# Documentation complète des API Apple pour la publication automatique d'applications

## Vue d'ensemble des API disponibles

Apple offre un écosystème complet d'API et d'outils permettant l'automatisation totale du processus de publication d'applications sur l'App Store, depuis la compilation jusqu'à la distribution. Fastlane Cette documentation couvre l'ensemble des API disponibles en 2025, incluant les nouvelles fonctionnalités de webhooks et l'API de build upload. DEV Community +3

## 1. App Store Connect API

### Architecture et authentification

L'App Store Connect API est une API RESTful utilisant le standard JSON:API avec authentification JWT. Andy Ibanez +4 Elle constitue le cœur de l'automatisation avec plus de **200 endpoints disponibles** depuis 2025. Apple Developer

**Base URL**: https://api.appstoreconnect.apple.com/v1

#### Génération de token JWT

```python
```

```python
import jwt
import time
from datetime import datetime, timedelta

def generate_jwt_token(key_id, issuer_id, private_key_path):
    with open(private_key_path, 'rb') as fh:
        private_key = fh.read()

    payload = {
        "iss": issuer_id,
        "iat": int(time.time()),
        "exp": int(time.time()) + 1200,  # 20 minutes maximum
        "aud": "appstoreconnect-v1"
    }

    headers = {
        "alg": "ES256",
        "kid": key_id,
        "typ": "JWT"
    }

    token = jwt.encode(payload, private_key, algorithm="ES256", headers=headers)
    return token
```

**Configuration requise**:

- **Issuer ID**: Identifiant unique de votre équipe
- **Key ID**: Identifiant de la clé API
- **Private Key**: Fichier .p8 téléchargé une seule fois (fastlane) (Andy Ibanez)
- **Algorithm**: ES256 (obligatoire)
- **Expiration**: Maximum 20 minutes (Medium +3)

## Endpoints principaux pour la gestion d'applications

### Gestion des versions et builds

```bash
```

```
# Créer une nouvelle version
POST /v1/appStoreVersions
{
  "data": {
    "type": "appStoreVersions",
    "attributes": {
      "versionString": "2.0.0",
      "platform": "IOS"
    },
    "relationships": {
      "app": {"data": {"type": "apps", "id": "app-id"}}
    }
  }
}

# Soumettre pour review
POST /v1/reviewSubmissions
{
  "data": {
    "type": "reviewSubmissions",
    "relationships": {
      "appStoreVersion": {"data": {"type": "appStoreVersions", "id": "version-id"}}
    }
  }
}

# Gérer les builds
GET /v1/builds
POST /v1/buildUploads  # Nouveau en 2025
GET /v1/builds/{buildId}/betaBuildLocalizations
```

### Upload de builds (Nouvelle API 2025)

```
python
```

```python
def upload_build_workflow(app_id, ipa_path):
    # 1. Créer l'upload
    upload_response = requests.post(
        f"{BASE_URL}/buildUploads",
        headers={"Authorization": f"Bearer {token}"},
        json={
            "data": {
                "type": "buildUploads",
                "attributes": {
                    "bundleVersion": "1.0.0",
                    "platform": "IOS"
                }
            }
        }
    )
    upload_id = upload_response.json()['data']['id']

    # 2. Créer le fichier de build
    file_size = os.path.getsize(ipa_path)
    file_response = requests.post(
        f"{BASE_URL}/buildUploadFiles",
        headers={"Authorization": f"Bearer {token}"},
        json={
            "data": {
                "type": "buildUploadFiles",
                "attributes": {
                    "fileName": "app.ipa",
                    "fileSize": file_size,
                    "assetType": "IOS_APP"
                },
                "relationships": {
                    "buildUpload": {"data": {"type": "buildUploads", "id": upload_id}}
                }
            }
        }
    )

    # 3. Upload du fichier binaire
    upload_instructions = file_response.json()['data']['attributes']['uploadOperations']
    for operation in upload_instructions:
        with open(ipa_path, 'rb') as f:
            requests.put(operation['url'], data=f.read(), headers=operation['requestHeaders'])

    # 4. Marquer l'upload comme terminé
    requests.patch(
        f"{BASE_URL}/buildUploads/{upload_id}",
```

```
        headers={"Authorization": f"Bearer {token}"},
        json={
            "data": {
                "type": "buildUploads",
                "id": upload_id,
                "attributes": {"uploaded": True}
            }
        }
    )
```

## Webhooks (Nouveau 2025)

Apple a introduit un système complet de webhooks permettant une architecture événementielle. DEV Community +3

