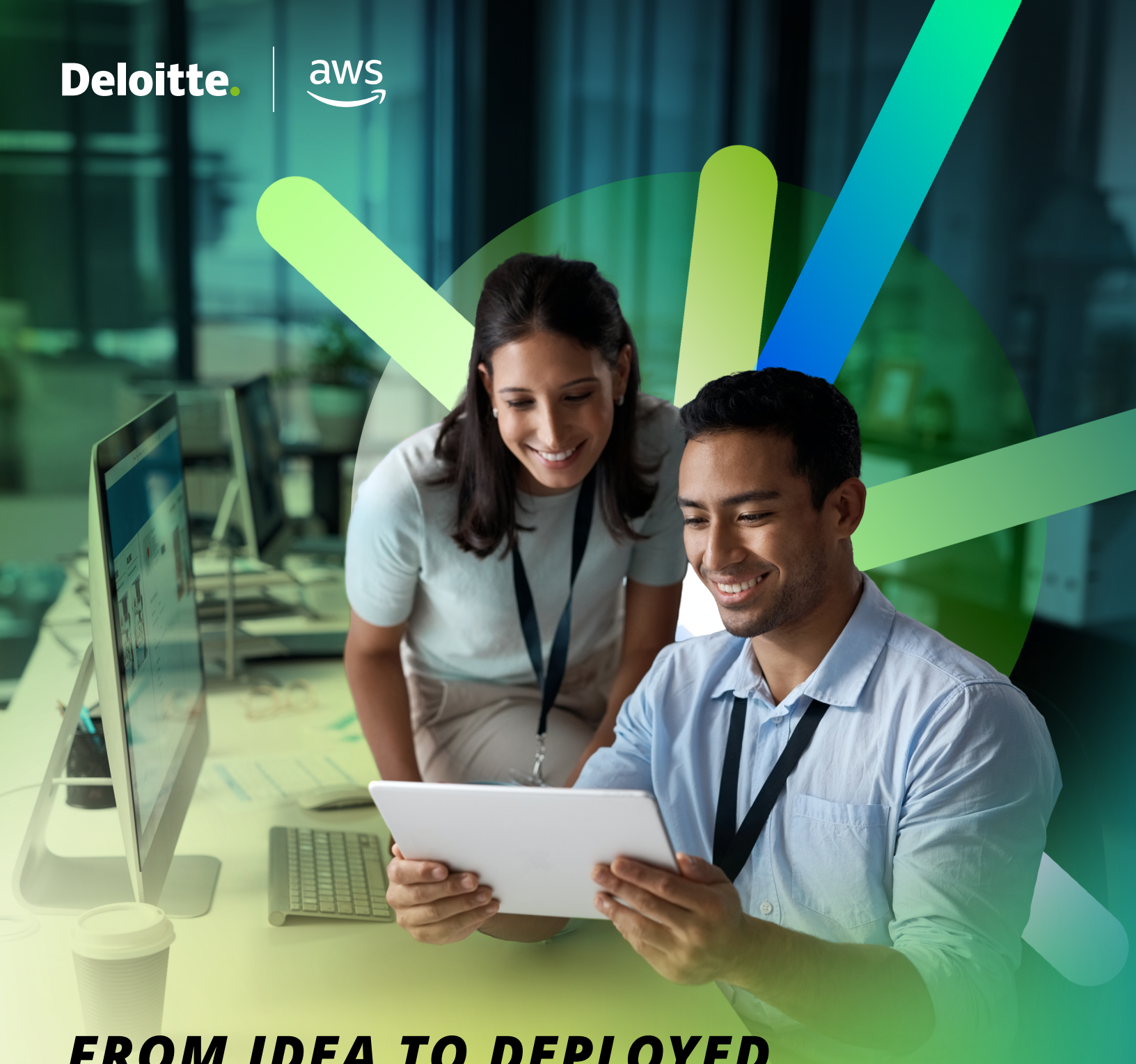
Deloitte. | aws



FROM IDEA TO DEPLOYED PROTOTYPE IN A SINGLE SESSION: *RAPID PROTOTYPING WITH CLAUDE CODE AND AMAZON BEDROCK*

How AI-assisted development turns weeks of work into hours—and why that changes the economics of innovation.

