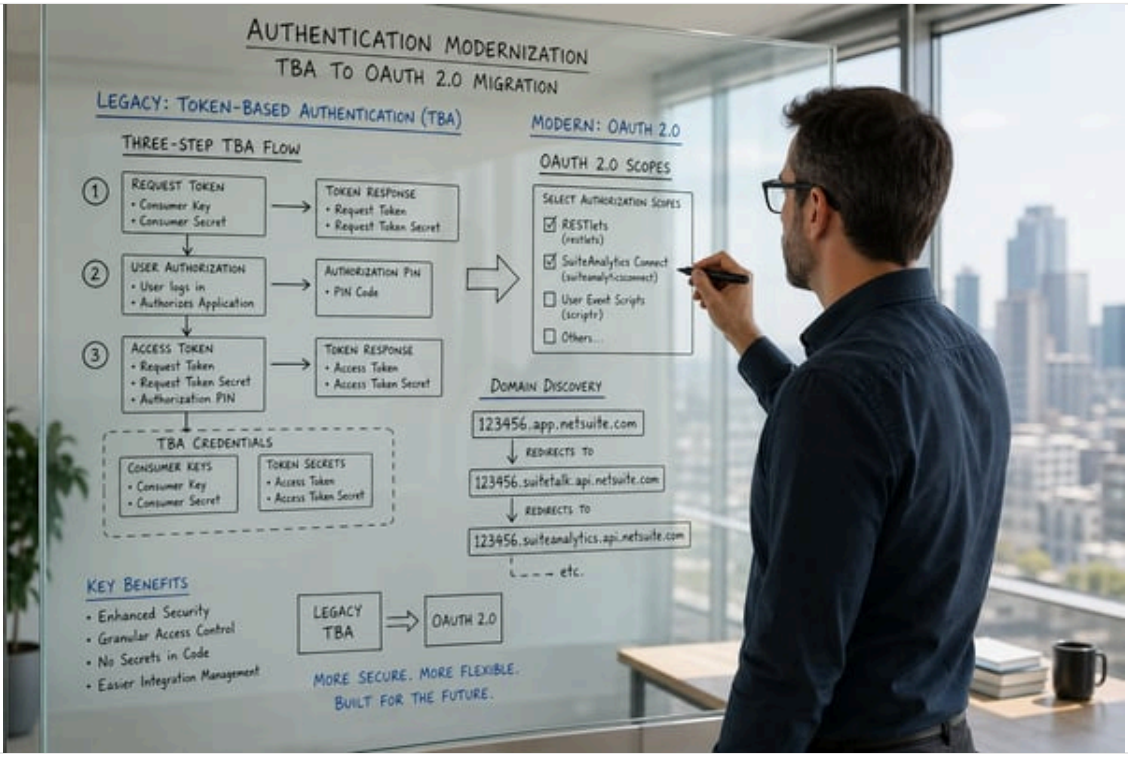
NetSuite RESTlet URLs: Domains, TBA & OAuth 2.0 Scopes

Published April 25, 2026 27 min read



Executive Summary

This report provides a comprehensive analysis of Oracle NetSuite's use of **account-specific domains** for RESTlet URLs and the authentication flows for accessing those URLs, focusing on both NetSuite's legacy Token-Based Authentication (TBA, an OAuth 1.0–based method) and the new OAuth 2.0 framework with scopes. Each NetSuite account (production, sandbox, release preview) has a unique tenant-specific domain (for example, `123456.app.netsuite.com` for a production account and `123456-sb1.app.netsuite.com` for a sandbox) (Source: docs.oracle.com) (Source: docs.oracle.com). These domains remain constant even if the account is migrated between data centers (Source: docs.oracle.com), eliminating the need to hard-code data-center-specific URLs. To discover the correct domain for any account, NetSuite provides REST discovery services (such as `rest.netsuite.com` DataCenterURLs and REST Roles services) or the SuiteScript `N/url.resolveDomain` API (Source: docs.oracle.com) (Source: docs.oracle.com).

On the security side, *Token-Based Authentication* (TBA) has been the longstanding method for RESTlet (and SuiteTalk) integrations. TBA uses OAuth 1.0a with HMAC-SHA256 signatures and four credentials (consumer key/secret plus token ID/secret) (Source: www.brokenrubik.com) (Source: docs.oracle.com). In this three-step flow, an external client (1) POSTs to the `/rest/requesttoken` endpoint to get an unauthorized request token, (2) directs the user to the `/app/login/secure/authorizetoken.nl` endpoint for authorization (JSO credentials/ SAML SSO and consent), then (3) POSTs to `/rest/accesstoken` to exchange the authorized token for an access token and secret (Source: docs.oracle.com) (Source: docs.oracle.com) (Source: docs.oracle.com). Once obtained, the OAuth 1.0 access token (plus secret) is included as HMAC header in every RESTlet call, e.g. with `Authorization: OAuth realm="<ACCOUNT_ID>", oauth_consumer_key="...", oauth_token="...", oauth_signature_method="HMAC-SHA256", …` (Source: www.brokenrubik.com) (Source: timdietrich.me). Fitzgerald's guide emphasizes that TBA "is the recommended approach for server-to-server" integrations (Source: www.brokenrubik.com) and that only HMAC-SHA256 is supported (SHA-1 is retired as of 2023.1) (Source: docs.oracle.com).