### Événements disponibles:

- BUILD_UPLOAD_STATE_CHANGED Apple Developer
- BUILD_BETA_STATE_CHANGED Apple Developer
- APP_STORE_VERSION_STATE_CHANGED Apple Developer
- BETA_FEEDBACK_SCREENSHOT_SUBMITTED Apple Developer
- BETA_FEEDBACK_CRASH_SUBMITTED Apple Developer
- APPLE_HOSTED_BACKGROUND_ASSET_STATE_CHANGED

### Configuration webhook:

json

```
POST /v1/webhooks
{
  "data": {
    "type": "webhooks",
    "attributes": {
      "url": "https://your-server.com/webhook",
      "secret": "your-secret-key",
      "events": [
        "BUILD_UPLOAD_STATE_CHANGED",
        "APP_STORE_VERSION_STATE_CHANGED"
      ]
    }
  }
}
```

## Rate Limits

- **Limite horaire**: 3,600 requêtes par heure Apple Developer
- **Limite par minute**: ~300-350 requêtes Apple Developer
- **Headers de réponse**: X-Rate-Limit, X-Rate-Limit-Remaining, X-Rate-Limit-Reset Apple Developer
- **Code erreur 429**: RATE_LIMIT_EXCEEDED Apple Developer

## 2. Xcode Cloud API

### Configuration des workflows CI/CD

L'API Xcode Cloud fait partie de l'App Store Connect API Apple Developer et permet l'automatisation complète des pipelines de build.

#### Endpoints principaux

```bash
# Workflows
GET /v1/ciWorkflows
POST /v1/ciWorkflows
PATCH /v1/ciWorkflows/{id}

# Build Runs
POST /v1/ciBuildRuns  # Déclencher un build
GET /v1/ciBuildRuns/{id}
DELETE /v1/ciBuildRuns/{id}  # Annuler

# Artifacts
GET /v1/ciArtifacts/{id}
GET /v1/ciBuildActions/{id}/artifacts
```

#### Déclenchement automatique de builds

```swift
```

```
func startWorkflow(workflowId: String, gitReferenceId: String) async throws {
    let buildRequest = CiBuildRunCreateRequest(
        data: .init(
            type: .ciBuildRuns,
            relationships: .init(
                workflow: .init(data: .init(type: .ciWorkflows, id: workflowId)),
                sourceBranchOrTag: .init(data: .init(type: .scmGitReferences, id: gitReferenceId))
            )
        )
    )

    let endpoint = APIEndpoint.v1.ciBuildRuns.post(buildRequest)
    let response = try await provider.request(endpoint)
    print("Build démarré: \(response.data.id)")
}
```

### Scripts personnalisés CI

Xcode Cloud supporte trois types de scripts dans le répertoire `ci_scripts/`: Apple Developer

```bash
#!/bin/bash
# ci_scripts/ci_pre_xcodebuild.sh

if [[ "$CI_XCODEBUILD_ACTION" == "archive" ]]; then
    echo "Configuration pour la production..."
    # Installation des dépendances
    brew install swiftlint
    pod install
fi
```

**Variables d'environnement disponibles:** Apple Developer

- `CI_WORKSPACE`: Chemin vers le workspace
- `CI_PRODUCT_PLATFORM`: Plateforme cible
- `CI_XCODEBUILD_ACTION`: Action en cours
- `CI_BUILD_NUMBER`: Numéro de build
- `CI_COMMIT`: SHA du commit

## 3. TestFlight API

### Gestion des beta testers et groupes

python

```python
class TestFlightAutomation:
    def create_beta_group(self, app_id, group_name):
        group_data = {
            "data": {
                "type": "betaGroups",
                "attributes": {
                    "name": group_name,
                    "isInternalGroup": False,
                    "publicLinkEnabled": True,
                    "publicLinkLimit": 1000
                },
                "relationships": {
                    "app": {"data": {"type": "apps", "id": app_id}}
                }
            }
        }

        response = requests.post(
            f"{self.base_url}/betaGroups",
            headers=self.get_headers(),
            json=group_data
        )
        return response.json()['data']['id']

    def add_testers_bulk(self, group_id, emails):
        for email in emails:
            tester_data = {
                "data": {
                    "type": "betaTesters",
                    "attributes": {
                        "email": email,
                        "firstName": email.split('@')[0],
                        "lastName": "Tester"
                    }
                }
            }
            requests.post(f"{self.base_url}/betaTesters", json=tester_data)
```

