| **#** | **Environment** | **Scenario** | **Description** | **Steps** | **Route Maps** |
| --- | --- | --- | --- | --- | --- |
| 1 | **DEV** | Bug Found in DEV | A bug is identified during development of a Synapse pipeline. | 1. Create a new branch from DEV. 2. Apply the fix in the pipeline code. 3. Test the fix in the DEV environment. 4. Push the changes to GitHub. 5. Merge into DEV once validated. | DEV ➜ [Fix Branch] ➜ DEV (merge) ➜ GitHub |
| 2 | **DEV** | New Pipeline Development | Developing new pipelines or enhancements in Synapse. | 1. Create a feature branch from DEV. 2. Implement the new pipeline. 3. Test locally using Synapse. 4. Push changes to GitHub. 5. Create a PR to merge into DEV. | DEV ➜ [Feature Branch] ➜ GitHub ➜ DEV (PR & merge) |
| 3 | **DEV** | Simultaneous Pipeline Development & Hotfix in PROD | Multiple developers are working on pipelines in DEV while a critical bug is discovered in PROD. | 1. Create a hotfix branch from PROD. 2. Apply and test the fix in the pipeline. 3. Merge the hotfix into PROD. 4. Backport the hotfix to relevant feature branches in DEV. | PROD ➜ [Hotfix Branch] ➜ PROD (merge) ➜ DEV (backport) |
| 4 | **DEV** | Stale Pipeline Branches | A pipeline branch in DEV becomes stale, complicating merging into QAT/PROD. | 1. Regularly rebase or merge from DEV into the feature branch. 2. Resolve any conflicts before creating a PR. 3. Conduct testing before merging into QAT. | DEV ➜ [Rebase/Merge] ➜ [Feature Branch] ➜ QAT |
| 5 | **DEV** | Data Source Changes | Changes to underlying data sources that affect existing pipelines. | 1. Assess the impact of data source changes. 2. Update pipeline configurations accordingly. 3. Test all affected pipelines before deploying to QAT and PROD. | DEV ➜ [Update Configurations] ➜ QAT ➜ PROD |
| 6 | **DEV** | Pipeline Dependencies | A pipeline relies on other pipelines or datasets that are modified. | 1. Identify dependencies between pipelines. 2. Update dependent pipelines accordingly. 3. Test all pipelines in QAT before production deployment. | DEV ➜ [Dependency Update] ➜ QAT ➜ PROD |
| 7 | **DEV** | Automated Testing Failures | Automated tests fail during deployment to QAT. | 1. Investigate and resolve the causes of test failures. 2. Perform manual testing if necessary. 3. Fix the code and retest in DEV before re-deploying to QAT. | DEV ➜ [Fix Code] ➜ QAT (retest) |
| 8 | **DEV** | Scheduled Pipeline Runs | Issues with scheduled runs causing data inconsistencies. | 1. Review the schedule and the last successful runs. 2. Manually trigger or fix the scheduled triggers as necessary. 3. Ensure the pipeline handles data accurately after a failure. | DEV ➜ [Schedule Review] ➜ QAT (manual triggers) |
| 9 | **QAT** | Bug Found in QAT | A bug is found during testing in the QAT environment of a Synapse pipeline. | 1. Create a hotfix branch from QAT. 2. Apply the fix in the pipeline. 3. Merge the fix back into QAT. 4. Validate the fix before promoting to PROD. | QAT ➜ [Hotfix Branch] ➜ QAT (merge) ➜ PROD (validation) |
| 10 | **QAT** | Pipeline Feature Testing | Testing features of Synapse pipelines that are ready to go into production. | 1. Merge feature branches into QAT. 2. Conduct thorough testing of the pipelines. 3. Identify any bugs or issues. 4. Fix identified issues and retest. | QAT ➜ [Feature Branches Merge] ➜ QAT (test) ➜ PROD |
| 11 | **QAT** | Feature Branch Ahead of QAT | New pipelines are ahead of QAT, but a bug has been found. | 1. Create a hotfix branch from QAT. 2. Apply and test the fix in the pipeline. 3. Merge the hotfix back into QAT. 4. Ensure the fix is merged into relevant feature branches. | QAT ➜ [Hotfix Branch] ➜ QAT (merge) ➜ [Feature Branches Merge] |
| 12 | **PROD** | Bug Found in PROD | A bug is identified in the production environment of a Synapse pipeline. | 1. Create a hotfix branch from PROD. 2. Apply the fix in the pipeline code. 3. Push the hotfix branch to GitHub. 4. Create PR and merge into PROD. 5. Backport fix to QAT and feature branches. | PROD ➜ [Hotfix Branch] ➜ PROD (merge) ➜ [QAT & Feature Branches] |
| 13 | **PROD** | Hotfix and Fast-Track Pipeline | Deploy an existing pipeline feature branch that includes a critical fix to production. | 1. Merge feature branch into PROD. 2. Push the changes to GitHub. 3. Validate the fix in production. 4. Re-test in QAT if necessary. | PROD ➜ [Feature Branch] ➜ PROD (merge) ➜ QAT (re-test) |
