To achieve the workflow you're describing, where developers can publish and deploy code to a development database for testing without PR (Pull Request) reviews, and then have their code reviewed before it's merged into a more stable environment like staging, you don't necessarily need a separate repository. This can be managed with a combination of branch strategies, access control, and continuous integration/continuous deployment (CI/CD) pipelines. Here's a general approach:

**Branch Strategy**

1. **Development Branches**: Each developer works on their own feature or bugfix branch. These branches are created from the **develop** branch (or whatever your main development branch is).
2. **Develop Branch**: This branch serves as the main branch where all development happens. Developers merge their feature branches into this branch after initial testing.
3. **Staging Branch**: This branch is used for pre-production. Code here is what you intend to go to production next. It should be stable and undergo rigorous testing.
4. **Master/Main Branch**: This branch reflects what's in production.

**Access Control and Workflow**

1. **Development Environment**: Developers can merge their feature branches into the **develop** branch without PR reviews. This allows them to test their changes in a shared development environment.
2. **Staging Environment**: Once developers are confident in their changes in the **develop** branch, they create a PR to the **staging** branch. This PR requires review by a tech lead or architect.
3. **Production Environment**: After staging has been thoroughly tested and is deemed stable, changes can be merged into the **master** branch, which might automatically trigger a production deployment.

**CI/CD Pipeline**

1. **Automated Testing and Deployment**: Set up CI/CD pipelines for each environment.
   * For the **develop** branch, the pipeline can automatically deploy changes to the development environment and run tests.
   * For the **staging** branch, the pipeline can deploy to a staging environment after PR approval.
   * For the **master** branch, the pipeline can handle deployments to production.
2. **Database Migrations**: Ensure your CI/CD pipeline can handle database migrations. This might involve different configurations or scripts for each environment.

**Environment Configuration**

* Each environment (development, staging, production) should have its own database instance to avoid conflicts and to ensure realistic testing.

**Tools**

* **Version Control System**: Such as Git.
* **CI/CD Tools**: Such as Jenkins, GitLab CI, GitHub Actions, etc.
* **Project Management Tools**: For tracking tasks, bugs, and features (e.g., JIRA, Trello).

**Summary**

* Developers work in feature branches and merge into **develop** for initial testing.
* Code is reviewed before being merged into **staging**.
* Final production deployment happens from the **master** branch.
* CI/CD pipelines automate testing and deployment processes.
* Access control ensures the right level of review at each stage.

This approach allows for rapid development and testing in a shared environment while still maintaining code quality and stability before production releases. It leverages existing tools and practices in modern software development workflows.