

From Build to On-Call with AI

Technical Deep Dive: Kiro & DevOps Agent

Jason Mimick

WW DevOps & Observability Sr PSA
Amazon Web Services



The promise of agentic development

Autonomy

Agents can take on and complete increasingly challenging tasks autonomously

True Collaboration

Developers and agents work together to get more done

Higher quality

Agents handle the heavy lifting of increasing test coverage, documenting code, optimizing performance, fixing security vulnerabilities

Challenges with AI development

Scaling AI development

AI coding tools excel at small tasks but can fail with complex projects

Limited control

Existing tools make it difficult to collaborate with and manage agents

Code quality

Getting a project from proof-of-concept to production while maintaining quality control becomes increasingly difficult

Solution: Spec-driven development

Vibe coding ships the prototype,
traditional SDLC practices ship the product



Kiro helps developers and engineering teams
ship high quality software with AI agents

One agent, every surface

The collage consists of four overlapping screenshots of the Kiro agent interface:

- Top Left (Code Editor):** Shows a file explorer with a project structure including 'backend', 'src', 'providers', 'api.rs', 'auth.rs', 'cache.rs', 'creds.rs', 'error.rs', 'main.rs', 'mcp.rs', 'state.rs', 'target', '.gitignore', 'Cargo.lock', 'Cargo.toml', 'deploy', and 'docs'. The 'auth.rs' file is open, showing Rust code for authentication logic.
- Top Right (Terminal):** Displays system metrics and process information. The top section shows CPU usage (58.7%, 52.3%, 48.3%, 42.4%) and memory usage (19.7G/24.0G). The bottom section shows a table of running processes with columns for PID, USER, PRI, NI, VIRT, RES, S, CPU%, MEM%, TIME+, and Command.
- Bottom Left (CLI):** Shows a prompt 'Let's build' with a plan, spec, and bug fix structure. Below it, a list of tasks is displayed: 'Start with a workflow', 'Spec Structur', 'Plan Plan-only', 'Bug Fix Struc', and 'Quick Spec F'.
- Bottom Right (Web Browser):** Shows a 'Kiro Crew Landing Page Design' page. It includes a 'Sessions' panel on the left, a 'Thinking' section in the middle, and a 'Message Kiro Crew...' input field at the bottom.

An early release of Kiro CLI V3 is now available! Try it out: `kiro-cli --v3`
<https://kiro.dev/docs/cli/v3/>

Tip: Share your thoughts anytime – type `/feedback` to send feedback to the Kiro team.

✨ **What's new:** Cloud sessions, specs, expanded hooks, and an improved trust model.

`ctrl+o` to expand

Ask a question or

@ GPT5 kiro_default · claude-opus-4.8 · 10%

thinking: default · ~/work/pixel-quest · (main)

