Hourglass Quality Manager — User Guide

Oct 31, 2026 · @Fizz

Overview

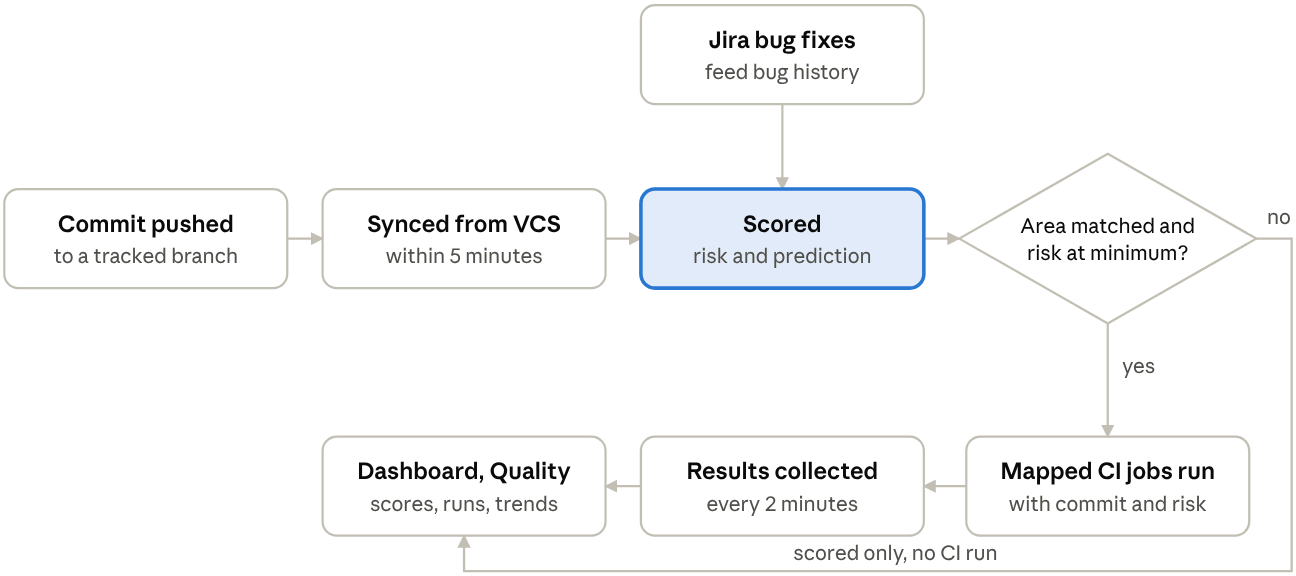
Hourglass Quality Manager scores every new commit for risk, predicts how likely it is to introduce a bug, and runs the right tests in your CI system automatically. It connects to your version control, Jira and CI tools. It learns from your own bug history, and it shows how quality is trending over time.

The main ideas:

- **Projects.** Each product you track is a project, with its own integrations, models and data. Administrators create projects and give people access.
- **Risk score (0–100).** Calculated for each commit from its size, churn, author experience, file types, message, timing, recent activity and the bug history of the files it touches.
- **Defect probability.** A machine-learning model, trained on the commits that introduced past Jira bugs, estimates the chance that a commit introduces a bug.
- **CI triggering.** When a commit touches a source area you defined, Hourglass starts the mapped CI jobs and passes along the commit, risk score and defect probability.
- **Test results.** Hourglass collects the results of those CI runs and shows passed, failed and skipped tests for each run.
- **AI test generation.** On request, Hourglass writes runnable tests for a commit's changes, which you review and add to your repository.
- **Quality trends.** Weekly charts of commit risk, CI test health, bugs and commit activity.

The diagram below shows what happens to a commit, from the moment it is pushed to the moment its results appear.

Every commit is scored; risky changes in mapped areas get tested



life of a commit · 7 steps, 1 decision

Commits outside every source area, or below a mapping's minimum risk, are still scored and counted on the Quality page; they just don't start a CI run.

Installing from AWS Marketplace

Hourglass runs in your own AWS account. You subscribe in AWS Marketplace, then launch a CloudFormation stack that creates the server, storage and permissions; nothing else needs installing.

