FLIP-XXX : Remote Artifact Fetch for Session-Mode Job Submission

Status	discussion
Discussion thread	https://lists.apache.org/thread/qsqxkn8kclzqjwrqly1hm5tv9thxtcmd
Vote thread	
JIRA	
Release	

Please keep the discussion on the mailing list rather than commenting here.

Motivation

Flink lets a cluster fetch its own job jar from a remote location instead of requiring a client to hand it a local file.

Given a URI, `org.apache.flink.client.program.artifact.ArtifactFetchManager` resolves the artifact through Flink's `FileSystem` abstraction. Any scheme with a registered filesystem plugin (HTTP, S3, HDFS, GCS, ...) works, and use the cluster's own configured credentials.

Today, this capability is wired into two call sites:

- `StandaloneApplicationClusterEntryPoint`
- `KubernetesApplicationClusterEntrypoint`

Both of these run during Application Mode cluster start-up. There is no equivalent route for Session Mode.

A session cluster is already running when a job is submitted to it. Flink's REST submission path (POST `/jars/upload` followed by POST `/jars/{jarid}/run`, implemented by `JarHandlerUtils` / `JarRunHandler`) was built around a client that already holds the jar's bytes and uploads them directly. There is no option to instead give the running cluster a URI and let it fetch the artifact itself.

This can be seen in the behaviour of the Flink Kubernetes Operator. When a `FlinkSessionJob` names a `jarURI`, the Operator's `AbstractFlinkService.uploadJar` downloads the jar from that URI into the Operator's own pod, then re-uploads the resulting bytes to the session cluster's `/jars/upload` endpoint, before finally calling `/jars/{jarid}/run`. Every session job submission that references a remote jar needs two copies of the artifact (one in the Operator, one in the Job Manager). It means the Operator must be granted read access to every artifact storage source any Flink session job in every session cluster might reference, a broader credential footprint than the job itself needs, and one that scales with the number of distinct artifact sources across every cluster the Operator manages.

Application Mode does not have this problem, because the cluster can fetch its own artifact with its own, appropriately-scoped credentials. Session Mode feels like a more inconsistent case: the capability to fetch a remote artifact directly already exists in Flink, it is just not reachable from the submission path.

This FLIP proposes extending Flink's existing remote-artifact-fetch capability to session-mode job submission, so that a job can be submitted by URI, with the Job Manager performing the fetch itself, the same way an Application Mode cluster already does at start-up.

The URI can be provided as a configuration key on the existing job submission request, rather than as a new field or endpoint.

Public Interfaces

- **New** `ConfigOption`: **`user.artifacts.job-jar`** (string, no default) added to `ArtifactFetchOptions`
    - read from the Configuration that `JarRunRequestBody` already carries on every `POST /jars/{jarid}/run` request
    - `flinkConfiguration`-shaped request configuration, merged with the cluster's own configuration, is the same mechanism `JarHandlerUtils` uses on every submission today
    - No new field is added to `JarRunRequestBody` and no new REST endpoint is introduced. The key is a new, documented, recognized entry in a configuration map that the request already carries
    - When `user.artifacts.job-jar` is present in the submission's configuration, the handler fetches the named artifact and uses it as the job jar in place of whatever the `{jarid}` path segment resolved to.
        - This means a "real" jarid is not required for this. A client could skip the upload step, and use any string (e.g. "remote") for the `{jarid}` path parameter.
        - This is also compatible with submissions that always populate `{jarid}` with something. A client could upload a placeholder/empty jar first, and obtain a "real" jarid to call `/run` on - and then specify the real jar in the request config. Although not required, this could simplify adoption.

    - The URL for the jar must remain accessible for as long as the job may be redeployed.
    - `user.artifacts.*` was chosen as it is the config-set that is already used to configure this fetcher, so this keeps it besides other related settings.
- **New cluster-level configuration option**: **`web.submit.jar-uri.enable`** (boolean, default `false`) added to `WebOptions`
    - When `false` a request with configuration containing the jar-URI config option is rejected with `400 Bad Request`
    - This flag exists because accepting a caller-supplied URI to be fetched with the cluster's own credentials is a different trust model from "upload the bytes you already have". This change in trust model should be an explicit, deliberate opt-in by whoever operates the cluster.
    - `web.submit.jar-uri.enable` was chosen to sit this alongside the existing `web.submit.enable` option
- **New** `ConfigOption`: **`user.artifacts.fetch-timeout`** (Duration, default 5 minutes) added to `ArtifactFetchOptions`
    - read from the session cluster's configuration, before the request body's configuration is merged in (the same trust boundary as `web.submit.jar-uri.enable` for the same reason)
    - This bounds how long the handler should wait for a fetch before the request is failed with a `400 Bad Request`

