Configuring Salesforce Named Credentials for ServiceNow

OAuth 2.0 & Basic Authentication — a screen-by-screen reference

NexGen Architects | July 2026

A complete, screen-by-screen reference for authenticating the ServiceNow Flow Connector — covering OAuth 2.0 Authorization Code, Client Credentials, and Password grants, plus Basic Authentication. Includes the ServiceNow-side Application Registry setup that most guides omit.

KEY FACTS

Supported methods	4 — Basic Authentication, OAuth 2.0 Authorization Code, OAuth 2.0 Client Credentials, OAuth 2.0 Password (Resource Owner Password Credentials).
Recommended	OAuth 2.0 Authorization Code. It is the only method that avoids storing a reusable ServiceNow password in Salesforce while still giving a durable, refreshable session.
Authorization URL	`https://<instance>.service-now.com/oauth_auth.do`
Token & refresh URL	`https://<instance>.service-now.com/oauth_token.do`
OAuth scope	`useraccount`
Named Credential URL	`https://<instance>.service-now.com` — instance root only, no path, no trailing slash.
Credential storage	Salesforce encrypted Named Credential store. Secrets are never held by the connector or by NexGen Architects.

How the pieces fit together

Salesforce splits callout authentication into two objects, and understanding the split removes most of the confusion in this process:

- The **External Credential** holds the authentication — the protocol (OAuth 2.0 or Basic), the client ID and secret, and one or more Principals. A Principal is an identity that holds a secret or a token.
- The **Named Credential** holds the destination — the ServiceNow instance URL — and points at an External Credential for its authentication.
- A **permission set** grants users access to a Principal. Without it, the callout is rejected regardless of how correct the credentials are.

The connector declares four connection types in its model — `basicAuth`, `oauth2authorizationCode`, `oauth2clientCredentials`, and `oauth2password`. Each maps directly to one of the configurations below.

Choosing an authentication method

All four methods work. They differ sharply in what they expose and what they cost to operate.

Method	Stores a password?	Needs interactive login?	Use it when
OAuth 2.0 Authorization Code *(Recommended)*	No	Yes — once	Always, unless you have a specific reason not to. Salesforce stores only a refresh token, which can be revoked from ServiceNow without changing any password. This is the correct default for production.
OAuth 2.0 Client Credentials	Secret only	No	Fully unattended server-to-server integration with no user context, and you have confirmed your ServiceNow version supports this grant. Verify support on your instance before committing to it.
Basic Authentication	Yes	No	Rapid prototyping, sandbox testing, or an instance where OAuth is genuinely unavailable. Workable in production only with a strict password-rotation process.
OAuth 2.0 Password (ROPC)	Yes	No	Least preferred. It stores a full user password and carries OAuth's configuration overhead, giving you the drawbacks of both. Supported for legacy compatibility; choose Authorization Code or Basic instead.

Part A — Register the OAuth application in ServiceNow

This half happens in ServiceNow and must be done first, because it produces the Client ID and Client Secret that Salesforce needs. It requires the `admin` or `oauth_admin` role. Skip this part entirely if you are using Basic Authentication and go straight to Part C.

1. **Open the Application Registry.** In ServiceNow, navigate to **All** → **System OAuth** → **Application Registry**.
2. **Create a new endpoint for an external client.** Choose **New**, then select **Create an OAuth API endpoint for external clients**. This is the correct option — the other choices connect to a third party, which is the reverse of what you want here.
3. **Name it recognisably.** Use something that will still make sense to a colleague in two years, such as `Salesforce Flow Connector (Production)`. Keep separate registrations per Salesforce org.
4. **Record the Client ID and Client Secret.** ServiceNow generates the Client ID. If you leave the Client Secret blank, ServiceNow generates one — but you must open the record again to read it. Copy both now; the secret is far more awkward to retrieve later.
5. **Set the Redirect URL.** This must exactly match the callback URL Salesforce generates for your External Credential — an inexact match here is the single most common cause of a failed OAuth handshake. The callback URL has the form `https://<your-domain>.my.salesforce.com/services/authcallback/<ExternalCredentialApiName>`; because it embeds the External Credential's API name, create the External Credential in Salesforce first (Part B, steps

1–2), copy the callback URL Salesforce displays, then paste it here. A trailing slash, an `http` instead of `https`, or a sandbox domain used against a production org will all produce `redirect_uri_mismatch`.

