That's an excellent next step for your agent. Moving from individual file/snippet refactors to **repository-wide, context-aware migration** significantly increases the agent's value.

Here is a supplementary Product Requirements Document (PRD) focusing on the **Repository Scan and Refactor Capability** for your Universal Cloud Refactor Agent.

🚀 Universal Cloud Refactor Agent: Repository-Level Migration

---

1. Goals & Objectives

- **Primary Goal:** Enable developers to initiate a cloud migration for an entire codebase/repository with a single command or action.
- **Key Objective 1:** Automatically identify **all** cloud-specific references (e.g., AWS SDK calls, configuration files, deployment scripts) across the entire repository.
- **Key Objective 2:** Generate a consolidated, single Pull Request (or equivalent branch) containing all refactored code, ensuring atomic and consistent changes across the entire codebase.
- **Key Objective 3:** Preserve or generate new **tests** to validate the behavior of the refactored code, ensuring no silent regressions.

---

2. User Stories (As a Developer...)

ID	User Story	Acceptance Criteria
US-R1	I want to **point the Agent at a Git repository URL** (e.g., GitHub, GitLab) so that it can access and clone the entire codebase for analysis.	The Agent successfully initiates a scan from a given HTTPS/SSH Git URL and branch name.

US-R2	I want the Agent to **perform a dependency and code pattern analysis** to generate an estimated migration plan before making any changes.	The Agent outputs a preliminary **Migration Assessment Report** (MAR) listing identified services (S3, Lambda, etc.), estimated lines of change, and a confidence score.
US-R3	I want the Agent to **refactor all identified cloud services** (e.g., S3 to GCS) consistently across all files (code, config, templates) in the repository.	A single, complete branch/PR is generated with all necessary file modifications (code, imports, configuration).
US-R4	I want the Agent to **detect and update any cross-file dependencies** (e.g., a constant defined in one file used in a service call in another) to ensure logical consistency.	Changes to constants, variable names, or configuration keys are propagated correctly throughout the project.
US-R5	I want the Agent to **run existing unit tests** or **generate new tests** for the refactored functions to validate functional parity.	The Agent can execute a predefined test command (npm test, pytest) or generate a test suite that passes on the new cloud code.
US-R6	I want a **clear log and diff summary** of all changes made by the Agent so I can easily review the generated Pull Request.	The generated PR includes a markdown summary detailing the service migrations, files changed, and test results.
US-R7	I want a **Rollback Strategy** in case the refactored code fails in staging.	All changes are confined to a new Git branch, allowing an easy rollback via branch

		deletion or PR close.

---

3. Functional Requirements (FRs)

3.1. Repository Ingestion & Analysis

- **FR 3.1.1 (Source Integration):** The agent must support integration with major Git providers (GitHub, GitLab, Bitbucket) via SSH or API keys for cloning.
- **FR 3.1.2 (Language/File Parsers):** The agent must have enhanced parsers capable of understanding project structure, including imports, dependency files (package.json, pom.xml, requirements.txt), and configuration files (YAML, JSON, Terraform/CloudFormation templates).
- **FR 3.1.3 (Full Codebase Indexing):** The agent must build an internal model or graph of **cross-file dependencies** (e.g., a Python import of an S3 utility class).
- **FR 3.1.4 (Migration Recipe Matching):** The agent must scan the indexed codebase against a defined library of cloud-to-cloud migration recipes (e.g., all patterns for boto3.client('s3') map to GCS equivalents).

3.2. Refactoring Execution & Consistency

- **FR 3.2.1 (Atomic Refactoring):** The agent must apply all changes in a single, coherent pass to maintain state consistency across files.
- **FR 3.2.2 (Error Handling Refactor):** The agent must identify cloud-provider-specific error handling (e.g., Boto3.ClientError) and replace it with the target cloud's equivalent or a provider-agnostic abstraction.
- **FR 3.2.3 (Configuration Migration):** The agent must update all associated infrastructure-as-code (IaC) files (e.g., Terraform, CloudFormation, Pulumi) and CI/CD pipelines to reference the new cloud services and configurations.

3.3. Verification & Output

- **FR 3.3.1 (Automated Test Execution):** The agent must be configurable with a **test command** to run the existing suite post-refactor and report the pass/fail status.
- **FR 3.3.2 (Test Scaffolding):** If no tests are found, the agent must generate **basic functional unit tests** for critical refactored functions (e.g., an S3 interaction function) to provide immediate validation.
- **FR 3.3.3 (PR Generation):** The agent must commit all changes to a new branch, push it to the source repository, and open a Pull Request with the **Migration Assessment Report** as the body.

---

4. Non-Functional Requirements (NFRs)

- **NFR 4.1 (Performance):** The entire scan and refactor process for a repository of up to **20,000 Lines of Code (LoC)** should complete in less than 30 minutes.
- **NFR 4.2 (Security):** The agent must adhere to **least-privilege principles** when interacting with Git providers and cloud environments. It must not store user credentials/tokens longer than the duration of the job.
- **NFR 4.3 (Maintainability):** The migration recipes must be modular and easily updatable (e.g., YAML or JSON configuration files) to rapidly support new cloud services or API changes.
- **NFR 4.4 (Code Quality):** All generated code must pass a standard linter (e.g., ESLint, Black) and conform to general language best practices.

---

5. Open Questions & Future Considerations

- **How should the agent handle multi-language repositories?** (e.g., Python backend + JavaScript frontend). *Initial approach: Process each language sequentially, generating a single PR.*
- **Should the agent attempt to refactor resource configuration (e.g., S3 bucket policies)?** *Initial approach: Identify policy files and flag them in the MAR for manual review, but only refactor resource identifiers/ARNs in code.*
- **What is the maximum supported repository size for the initial release?** *Suggest: Limit to repositories under 50,000 LoC initially to manage complexity.*

Would you like to focus on the **technical architecture** of the Repository Indexing and Cross-File Dependency mapping next?