Before you launch

- **SageMaker quotas** (for defect prediction): in Service Quotas → Amazon SageMaker, request at least 1 for `ml.m5.xlarge for training job usage` and `ml.m5.large for endpoint usage`. New AWS accounts often start at 0. If you choose other instance types in the stack, request those instead.
- **Amazon Bedrock** (for AI test generation): make sure the model `mistral.mistral-large-3-675b-instruct` is available to your account in the stack's region.
- **Permissions:** the person launching the stack needs to be allowed to create IAM roles. The stack creates least-privilege roles for the server and for SageMaker; no access keys are used.

Launch

1. In the listing, choose **Continue to Subscribe**, pick a plan, then **Continue to Configuration** and **Launch with CloudFormation**.
2. Choose a VPC and a subnet that can reach the internet. In us-east-1, don't pick a subnet in **us-east-1e**: it doesn't offer the supported instance types.
3. Set **AllowedCidr** to your office or VPN range (for example `203.0.113.0/24`). Only that range can open the app.
4. Optional: **DomainName** and **TlsEmail** for a trusted HTTPS certificate. Without them the app uses a self-signed certificate and the browser shows a warning.
5. **DatabaseMode:** `rds` (Amazon RDS, recommended for production) or `local` (a PostgreSQL database on its own encrypted disk, with daily snapshots; lower cost).
6. When the stack shows **CREATE_COMPLETE** (about 5 minutes with `local`, 15 with `rds`), open **AppUrl** from its Outputs and create the first account right away: it becomes the administrator.

Upgrades

To move to a new version, update the stack with the new version from AWS Marketplace. The server is replaced, but your data, models and settings are kept. With the local database, the new server takes over the database disk, which means a few minutes of downtime.

Getting help on the server

Application logs are in CloudWatch Logs, log group `/hourglass/<stack name>`. For a shell on the server, use EC2 → the instance → **Connect → Session Manager** (there is no SSH).

Getting started

Accounts are created by an administrator. You receive your email address and a temporary password from them.

1. Open Hourglass in your browser and sign in with your email and the temporary password.
2. Choose your own password (at least 8 characters). Until you do, the rest of the app stays locked.
3. You land on the **Dashboard** of your current project.

The very first account on a new installation is the exception: it is created on the **Sign up** page and becomes the administrator. After that, sign-up is closed.

The navigation bar

Item	What it's for
Dashboard	Risk scores, predictions, CI runs and recent activity for the current project
Quality	Trends over time: risk, CI test health, bugs, commit activity
VCS	Connect GitHub, GitLab, Bitbucket or AWS CodeCommit, and choose repositories and branches
Jira	Connect Jira and choose which Jira projects count
CI/CD	Connect CI systems and map jobs to source areas
Source Areas	Define parts of your code base (for example "auth" or "payments")
Settings	Your profile, your password, and the ML model
Project switcher	Shows the current project; pick another one to switch

On narrower screens these items are in the menu button at the top right.

Switching projects

Click the **Project** box in the navigation bar and pick a project. The page reloads and every view shows that project's data. Hourglass remembers your choice for your next visit.

You only see projects you've been added to. If the box says **No project assigned**, ask an administrator to add you.

Projects and accounts (administrators)

Administrators manage projects and people on the **Projects and accounts** page: open the project switcher and choose **Manage projects and accounts**. Administrators can open every project. Other people see only the projects they belong to, and have full access to them.

Create or rename a project

1. Click **+ New project**, enter a name (and an optional description), then **Create project**. You are added as its first member.
2. To rename a project, select it in the list and click **Rename**.

Project names must be unique. Each project starts empty: connect its version control, Jira and CI systems after switching to it.

Add people

Select the project, then use **Add people**:

- **New account:** enter the person's email (and optionally their name). Tick **Administrator** to make them an administrator. Click **Create account and add**. A temporary password appears once: copy it and send it to them securely. They choose their own password at first sign-in.
- **Existing account:** pick someone who already has an account, then **Add to project**.

