

# DevStar

## Quarkus Developer Tooling Workgroup

Rethinking how developers — and AI agents —  
interact with Quarkus at dev time

Dev UI

Dev MCP

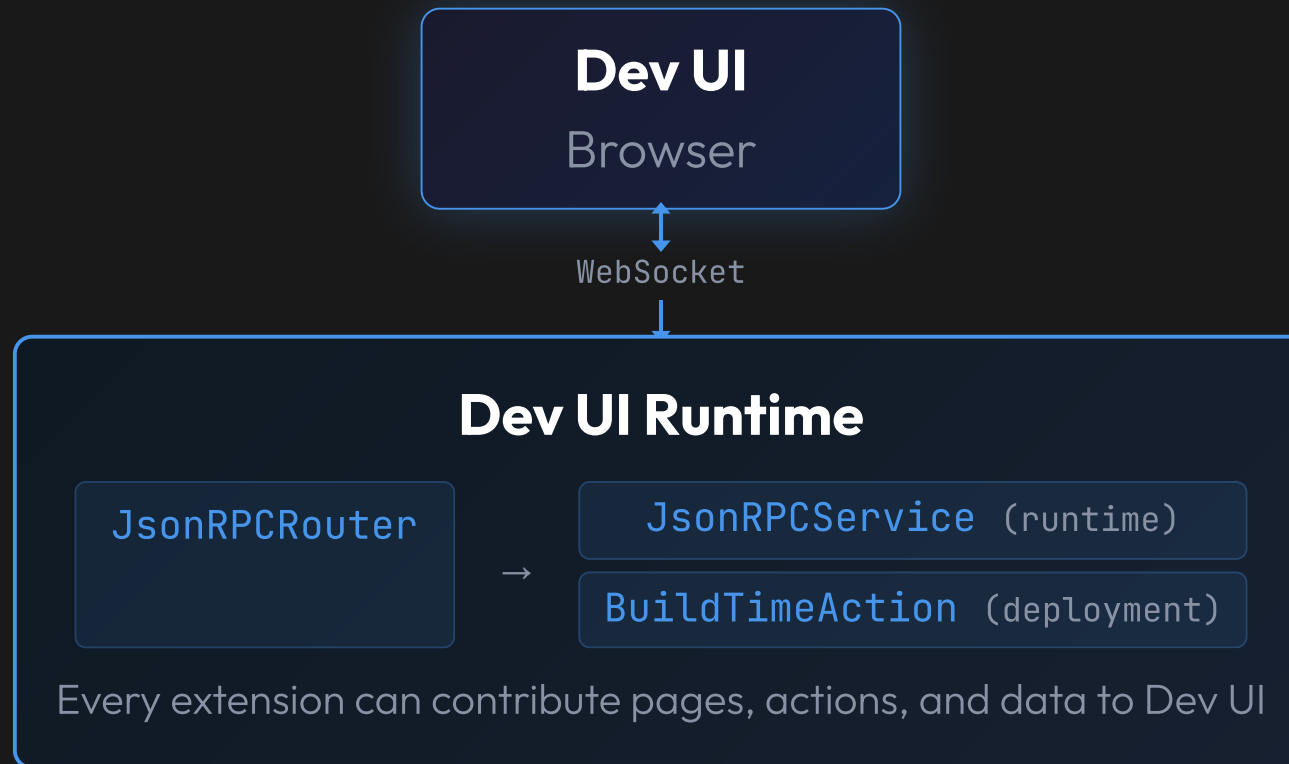
Dev Shell

Dev Assistant

Continuous Testing

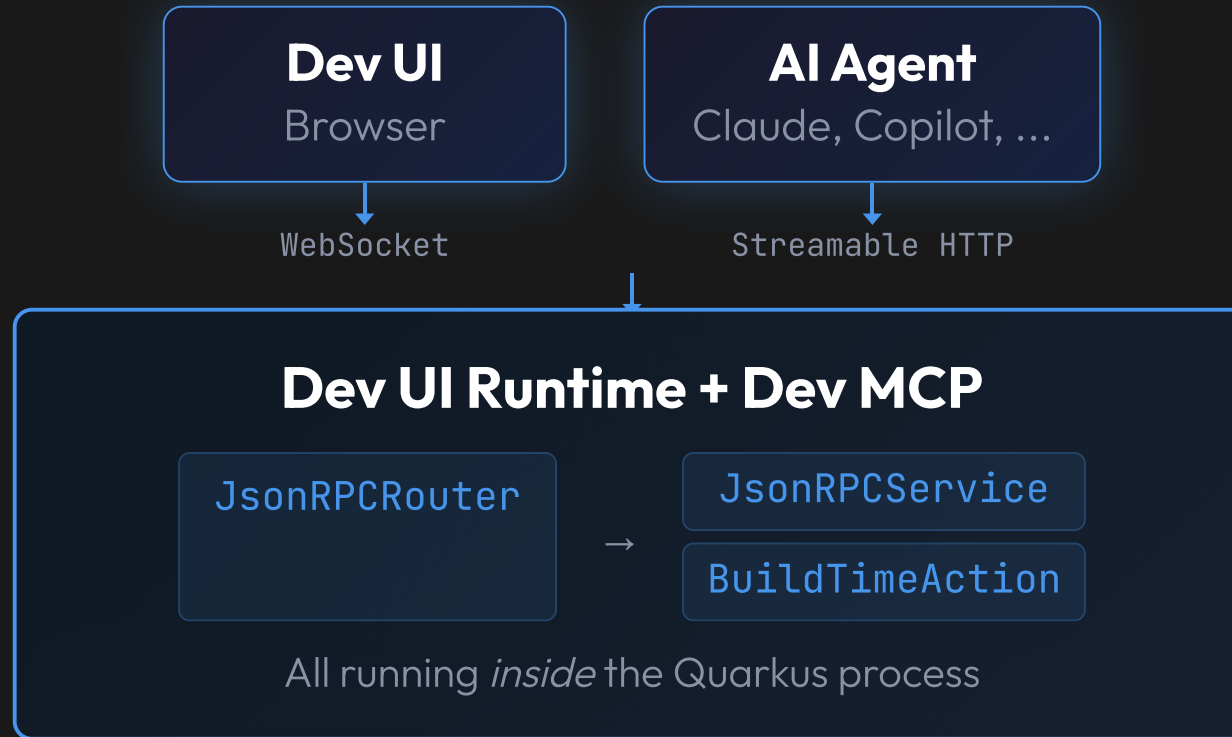
# Dev UI — The Foundation

Extensions expose functionality via JsonRPC services. Dev UI renders them in the browser.



# Dev MCP — Adding AI Agents

AI agents connect to the same JsonRPC backend via streamable HTTP — same tools, new consumer.



# Two Tracks of Work

The DevStar workgroup is tackling this from two directions



## Quarkus 4 — Dev UI refactor

Break Dev UI into smaller components so the JsonRPC backend can be reused independently. Dev MCP and Dev Shell can run *without* Dev UI.

