anynines

Cloud Foundry migration guide

A practical guide to migrating between Cloud Foundry environments with low risk and minimal downtime. For platform, security, and engineering leaders evaluating a Cloud Foundry migration.

Table of contents

Introduction

Cloud Foundry environments are subject to change due to contract updates, provider strategies, data center consolidation, and infrastructure decisions. When migration becomes necessary, it must happen without significant downtime, since any disruption to established workflows affects development velocity and, ultimately, competitiveness.

A successful migration depends on access to four things: the source Cloud Foundry configuration, deployable application artifacts, identity settings, and a supported method for transferring service data. With these, you can transition to a new distribution, provider, or location while keeping the Cloud Foundry workflows your teams already rely on.

This guide outlines the technical steps for Cloud Foundry-to-Cloud Foundry migration, key risks to address, and the eight-step approach anynines uses to verify compatibility, migrate applications and data services in stages, and establish a sustainable Day 2 operating model.

What migration gets you:

Retention of the Cloud Foundry development model your teams already know

Reduced vendor lock-in and cost uncertainty

A more sustainable, better-supported operating model going forward

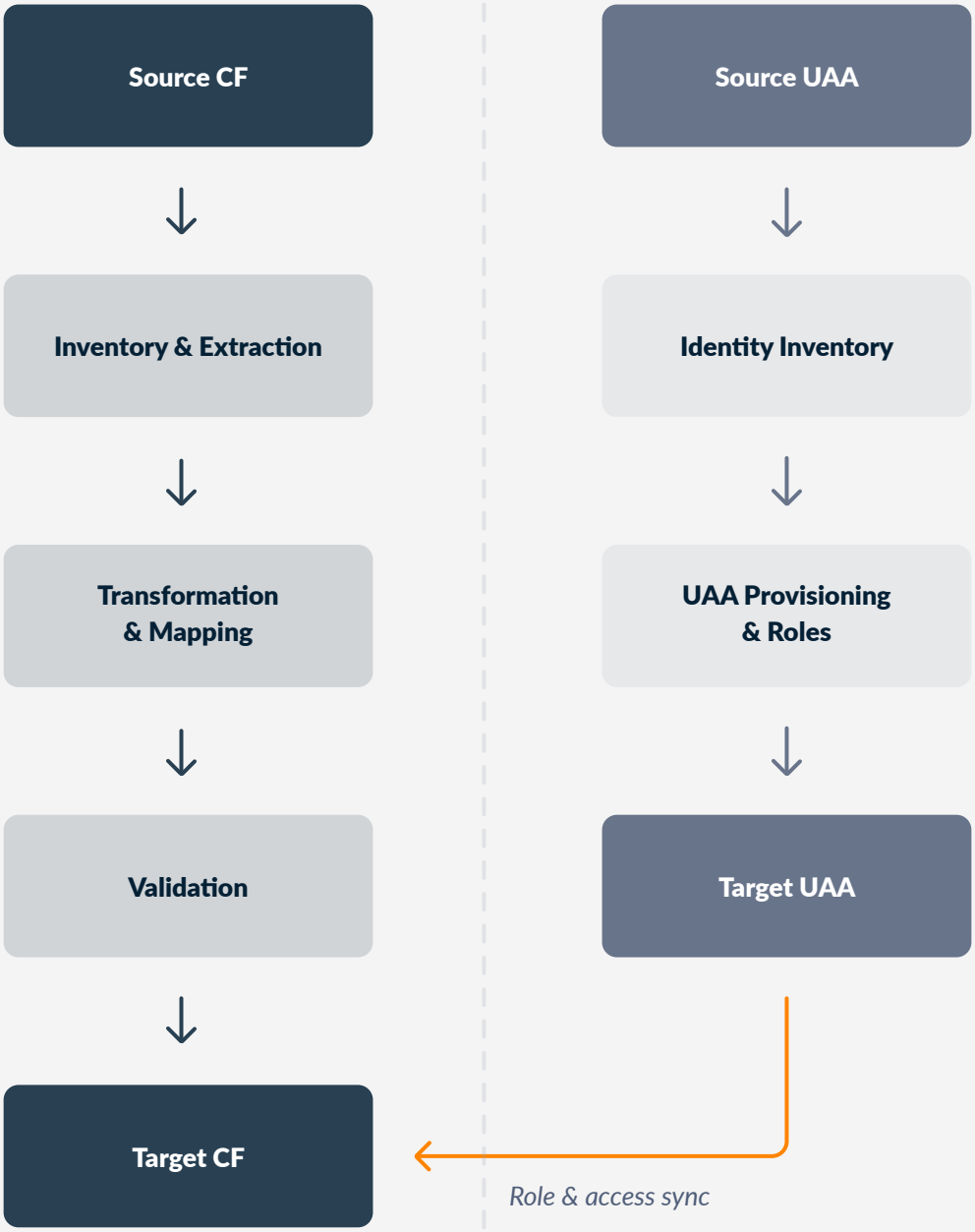
What you'll find in this guide

This guide covers:

- How to assess migration motivations, risks, and architectural suitability
- How to inventory and recreate Cloud Foundry objects and dependencies
- Considerations for identities, UAA, roles, domains, routes, and certificates
- Incremental migration waves, synchronization, cutover, and rollback
- Approaches to migrating service data based on consistency and recovery needs
- Control measures for security, performance, CI/CD, troubleshooting, and monitoring
- The typical eight-step migration path anynines follows
