

Agent Nova — Jira-Triggered Salesforce Development System

Detailed Project Requirements & Technical Design

Version 1.0 | August 19, 2026 | Draft for Client Review and Implementation Planning

Table of contents

1. Executive Summary
2. Business Problem
3. Scope
4. Target Architecture
5. System Components
6. Agent Pipeline
7. End-to-End Workflows
8. Workflow States
9. Integration Design
10. Data and Artifact Model
11. Implementation Plan
12. Recommended Technology Stack
13. Security and Governance
14. Success Metrics
15. Recommended MVP
16. Final Project Definition

1. Executive Summary

Agent Nova is an AI-powered Salesforce development automation platform. It connects Jira, Confluence, a Salesforce source repository, and a lower Salesforce environment to automate the journey from a development requirement to a Pull Request for human review.

When a qualifying Jira story is created or updated, Agent Nova retrieves the story, comments, attachments, relevant Confluence documentation, and relevant repository context. A Requirement Analyst determines whether the requirement is complete. If clarification is required, the system posts structured questions to Jira and pauses until a human responds. If the requirement is clear, the workflow proceeds through solution architecture, Salesforce implementation, code review, QA, and deterministic DevOps automation.

Production deployment and merge are outside the autonomous scope of v1. The system is deliberately designed with gates, auditability, allowlists, retries, and a mandatory human approval step.

2. Business Problem

Salesforce development frequently requires developers to spend significant time understanding Jira stories, locating documentation, discovering existing metadata and code patterns, clarifying acceptance criteria, implementing repetitive changes, testing, reviewing, and preparing Pull Requests. Agent Nova is intended to reduce this overhead while preserving engineering controls.

2.1 Goals

- Automatically ingest Jira stories, comments, attachments, and relevant status changes.
- Cross-reference Jira requirements with Confluence documentation and the existing Salesforce repository.
- Identify missing requirements before implementation and ask clarification questions in Jira.

- Generate a technical solution aligned with existing Salesforce architecture and coding standards.
- Implement approved, in-scope changes only in lower Salesforce environments.
- Run static analysis, Salesforce tests, and AI-assisted code review.
- Create a Draft Pull Request containing implementation and validation information.
- Maintain traceability between Jira, design artifacts, code changes, tests, and the PR.
- Keep humans responsible for final review and merge.

3. Scope

3.1 Version 1 In Scope

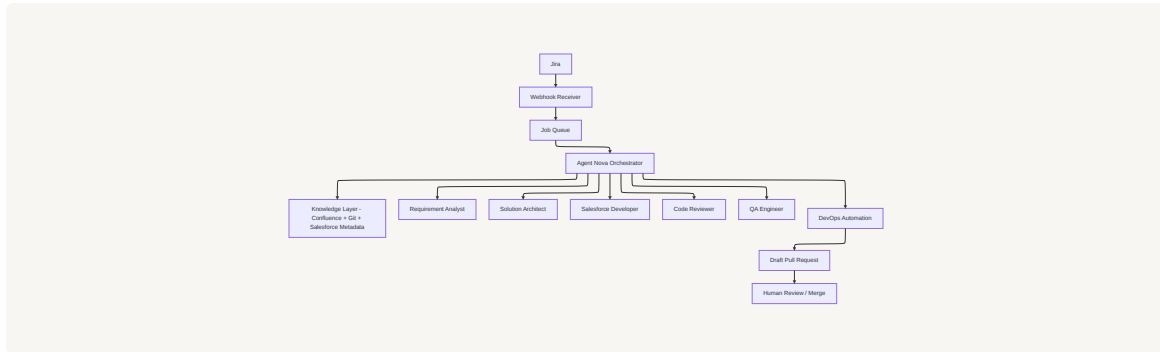
- Jira webhook-triggered intake and issue/comment/attachment processing.
- Confluence search and retrieval for selected project spaces.
- SFDX-format Salesforce repository analysis.
- Feature branch, commit, push, and Draft PR automation.
- Lower Salesforce org metadata retrieval, deployment/validation, and Apex test execution.
- Requirement Analyst, Solution Architect, Salesforce Developer, Code Reviewer, QA Engineer, and deterministic DevOps stages.
- Initial metadata allowlist: custom fields, validation rules, record-triggered Flows, Apex classes/triggers, LWCs, and permission sets.

