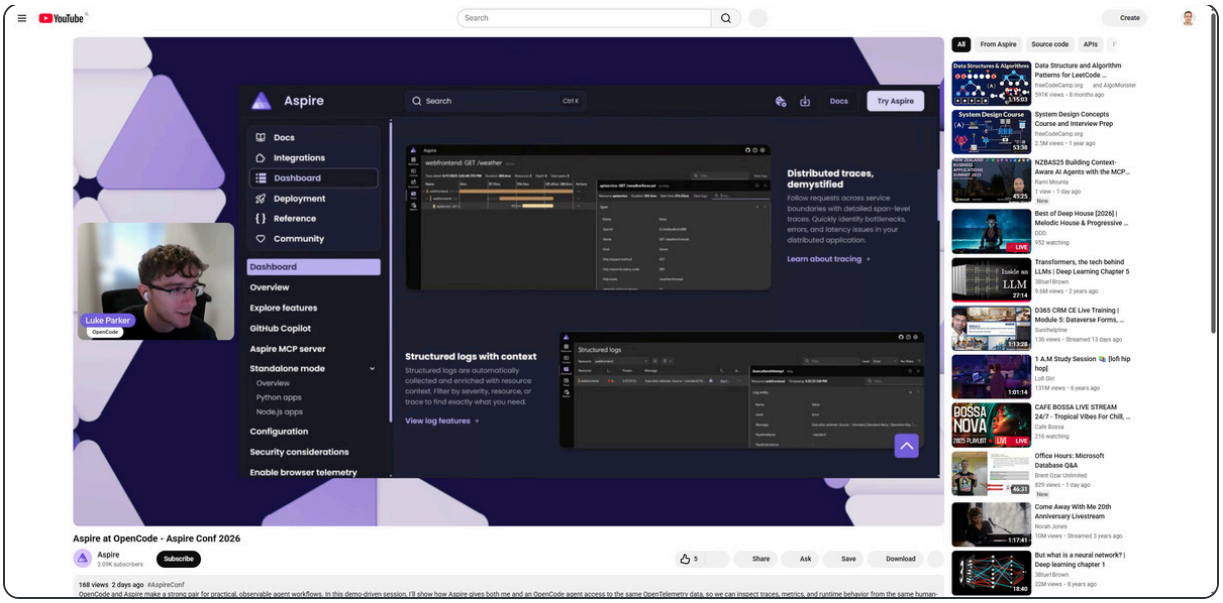
· Demos & Live Code

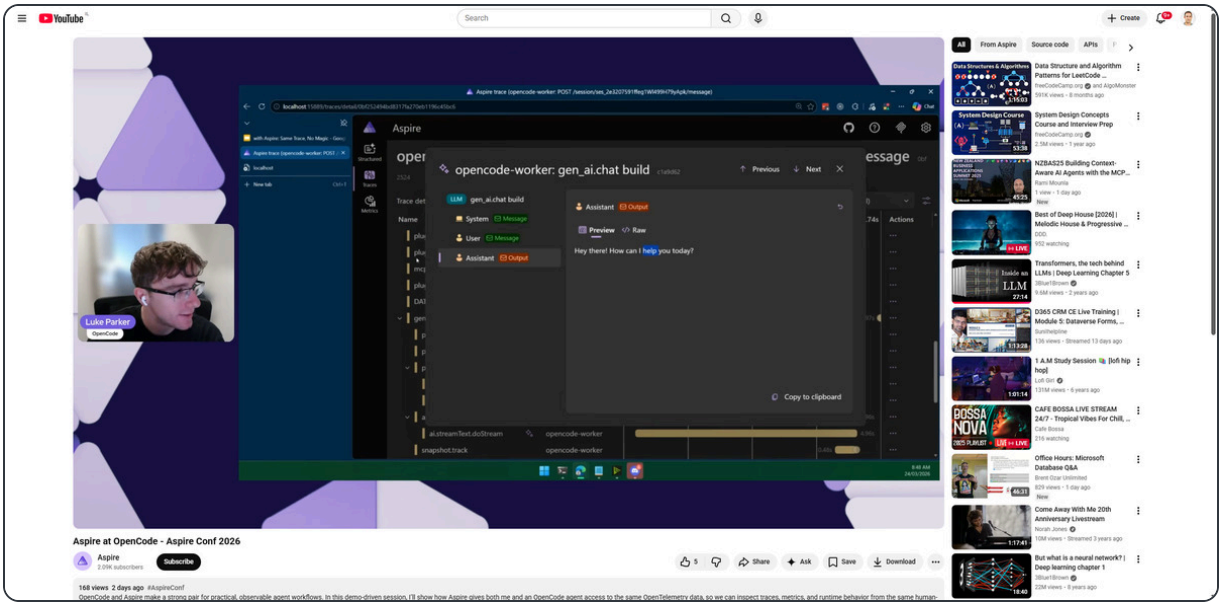
OpenCode wired to Aspire standalone dashboard — Set OpenTelemetry endpoint URL in OpenCode's environment variables, ran the standalone Aspire dashboard from source (not Docker), sent a 'hi' message to OpenCode, and observed the resulting spans in the dashboard: title generation, context building, LLM call (GPT-5 Nano), tool definitions, and streaming chunks.