By **Bojan Ciric**, Technology Fellow, Deloitte & **Kasi Muthu**, Global AI Solutions Architect, AWS

THE INNOVATION BOTTLENECK

Every enterprise has a backlog of ideas that never make it past the whiteboard. A wealth management platform that adapts to client personalities. A monitoring dashboard that predicts failures before they happen. A client portal that makes complex financial concepts accessible.

These ideas die not because they lack merit, but because the cost of proving them is too high. Building a prototype traditionally means assembling a team, allocating a sprint (or three), and hoping the result is compelling enough to justify the investment. By the time the prototype is ready, the window of opportunity may have closed, priorities have shifted, or stakeholders have moved on.

Rapid prototyping breaks this bottleneck. When you can go from a concept document to a working, deployed application in hours instead of weeks, you fundamentally change the economics of innovation. Ideas that were too risky to explore become low-cost experiments. Stakeholder conversations shift from "imagine if we could..." to "let me show you what this looks like."

This article walks through exactly how we did this—turning a 46-page advisory playbook into a fully functional, multi-persona wealth management prototype, deployed on AWS, in a single working session.

THE STACK: CLAUDE CODE ON AMAZON BEDROCK

Claude Code is Anthropic's agentic coding tool—a CLI and IDE-integrated assistant that can read files, write code, execute commands, search codebases, and orchestrate multi-step development workflows. Unlike a chatbot that suggests code snippets, Claude Code operates directly in your development environment: it reads your project files, understands your architecture, runs your build tools, and iterates on errors until the code compiles and runs.

When Claude Code runs on Amazon Bedrock, you get the capability of Anthropic's Claude models within the AWS ecosystem—with enterprise-grade security, no data leaving your AWS environment, and seamless integration with the deployment and infrastructure services you already use.

This combination creates a uniquely powerful prototyping workflow:

CAPABILITY	WHAT IT ENABLES
Document comprehension	Feed Claude Code a PDF, specification, or design doc—it extracts requirements directly
Full-stack code generation	Produces TypeScript, React components, state management, data models, and styling in one pass
Iterative error correction	Runs the build, reads the errors, fixes them, and re-runs—autonomously
Multi-agent parallelism	Spawns background agents to build data layers, pages, and components concurrently
Deployment orchestration	Executes AWS CLI commands to create infrastructure and deploy artifacts

> *The choice of Amazon Bedrock as the LLM provider deserves its own examination—see the next section for a deeper look at what this integration delivers.*

The human stays in the loop at every decision point—reviewing architecture choices, approving file changes, and steering direction—while Claude Code handles the volume of code that would otherwise require a team.

WHY AMAZON BEDROCK AS THE FOUNDATION

Running Claude Code through Amazon Bedrock isn't just a deployment choice, it's an enterprise architecture decision. Here's what this integration delivers beyond using Claude Code with Anthropic's direct API:

Data sovereignty and security

Your prompts, code, and proprietary documents are processed within AWS Bedrock under a user's account controls, and Anthropic does not receive or train prompts, code, or documents. With Bedrock, data is processed within your VPC, governed by your IAM policies, and encrypted with your KMS keys. For regulated industries—financial services, health care, government—this isn't optional, it's table stakes. In our Wealth Advisor example, the 46-page advisory playbook contained proprietary investment strategies. Bedrock ensured that intellectual property stayed within the organization's security boundary.

