Project Mantle: A Case Study in AI-First Development at AWS

How a small team achieved 10x development velocity building Amazon Bedrock's inference engine

January 2026

Project Mantle: A Case Study in AI-First Development at AWS

TL;DR: A team of ~9 senior engineers built a new inference engine for Amazon Bedrock using AI-first development, achieving 10x commit velocity. The project started in May 2025 and covered routing/load balancing—not all of Bedrock.

Key success factors:

- Experienced team composition — 1 Distinguished Engineer + 8 Principal Engineers from EC2, not junior developers
- Physical co-location — Entire team in one area for high-bandwidth, low-latency communication
- Local integration testing — Full stack runnable locally with fast feedback loops (minutes, not 20-30 min)
- Human accountability — Engineers responsible for every line of AI-generated code, no autonomous agents
- Context window discipline — Clear context between requests; use files as persistent memory for multi-step work
- Calibrated prompting — Find the "maximally supportable request"—not too ambitious, not too trivial
- Reduce ambiguity first — Brainstorm approaches with AI before implementation to eliminate dead ends
- Testing investment — 25% of team effort dedicated to testing, tooling, and development practices
- Async development — Queue prompts overnight; check results in the morning
- Minimal meetings — Only meetings where you're a sponsor, decision-maker, or implementer

Introduction

In January 2026, Joe Magherimov, VP and Distinguished Engineer at AWS, sat down for the AWS Podcast to discuss something that had been generating significant internal buzz: Project Mantle. What made this conversation remarkable wasn't just the technical achievement—a new inference engine powering Amazon Bedrock—but the methodology behind it. Magherimov's team had adopted an AI-first development approach and claimed to have achieved 10x development velocity as a result.

This case study examines the Mantle project through Magherimov's firsthand account, exploring the techniques, team dynamics, and hard-won lessons that made it possible. It also incorporates perspectives from other Amazon employees who provided additional context in public discussions.

Background: The Engineer Behind Mantle

Joe Magherimov brings an unusual depth of experience to the conversation. Over twenty years at Amazon, his career divides neatly into two halves. The first decade was spent building the systems behind amazon.com—shipping, payments, and marketplace infrastructure. The second decade moved into AWS, where he worked on foundational services: VPC, load balancers, NAT gateways, and later the container and serverless stack including ECS, Lambda, and EKS.

This background matters. When Magherimov talks about building for scale and reliability, he's drawing on experience with systems processing trillions of packets in real-time. "Complexity tends to be the enemy of reliability," he notes. "There's this constant tension behind engineering for scale versus engineering for reliability, and you have to navigate that tension."

At Amazon, engineers receive an acrylic puzzle piece for each patent they're granted. The physical evidence of Magherimov's contributions sits behind him during the interview: approximately 72 pieces—one for each patent—arranged in a 6×6×2 cube. "The whole concept of stacking those pieces and building out of them," he reflects, "that's really what it feels like. I'm a builder at heart."

The Problem: Inference as a Scheduling System

Project Mantle emerged from a key insight that came after years of operating Amazon Bedrock at scale. As Magherimov explains: "At the heart of it, inference is not quite a web service but more of a scheduling system."

When a request arrives at Bedrock, the system must handle concerns typically associated with scheduling: prioritization, fairness, placement, and co-placement of multiple requests. The original architecture, built quickly to meet market demands, had accumulated significant technical debt. (As one former Bedrock engineer later noted in a public forum: "If you haven't worked on Bedrock you have absolutely no idea what an absolute dumpster fire the service is at the backend.")

Mantle represented a rethinking of this architecture. Rather than iterating on the existing system, the team built a new inference engine designed around the scheduling metaphor from the start. Today, when customers use Bedrock with models like MiniMax, Mistral, or other open-source offerings, Mantle is what's actually placing and executing those requests.

The benefits are twofold. For customers: better latency, better performance, and more features. For AWS: efficient resource utilization in what Magherimov describes as "a demand-constrained space where there's enough demand for the customers." Every efficiency gain translates directly to serving more customers who want to use the service.

