DEDICATTED

DEDICATTED ENGINEERING GUIDE

From AI-assisted to AI-driven

A practical guide to adopting AWS AI-DLC in enterprise delivery

Written for CTOs and engineering leaders who bought the licenses, watched developers get faster, and are still waiting for the delivery numbers to move. This guide covers what AI-DLC is, what actually changes, what it will not fix, and how to run it on a real workload in 90 days.

Dedicatted
AWS Premier Tier Services Partner
Montreal and Toronto

Dmytro Petlichenko
AI-DLC AWS Ambassador
August 2026 · dedicatted.com

What is in this guide

Twelve sections. The first four set up the model, the middle five walk the lifecycle, and the last three are the practical part you can act on this quarter.

A note on sources

AI-DLC is an open methodology published by AWS in July 2025, with the workflow rules open sourced that November. Every figure in this guide is attributed to its published source. Where a number belongs to Dedicatted rather than AWS, it is marked as ours.

Your licenses are not your lifecycle

By 2026, almost every engineering organization has AI coding tools in place. Most can point to developers who are visibly faster. Very few can point to a release that shipped sooner because of it. The gap is not the tool. It is that the lifecycle around the tool never changed.

1 **The build stage got faster. Nothing around it did.** Code, tests and documentation come out quicker. Requirements still wait on a stakeholder. Sign-off still waits on a committee. Speed added in the middle of a slow pipeline just moves the queue somewhere else.

2 **AI-DLC is a methodology, not a product.** AWS published it in July 2025 and open sourced the workflow rules that November. It ships as rule files, so it runs on the agent your team already uses: Kiro, Amazon Q Developer, Claude Code, Cursor, Copilot and others. There is nothing new to buy.

3 **The unit of work changes shape.** Sprints become bolts measured in hours or days. Epics become units of work. Scrum teams of about seven become pods of two or three people working live, together, validating what the agent proposes as it proposes it.

4 **Governance moves into the agent.** Steering files put your security policy, architecture standards and approved dependencies into the agent's operating parameters. That, plus traceability from business intent through to test, is what makes this defensible on a regulated workload.

AI does the work. You keep the decisions.

Everything else in AI-DLC follows from that one split. The agent drafts the plan and asks what it is missing. A human answers. Only then does anything get built.

Who this is for

Engineering leaders running real delivery on real deadlines, in organizations where a wrong release costs something. If you are prototyping alone, most of this will feel like overhead. It is not written for that case.

What you will walk away with

A clear read on what AI-DLC changes, an honest list of what it does not, a measurement set you can pull from Git, a 90 day sequence, and a self-check to find your actual starting point.

What AI-DLC actually is

AI-DLC stands for AI-Driven Development Lifecycle. AWS built it to replace two approaches that both underdeliver: bolting AI onto an unchanged process as a smarter autocomplete, and handing an agent a prompt and hoping a product falls out.

The difference is where AI sits. In AI-assisted development, a developer works and the agent helps. In AI-DLC, the agent runs the workflow and the team validates it. The agent writes the plan, flags what it does not know, and stops. You decide. Then it executes. That pattern repeats for every activity in the lifecycle.

STEP 1
AI drafts a plan
Requirements, design or code, depending on the phase.

STEP 2
AI asks what it is missing
It surfaces the gaps instead of guessing at them.

STEP 3
You decide
Business context and trade-offs stay with the humans.

STEP 4
AI executes
Then the loop restarts on the next slice of work.

The loop runs inside three phases

PHASE 01
Inception
Business intent becomes requirements, stories and small units of work, agreed live by the whole team. AWS calls this mob elaboration.

›

PHASE 02
Construction
The agent proposes logical architecture, code and tests against that context. The team clarifies technical decisions in real time.

›

PHASE 03
Operations
Infrastructure as code, deployment and incident work, with production signals fed back into the agent's context.

Context compounds, and that is the whole point

Each phase writes its plans, requirements and design decisions back into the project repository. The next phase starts better informed than the last one did, and work survives across sessions. Teams that skip this and treat every prompt as a fresh start get a faster autocomplete, not a different lifecycle.

It is agent agnostic

AI-DLC is delivered as rule files, not software. The content is the same everywhere and only the file path changes. AWS publishes the open source workflow at github.com/awslabs/aidlc-workflows, with setup documented for Kiro, Amazon Q Developer, Cursor, Cline, Claude Code, GitHub Copilot and OpenAI Codex.