If you enter an email that already has an account, that account is simply added; no new password is created.

Remove people or reset a password

In the **Members** table:

- **Remove** takes the person out of this project only. Their account and their other projects are unaffected.
- **New password** issues a fresh temporary password and makes the old one stop working. Use it when someone forgets their password.

A member marked *temporary password* hasn't signed in and chosen their own password yet.

Subscription and seats

Your AWS Marketplace plan sets how many people can use Hourglass. Projects are unlimited on every plan.

Plan	Active users
Team	10
Business	50
Enterprise	Unlimited

Administrators see the plan, the seats in use and the contract end date at the top of **Projects and accounts**, with **Check now** and **Manage subscription**.

- **Subscription required:** without an active subscription, everyone sees this screen after signing in and background syncing stops. Nothing is deleted. Right after subscribing or renewing, an administrator clicks **Check now**; otherwise the app checks every 15 minutes.
- **Grace period:** if the subscription can't be confirmed (for example it lapsed), the app keeps working for 72 hours and administrators see a banner.
- **Seats:** when every seat is in use, new accounts are refused until a seat is freed. Under **All accounts**, an administrator can **Deactivate** someone to free their seat (their data stays) and **Reactivate** them later.
- **Changing plans:** upgrade or downgrade in AWS Marketplace, then click **Check now**. After a downgrade, administrators keep their seats first, then the longest-standing accounts; others see "No seat available" until a seat is freed.

Connecting version control

Hourglass reads commits from your repositories every 5 minutes, scores them, and decides which tests to run. Connect at least one provider on the **VCS** page.

Provider	What you need	Notes
GitHub (and GitHub Enterprise)	A personal access token that can read the repositories	For GitHub Enterprise, also enter the API base URL

Provider	What you need	Notes
GitLab	A personal, group or project access token with read access	A project token works only for its own repository, so add that repository with the token
Bitbucket Cloud	Either a repository or workspace **access token** (starts with `ATCTT`) with Repositories: Read, or an **Atlassian API token** (starts with `ATATT`) created "with scopes" for Bitbucket, including `read:repository:bitbucket`	With an access token, leave the email empty. With an Atlassian API token, also enter the email of the Atlassian account that created it. Add the repository first: the connection is checked against it
AWS CodeCommit	Nothing extra: the server's AWS role is used	Enter the AWS region and the repository name. Only repositories in the same AWS account as the installation can be reached

Steps

1. On **VCS**, choose the provider and enter the token (and base URL or region if needed).
2. Add the repositories to track (owner and name, or the CodeCommit repository name). A repository can have its own token if your main token can't read it.
3. Open **Branches** for a repository and pick the branches to track, for example `main` and `dev`. Only commits on tracked branches are analyzed and can trigger CI.
4. Click **Sync** to fetch commits now instead of waiting for the next 5-minute run.

Good to know

- On the first sync, existing history is imported but **does not trigger CI**. Only commits that arrive after that start test runs.
- Each commit is analyzed for the project that owns its repository. Two projects can track the same repository; each keeps its own data.
- Tokens are stored encrypted when your administrator has enabled encryption (see the appendix).

Existing history: the first sync of a repository stores its history without triggering CI. Hourglass then scores the newest 50 commits of each repository in the background, about 25 every 5 minutes, so risk scores appear within half an hour of connecting.

Connecting Jira and linking commits to bugs

Jira tells Hourglass which commits fixed bugs. From those fixes it works out which earlier commits introduced the bugs, and that history drives both the risk score and the ML model.

Connect

1. On **Jira**, enter your Jira URL (for example `https://yourcompany.atlassian.net`), your Jira email and an API token. Jira Cloud and Jira Server/Data Center both work.
2. Choose the Jira projects that belong to this Hourglass project.

Use a **classic API token** (Atlassian account → Security → Create API token) with your Jira site URL, such as `https://yourcompany.atlassian.net`, and the email of the account that owns the token. If the connection is refused, the message says whether the email and token were rejected or the account can't see any projects.

