

# One Codebase, Many Banks

How we build one mobile banking product that ships as a separate app for every bank, with each bank's own features, flows, backend and release schedule, and how we run it safely.

**Status** Proposal     **Audience** product, engineering, operations, bank partners

**Covers** mobile apps, backend, delivery, operations

|                                                                                         |                                                                                |
|-----------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| <b>Tenant flow</b><br>How a bank is created, built into an app and checked on the phone | <b>Feature flow</b><br>How a feature goes from idea to one bank's customers    |
| <b>Per-bank flows</b><br>How one feature works differently for each bank                | <b>Development flow</b><br>Branches, automatic checks and rules for developers |
| <b>Deployment flow</b><br>App builds, backend deployments and versions per bank         | <b>OTA flow</b><br>The weekly JS train, per-bank channels and rollback         |
| <b>Operations flow</b><br>Who changes what, approvals, monitoring, incidents            | <b>One bank's update</b><br>Exactly how each kind of change reaches Bank B     |

## 00 Summary

**The problem.** Many banks want the same mobile banking product. Each needs its own app in the stores, its own set of features, its own way of doing some things, and its own data kept apart. A separate app per bank would multiply cost and bugs with every new bank.

**Our approach.** One product, configured per bank:

- **One app codebase and one backend codebase.** A bank is a tenant pack for the app and a separate deployment of the backend.
- **Features decided by the bank and the server,** through five layers, down to the individual customer. A bank's "off" is final (how to switch a feature).
- **Different flows without different code:** settings, server-driven security steps and journey variants, never `if (bank)` .
- **Five update lanes,** from a live switch in minutes to a store release in weeks. Every code change passes the bank's own UAT first.
- **Operations built in:** an admin portal with two-person approval, an audit log and monitoring for every bank.

**Decided:** one backend codebase, deployed separately for each bank. **To decide:** eight questions in section 15, starting with where each bank's backend runs.

| WHAT BANKS NEED                              | HOW WE DELIVER IT                                                                               |
|----------------------------------------------|-------------------------------------------------------------------------------------------------|
| Many banks, one product                      | One app codebase and one backend codebase; each bank is a tenant pack plus a backend deployment |
| Their own app in the stores                  | A build per bank: its own app ID, name, icon, signing key and store account                     |
| Different features per bank and per customer | Five feature layers; the bank and the server decide; a bank's "off" is final                    |
| Different flows per bank                     | Settings, server-driven security steps and journey variants                                     |
| Their data kept apart                        | A separate backend, database and key pair for every bank                                        |
| Fast, safe updates                           | OTA per bank with UAT sign-off, staged rollout and instant rollback                             |
| Control and accountability                   | Admin portal with two-person approval, audit log, per-bank monitoring                           |
| Low cost per bank                            | A new bank in days; one fix reaches every bank                                                  |

## 01 The problem

Each bank is a separate customer with its own brand, products, rules and approval process. Differences appear at three levels:

| LEVEL    | EXAMPLE                                                       |
|----------|---------------------------------------------------------------|
| Bank     | Bank A offers loans and cards; Bank B offers only payments    |
| Role     | Customers get the banking app; field agents get the agent app |
| Customer | A customer whose KYC is pending cannot apply for a loan       |

Banks also want the same feature to work differently:

| FEATURE            | BANK A                                         | BANK B                                    | BANK C                                              |
|--------------------|------------------------------------------------|-------------------------------------------|-----------------------------------------------------|
| Approve a transfer | TPIN, then OTP                                 | OTP only up to ₹5,000; TPIN and OTP above | Biometric, then TPIN                                |
| Register a device  | Mobile, SMS OTP, MPIN                          | Mobile, debit card and PIN, OTP, MPIN     | Mobile, account number and date of birth, OTP, MPIN |
| Add a payee        | Verify the name, then save                     | Save, then a 24-hour cooling period       | Save with an OTP                                    |
| Loan application   | Product, amount, employment, documents, review | Product, amount, documents, review        | No loans                                            |

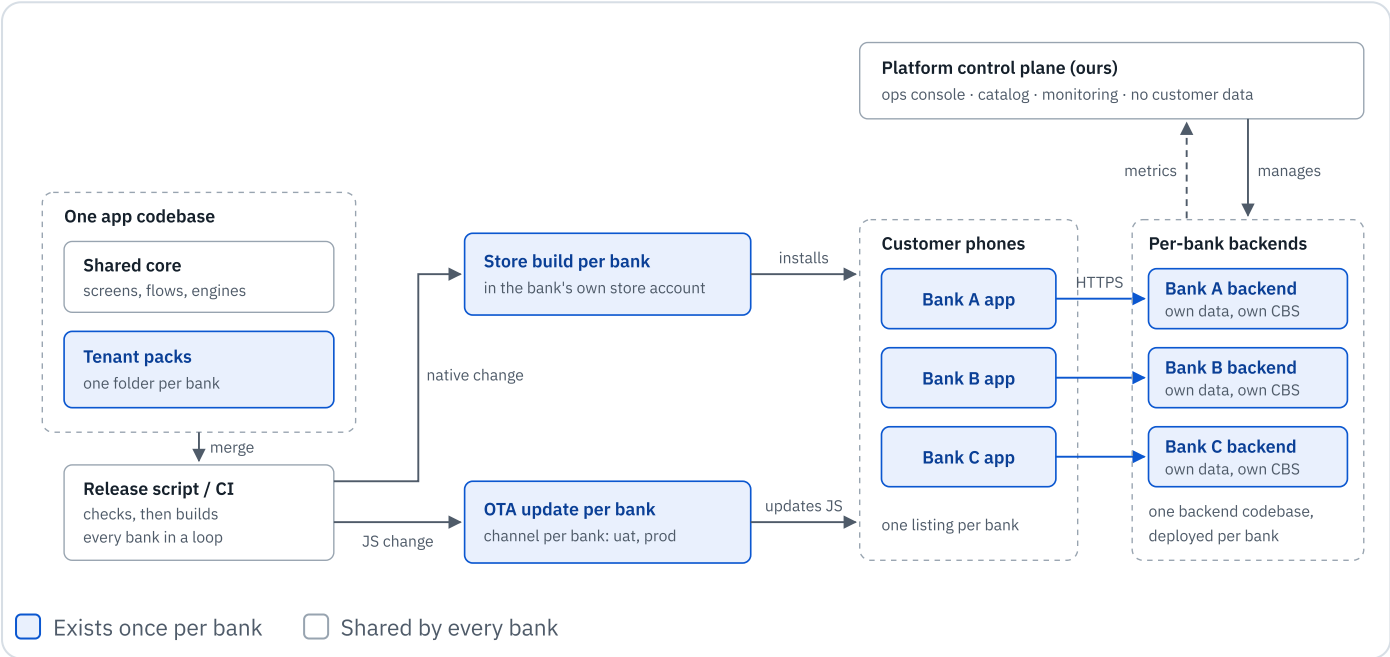
## What the platform must guarantee

| #  | REQUIREMENT                                                               |
|----|---------------------------------------------------------------------------|
| R1 | One codebase and one set of screens for every bank                        |
| R2 | Each bank's own branding, app identity and store listing                  |
| R3 | Per-bank features, changeable without a store release                     |
| R4 | Per-role and per-customer features, decided by the server                 |
| R5 | A bank's "off" can never be overridden by a role, a customer or a device  |
| R6 | Correct behaviour offline and before the network answers                  |
| R7 | Each bank's data, keys and backend kept apart                             |
| R8 | One command releases any bank, and each bank releases on its own approval |
| R9 | A new bank in days, and the server enforces everything the app hides      |

**What we will not do:** a separate codebase per bank, a Git branch per bank, or `if (bank === ...)` checks in shared code. Each of these multiplies cost and bugs with every new bank.

## 02 Our approach

- **One product, configured per bank.** One app codebase and one backend codebase. A bank is a tenant pack for the app and a separate deployment of the backend.
- **Bank differences are data.** They live in tenant packs and server configuration, never in code branches.
- **The server decides access and security.** The app follows the server and never widens what it allows.
- **Fail closed.** An unknown feature, step or setting means off or stop.
- **Every bank is isolated:** its own app build, backend, data, keys, update channels, store listing and approvals.
- **Every change has a lane.** We use the fastest lane a change allows, and no code reaches a bank's customers without that bank's sign-off.



Everything blue exists once per bank: the app build, its OTA channel and its backend deployment. Each app talks only to its own bank's backend. The control plane holds no customer data; it manages the backends over an audited link and collects their metrics.