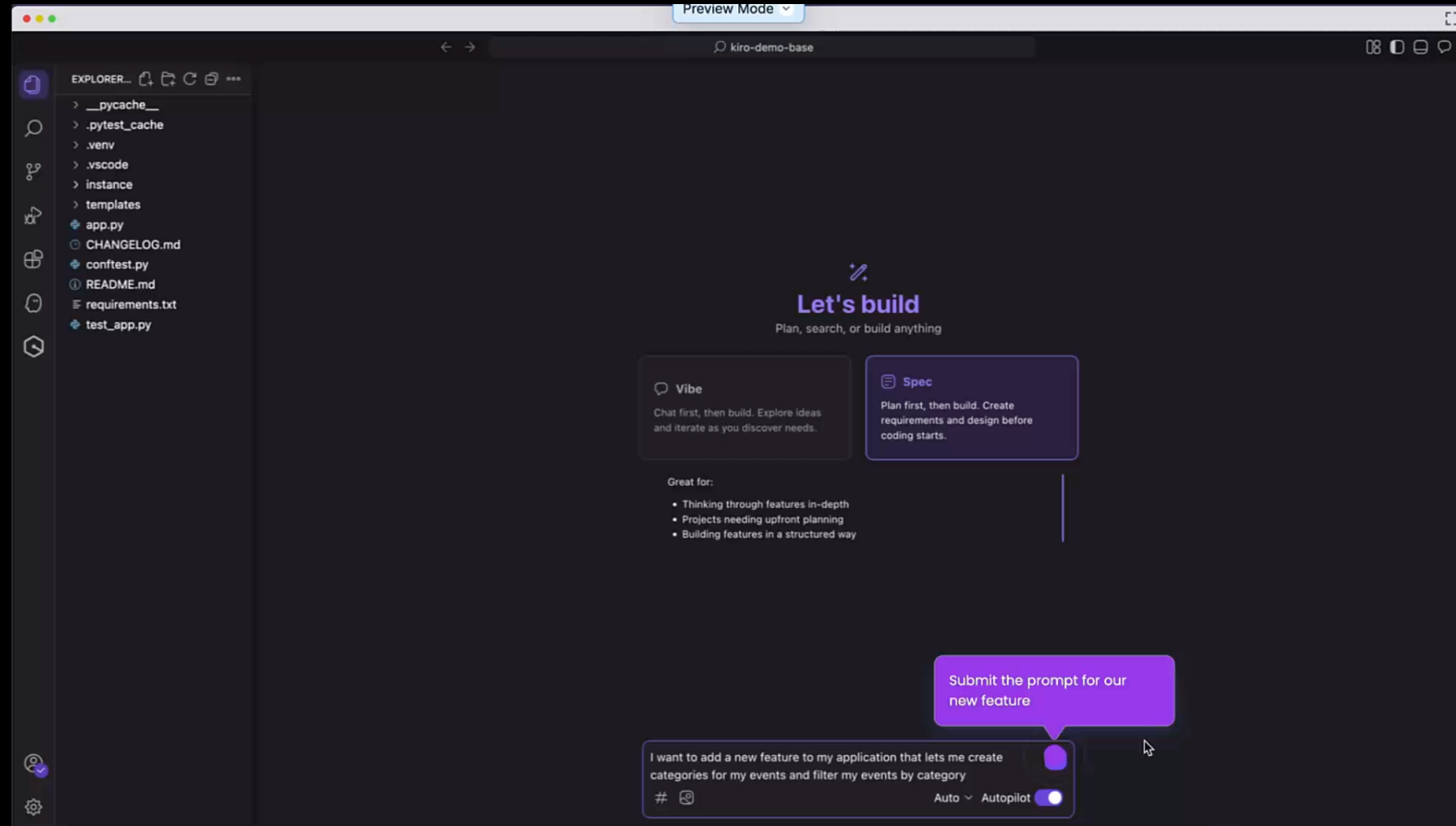
> Ask a question or describe a task

/copy to clipboard

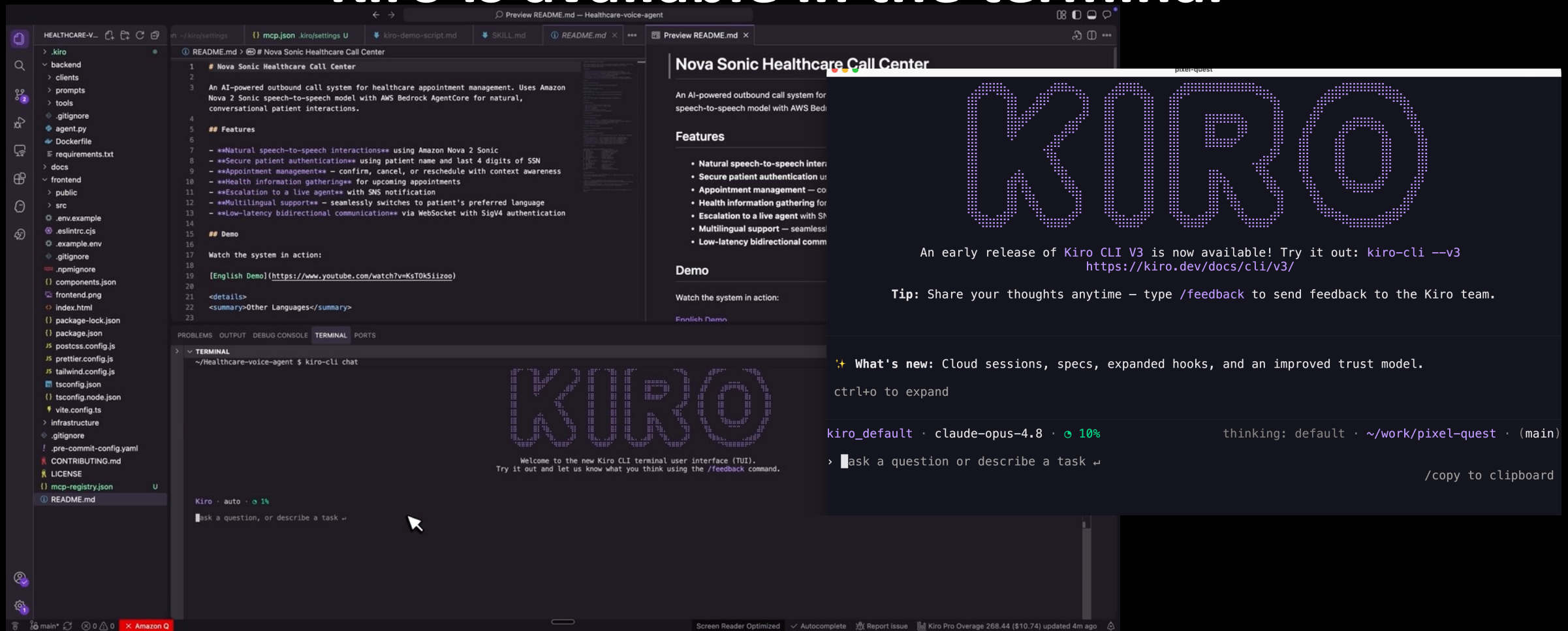
One unified agent harness powers Kiro across IDE, CLI, Web, and Crew.
Your specs, steering, and .kiro/ config travel with the repo and work on
every surface.

Spec-driven development

- Kiro turns your prompt into clear requirements, system design, and discrete tasks
- Iterate with Kiro on your spec and architecture
- Kiro agents implement the spec while keeping you in control



Kiro is available in the terminal



The screenshot shows a VS Code editor with a project named "Nova Sonic Healthcare Call Center". The project files include a README.md, .gitignore, agent.py, Dockerfile, requirements.txt, docs, frontend, public, src, .env.example, .eslintrc.cjs, .example.env, .gitignore, .npmignore, components.json, frontend.png, index.html, package-lock.json, package.json, postcss.config.js, prettier.config.js, tailwind.config.js, tsconfig.json, tsconfig.node.json, vite.config.ts, infrastructure, .gitignore, .pre-commit-config.yaml, CONTRIBUTING.md, LICENSE, mcp-registry.json, and README.md.

The terminal window shows the Kiro CLI running. It displays the Kiro logo and a welcome message: "Welcome to the new Kiro CLI terminal user interface (TUI). Try it out and let us know what you think using the /feedback command." Below the logo, it says "Kiro · auto · 1%" and "Ask a question, or describe a task".

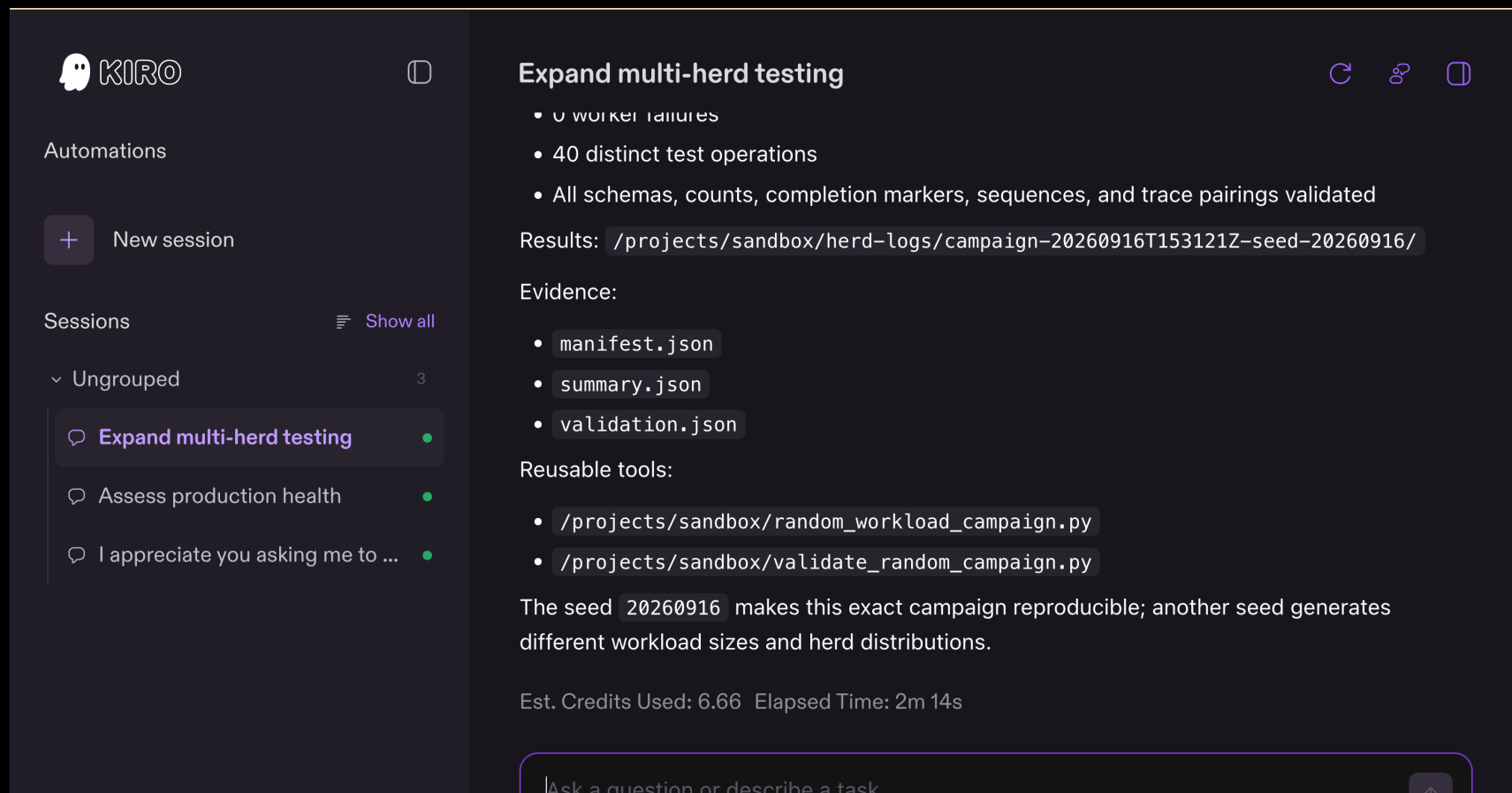
On the right side of the image, there is a large KIRO logo made of dots. Below it, text reads: "An early release of Kiro CLI V3 is now available! Try it out: `kiro-cli --v3` <https://kiro.dev/docs/cli/v3/>". Below that, a tip says: "Tip: Share your thoughts anytime – type `/feedback` to send feedback to the Kiro team." Below the tip, there is a section titled "What's new: Cloud sessions, specs, expanded hooks, and an improved trust model." followed by "ctrl+o to expand". Below that, there is a status bar showing "kiro_default · claude-opus-4.8 · 10%" and "thinking: default · ~/work/pixel-quest · (main)". Below the status bar, there is a prompt: "> Ask a question or describe a task" followed by a link to "/copy to clipboard".