It adapts to the job

Not every change earns the full ceremony. The workflow checks whether the work is greenfield or brownfield, looks at codebase complexity, and decides which stages are worth running. A small bug fix skips most of it. A system migration gets all of it.

The vocabulary changes, and that matters

AI-DLC renames things on purpose. If your team keeps running two week sprints and starts calling them bolts, nothing has changed. Here is what actually moves, and why.

TRADITIONAL SDLC	AI-DLC	WHY IT CHANGES
Sprint, two weeks	**Bolt, hours to days**	Batching existed because human coordination was expensive. It is not any more.
Epic	**Unit of work**	Small enough for an agent to hold in context and a team to validate in one sitting.
Scrum team, about seven developers	**AI pod, two to three developers in mob format**	Fewer hands on keys, more eyes on output.
Backlog grooming, asynchronous	**Mob elaboration, live**	Specificity is the input that decides output quality. You get it faster in a room.
Review after completion	**Continuous review**	The volume of generated code makes end-of-cycle review unworkable.
Documentation written later	**Documentation generated with the change**	The agent already holds the context. Nothing is saved by deferring it.
Low fidelity mocks	**High fidelity tests, security included**	More code needs proportionally more automated control, not more reviewers.

The team shape is the hard part

Renaming a sprint costs nothing. Going from seven developers working in parallel to three working together on one thing is a real organizational change, and it is the one that gets resisted. It also happens to be where the reported gains come from.

Expect this to be a conversation with delivery managers and finance, not just with engineers.

One rule worth taking seriously

During Inception, get the team in the same room at the same time, or at least on the same hours. AWS is explicit that synchronous collaboration in this phase drives context quality, and context quality drives everything downstream.

Construction and Operations tolerate distributed work far better. Inception does not.

What AWS has published, and how to read it

AI-DLC arrives with headline numbers. They are real and they are published. They also came from teams operating under conditions most organizations do not have yet. Use them to open a conversation, not to set a target.

76 days

Amazon Bedrock inference engine, rebuilt

Original estimate was 40 engineers and a full year

6 of 40

Engineers who actually did it

Reported in Amazon's 2025 letter to shareholders

15 → 35

Features per cycle, European financial services firm

Delivered by 3 developers rather than 12

10x

Modelled return in AWS's cost-to-serve example

$2M invested against $20M in avoided cost

Read the conditions, not just the result

Every one of these ran on fast CI/CD, high fidelity test environments, automated security and compliance gates, and clear ownership of decisions. AWS states this directly: AI-DLC amplifies the engineering practice you already have. It does not compensate for foundational gaps. A team with a 40 minute build and manual security review will not reproduce these numbers, and should not promise them.

You do not need a 12x compression

AWS's cost-to-serve-software example is the useful one for a business case, because the arithmetic is boring. Take a 1,000 developer organization at roughly $130K per developer per year, all-in with tooling. That is $130M. A 15 percent improvement is around $20M in avoided cost against a $2M investment.

Fifteen percent is a number a well-run team can actually hit and defend.

Find your own number, then use it

The value of measuring is not to confirm someone else's result. It is so you can put a defensible figure into your next estimate. A consistent 15 percent reduction in cycle time across your portfolio is 150 hours on a 1,000 hour estimate, and that belongs in the proposal.

Which means the baseline comes first. See section 10.

TO BE COMPLETED BEFORE PUBLISHING

Dedicatted proof point. This page needs one of our own numbers to sit alongside AWS's. Pick a delivery where we can show a real before and after pulled from Git: cycle time, PR throughput, or incident volume. Send me the engagement and the figures and I will write it up as a short case panel here.

01 Inception

MOB ELABORATION

Business intent goes in. Requirements, stories and small units of work come out, agreed by everyone in the room while the agent is still asking questions.

What happens

- The team gives the agent context on the existing codebase, not just on the new ask.
- The agent drafts requirements and user stories, then raises follow-up questions at each step.
- Developers, product, business analysts and QA answer those questions together, out loud, in one session.
- The output is a set of units of work small enough to build and validate inside a single bolt.

Who is in the room

Cross-functional and synchronous. This is the phase where being in the same room, or at least on the same hours, genuinely matters. What you agree here sets the ceiling on everything the agent produces later.