However, Oracle has declared that **TBA is being phased out**. NetSuite 2027.1 and onward will allow **no new RESTlet or web service integrations using TBA** (Source: docs.oracle.com) (Source: www.houseblend.io). Instead, OAuth 2.0 is now the preferred method. OAuth 2.0 simplifies the process (no per-request signature) and supports modern security features (bearer tokens, refresh tokens, stronger encryption, integration with SAML/OIDC

login, 2FA) (Source: docs.oracle.com) (Source: www.houseblend.io). In particular, OAuth 2.0 introduces fine-grained *scopes* into NetSuite's integration model. In practice, scopes are configured on the Integration record by checking features such as **RESTlets**, **REST Web Services**, **SuiteAnalytics Connect**, etc. (Source: docs.oracle.com). An OAuth 2.0 access token will only be valid for the scopes enabled on its application (Source: docs.oracle.com) (Source: blogs.oracle.com). For example, a JWT access token issued for a REST Web Services call might have `"scope": ["RESTLETS"]` in its payload (Source: docs.oracle.com), indicating it grants RESTlet access. (Oracle's documentation for *SuiteProjects Pro* similarly shows scope values like `rest`, `soap`, `xml`, `bi` that can be combined (Source: docs.oracle.com), analogous to NetSuite's scopes.)

The following sections elaborate on NetSuite account-specific domains and URL formats, detail the mechanics of the TBA flow, and explain NetSuite's OAuth 2.0 integration flows and scopes. We compare authentication methods (TBA vs OAuth2) and discuss implications for integration design, security, and future directions. Each claim is supported by Oracle's documentation, expert blogs, and case studies.

Introduction

Oracle NetSuite is a cloud ERP platform used by tens of thousands of businesses worldwide (Source: www.houseblend.io). As a multi-tenant SaaS ERP, each account (tenant) is identified by a numeric ID (with suffixes for sandbox or release preview). NetSuite provides various APIs for integration, notably SuiteTalk (SOAP and REST Web Services and SuiteScript script endpoints (RESTlets and Suitelets). A **RESTlet** is a custom SuiteScript (SuiteScript 2.x deployed on a NetSuite account that exposes HTTP endpoints (GET/POST/PUT/DELETE) to external clients (Source: www.brokenrubik.com) (Source: timdietrich.me). RESTlets let you implement arbitrary business logic and data processing directly in NetSuite, providing JSON or text responses. They complement SuiteTalk REST (the built-in record-based REST API) by allowing complex or specialized operations in full SuiteScript.

When calling any NetSuite REST endpoint (SuiteTalk or RESTlet) from an outside client, the request URL must use the *account's unique domain*. Historically, NetSuite URLs included data center identifiers (e.g. *.restlets.api.netsuite.com* vs *.restlets.api.na3.netsuite.com*), which required updates if an account was moved between datacenters. Today, Oracle uses **account-specific domains** so that URLs remain stable across migrations (Source: docs.oracle.com). In other words, the domain includes the account ID (and account type) itself, and does not change when NetSuite rebalances data availability (Source: docs.oracle.com) (Source: docs.oracle.com). For example, a production account with ID `123456` might use URLs like `https://123456.app.netsuite.com/...` or `https://123456.suitetalk.api.netsuite.com/...`, whereas a sandbox account `123456_SB1` uses `https://123456-sb1.app.netsuite.com/...` (note underscore replaced by hyphen and suffix lowercased) (Source: docs.oracle.com).The official account-specific URL for each service is listed on the *Company URLs* subtab in NetSuite's Setup > Company > Company Information page (Source: docs.oracle.com) (Source: docs.oracle.com). Administrators should never manually "construct" these URLs; instead, use the values from *Company URLs* or dynamic discovery APIs (Source: docs.oracle.com) (Source: docs.oracle.com).

To handle multiple accounts or unknown accounts dynamically, NetSuite provides discovery services. For SOAP (SuiteTalk) clients, the `getDataCenterUrls` SOAP call or the `webservices.netsuite.com` endpoint can be used. For REST clients (RESTlets, REST web services, etc.), Oracle provides a REST-based DataCenterUrls service and a standalone *REST Roles Service* at `rest.netsuite.com` (Source: docs.oracle.com). Calling `https://rest.netsuite.com` with valid credentials returns the correct base domains (e.g. the `.suitetalk.api.netsuite.com` or `.app.netsuite.com` URLs) for that account. Similarly, within SuiteScript 2.x you can call the `N/url` module's `resolveDomain({ hostType: url.HostType.RESTLET, accountId: 'ACCT' })` function to retrieve the proper RESTlet domain (Source: docs.oracle.com). These dynamic discovery mechanisms ensure that client integrations always use the correct, up-to-date domain for any NetSuite account.

Table 1 below summarizes the main NetSuite integration options and their characteristics, illustrating where RESTlets fit alongside SuiteTalk SOAP/REST and Suitelets (Source: www.brokenrubik.com).

Table 1: Comparison of NetSuite Integration Methods (adapted from [BrokenRubik][4])

FEATURE	RESTLETS	SUITETALK SOAP WEB SERVICES	SUITETALK REST WEB SERVICES	SUITELETS (UI SCRIPTS)
Logic	Full SuiteScript (custom code)	Limited (CRUD via WSDL)	Limited (CRUD via standard REST)	Full SuiteScript
Response Format	JSON or Text	XML (SOAP)	JSON	HTML or JSON
Authentication	TBA (OAuth 1.0) / OAuth 2.0 Bearer	TBA / OAuth 1.0	(Only) OAuth 2.0	Session ID / Token
Governance Units	5,000 units/request	N/A	N/A	1,000 units/request
Typical Use Cases	Custom APIs & data logic	Standard record CRUD (all types)	Standard record CRUD	Custom web pages (UI)

(Table 1 sources: NetSuite documentation and integration guides [4], supplemented by [52] for authentication modes.)

This context sets the stage for our deep dive into RESTlet URLs and authentication. The next sections discuss **account-specific domains** in detail, then explain **TBA flow** end-to-end, and finally treat **OAuth 2.0 and scopes**.