Breaking for extensions

Not yet started



## Agent MCP — standalone server

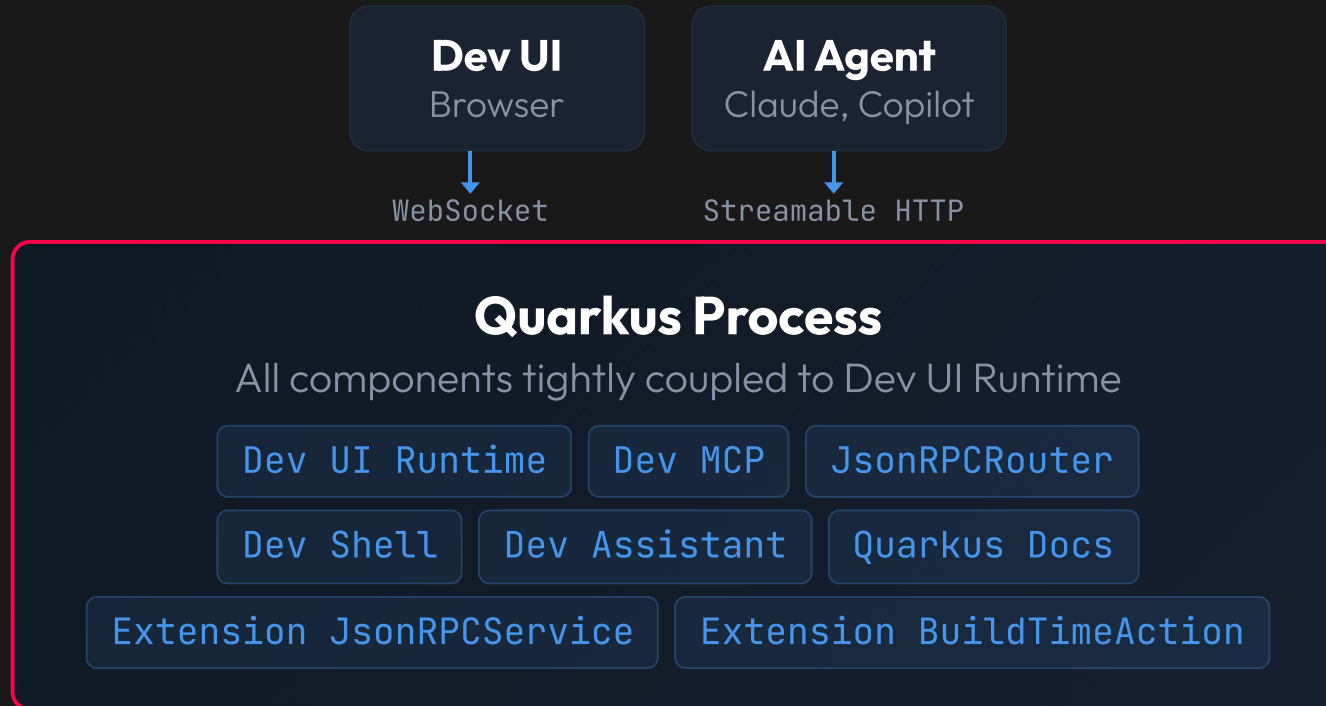
A separate MCP server that runs outside the Quarkus process. Manages lifecycle, skills, docs, and proxies to Dev MCP.