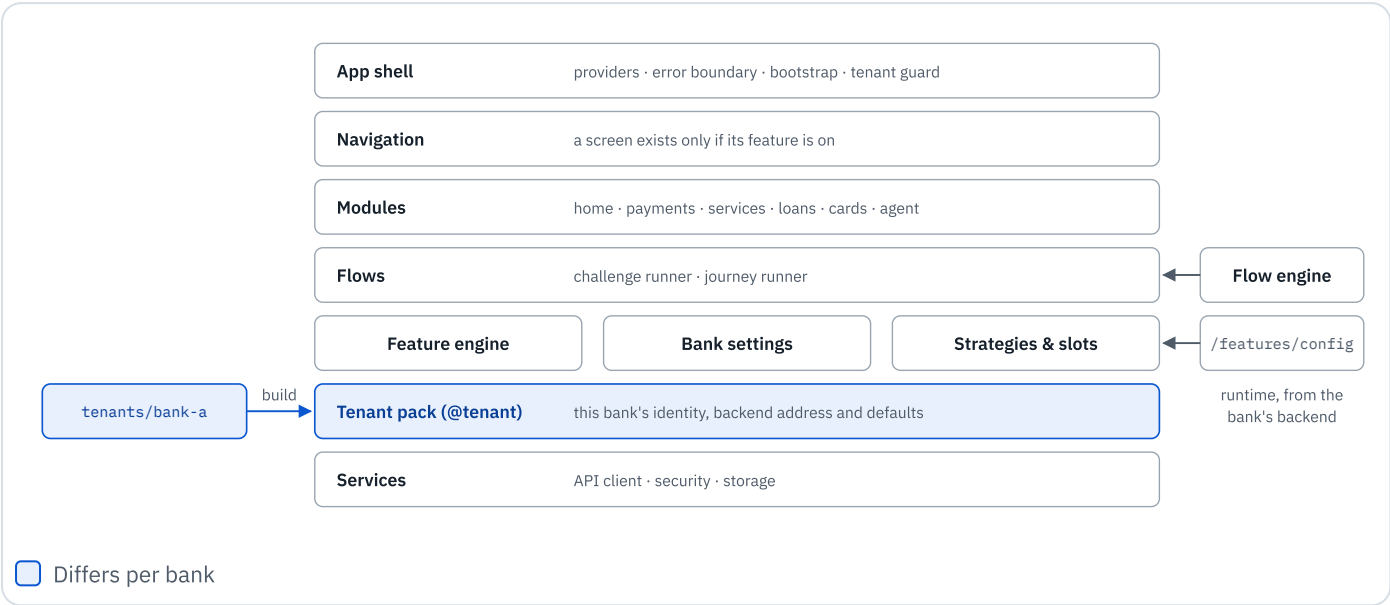
### How each kind of difference is handled

| WHAT DIFFERS BETWEEN BANKS     | HANDLED BY                                               | HOW A CHANGE REACHES PHONES                                             |
|--------------------------------|----------------------------------------------------------|-------------------------------------------------------------------------|
| Identity and branding          | Tenant pack, chosen at build time                        | OTA for text, colours and images; store build for icon, name and app ID |
| Which features the bank offers | Pack defaults and the bank's live switches               | Next config refresh, within minutes                                     |
| Which features a customer gets | Server rules applied at sign-in                          | Next sign-in or refresh                                                 |
| How a feature works            | Settings, server-driven security steps, journey variants | Next config refresh, or the next flow started                           |

| WHAT DIFFERS BETWEEN BANKS | HANDLED BY                        | HOW A CHANGE REACHES PHONES                        |
|----------------------------|-----------------------------------|----------------------------------------------------|
| Where data lives           | The bank's own backend deployment | Not applicable: it never leaves the bank's backend |

### 03 Architecture

#### Inside the app



Screens never read the tenant pack or the server directly. They ask the decision layer ("is it on?", "what value?", "which rule?"), so a screen is the same for every bank.

#### Backend: one codebase, one deployment per bank

The backend follows the same rule as the app: one codebase and one versioned build, deployed separately for each bank and each environment. Banks differ only in configuration: their core-banking connection, database, keys, certificates and feature settings.

| COMPONENT                             | RUNS                   | JOB                                                                                                              |
|---------------------------------------|------------------------|------------------------------------------------------------------------------------------------------------------|
| Mobile and business APIs              | In each bank's backend | Banking APIs, with an entitlement check on every call ( FEATURE_NOT_AVAILABLE )                                  |
| Config service                        | In each bank's backend | The bank's switches, settings and journey variants, plus per-customer entitlements after sign-in                 |
| Flow engine                           | In each bank's backend | The bank's security policy; replies one step at a time (TPIN, OTP, biometric)                                    |
| Bank admin portal                     | In each bank's backend | Bank staff change their own bank, with two-person approval                                                       |
| Audit log                             | In each bank's backend | Every change and every ops action, kept with the bank's own data                                                 |
| Database, key pair, core-banking link | In each bank's backend | The bank's customers and sessions, its PIN encryption key pair, its connection to its core banking system        |
| Ops console                           | Central                | Onboarding, security policies, emergency kill, support lookup; reaches each backend over a private, audited link |

| COMPONENT       | RUNS    | JOB                                                                               |
|-----------------|---------|-----------------------------------------------------------------------------------|
| Feature catalog | Central | Every feature key the app knows; each backend imports it when it is deployed      |
| OTA server      | Central | App bundles and update channels, self-hosted                                      |
| Monitoring      | Central | Crash reports, metrics and logs without customer data, tagged by bank and version |

Environments

| ENVIRONMENT | USED BY                    | BACKEND                             | APP BUILD        | OTA CHANNEL | TEST DATA                  |
|-------------|----------------------------|-------------------------------------|------------------|-------------|----------------------------|
| Dev         | Developers                 | Shared dev backend, or a local mock | bankADevDebug    | none        | Allowed, debug builds only |
| UAT         | Our QA, the bank's testers | Bank A's UAT backend                | bankAUatRelease  | bank-a-uat  | Never mock data            |
| Prod        | Customers                  | Bank A's production backend         | bankAProdRelease | bank-a-prod | Never                      |

**No hand-edited server addresses.** Each bank's backend address and certificate pins, per environment, live in its tenant pack. The build picks them from the bank and the environment.

# 04 Tenant flow

A tenant is one bank. It exists in four places, tied together by one id such as `bank-b` .

## Code

`tenants/bank-b/`

- Identity: org id, name, logo, brand colours, wording
- Backend address and certificate pins per environment
- Feature defaults, settings, flow choices

## Native shell

Android `flavor bankB` · iOS `scheme BankB`

- App ID, launcher icon, home-screen name
- Upload signing key, kept in CI secrets
- OTA channel baked in: `bank-b-prod`

## Backend

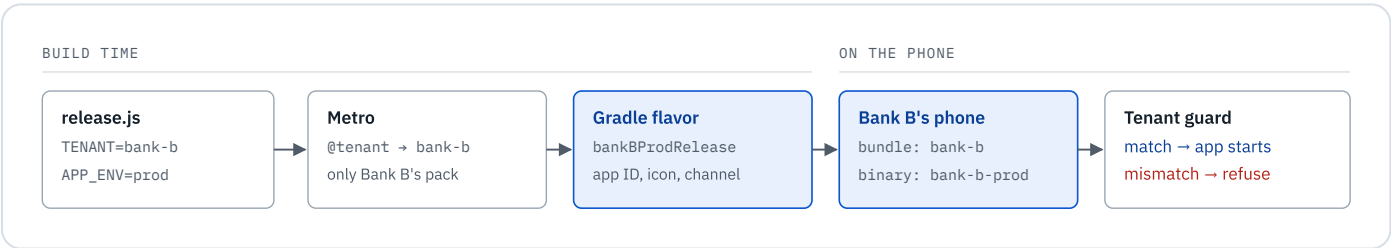
Bank B's own deployment · UAT and prod

- Its own database, key pair and core-banking connection
- Live switches, settings, security policy, customer rules
- Admin portal, approvals, audit log

## Delivery

`tenants/registry.json` · `stores` · OTA server

- Store listing in Bank B's own developer account
- OTA channels `bank-b-uat` and `bank-b-prod`
- One registry entry: flavor, app IDs, version

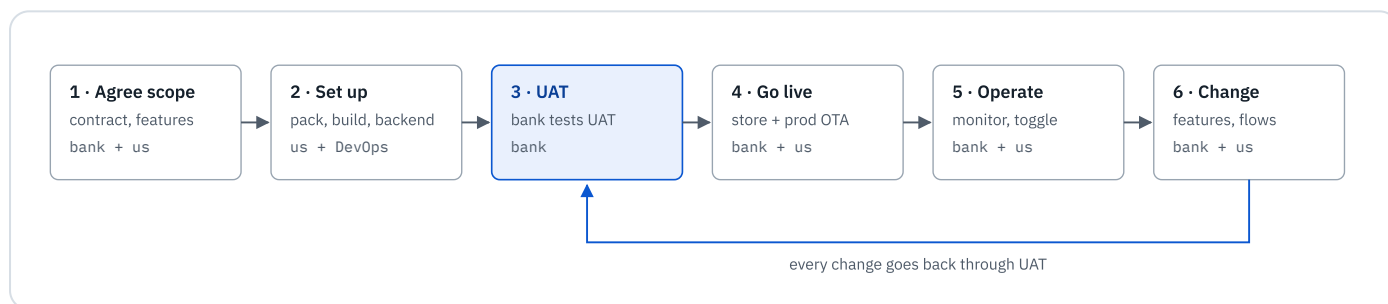


The bank is fixed at build time, so Bank B's app never contains Bank A's logo, config or server address. If a wrong OTA bundle ever lands on a phone, the guard refuses it before the first screen and the OTA client rolls back. The app also checks the org id in the first config reply, so a pack pointing at the wrong bank's backend stops at once.