Terminal-native Kiro: headless mode for CI/CD, session management, and cloud sessions with `kiro-cli --cloud` — turn ideas into implementation while staying in complete control every step of the way.

Kiro in your browser — now generally available

Build, delegate, and steer from your browser

- Browser-native agentic development at app.kiro.dev — no local setup required.
- Assign a goal, step away, and come back to a pull request.
- Autonomous mode: Kiro plans, delegates to sub-agents, verifies each step, and opens a PR — nothing merges without your approval.
- One conversation across multiple repos, with coordinated PRs.
- GitHub-native: a kiro label or /kiro comment assigns work; steering files carry your conventions across IDE, CLI, and Web.



Kiro Crew — your open-source personal agent

Persistent context. Continuous learning.

Progress that keeps moving.

- <https://kiro.dev/crew/>
- Open-source personal AI agent that runs on your hardware — local or remote
- Persistent and self-improving: autonomous tasks, scheduling, heartbeats, and sub-agents
- Multi-channel — drive it from Slack, Discord, Telegram, Teams, and more
- Same specs and steering that power the rest of Kiro — it's part of the family

The screenshot displays the KIRO Crew interface. On the left sidebar, under 'Automations', there is a '+ New session' button. Under 'Sessions', there is a 'Show all' link and a list of sessions under the 'Ungrouped' category. The selected session is 'Expand multi-herd testing', which has a green status indicator. Below it are two other sessions: 'Assess production health' and 'I appreciate you asking me to ...', both also with green status indicators.

The main content area shows the details for the 'Expand multi-herd testing' session. It includes a title bar with refresh, share, and close icons. The content lists several bullet points: '0 worker failures', '40 distinct test operations', and 'All schemas, counts, completion markers, sequences, and trace pairings validated'. Below this, the 'Results' are shown as a file path: '/projects/sandbox/herd-logs/campaign-20260916T153121Z-seed-20260916/'. The 'Evidence' section lists three files: 'manifest.json', 'summary.json', and 'validation.json'. The 'Reusable tools' section lists two Python scripts: '/projects/sandbox/random_workload_campaign.py' and '/projects/sandbox/validate_random_campaign.py'. A note states: 'The seed 20260916 makes this exact campaign reproducible; another seed generates different workload sizes and herd distributions.' At the bottom, it shows 'Est. Credits Used: 6.66' and 'Elapsed Time: 2m 14s'. A text input field at the very bottom says 'Ask a question or describe a task...'.

Connect Kiro Crew to Dynatrace

A purpose-built Crew app brings Dynatrace context into persistent agent workflows.

Install once

The Crew app installs dtctl and the integration workflow.

Choose authentication

Sign in with Dynatrace, or use an access token for automation and non-SSO environments.

Keep work moving

Sessions can investigate, create artifacts, and continue without a developer waiting at the terminal.

The screenshot displays a three-step process for connecting the Kiro Crew app to Dynatrace:

- Step 1:** "Install dtctl" with a green checkmark and the status "Installed".
- Step 2:** "Connect your tenant". This step includes:
 - A purple button labeled "Sign in with Dynatrace" with the text "Recommended · no token needed" below it.
 - An "Access token" box with the text "For automation or no SSO".
 - A text input field for "YOUR ENVIRONMENT URL" containing the value "https://hgd83841.apps.dynatrace.com".
 - A note: "This is the URL you open in your browser to use Dynatrace."
 - A purple button labeled "Sign in with Dynatrace".
- Step 3:** "You're connected" with a green checkmark.

Kiro Crew app example · Dynatrace



🗨

Sessions

📅

Schedule

🔗

Artifacts

Apps

🔍

Discover

🗄

Library

🌈

Dynatrace

🚀

Task Runner**Dynatrace**

hgd83841.apps.dynatrace.com

Problems

Services

Applications

Smart (w

UNHEALTHY



2

**DynaTraceLambdaApiGatewayEnvi-
ItemsHandlerFB09CCF4-
ZKWC0YSuSedr in us-west-2**

2

1

active

recent

SERVICE-604A99F533...

**ingestion-demo-s3-writer in us-
west-2**

1 active

SERVICE-0BC129A7436EB1...

WATCH



3

**DynaTraceLambdaApiGateway-
GetItemByIdHandlerB513D6-
1plRuKRBqJ99 in us-west-2**

12 recent

SERVICE-123DCA9659FD53...

frontend

service:frontend

stage:production

aws.region:us-east-1

+1

2 recent

SERVICE-6EE96C17638CD0...

requestlogger

1 recent

SERVICE-9B19C6FE6E062C...