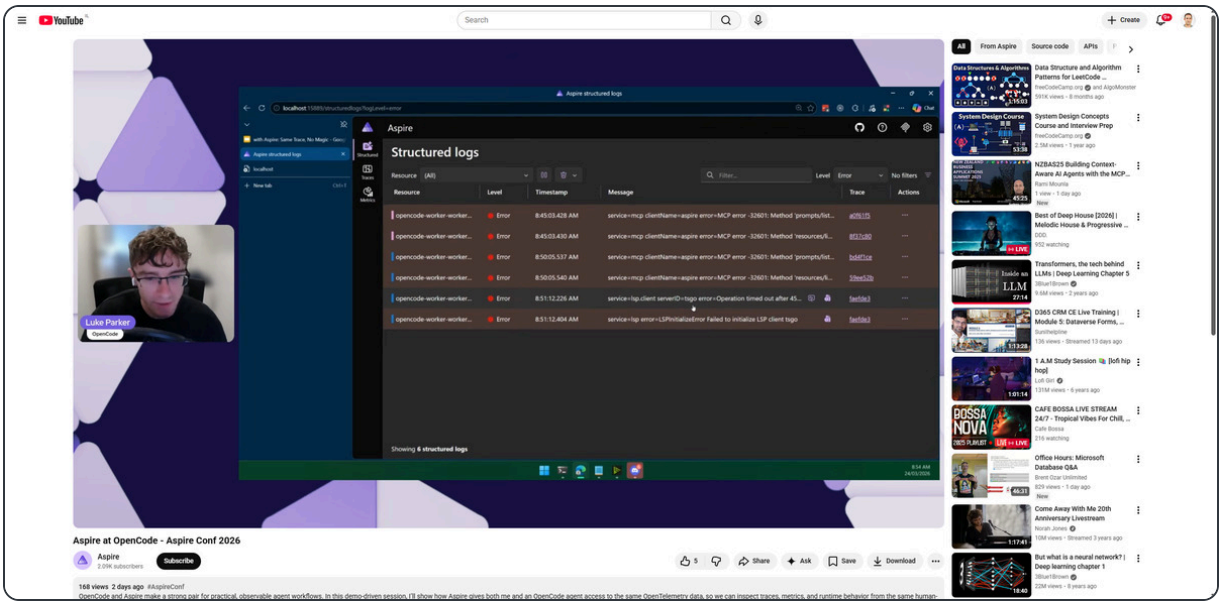
Diagnosing a 3-second bash span — Asked OpenCode to summarize a codebase, observed in the dashboard that a bash tool call took 3 seconds (anomalous vs. expected <500ms). Pasted a screenshot of the trace into the OpenCode chat. The agent — using `aspire list traces` via MCP — identified a plugin configured with a deliberate thread.sleep(3000) and pinpointed the exact file and line.