The Methodology: AI-First Development

Evolution of an Approach

Magherimov's journey to AI-first development mirrors what many engineers experienced. When large language models first emerged with coding capabilities, results were mixed. "You can try, you can maybe have it write a little piece of code. Occasionally it would get things right. Twitter is full of all the jokes and comments making fun of the code produced by the models."

He began using models for prototypes—having an idea, generating code to test it, then rebuilding properly for production. "I wasn't quite convinced yet whether this is something that could be used in production. I was using it maybe more as a gimmick or as a tool to help me learn rather than something that would make me more effective."

The inflection point came roughly six to nine months before the interview. “All of a sudden the code produced by the model was getting good enough to where—I still needed to make modifications, I still needed to constrain it—but all of a sudden it was solving the problems in a more robust way.”

The Core Principle

What distinguishes the Mantle team’s approach from “vibe coding” is a fundamental rule: human accountability for every line of code.

“At the end of the day, any line of code committed into the repository has a human name attached to it. A human is ultimately responsible for the quality of the code.”

This creates what Magherimov calls “right tensions.” The engineer responsible for the code must figure out how to make the model produce quality output. It’s analogous to a compiler or programming language—a tool that accelerates work—rather than an autonomous agent making independent decisions.

The workflow is iterative. Magherimov prompts the model, reviews the output, and makes a series of decisions: Did the model solve the problem correctly? Did it use the right practices and libraries? Is it overly complicated? Did it miss edge cases? Sometimes he takes over manually when the model is close but needs minor fixes. Other times he prompts again with feedback, “no different than how you would do it with another engineer.”

Finding the Maximally Supportable Request

One of the key skills Magherimov developed was calibrating request complexity. “One of the intuitions you build up is finding the maximum supportable request from the model. Ask too much and the model is likely to fail—it might run out of the context window or do weird stuff. Ask for too little and you’re not getting the speed-ups because you’re constantly in the loop.”

This calibration isn’t static. It changes with different models, projects, and domains. The only way to develop this intuition is through practice.

Ambiguity is another enemy of productivity. When multiple approaches could solve a problem, Magherimov often brainstorms with the model before implementation. “I literally started with: here’s the problem I have, list solutions. Immediately it became obvious that one of those was really not going to work. We worked back and forth to the point where I finally felt confident this was going to work.”

Managing Context Windows

Context window management became critical to the workflow. “I clear the context window between every request,” Magherimov explains. For tasks requiring multiple iterations, he uses files as long-term memory—design documents that persist between sessions.

“I would start with a file that describes a high-level approach. Then we break it down. I would start a prompt with: okay, here’s our design we agreed on, we’re now on step two out of seven, and here’s what we’re going to do in this step.”

When a step completes, the code is committed, the context is cleared, and the next session begins fresh with the durable spec file reloaded. This approach mirrors how humans naturally work on large problems—decomposing them into manageable pieces—while working within the model’s constraints.

The Results: 10x Development Velocity

Measuring the Impact

The team measured their velocity in commits—an imperfect but tangible metric. "We've written on average close to 10x, maybe even more than 10x, number of commits across the whole team. And it's not just one team member. It's every team member on the Mantle team."

For Magherimov personally, the impact extended beyond raw output. As a distinguished engineer, continuous blocks of coding time had become rare. "With agentic development, I just find that I accomplish so much more in the same amount of time."

More surprising was the shift to asynchronous development. "The thing that probably took me most by surprise is that asynchronous nature of software development. All of a sudden, I don't actually need to be present or even paying attention for 80% to 90% of the cycle of making a software change."

He describes prompting models overnight, checking results in the morning. His family jokes about his habit of queuing work before sleep. "It bothers me that this whole time isn't being used. I can give it a prompt and wake up in the morning and maybe it was too ambitious, it doesn't work, but let's just keep working. Nothing to lose."

Quality at High Velocity

The increased velocity created new challenges. "Even if the rate of bugs was lower than what would have been with a human, they still happen. And when you write a lot more code, you're gonna have a lot more bugs."

More critically, bugs at this velocity impact the entire team. A buggy check-in can break other engineers' workflows, creating "tools-down" scenarios where everyone chases down what changed.

The team's response was to raise the bar on testing and verification. "We needed to focus on how do we catch as many bugs as possible before they're checked in, before they get into production source code, before they impact other engineers doing testing."