Nearly ready

Quarkus 3.35+

# Current State

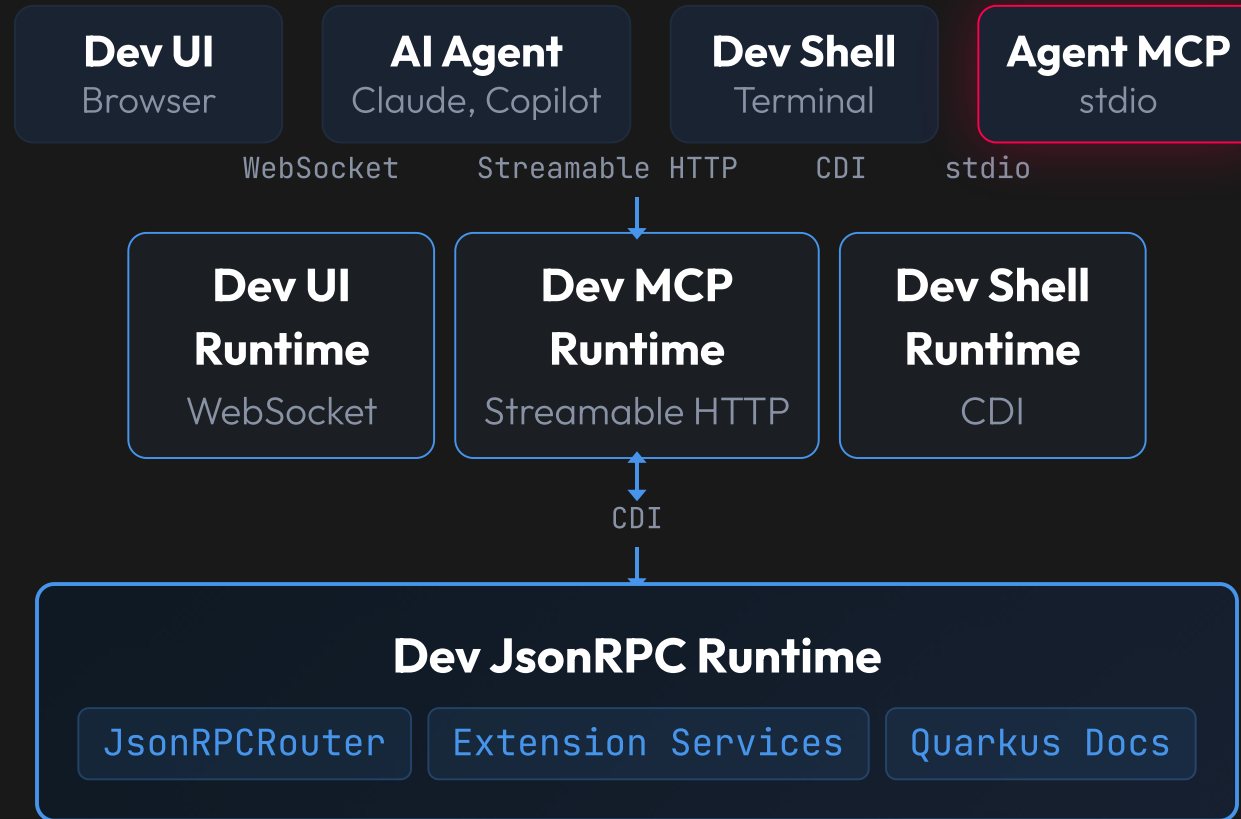
Everything lives inside the running Quarkus process — tightly coupled to Dev UI Runtime