HEALTHY

**agentcore-de**

Healthy

SE

agentcore-de

Healthy

SE

app

aws.account.id

aws.region:us-v

Healthy

SE

backend

stage:producti

aws.region:us-e

project:dt-orde

Healthy

SE

bot-action

aws.account.id

aws.region:us-e

Healthy

SE

**DynaTraceLambdaApiGatewayEnvi-
ItemsHandlerFB09CCF4-ZKWC0YSuSedr in us-west-2**

×

DETAILS

Name	DynaTraceLambdaApiGatewayEnvi- ItemsHandlerFB09CCF4-ZKWC0YSuSedr in us- west-2
Type	SERVICE
Entity ID	SERVICE-604A99F5338C9A07

ACTIVE PROBLEMS

- Lambda arn:aws:lambda:us-west-
2:939328548145:function:DynaTraceLambdaApiGatewayEnvi-
ItemsHandlerFB09CCF4-ZKWC0YSuSedr Failed
- ERROR Multiple service problems P-2609326

RECENT PROBLEMS

- Lambda arn:aws:lambda:us-west-
2:939328548145:function:DynaTraceLambdaApiGatewayEnvi-
ItemsHandlerFB09CCF4-ZKWC0YSuSedr Failed
- ERROR

LINKS

[Open in Dynatrace](#)

Cloud sessions — the unified agent harness, in the cloud

Cloud sessions

- A cloud session runs the unified agent harness in a managed cloud sandbox tied to your Kiro account — not a single app.
- Start a task in the browser, hand it off, close the laptop, and resume from the CLI or IDE. Launch from the terminal with `kiro-cli --cloud`.
- You control what each session can reach: network access, MCP tools, powers, and secrets.

The screenshot displays the KIRO Sessions web interface. On the left, a sidebar shows 'Automations' and 'Sessions' (4 of 4 Sessions). The 'Sessions' section lists four active sessions: 'Project Overview Discussion', 'Expand multi-herd testing', 'Assess production health', and 'I appreciate you asking me to c...'. The main panel shows a table of sessions with columns for Name, Session ID, Repos, Group, Status, Last updated, and Actions. The terminal window shows the KIRO CLI V3 upgrade message, including instructions for upgrading agent configurations and fetching cloud config, learnings, and steering files.

Sessions
4 of 4 Sessions

Active Archived

Name ↓	Session ID	Repos	Group	Status ↓	Last updated ↓	Actions
Project Overview Discussion	ba92d99f...	—	—	Active	Sep 23, 2026, 11:38 AM	
Expand multi-herd testing	749c8eb6...	—	—	Active	Sep 16, 2026, 11:23 AM	
Assess production health						
I appreciate you asking me to continue, but I want to clarify what you're looking for...						

Total sandbox time used this month: 2 hours, 20 minutes

Welcome to KIRO CLI V3!
Upgrade your V2 agent configurations to V3 with `/upgrade-agent`
<https://kiro.dev/docs/cli/v3/>
Prefer the previous experience? Launch with `kiro-cli --v2`

Tip: Press **Ctrl+R** to reverse-search prompt history; press again for older matches, **Esc** accepts.

What's new: Cloud sessions, specs, expanded hooks, and an improved trust model.

```
ctrl+o to expand
✓ Connected to kiro.dev
✓ Cloud session created
○ No repository attached, /repo to connect one (optional)
Your cloud workspace doesn't have your local setup by default. Go to app.kiro.dev/settings/cloud-config to bring your agents, MCP servers, hooks, and steering from ~/.kiro/ (home directory) to the cloud.
```

Fetching your cloud config
toolName=fetch_cloud_config

Fetching your learnings
toolName=get_learnings_for_prompt, limit=50

Fetching steering files
toolName=get_steering_files

Default · Claude Opus 4.8 · high · 1% thinking: on · Cloud (Preview)

ask a question or describe a task

/sessions to resume · /copy to clipboard

The best model is the one that fits the job

Choose across OpenAI, Anthropic, and open-weight models — or let Auto route for you.

OpenAI

GPT-5.6 Sol · Terra · Luna

Long-horizon reasoning
Terminal workflows
Tool coordination

Anthropic

Claude Opus · Sonnet

Planning + self-verification
Complex debugging
End-to-end feature work

Open-weight

MiniMax · GLM · DeepSeek · Qwen

Long agentic sessions
Repository-scale context
Specialized workflows

AUTO — RECOMMENDED · Frontier-model routing with automatic fallback

You control the model — or delegate the choice to Kiro.

Your code is now deployed!
Let's move to Ops





AWS DevOps Agent

Drive operational excellence with an **autonomous AI agent** that resolves and proactively prevents incidents in **AWS, multi-cloud, and on-prem** environments

On-demand SRE task handling

Autonomous incident response

Proactive incident prevention

AWS DevOps Agent: Transforming Incident Response and Operations

Metrics derived from 10+ Customer & Partner references



MTTR REDUCTION

75%

Average

Range: 50–83%



INVESTIGATION TIME

80%

Faster

42 min → 8 min per incident



ROOT CAUSE ACCURACY

>90%

Pass rate

Across 85 tested scenarios



RESOLUTION SPEED IMPROVEMENT

8.5x

Faster

Range: 7-10x faster



PROVEN IN PRODUCTION

77%

WGU MTTR reduction

2 hrs → 28 min investigation

Custom SRE Agents

Encode expertise. Automate repeatable operational workflows.