Approximately 25% of the team's energy went into improving development practices rather than feature work. They worked with build system teams to speed up feedback loops—minutes instead of twenty to thirty minutes for builds and integration tests.

"A lot of it are things our industry has been talking about," Magherimov acknowledges. "But all of a sudden they become even more important. At these rates of change, if you don't have a way to curtail chaos, it just explodes."

Team Dynamics: Communication at Speed

The Hypothesis

The Mantle team made a deliberate bet on co-location. "Our hypothesis early on was that it will be really hard for us to move fast without communicating frequently, at high throughput, and with high fidelity."

The entire team sits together in one small area. Communication is constant—walking to desks rather than scheduling meetings, whiteboard sessions emerging organically from quick questions. Magherimov estimates spending at least an hour daily in face-to-face technical discussions.

Prioritizing Shared Understanding

When someone approaches with a question or decision, Magherimov treats it as the highest priority. "If somebody comes to me with a problem or a decision they have to make, I actually think that's probably the

most important thing I could be doing at the time."

The reasoning is economic: "The cost of not having that shared understanding is that you're gonna go build something wrong, and then you're gonna have to double back and redo it. It's more expensive in the long term."

Reducing Meetings

The team took a strict stance on meetings. Beyond whiteboard conversations and discussions directly related to building Mantle, they constantly asked: is this meeting necessary?

Magherimov applies a mental framework: Am I a sponsor pushing an idea forward? A decision maker? The person who will do the work? "If the answer is no to all of those, it's probably a meeting that would be okay to skip."

Key Lessons

What Made It Work

When asked what advice he would give teams starting this journey, Magherimov offers two principles.

First: "Try. Build. There's just no substitute for building. Nothing I say right now, nothing anybody else says, nothing you're gonna read online is gonna teach you how to apply the tools to your problem space, to your domain."

Second: "Lean in. The thing that made the Mantle team successful was that we all believed it was possible, and we knocked down the barriers, and we changed our own mental models and approaches to make it so."

The mindset shift is crucial. "Start not with 'Is AI gonna work or not?' but with 'AI works. What do I need to change about my systems, my approaches, my software development approaches to make it so?' "

The Magnification Effect

Perhaps the most significant insight came from a colleague: "AI-assisted development is gonna magnify well-functioning teams much more than it magnifies not-so-well-functioning teams."

Teams that do the fundamentals well—clear communication, strong testing, fast feedback loops, shared understanding—see those advantages amplified. Teams with dysfunction see that amplified too.

Conclusion

Project Mantle represents one of the first large-scale production deployments of AI-first development methodology at AWS. The 10x velocity claim, while hard to verify precisely, is supported by commit data and corroborated by multiple team members.

What emerges from Magherimov's account is not a story of AI replacing engineers, but of AI changing the nature of engineering work. The fundamentals remain: clear thinking, accountability, testing, communication. What changes is the execution speed and the rhythm of work—more asynchronous, more iterative, more reliant on well-calibrated human judgment about what to ask and how to verify.

As Magherimov observes: "We are seeing a big shift in the industry. A lot of how things work are going to change."

The Mantle project offers a glimpse of what that change might look like in practice.

Based on AWS Podcast Episode 753, released January 26, 2026. Transcript generated using OmniVoice CLI.

References

- Magherimov, Joe. "Amazon Bedrock Mantle and Developing at the Speed of AI." AWS Podcast. Episode 753. AWS. January 2026. podcasts.apple.com/us/podcast/753-amazon-bedrock-mantle-and-developing-at-the-speed-of-ai/id1122785133

Appendix: Public Discussion

The following comments are from a Reddit discussion (r/amazonemployees, November 2025) providing additional context and perspectives from Amazon employees on the AI-first development claims.

Source: reddit.com/r/amazonemployees/comments/1oln543

Thread: "Was AWS Bedrock entirely refactored by AI in one month?" (71 points, 86% upvoted, 36 comments)

Clarifications on Scope and Timeline

TemporalCoral (82 points): > More than a month. Hard to give it a number because more engineers were added to the project later, but it started in May. It was A/B tested a month or two ago irrc. > > And it wasn't a refactor as much as a rewrite. And yes they used AI, but "they" in this case was a DE and like 8 TT PEs from EC2. So it's not like they told Q to "refactor Bedrock." > > And it wasn't the entirety of Bedrock's backend.