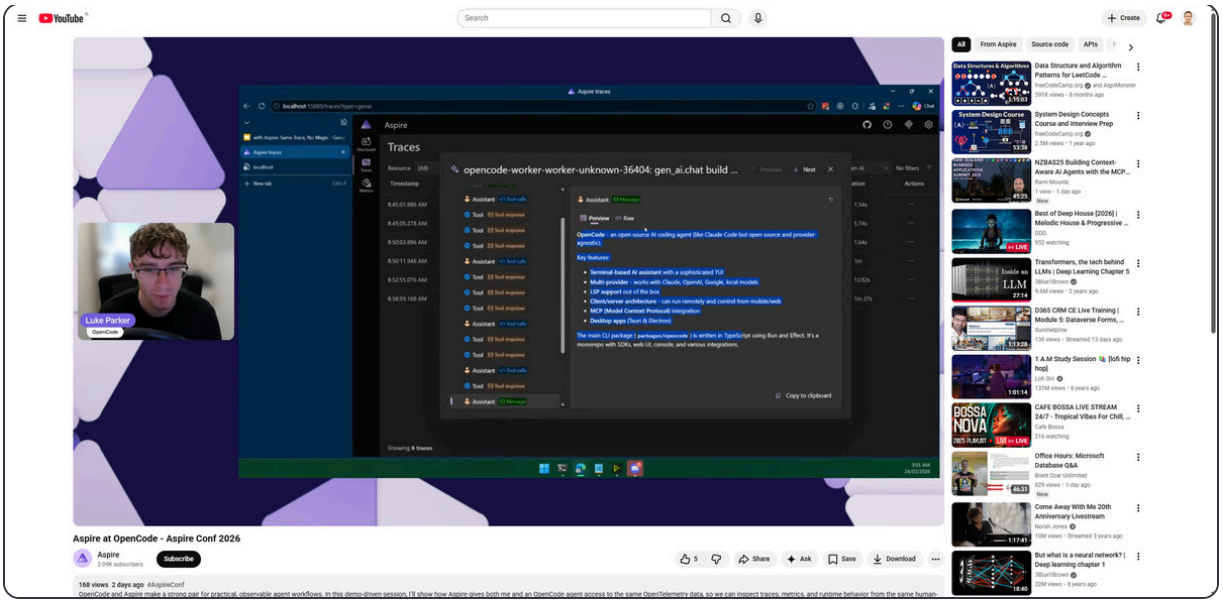
TSGO LSP timeout bug investigation — Switched to GPT-5.4 model and asked 'why is my TSGO LSP not working?'. OpenCode used Aspire MCP to query dashboard logs, found a timeout error that was invisible in raw log files, and suggested a configuration fix.

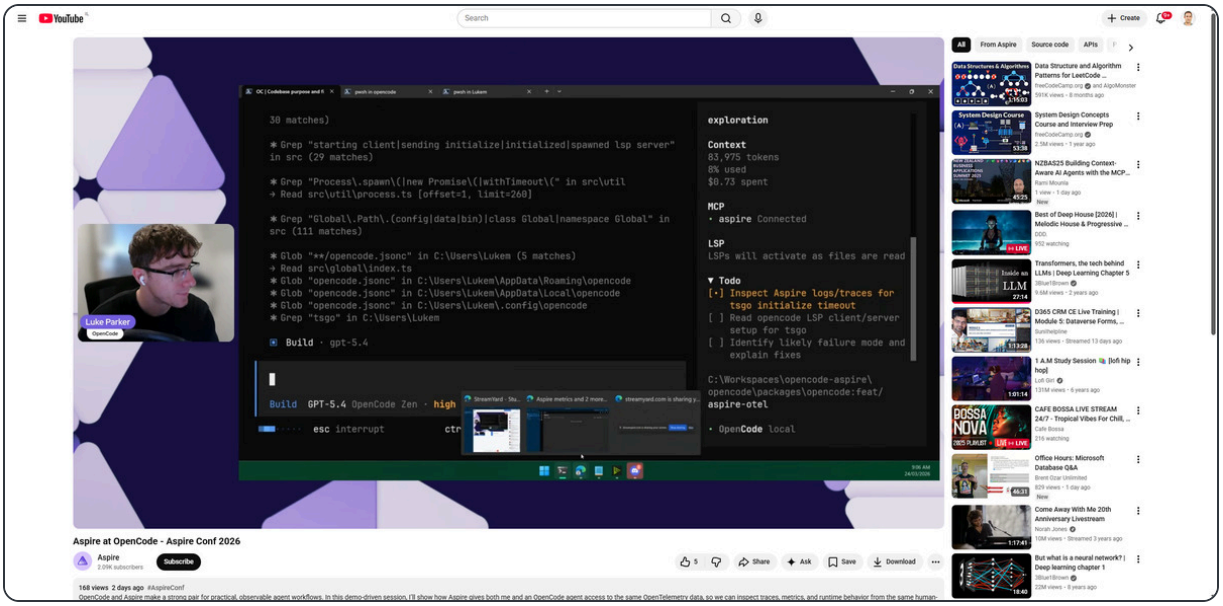
Browsing complete LLM context windows in spans — Navigated the tool definitions span to see the full bash tool definition, required inputs, and every tool OpenCode had available — useful for diagnosing context window overflow.


