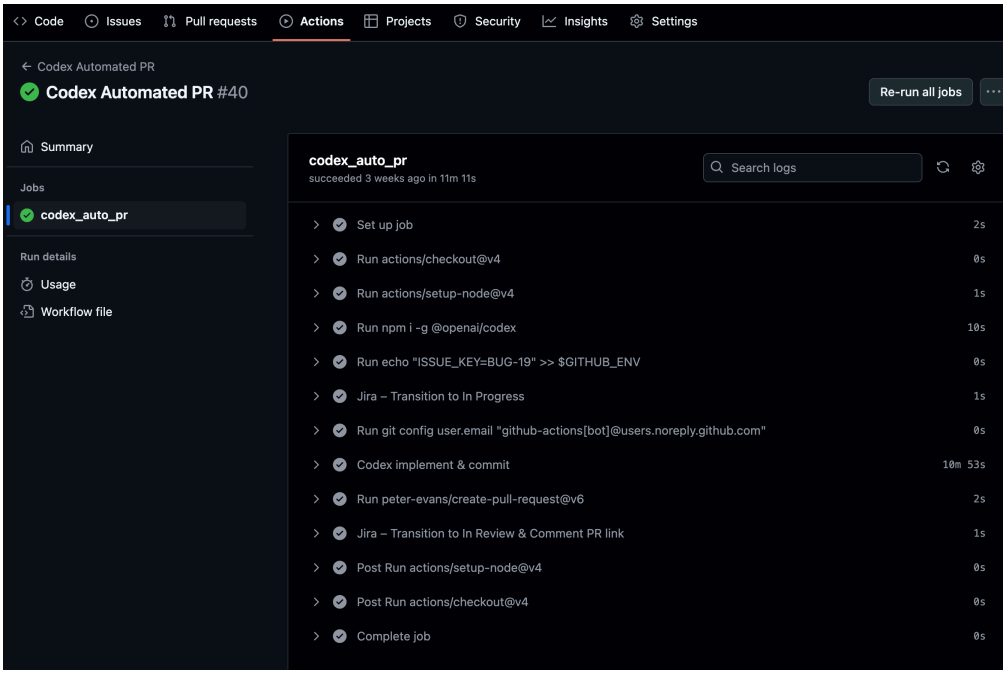
Automate Jira ↔ GitHub with `codex-cli`

Purpose of this cookbook

This cookbook provides a practical, step-by-step approach to automating the workflow between Jira and GitHub. By labeling a Jira issue, you trigger an end-to-end process that creates a **GitHub pull request**, keeps both systems updated, and streamlines code review, all with minimal manual effort. The automation is powered by the `codex-cli` agent running inside a GitHub Action.



The flow is:

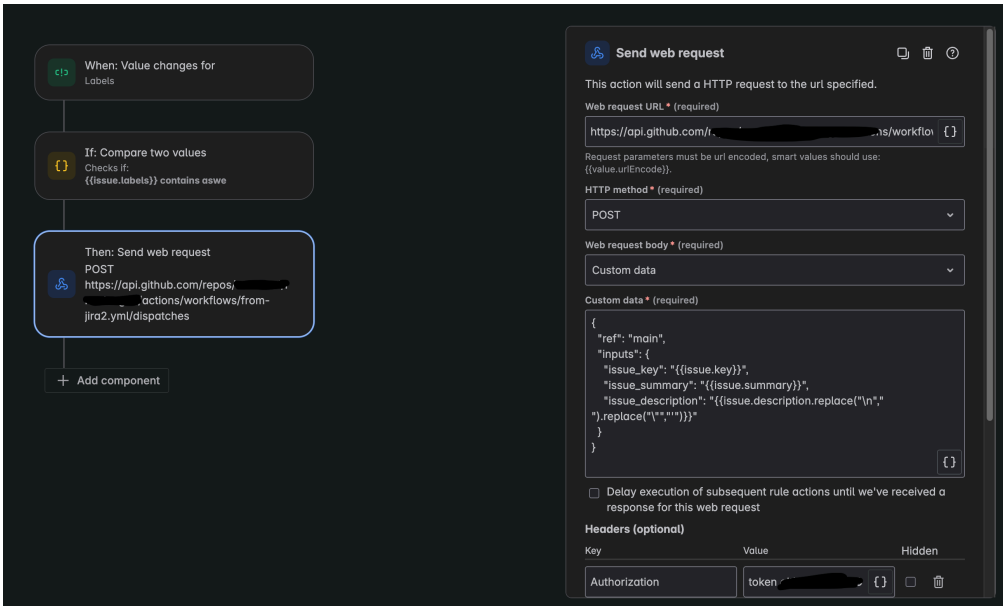
1. Label a Jira issue
2. Jira Automation calls the GitHub Action
3. The action spins up `codex-cli` to implement the change
4. A PR is opened
5. Jira is transitioned & annotated - creating a neat, zero-click loop. This includes

changing the status of the ticket, adding the PR link and commenting in the ticket with updates.

