| **Aspect** | **Monorepo** | **Polyrepo** |
| --- | --- | --- |
| **Dependency Management** | Simplified across projects but requires updates everywhere. | Complex and may require tooling to synchronize dependencies. |
| **Scalability** | Challenging as the repository grows; needs advanced tools. | Easier to scale with modular, smaller repositories. |
| **Access Control** | Harder to implement fine-grained controls. | Simplified with repo-specific permissions. |
| **Coordination** | Easier for projects with high interdependence. | Requires more effort for cross-repository changes. |

**Key Differences**

| **Feature** | **Monorepo** | **Polyrepo** |
| --- | --- | --- |
| **Structure** | Single repository for all projects and dependencies. | Each project/service has its own repository. |
| **Code Sharing** | Simplified sharing through a unified repository. | Code sharing requires publishing shared libraries (e.g., via package managers). |
| **Dependency Management** | Easier to manage shared dependencies (single version). | Each repository manages its dependencies independently. |
| **Tooling Complexity** | Requires advanced tooling to manage build, CI/CD, and code ownership. | Simpler tooling, each repo has its own CI/CD pipeline. |
| **Atomic Changes** | Supports changes across multiple projects in a single commit. | Changes across projects require updates in separate repositories. |
| **Scalability** | Can be challenging for very large codebases without specialized tooling. | Scales well but can lead to duplication of effort. |
| **Team Autonomy** | Limited autonomy as teams depend on central policies and structures. | High autonomy, teams manage their repos independently. |
| **Collaboration** | Easy for teams working on interrelated projects. | Collaboration across repos can be harder to coordinate. |
| **Performance** | Can suffer from performance issues in large repositories. | Repos are smaller, so performance isn’t typically a concern. |