4. Target Architecture

The system follows an orchestrated workflow architecture. Jira events are received asynchronously, normalized, queued, and processed by an Agent Nova orchestrator. The orchestrator invokes specialized agent skills and deterministic tools while persisting state and artifacts after every stage.

The key design principle is that the LLM performs reasoning and planning, while deterministic tooling performs Git operations, Salesforce CLI operations, static analysis, testing, and PR creation.

High-level architecture (at a glance)



5. System Components

5.1 Jira Webhook Receiver

Receives issue-created, issue-updated, and comment-created events. It validates the webhook, filters events by project/status/label, normalizes the payload, and enqueues work. Idempotency is required so the same Jira event cannot create duplicate runs or PRs.

5.2 Job Queue

The queue decouples Jira's fast webhook response from long-running AI and Salesforce operations. It should support retries, exponential backoff, dead-letter handling, and at most one active workflow per Jira issue.

5.3 Agent Nova Orchestrator

The orchestrator is the central state machine. It loads the current run state, invokes the next skill, validates structured outputs, persists artifacts, enforces gates, handles clarification pauses and retries, and emits operational events.

5.4 Knowledge Layer / RAG

Confluence and repository content should be indexed and retrieved on demand rather than sending the entire knowledge base to the model. Confluence provides business rules, architecture standards, naming conventions, and runbooks. The repository provides Apex, LWC, Flow, object metadata, tests, and existing implementation patterns.

5.5 Execution Layer

- Git CLI/API for clone, fetch, branch, commit, and push.
- Salesforce CLI for metadata retrieval, deployment, validation, and Apex test execution.
- PMD and ESLint or client-approved static analysis tools.
- CI runner for Pull Request validation.

5.6 Persistence and Audit

PostgreSQL should store workflow state, Jira keys, current stages, retry counts, branch and PR references, and errors. Artifacts should include requirements.json, technical design, metadata plan, implementation summary, review report, and test report. Tool calls and agent execution records should be retained in an auditable form with sensitive values redacted.

6. Agent Pipeline

6.1 Requirement Analyst

Inputs include the Jira summary, description, acceptance criteria, comments, attachments, Confluence results, and optionally relevant repository search results. The analyst extracts requirements, identifies missing objects/triggers/personas/edge cases, normalizes acceptance criteria, and creates clarification questions.

Output: requirements.json containing the Jira key, summary, acceptance criteria, clarification questions, clarity decision, confidence, referenced documentation, and out-of-scope items.

Gate: if the requirement is unclear or questions exist, post numbered questions to Jira, set the workflow to NEEDS_CLARIFICATION, and stop until a new Jira

comment is received.

6.2 Solution Architect

The architect maps the clarified requirement to the existing Salesforce solution. It examines objects, fields, automation, Apex, LWC, dependencies, patterns, and standards. It produces a technical design and metadata plan.

The architect must escalate changes outside the v1 allowlist or changes requiring complex sharing/security redesign.

6.3 Salesforce Developer

The developer creates a feature branch, retrieves required metadata, implements configuration or code according to the technical design, creates or updates tests, and follows repository and Confluence standards. Changes should be constrained to the metadata plan plus required generated tests or metadata.

6.4 Code Reviewer

The reviewer analyzes the Git diff against the Jira requirement, acceptance criteria, and technical design. It checks Salesforce security, CRUD/FLS, sharing, SOQL/DML patterns, bulkification, governor limits, naming, error handling, and test quality. Static analysis results are included.

If findings exist, the developer may receive a bounded fix/review loop. A recommended maximum is three review cycles before human escalation.

6.5 QA Engineer

QA derives positive and negative test scenarios from the acceptance criteria, executes Apex tests, validates deployed behavior where automation is available, and documents manual scenarios for human QA. QA produces a test report and blocks PR creation when mandatory tests fail.

6.6 DevOps Automation

DevOps should be deterministic rather than an LLM agent. It validates metadata, commits using a conventional message containing the Jira key, pushes the feature branch, creates a Draft PR, links the PR to Jira, and triggers CI if configured.