## Endpoints TestFlight

bash

```
# Gestion des builds
GET /v1/builds/{buildId}/betaBuildLocalizations
PATCH /v1/betaBuildLocalizations/{id}  # Mettre à jour "What to Test"

# Testers
POST /v1/betaTesters
POST /v1/betaTesterInvitations
GET /v1/betaTesters/{testerId}/metrics

# Groupes
POST /v1/betaGroups
POST /v1/betaGroups/{groupId}/relationships/builds
PATCH /v1/betaGroups/{groupId}  # Activer lien public

# Soumission pour review
POST /v1/betaAppReviewSubmissions
```

### Limites TestFlight

- **Testers internes**: 100 utilisateurs App Store Connect
- **Testers externes**: 10,000 par application Apple Developer Apple Developer
- **Builds maximum**: 100 par application
- **Expiration des builds**: 90 jours
- **Lien public**: Jusqu'à 10,000 testers Apple Developer 9to5Mac

## 4. Notarization API (macOS)

### notarytool (Outil moderne)

bash

```
# Soumettre pour notarisation
xcrun notarytool submit MyApp.dmg \
  --key /path/to/AuthKey.p8 \
  --key-id "KEY_ID" \
  --issuer "ISSUER_ID" \
  --wait

# Vérifier le statut
xcrun notarytool info [submission-id] \
  --keychain-profile "profile-name"

# Récupérer les logs
xcrun notarytool log [submission-id] \
  --keychain-profile "profile-name" \
  output.json
```

## Stapling automatique

bash

```
# Après notarisation réussie
xcrun stapler staple "MyApp.app"
xcrun stapler validate "MyApp.app"
```

## API REST pour notarisation

python

```python
def notarize_app(app_path, key_id, issuer_id, private_key):
    # 1. Générer JWT
    token = generate_jwt_token(key_id, issuer_id, private_key)

    # 2. Soumettre pour notarisation
    headers = {"Authorization": f"Bearer {token}"}

    with open(app_path, 'rb') as f:
        response = requests.post(
            "https://appstoreconnect.apple.com/notary/v2/submissions",
            headers=headers,
            files={'file': f}
        )

    submission_id = response.json()['id']

    # 3. Attendre la fin du traitement
    while True:
        status_response = requests.get(
            f"https://appstoreconnect.apple.com/notary/v2/submissions/{submission_id}",
            headers=headers
        )

        status = status_response.json()['status']
        if status in ['Accepted', 'Rejected']:
            break

        time.sleep(30)

    return status
```

## 5. Provisioning et Certificats API

### Gestion automatique des certificats

bash

```
# Créer un certificat
POST /v1/certificates
{
  "data": {
    "type": "certificates",
    "attributes": {
      "certificateType": "IOS_DISTRIBUTION",
      "csrContent": "base64-encoded-csr"
    }
  }
}

# Créer un profil de provisioning
POST /v1/profiles
{
  "data": {
    "type": "profiles",
    "attributes": {
      "name": "Production Profile",
      "profileType": "IOS_APP_STORE"
    },
    "relationships": {
      "bundleId": {"data": {"type": "bundleIds", "id": "bundle-id"}},
      "certificates": {"data": [{"type": "certificates", "id": "cert-id"}]}
    }
  }
}
```

## Enregistrement de devices

python

```python
def register_devices(udids):
    for udid in udids:
        device_data = {
            "data": {
                "type": "devices",
                "attributes": {
                    "name": f"Device {udid[:8]}",
                    "platform": "IOS",
                    "udid": udid
                }
            }
        }
        requests.post(f"{BASE_URL}/devices", json=device_data)
```

## 6. Fastlane - Orchestration complète

### Configuration Fastfile complète

ruby

```ruby
# Fastfile
default_platform(:ios)

platform :ios do
  before_all do
    # Configuration API
    app_store_connect_api_key(
      key_id: ENV["ASC_KEY_ID"],
      issuer_id: ENV["ASC_ISSUER_ID"],
      key_filepath: "./AuthKey.p8",
      duration: 1200
    )
  end

  desc "Pipeline complet de publication"
  lane :release_pipeline do
    # 1. Gestion des versions
    increment_build_number(build_number: number_of_commits)
    increment_version_number(bump_type: "patch")

    # 2. Certificats et provisioning
    match(
      type: "appstore",
      readonly: false,
      git_url: ENV["MATCH_GIT_URL"]
    )

    # 3. Build
    build_app(
      scheme: "MyApp",
      workspace: "MyApp.xcworkspace",
      export_method: "app-store",
      include_bitcode: true,
      include_symbols: true
    )

    # 4. Screenshots automatiques
    capture_screenshots
    frame_screenshots

    # 5. Upload TestFlight
    upload_to_testflight(
      skip_waiting_for_build_processing: false,
      changelog: "Nouvelles fonctionnalités",
      beta_app_feedback_email: "beta@company.com",
      groups: ["Beta Testers", "Internal Team"]
```

```
  )

  # 6. Soumission App Store
  upload_to_app_store(
   submit_for_review: true,
   automatic_release: false,
   force: true,
   metadata_path: "./fastlane/metadata",
   screenshots_path: "./fastlane/screenshots",
   submission_information: {
    add_id_info_uses_idfa: false,
    export_compliance_uses_encryption: false
   }
  )
 end
end
```