The pack is a typed file. The build fails if a field is missing.

```
// tenants/bank-b/index.ts
export default {
  id: 'bank-b',
  orgId: '...', // from onboarding
  displayName: 'Bank B Mobile',
  api: { // Bank B's own backend, per environment
    uat: 'https://uat-api.bank-b.example',
    prod: 'https://api.bank-b.example',
  },
  pins: { uat: ['sha256/...'], prod: ['sha256/...'] }, // TLS certificate pins
  features: { loans: false, cards: true },
  settings: { payments: { rtgsMinimum: 200000 } },
  flows: { transferAuthorisation: 'server-driven' },
  brand,
  logo: require('./assets/logo.png'),
} satisfies TenantConfig;
```

## Tenant lifecycle



A bank moves through the same six stages. After go-live, every change, large or small, goes back through that bank's UAT.

## Onboarding a bank

### 01 Copy

```
tenants/_template
```

```
tenants/bank-c  
c
```

tenants/bank-c identity, brand colours, logo, features, settings, backend addresses and certificate pins.

### 02 Run the contrast check with Bank C's colours and adjust until every text passes.

### 03 Add Bank C to

```
tenants/_template
```

tenants/bank-c and CI jobs appear automatically.

### 04 Add launcher icons and the iOS configuration for Bank C.

### 05 DevOps stores Bank C's signing key in CI secrets.

### 06 DevOps creates Bank C's UAT and production backends from the current backend release with the infrastructure scripts: database, key pair, TLS certificate, core-banking connection.

### 07 Ops configures both backends: feature switches, customer rules, security policy, admin users.

08 The bank creates its store listing and gives our team access.

---

09 Build to

bank-; the bank tests and signs off; release to production.

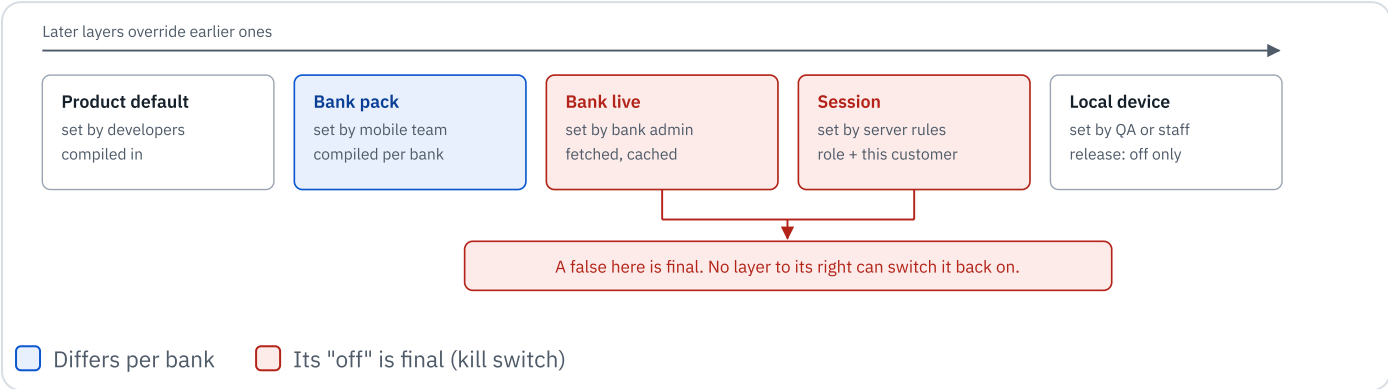
c-  
uat

---

**Offboarding:** switch the bank's features off live, unpublish its store listing with the bank, retire its two OTA channels, hand over or archive its data as the contract requires, shut down its backends, and remove the pack from the registry.

## 05 Feature flow

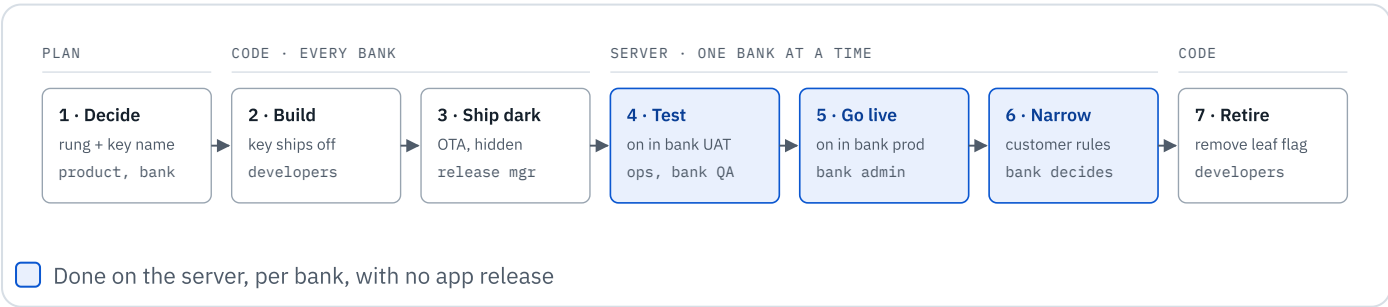
Every screen, tab, button and action has a feature key such as `loans.apply`. Keys form a tree: `loans` off hides everything under it. The app resolves each key through five layers, and a later layer overrides an earlier one, with one exception that protects the bank.



Two more rules: a feature is on only if every parent is on, and an unknown key is always off. The last known bank config is cached, so a kill switch still holds when the phone starts offline.

| FEATURE                                 | BANK A · CUSTOMER X (KYC PENDING) | BANK A · CUSTOMER Y           | BANK B · ANYONE                     |
|-----------------------------------------|-----------------------------------|-------------------------------|-------------------------------------|
| <code>loans</code>                      | <span>on</span> bank pack         | <span>on</span> bank pack     | <span>off</span> bank pack          |
| <code>loans.apply</code>                | <span>off</span> session rule     | <span>on</span>               | <span>off</span> parent is off      |
| <code>cards</code> (Bank A outage)      | <span>off</span> bank live        | <span>off</span> bank live    | <span>off</span> default            |
| Staff phone turns <code>cards</code> on | <span>off</span> still killed     | <span>off</span> still killed | <span>off</span> release ignores it |

### A feature's life



Steps 4 to 6 repeat for each bank that wants the feature, on that bank's schedule. Module roots (such as `loans`) and money actions keep their switch for ever as kill switches; leaf flags are removed once fully on.

- **The root key decides a module.** Children default to on and are switched off only to remove one capability.
- **New keys ship off.** A CI check fails any new key that defaults to on without a written reason, so an update never shows a new feature to a bank that did not ask for it.

- **The backend knows every key.** CI publishes a feature catalog on each merge; each backend imports it, so the admin portal lists the same keys the app understands.
- **The server checks again.** Every endpoint maps to its feature key and refuses with `FEATURE_NOT_AVAILABLE`.
- **Backend first, then switch on.** A feature that needs a newer backend stays off for a bank until that bank's backend is upgraded. Each backend reports what it supports, and its config service never switches on a feature it cannot serve.

## How to switch a feature on or off

There are five places to switch a feature. Which one to use depends on who decides and for whom. Day to day, the bank admin portal is the normal way; the others are defaults, automatic rules or testing tools.

| WHO SWITCHES IT | AFFECTS                                 | WHERE                                                         | TAKES EFFECT                   |
|-----------------|-----------------------------------------|---------------------------------------------------------------|--------------------------------|
| Developer       | Every bank: the product default         | The default feature config, in code                           | Next app update                |
| Mobile team     | One bank's starting setup               | <code>tenants/bank-b/features.ts</code>                       | Next app update for that bank  |
| Bank admin      | <b>All customers of that bank, live</b> | <b>Admin portal, with two-person approval</b>                 | <b>Minutes, no app release</b> |
| Server rules    | One role or one customer                | Rules in the bank's backend, or a manual block with an expiry | The customer's next sign-in    |
| QA or staff     | One phone                               | The feature switchboard screen in the app                     | Immediately, on that phone     |

The bank's backend sends its decisions in one config reply. Bank-wide switches arrive at app start, on return to the app and at sign-in. The customer's own switches arrive after sign-in.

```
{
  "version": 44,
  "overrides": { "loans": false, "cards": true },           // the whole bank
  "roleOverrides": { "payments.beneficiaries.add": false } // this customer, after sign-in
}
```

## How the app decides

1. An unknown key is off.
2. If the bank or the session says off, it is off. Nothing can undo it.
3. If a parent is off (for example `loans` for `loans.apply`), it is off.
4. Otherwise, the most specific switch that has a value wins.