- How Cloud Foundry and anynines' managed Day-2 operations support the transition

> ***Note on scope:*** *This guide provides a general migration framework. Exact procedures vary depending on the Cloud Foundry distribution, version, identity configuration, service broker, and underlying services in play.*

Technical details on migration



Object migration and the identity/UAA workstream run as two separate, synchronized tracks.

The core migration pattern

Most migrations use a two-step pattern for Cloud Foundry objects. This API-driven work must be paired with a complete inventory of application artifacts, buildpacks, stacks, routes, identities, quotas, security policies, service brokers, and underlying services. Omitting the inventory often leads to unexpected issues.

1. **Inventory and extraction**:

   Retrieve organizations, spaces, apps, routes, configurations, roles, and associated metadata from the source environment.

2. **Recreation and validation**:

   Set up the corresponding objects and dependencies in the target environment, then verify their relationships and behavior.

Identity and authorization need their own workstream

User records, identity providers, UAA clients, and Cloud Controller role assignments do not follow the standard object pattern. A reliable migration treats identity and authorization as a dedicated workstream:

- Inventory identity providers, UAA clients, users, groups, and role assignments
- Configure the required identity providers and UAA clients in the target environment
- Create or synchronize the required users in accordance with the target identity model
- Restore organization and space roles, then verify SSO, client redirects, login credentials, and access rights

> ***Potential pitfall**: Compatibility is more than just the API. Verify CF API versions, runtime environments, stacks, buildpacks, routing, identities, quotas, security groups, brokers, and service plans. Convert incompatible objects or workflows before recreating them in the target environment.*