Mention the issue key in your work

Hourglass links a commit to a Jira issue when the issue key appears in either of these places:

- **The commit message**, for example `PROJ-123 fix null check in login`.
- **The branch name**, for example `feature/PROJ-123-login-timeout`. Every commit on that branch is linked.

Only keys from the Jira projects you selected count, so text like `UTF-8` or `SHA-256` is never mistaken for an issue.

What counts as a bug

Issues of type **Bug**, **Defect**, **Incident** or **Problem**. Commits linked to stories or tasks are linked, but they don't count as bug fixes.

What happens next

After each sync, Hourglass looks at the code each bug fix changed and finds the earlier commit that wrote those lines. That commit is recorded as **bug-introducing**. Files and folders with recent bug fixes then raise the risk of new commits that touch them, and the bug-introducing commits become the training examples for the ML model.

Source areas

A source area names a part of your code base, such as "auth" or "payments", by its file paths. Hourglass only triggers CI for a commit when the commit changes a file in at least one area, so set up your areas before your CI jobs.

1. On **Source Areas**, click to add an area and give it a name.
2. For each repository, list path patterns in glob form, for example `src/auth/**` or `services/payments/**/*.ts`.
3. Save. The area can now be chosen in CI job mappings.

Tips:

- `**` matches any number of folders and `*` matches part of a name.
- A commit that changes files in two areas triggers the jobs mapped to both.
- Changes outside every area (for example documentation only) don't trigger CI. They are still scored and appear on the dashboard and Quality page.

CI/CD integrations

Hourglass starts your existing CI jobs when a risky commit touches a source area, then collects the test results. Set it up on the **CI/CD** page in two parts: connect the CI system, then map jobs to source areas.

1. Connect a CI system

CI system	What you enter
Jenkins	Server URL, user, API token, job name
CircleCI	Project slug (for example `gh/org/repo`), API token, optionally the workflow name or pipeline definition ID

CI system	What you enter
TeamCity	Server URL, access token, build configuration ID
GitHub Actions	Owner, repository, workflow file name, token. The workflow needs a `workflow_dispatch` trigger and must exist on the branch being tested
GitLab CI	GitLab URL, project ID, token
AWS CodePipeline	Region, pipeline name, AWS access key and secret
Azure Pipelines	Organization URL (`https://dev.azure.com/<org>`), Azure project, pipeline ID, personal access token

Use **Test connection** before saving.

2. Map jobs to source areas

A job mapping says: *when a commit changes this area, run this job*. For each mapping choose:

- the **source area** and the **CI integration**
- the **job** to run (defaults to the integration's job)
- the **test suite**, for example `smoke` or `full`
- the **minimum risk score** that triggers it (0 runs it for every matching commit)
- optionally, a **branch pattern** such as `main` or `release/*`

What your job receives

Every triggered run receives the commit hash, branch, test suite, risk score and defect probability as parameters or variables, so your pipeline can check out that exact commit and adjust what it runs.

Run now

Run now starts a mapping's job immediately on the latest commit of the branch (or a commit you choose). Use it to test a mapping without waiting for a new commit.

Before a run, Hourglass re-analyses the commit if its saved defect probability came from an older model or from before a model existed, so CI receives the same numbers the dashboard shows.

Building an exact commit

Jenkins, TeamCity, Azure Pipelines and AWS CodePipeline (V2) build the exact commit Hourglass sends. **CircleCI, GitHub Actions and GitLab CI** can only start a build on a branch: Hourglass passes the commit as a parameter (`commit_hash`, or `COMMIT_HASH` for GitLab), and your pipeline must check it out, otherwise it builds the latest commit on the branch. Run now shows a warning as a reminder. For CircleCI, add this to `.circleci/config.yml` on every branch you build:

```
parameters:
  test_suite:         { type: string, default: "smoke" }
  commit_hash:        { type: string, default: "" }
  risk_score:         { type: string, default: "0" }
  defect_probability: { type: string, default: "0" }

# in the job's steps, right after checkout:
- when:
    condition: << pipeline.parameters.commit_hash >>
    steps:
      - run: |
          git fetch --quiet origin << pipeline.parameters.commit_hash >>
          git checkout --quiet << pipeline.parameters.commit_hash >>
```

GitHub Actions does the same with a `commit_hash` input passed to `actions/checkout` as its `ref`, and GitLab CI with `git checkout $COMMIT_HASH` when the variable is set.