Aspire
Search
Docs
Try Aspire
Docs
Integrations
Dashboard
Deployment
Reference
Community
Dashboard
Overview
Explore features
GitHub Copilot
Aspire MCP server
Standalone mode
Overview
Python apps
Node.js apps
Configuration
Security considerations
Enable browser telemetry
Distributed traces, demystified
Follow requests across service boundaries with detailed span-level traces. Quickly identify bottlenecks, errors, and latency issues in your distributed application.
Learn about tracing
Structured logs with context
Structured logs are automatically collected and enriched with resource context. Filter by severity, resource, or trace to find exactly what you need.
View log features
Structured logs
Luke Parker
OpenCode
Aspire at OpenCode - Aspire Conf 2026
Aspire
2.09K subscribers
Subscribe
168 views 2 days ago #AspireConf


Aspire
opencode-worker: gen_ai.chat build
Previous
Next
gen_ai.chat build
Assistant
Output
System
Message
User
Message
Assistant
Output
Preview
Raw
Hey there! How can I help you today?
Copy to clipboard
ai.streamText.doStream
opencode-worker
snapshot.track
opencode-worker
Luke Parker
OpenCode
Aspire at OpenCode - Aspire Conf 2026
Aspire
2.09K subscribers
Subscribe
168 views 2 days ago #AspireConf


Aspire
Structured logs
Resource
Level
Timestamp
Message
Trace
Actions
Showing 6 structured logs
Luke Parker
OpenCode
Aspire at OpenCode - Aspire Conf 2026
Aspire
2.09K subscribers
Subscribe
168 views 2 days ago #AspireConf





· Code Samples

Run Aspire standalone dashboard via Docker (no AppHost needed)

```
docker run --rm -it -p 18888:18888 -p 4317:18889 \
  -e DOTNET_DASHBOARD_UNSECURED_ALLOW_ANONYMOUS=true \
  mcr.microsoft.com/dotnet/aspire-dashboard:latest
```

Configure OpenTelemetry in a Node.js/Bun app to send to Aspire dashboard

```
# Set environment variable to point to standalone dashboard OTLP endpoint
OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
OTEL_EXPORTER_OTLP_PROTOCOL=grpc
```