Resource mapping

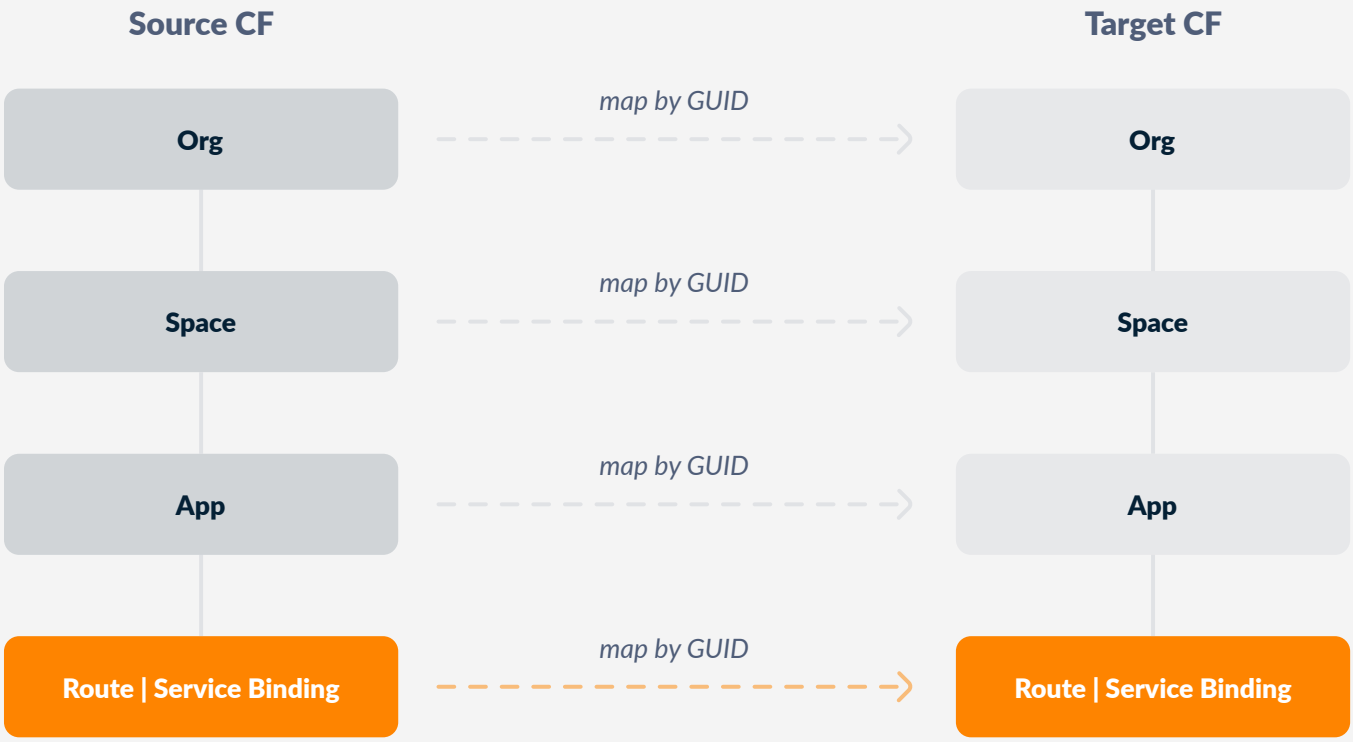
Migrating objects without a controlled mapping strategy can cause duplicates and mis-mappings. Map each object's identity and relationships between source and target before recreation, and validate mappings during every migration run.

The resource mapping process:

- Identify objects in the source environment by name, scope, type, and current GUIDs.
- Search for the corresponding objects in the destination environment, or create them.
- Record and validate the relationships between source and destination objects.

Example of mapping an app configuration:

- Retrieve the source scope and organization for the app.
- Identify the matching organization in the destination environment based on its verified name and scope.
- Locate the target scope within that organization, or create it.
- Locate the app by its name and target scope, or create it.
- Verify routes, processes, configurations, dependencies, and service bindings.



***Potential risk**—Outdated mapping status store: Mappings in a controlled migration state store and re-verify them with every run. Never reuse a source-to-target GUID mapping without first confirming that both the objects and their relationships are still current.*

Every level—org, space, app, route, and service binding is mapped by GUID between source and target.

Changes to domains, routes, and certificates

Source and target environments usually operate in separate systems and application domains. While both are active, use temporary domains for the target until the final switchover, while the source environment continues serving production traffic.