6. **Review the token lifespans.** ServiceNow defaults to an access token lifespan of 1,800 seconds (30 minutes) and a refresh token lifespan of 8,640,000 seconds (100 days).
7. **Save the record.**

Part B — Configure OAuth 2.0 in Salesforce

B1. Create the External Credential

1. Go to **Setup** → **Security** → **Named Credentials** → **External Credentials** → **New**.
2. **Set the Label and Name.** The Name (API name) is what appears in the callback URL, so choose it deliberately — for example `ServiceNow_Prod`. Changing it later invalidates the redirect URL you registered in ServiceNow.
3. Set **Authentication Protocol** to **OAuth 2.0**.
4. Set the **Authentication Flow Type** to match your chosen grant:
   - **Browser Flow** — for the Authorization Code grant.
   - **Client Credentials** — for the Client Credentials grant.
5. Enter the ServiceNow endpoints:
   - Identity Provider URL / Authorization Endpoint: `https://<instance>.service-now.com/oauth_auth.do`
   - Token Endpoint: `https://<instance>.service-now.com/oauth_token.do`
   - Scope: `useraccount`
6. Paste the Client ID and Client Secret from the ServiceNow Application Registry.
7. Save, then copy the **Callback URL** that Salesforce now displays — and paste it into the ServiceNow Application Registry's Redirect URL field (Part A, step 5).

B2. Create the Principal

1. On the External Credential, add a Principal. Set the **Sequence Number** to 1 and the **Identity Type** to **Named Principal** — this means all users share one ServiceNow identity, which is what you want for an integration account.
2. Name the Principal something durable, such as `ServiceNow_Integration_User`.

B3. Authorize the Principal

Required for the Authorization Code grant only. Client Credentials needs no interactive step.

1. Open the row-level menu on the Principal and choose **Authenticate**.
2. Sign in to ServiceNow as the integration user — not as yourself.
3. Grant the requested access. Salesforce exchanges the code for an access token and a refresh token, and stores both encrypted.
4. Confirm the Principal's status reads **Authenticated**.

Part C — Configure Basic Authentication in Salesforce

Basic Auth is materially simpler: there is no ServiceNow-side registration at all.

1. Create an External Credential with **Authentication Protocol = Basic Authentication**.
2. Add a Principal with **Identity Type = Named Principal**.
3. Enter the ServiceNow integration user's username and password in the Principal's Authentication Parameters. Salesforce encrypts them at rest and constructs the `Authorization: Basic` header at call time.
4. Save. No authorization step is needed — Basic Auth has no handshake.

Part D — Create the Named Credential

This step is identical for all four authentication methods.

1. Go to **Setup** → **Named Credentials** → **New**.
2. Set the URL to the instance root: `https://<instance>.service-now.com`. No path. No trailing slash.
3. Link the External Credential you created above.
4. Enable **Generate Authorization Header**. This is what causes Salesforce to inject the Authorization header on every callout. Without it, requests leave unauthenticated and ServiceNow answers with an HTML login page.
5. Enable **Allow Formulas in HTTP Header** and **Allow Formulas in HTTP Body** if your flows construct dynamic values.
6. Save.

Part E — Grant access via a permission set

Skipping this step is the most common reason a correctly-configured credential still fails.

1. Create a permission set, for example `ServiceNow Integration Users`.
2. Open **External Credential Principal Access** and enable your Principal.
3. Assign the permission set to every identity that will run a ServiceNow flow — including the Automated Process user if you use record-triggered or scheduled flows.

Verifying the connection

The connector's built-in test issues `GET /api/now/table/sys_user` and treats HTTP 200 as success. To verify credentials independently of Salesforce, reproduce the call directly.

Basic Authentication:

```
curl -u '<user>:<password>' \
  'https://<instance>.service-now.com/api/now/table/sys_user?sysparm_limit=1'
```

OAuth 2.0 — fetch a token, then call the API:

```
curl -X POST 'https://<instance>.service-now.com/oauth_token.do' \
  -d 'grant_type=password' \
  -d 'client_id=<client_id>' \
  -d 'client_secret=<client_secret>' \
  -d 'username=<user>' \
  -d 'password=<password>'

curl -H 'Authorization: Bearer <access_token>' \
  'https://<instance>.service-now.com/api/now/table/sys_user?sysparm_limit=1'
```

A JSON body with a `result` array confirms success. Use this to settle the question of where a failure lives: if `curl` succeeds but the flow fails, the problem is in Salesforce configuration. If `curl` fails too, the problem is in ServiceNow.

Troubleshooting authentication

OAuth and Basic Auth failures, mapped to their causes.