**The switchboard (testing only).** The switchboard lists every key with its current value and the reason, such as "turned off by the bank" or "parent is off". In UAT builds, testers can switch features on and off. In production

builds it can only switch features off, it never undoes a bank's switch, and only staff the bank allows can open it.

What the customer sees when a feature turns off

- The tab or menu item disappears. There is no greyed-out button and no "coming soon" message.
- A customer already on that screen is taken back. During a payment, the app waits until the payment ends.
- Any remaining action is blocked in the app, and the server refuses it with `FEATURE_NOT_AVAILABLE` .

Making a new feature switchable

```
// 1. Add a key; new keys always ship off
LOANS_APPLY = 'loans.apply'
[FeatureKey.LOANS_APPLY]: { enabled: false }

// 2. Gate the screen: the route is not registered while the feature is off
ApplyLoan: FeatureKey.LOANS_APPLY

// 3. Gate the button
<FeatureGate feature={FeatureKey.LOANS_APPLY}>
  <Button title="Apply for loan" onPress={apply} />
</FeatureGate>

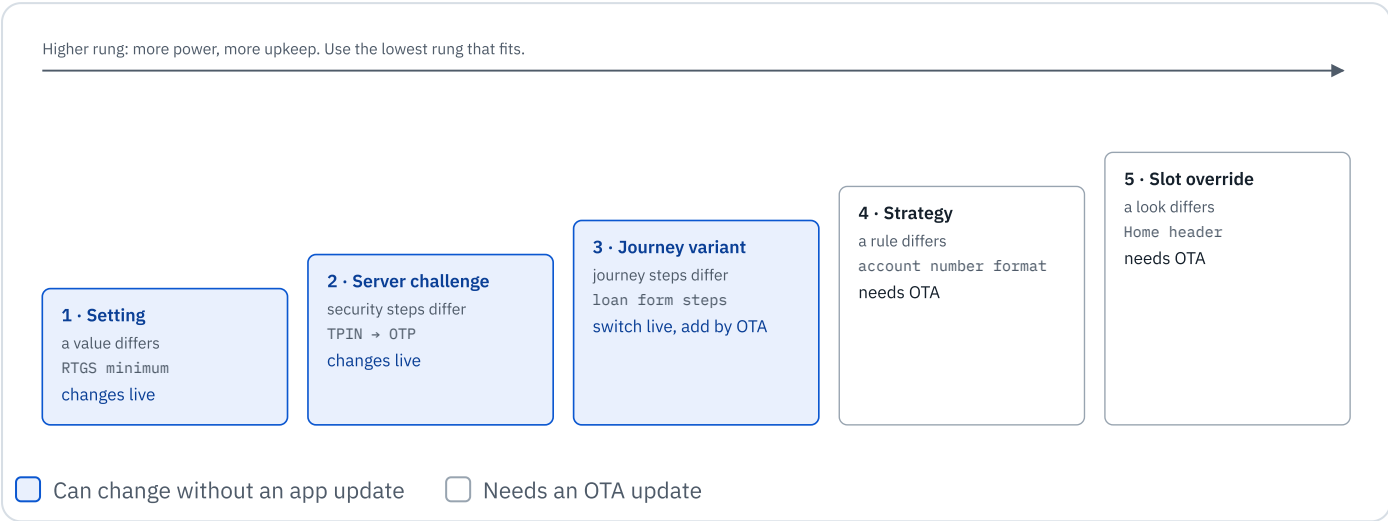
// 4. Guard the action
const apply = useFeatureAction(FeatureKey.LOANS_APPLY, submitApplication);
```

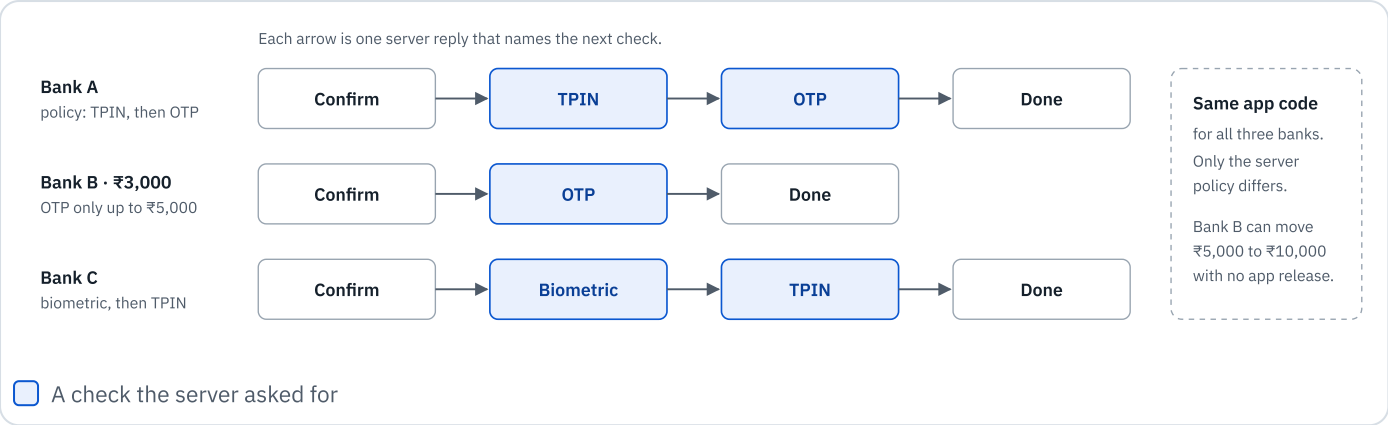
On the server, the loan-application endpoint maps to `loans.apply` and refuses the call while it is off. CI publishes the new key in the feature catalog, so it appears in every bank's admin portal.

| YOU WANT TO...                               | USE                                                                | TAKES EFFECT                            |
|----------------------------------------------|--------------------------------------------------------------------|-----------------------------------------|
| Turn loans off for Bank B during an incident | Bank B's admin portal                                              | Minutes                                 |
| Launch UPI for Bank A only                   | The key ships off in an app update; Bank A's admin turns it on     | Minutes after the update reaches phones |
| Stop one customer adding payees for a day    | A manual block with a 24-hour expiry                               | That customer's next sign-in            |
| Test cards on a QA phone                     | The switchboard on a UAT build                                     | Immediately                             |
| Remove a feature from one bank for good      | The bank's pack, with the bank's switch off until the update lands | Next app update                         |

## 06 Per-bank flows

When a bank wants a feature to work differently, we pick the lowest rung that fits. Each step up adds power and upkeep.



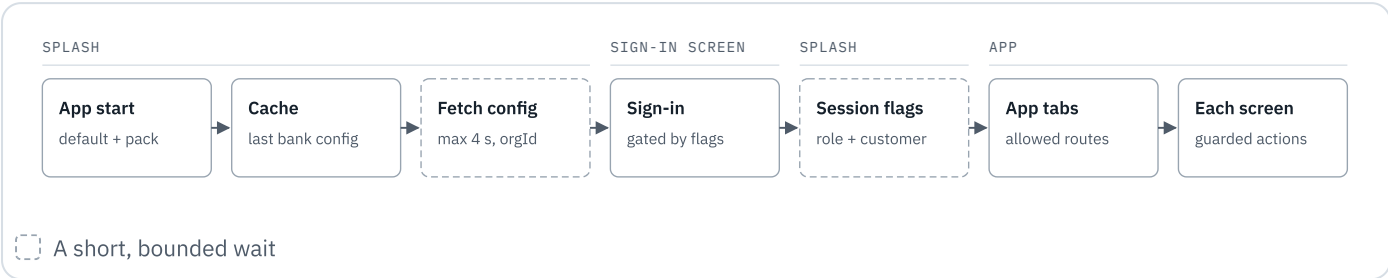


A biometric step is verified by the server: the app signs a one-time code with a key that only the customer's fingerprint or face unlocks. Older APIs that take a PIN and an OTP in fixed steps are wrapped by a small adapter, so the app works the same way everywhere.

**Not allowed:** `if (tenant.id === 'bank-b')` or `switch (tenant.orgId)` in shared code, a Git branch per bank, or copied screens. An automatic lint check fails the build on the first two.