Prerequisites

- Jira: project admin rights + ability to create automation rules
- GitHub: write access, permission to add repository secrets, and a protected `main` branch
- API keys & secrets placed as repository secrets:
  - `OPENAI_API_KEY` – your OpenAI key for `codex-cli`
  - `JIRA_BASE_URL`, `JIRA_EMAIL`, `JIRA_API_TOKEN` – for REST calls from the action
- `codex-cli` installed locally (`pnpm add -g @openai/codex`) for ad-hoc testing
- A repository that contains a `.github/workflows/` folder

Create the Jira Automation Rule



The first step in this rule listens for changes to an issue's labels. This ensures we only trigger the automation when a label is added or modified—no need to process every update to the issue.

Next, we check whether the updated labels include a specific keyword, in our example we are using `aswe`. This acts as a filter so that only issues explicitly tagged for automation proceed, avoiding unnecessary noise from unrelated updates.

If the condition is met, we send a `POST` request to GitHub's `workflow_dispatch` endpoint. This kicks off a GitHub Actions workflow with the relevant issue context. We pass in the issue key, summary, and a cleaned-up version of the description—escaping quotes and newlines so the payload parses correctly in YAML/JSON. There are additional fields available as variables in JIRA to give the codex agent more context during its execution.

This setup allows teams to tightly control which Jira issues trigger automation, and ensures GitHub receives structured, clean metadata to act on. We can also set up multiple labels, each triggering a different GitHub Action. For example, one label could kick off a quick bug fix workflow, while another might start work on refactoring code or generating API stubs.

Add the GitHub Action

GitHub Actions enable you to automate workflows within your GitHub repository by defining them in YAML files. These workflows specify a series of jobs and steps to execute. When triggered either manually or via a POST request, GitHub automatically provisions the necessary environment and runs the defined workflow steps.