Tailored to your environment

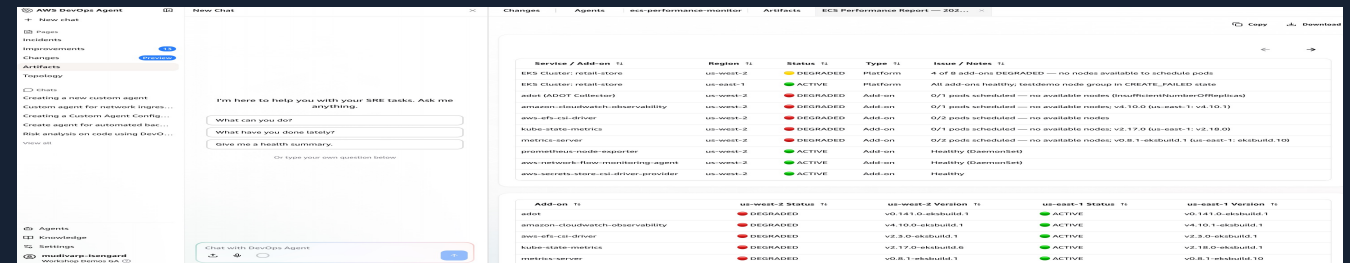
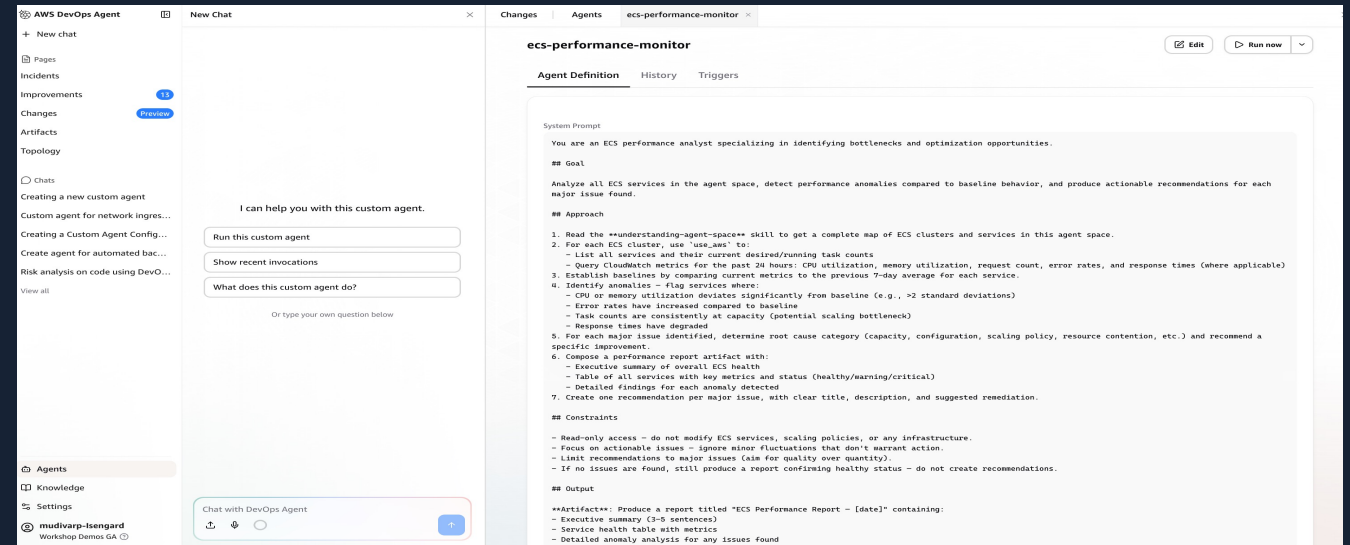
Define the goal, prompt, curated MCP tools, and specialized skills.

Run when the work needs to happen

Invoke on demand or on a recurring schedule inside an Agent Space.

Reduce repetitive SRE toil

Apply consistent operational rigor and produce reusable artifacts every time.



Examples: daily health reports · anomaly reviews · cross-service readiness checks

Skills · Custom agents · MCP servers



Bring production intelligence to the developer

From alarm to fix — without leaving the IDE

Shift left reliability

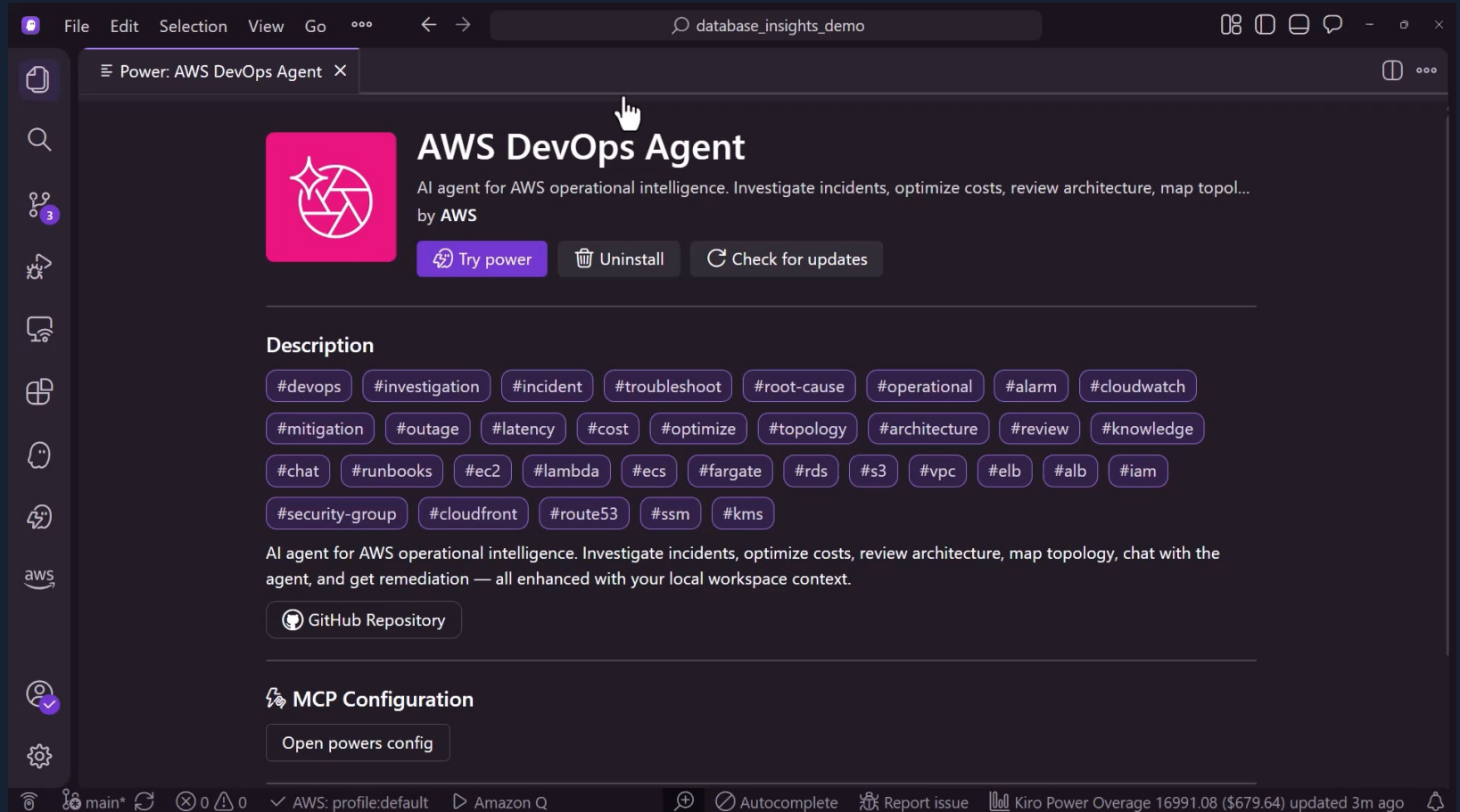
Run proactive reliability checks before code leaves the desktop—not after it breaks production.