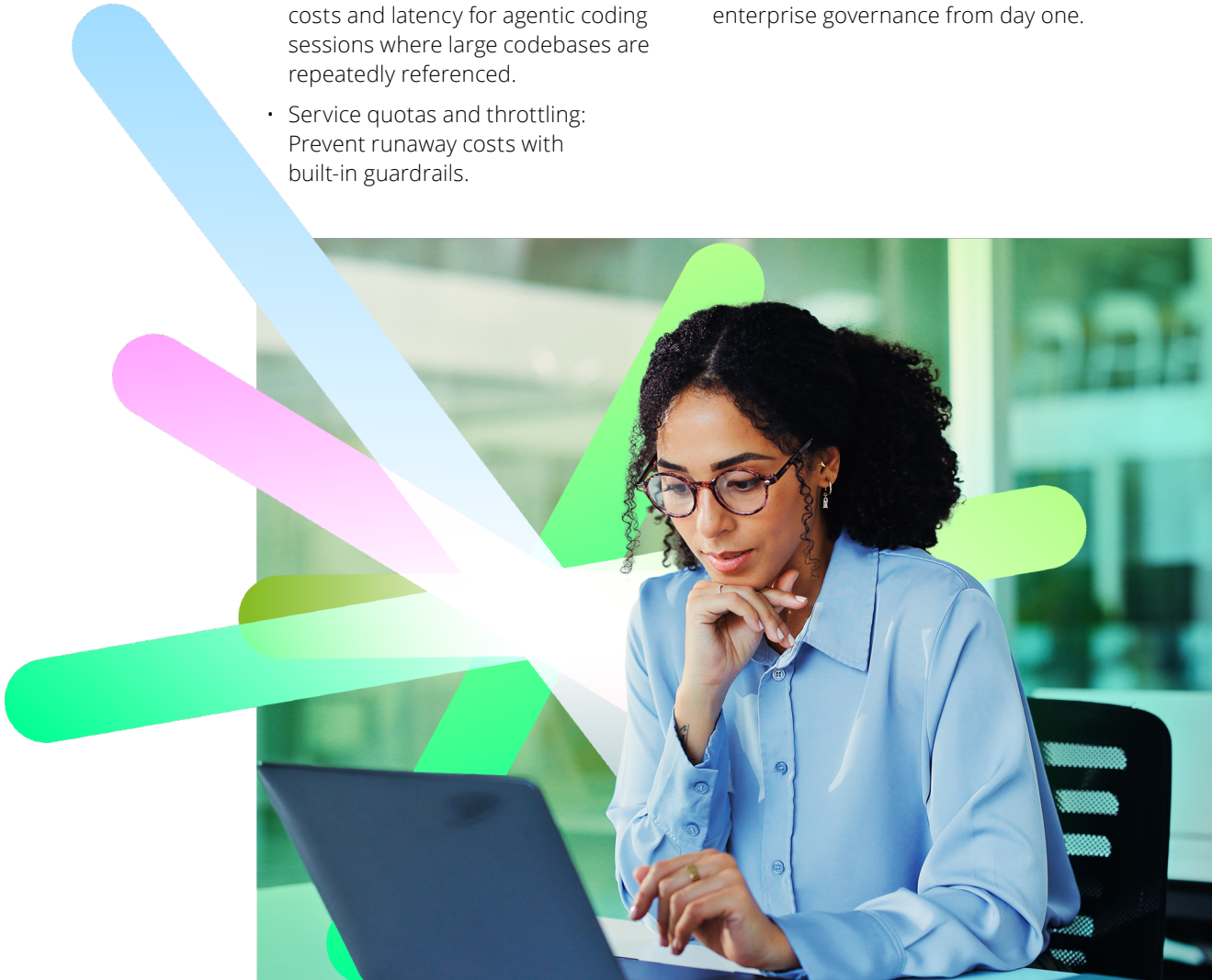
Operational control

- IAM-based access control: Govern who can use Claude Code and which models they can access, using the same identity framework your organization already manages.
- CloudWatch integration: Monitor token usage, latency, and costs in real time alongside your other AWS workloads.
- Prompt caching: Amazon Bedrock's prompt caching significantly reduces costs and latency for agentic coding sessions where large codebases are repeatedly referenced.
- Service quotas and throttling: Prevent runaway costs with built-in guardrails.

Getting started

1. Configure IAM permissions: Ensure your developer role has bedrock:InvokeModel permissions for the relevant model ARNs.
2. Point Claude Code to Bedrock: Set the CLAUDE_CODE_USE_BEDROCK=1 environment variable and configure your AWS region. Claude Code handles the rest.

A developer can go from zero to prototyping in minutes, with full enterprise governance from day one.



THE CASE STUDY: WEALTH ADVISOR

The starting point

We began with a single artifact: a 46-page advisory playbook PDF describing a wealth management framework. The document covered investment principles (Discipline, Diversification, Time, Risk & Return), a six-step advisory process, portfolio construction methodology (Strategic and Tactical Asset Allocation), client profiling, and the GIC House View cascade.

The goal: turn this document into a working, interactive prototype that a business stakeholder could click through and experience—not a slide deck, not a wireframe, but a real application.

The proof: A fully compiling, navigable application—not a mockup, not a design file—running on localhost.

Step 1: Specification extraction (20 minutes)

The first prompt fed the PDF directly to Claude Code:

> "Create a specification for a Wealth Advisor prototype in Next.js based on this document. Cover all concepts from the PDF. Include functional and technical requirements."