What good looks like

- Every unit of work traces back to a stated business intent.
- The agent's open questions get answered in the session, not deferred to a follow-up.
- Plans and decisions are written into the repository, not into a meeting doc nobody reopens.
- Acceptance criteria are specific enough that a test can be generated from them.

Where it goes wrong

- One person elaborates alone and the rest of the team reviews it later. That is the old process with an extra step.
- Units of work are sized like epics. The agent loses the thread and the team cannot validate the output in one pass.
- The team elaborates the new feature and never gives the agent the existing code. It designs against a system that does not exist.

Rush Inception and Construction will produce confident, well-tested code for the wrong thing.

02 Construction

MOB CONSTRUCTION

With the context agreed, the agent leads on logical design, code, tests and infrastructure as code. Your team's job shifts from writing to validating, continuously.

What happens

- The agent proposes a logical architecture and domain model against the units of work from Inception.
- Code, tests and infrastructure as code are generated together, not in sequence.
- Security and resilience tests are part of the same output, not a separate pass at the end.
- Documentation is generated with the change and updated automatically as it moves.
- When a test fails, the agent self-corrects against the steering files before a human sees it.

The review model has to change

"Write code, then review" does not survive the volume. Review becomes continuous and largely automated, with people held back for the calls only people can make. If your pipeline still batches review to the end of a cycle, fix that before you scale this phase.

What good looks like

- Human review time goes to logic, authorization and business rules, not to formatting and boilerplate.
- Every merged change traces to a unit of work and to a test.
- Quality gates run automatically, and the pipeline is fast enough that nobody routes around them.
- The pod is two to three developers, working on one thing, together.

Where it goes wrong

- Generated code looks clean, passes surface checks, and mishandles the exception path. Weight review toward edge cases, boundaries and authorization, not the happy path.
- Reviewers stop being skeptical. High volume plus plausible output erodes attention faster than people expect.

Two to three developers, in mob format, reviewing continuously. That is the shape.

03 Operations

CLOSE THE LOOP

AI-DLC does not stop at merge. The same accumulated context runs deployment, and what happens in production feeds back into it.

What happens

- Infrastructure as code generated during Construction deploys through your existing CI/CD.
- Agents assist incident work: gathering signals, investigating root cause, drafting the fix.
- Monitoring output goes back into the agent's context and shapes the next Inception.
- Releases get smaller and more frequent, because small and reversible is less risky than large and batched.

This is where it compounds

Most teams never reach this phase, and it is the one that pays. An agent holding your requirements, your design decisions, your test suite and your incident history is a different tool from one holding your open file. The gap between those two widens every cycle you run.

What good looks like

- Deployment is small, frequent and easy to reverse.
- Routine changes flow through automated validation. Only material changes wait for a person.
- Incident findings land in the repository, not in one engineer's memory.
- Recovery time is tracked as closely as deployment time.

What it needs from you

- Fast pipelines. If a build takes 40 minutes, a cycle measured in hours is a fiction.
- Risk-based change categorization. One approval path for every change is the bottleneck, and it is the one that quietly cancels out the gains.

Each phase makes the next one better informed. Skip Operations and you keep restarting from zero.

Governance that travels with the code

This is the part that decides whether AI-DLC is usable on a regulated workload, and it is the main reason to prefer it over ad-hoc agent use. Governance stops being a review step and becomes a property of how the code gets written.

Steering files

Structured configuration that tells the agent how to behave on every prompt: security policy, architecture standards, approved dependency lists, regulatory guidance such as the EU AI Act. The rules go in at the start rather than getting bolted on at review, so the agent refuses to break them at the point of generation.

End-to-end traceability

A failing test traces to code. The code traces to a user story. The story traces to a stated business intent. That chain is exactly what an auditor asks for, and AI-DLC produces it as a by-product of how the phases hand off to each other rather than as a separate documentation effort.

Risk-based gates

Not every change needs a committee. Categorize by risk, automate the routine path, and reserve human sign-off for changes that carry real exposure. This is the single highest-value pipeline change most organizations can make, and it is worth doing whether or not you adopt the rest.

Accountability stays put

The engineering and product teams sponsoring AI-assisted work take documented ownership of the risk it introduces, join the risk assessment before deployment, and name the exposures their tooling creates. Security moves from absorbing risk to overseeing it. That is a workload change, not a headcount request.