Before switchover:

- Create and validate the target system's system and application domains, routes, and required DNS entries.
- Provision valid certificates for the new domains and verify the entire chain of trust.
- Update dependent configurations, verify routing and TLS, then switch the routes or DNS records using a tested rollback path.

> ***Potential risk***—*Default shared domain: If an application is deployed without an explicit domain, Cloud Foundry defaults to the first shared domain created during deployment. Create shared domains intentionally and specify routes explicitly in migration manifests to avoid landing on the wrong one.*

Incremental migration and switchover

For large environments, migrating in pre-tested waves is safer than a single cutover. Repeated synchronizations limit the scope of each change and reveal compatibility issues long before they can affect production:

1. **Initial migration or test run**:

   Move the majority of objects and data well ahead of switchover, then validate the target environment.

2. **Synchronization before switchover**:

   Capture the platform and service-data changes made since the first run.

3. **Final synchronization and switchover**:

   Lock changes to the platform configuration and, if necessary, application write operations; synchronize any remaining changes, validate the target environment, then switch the routes or DNS.

Service instances and data migration

Service migration depends on the specific service: its persistence model, supported interfaces, RPO and RTO targets, and the amount of downtime or parallel operation the application can tolerate.

What needs to be evaluated:

- Persistent data stores such as PostgreSQL or MariaDB.
- Messaging, key-value, cache, configuration, and secret services, including RabbitMQ and Redis-compatible services
- External, SaaS, user-provided, and brokered services and bindings

Supported data transfer options:

- Backup and restore operations supported by the broker, export/import, or service-specific migration procedures
- Replication, continuous synchronization, or a final delta synchronization, where supported
- Controlled transfer of service keys, user-provided service configurations, and login credentials

Data consistency and downtime:

- A database dump can provide a consistent online snapshot, but write operations after that snapshot won't be included
- RPO and RTO targets determine whether backup and restore, replication, delta synchronization, or another method is the right fit
- A brief write lock or application restart may still be required for the final switchover

Concurrently running applications:

- Not every application supports active-active operation across source and target environments
- Before running both versions in parallel, check for shared state, schedulers, singleton jobs, message receivers, and external dependencies

Migration approaches by scenario

1. Broker-based migration:

Where available, use the documented procedures for the service broker and its underlying services.

