npm-Based App Distribution for Twenty

Technical Design Document – February 2026

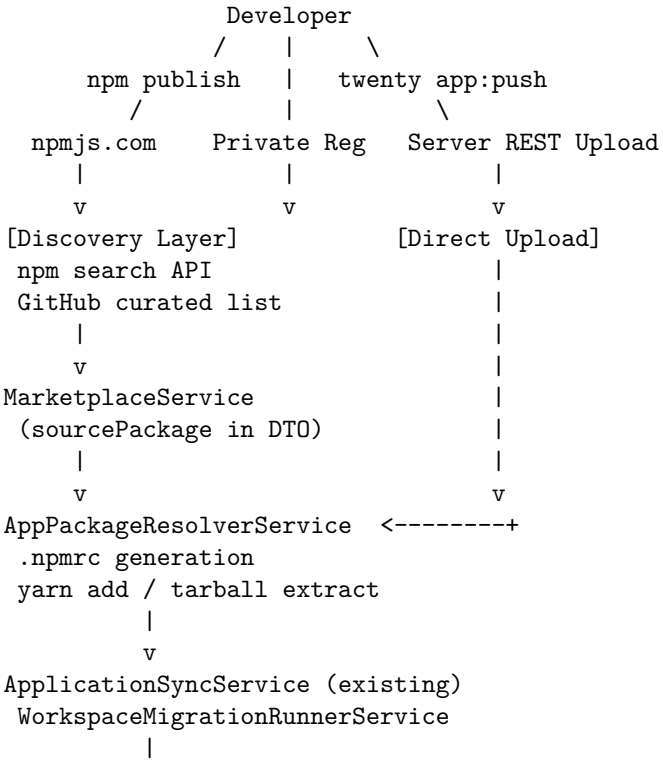
Overview

Add npm registry support for distributing Twenty apps (public and private), with per-AppRegistration registry overrides, direct tarball upload as escape hatch, and version upgrade detection.

Assumptions

- The marketplace install flow (currently a TODO in the resolver and frontend) will be implemented separately. This plan provides the infrastructure that install flow will call into.
- The existing `app:dev` flow (individual file uploads via CLI) remains unchanged.
- The existing GitHub-based marketplace discovery remains as a curated fallback.

Architecture



```
          v
AppRegistration + Application entities
```

Phase 1: Entity and Config Changes

1a. Extend AppRegistrationEntity

Add four columns to `application-registration.entity.ts`:

- **`sourcePackage`** (text, nullable) – npm package name, e.g. `"twenty-app-fireflies"` or `"@myorg/twenty-app-crm"`. Null for tarball-only or OAuth-only apps.
- **`tarballFileId`** (uuid, nullable) – FK to a FileEntity storing a directly-uploaded `.tar.gz`. Null when the app comes from npm.
- **`registryUrl`** (text, nullable) – per-registration npm registry override. Null means "use the server default `APP_REGISTRY_URL`." This is how a single server can pull public apps from npmjs.com while pulling `@mycompany/*` apps from GitHub Packages.
- **`latestAvailableVersion`** (text, nullable) – cached latest version from the registry, updated periodically. Compared against `Application.version` to surface upgrade availability.

Source resolution priority:

1. `sourcePackage` is set → resolve from npm via `yarn add`
2. `tarballFileId` is set → extract from file storage
3. Neither → OAuth-only app, no server-side code

1b. Extend ApplicationEntity.sourceType

Widen the `sourceType` union from `'local'` to `'local' | 'npm' | 'tarball'`:

- `'local'` – existing behavior (CLI `app:dev`, individual file uploads, workspace-custom)
- `'npm'` – installed from an npm registry via `yarn add`
- `'tarball'` – installed from a directly-uploaded tarball

This lets the system distinguish how an app was installed, which matters for upgrade logic (npm apps can check the registry for newer versions; tarball apps cannot).

1c. Add server-wide config variables

Add a new `APP_REGISTRY_CONFIG` group to ConfigVariablesGroup:

- **`APP_REGISTRY_URL`** (string, default `https://registry.npmjs.org`) – default npm registry URL
- **`APP_REGISTRY_TOKEN`** (string, optional, sensitive) – auth token for the default registry

1d. Generate migration

TypeORM migration adding `sourcePackage`, `tarballFileId`, `registryUrl`, `latestAvailableVersion` to `core.applicationRegistration`.

Phase 2: App Package Resolver Service

2a. Create AppPackageResolverService

New service with core method:

```
resolvePackage(appRegistration, options?: { targetVersion? }) → ResolvedPackage | null
```

Returns `{ manifestPath, packageJsonPath, filesDir }` or null for OAuth-only apps.

Resolution logic:

```
if sourcePackage:
  1. Determine registry: appRegistration.registryUrl ?? APP_REGISTRY_URL
  2. Determine auth token for the resolved registry
  3. Generate temporary .npmrc in an isolated working directory
  4. Run: yarn add <sourcePackage>@<targetVersion ?? latest>
  5. Read manifest from node_modules/<sourcePackage>/.twenty/output/manifest.json
  6. Return paths

if tarballFileId:
  1. Download tarball from FileStorageService
  2. Extract to temporary directory
  3. Read manifest from extracted files
  4. Return paths

else:
  return null (OAuth-only)
```

2b. Isolated working directories

Each resolution runs in a temporary directory under `{os.tmpdir()}/twenty-app-resolver/{uuid}/`. This avoids contaminating the server's own `node_modules` and isolates apps from each other. Cleaned up after files are copied to storage.