7. End-to-End Workflows

7.1 Happy Path

- Product Owner creates a Jira story and marks it as ready for Agent Nova.
- Jira sends an event to the webhook receiver.
- The orchestrator creates an idempotent run and queues it.
- Requirement Analyst retrieves Jira, Confluence, and repository context.
- The analyst determines that the requirement is clear.
- Solution Architect creates the technical design and metadata plan.
- Optional design approval gate is completed.
- Salesforce Developer creates a feature branch and implements the change.
- Changes are deployed or validated in the lower Salesforce environment.
- Code Reviewer and static analysis run.
- Developer fixes review findings when required, subject to retry limits.
- QA runs Salesforce tests and acceptance-criteria validation.
- DevOps validates the final metadata and creates a Draft PR.
- The PR is linked back to Jira.
- Human reviewer reviews and either requests changes or merges.

7.2 Clarification Workflow

- Requirement Analyst identifies missing information.
- Agent Nova posts numbered clarification questions in Jira.
- Workflow changes to NEEDS_CLARIFICATION.
- The system stops further development.
- A human answers through a Jira comment.
- The comment webhook resumes the workflow.
- The analyst reevaluates the requirement with the new information.

- The workflow either proceeds or asks additional questions.

7.3 Failure Workflow

- A stage fails after its permitted retry count.
- Agent Nova records the failure and relevant findings.
- A concise failure summary is posted to Jira.
- The workflow moves to an escalation state.
- A human developer takes over.

8. Workflow States

Recommended states:

- RECEIVED — event received and normalized.
- ANALYZING — requirement analysis in progress.
- NEEDS_CLARIFICATION — waiting for human Jira response.
- ARCHITECTING — technical design in progress.
- DESIGN_REVIEW — optional human design approval.
- IMPLEMENTING — Salesforce development in progress.
- REVIEWING — code review and static checks.
- TESTING — QA and Salesforce tests.
- PR_CREATED — Draft PR exists.
- HUMAN_REVIEW — awaiting human action.
- COMPLETED — merged/accepted according to client workflow.
- FAILED — execution stopped with reason.

9. Integration Design

9.1 Jira

- Receive issue-created, issue-updated, and comment-created events.

- Read issues, comments, attachments, and required fields.
- Post clarification and failure comments.
- Optionally transition Jira status.
- Create a remote link between Jira and the Pull Request.

9.2 Confluence

- Search selected project spaces.
- Retrieve page content and convert it into indexable text.
- Chunk content by heading and maintain source references.
- Refresh the index periodically or when pages are published.
- Exclude restricted content according to client permissions.

9.3 Git

- Clone/fetch the Salesforce repository.
- Create feature branches using Jira keys.
- Commit and push changes using a bot identity.
- Create Draft Pull Requests through the hosting provider API.
- Keep Git as the source of truth for Salesforce metadata.

9.4 Salesforce

- Authenticate only to approved lower environments.
- Retrieve scoped metadata.
- Deploy and validate metadata.
- Run Apex tests and collect machine-readable results.
- Describe Salesforce objects when required for architecture context.

10. Data and Artifact Model

Each Jira issue processed by Agent Nova should have a unique run record. The run record should include the Jira key, workflow state, timestamps, retry count, branch name, PR URL, error information, and artifact location.

The artifact chain should be:

requirements.json → tech-design.md + metadata-plan.json → implementation-summary.md + Git diff → review-report.json → test-report.json → PR metadata

Artifacts should be retained for a configurable period and linked to the run ID for audit and replay.

11. Implementation Plan

Phase 0 — Discovery and Foundation

- Obtain Jira, Confluence, Git, and Salesforce lower-org access.
- Inventory repository structure and Salesforce metadata types.
- Confirm Jira workflow, webhook, project, and agent-ready trigger.
- Map Confluence spaces and priority documents.
- Collect 5–10 representative Jira stories for evaluation.
- Confirm scope allowlist and environment strategy.

Phase 1 — Requirement Analyst MVP

- Build webhook receiver and job queue.
- Implement Jira read/write integration.
- Implement basic Confluence and repository RAG.
- Implement Requirement Analyst skill.
- Implement clarification comments and paused/resumed workflow.
- Add run persistence and basic operational logging.