| 14 | **PROD** | Regression Testing Required | After a fix is applied, ensure that other pipeline features are not affected. | 1. Perform regression tests in QAT and PROD. 2. Document any new issues found. 3. Address any regressions with hotfixes or new feature updates. | QAT ➜ [Regression Testing] ➜ PROD |
| 15 | **PROD** | DEV Ahead of PROD with Bug in PROD | A bug is found in production while the development environment has new features. | 1. Create a hotfix branch from PROD. 2. Apply the fix locally. 3. Push the hotfix branch to GitHub. 4. Create PR and merge into PROD. 5. Backport the fix to DEV. 6. Perform regression testing in both PROD and DEV. | PROD ➜ [Hotfix Branch] ➜ PROD (merge) ➜ DEV (backport) ➜ [Regression Testing] |
| 16 | **PROD** | Rollback a Hotfix in PROD | A hotfix deployed to PROD introduces new issues that were not caught during testing. | 1. Create a rollback branch from PROD before the hotfix was applied. 2. Test the rollback locally. 3. Push the rollback changes to GitHub. 4. Merge them back into PROD. | PROD ➜ [Rollback Branch] ➜ PROD (merge) |
| 17 | **PROD** | User Permissions Issues | Issues arising from insufficient permissions for users to access or modify Synapse Pipelines. | 1. Review user permissions and roles. 2. Adjust permissions in Azure to grant necessary access. 3. Document access changes for auditing. | PROD ➜ [Permissions Review] ➜ PROD (adjust permissions) |
| 18 | **PROD** | Azure Service Outages | Outages in Azure services affecting pipeline execution or deployment. | 1. Monitor Azure service health. 2. Implement contingency plans or fallback procedures. 3. Communicate status updates to stakeholders. | PROD ➜ [Monitor Services] ➜ PROD (implement contingency) |
| 19 | **DEV/QAT** | Deploying Pipelines with Feature Flags | New pipeline features are being developed and tested but need to be gradually rolled out to PROD. | 1. Implement feature flags for new features. 2. Deploy with flags turned off. 3. Monitor feature performance. 4. Gradually enable features based on feedback. | DEV ➜ QAT ➜ PROD (feature flags enablement) |
| 20 | **QAT** | Multi-environment Deployment | Pipelines need to be deployed across multiple environments (e.g., DEV, QAT, UAT) before going to PROD. | 1. Deploy to DEV first and test thoroughly. 2. Move to QAT for integration testing. 3. Once validated, promote to PROD with necessary approvals. | DEV ➜ QAT ➜ PROD |

**Scenarios Involving Multiple Developers**

| **Serial No.** | **Environment** | **Scenario** | **Description** | **Steps** | **Route Map** |
| --- | --- | --- | --- | --- | --- |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | DEV | Simultaneous Feature Development | Multiple developers are working on different features simultaneously. | 1. Each developer creates a feature branch from DEV. 2. Implement respective features. 3. Push changes to GitHub. 4. Create PRs for merging into DEV. | DEV → GitHub → Feature Branches |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | DEV | Merge Conflicts During PR | Two developers submit PRs that affect the same files, leading to merge conflicts. | 1. Review conflicting PRs. 2. Communicate with involved developers. 3. Resolve conflicts locally. 4. Push resolved changes and complete the PR. | DEV → GitHub → Resolve Conflicts → Merge |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | QAT | Multiple Features Under Testing | Multiple new features are merged into QAT for testing by different developers. | 1. Merge feature branches into QAT. 2. Conduct independent testing for each feature. 3. Log any bugs found. 4. Communicate findings to respective developers. | QAT → GitHub → Feature Testing |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | QAT | Coordinated Bug Fixes | Multiple developers are tasked with fixing different bugs found during QAT testing. | 1. Assign bugs to respective developers. 2. Each developer creates a hotfix branch from QAT. 3. Fix and test their respective bugs. 4. Merge fixes back into QAT. | QAT → GitHub → Hotfix Branches |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | PROD | Staggered Feature Releases | Different developers have features ready for release at different times. | 1. Review features for release readiness. 2. Merge approved features into PROD based on priority. 