Use production context

Review architecture, incidents, topology, and operational signals alongside local workspace context.

Automate the check

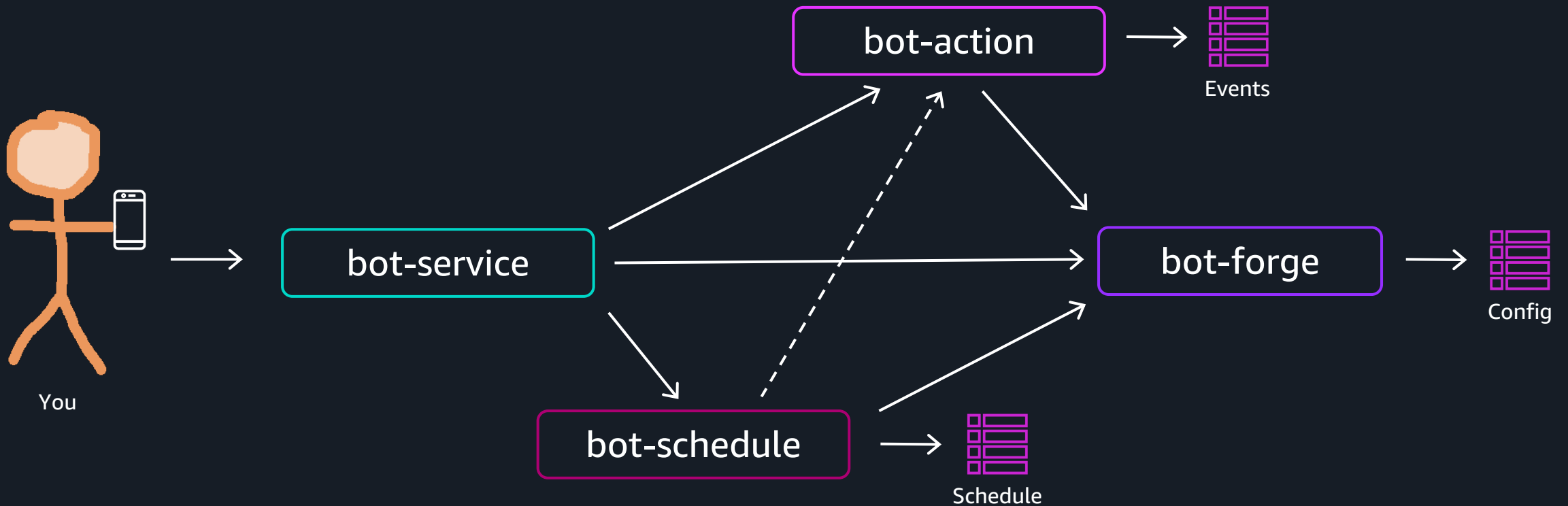
Invoke release-readiness reviews from a Kiro hook after substantial spec tasks.