Test results

Hourglass checks run status and fetches test reports every 2 minutes. To see test counts, your pipeline must publish JUnit-style results:

CI system	How to publish results
Jenkins	`junit` step in the job
CircleCI	`store_test_results` step
TeamCity	Any built-in test reporting (read automatically)

CI system	How to publish results
GitHub Actions	Upload the JUnit XML file as a workflow artifact
GitLab CI	`artifacts: reports: junit:` in the job
AWS CodePipeline	A CodeBuild report group in the build stage
Azure Pipelines	`PublishTestResults@2` task

If no report appears after 3 tries, the run shows *no test results* with a hint for that CI system.

The dashboard

The **Dashboard** is the day-to-day view of the current project: what changed, how risky it is, and what CI did about it.

- **Summary tiles:** connected integrations, tracked repositories and synced commits.
- **Commit risk:** recent commits with their risk score, risk level and defect probability. Open a commit to see the factors behind its score and to generate tests for it.
- **ML model status:** whether a trained model is active for this project.
- **CI runs:** recent runs Hourglass triggered, with status, branch, test suite, the risk that triggered them and a **Tests** column (passed / failed / skipped). Expand a run to see its failing tests; tests written by Hourglass carry an **AI** badge.
- **Recent activity:** the latest commits, syncs and runs.

Risk levels are shown with these bands:

Level	Risk score
Low	0–24
Medium	25–49
High	50–74
Critical	75–100

A defect probability of 0 with *ML unavailable* means no model is active for the project yet (see the next section).

Risk scores and defect prediction

Every commit gets two numbers: a **risk score** from fixed rules, and a **defect probability** from a model trained on your project's own bug history.

How the risk score is built

- **The commit itself:** number of files, lines added and removed, file types, message quality and time of day.
- **The author:** how many commits the author has made so far (newer contributors score higher).
- **Recent activity in the repository:** commit frequency and how many different authors touched it in the last 7 and 30 days.
- **Bug history:** Jira bug fixes in the same files and folders over the last 90 days, and how recently the last fix was. This adds up to 15 points. Code with no bug history gets no extra points.

Train the defect model

The model learns which commits tend to introduce bugs. Train it once you have enough history:

1. Go to **Settings → ML Models**. The training data summary shows how many commits are available and how many introduced a bug. The page also shows the training and prediction instance types, and whether training or deployment is in progress.
2. When there are at least **10 bug-introducing commits** and **10 clean commits** with their changed files known, click **Train New Model**. Training and deployment take about 15 minutes; the current model keeps serving until the new one is ready.
3. When it finishes, the new model replaces the old one. Recent commits are then re-scored with it in the background, so the dashboard, the Quality page and CI thresholds all use the new predictions within a few minutes.

Retrain every few weeks, or after many new bugs are fixed, so the model keeps up with your code. Hourglass removes the previous model's endpoint automatically, so you only pay for one.

If training says there aren't enough bug-introducing commits, its message ends with a **Diagnosis** that shows where the data stops, for example Jira not connected, linked issues that aren't Bugs, or fixes that only touched documentation.

If AWS refuses for lack of **SageMaker quota**, the page names the exact quota to request (for example `ml.m5.xlarge for training job usage`) and links to Service Quotas. If only the prediction endpoint's quota was missing, the trained model is kept: once the quota is approved, click **Deploy trained model** instead of training again. Your AWS administrator can also change the instance types in the CloudFormation stack.

Each installation keeps its own models, even when several installations share one AWS account. Upgrading to a new version keeps the current model; there's no need to retrain.

AI test generation

Hourglass can write runnable tests for the source files a commit changed, in the language and test framework your repository already uses. Generation runs only when you ask; nothing is committed to your repository.