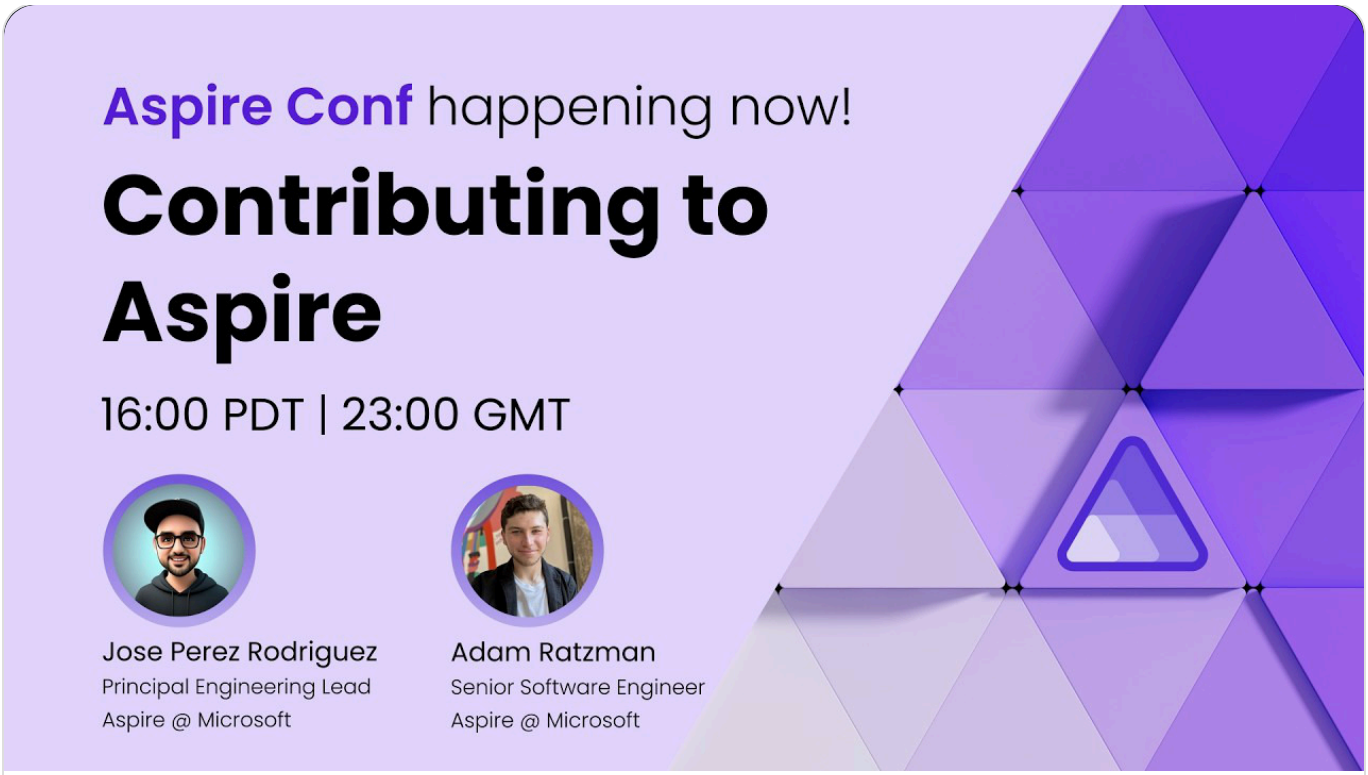
```
OTEL_TRACES_SAMPLER=always_on
OTEL_DOTNET_EXPERIMENTAL_GENAI_CAPTURE_MESSAGE_CONTENT=true
```

Aspire CLI command to list traces (used by OpenCode MCP)

```
aspire list traces [--resource ] [--top ]
```

Aspire CLI help for OTel data access in 13.2

```
aspire help
# Includes: aspire logs, aspire traces, aspire resources
```



💛 COMMUNITY

Contributing to Aspire

🎤 Jose Perez Rodriguez, Adam Ratzman, Maddy Montaquila

⏱ 24:45 | ▶ Watch Session

> *Aspire is open-source, and our community is the best in the game. Getting involved is easier than you think — whether that's filing an issue, contributing code to the core repo, helping build out aspire.dev, or shipping integrations in the Community Toolkit. In this session, Jose (Aspire's engineering manager) and Adam (one of the devs on the team) will break down all the ways you can contribute and pull back the curtain on how our code gets reviewed, tested, and released. This talk is from #As*

· Summary

The closing session of AspireConf 2026 was a warm, community-focused celebration and practical guide to contributing to , delivered by Engineering Manager Jose Perez Rodriguez and Senior Software Engineer Adam Ratzman. Jose opened with gratitude, acknowledging a remarkable milestone: over 600 community PRs merged, 170+ contributors, and 2,400+ community issues filed since Aspire's open-source journey began. These are not just numbers — the

community issues have directly shaped Aspire's roadmap, with the team repeatedly citing specific community requests as the reason features got prioritized. The session was equal parts 'thank you' and 'here's how to join.'