## 07 How the app renders

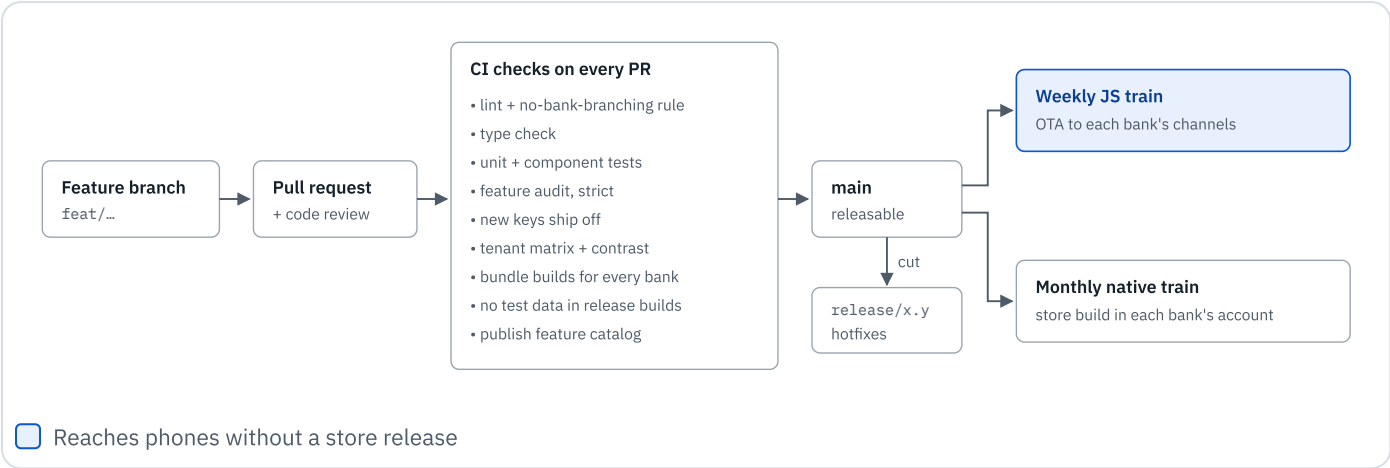
The app never shows anything before it knows what this bank and this customer may use, and it never shows something the server would refuse.



The cache makes the first step instant and keeps kill switches working offline. The two waits are short and bounded, so a weak network never leaves the customer on the splash screen.

| RULE                                               | HOW                                                                                                   |
|----------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| Nothing shows before the app knows what is allowed | Cached bank config first; the config fetch is capped at a few seconds                                 |
| No flash after sign-in                             | The app waits for the customer's flags, or receives them in the login response                        |
| A disabled feature leaves no trace                 | Its routes are not registered, its UI is not drawn and its actions are guarded; no greyed-out buttons |
| Money flows are never interrupted                  | A running payment holds a lease; a config change takes effect after it ends                           |
| Failures are contained                             | An error boundary at the root and on each tab; failures are reported with bank tags                   |
| Screens hold no bank rules                         | Flows, settings and strategies supply them, so a screen is the same for every bank                    |
| Branding comes from tokens                         | Colours, logo and fonts come from the pack through the theme and shared text components               |
| Re-render only on real change                      | A config refresh that changes nothing changes nothing on screen                                       |
| Test data never reaches customers                  | Mock data and test codes exist only in dev builds; CI checks every release build                      |

## 08 Development flow



Nothing merges unless every check passes. Fixes for an older native version are cherry-picked onto its `release/x.y` branch and ship on the same OTA train.

| BRANCH OR TAG               | PURPOSE                                                         | LIFETIME                 |
|-----------------------------|-----------------------------------------------------------------|--------------------------|
| <code>main</code>           | Always releasable; the weekly JS train is cut from here         | Permanent                |
| <code>feat/...</code>       | One change, merged by pull request                              | Days                     |
| <code>release/1.6</code>    | Created at each store release; hotfixes for that native version | Until 1.6 leaves support |
| <code>v1.6.0</code>         | Tag on every store release                                      | Permanent                |
| <code>ota-2026-11-05</code> | Tag on every JS train                                           | Permanent                |

Working locally, one bank at a time:

```
npm run start:bank-a           # Metro for Bank A, dev environment
npm run android:bank-a         # installs bankADebug on a device
npm run start:bank-b -- --reset-cache # switch to Bank B
```

### Rules for developers

- Never branch on the bank. Use a setting, the server policy, a journey variant, a strategy or a slot.
- Every new screen, tab, button and money action has a feature key, and the key ships off.
- Values a bank might change are settings, not constants.
- Security steps go through the flow framework, never a hand-written step machine.
- Test data lives in the dev environment only.
- Bank-specific code lives in that bank's pack, with a note in the pack's README saying why.

**A feature is done when** its key ships off; its route, UI and actions are gated; the server endpoint maps to the key; tests cover the flow; every CI check passes; the catalog is published; and the bank-facing notes are

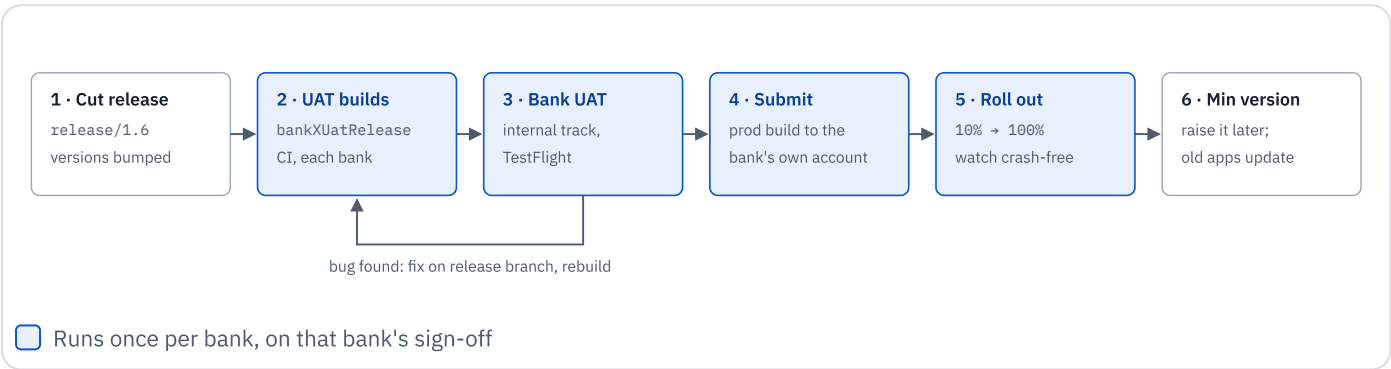
written.

## 09 Deployment flow

### App builds per bank

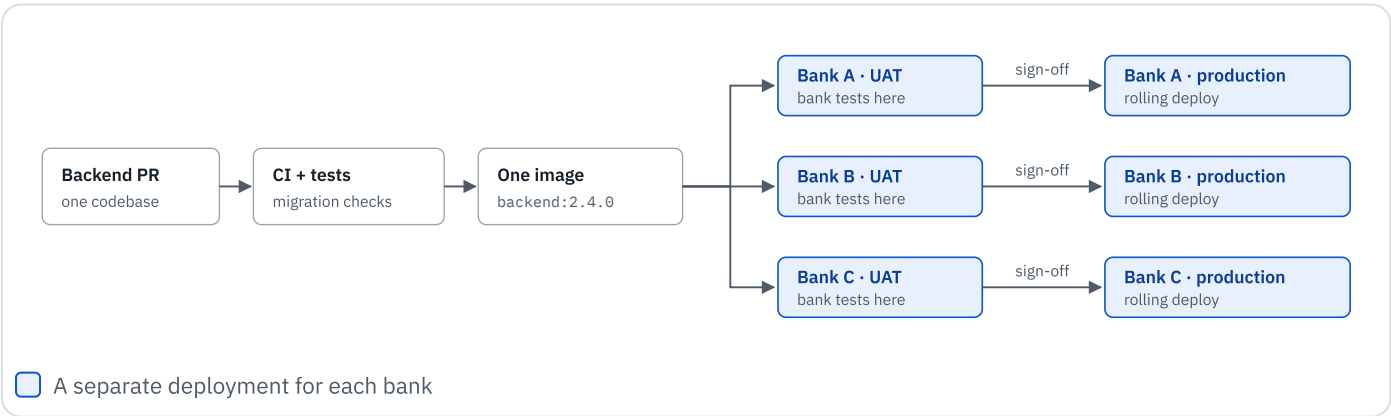
| BUILD VARIANT    | ENVIRONMENT | SIGNED WITH         | INSTALLED FROM             | OTA CHANNEL |
|------------------|-------------|---------------------|----------------------------|-------------|
| bankADevDebug    | Dev         | Debug key           | Developer machine          | none        |
| bankAUatRelease  | UAT         | Bank A's upload key | Internal track, TestFlight | bank-a-uat  |
| bankAProdRelease | Prod        | Bank A's upload key | Play Store, App Store      | bank-a-prod |

### Native release train



Each bank moves through steps 2 to 5 on its own schedule. CI uploads to each bank's store account with credentials the bank grants.