NetSuite Account-Specific Domains and RESTlet URLs

NetSuite's shift to account-specific domains means each account has its own unique domain prefix for all services (Source: docs.oracle.com) (Source: docs.oracle.com). The domain string includes the account ID and also encodes the account type. For a production account (e.g. ID `123456`), you might see domains like `123456.app.netsuite.com` or `123456.suitetalk.api.netsuite.com`. For a sandbox ID such as `123456_SB1`, the domain becomes `123456-sb1.app.netsuite.com` (underscore → hyphen, all lowercase) (Source: docs.oracle.com). Release Preview accounts (e.g. `123456_RP`) similarly use `123456-rp...` domains (Source: docs.oracle.com). Figure 1 (in Table 2) lists examples of account-specific domain formats:

Table 2: Account-Specific Domain Examples by Account Type

ACCOUNT TYPE	EXAMPLE ACCOUNT ID FORMAT	ACCOUNT-SPECIFIC DOMAIN URL EXAMPLE
Production	`123456`	`123456.app.netsuite.com` (UI/RESTlet endpoints)
Sandbox	`123456_SB1` → `123456-sb1`	`123456-sb1.app.netsuite.com` (UI/RESTlet endpoints)
Release Preview	`123456_RP` → `123456-rp`	`123456-rp.app.netsuite.com` (UI/RESTlet endpoints)

Each account also has corresponding domains for SOAP/REST web services (using the `.suitetalk.api.netsuite.com` suffix) and for other services like SuiteAnalytics and external Catalog Sites (Source: docs.oracle.com) (Source: docs.oracle.com). For example, a SuiteTalk REST endpoint might be `https://123456.suitetalk.api.netsuite.com/services/rest/record/v1/customer/...` (Source: docs.oracle.com). The precise URLs for all relevant services are visible on the *Company URLs* subtab for that account (Source: docs.oracle.com). **Importantly**, Oracle advises **never to hard-code these domains**. Instead, use the provided values or discovery APIs, since even account-specific domains, though stable across data centers, could change in future architectures. Indeed, NetSuite forbids certificate pinning on these domains because they can change without notice (Source: docs.oracle.com) (Source: docs.oracle.com).

Because accounts can be moved among data centers and enterprises often integrate multiple accounts, NetSuite provides domain discovery endpoints. For RESTlet clients, calling `https://rest.netsuite.com` (NetSuite's REST discovery service) with the account credentials returns the correct RESTlet domain names (Source: docs.oracle.com). Alternatively, SuiteScript can programmatically find domains: e.g.

```
N/url.resolveDomain({ hostType: url.HostType.RESTLET, accountId: '123456' });
```

will return the RESTlet domain for account 123456 (Source: docs.oracle.com). Using these dynamic methods avoids brittle dependencies on fixed URLs.

Once the correct domain is known, a RESTlet is invoked by constructing an HTTPS request to that domain with the RESTlet's script and deploy identifiers in the path or query parameters. For example, a typical external RESTlet call uses a URL of the form:

```
https://<accountDomain>/app/site/hosting/restlet.nl?script=<SCRIPT_ID>&deploy=<DEPLOY_ID>
```

For a production account ID 123456, the example domain is `123456.app.netsuite.com`, so a call might look like `https://123456.app.netsuite.com/app/site/hosting/restlet.nl?script=1&deploy=1` (Source: docs.oracle.com). (NetSuite's REST Web Services use a different base path under `/services/rest/...`, as shown in [52].) Since the account domain is specific to each tenant, clients must use the correct domain for the target account. The policy is: *"the URL must include a domain specific to your NetSuite account"* (Source: docs.oracle.com). If you access another account (e.g. a sandbox or customer account), you must use that account's domain, not your own.

In practice, most integrations handle domain selection in one of these ways:

- **Static configuration:** Administrators copy the correct domain from NetSuite's UI and store it in the integration's settings or code.
- **Dynamic discovery:** At runtime, the integration first queries NetSuite (e.g. `rest.netsuite.com` or SuiteScript) to obtain the domain.
- **N/url.resolveDomain:** If written as a SuiteScript (Suitelet) within NetSuite, use the `N/url` module as shown in *Retrieve the Domain for Calling a RESTlet* (Source: docs.oracle.com).

In summary, account-specific domains decouple integrations from physical data center addresses (Source: docs.oracle.com) and must always be used in RESTlet URLs. Table 2 highlights how the account ID maps into domains for different account types (Source: docs.oracle.com) (Source: docs.oracle.com). The next sections discuss authentication when calling these RESTlet URLs.

NetSuite RESTlet Authentication

RESTlets require authentication for every external call (Source: www.brokenrubik.com) (Source: docs.oracle.com). NetSuite supports multiple auth methods, but for external RESTlet clients the main options are **Token-Based Authentication (OAuth 1.0a)** and **OAuth 2.0** (Source: docs.oracle.com). (A legacy method called NLAuth — which passes company, user name, password, role, etc. in an HTTP header — exists but Oracle discourages it for new integrations (Source: docs.oracle.com).) Critically, any RESTlet call from outside must have an HTTP `Authorization` header (no anonymous calls except if the RESTlet is explicitly deployed *without* login, which is rare).

In general, Oracle's current guidance is that **OAuth 2.0 is preferred** for RESTlets when possible (Source: docs.oracle.com). However, many existing integrations still use NetSuite's Token-Based Authentication (TBA) because it has been the de facto standard for server-to-server RESTlet/SOAP integrations for years. We will detail both approaches below.