One prerequisite, stated plainly

AI-DLC assumes mature DevSecOps. If your pipeline is slow, your test environments are thin, or your security scanning is periodic and manual, fix that first. AI-DLC amplifies the engineering practice you already have, in both directions. Higher code volume on a weak foundation produces more of what the foundation was already missing.

The security review problem is real

A typical enterprise runs somewhere between one security engineer per 100 developers and one per 200. That ratio was under strain before agents started producing code at multiples of the old rate. You cannot hire your way out of it, and adding an AI-specific review checklist on top of an understaffed function makes it worse.

So move it left, and cut the queue

Put the policy in the steering files so violations never get written. Then filter findings for whether the vulnerable path is actually reachable in your application, rather than present in the abstract. Reserve human attention for logic-level decisions, which is the part tooling is still weakest at.

What AI-DLC will not fix

Worth being direct about this, because disappointment usually comes from here rather than from the tooling. AI-DLC compresses the part of delivery where work gets made. The parts where work gets agreed, approved and decided run at the same speed they always did.

1. **Requirements you cannot get.** If stakeholders are hard to reach, inconsistent about what they want, or unable to state what "done" means, no agent changes that. Mob elaboration surfaces the gap faster and more visibly. It does not fill it. The developer still waits.

2. **Sign-off that was never redesigned.** The constraint moves from creation to approval. A team producing in days, feeding an acceptance process built for weeks, will not ship sooner. It will build a longer queue and everyone involved will find that more frustrating than the original problem.

3. **Decisions nobody owns.** AI-DLC deliberately routes decisions to humans, which is the right design. But if nobody in the room has the authority or accountability to choose, the work stops there. It now stops faster, and in front of more people.

4. **A weak engineering foundation.** Slow builds, thin test coverage, manual security review, unclear ownership. Every one of these gets worse under higher volume, not better. This is the failure mode we see most often, and it is the reason section 11 puts foundations before methodology.

Use the speed data to open the conversation about the rest.

You now have evidence that the build stage moved. That evidence is the lever for renegotiating review cycles, acceptance windows and decision rights with the business or the client. A faster engineering team feeding an unchanged approval process frustrates everyone and proves nothing.

A pattern worth watching for

When individual capacity goes up and the organization does not change what it asks for, the extra capacity does not come back to the project. It goes somewhere you cannot see. That is a rational response to an unchanged bar, not a discipline problem, and the fix is to raise the bar on the primary work rather than to police anything.

Which is why measurement comes first

You cannot raise a bar you have not measured, and you cannot defend a raised bar without data your team can see for themselves. Section 10 covers what to track. The short version is that all of it should come out of your pipeline automatically, and none of it should depend on anyone filling in a form.

A KPI set that cannot be gamed

AI-DLC's measurement approach uses metrics that pull against each other on purpose, so improving one at the expense of the others shows up immediately. Pull all of them from your pipeline, not from a developer survey.

Mean time to deployment

From agreed unit of work to running in production. This is the headline number, and it is the one that goes into an estimate.

Mean time to recovery

The counterweight. If deployment time falls and recovery time rises, you have moved cost rather than removed it.

Failed deployment rate

Catches generated code that looks correct and breaks something downstream. Watch the trend, not the single reading.

Events by severity

Total incident volume is noise. The severity mix is the signal, and it tells you whether quality is holding as volume climbs.

Technical debt level

Guards against velocity bought with future cost. Higher output makes this easier to accumulate and harder to notice.

Customer NPS

The outcome the other five are supposed to serve. If it does not move, the rest is internal optimization.

Two rules that matter more than the metric list

Instrument from Git and your pipeline, with no self-reporting. Developers have no incentive to report accurately on their own productivity, and asking them to changes what you measure. **Capture a baseline before you change anything.** Without a before, you have no way to prove what moved, and no defensible number to put in front of a client or a CFO.

Share the numbers with the team, not just with management

This is the cheapest intervention available. When developers can see how their contribution compares to the rest of the pod, most adjust on their own. Adoption climbs without any managerial pressure, and the conversations you do have get easier because they start from shared data.

Then put it in the estimate

Measurement is not only an engineering tool. If your data shows a consistent reduction in cycle time across the portfolio, that belongs in your next proposal as a stated, evidenced advantage. That is the point at which internal gains become something the client can actually see.

Your first 90 days

Sequenced, not parallel. Each step earns the right to the next one, and skipping step two is the most common reason the whole thing stalls at pilot.