### Backend deployment, one bank at a time



Every bank runs the same image; only configuration differs. Each bank moves to production on its own sign-off, and at most two backend versions run in production at once.

- **One image everywhere.** Banks differ only in configuration and secrets: core-banking endpoint, database, key pair, certificates and org id.
- **Infrastructure as code.** One script creates a new bank's UAT and production backends. The target is one day per bank.
- **Expand, then contract.** Database and API changes are added first and removed later, so the previous backend version and older apps keep working during a rollout.

- **Each backend reports itself.** `GET /meta` returns its version and capabilities, and the release registry records which version each bank runs.
- **Each backend serves one bank.** It accepts only its own org id and refuses anything else.

## Versions

- Each bank's app version lives in `tenants/registry.json` as `MAJOR.MINOR.PATCH`. A native release bumps MINOR; a native hotfix bumps PATCH.
- The store version code is computed:  $MAJOR \times 10000 + MINOR \times 100 + PATCH$ , so 1.6.2 becomes 10602.
- OTA bundles target exact native versions ( `-t 1.6.2` ), never wide ranges.
- We support the last two minor app versions; below that, the server's minimum version sends the customer to the store.

## Signing and secrets

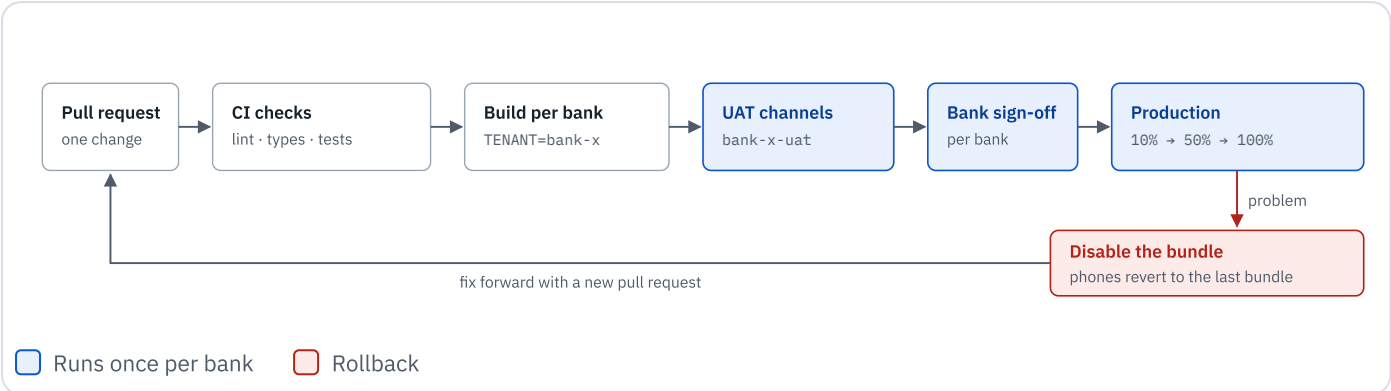
- One upload key per bank, stored only in CI secrets and backed up offline.
- Google Play App Signing per bank app; iOS certificates in each bank's Apple account.
- OTA credentials per environment; only CI may deploy to production channels.

## Keeping app and backend compatible

- A bank's backend is upgraded before any app feature that needs it is switched on for that bank.
- API changes are additive. A field is removed only after every bank's backend and every supported app version have moved past it.
- The config response carries a schema version, and the app ignores keys it does not know.

10 OTA flow

OTA updates replace the app's JavaScript and images without a store release. Every bank has two channels: `bank-x-uat` , installed by its testers, and `bank-x-prod` , installed by its customers. The channel is baked into the app build, so a bundle for Bank B can never reach Bank A.



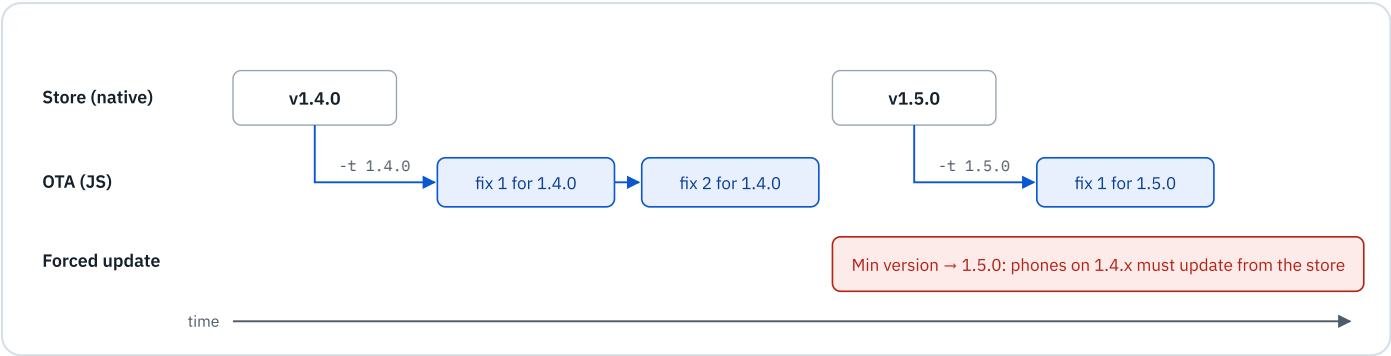
If a bundle crashes before the first screen, the OTA client rolls that phone back by itself. Later crashes are caught by error boundaries and reported, and the release manager disables the bundle.

The weekly JS train

| DAY               | STEP                                                                                                                                              | WHO                 |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| Monday            | Cut from <code>main</code> ; CI builds a bundle per bank and per supported native version, and deploys to every <code>-uat</code> channel at 100% | CI, release manager |
| Tuesday–Wednesday | Each bank tests on its UAT app                                                                                                                    | Bank QA, our QA     |
| Wednesday         | Sign-off per bank, recorded in the release ticket                                                                                                 | Bank                |
| Thursday          | Production at 10%, only for banks that signed off                                                                                                 | Release manager     |
| Friday            | 50%, if crash-free sessions stay at or above 99.5% and no error rate jumps                                                                        | Release manager     |
| Next Monday       | 100%                                                                                                                                              | Release manager     |

- **Hotfix:** the same path compressed into one day: a UAT smoke test within two hours, then 10% and 100% the same day.
- **Rollback levels:** automatic (crash before the first screen); disable one channel's bundle; disable a release on every channel with one script; then fix forward.
- **A bank that has not signed off** stays on its previous bundle. It may lag at most two trains before the release manager escalates.

# Keeping versions compatible

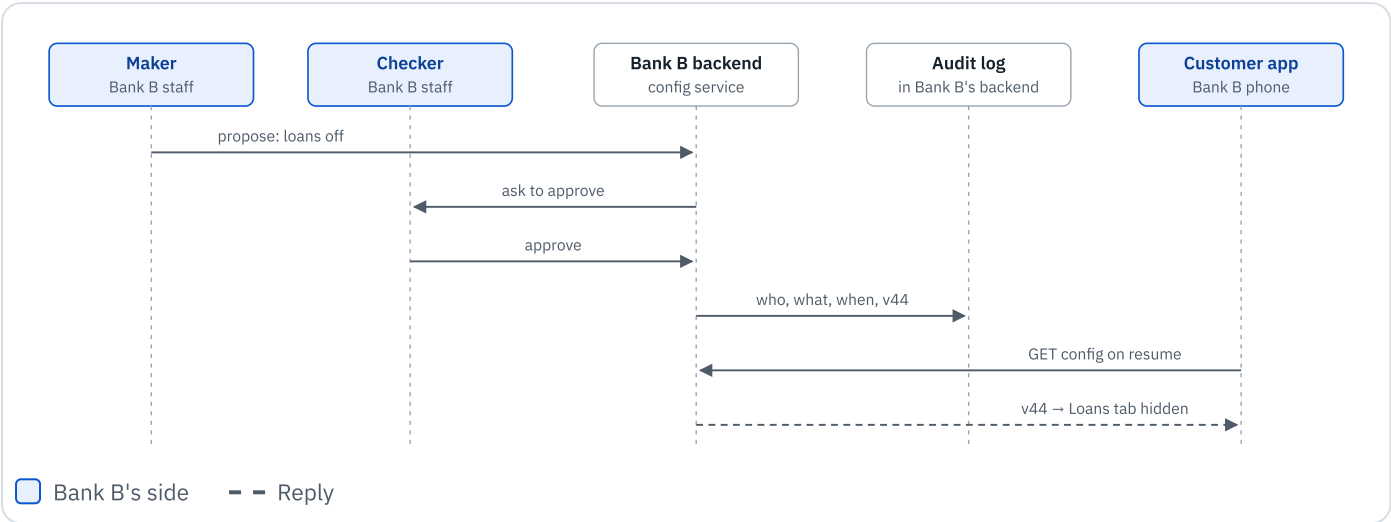


An OTA bundle reaches only the native version it was built for, so JavaScript never runs on a binary that lacks its native code.