## Actions Fastlane disponibles

### App Store Connect:

- upload_to_app_store (deliver) Fastlane Fastlane
- upload_to_testflight (pilot) Fastlane +2
- app_store_build_number Fastlane
- latest_testflight_build_number
- download_dsyms
- precheck
- upload_app_privacy_details_to_app_store Fastlane Fastlane

### Code Signing:

- match (sync_code_signing) Fastlane
- cert (get_certificates)
- sigh (get_provisioning_profile)
- register_device
- register_devices

### Build:

- build_app (gym) Fastlane
- build_ios_app
- build_mac_app

- `xcarchive`
- `xcexport`

**Screenshots**:

- `capture_screenshots` (snapshot)
- `frame_screenshots` (frameit)

## 7. Transporter Tool (iTMSTransporter)

### Commandes principales

```bash
# Upload avec authentification JWT
iTMSTransporter -m upload -f package.itmsp -jwt $JWT_TOKEN

# Validation du package
iTMSTransporter -m verify -f package.itmsp -u username -p password

# Vérification du statut
iTMSTransporter -m status -u username -p password -apple_id 123456789

# Récupération des métadonnées
iTMSTransporter -m lookupMetadata -u username -p password -apple_id 123456789
```

Medium Medium

### Structure du package ITMSP

```
MyApp.itmsp/
├── metadata.xml
├── MyApp.ipa
├── screenshots/
│   ├── iOS-5.5-in/
│   │   ├── 01_screenshot.png
│   │   └── 02_screenshot.png
│   └── iOS-6.5-in/
│       ├── 01_screenshot.png
│       └── 02_screenshot.png
└── previews/
    └── preview.mov
```

Apple Developer +2

### Protocoles de transport optimisés

```bash
# HTTP (par défaut)
iTMSTransporter -m upload -f package.itmsp -t DAV

# Signiant (plus rapide pour gros fichiers)
iTMSTransporter -m upload -f package.itmsp -t Signiant

# Aspera (le plus rapide, nécessite configuration firewall)
iTMSTransporter -m upload -f package.itmsp -t Aspera
```

Fastlane +2

## 8. Outils CLI additionnels

### xcrun pour distribution

```bash
# Validation d'application
xcrun altool --validate-app -f MyApp.ipa -t ios --apiKey KEY_ID --apiIssuer ISSUER_ID

# Upload d'application
xcrun altool --upload-app -f MyApp.ipa -t ios --apiKey KEY_ID --apiIssuer ISSUER_ID

# Export avec upload direct
xcodebuild -exportArchive \
  -archivePath MyApp.xcarchive \
  -exportPath ./export \
  -exportOptionsPlist exportOptions.plist
```

Stack Overflow

### agvtool pour gestion des versions

```bash
```

```
# Afficher la version actuelle
agvtool what-version
agvtool what-marketing-version

# Incrémenter les versions
agvtool next-version -all
agvtool new-marketing-version 2.0.0

# Script automatique basé sur git
BUILD_NUMBER=$(git rev-list --count HEAD)
agvtool new-version $BUILD_NUMBER
```

Apple Developer DZone

## 9. Authentification et sécurité

### Types d'authentification

1. **JWT Token (Recommandé)**
   - Durée de vie: 20 minutes maximum Andy Ibanez +2
   - Algorithme: ES256 Tanaschita
   - Nécessite: Key ID, Issuer ID, Private Key (.p8) fastlane
2. **App-Specific Password**
   - Pour outils legacy
   - Génération sur appleid.apple.com
   - Stockage sécurisé dans keychain
3. **Certificats et Provisioning**
   - Types: Development, Distribution, Developer ID
   - Gestion via match pour centralisation Fastlane

### Stockage sécurisé des credentials

bash

```
# Keychain pour mots de passe
security add-generic-password -a "apple-id" -w "password" -s "notary-password"

# Variables d'environnement CI/CD
export ASC_KEY_ID="YOUR_KEY_ID"
export ASC_ISSUER_ID="YOUR_ISSUER_ID"
export ASC_PRIVATE_KEY_PATH="/secure/path/AuthKey.p8"

# Permissions fichiers
chmod 600 ~/.appstoreconnect/private_keys/*.p8
chmod 700 ~/.appstoreconnect/private_keys/
```