1. Open a commit from the dashboard and choose to generate tests.
2. Optionally add **instructions**, for example "focus on expired tokens" or "use pytest fixtures from conftest.py". Instructions are saved with the commit's tests.
3. Wait for generation to finish (usually under a minute). The result lists one test file per changed source file.
4. Review the tests, then copy or download them and add them to your repository yourself.

Good to know:

- Hourglass reads existing tests near the changed files so the new ones match your style, and it skips test cases that already exist.
- Commits with generated tests show a **tests** indicator on the dashboard.
- Once those tests run in CI, they appear with an **AI** badge in the run's test results.
- Generation needs source-code changes. A commit that only touches configuration or documentation has nothing to test.

Quality trends

The **Quality** page shows whether quality is getting better or worse: weekly figures for the last 12 weeks by default, each compared with the previous 12 weeks.

Section	What it shows
Commit risk	Average risk score; high and critical commits per week
CI test health	Test pass rate; failed tests; CI runs and failed runs
Bugs	Jira bugs reported and resolved per week; open bugs at the end of each week; bug-fix commits
Commit activity	Commits, lines changed and active authors

Each number has an arrow showing the change from the previous period, green when it improved and red when it got worse.

- **Time range:** 30 days (daily points), 12 weeks, 6 months or 1 year (weekly points).
- **Filters:** choose a repository and a branch. Bug counts from Jira cover the whole Jira project, because Jira bugs aren't tied to a repository; everything else follows the filters.
- **Show table:** every section can switch to a table of the same numbers.
- Hover a chart to see the exact values for a week.

Your account settings

Open **Settings** in the navigation bar.

- **Profile:** change your first and last name. Your email can't be changed; ask an administrator to create a new account if it needs to.
- **Password:** enter your current password and a new one (at least 8 characters).
- **ML Models:** the training data summary and model training for the current project (see *Risk scores and defect prediction*).

Forgot your password? Ask an administrator for a new temporary password. You'll choose your own again at your next sign-in.

Your session stays signed in through short server interruptions. You're asked to sign in again only when your session has expired. If an administrator issues you a new temporary password, Hourglass asks you to choose a new password using it.

Troubleshooting and FAQ

What you see	Why	What to do
Subscription required	No active AWS Marketplace subscription was found, or it lapsed more than 72 hours ago	Subscribe or renew in AWS Marketplace, then an administrator clicks **Check now**
"No seat available"	Every seat of the plan is in use	Deactivate an account under **All accounts**, or move to a larger plan
"You haven't been added to a project yet"	Your account isn't a member of any project	Ask an administrator to add you
You're sent to *Choose your password*	You signed in with a temporary password	Enter it, then choose your own
No commits appear	The repository or branch isn't tracked, or the token can't read it	Check **VCS**: repositories, **Branches**, then **Sync**
Bitbucket refuses the token	An access token was entered with an email, or an Atlassian API token lacks the Bitbucket scope	The message says which: leave the email empty for an `ATCTT` token; create an `ATATT` token with `read:repository:bitbucket`
CodeCommit can't find the repository	It's in another AWS account or region	Use the region shown in the message; the repository must be in the installation's own account
A commit didn't trigger CI	It changed no file in a source area, its risk was below the mapping's minimum, or its branch doesn't match the mapping	Check the area's path patterns and the mapping's risk and branch settings
CI built the latest commit instead of the one chosen	CircleCI, GitHub Actions and GitLab CI start builds on a branch	Check out `commit_hash` in your pipeline (see *Building an exact commit*)

What you see	Why	What to do
A GitHub Actions run fails at once with a workflow error	The workflow file named in the integration doesn't exist on that branch, or lacks `workflow_dispatch`	Add the workflow to the branch, or fix the file name
A TeamCity run fails with "0 VCS roots"	The build configuration has no VCS root attached	Attach the repository's VCS root in TeamCity
The CI run shows *no test results*	The pipeline doesn't publish a JUnit report	Add the reporting step for your CI system (see *Test results*)
Defect probability is 0 with *ML unavailable*	No model has been trained for this project yet	Train the model in **Settings → ML Models**
Training says "Not enough bug-introducing commits"	Too few Jira Bugs are linked to fixes yet	Read the *Diagnosis* in the message; mention issue keys in fix commits or branch names
Training says "Not enough clean commits"	Too few commits have their changed files stored yet	Wait for the background scoring to catch up (about half an hour after connecting), or keep working: new commits are stored with their files
Training or deployment fails with a SageMaker quota message	The AWS account's quota for that instance type is 0	Request the quota the message names in Service Quotas, then train again or click **Deploy trained model**
AI test generation says there are no source changes to test	The commit only changed configuration, documentation or tests	Pick a commit that changed source files