Claude Code read the entire 46-page document, extracted every concept, and produced a comprehensive SPECIFICATION.md—14 sections covering layout, navigation, user profiles, dashboards, the advisory process, portfolio construction, monitoring, and educational content.

We then asked it to validate: "Did you align requirements with every concept in the PDF?" It performed a concept-by-concept audit, identified 20 gaps (missing GIC cascade levels, the foundation vs. opportunistic portfolio split, the 4-layer monitoring framework), and fixed all of them.

The proof: A traceable specification where every requirement maps back to a specific page in the source document.

Step 2: Full prototype build (90 minutes)

With the specification locked, the next prompt was straightforward:

> "Use the specification and create the prototype."

Claude Code scaffolded a Next.js 14 application with TypeScript, Tailwind CSS, and DaisyUI, then built the entire application:

- 11 interactive pages: Dashboard, discovery (CIP builder), house view (4-tab GIC showcase), portfolio (holdings, rebalancing, stress testing), markets (indices, news, calendar), watchlist, trades, reports (performance, risk, health, 6-step review), goals (projections, RSP analysis), Wealth Academy (principles, glossary), and settings.
- 15 data files: Mockup data for securities, portfolios, market indices, news, alerts, research, glossary terms, and investment principles.
- 7 interactive chart types: Line charts for performance and projections, bar charts for allocation comparison, pie/donut charts for asset mix, scatter plots for risk-return analysis—all built with Recharts.
- AI advisor panel: A slide-in chat interface with a scripted advisor ("Mandy") that responds to portfolio, goal, and market queries.
- State management: Zustand store for theme, sidebar, active user, chat messages, and alerts.
- Dark and light themes: Custom DaisyUI themes with full semantic color tokens.
- Prompt caching: Amazon Bedrock's prompt caching significantly reduces costs and latency for agentic coding sessions where large codebases are repeatedly referenced.
- Service quotas and throttling: Prevent runaway costs with built-in guardrails.

Claude Code parallelized the work—spawning background agents to build the data layer, utility libraries, and page components simultaneously while building the layout and dashboard directly. It ran the build, encountered TypeScript errors (Recharts tooltip formatter type conflicts, unused imports, store type mismatches), diagnosed each one, and fixed them iteratively until all 11 routes returned HTTP 200.

THE CASE STUDY: WEALTH ADVISOR

Step 3: Persona-driven data (60 minutes)

The initial prototype had a single hardcoded user. The next request:

> "When I switch personas, the data is not changing."

Claude Code audited every page and component, identified that all data was inline-hardcoded (not connected to the store), and executed a systematic refactoring:

1. Created persona-data.ts: a comprehensive PersonaData interface with holdings, allocations, goals, trades, alerts, risk metrics, health components, watchlist, and settings for each persona.
2. Built usePersona(): a hook that reads the active user from the Zustand store and returns the matching data set.
3. Refactored all 11 pages and 4 dashboard components to source data from the hook instead of hardcoded values.

Three complete personas with distinct financial profiles:

PERSONA	SEGMENT	RISK PROFILE	PORTFOLIO	KEY CHARACTERISTIC
David Liu, 60	Affluent	Moderate	$312,500	Property goal, China exposure concern
Sarah Chen, 35	Priority	Aggressive	$518,400	ESG focus, private equity, high growth
James Park, 28	Retail	Conservative	$52,400	First-time investor, savings plan

Switching personas in the top-right dropdown instantly updates every number, chart, table, alert, and recommendation across the entire application.

The proof: Click the avatar, select a different persona, watch the dashboard transform—portfolio value, asset allocation, health score, alerts, goals, and performance charts all reflect the new client.

Step 4: Deployment to AWS Amplify (5 minutes)

The final step:

> "Deploy this to AWS. Use Amplify, deploy by zip and upload."

Claude Code executed the full deployment pipeline:

1. Configured Next.js for static export (output: 'export').
2. Built the production bundle: 15 static pages, 747KB compressed.
3. Created an Amplify app via the AWS CLI (aws amplify create-app).
4. Created a production branch, obtained a pre-signed upload URL.
5. Uploaded the zip and started the deployment.
6. Added SPA rewrite rules for client-side routing.
7. Verified the deployment returned HTTP 200.