Error	Cause	Fix
`redirect_uri_mismatch`	The Redirect URL in the ServiceNow Application Registry does not exactly match the Callback URL Salesforce generated.	Copy the Callback URL from the Salesforce External Credential and paste it into ServiceNow verbatim. Check for a trailing slash and confirm you are using the right My Domain for the org.
`invalid_client`	Wrong Client ID or Client Secret, or the secret was regenerated in ServiceNow.	Re-copy both from the Application Registry into the External Credential.
`invalid_grant`	The refresh token has expired (the 100-day default), or was revoked in ServiceNow's OAuth token registry.	Re-run **Authenticate** on the Principal. If this recurs on idle integrations, extend the refresh token lifespan.
`unauthorized_client`	The grant type you configured is not enabled for that OAuth entity in ServiceNow — most often when using Client Credentials.	Confirm your ServiceNow version supports the Client Credentials grant and that it is enabled on the OAuth entity. If it is not available, switch to Authorization Code.
HTTP 401 on every call, having previously worked	Basic Auth: the ServiceNow password expired. OAuth: the refresh token expired or was revoked.	Update the Principal's password, or re-authenticate. Then address the underlying expiry policy so it does not recur.
An HTML login page is returned instead of JSON	Generate Authorization Header is switched off on the Named Credential, so requests are leaving unauthenticated — or SSO is intercepting the integration user.	Enable **Generate Authorization Header**. If SSO is enforced instance-wide, exempt the integration account or use OAuth.
Insufficient access when the flow runs, though Debug works	The running user lacks External Credential Principal Access.	Assign the permission set from Part E to the flow's running user, including Automated Process.

Frequently asked questions

Which authentication method should I use for the ServiceNow Flow Connector in production?

Use OAuth 2.0 Authorization Code with a Named Principal. It is the only supported method that avoids storing a reusable ServiceNow password in Salesforce, and it lets you revoke the integration's access from ServiceNow's OAuth token registry without changing any password or disturbing other systems that use the same account. Basic Authentication is acceptable for sandboxes and prototypes. The Password (ROPC) grant is supported for legacy compatibility but is the least preferred option, because it stores a full user password while also carrying OAuth's configuration overhead.

What are the ServiceNow OAuth 2.0 endpoint URLs for a Salesforce Named Credential?

ServiceNow uses `/oauth_auth.do` for authorization and `/oauth_token.do` for both token issuance and refresh. In full, against your instance: the authorization URL is `https://<instance>.service-now.com/oauth_auth.do`, and the token and refresh URL is `https://<instance>.service-now.com/oauth_token.do`. The OAuth scope is `useraccount`. The Named Credential's own URL is the bare instance root, `https://<instance>.service-now.com`, with no path.

What is the correct callback URL to register in the ServiceNow Application Registry?

The callback URL is `https://<your-domain>.my.salesforce.com/services/authcallback/<ExternalCredentialApiName>`, and it must match character-for-character. Because it embeds the External Credential's API name, create the External Credential in Salesforce first, copy the callback URL that Salesforce then displays, and paste that exact string into the Redirect URL field of the ServiceNow Application Registry entry. A trailing slash or a mismatched My Domain produces a `redirect_uri_mismatch` error.

Why does my ServiceNow connection suddenly return 401 after working for months?

The two usual causes are an expired ServiceNow password (Basic Auth) or an expired refresh token (OAuth). ServiceNow's default refresh token lifespan is 8,640,000 seconds — 100 days — so an integration that goes idle longer than that will find its token dead even though nothing was changed. Re-run **Authenticate** on the External Credential Principal to recover. To prevent recurrence, either extend the refresh token lifespan on the OAuth entity or schedule a periodic keep-alive call. For Basic Auth, exempt the integration account from password expiry.

My flow works in Debug but fails with an authorization error when it runs for real. Why?

The flow's running user has not been granted External Credential Principal Access. In Debug, the flow runs as you — an administrator who typically has access. In production, record-triggered and scheduled flows run as the Automated Process user or the triggering user. Create a permission set, enable your Principal under External Credential Principal Access, and assign it to the flow's actual running user. This is the most frequent configuration error in the entire setup.

Should I choose Named Principal or Per User identity for the ServiceNow connection?

Choose Named Principal for virtually all integrations. Named Principal means all Salesforce users share a single ServiceNow integration identity, which is what makes automated flows possible. Per-User identity requires every individual user to complete an interactive OAuth browser login before they can run a flow — which record-triggered and scheduled flows can never satisfy, because they run as the Automated Process user, who cannot complete a browser login.

Can Salesforce store my ServiceNow password securely with Basic Authentication?

Yes — Salesforce encrypts External Credential Principal secrets at rest, and they are not readable from flows, debug logs, or the API. The residual risk of Basic Auth is not storage; it is that a password is replayable and shared. Anyone who obtains it has full use of that ServiceNow account, and revoking it means a password change that breaks every other system using the same account. That is why OAuth 2.0 Authorization Code is recommended for production even though Basic Auth's storage is itself secure.

Does the ServiceNow Flow Connector support the OAuth 2.0 Client Credentials grant?

Yes — the connector declares a `oauth2clientCredentials` connection type, obtaining tokens from `https://<instance>.service-now.com/oauth_token.do` with the `useraccount` scope. Before committing to it, confirm that your ServiceNow instance version supports and has enabled the Client Credentials grant on the OAuth entity, as availability varies by release and it may require additional configuration. If you see `unauthorized_client`, the grant is not enabled — use the Authorization Code grant instead.