⚠ App not running = no tools, no docs, no shell

# Quarkus 4 — Planned Refactor

Break apart into independent runtimes, each with its own transport — connected via CDI





# Quarkus Agent MCP

---

## AI-Powered Quarkus Development

An MCP server that lets AI coding agents  
create, manage, and develop Quarkus  
applications

Claude Code

GitHub Copilot

Cursor

Windsurf

# Dev MCP — Not Enough

Quarkus ships a Dev MCP server at `/q/dev-mcp` — but it has significant limitations



## Only when the app is running

Dev MCP lives inside the Quarkus process — no app, no tools. Can't scaffold, can't recover from crashes.



## Too many tools at once

Exposes every Dev UI method as a tool. Agents get overwhelmed by a flat list of 50+ tools with no context.



## Tools without knowledge

Agents can call tools but don't know *when* or *why*. No skills, no patterns, no best practices.



## No documentation access

Agents can't search Quarkus guides or API docs — they rely on stale training data instead.



# The Solution



## Purpose-built MCP server

Tools, skills, and docs designed specifically for Quarkus development



## Standalone process

Runs separately from the app — survives crashes, manages lifecycle



## Agent-agnostic

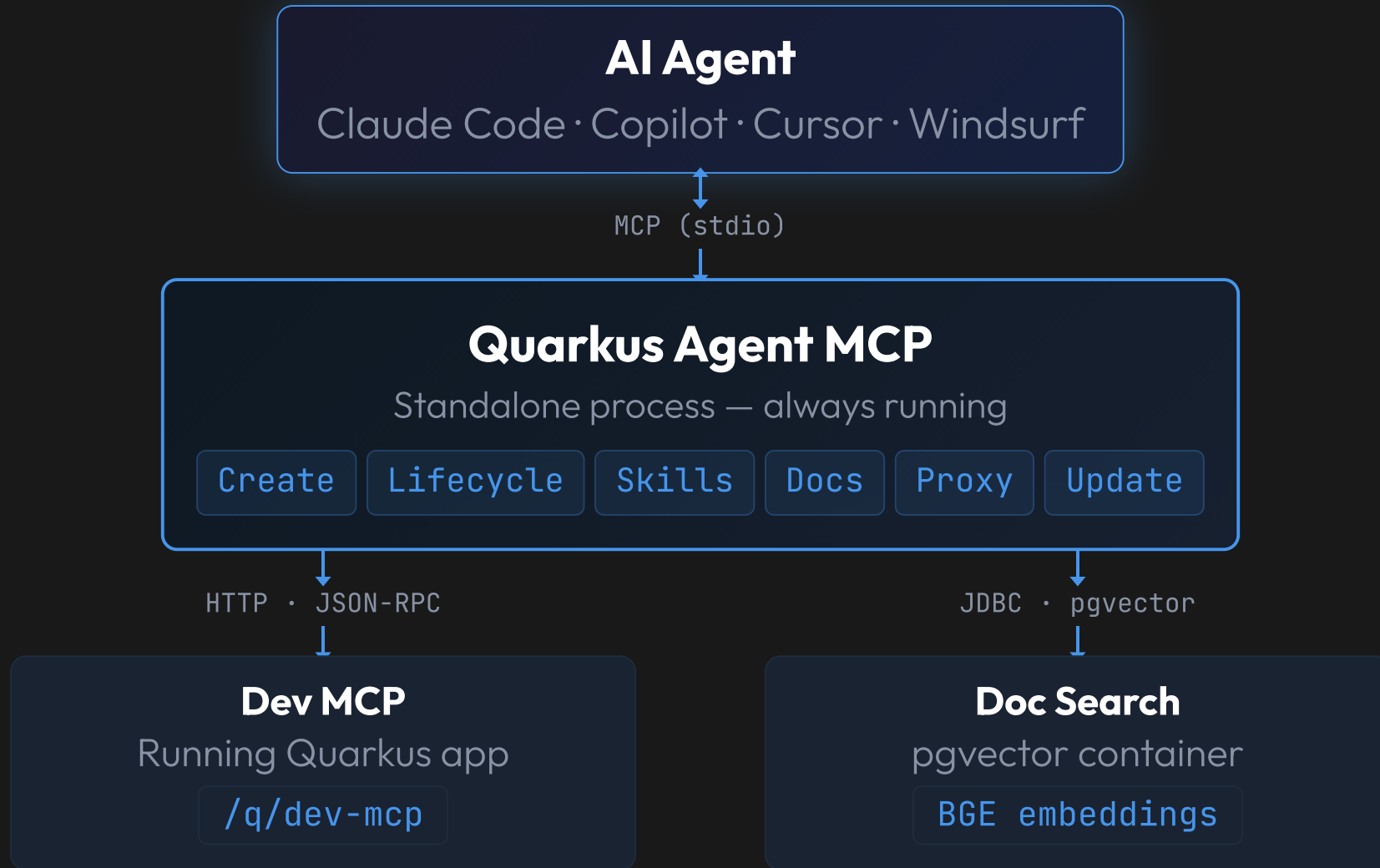
Works with any MCP-compatible agent via stdio protocol



## Extension skills + semantic docs

Teaches agents the right patterns, with version-aware documentation search

# Architecture



# The Tools



## Scaffolding

`quarkus/create` scaffold & auto-start



## Intelligence

`quarkus/skills` extension patterns

`quarkus/searchDocs` semantic doc search



## Lifecycle

`quarkus/start` dev mode

`quarkus/stop` graceful shutdown

`quarkus/restart` force restart

`quarkus/status` instance state

`quarkus/logs` recent output

`quarkus/list` all instances



## Dev Integration

`quarkus/searchTools` discover dev tools

`quarkus/callTool` invoke dev tools

---

`quarkus/update` version check

# Workflow — New Project

- 1 `quarkus/create` → Scaffolds project & auto-starts in dev mode
- 2 `quarkus/skills` → Load extension-specific patterns & pitfalls
- 3 `quarkus/searchDocs` → Look up APIs, config, and best practices
- 4 **Write code + tests**
- 5 `quarkus/callTool` → Run tests, reload, add extensions
- 6 Update `README.md`

# Workflow — Existing Project

- 1 `quarkus/update` → Check version & suggest upgrades `via subagent`
- 2 `quarkus/start` → Start in dev mode if not running
- 3 `quarkus/skills` → Load extension patterns before coding
- 4 **Write code + tests**
- 5 `quarkus/callTool` → Test & reload