Total time from "deploy to AWS" to a live URL: under 5 minutes. No CI/CD pipeline to configure, no Docker containers to build, no infrastructure-as-code to write. The prototype was live and shareable.

The proof: A public Amplify URL that anyone with the link can open in a browser and experience the full application.

With the prototype live and shareable, it's worth being explicit about what this output is—and what it isn't.

THE PROTOTYPE BOUNDARY: WHY HUMANS STAY IN THE DRIVER'S SEAT

Let's be direct: the code Claude Code generates is prototype-grade, not production-grade. That's not a limitation—it's the point. The value proposition is speed-to-insight, not speed-to-production.

Here's what prototype code typically lacks that production demands:

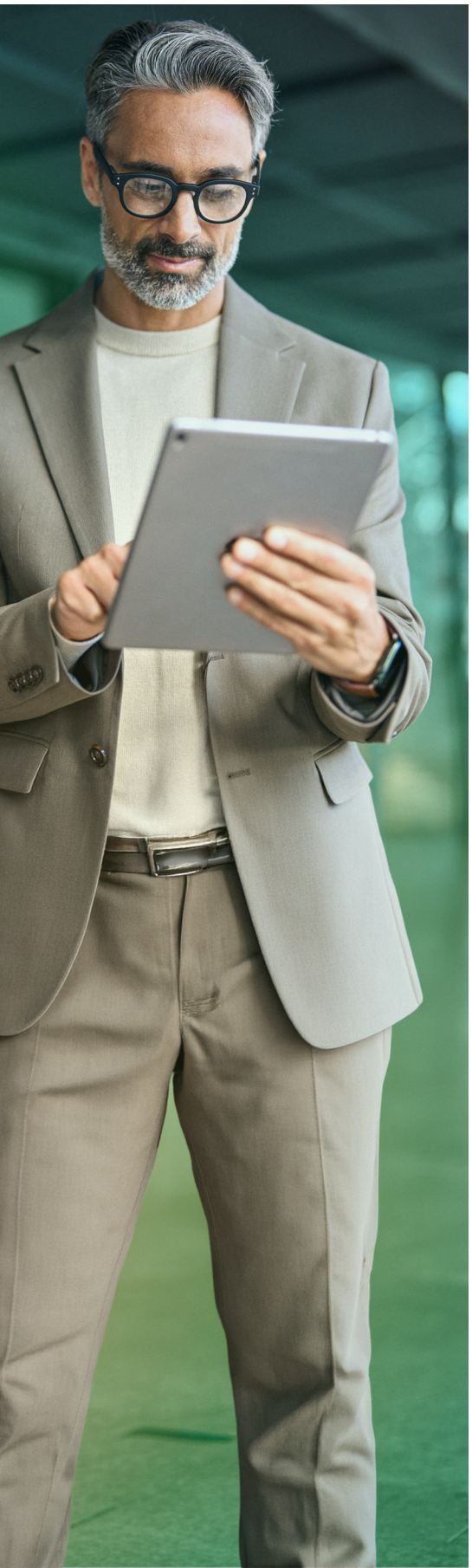
CONCERN	*PROTOTYPE REALITY*	*PRODUCTION REQUIREMENT*
Security	Hardcoded configs, no auth flows	OAuth/OIDC, secrets management, penetration testing
Error handling	Happy-path focused	Graceful degradation, retry logic, circuit breakers
Data layer	Static JSON files, mock data	Database design, migrations, backup/recovery
Testing	Minimal or none	Unit, integration, E2E, load testing
Compliance	Not considered	SOC 2, HIPAA, GDPR depending on domain
Observability	Console logs	Structured logging, distributed tracing, alerting
Scalability	Single-user assumptions	Concurrency, caching, CDN, auto-scaling

In our Wealth Advisor example, Claude Code generated 15 data files with static JSON—perfect for demonstrating portfolio allocation concepts to stakeholders, but a production system needs real-time market data feeds, secure API integrations with custodians, and regulatory-compliant audit trails.

The right mental model: Claude Code is your co-pilot for the first 80% of the journey—the exploration, the visualization, the "what if." A human engineer is essential for the last 20% that determines whether the system is secure, reliable, and maintainable. The prototype tells you what to build. Humans decide how to build it right.