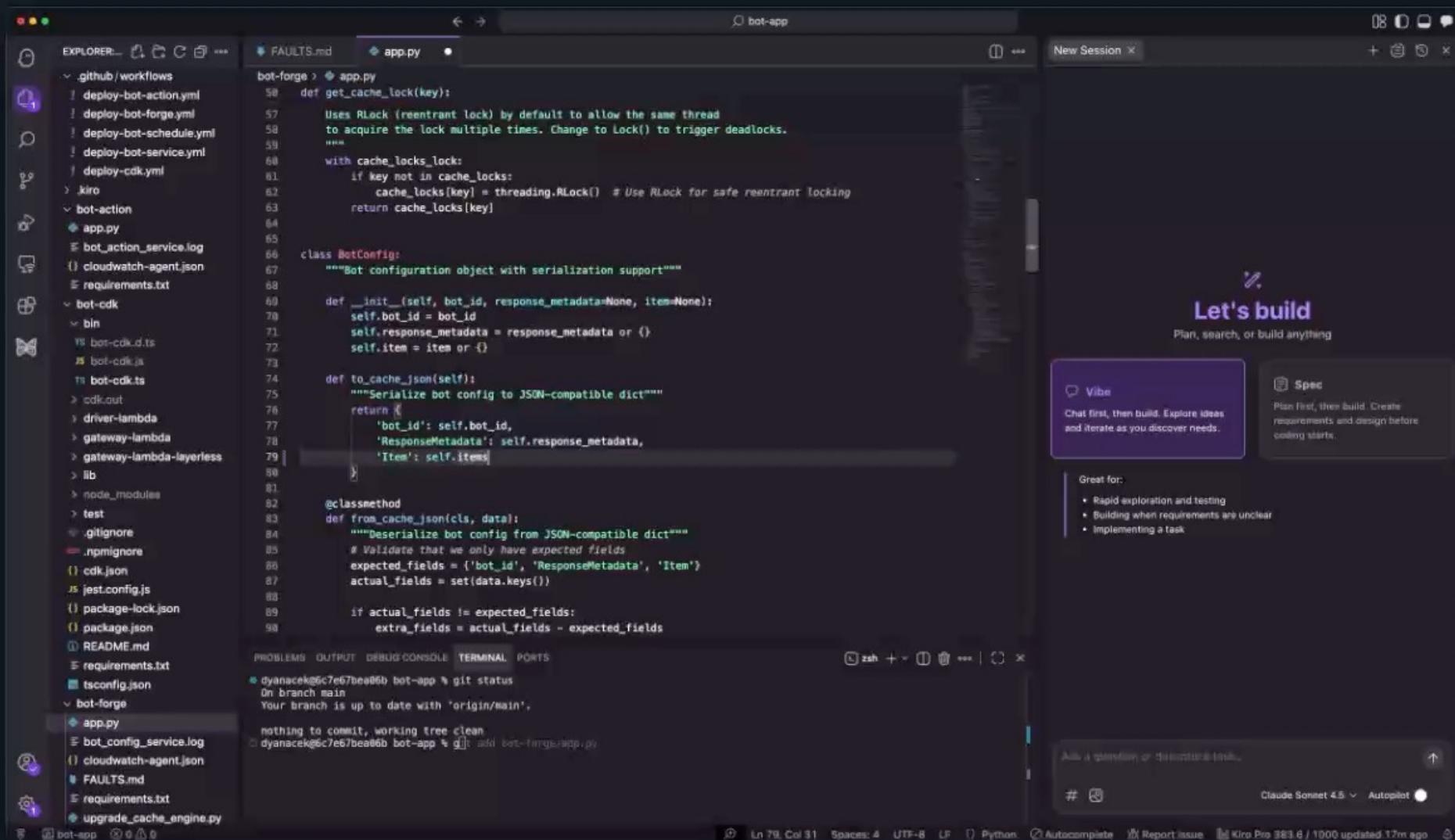
AWS DevOps Agent Power for Kiro



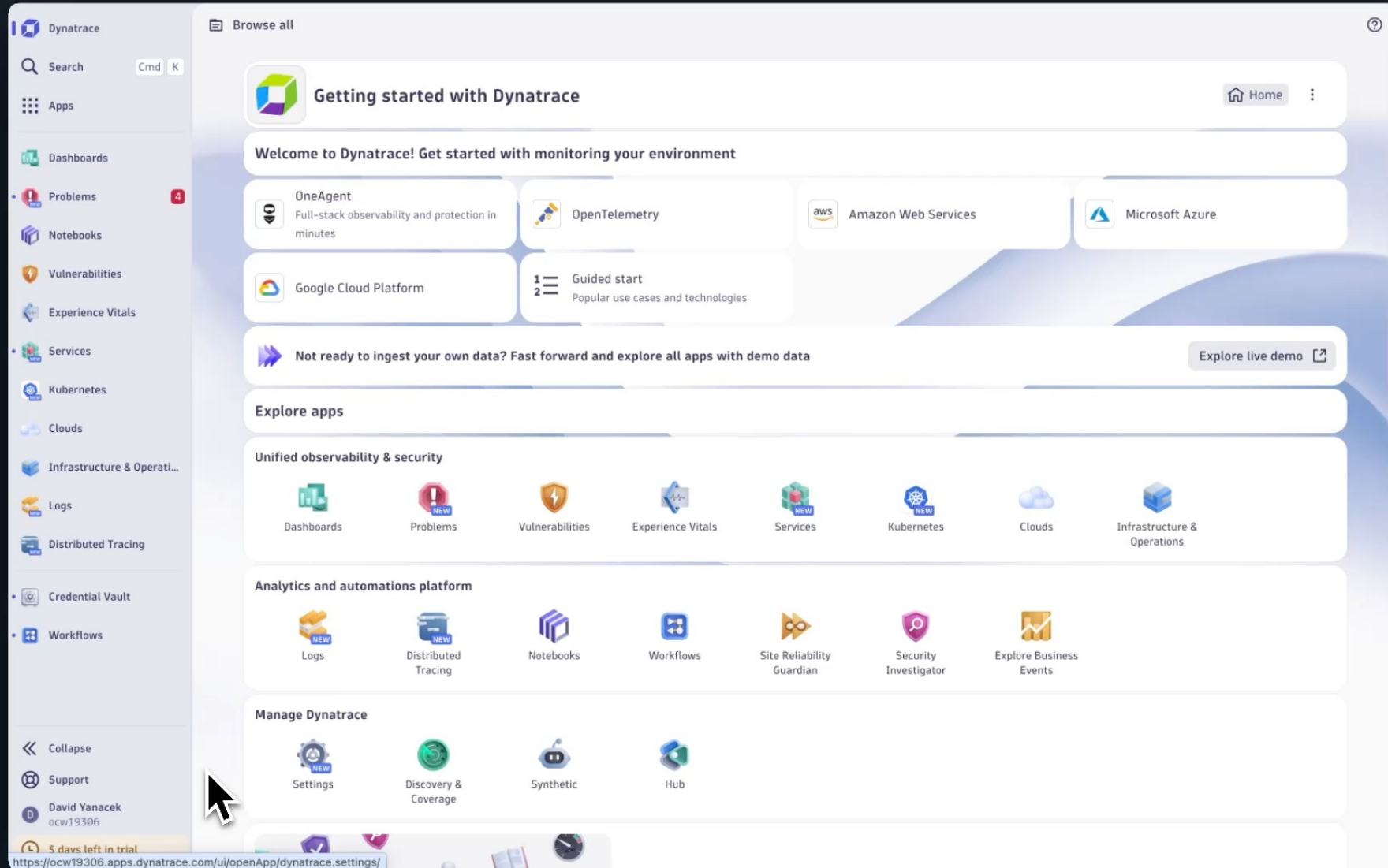
Demo - Bot app architecture



Pushing a bug to production



Dynatrace detects the problem



AWS DevOps Agent investigates — in parallel

The screenshot displays the AWS DevOps Agent investigation interface. The top navigation bar shows 'Incidents' and 'Failure rate increase (P...'. The main content area is titled 'Investigation timeline' and shows a sequence of events:

- +5m51s: Now I have all the evidence needed. Let me write the final investigation report.
- Root cause +5m51s: **Commit a12fb79 adds 'timestamp' field to cached payload that bot-forge's strict deserialization rejects with ValueError**
- +5m52s: Investigation completed

The root cause event is expanded, showing a detailed description of the issue:

Commit a12fb79 ('add timestamp to cached bot config') adds ``cached_payload["timestamp"] = time.time()`` at line 214 of `bot-forge/app.py` before storing in the in-memory cache. When the cache is subsequently read, ``BotConfig.from_cache_json()`` at line 94 performs strict field validation that only allows exactly `{"bot_id", "ResponseMetadata", "Item"}`. The extra "timestamp" field causes a `ValueError('Invalid bot config format: Unexpected fields: {"timestamp"}')` on every cache hit, which is unhandled and results in HTTP 500 Internal Server Error. The first request per `bot_id` succeeds (cache miss → DynamoDB read), but all subsequent requests fail (cache hit → deserialization `ValueError`). CloudWatch logs confirm 543 `ValueError` occurrences starting at 18:05:55 UTC, sustained at ~60 errors/minute. This commit was previously introduced (2677357), reverted (094e4fb), and then re-introduced as a12fb79.

A button labeled 'Go to root cause' is visible below the description.

At the bottom, a progress bar indicates 'Context used for investigation 31%'. Below the progress bar, there is a feedback prompt: 'Was this investigation helpful? Your feedback helps us improve.' with thumbs up and thumbs down icons.

Root cause: commit a12fb79

Incidents Failure rate increase (P... x

Investigation timeline Root cause Mitigation plan

Observation +4m47s

bot-action returning 500s due to bot-forge (botforge.bot.internal) returning 500 Internal Server Error

bot-action (port 8002) is called by bot-service and in turn calls bot-forge (botforge.bot.internal). Starting at 18:05, bot-action consistently logs 'bot-forge error' with HTTP 500 from botforge.bot.internal for ALL bot IDs (bot-001 through bot-005). This results in bot-action returning 500 to its caller (bot-service) using python-requests/2.32.3. 48 errors were logged in the primary window from bot-action, all pointing to bot-forge as the failing dependency. The caller IPs (10.20.10.167 and 10.20.11.8) are bot-service instances. Baseline also shows errors (16 in prior 50 min window) indicating bot-forge was intermittently failing before the sustained failure began at ~18:05-18:07. The root cause is bot-forge returning 500 INTERNAL SERVER ERROR.

bot-action receives 500 Internal Server Error from botforge.bot.internal for all bot IDs, sustained from 18:05 onward