Adam used community contributor Alex as a concrete example of the impact individuals can have. Alex — one of the project's most prolific contributors and a trusted community triager — fixed a frustratingly vague error that appeared when a resource dependency failed to start. Previously, developers got a generic 'operation was canceled' exception with no context. Alex's fix enriched the error output to include which resources were involved, their current states, expected states, and health check statuses. This improvement benefited not just end users but the Aspire team itself, eliminating a class of flaky tests that previously timed out with no diagnostic information. The team also highlighted that Aspire now ships 'skills' checked into the repository to help AI coding agents make correct contributions — for example, a skill instructing agents to always add unit tests when modifying integration code.

The bulk of the session was a guided tour of the Aspire repository structure at `github.com/microsoft/aspire`: `source/` (AppHost, dashboard in Blazor, CLI, integrations), `test/` (unit and end-to-end), `docs/` (working documentation), `playground/` (dogfood apps for testing local package changes without publishing), `inch/` (build infrastructure), and `extension/` (VS Code extension). Adam — who built the VS Code app host viewer feature that Maddy and David Fowler initially said no to, only to enthusiastically ship once he showed it — walked through the contribution workflow: fork, run `restore.cmd`/`restore.sh`, make changes, add tests, submit a PR. A bot then posts a 'dogfood script' to the PR so contributors and reviewers can install the exact build from that PR and test it locally without polluting their NuGet cache.

Jose closed with a clear map of contribution pathways: new integrations go to the Community Toolkit (`github.com/community-toolkit/aspire`); fixes to existing integrations and the Blazor dashboard go directly to the main repo; documentation and samples live at `github.com/microsoft/aspire.dev`. Community triagers — trusted contributors like Alex who help triage issues — represent a non-code contribution path. Daily builds and staging builds are available for early adopters. The team committed to repopulating 'good first issue' and 'help wanted' labels on GitHub in the days following the conference, and emphasized that diverse perspectives — especially first-time open-source contributors — are genuinely valued.

· Key Takeaways

→ Aspire has merged 600+ community PRs from 170+ contributors, with 2,400+ community issues shaping its roadmap

→ The Aspire repository moved to `github.com/microsoft/aspire`; new community integrations go to `github.com/community-toolkit/aspire`

→ Community contributor Alex fixed the vague 'operation was canceled' error to show full resource dependency failure details, including states and health checks

→ A 'dogfood script' bot posts build artifacts to every PR, letting contributors and reviewers test changes locally without publishing to a NuGet feed

→ Aspire ships AI agent skills in the repository to guide coding agents making contributions (e.g., always add unit tests)

→ Daily builds and staging builds are available for installation via a one-line script with a `--quality` parameter

→ Adam Ratzman built the VS Code app host viewer that was initially rejected by Maddy and David Fowler, but shipped immediately once demoed

→ Community triagers (trusted external contributors like Alex) help triage issues alongside the core team

· Announcements & Features

Aspire Repository Migration to Microsoft Org — The Aspire GitHub repository has moved to `github.com/microsoft/aspire`. API documentation is now hosted at `aspire.dev`. All conceptual docs live at `github.com/microsoft/aspire.dev`.

Community Toolkit for New Integrations — New community-contributed integrations should go to the Aspire Community Toolkit at `github.com/community-toolkit/aspire`, keeping the main repo manageable while still welcoming community additions.

Good First Issue Label Repopulation — The team committed to repopulating 'good first issue' and 'help wanted' labels on GitHub in the days following AspireConf, providing clear entry points for new contributors.

AI Agent Skills in Repository — Aspire now ships agent skills (instruction files) checked into the repository to guide AI coding agents in making correct contributions, such as always adding unit tests when modifying integration code.