2. Step-by-step connection switchover:

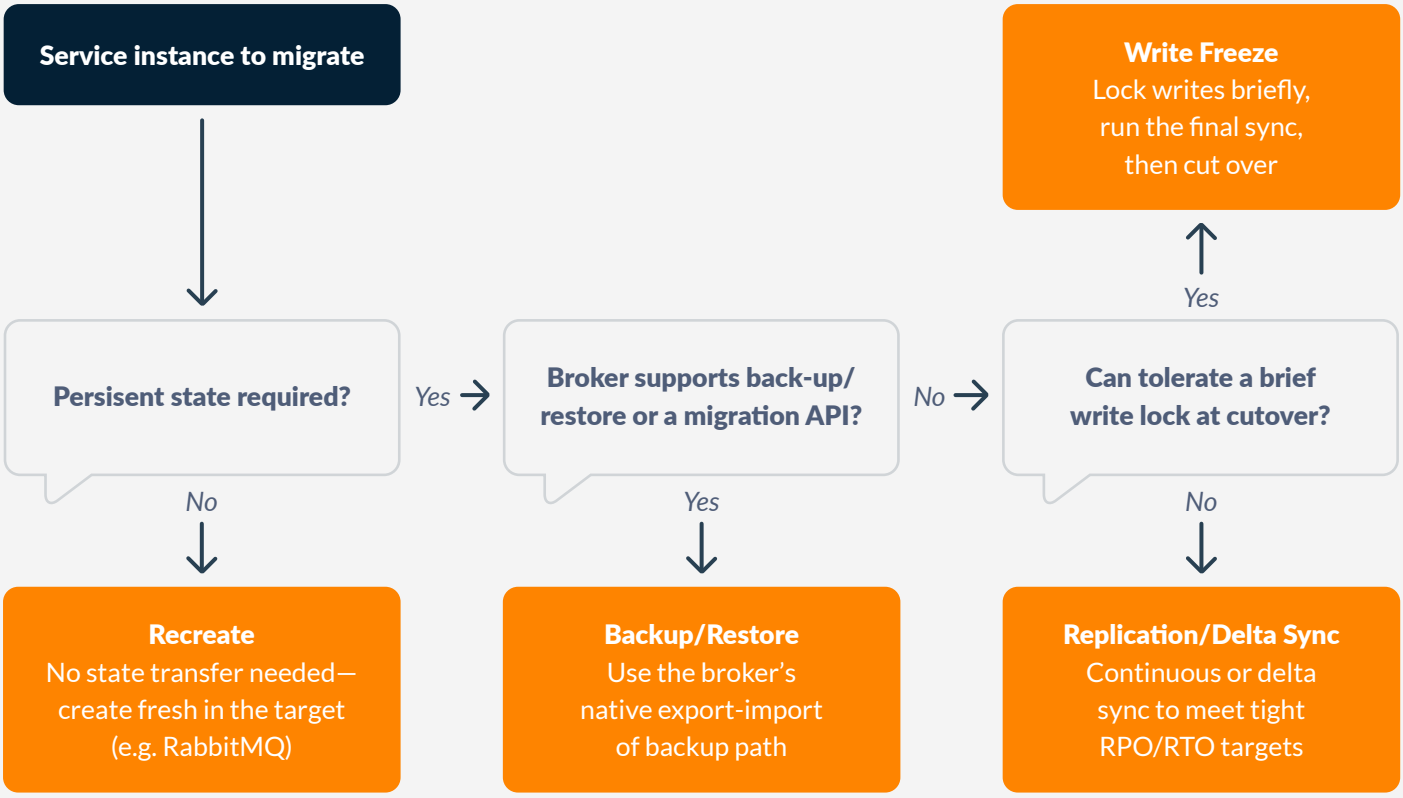
Where control mechanisms allow it, connect source applications to a target service via a user-provided service during validation.

3. Controlled final synchronization:

Use replication, delta synchronization, or a temporary write lock to meet consistency requirements.

4. Recreation without state transfer:

Use only after confirming no required persistent state exists. For RabbitMQ, evaluate queues, messages, definitions, policies, users, and persistence settings first.



A simplified decision path for choosing a service data migration approach.

Additional migration checks

Security and compliance

- Restore and validate user roles, permissions, identity policies, and audit trails in the target system
- Rotate credentials and secrets, verify encryption and trust chains, and protect exported migration data
- Before migrating production, review requirements around data location, retention periods, logging, and legal regulations

Performance and capacity testing

- Load-test representative applications and services in the target environment before migration
- Compare performance, capacity, quotas, scaling behavior, and error handling against your agreed acceptance criteria

CI/CD and automation adjustments

- Update pipelines, targets, credentials, manifests, and deployment automation for the new environment
- Verify staging, deployment, scaling, route mapping, service binding, and rollback through automation

Troubleshooting common issues

- **Application errors**: check logs, buildpacks, stacks, processes, environment configuration, and dependencies
- **Authentication errors**: check identity providers, UAA clients, redirects, users, and target role assignments
- **Network issues**: check DNS, routes, TLS, load balancers, security groups, and firewall rules
- **Service binding errors**: check the target service, credentials, and application binding

Validation, rollback, and monitoring after migration

- Validate functionality, security, performance, backup and recovery, and operational acceptance criteria
- Keep the rollback path available until production behavior and data integrity are confirmed
- Monitor operational status, trigger alerts on performance degradation, and document lessons learned for the next migration wave

The typical anynines migration path

The technical workstreams outlined above form the basis of any Cloud Foundry migration. anynines integrates these into a structured approach, beginning with your current environment, validating compatibility through a proof of concept, and migrating applications and dependencies to the target platform before establishing standardized Day 2 operations.