To process the `POST` request from JIRA we will create a Github action with a YAML like below in the `.github/workflows/` directory of the repository:

```yaml
name: Codex Automated PR
on:
  workflow_dispatch:
    inputs:
      issue_key:
        description: 'JIRA issue key (e.g., PROJ-123)'
        required: true
      issue_summary:
        description: 'Brief summary of the issue'
        required: true
      issue_description:
        description: 'Detailed issue description'
        required: true

permissions:
  contents: write           # allow the action to push code & open
        the PR
  pull-requests: write      # allow the action to create and update
        PRs

jobs:
  codex_auto_pr:
    runs-on: ubuntu-latest
```

```yaml
steps:
# 0 — Checkout repository
- uses: actions/checkout@v4
  with:
    fetch-depth: 0       # full history → lets Codex run tests /
    git blame if needed

# 1 — Set up Node.js and Codex
- uses: actions/setup-node@v4
  with:
    node-version: 22
- run: pnpm add -g @openai/codex

# 2 — Export / clean inputs (available via $GITHUB_ENV)
- id: vars
  run: |
    echo "ISSUE_KEY=${{ github.event.inputs.issue_key }}"
    >> $GITHUB_ENV
    echo "TITLE=${{ github.event.inputs.issue_summary }}"
    >> $GITHUB_ENV
    echo "RAW_DESC=${{ github.event.inputs.issue_description }}"
    >> $GITHUB_ENV
    DESC_CLEANED=$(echo "${{
    github.event.inputs.issue_description }}" | tr '\n' ' ' | sed
    's/"/'\''/g')
    echo "DESC=$DESC_CLEANED"
    >> $GITHUB_ENV
    echo "BRANCH=codex/${{ github.event.inputs.issue_key }}"
    >> $GITHUB_ENV

# 3 — Transition Jira issue to "In Progress"
- name: Jira — Transition to In Progress
  env:
    ISSUE_KEY:      ${{ env.ISSUE_KEY }}
    JIRA_BASE_URL:  ${{ secrets.JIRA_BASE_URL }}
    JIRA_EMAIL:     ${{ secrets.JIRA_EMAIL }}
    JIRA_API_TOKEN: ${{ secrets.JIRA_API_TOKEN }}
  run: |
    curl -sS -X POST \
      --url   "$JIRA_BASE_URL/rest/api/3/issue/$ISSUE_KEY/
    transitions" \
      --user  "$JIRA_EMAIL:$JIRA_API_TOKEN" \
      --header 'Content-Type: application/json' \
      --data  '{"transition":{"id":"21"}}'
```

```
      # 21 is the transition ID for changing the ticket status to
    In Progress. Learn more here:
    https://developer.atlassian.com/cloud/jira/platform/rest/v3/
    api-group-issues/#api-rest-api-3-issue-issueidorkey-
    transitions-get

# 4 — Set Git author for CI commits
- run: |
    git config user.email "github-
    actions[bot]@users.noreply.github.com"
    git config user.name  "github-actions[bot]"

# 5 — Let Codex implement & commit (no push yet)
- name: Codex implement & commit
  env:
    OPENAI_API_KEY:  ${{ secrets.OPENAI_API_KEY }}
    CODEX_QUIET_MODE: "1"          # suppress chatty logs
  run: |
    set -e
    codex --approval-mode full-auto --no-terminal --quiet \
          "Implement JIRA ticket $ISSUE_KEY: $TITLE. $DESC"

    git add -A
    git commit -m "feat($ISSUE_KEY): $TITLE"

# 6 — Open (and push) the PR in one go
- id: cpr
  uses: peter-evans/create-pull-request@v6
  with:
    token:  ${{ secrets.GITHUB_TOKEN }}
    base:   main
    branch: ${{ env.BRANCH }}
    title:  "${{ env.TITLE }} (${{ env.ISSUE_KEY }})"
    body: |
      Auto-generated by Codex for JIRA **${{ env.ISSUE_KEY }}**.
      ---
      ${{ env.DESC }}

# 7 — Transition Jira to "In Review" & drop the PR link
- name: Jira — Transition to In Review & Comment PR link
  env:
    ISSUE_KEY:      ${{ env.ISSUE_KEY }}
    JIRA_BASE_URL:  ${{ secrets.JIRA_BASE_URL }}
    JIRA_EMAIL:     ${{ secrets.JIRA_EMAIL }}
    JIRA_API_TOKEN: ${{ secrets.JIRA_API_TOKEN }}
    PR_URL:         ${{ steps.cpr.outputs.pull-request-url }}
```

```
      run: |
        # Status transition
        curl -sS -X POST \
          --url   "$JIRA_BASE_URL/rest/api/3/issue/$ISSUE_KEY/
        transitions" \
          --user  "$JIRA_EMAIL:$JIRA_API_TOKEN" \
          --header 'Content-Type: application/json' \
          --data  '{"transition":{"id":"31"}}'
          # 31 is the Transition ID for changing the ticket status to
        In Review. Learn more here: https://developer.atlassian.com/
        cloud/jira/platform/rest/v3/api-group-issues/#api-rest-
        api-3-issue-issueidorkey-transitions-get

        # Comment with PR link
        curl -sS -X POST \
          --url   "$JIRA_BASE_URL/rest/api/3/issue/$ISSUE_KEY/
        comment" \
          --user  "$JIRA_EMAIL:$JIRA_API_TOKEN" \
          --header 'Content-Type: application/json' \
          --data
        "{\"body\":{\"type\":\"doc\",\"version\":1,\"content\":[{\"type\":\"paragraph
        created: $PR_URL\"}]}]}}"
```