This is precisely why rapid prototyping is so powerful: by collapsing the cost of exploration, it frees your best engineers to focus on the hard problems that actually require human judgment—architecture decisions, security posture, data modeling, and operational excellence.

USING PROTOTYPES FOR STAKEHOLDER BUY-IN



A deployed prototype is not a deliverable, it is a conversation tool. Its purpose is to make abstract concepts tangible so that decision-makers can react to something real rather than something imagined.

What a prototype proves

QUESTIONS STAKEHOLDERS ASK	*WHAT THE PROTOTYPE ANSWERS*
What would this actually look like?	Click through 11 pages of working UI
Would it work for different client types?	Switch between three personas live
How would the advisory process translate digitally?	Walk the Discovery wizard from personality quiz to CIP summary
Can we visualize portfolio risk?	Show stress testing: "A COVID replay costs David $46,875"
What about the House View methodology?	Four tabs: Principles, CIO Outlook, Themes, Performance Evidence
How quickly could we build this?	"You're looking at it. It was built and deployed today."

The demo script pattern

We generated a complete demo script (DEMO_SCRIPT.md)—a 20-minute guided walkthrough written for non-technical presenters. It includes exact click paths, talking points for each section, a persona cheat sheet, and presenter tips ("Start in dark mode—it looks more polished on a projector").

This is the artifact that travels. The developer who built the prototype is not the person who presents it to the investment committee. A well-written demo script bridges that gap.

From "interesting" to "funded"

The economics are what make this approach powerful. When a prototype costs weeks of team effort, you build one, maybe two, and they need to be right. When a prototype costs a single session, you can build five variations, test them with different stakeholder groups, and iterate on the one that resonates.

This shifts the risk profile of innovation. Instead of betting large on a single concept, you run multiple low-cost experiments. The prototypes that generate excitement get funded. The ones that don't get archived, at near-zero sunk cost.

> *The economics become even clearer when you compare the traditional path to the rapid prototyping approach. See the Economics section.*

NEXT STEPS: FROM PROTOTYPE TO PRODUCTION

A prototype validates the idea. Production validates the architecture. The same Claude Code and Amazon Bedrock pattern that built the prototype can be applied—with human-in-the-loop oversight at each stage—to build the real thing.

Here is the path forward:

PHASE 1 • **Define the architecture**

The prototype was a static, client-only application. The production system needs:

- Backend services: API Gateway + Lambda or ECS containers for portfolio calculations, trade execution, market data feeds, and CIP management.
- Data layer: Amazon DynamoDB or Amazon Aurora for client profiles, portfolio holdings, and transaction history. Amazon S3 for document storage.
- Authentication: Amazon Cognito for client and advisor login, MFA, and role-based access.
- Real-time data: Amazon EventBridge or WebSocket APIs for live market data and alert delivery.
- AI/ML layer: Amazon Bedrock for the conversational AI advisor (replacing scripted responses with actual Claude model calls), plus Amazon SageMaker for portfolio analytics and risk models.

Claude Code can generate the architecture diagram, CloudFormation/CDK templates, and API contracts from the prototype's existing data shapes. The PersonaData interface we built—with its holdings, allocations, goals, trades, alerts, and risk metrics—becomes the starting point for the real API response schemas.

PHASE 2 • **Build the backend (human-in-the-loop)**

Using Claude Code on Bedrock, the development team builds each service with the same iterative pattern:

1. Describe the service: "Build a portfolio service that calculates real-time allocation, drift, health score, and rebalancing recommendations based on holdings and the TAA model."
2. Review the generated code: Claude Code produces the Lambda functions, DynamoDB table schemas, and API Gateway routes. The developer reviews every file, validates business logic, and approves or adjusts.
3. Test iteratively: Claude Code writes unit tests, runs them, and fixes failures. The developer adds integration tests against real data.
4. Security review: IAM policies, input validation, encryption at rest and in transit; the developer verifies each layer before deployment.

The human-in-the-loop is not a formality. It is the mechanism that ensures business logic is correct, security is sound, and edge cases are handled. Claude Code accelerates the implementation; the engineer ensures the quality.

PHASE 3 Evolve the frontend

The prototype's React components, page structure, and persona system become the foundation of the production frontend. Key changes:

- Replace mockup data imports with API calls (React Query or SWR for data fetching and caching).
- Add error states, loading skeletons, and offline handling for the PWA.
- Integrate Amazon Cognito for authentication flows.
- Connect the AI advisor panel to Amazon Bedrock's Claude API for real conversational intelligence.
- Add real-time WebSocket subscriptions for market data and alert notifications.