> *anynines approaches migration as a step-by-step validation and transition of your operating model, rather than a one-time switch. The process is tailored to your environment, service portfolio, risk tolerance, and preferred level of managed operations.*

- Assessment
- Architecture review
- Compatibility review
- Proof of concept
- Data services
- Phased migration
- Day-2 operations
- Team expansion

Assessment of the current state and migration risks

Identify dependencies across Cloud Foundry, Tanzu, infrastructure, applications, identities, and services. Determine costs, vendor lock-in, support, and operational risks, and define success criteria, responsibilities, RTO, and RPO.

Architectural suitability review

Evaluate a9s Cloud Foundry as a Cloud Foundry-compatible, infrastructure-agnostic platform for public cloud, private cloud, hybrid, and on-premises environments, and select the appropriate fully or partially managed operating model.

Compatibility review

Verify runtime and API compatibility to preserve the developer model your teams already use — cf push, routing, staging, scaling, and service binding — along with buildpacks, stacks, security, identity management, CI/CD, and service plans.

Demonstrate the approach using an initial application

Deploy a representative application and integrate a service as a proof of concept. Test performance, logging, operations, and rollback to confirm developers can keep working within the familiar Cloud Foundry model.

Integrate data services

Identify application data dependencies and plan deployment, backup and recovery, lifecycle management, service bindings, and credential rotation. Integrate a9s Data Services where they fit the target platform's model.

Phased migration

Migrate applications in workload groups or migration waves rather than a single big-bang cutover. Test each wave, synchronize changes, validate the target environment, and keep a tested rollback path until acceptance is complete.

Take over operations and standardize

Establish managed Day-2 operations for upgrades, security patches, monitoring, quality assurance, and release management. Standardize runbooks, responsibilities, and controls to reduce the operational burden on your internal teams.

Expand to additional teams

Turn the insights from the proof of concept and migration waves into a repeatable deployment pattern. Standardize automation, onboarding, governance, and support so the workflow scales to additional application teams.

How anynines supports the transition

anynines has operated cloud platforms and worked with Cloud Foundry since 2013. We provide expert-led migration to a9s Cloud Foundry, from initial assessment and proof of concept through migration waves to managed Day-2 operations.

1. Migration planning and execution:

Architecture assessment, compatibility testing, migration wave planning, testing, cutover, and rollback support.

2. Platform and data service continuity:

Keep your established Cloud Foundry workflows while integrating a9s Data Services and, where appropriate, a phased hybrid strategy that combines Cloud Foundry and Kubernetes.

3. Managed Day-2 operations:

Release management, quality assurance, security patches, monitoring, and support that reduce ongoing operational overhead after the migration.

Conclusion

Cloud Foundry migration is achievable once architectural compatibility, identity management, service data, operational controls, and rollback options are validated before production. Migrating a representative application in controlled phases reduces risk and preserves the developer experience.

a9s Cloud Foundry is anynines' enterprise application platform, built on open-source Cloud Foundry and used by large organizations worldwide. It offers a Cloud Foundry-compatible target for migrations from other distributions or providers, and operates across public cloud, private cloud, hybrid, and on-premises environments. anynines supports the entire migration process, from platform assessment and compatibility validation to phased migration and long-term managed operations.

Discover a9s Cloud Foundry

anynines

anynines is a European provider of platform, database, and data service automation, giving organizations control over their infrastructure and data rather than locking them into a single hyperscaler. anynines has been operating cloud platforms since 2008 and working with Cloud Foundry since 2013, and is certified under ISO/IEC 27001:2022. Its a9s Cloud Foundry and a9s Data Services solutions support public cloud, private cloud, hybrid, and on-premises environments, fully or partially managed.

anynines.com