tinybird

Deployment strategies

Deploying changes to a project can be complex, particularly if you are already in production and need to limit disruption to users.

This guide will discuss deployment strategies when you are:

- Adding, updating or deleting resources
- Maintaining streaming ingestion
- Handling user API requests

You'll learn about the default method for implementing Continuous Deployment (CD), how to bypass the default deployment strategy to create custom deployments, and finally, strategies to take into account when migrating data.

How deployment works

With the Git integration:

- Your project in Git is the real source of truth.
- Releases save a reference to the Git commit deployed.
- Multiple Releases allow you to move through "Live", "Preview" or "Rollback" states.

In the default CI/CD workflow templates the `tb deploy` command is used to deploy changes to Releases. This does the following:

- Checks the current commit in the Workspace and validates that is an ancestor of the commit in the Pull Request being deployed. If not, usually you have to `git rebase` your branch.
- Performs a `git diff` from the current branch to the main branch so it can get a list of the Datafiles that changed.
- Deploys them, in the case of CI to a Release in a CI temporary Branch or, in the case of CD to a Release in the Main branch.

Release states

After being deployed, each Release can exist in one of four distinct states. Understanding these states is essential for managing and tracking the lifecycle of your deployments:

1. Live state

The `Live` state is the default state for a Workspace.

This state represents the primary Release currently in production. Any actions via APIs and the CLI that do not specify a specific Release number will be performed on the `Live` Release. There can be **only one** `Live` Release.

2. Preview state

This state allows for rigorous validation, ensuring a Release is fully prepared before it becomes `Live`.

Data Migration: Facilitates data migrations such as backfilling resources before transitioning to `Live`. To learn more about backfilling, refer to the Backfill Strategy guide.

A/B Testing: Provides an environment for A/B testing to evaluate new features and changes effectively.

Performance Testing: Ideal for conducting performance tests to ensure scalability and stability under production load.

Gradual Client Migration: Assists in the gradual migration of clients using the API Endpoints, ensuring schemas/contract compatibility before moving to `Live`.

There can be **only one** Release in `Preview` state.

3. Rollback state

The `Rollback` state allows you to revert to the previous stable version, if necessary.

When a new Release transitions from `Preview` to `Live`, the previous `Live` Release changes to the `Rollback` state. This ensures a reliable and swift rollback mechanism to a stable version, safeguarding against unforeseen issues with the new `Live` Release.

There can be **only one** Release in `Rollback` state. The latest Release that shifts from `Live` to `Rollback` becomes the single rollback Release, replacing any previous rollback one, if it existed.

4. Deploying state

The `Deploying` state is a temporary state that occurs when a Release is being deployed, before entering the `Preview` state. If you see a Release in this state, it means that the `tb deploy` command is being executed or there have been errors during the deployment process.

You can have a maximum of **1 Preview, 1 Live, and 3 Rollback** Releases at any time.

SemVer deployment behavior

Deployments use a version number inspired by semantic versioning (SemVer) that must be incremented for each deployment.

The format of this number is `Major.Minor.Patch-PostRelease` (e.g., `0.1.2-3`).

This version number is passed as an argument to the `tb deploy` CLI command. If you're using the default CI/CD workflows, the value is obtained from the `.tinyenv` file and must be incremented before each deployment. All operations through the API or CLI accept this parameter to determine the specific Release to apply.

The structure of the version number is critical as it dictates the behavior of the deployment. The following scenarios illustrate the different behaviors:

Major version increment (e.g., 1.0.0 to 2.0.0)

- Creates a new Release.
- This Release is left in `Preview` state.
- The deployment uses the `fork downstream` strategy.

Minor or Patch version increment (e.g., 1.0.0 to 1.1.0 or 1.0.1)

- Creates a new Release.
- Begins in `Preview` but is automatically and instantly promoted to `Live` status.
- The previous `Live` version is demoted to `Rollback` status.
- The deployment uses the `fork downstream` strategy.

Post Release increment (e.g., 1.0.0 to 1.0.0-1)

- Does **not** create a new Release.
- Directly updates the existing `Live` Release.
- Does not offer the option to rollback to a Rollback Release.
- The deployment uses the `alter` strategy.

Multiple segment increments (e.g., 1.0.0 to 2.1.0)

When multiple segments of the version number are incremented, the deployment strategy aligns with the leftmost modified segment.

For instance, if both Major and Minor versions are incremented, the strategy for a Major version increment takes precedence.