1

WEEK 1 TO 2

Executive alignment

Get the C-suite clear on how AI-DLC differs from Agile and what organizational change it asks for, including the shift to smaller pods. Tie adoption to a business outcome someone already cares about. Without sponsorship, this stalls at the pilot and stays there.

2

WEEK 2 TO 4

Baseline and foundations

Capture the current numbers from Git and your pipeline. Audit build times, test fidelity and security gates honestly. Fix whatever would make a cycle measured in hours impossible. This is unglamorous and it is the step that decides the outcome.

3

WEEK 4 TO 6

Technical enablement

Take your architects and engineering leads deep on the agent you have standardized on and on the AI-DLC rule files. Write your first steering files against real policy. Identify two or three senior people who will champion it, because peer influence moves adoption faster than any mandate.

4

WEEK 6 TO 10

A real pilot on a real codebase

One team, their own repository, a genuine unit of work. Run bolts of two to three days. Document what worked, what cost time, and what the numbers did against the baseline. A pilot on a toy project proves nothing you can use.

5

FROM WEEK 10

Scale on evidence

Roll the validated pattern to the next teams. Deliberate waves and open self-service both work. Neither works without the baseline from step two, and neither survives a team that has not fixed its pipeline.

The rule files are open source.

AWS publishes them at github.com/awslabs/aidlc-workflows, with setup documented for Kiro, Amazon Q Developer, Cursor, Cline, Claude Code, GitHub Copilot and OpenAI Codex. Pull from a tagged release and keep the workflow artifacts in version control.

Readiness self-check

Tick honestly. The unticked lines are your next piece of work, and the group they sit in tells you where to start.

Foundations

- A baseline of delivery metrics is captured from Git, with no developer self-reporting.
- Build and test pipelines run fast enough to support a cycle measured in hours.
- Security scanning is automated and continuous, not periodic and manual.
- Test environments are high fidelity rather than thin mocks.

Ways of working

- One agent is standardized across the team and provisioned at project level, not individually.
- Elaboration happens live, with the whole team, rather than asynchronously.
- Units of work are sized to be built and validated inside one bolt.
- Review is continuous, with human attention held for logic and authorization.

Governance

- Steering files carry your security policy, architecture standards and approved dependencies.
- Every change traces from business intent, through story, to test.
- Change is categorized by risk, with human sign-off reserved for material changes.
- The team sponsoring AI-assisted work has documented ownership of the risk it introduces.

How to read your score

Fewer than eight. Start with foundations, not methodology. Adopting AI-DLC now will surface every gap at once.

Eight to ten. You are ready to pilot. Pick one team and one real codebase, and hold the line on the baseline.

Eleven or twelve. Scale it. Your constraint is almost certainly sign-off and decision latency, not engineering.

None of this was caused by AI

Slow sign-off, thin measurement, strained security review and decisions nobody owns were all there before. They were just hidden behind a development stage that took long enough to absorb them. What AI removed was the cover.

AI-DLC is useful because it is specific about the alternative. It names the phases, changes the unit of work, puts governance where the code gets written, and gives you a measurement set that resists being gamed. It asks for a real organizational change in return, and it is honest that it will not work on a weak foundation.

The organizations that get lasting value here are not the ones that moved first. They are the ones that treated a faster build stage as a reason to fix what it exposed.

Read the source

AWS DevOps Blog, "AI-Driven Development Life Cycle: Reimagining Software Engineering", July 2025. The founding post and the method definition paper it links to.

Get the workflow

github.com/awslabs/aidlc-workflows. Open source rule files, agent agnostic, with setup guides per assistant. Pull from a tagged release.

Talk to us

We run AI-DLC on AWS workloads as an AWS Premier Tier Services Partner, including in regulated environments where the governance layer has to hold up.

Ready to run AI-DLC on a real workload?

In 45 minutes we will map where your delivery sits today, pick the workload worth piloting first, and name the foundations that need fixing before you start. No deck, no discovery phase.

Book the 45 minutes →

[booking link] · dedicatted.com

AI-DLC is a methodology published by Amazon Web Services. Figures attributed to AWS in this guide are drawn from AWS's published blog posts and from Amazon's 2025 letter to shareholders. Dedicatted is an AWS Premier Tier Services Partner. This guide is Dedicatted's own point of view and is not an AWS publication.