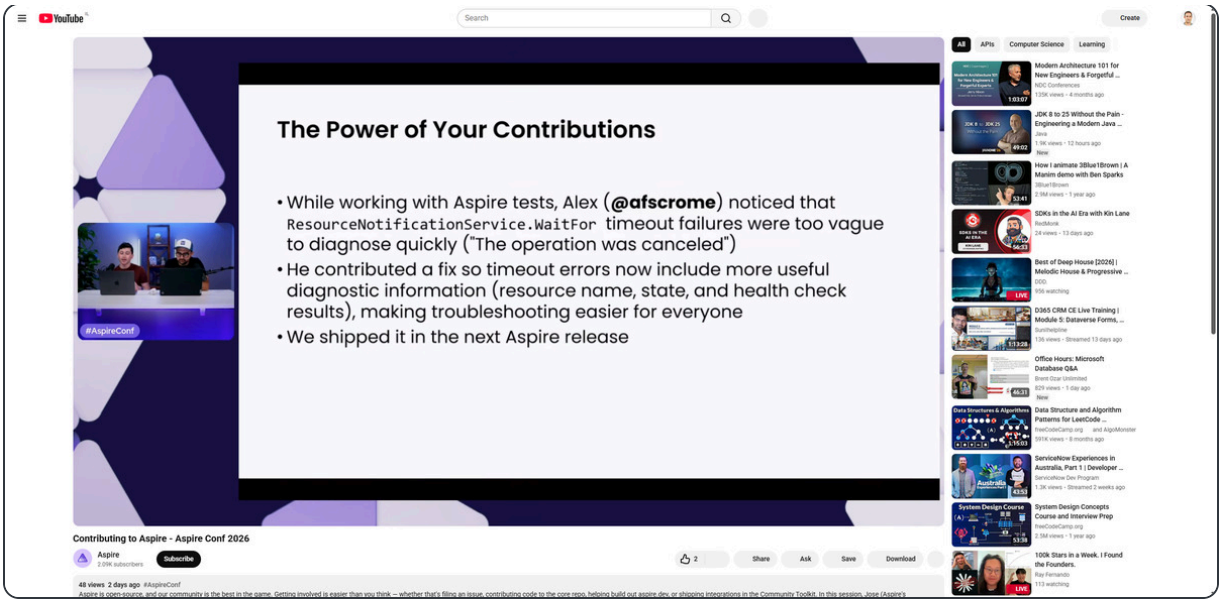
aspire.dev/community/thanks Page — David Pine built a dedicated page at aspire.dev/community/thanks to celebrate and acknowledge all community contributors.