Before diving into flows, note one more restriction: if the NetSuite role is marked **"Web Services Only"**, that user cannot call RESTlets at all (Source: docs.oracle.com) (Source: docs.oracle.com). RESTlets are a *SuiteScript* feature (part of the SuiteCloud platform), so the user or integration role must have normal SuiteScript permissions. (The docs caution that a WebServices-Only role will get an `INVALID_LOGIN_CREDENTIALS` if it attempts a RESTlet call (Source: docs.oracle.com).) In practice, one creates a dedicated integration role (or uses an existing role) with appropriate permissions, and then under Setup → Users/Roles creates an *Access Token* for that role (for TBA) or uses that role in an OAuth2 flow.

Token-Based Authentication (TBA) Flow

Token-Based Authentication in NetSuite is based on OAuth 1.0a. In this scheme, the integration sets up an **Integration Record** that generates a Consumer Key and Secret (Source: www.brokenrubik.com). Then an administrator creates an **Access Token** for a specific user and role under *Setup → Users/Roles → Access Tokens* (Source: www.brokenrubik.com), yielding a Token ID and Token Secret. The integration uses these four values (consumer key/secret and token id/secret) to sign API requests. Each RESTlet request includes an OAuth 1.0 header that NetSuite verifies.

The TBA setup steps are: enable TBA in account, create Integration record (get consumer key/secret), and create Access Token for a user-role (get token id/secret) (Source: www.brokenrubik.com). After setup, the three-step OAuth flow is as follows (Oracle calls it the "Three-Step TBA Authorization Flow" (Source: docs.oracle.com):

1. **Request Token (Step One):** The client sends an HTTP POST to the "request token" endpoint. The URL format is:

```
https://<ACCOUNT_ID>.restlets.api.netsuite.com/rest/requesttoken
```

where `<ACCOUNT_ID>` is the account's ID. (If the account ID is not known, one can also POST to `https://system.netsuite.com/rest/requesttoken` as a fallback (Source: docs.oracle.com).) The client includes an OAuth 1.0a Authorization header signed with consumer key/secret (no token yet) plus a nonce and timestamp (Source: docs.oracle.com). If successful, NetSuite returns an "unauthorized request token".

2. **User Authorization (Step Two):** The client then directs the user (or browser session) to the NetSuite user authorization URL with the request token. This is a simple HTTP GET to:

```
https://<ACCOUNT_ID>.app.netsuite.com/app/login/secure/authorizetoken.nl?oauth_token=<REQUEST_TOKEN>
```

including the `oauth_token` query parameter from Step One. (Source: docs.oracle.com). NetSuite will prompt the user to log in (if not already) and then display a consent/approval page. The user must log in using the appropriate role (NetSuite will allow selecting a role if needed (Source: docs.oracle.com) and click "Allow" to grant the request token access. At that point NetSuite redirects the browser back to the callback URL specified in the Integration record, with parameters including `oauth_token`, `oauth_verifier`, `company` (account ID), and `role` (the role internal ID) in the query string (Source: docs.oracle.com). (If the user clicks "Deny", the flow ends.)

3. **Access Token Exchange (Step Three):** Finally, the application takes the authorized `oauth_token` and the `oauth_verifier` from Step Two and posts back to the NetSuite access-token endpoint:

```
POST https://<ACCOUNT_ID>.restlets.api.netsuite.com/rest/accesstoken
```

with an OAuth 1.0a header that includes both the consumer credentials and the request token (and now the verifier) (Source: docs.oracle.com). The response returns the final **Access Token** and **Token Secret** (`oauth_token` and `oauth_token_secret`), which the client will use for subsequent authenticated requests (Source: docs.oracle.com). (As of NetSuite 2020.1, the OAuth `realm` parameter is optional and not required in this request (Source: docs.oracle.com).)

After Step Three, the integration has all four credentials for future RESTlet calls (consumer key/secret and access token/secret). Each future HTTP call to the RESTlet's URL includes an Authorization header like:

```
Authorization: OAuth
    realm="<ACCOUNT_ID>",
    oauth_consumer_key="<CONSUMER_KEY>",
    oauth_token="<TOKEN_ID>",
    oauth_signature_method="HMAC-SHA256",
    oauth_timestamp="<UNIX_TIMESTAMP>",
    oauth_nonce="<RANDOM>",
    oauth_version="1.0",
    oauth_signature="<SIGNATURE>"
```

where `<SIGNATURE>` is the HMAC-SHA256 of the base string using both secrets (Source: www.brokenrubik.com) (Source: timdietrich.me). For example, Tim Dietrich's guide shows precisely this format, including the `realm` (NetSuite account ID) and requiring HMAC-SHA256 (Source: timdietrich.me). With that header, the RESTlet receives the request as an authenticated API call. NetSuite logs each call against the Integration record's consumer key, enabling activity tracking (Source: docs.oracle.com).

It is important to note that **as of NetSuite 2027.1, Oracle will no longer allow creation of *new* TBA integrations** (Source: docs.oracle.com) (Source: www.houseblend.io). Existing TBA-based calls will continue to work but all new projects must use OAuth 2.0. This warning has appeared in multiple NetSuite docs and blog posts (Source: docs.oracle.com) (Source: www.houseblend.io). In fact, Houseblend's 2026 migration guide explicitly stresses that organizations must transition to OAuth 2.0 to remain compliant (Source: www.houseblend.io). (Legacy OAuth1.0 flows require per-request signatures and have no native refresh tokens, making them less aligned with modern security standards (Source: www.houseblend.io).)