TB_AUTO_PROMOTE env variable

The `TB_AUTO_PROMOTE` env variable controls the Release behavior. When this variable is disabled, every new Release is treated as a major change. For example, in a scenario where you bump just the patch number,if the `TB_AUTO_PROMOTE` is disabled then Tinybird treats the Release as a major version increment. This means it generates a Preview Release and you will have to manually promote it to Live.

Fork downstream strategy

Recreates the modified resources and **all dependent resources**.

In the case of Data Sources, if you want to preserve old data, backfilling data operations are required after employing this strategy. Some example use cases can be found here: Change the sorting key to a Landing Data Source, Change column type to a Materialized View.

Alter strategy

Updates existing resources that have been changed.

In the case of Data Source, it preserves the data. Particularly useful for tasks like adding a column or changing the Time-To-Live (TTL). It does **not** generate a new Release, which means it cannot be rolled back.

Not all operations can be performed with this strategy. For instance, you cannot change the Sorting Key of a Data Source with this strategy. An example use case can be found here: Add column to a Data Source.

Customizing deployments

The `tb deploy` command allows you to deploy a project with a single command. Under the hood, this command performs a series of actions that are common to most deployments.

However, your project might have specific requirements that mean you need to customize these actions, or run additional actions after the deployment process.

Custom deployments can be particularly useful if using staging and production Workspaces.

> You can use `tb release generate --semver <semver>` as a helper command to create the necessary files as described below.

Custom deployment actions

The `deploy.sh` file allows you to overwrite the default deployment process.

Use custom deployment actions if the default deployment process is not suitable for your project. For example, you might want to deploy resources in a specific order or handle errors differently.

To create custom deployment actions:

- Edit the `.tinyenv` file at the root of your project and increase the `VERSION` environment variable, following the `semver` notation. For example, `0.0.0` to `0.0.1`.
- Create a `deploy/0.0.1/deploy.sh` file and ensure it has execution permissions. For example, `chmod +x -R deploy/0.0.1/`.
- In the `deploy.sh` file, write the Tinybird CLI commands you want to execute during the deployment process.
- The CI/CD pipelines will find the `deploy.sh` file and execute it when deploying the matching version of the project.

Custom post-deployment actions

The `postdeploy.sh` file allows you to add additional post-deployment deployment actions.

Use post-deployment actions to perform data operations after the default deployment has been executed. For example, you might want to backfill data from a previous version of a Data Source to a new version.

To create custom post-deployment actions:

- Edit the `.tinyenv` file at the root of your project and increase the `VERSION` environment variable, following the `semver` notation. For example, `0.0.0` to `0.0.1`.
- Create a `deploy/0.0.1/postdeploy.sh` file and ensure it has execution permissions. For example, `chmod +x -R deploy/0.0.1/`.
- In the `postdeploy.sh` file, write the Tinybird CLI commands you want to execute after the deployment process.
- The CI/CD pipelines will find the `postdeploy.sh` file and execute it when deploying the matching version of the project.

Promote and rollback Releases

There are several methods available to promote a `Preview` Release to `Live`:

1. Using Tinybird Workflow Templates: Employ the 'Tinybird - Releases Workflow' action if you are utilizing the default workflow templates.

2. Promotion via User Interface (UI):: The UI provides an intuitive option to promote Releases.

3. Command Line Interface (CLI) Method:: To promote a Release using the CLI, execute the following command:

PROMOTE PREVIEW TO LIVE

```
tb release promote --semver 2.0.0
```

This command promotes the Release with the specified SemVer (2.0.0) to the `Live` state. The Release that was previously `Live` will transition to the `Rollback` state.

In the event of needing to revert to a previous Release in the `Rollback` state, the process can be initiated either through the UI or via the CLI:

ROLLBACK

```
tb release rollback --semver 1.0.0
```

With this command, the Release currently in the `Live` state is removed, and the specified Release (1.0.0) is reinstated to `Live`. It's important to note that the rollback action is irreversible, emphasizing the need for careful consideration before execution.

Deploying common use cases

Check out this repository to look for common use cases and scenarios or reach us at support@tinybird.co and we'll help you on the best deployment path given your use case.

Next steps

- Practice iterating on one of Tinybird's examples in the Use Case repository
- Learn about backfill strategies

Spain
Calle del Dr. Fourquet, 27
28012 Madrid

USA
41 East 11th Street 11th floor
New York, NY 10003