# How Skills Work

Three-layer override chain — each layer overlays the previous by extension name

## Project-level skills

`src/main/resources/META-INF/skills/{ext}/SKILL.md`

Team conventions & project-specific patterns

WINS

## User-level skills

`~/.quarkus/skills/{ext}/SKILL.md`

Personal overrides & extension dev testing

OVERRIDES

## JAR skills (baseline)

`quarkus-extension-skills` artifact from Maven Central

BASELINE

Ships with Quarkus — curated by extension authors

# Skill Anatomy

Extension developers write plain Markdown. The build composes it with metadata into a full skill.

## EXTENSION DEVELOPER WRITES

deployment/.../META-INF/quarkus-skill.md

```
←!— Just plain Markdown, no frontmatter  
→
```

### ### REST Endpoints

- Annotate with @Path, @GET, @POST
- Return RestResponse<T> for type-safe responses

### ### Testing

- Use @QuarkusTest + REST Assured

### ### Common Pitfalls

- Resources default to @Singleton

→  
aggregate  
-skills

## COMPOSED OUTPUT **SKILL.md**

META-INF/skills/quarkus-rest/SKILL.md

```
--- ← YAML frontmatter added by build  
name: "quarkus-rest"  
description: "Build RESTful web  
services..."  
license: "Apache-2.0"  
metadata:  
  guide:  
    "https://quarkus.io/guides/rest"  
---
```

### ### REST Endpoints

- Annotate with @Path, @GET, @POST
- Return RestResponse<T> for type-safe responses

```
runtime/.../quarkus-extension.yaml
```

```
name: "REST"
description: "Build RESTful web
services..."
metadata:
  guide:
    "https://quarkus.io/guides/rest"
```

### ### Testing

- Use @QuarkusTest + REST Assured

### ### Common Pitfalls

- Resources default to @Singleton



# Doc Search — Semantic Search



**Pre-indexed docs** — Quarkus documentation in a Docker container, ready to search



**Version-aware** — Matches container image to your project's Quarkus version



**Synonym expansion** — 28 mappings: `db → datasource` `auth → authentication`

`orm → hibernate`



**Metadata boosting** — Re-ranks results by title and path relevance

# Dev MCP Proxy

Agent MCP proxies to the running Quarkus app's built-in Dev MCP server



## Example Dev Tools

devui-testing\_runTests   devui-testing\_runTest   devui-extensions\_add  
devui-config\_getAll   devui-exceptions\_getLastException  
devui-logstream\_forceRestart

Filter by query: **testing**   **extension**   **config**   **exception**

# Key Design Decisions



## Standalone Process

Survives app crashes. Agent MCP keeps running even when the Quarkus app fails.



## Lazy Initialization

pgvector container starts on first doc search. No upfront cost if not needed.



## Version-First

Docs, skills, and containers are all version-aware. Matches your project exactly.

# For Extension Developers

How to make your extension work great with AI agents

- 1 **Build a Quarkus app** using Agent MCP with your extension
- 2 **Use a small model** — if it works with a budget LLM, it works everywhere

Haiku

GPT-4o  
mini
- 3 **Note what the agent gets wrong** — wrong patterns, missed conventions, bad defaults
- 4 **SKILL.md** → **Write a skill override** to fix those mistakes. Ask the agent to write it — it just learned the correct approach.
- 5 **Dev MCP** → **Check Dev MCP tools** — does the agent use them? What's missing or needs changing?
- 6 **PR against Quarkus** — add the skill + Dev MCP changes upstream

# Demo Time

Let's see it in action

- ▶ Create a new Quarkus app from a natural language prompt
- ▶ Watch skills guide the agent through extension-first development
- ▶ See semantic doc search find relevant Quarkus guides
- ▶ Run tests via Dev MCP proxy without leaving the conversation
- ▶ Hot reload and iterate in real time

# Thank You

Questions?

## GitHub

quarkusio/quarkus-agent-  
mcp

## Working Group

Quarkus  
DevStar

## Install

jbang quarkus-agent-  
mcp@quarkusio