The prototype's component structure, naming conventions, and UX patterns carry forward. The team is not starting over, they are upgrading.

PHASE 4 Deploy on AWS (Production Grade)

The Amplify manual deployment that took 5 minutes becomes a full CI/CD pipeline:

COMPONENT	*AWS SERVICE*	*PURPOSE*
Frontend hosting	AWS Amplify or Amazon CloudFront + S3	Global CDN, custom domain, SSL
API layer	Amazon API Gateway + AWS Lambda	Serverless backend APIs
Database	Amazon DynamoDB / Aurora	Persistent data storage
Authentication	Amazon Cognito	User identity and access management
AI/ML	Amazon Bedrock (Claude)	Conversational advisor, document analysis
Monitoring	Amazon CloudWatch	Logs, metrics, alarms
Infrastructure	AWS CDK	Infrastructure as code, repeatable deployments

Claude Code can generate the CDK stacks, write the deployment scripts, and configure the CI/CD pipeline—all reviewed and approved by the engineering team before execution.

PHASE 5 Iterate in production

The rapid prototyping pattern does not end at launch. New features—a tax-loss harvesting module, an ESG scoring dashboard, a client onboarding flow—follow the same cycle:

1. Describe the concept.
2. Claude Code generates a working prototype branch.
3. Stakeholders review and provide feedback.
4. The approved version is refined, tested, and merged.

The prototype-to-production pipeline becomes a permanent capability, not a one-time event.

WHAT WE BUILT: BY THE NUMBERS

Source artifact
46-page advisory playbook PDF

Specification
14-section requirements document, validated concept-by-concept

Application
11 interactive pages, 15 data files, 7 chart types

Personas
3 complete client profiles with distinct data across every page

Components
12 React components (layout, dashboard, common)

State management
Zustand store with theme, user, chat, and alert state

AI advisor
Conversational panel with scripted responses

Deployment
AWS Amplify, static export, live URL

Demo script
20-minute guided walkthrough for non-technical presenters

Total time
One working session

THE ECONOMICS OF RAPID PROTOTYPING

When prototyping costs drop by two orders of magnitude, the innovation calculus fundamentally changes. You stop asking "which idea should we bet on?" and start asking "why aren't we testing all of them?"

METRIC	*TRADITIONAL APPROACH*	*CLAUDE CODE + BEDROCK*
Time to first prototype	4–6 weeks	3 hours
Team required	2–3 engineers + designer	1 person with domain knowledge
Cost estimate	$30K–$80K (loaded labor)	~$50–$100 in API costs
Iterations before stakeholder review	1 (maybe 2)	3–5 in the same session
Risk if idea doesn't work	Significant sunk cost	Negligible

When a prototype costs weeks of team effort, organizations build one, maybe two, and they need to be right. When a prototype costs a single session, you can build five variations, test them with different stakeholder groups, and fund the one that resonates. The ones that don't get archived at near-zero sunk cost. This is how the economics of innovation change.

THE COST OF NOT TRYING

The most expensive prototype is the one you never build.

Every organization has ideas waiting for validation. Strategic initiatives that need a proof of concept before the board will approve funding. Modernization visions that stakeholders cannot evaluate from a slide deck. Client experiences that sound promising but have never been seen or touched.

Claude Code on Amazon Bedrock compresses the distance between idea and evidence. Not by cutting corners—the prototype we built has real data models, real state management, real interactive charts, and real deployment infrastructure—but by eliminating the calendar time and coordination overhead that traditionally makes prototyping expensive.

The question is no longer "Can we afford to build a prototype?" It is "Can we afford not to?"

Start with a document. End with a deployed application. Let the prototype make the argument.

> The Wealth Advisor prototype described in this article was built using Claude Code powered by Amazon Bedrock, deployed on AWS Amplify, and uses Next.js, TypeScript, Tailwind CSS, DaisyUI, Recharts, and Zustand. The full source code, specification, and demo script were generated in a single development session.

CONTACT US

Bojan Ciric
Technology Fellow
Deloitte Consulting LLP
bciric@deloitted.com

Kasi Muthu
Global Data & AI Solutions Architect
Amazon Web Services (AWS)
kasimuth@amazon.com