## 10. Limites et optimisation

### Limites de taux par API

| API | Limite horaire | Limite par minute | Timeout token |
|---|---|---|---|
| App Store Connect | 3,600 | ~300-350 | 20 min |
| Notarization | Pas de limite fixe | Variable | 20 min |
| TestFlight | 3,600 | ~300-350 | 20 min |

### Stratégies d'optimisation

python

```python
# Retry avec backoff exponentiel
import time

def api_call_with_retry(func, max_retries=3):
    for attempt in range(max_retries):
        try:
            return func()
        except RateLimitError as e:
            if attempt == max_retries - 1:
                raise
            wait_time = 2 ** attempt * 30  # 30, 60, 120 secondes
            time.sleep(wait_time)
```

### Traitement parallèle

ruby

```
# Fastlane - Screenshots parallèles
concurrent_simulators(true)
max_concurrent_simulators(4)

# Uploads multiples
lanes = [:ios, :tvos, :macos]
lanes.each do |platform|
  Thread.new { send(platform, :upload) }
end
```

Fastlane Fastlane

## 11. Pipeline CI/CD complet

### GitHub Actions exemple

yaml

```yaml
name: iOS Release Complete
on:
  push:
    tags: ['v*']

jobs:
  release:
    runs-on: macos-latest
    steps:
      - uses: actions/checkout@v3

      - name: Setup environment
        run: |
          echo "${{ secrets.ASC_PRIVATE_KEY }}" | base64 -d > AuthKey.p8
          bundle install

      - name: Run tests
        run: bundle exec fastlane test

      - name: Build and sign
        env:
          ASC_KEY_ID: ${{ secrets.ASC_KEY_ID }}
          ASC_ISSUER_ID: ${{ secrets.ASC_ISSUER_ID }}
          MATCH_PASSWORD: ${{ secrets.MATCH_PASSWORD }}
        run: |
          bundle exec fastlane match appstore --readonly
          bundle exec fastlane build_app

      - name: Upload to TestFlight
        run: bundle exec fastlane upload_to_testflight

      - name: Submit for review
        if: contains(github.ref, 'release')
        run: bundle exec fastlane upload_to_app_store submit_for_review:true

      - name: Notarize macOS version
        if: matrix.platform == 'macos'
        run: |
          xcrun notarytool submit MyApp.dmg \
            --key AuthKey.p8 \
            --key-id ${{ secrets.ASC_KEY_ID }} \
            --issuer ${{ secrets.ASC_ISSUER_ID }} \
            --wait
          xcrun stapler staple MyApp.dmg
```

## Webhook handler pour automatisation complète

python

```python
from flask import Flask, request
import hmac
import hashlib
import subprocess

app = Flask(__name__)

@app.route('/webhook/apple', methods=['POST'])
def handle_webhook():
    # Vérifier signature
    signature = request.headers.get('X-Apple-Signature')
    body = request.get_data()

    expected = hmac.new(
        WEBHOOK_SECRET.encode(),
        body,
        hashlib.sha256
    ).hexdigest()

    if not hmac.compare_digest(f'sha256={expected}', signature):
        return 'Unauthorized', 401

    payload = request.get_json()
    event_type = payload.get('eventType')

    if event_type == 'BUILD_UPLOAD_STATE_CHANGED':
        build_id = payload['data']['id']
        if payload['data']['attributes']['state'] == 'COMPLETE':
            # Déclencher distribution TestFlight
            subprocess.run([
                'fastlane', 'distribute_beta',
                f'build_id:{build_id}'
            ])

    elif event_type == 'APP_STORE_VERSION_STATE_CHANGED':
        if payload['data']['attributes']['appStoreState'] == 'READY_FOR_SALE':
            # Notifier l'équipe
            send_slack_notification("App publiée sur l'App Store!")

    return 'OK', 200
```

Apple Developer

## Conclusion

Cette documentation exhaustive couvre l'ensemble des API Apple disponibles pour l'automatisation complète du processus de publication d'applications. Apple Developer +4 Les nouveautés 2025 incluent notamment les webhooks pour App Store Connect, l'API de build upload, Apple Developer et l'amélioration significative de l'API Xcode Cloud. Apple Developer apple L'utilisation combinée de ces API avec des outils comme fastlane permet de créer des pipelines CI/CD sophistiqués GitHub +2 réduisant drastiquement le temps et l'effort nécessaires pour publier des applications sur l'App Store. GitHub Fastlane