Rundeck Job Migration to Github Runners

We can migrate Rundeck jobs to GitLab Runner (or GitHub Actions) workflows—but it's not a one-click, built-in migration. Instead, it involves transforming our Rundeck jobs into CI pipeline scripts.

How the workflows differ:

- **Rundeck** stores job definitions (YAML/XML) and executes via its own infrastructure on schedule or demand.
- **GitLab Runner** executes jobs defined in `.gitlab-ci.yml` using GitLab's CI/CD engine, triggered by commits, schedules, or API calls.

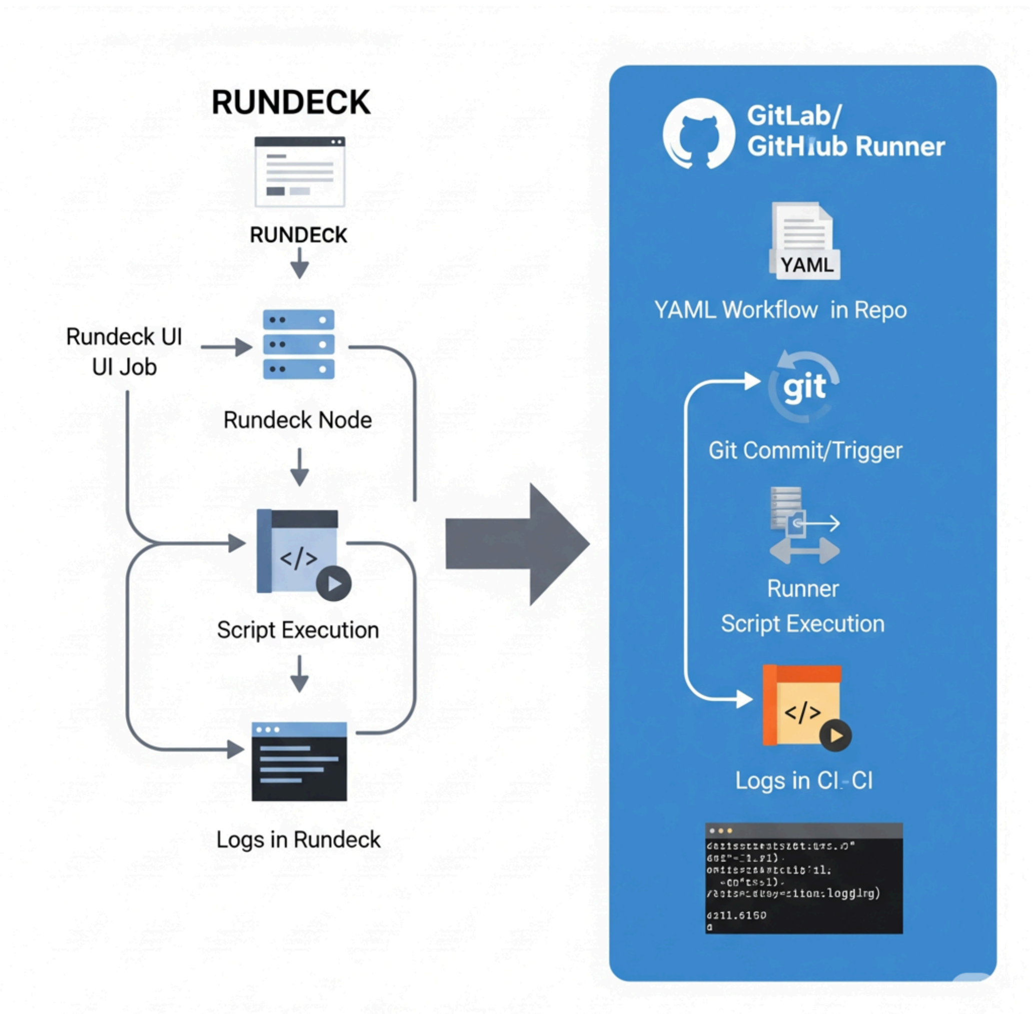
Key differences

Feature	Rundeck	GitLab/GitHub Runner
Trigger Types	Manual, schedule, event, API	Commit/push, schedule, API, manual
Job Definition	XML/YAML via UI or SCM plugin	YAML in repo (`.gitlab-ci.yml` or `.github/workflows`)
Execution Location	Nodes/agents defined in Rundeck	Runner machine/container
Access Control	RBAC in Rundeck	RBAC in GitLab/GitHub + repo permissions
Auditing	Built-in execution logs & history	Pipeline logs, commit history

Migration :

1. **Extract Rundeck job definitions**:
   - Use **SCM Export** to push jobs as YAML/XML into Git (including GitHub, GitLab, etc.) .
   - Or export manually via the UI ("Export Archive" or specific job export) .
2. **Translate job logic into CI pipelines**:
   - Convert individual step logic (shell commands, scripts) into stages in `.gitlab-ci.yml`.
   - Manage environment variables via GitLab's CI/CD settings.
   - Reuse any scripts in your repo, or migrate workflow logic as-is.

A real-world migration example: A team moved from Rundeck + Ansible to GitLab CI/CD, rewriting their deploy/build jobs inside `.gitlab-ci.yml`. It involved storing Docker build logic, SSH-based deployment, environment variable management, and secret handling via GitLab's UI



```
[illegible]
[illegible]
[illegible]
  [illegible]
[illegible]ion:logging)

d211.6160
d
```

Migration Approach

Step 1: Export Rundeck Jobs

- Use Rundeck **SCM Export Plugin** to store job definitions in Git (supports GitLab & GitHub).
- Or export via UI as job definition files (YAML/XML).

Step 2: Analyze Job Steps

- Identify scripts, commands, and integrations used.
- Note environment variables, secrets, and node targeting logic.

Step 3: Map to CI/CD Pipelines

- Create `.gitlab-ci.yml` or `.github/workflows/my-job.yml`.
- Translate Rundeck “steps” → CI “stages/jobs”.
- Replace node execution with appropriate runners or SSH steps.

Step 4: Handle Secrets & Config

- Move Rundeck job options/variables to GitLab/GitHub secrets.
- Configure runner environment accordingly.

Step 5: Test & Validate

- Run jobs in a test branch.
- Compare results with original Rundeck execution.

Challenges

- **Environment Differences** – Rundeck’s node targeting vs. runner environments.
- **Secret Management** – Different storage & retrieval mechanisms.
- **Scheduling Logic** – GitLab/GitHub cron syntax differs from Rundeck’s.
- **Plugin Replacement** – Rundeck plugins may need alternative implementations.