3. Validate and deploy each feature independently. | DEV → QAT → PROD (Staggered Release) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 | PROD | Coordinated Rollback | Multiple developers need to roll back features due to issues found in PROD. | 1. Identify affected features. 2. Communicate with all developers. 3. Create rollback branches. 4. Merge rollback changes into PROD. | PROD → GitHub → Rollback Branches |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 7 | QAT | Feature Overlap with Multiple Developers | Features developed by different developers conflict or overlap during testing. | 1. Identify overlapping features. 2. Collaborate to resolve overlaps. 3. Retest modified features in QAT. 4. Prepare for PROD release. | QAT → Collaborate → Resolve Overlaps |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 8 | DEV | Continuous Integration with Multiple Developers | Developers continuously integrate changes to a shared branch to avoid conflicts. | 1. Set up CI pipeline in Azure DevOps. 2. Developers frequently merge changes to the shared branch. 3. Run automated tests after each merge. | CI Pipeline → GitHub → Shared Branch |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 9 | QAT | Feedback Loop Between Developers | Continuous feedback and collaboration among developers during QAT testing. | 1. Establish regular check-ins or stand-up meetings. 2. Share testing results and progress updates. 3. Make adjustments based on feedback. | QAT → Collaboration → Feedback Loop |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | PROD | Multiple Hotfixes Being Deployed Simultaneously | Multiple hotfixes need to be deployed to PROD from different developers. | 1. Identify urgent hotfixes. 2. Create hotfix branches from PROD. 3. Merge and deploy hotfixes to PROD. 4. Validate each hotfix post-deployment. | PROD → GitHub → Hotfix Branches → Validate |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 11 | DEV | Refactoring Shared Code | Developers refactor shared components or libraries used by multiple features. | 1. Identify components to refactor. 2. Create a branch for refactoring. 3. Refactor code and run tests. 4. Merge changes after validation. | DEV → GitHub → Refactor Branch |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 12 | QAT | Testing Integration of Multiple Features | Testing the integration of several features developed by different teams. | 1. Merge all feature branches into QAT. 2. Conduct integration testing. 3. Log and address any integration issues. | QAT → GitHub → Integration Testing |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 13 | PROD | Pre-Release Checks | Conducting final checks and validations before releasing new features to PROD. | 1. Review features for deployment readiness. 2. Conduct final regression tests. 3. Validate with stakeholders before deployment. | QAT → PROD → Pre-Release Checks |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 14 | DEV | Shared Documentation Updates | Multiple developers need to update shared project documentation. | 1. Identify documentation needs. 2. Assign sections to developers. 3. Update documentation collaboratively. 4. Review changes before publishing. | DEV → GitHub → Documentation Branch |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 15 | QAT | Coordinated Release Readiness | Ensure all features are ready and validated for a production release. | 1. Review feature readiness. 2. Conduct final testing and validation. 3. Prepare release notes and deployment plan. | QAT → Release Planning |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 16 | PROD | Monitoring and Incident Response | Developers monitor the application and respond to incidents reported in production. | 1. Set up monitoring tools. 2. Assign developers to respond to incidents. 3. Create hotfixes for reported issues. 4. Communicate resolutions. | PROD → Monitoring → Incident Response |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 17 | DEV | Development Pair Programming | Two developers work together on a single task to enhance collaboration and knowledge sharing. | 1. Set up a development environment. 2. Collaborate on coding in real-time. 3. Share knowledge and techniques. 4. Complete the task and review. | DEV → Pair Programming |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 18 | QAT | User Acceptance Testing (UAT) | Conducting UAT to gather feedback from users on features before they go live. | 1. Prepare UAT environment. 