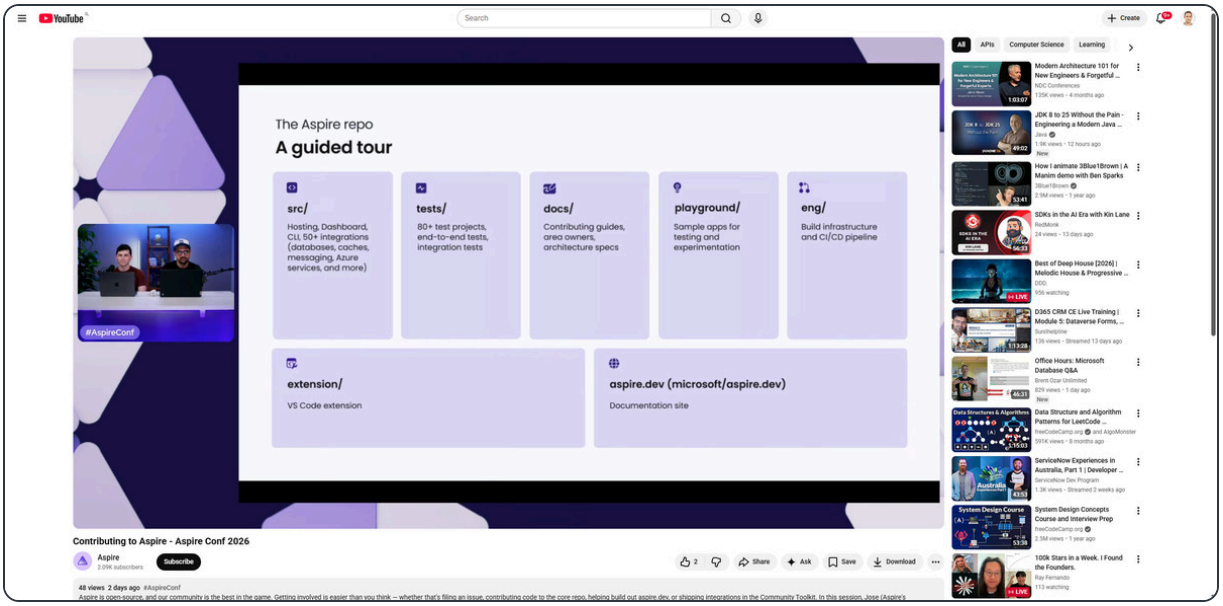
· Demos & Live Code

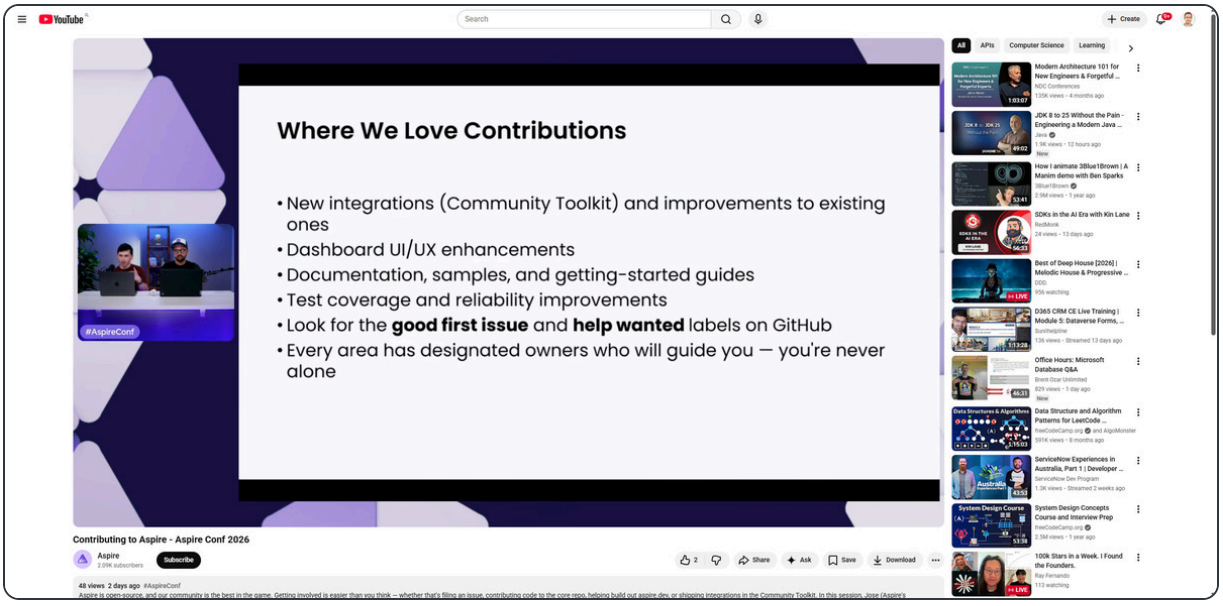
Repository structure walkthrough — Live tour of the Aspire GitHub repo: source/, test/, docs/, playground/ (dogfood apps with local package references), inch/ (build infra), extension/ (VS Code extension), and aspire.dev (separate repo).

Contribution workflow end-to-end — Walked through the full flow: fork repo → run restore.cmd → make changes → add tests → submit PR → bot posts dogfood script → team iterates on feedback → merge.

Daily builds installation — Demonstrated the one-line install script with `--quality daily` parameter for installing daily Aspire builds that include every merged PR, and `--quality staging` for pre-release candidate builds.







- Code Samples

Bootstrap the Aspire repo for local development

```
# Windows
restore.cmd
build.cmd

# Linux/macOS
./restore.sh
./build.sh
```

Install Aspire daily build (latest commits)

```
# Windows (PowerShell)
irm https://aspire.dev/install.ps1 | iex -Quality daily

# Linux/macOS
curl -sSL https://aspire.dev/install.sh | bash -s -- --quality daily
```

Install Aspire staging build (release candidate)

```
# Install the current staging/RC build
irm https://aspire.dev/install.ps1 | iex -Quality staging
```

Dogfood a specific PR's changes locally (posted by bot)

```
# Bot posts this script in every PR:
./dogfood.ps1 -PR 12345
# Downloads build artifacts into isolated hive — does not pollute global Nu
```

Aspire Conf 2026 — Book of News

Generated with ❤️ by Squad AI

Watch all sessions on YouTube