Source: Cloudwatch

```
[2026-06-08T18:05:55Z] [2026-06-08 18:05:55,849] ERROR bot-action [!dt dt.trace_id=662fa404c9e9e48cea0619e5fdee028e,dt.span_id=2084bd f852560f51,dt.trace_sampled=true] bot-forge error for bot_id=bot-001: 500 Server Error: INTERNAL SERVER ERROR for url: http://botforge.bot.internal/bot/bot-001
[2026-06-08T18:07:55Z] [2026-06-08 18:07:55,887] ERROR bot-action [!dt dt.trace_id=d374849521d769bc1be559540256882a,dt.span_id=43a872 af61199d52,dt.trace_sampled=true] bot-forge error for bot_id=bot-001: 500 Server Error: INTERNAL SERVER ERROR for url: http://botforge.bot.internal/bot/bot-001
[2026-06-08T18:07:56Z] [2026-06-08 18:07:56,472] ERROR bot-action [!dt dt.trace_id=6a7cb61a258dbb5085fb2462639ad3b4,dt.span_id=bd700e 623c6b298f,dt.trace_sampled=true] bot-forge error for bot_id=bot-003: 500 Server Error: INTERNAL SERVER ERROR for url: http://botforge.bot.internal/bot/bot-003
[2026-06-08T18:07:56Z] [2026-06-08 18:07:56,775] ERROR bot-action [!dt dt.trace_id=d01f378ba427a7ab64cf0fd2e4ca0b76,dt.span_id=c96426 75880d2f17,dt.trace_sampled=true] bot-forge error for bot_id=bot-004: 500 Server Error: INTERNAL SERVER ERROR for url: http://botforge.bot.internal/bot/bot-004
```

bot-action returns 500 to bot-service (python-requests/2.32.3 caller) for all /bot/* endpoints

Source: Cloudwatch

```
[2026-06-08T18:06:55Z] 10.20.10.167 - - [08/Jun/2026:18:06:55 +0000] "GET /bot/bot-001 HTTP/1.1" 500 49 "-" "python-requests/2.32.3"
[2026-06-08T18:07:55Z] 10.20.11.8 - - [08/Jun/2026:18:07:55 +0000] "GET /bot/bot-001 HTTP/1.1" 500 49 "-" "python-requests/2.32.3"
```

bot-action showed bot-forge 500 errors in baseline too (16 errors in prior 50 min), indicating intermittent failures pre-existed

Roll back first, ask questions later

The screenshot shows the GitHub interface for a repository named 'bot-app' owned by 'Yanacek'. The repository is private. The main view displays a list of files and folders, including '.github/workflows', '.kiro/specs', 'bot-action', 'bot-cdk', 'bot-forge', 'bot-schedule', 'bot-service', '.gitignore', 'GITHUB_ACTIONS_SETUP.md', 'GRACEFUL_SHUTDOWN_FIX.md', 'README.md', 'S3_LOGGING_SETUP.md', 'SPLUNK_LOGGING_SETUP.md', 'build-deployment-package.sh', 'build-service-tar.sh', and 'deploy.sh'. A mouse cursor is hovering over the 'bot-service' folder. The right sidebar contains sections for 'About' (a demo service that pretends to control robots), 'Releases' (no releases published), 'Packages' (no packages published), and 'Languages' (TypeScript 59.0%, Python 30.1%, Shell 8.2%, JavaScript 2.7%).

Yanacek / bot-app

Code Issues Pull requests Actions Projects Security Insights Settings

bot-app Private

Watch 0 Fork 0 Star 0

main 1 Branch 0 Tags

Go to file Add file Code

Yanacek feat: add timestamp to cache record serialization 877ec4e · 2 hours ago 57 Commits

.github/workflows	fix workflow step order to compute code hash first before...	13 hours ago
.kiro/specs	feat: move to separate stacks with native autoscaling depl...	yesterday
bot-action	Adding logger to action service for ping shutdown	18 hours ago
bot-cdk	fix: make the launch templates and code bundles be imm...	13 hours ago
bot-forge	feat: add timestamp to cache record serialization	2 hours ago
bot-schedule	Fix ping handling import	last week
bot-service	Trying to do graceful shutdown and deployments	last week
.gitignore	feat: move to separate stacks with native autoscaling depl...	yesterday
GITHUB_ACTIONS_SETUP.md	Experiment with new thread pool configuration	last week
GRACEFUL_SHUTDOWN_FIX.md	Add s3 log destination for application logs	last week
README.md	feat: move to separate stacks with native autoscaling depl...	yesterday
S3_LOGGING_SETUP.md	add instrumentation to cache	last week
SPLUNK_LOGGING_SETUP.md	Add s3 log destination for application logs	last week
build-deployment-package.sh	feat: move to separate stacks with native autoscaling depl...	yesterday
build-service-tar.sh	Add github actions	last week
deploy.sh	Add github actions	last week

About

a demo service that pretends to control robots

Readme Activity 0 stars 0 watching 0 forks

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Languages

TypeScript 59.0% Python 30.1% Shell 8.2% JavaScript 2.7%

Fixed!

The screenshot displays the Dynatrace Services Explorer interface. The left sidebar contains navigation links for Dynatrace, Search, Apps, Dashboards, Problems, Notebooks, Vulnerabilities, Experience Vitals, Services, Kubernetes, Clouds, Infrastructure & Operations, Logs, and Distributed Tracing. The main content area is titled 'Services' and shows a list of services with their performance and health metrics. The 'Services' section is expanded, showing a list of services with their failure rates, response times, and throughput. The 'Health' tab is selected, and the 'Failure rate' column is highlighted. The table lists four services: bot-forge, bot-service, bot-action, and bot-schedule. Each service has a failure rate, response time p50, throughput, and health alerts. The 'bot-schedule' service has a failure rate of 67.5% and a response time of 11.84 ms. The 'Health alerts' column shows 'Failure rate' for all services. The 'Custom alerts' column shows '-' for all services. The 'bot-schedule' service has a red diamond icon next to its failure rate, indicating a critical issue. A mouse cursor is pointing at the 'bot-schedule' service row.

Service	Failure rate	Response time p50	Throughput	Health alerts	Custom alerts
bot-forge	49%	4.01 ms	100 /min	Failure rate	-
bot-service	54.44%	17.49 ms	90 /min	Failure rate	-
bot-action	52.73%	10.86 ms	55 /min	Failure rate	-
bot-schedule	67.5%	11.84 ms	40 /min	Failure rate	-

From build to operations — one continuous loop

We saw the loop. Next, we'll zoom in on Dynatrace — then put it into practice.



WHAT'S NEXT

Dynatrace deep dive



Hands-on workshop

See how Dynatrace adds observability, topology, and operational context — then apply the pattern yourself.



Thank you!