2. Involve users in testing. 3. Collect feedback and log any issues. 4. Make necessary adjustments based on feedback. | QAT → UAT → Feedback Loop |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 19 | PROD | Post-Release Review | Conducting a review of a release to assess what went well and what can be improved. | 1. Gather team feedback on the release. 2. Review metrics and performance. 3. Document lessons learned for future releases. | PROD → Review → Continuous Improvement |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 20 | QAT | Cross-Functional Testing | Involving different roles (e.g., QA, DevOps) in the testing process to enhance coverage and quality. | 1. Identify cross-functional team members. 2. Collaborate on testing scenarios. 3. Conduct tests and share results. 4. Log and address issues found. | QAT → Cross-Functional Testing |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 21 | DEV | Feature Flag Development | Implementing feature flags to enable or disable features in different environments dynamically. | 1. Develop feature flag functionality. 2. Test flag implementation in DEV. 3. Document usage for other developers. 4. Merge changes. | DEV → Feature Flags → GitHub |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 22 | QAT | Parallel Testing of Multiple Features | Testing multiple features in parallel to speed up the testing process. | 1. Assign developers to test different features concurrently. 2. Log results independently. 3. Communicate any blocking issues. | QAT → Parallel Testing |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 23 | PROD | Real-Time Monitoring After Release | Implementing real-time monitoring after deploying new features to PROD to quickly identify issues. | 1. Set up monitoring dashboards. 2. Track key performance indicators (KPIs). 3. Respond to alerts or issues in real-time. | PROD → Monitoring → Real-Time Response |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 24 | QAT | Regression Testing for New Features | Conducting regression tests after new features are added to ensure existing functionality remains intact. | 1. Develop a regression test plan. 2. Execute regression tests in QAT. 3. Log any new issues. 4. Communicate results with the team. | QAT → Regression Testing |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 25 | DEV | Code Review Practices | Establishing a culture of code reviews among developers for quality assurance. | 1. Set up code review guidelines. 2. Review each other’s code via PRs. 3. Provide constructive feedback. 4. Merge changes post-review. | DEV → GitHub → Code Review |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 26 | PROD | Change Management for Releases | Implementing change management practices to track changes going into production. | 1. Document changes before deployment. 2. Review changes with stakeholders. 3. Track deployment and rollback plans. | PROD → Change Management |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 27 | DEV | Collaborative Troubleshooting | Team members troubleshoot issues collaboratively during development. | 1. Identify problematic areas. 2. Gather a team for collaborative debugging. 3. Resolve issues and document solutions. 4. Share findings with the team. | DEV → Collaborative Troubleshooting |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 28 | QAT | Team Stand-Up Meetings | Regular stand-up meetings to discuss testing progress and issues. | 1. Schedule daily or weekly stand-ups. 2. Team members share updates and blockers. 3. Adjust priorities based on discussions. | QAT → Stand-Up Meetings |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 29 | PROD | Continuous Feedback Loop from Users | Collecting and acting on feedback from user’s post-release to improve future features. | 1. Set up feedback channels (e.g., surveys, support tickets). 2. Analyze user feedback. 3. Plan for improvements based on user insights. | PROD → User Feedback Loop |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 30 | QAT | Final Sign-Off from Stakeholders | Obtaining final approval from stakeholders before moving features to PROD. | 1. Present final features and testing results to stakeholders. 2. Address any concerns or questions. 3. Obtain sign-off for deployment. | QAT → Stakeholder Review → Sign-Off |