**OTA goes live only with its safety net:** error boundaries, crash reporting tagged by bank and bundle, signed builds, `x.y.z` versions, signed bundles and the tenant guard. Without them, a staged rollout cannot see what it is doing.

# 11 Operations flow

## Changing a bank's switch



A change takes effect within minutes and can be undone the same way. Our ops team can apply an emergency kill to any bank without a checker; that action is audited and the bank is told.

## Who owns what

| TASK                                           | PLATFORM TEAM (US)       | BANK                                                | DEVOPS                                 |
|------------------------------------------------|--------------------------|-----------------------------------------------------|----------------------------------------|
| Feature scope                                  | advises                  | <b>decides</b>                                      | —                                      |
| Tenant pack, branding                          | <b>builds</b>            | approves                                            | —                                      |
| Store listing and developer account            | helps                    | <b>owns</b>                                         | —                                      |
| Signing keys, CI secrets                       | —                        | grants account access                               | <b>owns</b>                            |
| The bank's backend environment                 | builds and deploys to it | <b>owns it</b> if it runs in the bank's data centre | <b>runs it</b> if it runs in our cloud |
| Backend upgrades                               | <b>builds</b> , tests    | signs off                                           | deploys                                |
| Live feature switches                          | emergency kill only      | <b>owns</b>                                         | —                                      |
| Customer rules (KYC, account type, new device) | configures               | <b>decides</b>                                      | —                                      |
| Security policy                                | configures, reviews      | <b>decides</b> , signs off                          | —                                      |
| UAT testing                                    | supports                 | <b>owns</b>                                         | —                                      |
| OTA releases                                   | <b>owns</b>              | signs off                                           | runs the pipeline                      |
| Store releases                                 | builds                   | <b>approves</b> in its account                      | uploads via CI                         |

| TASK       | PLATFORM TEAM (US)     | BANK                   | DEVOPS                     |
|------------|------------------------|------------------------|----------------------------|
| Monitoring | <b>owns</b> app health | sees its own dashboard | <b>owns</b> infrastructure |

## Who can change what

| TOOL                            | USED BY                                                                  | CAN CHANGE                                                                 | CANNOT CHANGE                                                 |
|---------------------------------|--------------------------------------------------------------------------|----------------------------------------------------------------------------|---------------------------------------------------------------|
| Bank admin portal               | Bank staff, two-person approval; runs inside the bank's own backend      | Its own bank's switches, settings, journey variants, customer blocks       | Other banks (separate deployments), security policy, app code |
| Platform ops console            | Our operations team, over a private, audited link to each bank's backend | Onboarding, security policies, emergency kill for any bank, support lookup | App code                                                      |
| Git and CI                      | Our developers                                                           | Shared code, tenant packs, flow adapters                                   | Live switches                                                 |
| OTA console                     | Release managers                                                         | Rollout percentage, disable or roll back a bundle                          | Native code                                                   |
| Play Console, App Store Connect | Bank, with our help                                                      | Store listing, staged rollout, halt                                        | —                                                             |

## Watching each bank

Every crash report, API log and event carries five tags: `tenant` , `appVersion` , `bundleId` , `configVersion` and `channel` . Each bank's backend sends its metrics and logs, without customer data, to central monitoring. Thresholds below are starting points.

| SIGNAL                                   | ALERT WHEN                       | ACTION                                                     |
|------------------------------------------|----------------------------------|------------------------------------------------------------|
| Crash-free sessions, per bank and bundle | Below 99.5% after a rollout step | Pause the rollout; disable the bundle if it keeps falling  |
| <code>FEATURE_NOT_AVAILABLE</code> rate  | Rises after a release            | The app shows something the server refuses; fix the gating |
| OTP and challenge failures, by type      | Jump at one bank                 | Check that bank's SMS gateway or policy                    |
| Config fetch failures                    | Above 5%                         | Check that bank's backend and network                      |
| Phones below the minimum version         | After a forced update            | Bank sends a customer message                              |

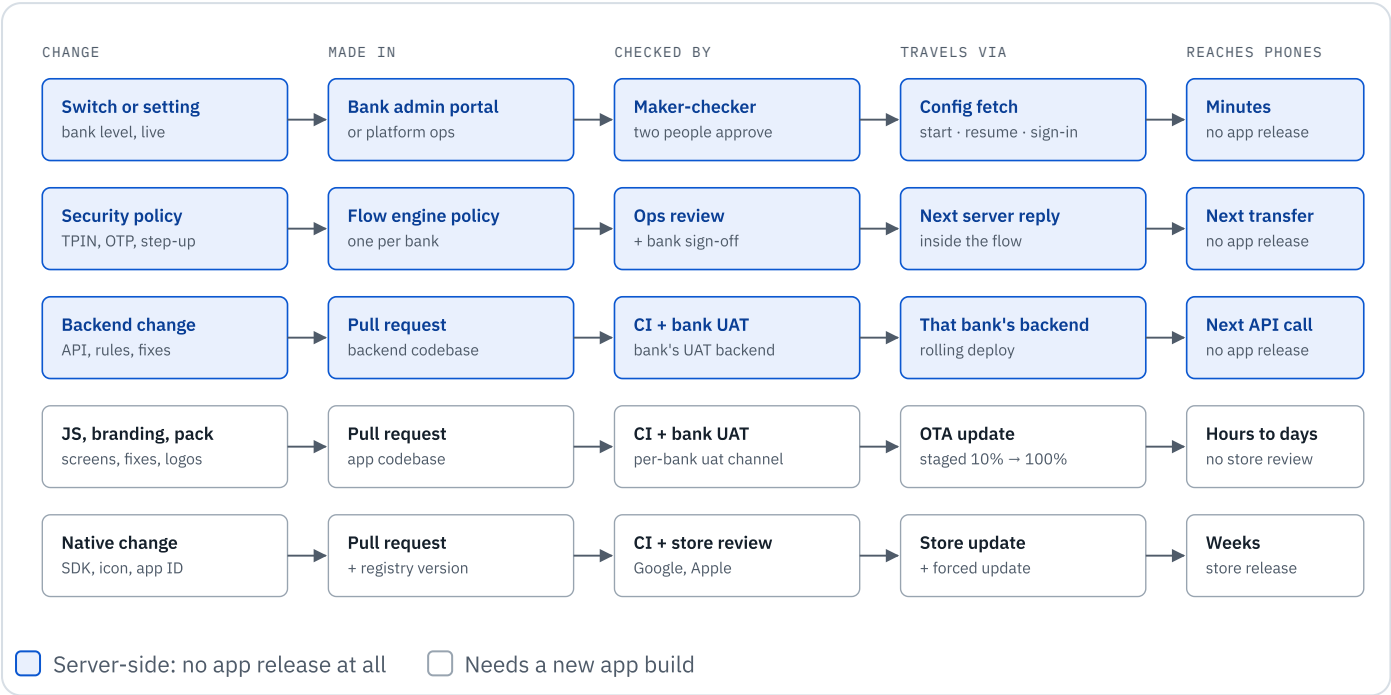
**Support lookup:** in the ops console, a support agent enters a customer's mobile number and sees the bank, app version, bundle, config version and every feature with the layer that decided it. The same resolution logic runs in the app and on the server, so both always agree.

## When something goes wrong

| PROBLEM                                                 | FASTEST FIX                                                                       | TAKES EFFECT  | WHO                   |
|---------------------------------------------------------|-----------------------------------------------------------------------------------|---------------|-----------------------|
| A feature misbehaves at one bank                        | Switch it off in the admin portal                                                 | Minutes       | Bank admin or ops     |
| A new JS bundle crashes on start                        | Nothing: the OTA client rolls that phone back                                     | Next launch   | Automatic             |
| A new JS bundle has a bug                               | Disable the bundle in the OTA console                                             | Minutes       | Release manager       |
| A security step must change now                         | Change that bank's security policy                                                | Next transfer | Ops, with the bank    |
| One bank's backend is down, or a backend release is bad | Roll that bank's backend back to the previous image; other banks are not affected | Minutes       | DevOps, with the bank |
| A native bug in an old version                          | Ship a fixed store build, then raise the minimum version                          | Hours to days | Platform team, bank   |
| A bad store build is rolling out                        | Halt the staged rollout in the store console                                      | Hours         | Bank's store account  |