2c. .npmrc generation

For scoped packages (`@scope/twenty-app-*`):

```
@scope:registry=https://npm.pkg.github.com
//npm.pkg.github.com/:_authToken=TOKEN
```

For unscoped packages with a non-default registry:

```
registry=https://my-verdaccio.internal:4873
//my-verdaccio.internal:4873/:_authToken=TOKEN
```

2d. Post-resolution file transfer

After resolving, copies files into the app's storage path using the existing FileStorageService layout:

```
{workspaceId}/{applicationUniversalIdentifier}/
  built-logic-function/...
  built-front-component/...
  dependencies/package.json
  dependencies/yarn.lock
  public-asset/...
  source/...
```

This reuses the same FileFolder enum paths that `app:dev` uses, so downstream ApplicationSyncService works unchanged.

Phase 3: Marketplace Discovery via npm

3a. Update MarketplaceService

Add npm-based discovery alongside the existing GitHub path:

- Query npm search API: `GET {registryUrl}/-/v1/search?text=keywords:twenty-app&size=250`
- Map each result to MarketplaceAppDTO using package.json metadata

Merge strategy:

1. Fetch from npm search API (apps with `keywords: ["twenty-app"]`)
2. Fetch from GitHub (existing curated list)
3. Merge by `universalIdentifier` – GitHub entries override npm entries (allowing curation)
4. Cache merged result with existing 1-hour TTL

3b. Add sourcePackage to MarketplaceAppDTO

The DTO needs a `sourcePackage: string | null` field so the install flow knows which npm package to resolve. For npm-discovered apps, this is the package name. For GitHub-only apps, this is null.

Phase 4: Version Upgrade Support

4a. Create AppUpgradeService

Periodic version check (npm-sourced apps only):

- Fetches `{registryUrl}/{sourcePackage}/latest` from the npm registry
- Stores result in `AppRegistration.latestAvailableVersion`

- Frontend compares against `Application.version` to show "Update available"

Upgrade trigger:

1. Resolve the new version via AppPackageResolverService
2. Sync via existing ApplicationSyncService (triggers workspace migration for schema changes)
3. Update Application.version

Rollback strategy: If sync fails (e.g., migration validation error), re-resolve the previous version and re-sync. This is possible because npm retains all published versions.

4b. Version check scheduling

Lightweight cron or check-on-access pattern. Iterates over AppRegistrations where `sourcePackage IS NOT NULL` and calls `checkForUpdates()`. Frequency: once per hour, matching the existing marketplace cache TTL.

Phase 5: SDK CLI Commands

5a. Finalize `twenty app:build`

Ensure `.twenty/output/` is npm-publishable. The build step generates a `package.json` in the output directory:

```
{
  "name": "twenty-app-fireflies",
  "version": "1.2.0",
  "keywords": ["twenty-app"],
  "twenty": {
    "universalIdentifier": "a4df0c0f-c65e-44e5-8436-24814182d4ac"
  },
  "files": ["manifest.json", "built-logic-function", "built-front-component", "public-asset"
}
```

The developer then publishes with standard `npm publish` – no custom command needed.

5b. `twenty app:pack` (new command)

`twenty app:pack [appPath]`

- Runs `app:build` if `.twenty/output/` doesn't exist or is stale
- Uses existing TarballService to create `{name}-{version}.tar.gz`
- Outputs the file path for manual distribution

5c. `twenty app:push` (new command)

```
twenty app:push [appPath] --server <url> --token <token>
```

- Runs `app:pack` to produce the tarball
- Reads universalIdentifier from manifest to find or create the AppRegistration
- Uploads via `POST /api/app-registrations/upload-tarball`
- Reports success with the registration ID
- Reuses `twenty auth:login` credentials if `--server` is not specified

Phase 6: Server Tarball Upload Endpoint

6a. REST controller

```
POST /api/app-registrations/upload-tarball
Content-Type: multipart/form-data
Body: tarball file + optional universalIdentifier
```

Validation:

- Max file size: 50MB
- Must be a valid `.tar.gz`
- Extracted contents must contain `manifest.json` with a valid `universalIdentifier`
- The `universalIdentifier` must not conflict with an existing registration owned by a different user

Flow:

1. Extract tarball to temp directory
2. Validate manifest structure
3. Find or create AppRegistration by universalIdentifier
4. Store tarball in FileStorageService under `FileFolder.AppTarball`
5. Set `tarballFileId` on the AppRegistration
6. Return the AppRegistration entity

6b. Add FileFolder.AppTarball

New enum value `AppTarball = 'app-tarball'` in FileFolder.

Key Design Decisions

Decision	Rationale
Per-AppRegistration registry override	`registryUrl` on the entity allows mixing registries. Public apps from npmjs.com, private from GitHub Packages/Verdaccio. Server-wide `APP_REGISTRY_URL` is the fallback.
npm publish is standard	No custom publish infra. Free versioning, README, `npm audit`, download stats, proven auth model.
Tarball as escape hatch	Air-gapped environments, CI pipelines, one-off installs. Cannot auto-upgrade.
sourceType distinction	`'npm' | 'tarball' | 'local'` lets the system know which upgrade path is available. Only npm apps can check for newer versions.
Backward compatible	`app:dev` flow unchanged. GitHub marketplace unchanged. All new fields nullable.
Upgrade rollback	Re-resolve previous version from npm on failure. Safe because npm never deletes published versions.

Edge Cases

- **npm unreachable**: Timeout after 30s, throw clear error. App remains at current installed version.
- **Package name conflicts**: The `universalIdentifier` in the `twenty` field of `package.json` is the source of truth, not the npm package name. Two packages with the same universalIdentifier conflict at the AppRegistration level (unique index).
- **Scoped vs unscoped packages**: Both work. Scoped packages naturally route to a private registry via `.npmrc` scope mapping.
- **Multiple workspaces, same server**: AppRegistration is server-level (core schema). Application is workspace-level. One AppRegistration can be installed in multiple workspaces at different versions.