Phase 2 — Solution Architect

- Build repository metadata inventory.
- Add optional Salesforce object describe.

- Implement Solution Architect skill.
- Generate technical design and metadata plan.
- Add optional human design approval.

Phase 3 — Developer and PR Pipeline

- Implement Salesforce Developer skill for allowlisted metadata.
- Implement Git branch/commit/push.
- Implement lower-org deployment/validation.
- Implement baseline Apex test generation.
- Create Draft PR and Jira remote link.

Phase 4 — Reviewer and QA

- Integrate PMD/security checks.
- Implement Code Reviewer skill.
- Implement QA test matrix generation.
- Execute Salesforce tests and interpret results.
- Add bounded review/fix loops.
- Integrate CI.

Phase 5 — Hardening and Pilot

- Handle Jira story revisions and stale runs.
- Strengthen idempotency and concurrency controls.
- Add failure notifications and operational runbook.
- Run pilot on one Jira project.
- Measure quality and developer time savings.

12. Recommended Technology Stack

- Backend/API: Python with FastAPI.

- Workflow orchestration: LangGraph initially; Temporal can be considered when durable long-running workflows become a priority.
- LLM: client-approved model provider such as OpenAI or Anthropic.
- Database: PostgreSQL.
- Queue: Redis with Celery/ARQ, or a durable workflow platform.
- Knowledge retrieval: pgvector or another approved vector store plus exact code search.
- Salesforce tooling: Salesforce (sf) and SFDX project format.
- Git/PR: GitHub, GitLab, or Bitbucket according to client environment.
- Deployment: Docker-based service on the client's approved infrastructure.

13. Security and Governance

- Store credentials in a secrets manager; never place secrets in prompts or source control.
- Use separate service accounts and minimum required permissions.
- Do not provide production Salesforce credentials to Agent Nova in v1.
- Enforce a metadata allowlist at the orchestrator level.
- Reject destructive or out-of-scope changes unless explicitly approved.
- Default new Apex classes to appropriate sharing/security controls.
- Check CRUD/FLS for user-facing queries and DML.
- Prevent hard-coded Salesforce IDs.
- Sanitize Jira and Confluence content to reduce prompt-injection risk.
- Do not allow arbitrary shell commands to be generated by the model.
- Record agent/tool activity with sensitive information redacted.
- Require human approval before merge.

14. Success Metrics

- Clarification quality: target human rating of at least 4/5.
- Time from Jira creation to PR: target 50% reduction for simple stories after pilot stabilization.
- First-pass PR approval: target at least 40% for supported story types.
- First DevOps test pass rate: target at least 90%.
- Track human escalation/override rate and use it to refine scope.
- Developer satisfaction target: at least 3.5/5 during the pilot.

15. Recommended MVP

The first practical MVP should focus on a controlled end-to-end path rather than attempting complete autonomous Salesforce development immediately.

Jira

↓

Webhook

↓

Agent Nova

↓

Requirement Analyst

↓

Confluence + Repository Search

↓

Clarification Gate

↓

Solution Architect

↓

Salesforce Developer

↓

Lower Salesforce Org

↓

Automated Tests

↓

Git Branch

↓

Draft Pull Request

↓

Human Review

Once this path is reliable for a small number of supported story types, Code Reviewer and QA automation can be expanded, followed by stronger closed-loop remediation and production hardening.

16. Final Project Definition

Agent Nova is a Jira-triggered, human-supervised AI development platform for Salesforce. Its responsibility is to understand requirements, gather project context, plan and implement supported changes in a lower Salesforce environment, validate those changes, and prepare a Pull Request. Its responsibility does not include autonomous production deployment or final merge.

The architecture should keep AI reasoning separate from deterministic execution. Every important stage should produce structured artifacts, every external action should be controlled by explicit tools and permissions, and every workflow should be resumable, auditable, idempotent, and capable of escalating to a human.

The recommended delivery strategy is phased: establish integrations and workflow state first, then deliver Requirement Analysis, Architecture, Development/PR automation, Review/QA, and finally hardening and pilot deployment.

17. Workflow (Mermaid)