Table 3 below outlines the RESTlet TBA flow steps with example endpoints:

STEP	PURPOSE	HTTP METHOD & ENDPOINT	NOTES
1. Request Token	Obtain unauthorized OAuth token	`POST https://<ACCOUNT>.restlets.api.netsuite.com/rest/requesttoken` (Source: docs.oracle.com)	Include OAuth header (consumer key/secret, nonce, timestamp).
2. Authorize Request Token	User grants access (login & consent)	`GET https://<ACCOUNT>.app.netsuite.com/app/login/secure/authorizetoken.nl?oauth_token=<REQUEST_TOKEN>` (Source: docs.oracle.com)	Redirect user to this URL. After login, NetSuite will redirect back with `oauth_verifier`.
3. Obtain Access Token	Exchange for final OAuth token	`POST https://<ACCOUNT>.restlets.api.netsuite.com/rest/accesstoken` (Source: docs.oracle.com)	Include OAuth header (consumer key/secret, request token/secret, verifier). Responds with `oauth_token`, `oauth_token_secret`.

Table 3: OAuth 1.0 (TBA) Flow for NetSuite RESTlets. Example domains use the account-specific domain pattern; see text for full URL formats.

The integration record is created (if not pre-created) when exchanging the token. Crucially, NetSuite automatically **installs the Integration record** for the requesting application with the new token (Source: docs.oracle.com) (Source: docs.oracle.com). An administrator can control whether the record is auto-enabled via the *SOAP Web Services Preferences* (Require Approval during auto-install) (Source: docs.oracle.com) (Source: docs.oracle.com). This ensures that the integration's consumer key gets registered in the account and can be tracked or blocked if needed (Source: docs.oracle.com). (An integration's calls can be reviewed on the *Integration Record* or via the *RESTlets Execution Log* (Source: docs.oracle.com).)

TBA Example and Considerations

In practical terms, library support (e.g. OAuth 1.0a libraries) or tools like Postman can handle the TBA signing. The BrokenRubik blog gives an example Header structure and flow (Source: www.brokenrubik.com) (Source: www.brokenrubik.com). One must also handle nuances like URL encoding and parameter inclusion in the signature base string (Source: docs.oracle.com) (Source: docs.oracle.com). Note that NetSuite's TBA expiration is tied to the integration record state; tokens do not expire by themselves unless revoked. Also, if the integration's role requires enhancements (e.g. 2FA or changed permissions), the user may need to repeat these steps to re-authorize. The NetSuite docs offer an alternate *IssueToken* endpoint for non-user-interactive token issuance, but Oracle recommends using the full three-step flow for new integrations (Source: docs.oracle.com).

Critical point: TBA requires **HMAC-SHA256** signatures—HMAC-SHA1 is no longer supported as of 2023.1 (Source: docs.oracle.com). Every TBA call signature must use SHA-256. Also, the OAuth 1.0 realm parameter (account ID) is now optional (Source: docs.oracle.com). When building an application, ensure your OAuth library is configured for NetSuite's exact requirements (signature method, encoding, parameter ordering, etc.).

OAuth 2.0 Authentication and Scopes

Oracle now provides OAuth 2.0 support for NetSuite's REST APIs (RESTlets and SuiteTalk REST). Unlike TBA, OAuth 2.0 uses bearer tokens (no per-request signature) and is easier to implement for both user-centric and machine-centric scenarios (Source: www.brokenrubik.com) (Source: docs.oracle.com). OAuth 2.0 integration setup involves: enabling the OAuth 2.0 feature in the account, creating an Integration record configured for OAuth2, and then running either the Authorization Code or Client Credentials flow in the client app (Source: docs.oracle.com) (Source: docs.oracle.com).

The Integration record (Setup > Integration > Manage Integrations) has an *Authentication* subtab where two grant types can be enabled: **Authorization Code Grant** (for interactive flows) and **Client Credentials (Machine-to-Machine) Grant** (for server-to-server flows) (Source: docs.oracle.com) (Source: docs.oracle.com). One can check both boxes if needed. The record also includes fields for redirect URIs (for code grant), and a "Public Client" checkbox/PKCE settings if using a public client (no secret) [* (Source: docs.oracle.com)*]. Importantly, the integration record has checkboxes labeled

RESTlets, *REST Web Services*, *SuiteAnalytics Connect*, *AI Connector Service*, etc. Checking **RESTlets** authorizes this integration to call RESTlets; checking **REST Web Services** authorizes SuiteTalk REST; and so on (Source: docs.oracle.com). When you later request an OAuth 2.0 token, the scopes of that token will be confined to these selected resources.

For a user-delegated scenario, the **Authorization Code Grant** flow works as follows (this is standard OAuth2 code grant). The application directs the user's browser to NetSuite's authorization endpoint with query parameters: `response_type=code`, `client_id=<CLIENT_ID>`, `redirect_uri=<your_app_URI>`, `scope=<scopes>`, and a random `state` string (Source: docs.oracle.com). The endpoint URL is of the form:

```
https://<ACCOUNT_DOMAIN>/login/oauth2/v1/authorize
    ?response_type=code
    &client_id=<clientId>
    &redirect_uri=<yourRedirectUri>
    &scope=<scopeList>
    &state=<randomString>
```

For example (illustrative): `https://123456.app.netsuite.com/login/oauth2/v1/authorize?response_type=code&client_id=MyAppID&redirect_uri=https://app.example.com/callback&scope=rest+soap+xml&state=xyz` (Source: docs.oracle.com). Note that `<ACCOUNT_DOMAIN>` is the account-specific domain. The user logs in (or is SSO'd) and approves the scopes. NetSuite then redirects to the `redirect_uri` with a `code` parameter and the same `state`.