Does Hourglass change my repository or CI configuration? No. It reads commits, starts jobs you mapped, and reads results. Generated tests are only added when you add them yourself.

How quickly do new commits appear? Within 5 minutes, or straight away with **Sync**. CI status and test results update every 2 minutes.

Can two projects share a repository? Yes. Each project keeps its own commits, scores, CI mappings and model.

Appendix: for administrators

Installed from AWS Marketplace? The CloudFormation stack sets all of these for you, generates the secrets in AWS Secrets Manager and refreshes them at every start, so this appendix doesn't apply. It's for running Hourglass yourself, for example in development.

These settings live in the backend's `.env` file (or the equivalent environment variables). Restart the backend after changing them.

Setting	Purpose	Default
`DB_HOST`, `DB_PORT`, `DB_NAME`, `DB_USER`, `DB_PASSWORD`, `DB_SSL`	PostgreSQL connection	`DB_PASSWORD` has no default and is required
`JWT_SECRET`, `JWT_REFRESH_SECRET`	Sign sign-in sessions; long random values	none
`HQM_ENCRYPTION_KEY`	Encrypts stored integration tokens (32 random bytes, base64)	none: tokens stored unencrypted
`ALLOW_SELF_REGISTRATION`	`true` reopens the Sign up page for everyone	off
`AWS_REGION`	Region for SageMaker, Bedrock and CodeCommit	`us-east-1`
`S3_BUCKET`	Bucket for training data and models	must be set per installation
`IAM_ROLE`	Role SageMaker uses for training and endpoints	must be set per installation
`AWS_BEDROCK_MODEL_ID`	Model used for AI test generation	as configured
`ML_MIN_POSITIVES`	Bug-introducing commits needed to train	`10`
`ML_MAX_TRAINING_COMMITS`	Most recent commits used for training	`5000`
`ML_DELETE_OLD_ENDPOINTS`	`false` keeps previous model endpoints running	on

Encryption key

Generate a key with `node -e "console.log(require('crypto').randomBytes(32).toString('base64'))"`, set `HQM_ENCRYPTION_KEY`, then run `node scripts/encrypt-secrets.js` once to encrypt existing tokens. Back the key up somewhere safe: without it, stored tokens can't be read and every integration must be reconnected.

Maintenance scripts

Run from the `backend` folder.

Script	What it does
`npx knex migrate:latest`	Applies database updates; run after every upgrade
`node scripts/encrypt-secrets.js`	Encrypts tokens still stored in plain text; safe to repeat
`node scripts/cleanup-ml-endpoints.js [projectId] [--delete]`	Lists, or with `--delete` removes, old SageMaker endpoints of a project
`node scripts/resume-ml-deploy.js [projectId]`	Finishes a training run interrupted by a restart (the backend also does this at startup)

Costs to keep in mind

Each project with a trained model keeps one SageMaker endpoint running, billed by the hour. Training runs and AI test generation are billed per use in your AWS account.

On an AWS Marketplace install, AWS bills these resources to your account separately from the subscription: the EC2 server, the database (Amazon RDS, or the local database's disk and snapshots), S3 storage, the always-on SageMaker prediction endpoint (about $0.115 an hour for `ml.m5.large` in us-east-1), training runs while they last, and Amazon Bedrock per request. Deleting the stack keeps the S3 bucket, secrets, database snapshot and local database disk so no data is lost; delete those by hand when you no longer need them, along with any `hourglass-endpoint-` endpoints in SageMaker.