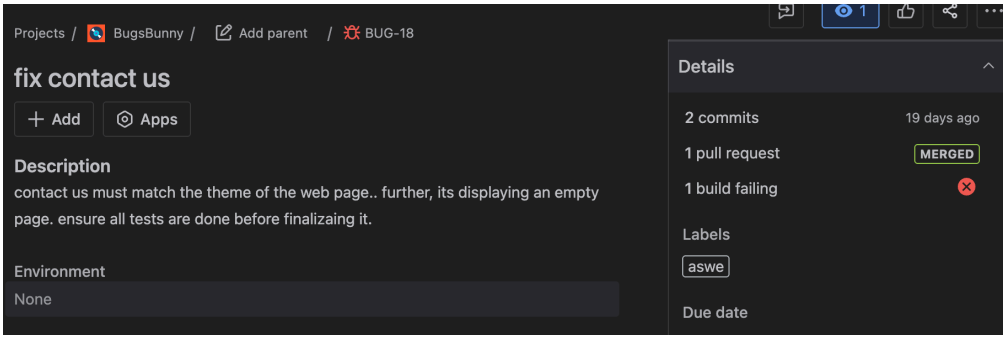
Key Steps in the Workflow

1. **Codex Implementation & Commit** (Step 5)
   - Uses OpenAI API to implement the JIRA ticket requirements
   - Runs codex CLI in full-auto mode without terminal interaction
   - Commits all changes with standardized commit message
2. **Create Pull Request** (Step 6)
   - Uses peter-evans/create-pull-request action
   - Creates PR against main branch
   - Sets PR title and description from JIRA ticket info
   - Returns PR URL for later use
3. **JIRA Updates** (Step 7)
   - Transitions ticket to “In Review” status via JIRA API
   - Posts comment with PR URL on the JIRA ticket
   - Uses curl commands to interact with JIRA REST API

Label an Issue

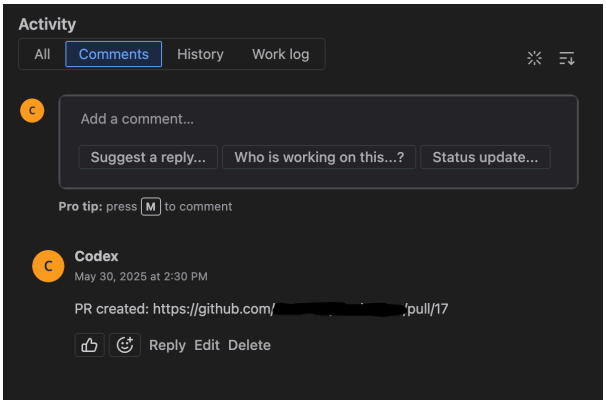
Attach the special `aswe` label to any bug/feature ticket:

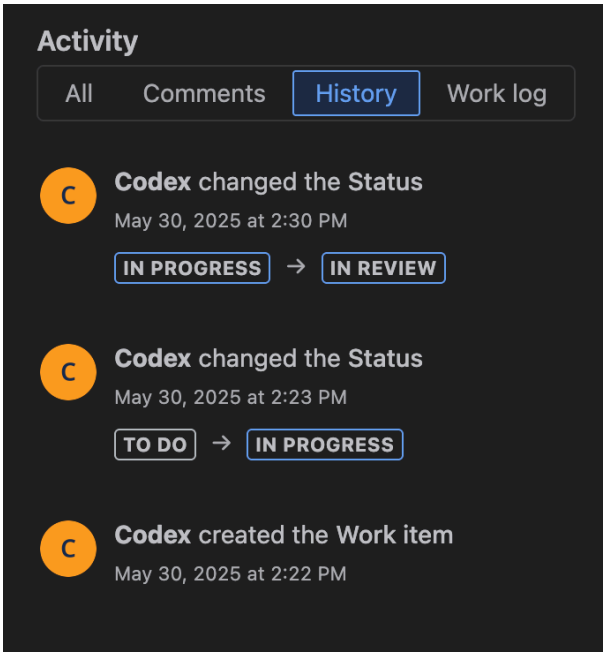
1. **During creation** – add it in the “Labels” field before hitting *Create*
2. **Existing issue** – hover the label area → click the pencil icon → type `aswe`



End-to-end Flow

1. Jira label added → Automation triggers
2. `workflow_dispatch` fires; action spins up on GitHub
3. `codex-cli` edits the codebase & commits
4. PR is opened on the generated branch
5. Jira is moved to **In Review** and a comment with the PR URL is posted
6. Reviewers are notified per your normal branch protection settings





Review & Merge the PR

You can open the PR link posted in the JIRA ticket and check to see if everything looks good and then merge it. If you have branch protection and Smart Commits integration enabled, the Jira ticket will be automatically closed when the pull request is merged.

Conclusion

This automation streamlines your development workflow by creating a seamless integration between Jira and GitHub:

- **Automatic status tracking** - Tickets progress through your workflow without manual updates

- **Improved developer experience** - Focus on reviewing code quality instead of writing boilerplate code
- **Reduced handoff friction** - The PR is ready for review as soon as the ticket is labeled

The `codex-cli` tool is a powerful AI coding assistant that automates repetitive programming tasks. You can explore more about it here