Next, the client application exchanges this code for tokens by POSTing to the token endpoint at, for example,

```
https://<ACCOUNT_DOMAIN>/services/rest/auth/oauth2/v1/token
```

with form data `grant_type=authorization_code`, `code=<AUTH_CODE>`, `redirect_uri=<same>`, and using HTTP Basic auth with `client_id:client_secret` (Source: blogs.oracle.com) (Source: blogs.oracle.com). The response is a JSON Web Token access token (plus a refresh token) in NETSUITE's JWT format (Source: blogs.oracle.com) (Source: blogs.oracle.com). The example from the Oracle blog demonstrates the endpoint and that the bearer token is JWT:

```
POST https://123456.suitetalk.api.netsuite.com/services/rest/auth/oauth2/v1/token
?code=<CODE>&redirect_uri=https://app.example.com/callback&grant_type=authorization_code
```

with `Authorization: Basic <base64(client_id:secret)>` yields a JSON body containing `"access_token": "eyJraWQiOiIyMDIwXzEiLCJ0eXAiOiJKV1QiLCJh...","refresh_token":"...","scope":["RESTLETS"]...` (Source: blogs.oracle.com) (Source: blogs.oracle.com). (In this example the returned JWT's "scope" claim happens to be `RESTLETS` indicating that the token may call RESTlets; similarly it could include `"soap"` or `"rest"` if SuiteAnalytics or SOAP were enabled on the integration.)

The **Client Credentials Grant** is used for server-to-server when user interaction is not needed. In NetSuite, this often involves an RSA key pair. The administrator uploads the public certificate via *Setup → Integration → Manage Authentication → OAuth 2.0 Client Credentials (M2M) Setup* (Source: docs.oracle.com) and selects the integration application whose "Client Credentials Grant" box is checked (Source: docs.oracle.com) (Source: docs.oracle.com). At runtime the client uses its private key to sign a JWT assertion to request a token from the same `/token` endpoint (with `grant_type=client_credentials`). The response is an access token (JWT) which the integration can use. The Houseblend guide notes that Client Credentials is ideal for the long-term machine-to-machine connection, and often paired with a short-lived code grant for initial setup (Source: www.bundlet.com) (Source: www.bundlet.com). Indeed, a recommended pattern is: use a brief Authorization Code Grant (with an admin-level role selected) to complete one-time provisioning tasks, then switch to Client Credentials for ongoing access (Source: www.bundlet.com) (Source: www.bundlet.com).

OAuth 2.0 Scopes

In OAuth 2.0, **scopes** define the extent of access granted by a token. In NetSuite's implementation, the scopes correspond directly to the features enabled on the Integration record. Key scopes include `RESTlets` (allow calling RESTlets), `REST web services` (SuiteTalk REST), `SuiteAnalytics Connect`, and others (Source: docs.oracle.com). When the application requests authorization, it may include a scope parameter (e.g. `scope=rest+soap+xml` as shown in an example for SuiteProjects (Source: docs.oracle.com). Ultimately, the issued token's JWT contains a list of

scopes in its payload, and Oracle enforces that the token can only be used for the corresponding APIs. For example, the REST Web Services example token included `"scope":["RESTLETS"]` (Source: docs.oracle.com), meaning it permitted RESTlet calls. Changing or requesting different scopes typically requires a new authorization flow; for instance, attempting to combine incompatible scopes (like a BI connector and other scopes) will yield an `invalid_scope` error (Source: docs.oracle.com) in SuiteProjects scenarios.

From the integration perspective, the **Integration Record's checkboxes are effectively the OAuth scopes**. As the official help describes, you *"check [RESTlets] if your OAuth 2.0 integration requires accessing RESTlets",* and similarly for REST web services, Connect, AI Connector, etc. (Source: docs.oracle.com). These ensure the token (when acquired) is automatically scoped. In effect, NetSuite ties OAuth scope to which API features the integration was permitted to use. Houseblend explains that OAuth2 allows "fine-grained scopes" while OAuth1.0/TBA did not (Source: www.houseblend.io); indeed, NetSuite's scopes provide more precise control over token permissions than the all-or-nothing TBA approach.

After obtaining the access token, the RESTlet calls include a simple bearer header:

```
Authorization: Bearer <access_token>
```

as shown in the docs and examples (Source: docs.oracle.com) (Source: docs.oracle.com). No signature is needed. NetSuite validates the token's JWT, checks its signature (with NetSuite's public key), and ensures it covers the requested resource. For RESTlets, the actual call URL might be, for instance, `https://123456.app.netsuite.com/app/site/hosting/restlet.nl?script=1&deploy=1` (Source: docs.oracle.com), and the header `Authorization: Bearer eyJraWQiOiI...` (Source: docs.oracle.com). The BrokenRubik tutorial and Oracle examples confirm this usage (Source: docs.oracle.com) (Source: docs.oracle.com).

Example: Authorization Code Flow

To illustrate, consider an authorization code grant for a RESTlet integration. First, the integration record is created with **Authorization Code Grant** enabled, the appropriate Redirect URI(s) entered, and the scopes (RESTlets, etc.) checked (Source: docs.oracle.com) (Source: docs.oracle.com). The user logs into the client app, which navigates a browser to:

```
https://123456.app.netsuite.com/login/oauth2/v1/authorize
    ?response_type=code
    &client_id={CLIENT_ID}
    &redirect_uri=https://myapp.com/callback
    &scope=restlets+reststore  (for example)
    &state=random123
```

(The exact scopes syntax uses `+` or `%20` to separate values (Source: docs.oracle.com).) NetSuite then prompts the user to authorize, and if granted, redirects to `https://myapp.com/callback?code=ABC123&state=random123`. The client exchanges `ABC123` at

```
POST https://123456.suitetalk.api.netsuite.com/services/rest/auth/oauth2/v1/token
```

with headers `Authorization: Basic <base64(client_id:client_secret)>` and body `grant_type=authorization_code&code=ABC123&redirect_uri=https://myapp.com/callback` (Source: blogs.oracle.com). The response contains JSON like `{ "access_token": "eyJraWQiOiIyMDIwXzEi...","refresh_token":"...","scope":["RESTLETS"], ... }` (Source: blogs.oracle.com) (Source: blogs.oracle.com). The client then calls the RESTlet URL with `Authorization: Bearer eyJh...` and NetSuite processes it.