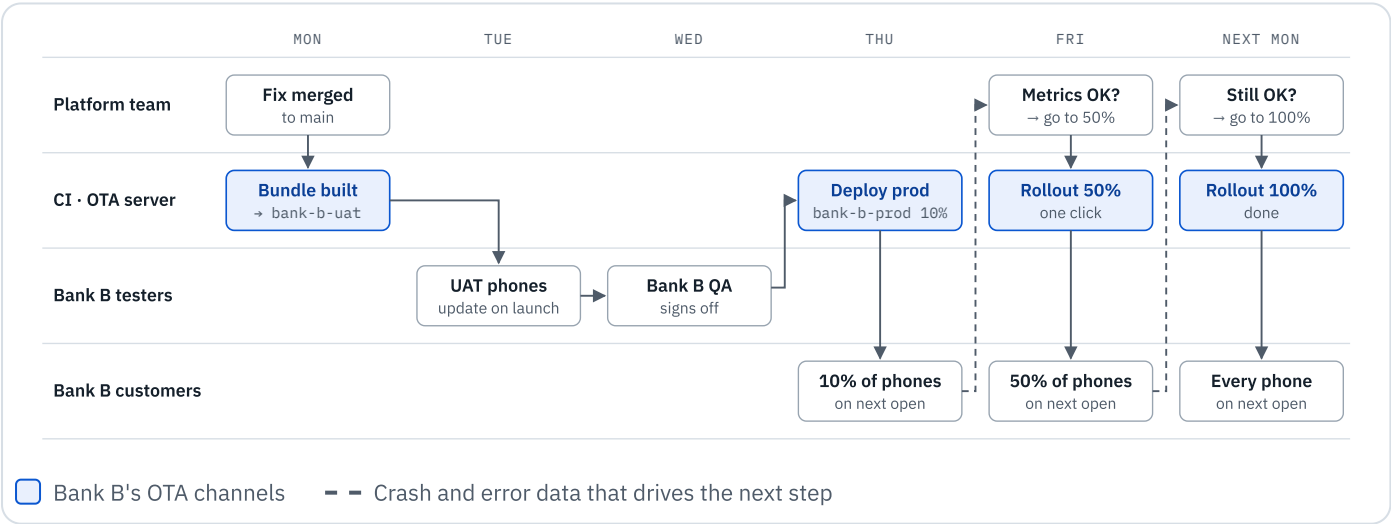
## 12 One bank's update

Every change takes one of five lanes. The lane depends on what changed, not on urgency.



The three blue lanes change only the bank's backend, so no app release is needed. Every lane that ships code, on the server or on the phone, passes through that bank's UAT first.

### A bug fix for Bank B, day by day



Bank A and Bank C run the same steps on their own channels and their own sign-off. If Bank B finds a problem at any point, the release manager disables the bundle and Bank B's phones return to the previous one.

## Every kind of change, for Bank B

| BANK B WANTS...                        | LANE                      | STEPS                                                                                                                         | REACHES CUSTOMERS                                 | WHO ACTS                           |
|----------------------------------------|---------------------------|-------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------|------------------------------------|
| Loans off today, during an incident    | switch                    | Admin portal → second approver → config v+1 → phones refresh on resume                                                        | Minutes                                           | Bank B admin                       |
| A lower RTGS minimum                   | setting                   | Same as a switch                                                                                                              | Minutes                                           | Bank B admin                       |
| OTP only up to ₹5,000                  | policy                    | Ops edits Bank B's policy on its UAT backend → Bank B tests → same policy in production                                       | Next transfer after go-live                       | Ops, Bank B                        |
| A backend fix or a new API             | backend                   | Backend pull request → new image → Bank B's UAT backend → Bank B signs off → Bank B's production backend                      | Next API call after the deploy                    | Backend team, Bank B               |
| A transfer-screen bug fixed            | OTA                       | The week shown above                                                                                                          | 3–7 days; a hotfix the same day                   | Platform team, Bank B QA           |
| New logo or colours                    | OTA                       | Change <code>tenants/bank-b/</code> → only Bank B's bundles rebuild → UAT → prod                                              | Days                                              | Platform team, Bank B              |
| A new feature (for example UPI)        | backend, OTA, then switch | Bank B's backend upgraded first → feature ships off in a JS train → on for Bank B UAT → Bank B signs off → on for Bank B prod | After the backend upgrade, the next train and UAT | Backend and platform teams, Bank B |
| A new native SDK, app icon or app name | store                     | Native train → Bank B UAT build → submit in Bank B's account → staged rollout → raise minimum version                         | 1–3 weeks                                         | Platform team, Bank B store owner  |

### What keeps Bank B's change away from other banks

- A change only under `tenants/bank-b/` : CI detects it and rebuilds only Bank B's bundles.
- A change to shared code: every bank gets it in UAT, but each bank's production moves only after its own sign-off.
- A backend change or a switch: it happens in Bank B's own backend deployment. Other banks run separate deployments, and Bank B's admins cannot reach them.
- On the phone: the channel and the tenant guard stop any other bank's bundle from running.

## 13 Security

1. **The server is the boundary.** Every business API checks the bank, the role and the customer again. App-side gating is for the customer's convenience.
2. **Each bank's data stays in its own backend:** a separate database, key pair and certificate per bank. The central platform holds no customer data.
3. **HTTPS with per-bank certificate pinning,** for config, security steps and every API. The PIN encryption key comes from the bank's backend, and pinning is what makes that key trustworthy.
4. **Fail closed.** An unknown feature key, challenge type or unreadable config means off or stop.
5. **The bank's off is final.** Session and device layers cannot re-enable it; release builds let the device layer switch features off only.
6. **Kill switches survive offline,** and the cached config only moves forward in version.
7. **No leaks between customers or banks:** session flags wiped at sign-out, a bundle per bank, the tenant guard, and token-scoped config.
8. **Test data never ships.** Mock data and test codes exist only in dev builds, and CI checks every release build.
9. **Signed code:** a store key per bank, signed OTA bundles, production deploys only from CI.
10. **Crash reports without personal data,** stored where the banks' data rules allow.
11. **Every change recorded:** two-person approval and an audit log for switches and policies.

## 14 Risks

| RISK                                            | WHY IT MATTERS                                  | HOW WE REDUCE IT                                                                                            |
|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| Banks are slow to give store access or sign off | Releases stall per bank                         | Start account setup early; put UAT timelines in the contract                                                |
| The build matrix grows                          | Banks × platforms × environments per release    | Parallel CI, rebuild only affected banks, bundle diffing                                                    |
| Banks drift onto different app versions         | JS must work on several binaries                | Two-version support window, forced update, train discipline                                                 |
| Running many backends                           | Patching, monitoring and backups for every bank | One image, infrastructure scripts, automated deploys, central dashboards, written runbooks                  |
| Bank backends drift onto different versions     | The app must work with several backend versions | Features gated by what each backend reports; at most two backend versions in production; additive APIs only |
| The OTA server is compromised                   | One door into every bank's app                  | Signed bundles, CI-only production deploys, audit of every deploy                                           |
| Too many flags                                  | Combinations become impossible to test          | Retire leaf flags; snapshot test of each bank's effective config; quarterly catalog review                  |
| Too many per-bank overrides                     | Overrides drift from the shared screens         | Lowest rung first; at most three overrides per bank; each has an owner                                      |
| Crash reports carry personal data               | Banking privacy rules                           | Scrub before sending; choose storage that meets the banks' data rules                                       |

## 15 Decisions

| QUESTION                                    | RECOMMENDATION                                                                                                                                                                                                                                | WHO DECIDES              |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| One backend for all banks, or one per bank? | <b>Decided</b> One backend codebase, deployed separately for each bank and environment                                                                                                                                                        | Decided                  |
| Where does each bank's backend run?         | In our cloud, one isolated account per bank, in India (RBI's payment-data storage rules), unless a bank requires its own data centre. The image and deploy steps are the same; in a bank's data centre, its operators deploy our signed image | DevOps, security, banks  |
| Whose store accounts publish each app?      | Each bank's own Google Play and Apple accounts. Apple's guideline 4.2.6 expects the content owner to submit template-based apps                                                                                                               | Product, banks           |
| Adopt the server-driven security protocol?  | Yes; adapters cover older APIs until every backend supports it                                                                                                                                                                                | Backend lead, security   |
| Who operates the admin portal?              | Bank staff with two-person approval; our ops for onboarding and emergencies                                                                                                                                                                   | Product, banks           |
| Release rhythm                              | Weekly JS train, monthly native train, backend releases per bank on sign-off, hotfix any day                                                                                                                                                  | Mobile and backend leads |
| Where OTA bundles are hosted                | A self-hosted OTA server in our own cloud, with signed bundles                                                                                                                                                                                | Security, DevOps         |
| Crash reporting tool                        | One that can scrub personal data and store it where the banks' rules allow                                                                                                                                                                    | Security                 |
| What a restricted customer sees             | Hide what the bank does not offer; explain what the customer can unlock ("Complete KYC to apply")                                                                                                                                             | Product                  |

The version support window (the last two app versions and at most two backend versions in production) is part of this proposal and needs the banks' agreement.

Proposal prepared October 2026. Detailed technical specifications for each section accompany this document.