Calloused_Samurai (10 points): > No, it wasn't. A new service was created that solves many of the existing problems, but there are huge parts of Bedrock that have not been migrated.

nope_nope_nope_yep_ (L7 Principal SA & Bar Raiser, 7 points): > No, this is not true it wasn't the entirety of Bedrock.

On the Team and Process

Calloused_Samurai (1 point): > Yup. The work is VERY impressive, but that's what happens when you give 5 engineers with 100+ collective years of experience this type of tooling.

[deleted] (1 point): > And it wasn't just AI. It was a whole team with a lot of PEs and they invested in a top notch development environment (you can run the whole stack locally and run integration tests against it, locally) and it's still the engineers driving the AI, giving it narrowly focused prompts. They didn't go "hey, Claude: rewrite Bedrock."

GloppyGloP (1 point): > Not in this case. It was a distinguished engineer and a senior PE making sure it was behaving and being very clever in how to use it and validating the output.

On AI as a Tool

Remarkable_Ad7161 (12 points): > Mostly the way routing and load balancing worked. Bedrock is a product. Most products have tons of components that will make it work. After years of operating and running the product, lots of lessons have been learnt and it's fixable. AI is a glorified autocomplete. If you know what you want from it, but don't have the time to type stuff, it does exceptionally well. But without that instinct these people who could have written it without an AI built over decades, it's nowhere near capable of delivering on its own.

behusbwj (1 point): > Vibe coding is idiotic and they knew that going in. I say this often. The people downplaying AI are too arrogant to understand it's not supposed to be AGI yet. They expect it to be magic instead of a medium for their own thoughts. You give it unstructured slob prompts and it'll give you unstructured slob code. It's like driving a car straight off a bridge without touching the wheel and saying it can't turn

properly. Kiro's competitive edge isn't its agents. It's forcing users to use AI properly with a guided experience. > > What you've hearing about is a team of people senior enough to know how to prompt well and had access to top of the field advisors.

Skepticism and Concerns

Jaded_Concentrate713 (21 points): > Guess Bedrock is gonna go down soon then.

SirSpankalott (9 points): > It's already such a spotty service. Issues every week with TTFT, random errors, etc.

RaisedByCakes (4 points): > I worked on Bedrock for 2 years. From pre-GA until this year. If you haven't worked on Bedrock you have absolutely no idea what an absolute dumpster fire the service is at the backend. Almost each on-call is capable of leading to an outage and COE. > > My first on-call after GA had an outage. Because of a lack of nullptr handling. Because of code that was deployed without any testing. Code that was written by a senior SDE. Pushed through one of the main pipelines that didn't have any testing or rollbacks. After this we did weekly manual deployments of pipelines using MCMs. For more than a year. Absolutely miserable time. Was thrilled to leave. > > I've heard from former colleagues recently that things have begun improving after the old leadership was phased out and EC2 leadership moved in, but it's still far from stable or normal.

BejahungEnjoyer (6 points): > Yes but only for shitty open source models that are like 2% of total traffic. Will their refactor scale to Claude too? Time will tell.

On Testing

git0ffmylawnm8 (8 points): > One would hope unit tests are extensive enough to identify issues from AI generated code.

SoftwareEngineer735 (OP, 14 points): > The principal engineer says in the video that they used AI precisely to generate the unit tests for all the code they had written without [tests].

git0ffmylawnm8 (19 points, satirically): > def unit_test(input): if ai_generated_func(input): return True else: return True

---

Key Takeaways from Discussion

1. Timeline was longer than "one month" - Started in May, A/B tested months later
2. Scope was limited - Not all of Bedrock, focused on routing and load balancing
3. Team was highly experienced - Distinguished Engineer + ~8 Principal Engineers from EC2
4. Human-driven process - Engineers drove the AI with focused prompts, not autonomous
5. Strong development environment - Full local stack with integration testing
6. Mixed reception - Some impressed by velocity, others skeptical of long-term maintainability