After live synchronization: accepting a repository cutover

2026-11-24 · 8plus

On November 19, 2026, GitHub announced general availability of Enterprise Live Migrations from Azure DevOps Services to GitHub Enterprise Cloud with data residency. Continuous synchronization lets a team schedule its final cutover. A synchronized repository, however, still needs delivery acceptance.

Define the boundary of the move

The announcement covers Git history and pull requests, with branch policies becoming rulesets. Azure Boards work items and Azure Pipelines definitions remain in Azure DevOps. The whole development system does not move as a single unit.

The following is an acceptance approach proposed by 8plus, not a measured migration case study or a promise of zero downtime. Confirm product eligibility and operational limits against the official documentation and your environment.

Accept a working path

Consider a daily path: open a branch from a work item, submit a pull request, wait for checks, merge, and deploy. A successful clone verifies only one part. Choose a test change without sensitive information and exercise the entire path.

Keep observable results for each step: the expected branch exists, the test commit matches, required checks actually prevent an invalid merge, and deployment targets the intended environment. Do not bypass protections to make a demonstration pass.

Agree on the cutover while synchronization continues

Continuous synchronization cannot decide when the team should stop using one writing destination and start using another. Agree on the final source check, the moment old writes stop, the conditions for opening the new destination, and the person authorized to declare completion.

Record the source commit, target commit, and check time together. If the source receives another write, compare again. An earlier screenshot is not evidence of the current state. The handover should tell the next operator exactly which state was accepted.

Check dependencies that remain behind

When work items and pipeline definitions stay in the original system, the new repository is only one part of the workflow. Verify connections, trigger conditions, service identities, and deployment destinations separately. Complete-looking pull request history does not establish that delivery works.

Keep system names, responsible roles, and check results in the acceptance record. Leave credentials and unnecessary personal or customer information out of public issues. An untested connection should be listed as unfinished, rather than described as probably working.

Plan for writes made after the move

Pointing people back to the old URL may not restore a workflow once new commits or discussions exist at the destination. A recovery plan should explain which new writes must be preserved, who compares them, and when further switching stops. Otherwise, both systems can become competing sources of truth.

In a rehearsal environment, deliberately fail a required check and observe whether everyone knows where to stop. This tests the operating agreement without damaging production.

Make the next working day possible

A useful handover answers five questions: where to work, which state was accepted, which workflow passed, what remains unresolved, and who handles a failure. Better migration tooling reduces movement overhead; a clear acceptance record makes the work transferable.

Sources and related reading

- GitHub announcement, November 19, 2026
- Identity checks before delivery
- State checks after PR notifications