Example: Client Credentials Flow

In a machine-to-machine scenario, the administrator uploads a public certificate (generated by the app) under *Manage Authentication → OAuth 2.0 Client Credentials (M2M) Setup* (Source: docs.oracle.com). A mapping is created linking the Integration Record and role. When the app runs, it requests a client-credentials token similarly from `/services/rest/auth/oauth2/v1/token?grant_type=client_credentials`, authenticating itself with its client ID and the signed certificate. The response is again a JWT access token which can be used to call RESTlets or REST web services, subject to the integration's scopes (e.g. "RESTlets" checkbox) (Source: docs.oracle.com) (Source: docs.oracle.com). Oracle's documentation implies that the token covers exactly the scopes enabled on the app.

In all cases, once an OAuth 2.0 access token is held, RESTlet calls are simply standard HTTPS requests to the account-specific domain with a Bearer header (Source: docs.oracle.com) (Source: docs.oracle.com). For example:

```
GET https://123456.app.netsuite.com/app/site/hosting/restlet.nl?script=5&deploy=1
Authorization: Bearer <access_token>
```

This simplicity (no per-call signing) is one advantage of OAuth2 mentioned in NetSuite's docs (Source: www.brokenrubik.com) (Source: docs.oracle.com).

Comparisons and Implications

The move from TBA to OAuth2 has broad implications for NetSuite integrations. Table 4 highlights key contrasts between TBA (OAuth 1.0) and OAuth 2.0 from a NetSuite perspective:

ASPECT	TBA (OAUTH 1.0)	OAUTH 2.0 (NETSUITE)
Flow	3-step token exchange (with user consent) (Source: docs.oracle.com)	Code Grant or Client Credentials (with refresh tokens)
Credentials	Consumer Key/Secret + Token ID/Secret (4 values) (Source: www.brokenrubik.com)	Client ID/Secret (plus certificate for M2M)
Request Signing	HMAC-SHA256 signature on every request (Source: docs.oracle.com)	No per-request signature (Bearer token header only)
Token Expiration	Tokens do not expire by default (revoked manually)	Access tokens have limited lifetime; refresh tokens can be used
Scopes/Granularity	No concept of scopes (all API allowed if token valid)	Scopes selectable (RESTlets, REST WS, Connect, etc.) (Source: docs.oracle.com)
User Interaction	Required (consent page) unless using IssueToken bypass (Source: docs.oracle.com)	Code Grant requires user login/consent; Client Credentials is headless
Use Cases	Server-server, legacy integrations (Source: www.brokenrubik.com)	Both user-delegated (portal, UI) and server-server scenarios (Source: docs.oracle.com)
Deprecation Status	Deprecated for new integrations after 2027 (Source: docs.oracle.com) (Source: www.houseblend.io)	Preferred for new integrations; actively supported

(Sources: NetSuite documentation [12][30], expert guides [6][72][76].)

From a security standpoint, OAuth 2.0 offers advantages: it natively supports modern auth methods (e.g. SAML or OIDC user login can be used for the initial code grant) (Source: docs.oracle.com), and it produces JWT tokens that carry scopes and other claims (helpful for audit and role enforcement). Houseblend's analysis notes that OAuth2 aligns NetSuite with enterprise compliance standards (NIST, SOC2, PCI), whereas TBA's OAuth1.0 signature process is considered outdated (Source: www.houseblend.io) (Source: www.houseblend.io). Indeed, Oracle's own materials emphasize controlling access via integration records and tokens for auditing: "use integration records to manage RESTlet activity; each record shows RESTlet calls that authenticated by using that record's consumer key" (for TBA) (Source: docs.oracle.com). In the OAuth2 world, the integration record similarly manages which scopes and roles the token can assume.

Practically, migrating integrations from TBA to OAuth2 typically involves creating a new Integration record with OAuth2 enabled, updating authentication code, and possibly involving a one-time user consent or certificate upload. Customers should inventory existing TBA tokens (around 38,000 tokens globally as of early 2024 according to one estimate (Source: www.houseblend.io) and plan to re-authorize as needed. The Houseblend guide provides

extensive migration strategies and emphasizes that by 2027 organizations *must* have replaced TBA with OAuth2 for any new development (Source: www.houseblend.io).

Case Studies and Examples

Several practitioners have documented real-world scenarios of using RESTlets with these auth flows. For example, Oracle SDN engineer Vlastimil Martinek describes a *SuiteApp* onboarding flow where the old method (manually creating a token) is replaced by automated OAuth2 steps (Source: blogs.oracle.com) (Source: blogs.oracle.com). His blog outlines using the authorization code flow to obtain an access token without user copying credentials and then programmatically uploading client credentials certificates via REST calls (see his Section "What is there to automate?" (Source: blogs.oracle.com). In essence, this replicates the "authorization code then client credential" pattern described above.

Likewise, Bundlet's 2026 blog gives a detailed customer onboarding scenario: they package their integration as a SuiteApp and have the customer grant a one-hour administrator-level authorization code (via a public OAuth client with PKCE) solely to perform setup, then switch to a longer-term client credentials connection for day-to-day operations (Source: www.bundlet.com) (Source: www.bundlet.com). They highlight that limiting the code grant token to 1-hour and admin-level scope is actually more secure than simply giving a long-lived admin credential, because the process is user-approved and ephemeral (Source: www.bundlet.com) (Source: www.bundlet.com).

These case studies underscore best practices:

- Use *least privilege* roles for tokens (admin only if absolutely needed and then only short-lived) (Source: www.bundlet.com).
- Automate certificate management where possible to minimize manual errors. (NetSuite 2026.1 introduced a *Certificate Rotation Endpoint* for automated Client Credentials key updates (Source: www.bundlet.com).)
- Provide a smooth UX by bundling OAuth2 steps into the SuiteApp/SDF deployment or portal workflow, rather than manual copy-paste of tokens (Source: blogs.oracle.com) (Source: blogs.oracle.com).

Future Directions and Implications

Looking ahead, Oracle has signaled that by 2027 OAuth 2.0 will be the only supported method for new integrations (Source: docs.oracle.com) (Source: www.houseblend.io). Organizations should therefore design all new RESTlet and SuiteTalk integrations with OAuth2 in mind. This includes mastering the use of Integration Records (with proper scopes) and possibly adopting features like PKCE (required for public clients) and certificate rotation APIs. The scope model may also evolve: currently scopes like "RESTlets" and "REST Web Services" exist, but as NetSuite develops new services (e.g. AI Connector, SuiteQL web services, etc.) additional scopes will be added, as hinted by new checkboxes in the Integration record (Source: docs.oracle.com).

One topic to watch is how OAuth 2.0 tokens will be used with SuiteAnalytics Connect (ODBC), which traditionally has its own token system (SuiteAnalytics tokens). NetSuite has already added an OAuth2 scope "SuiteAnalytics Connect" (Source: docs.oracle.com). Another implication is that any client libraries or tools used by partners must support OAuth2. Many existing NetSuite SDKs are being updated accordingly.

From a security standpoint, the industry trend away from OAuth1.0 (paralleled by Intuit's deprecation in 2020 (Source: www.houseblend.io) suggests that future NetSuite releases may eventually deprecate the older NLAuth basic login method for web services, and possibly even sunset Alloy Single-use Token (NLAuth's short-lived one-time tokens). For now NetSuite still allows username/password token login (NLAuth) for suitelets and RESTlets, but the documentation warns it is "not supported for new RESTlets" (Source: docs.oracle.com). Enterprises should consider eliminating reliance on such older methods as well.

With OAuth2 adoption, toolchain changes will be needed: API gateways and integration platforms must handle OAuth2 flows. Monitoring will shift from looking at static keys to tracking bearer token usage (the integration record execution logs are one place).

Finally, by using scopes and fine-grained tokens, enterprises can better meet compliance standards (principle of least privilege, audit logs, easier credential rotation). Houseblend's report argues this aligns NetSuite with NIST guidelines and PCI/SOC2 guidelines by 2027 (Source: www.houseblend.io). In short, the move to account-specific domains and OAuth 2.0 is a significant modernization that will improve long-term stability and security, but it requires re-educating developers and administrators about the new processes.

Conclusion

NetSuite RESTlet integrations rely on account-specific domains to insulate integrations from infrastructure changes, and these domains form the base of all REST endpoint URLs (Source: docs.oracle.com) (Source: docs.oracle.com). Effective integration requires discovering and using the correct domain for each account, either via NetSuite's dynamic services or the Company URLs page (Source: docs.oracle.com) (Source: docs.oracle.com).

Authentication to these RESTlet endpoints can be done via Token-Based Authentication (OAuth1.0) or OAuth 2.0. TBA has been the legacy choice for server-to-server calls, using signed OAuth1 headers and a multi-step token exchange (Source: docs.oracle.com) (Source: docs.oracle.com). However, Oracle strongly recommends transitioning to OAuth 2.0: it simplifies implementation and aligns with modern security practices (Source: docs.oracle.com) (Source: docs.oracle.com). OAuth 2.0 in NetSuite creates bearer tokens with refresh support and scope restrictions, contrasted to TBA's unsigned, long-lived tokens (Source: www.houseblend.io). NetSuite integration records now allow precise control of OAuth 2.0 scopes (e.g. RESTlets vs REST Web Services) and support both interactive (authorization code) and non-interactive (client credentials) flows (Source: docs.oracle.com) (Source: docs.oracle.com).

By 2027, Oracle will no longer permit new TBA-based integrations (Source: docs.oracle.com) (Source: www.houseblend.io). This deadline means organizations must plan migrations. The good news is that extensive documentation and community expertise now exist, including official guides (Source: docs.oracle.com) (Source: docs.oracle.com) and case-study blogs (Source: www.bundlet.com) (Source: blogs.oracle.com). Integrators should take advantage of these resources to switch their RESTlets and SuiteTalk clients to OAuth 2.0 with account-specific domains well before the cutoff. In doing so, they will achieve more robust, maintainable integrations that meet NetSuite's future roadmap.

References: All facts and procedures above are drawn from NetSuite's official documentation and authoritative technical sources (Source: docs.oracle.com) (Source: docs.oracle.com) (Source: docs.oracle.com) (Source: docs.oracle.com) (Source: blogs.oracle.com), including the NetSuite Help Center, Oracle Support blogs, and expert analyses (Source: www.brokenrubik.com) (Source: timdietrich.me) (Source: www.houseblend.io) (Source: blogs.oracle.com). Each cited section provides step-by-step or conceptual details on the topics discussed. The tables and examples summarize and contrast information explicitly documented in those sources.

Tags: netsuite restlets, account-specific domains, token-based authentication, tba flow, oauth 2.0, netsuite integration, suitescript api