Endpoints affected

`JarHandlerContext.fromRequest` is shared by three endpoints, so these endpoints would honour the new config options:

- `POST /jars/{jarid}/run`
- `POST /jars/{jarid}/run-application`
- `POST /jars/{jarid}/plan`

No changes to `JarRunRequestBody`, any other REST request/response shape, the binary log format, the network protocol between Job Manager and Task Manager, monitoring/metrics, or command-line tool arguments. Existing `ArtifactFetchOptions` configuration keys (`user.artifacts.raw-http-enabled`, `user.artifacts.http-headers`, `user.artifacts.base-dir`, `user.artifacts.artifact-list`) can be reused as-is, and are read from the cluster's configuration only. The only new configuration introduced for the fetch mechanism itself is `user.artifacts.fetch-timeout`, the rest is used to gate whether session-mode submission may use it, and for naming the artifact.

No existing interface is removed or changed in an incompatible way. This is a purely additive proposal. It requires no coordinated change in any existing REST client (including the Flink Kubernetes Operator) because the mechanism is a config key rather than a request-body field.

Proposed Changes

Changes

1. When a jar-submission request's configuration includes `user.artifacts.job-jar` and the cluster's `web.submit.jar-uri.enable` is true, the handler fetches the named artifact via `ArtifactFetchManager`, in the same way that `StandaloneApplicationClusterEntryPoint` / `KubernetesApplicationClusterEntrypoint` already does for Application Mode start-up.
2. The fetched file is then used as the job jar for the rest of the existing submission path (including `PackagedProgram` construction, and adding the program's jars to `PipelineOptions.JARS` so they are distributed to every Task Manager) and invoking the program exactly as today.
    a. This only changes where the bytes for `jarFile` comes from. No new classloading behavior is introduced.
    b. The `{jarid}` path segment is still resolved as it is today, but the jar it names is discarded in favour of the fetched one. This means this will work whether `{jarid}` identifies a previously uploaded placeholder / empty jar or if the jarid string is itself a placeholder (e.g. "remote").
3. The gate is read from the cluster's configuration, before any merge. `JarHandlerUtils` applies the request's configuration over the top of the cluster's configuration, so the gate should be read from the cluster configuration before this. This is similar to how `PIPELINE_FIXED_JOB_ID` is re-stamped after the merge to prevent callers overriding it.
4. The fetch uses the cluster's configuration, with only the submitted jar URI added from the caller's request. The submitter can choose the artifact to fetch, and the cluster specifies how it is fetched. (This means that submitters cannot override options like `user.artifacts.base-dir` or `user.artifacts.http-headers`). (*More about **Credentials** below*)
5. The fetch target directory is pinned for each submission, this means fetchArtifact can resolve quickly if the file already exists. An independent sub-directory for each fetch will prevent the risk of collisions in a long-lived multi-tenant session cluster Job Manager.
6. The fetched artifact's temporary target directory is deleted once it is no longer required:
    a. Every failure path deletes it, including when the fetch throws an error before the job is built
    b. Successful calls to `/run` and `/plan` delete it once the request is answered (both consume the jar within the request lifecycle - uploading to the blob server by the time the call is responded to)
    c. Successful calls to `/run-application` need to leave the directory in place beyond the one call, for the duration of the application, deleted once the application reaches a terminal state

7. Failed fetches for jars will produce a `RestHandlerException` naming the URI, with an HTTP-400, and not create the job. (Failing to create the jar file - e.g. disk full, unwritable directory - would produce an HTTP-500 Internal Server Error, and still not create the job)
8. `user.artifacts.job-jar` accepts a single artifact for the initial proposal, matching the job's single entry jar. (*`ArtifactFetchManager.fetchArtifacts` already accepts an array of URIs and already distinguishes the job jar from additional dependency artifacts - the mechanism Application Mode uses when both a job jar and extra dependencies are supplied at start-up. Flink already has a list-typed configuration key: `user.artifacts.artifact-list`. So extending this to a list, either by changing `user.artifacts.job-jar` to a list type, or by having it interact with `user.artifacts.artifact-list` directly would be a natural follow-up but left out of this FLIP to keep the change small.*)

Unchanged

1. When `user.artifacts.job-jar` is absent from the submission's configuration, or the feature flag is `false`, behaviour is identical to today. The existing local-jar-id path is left untouched, so today's primary use case (a client uploading a jar it holds locally, including the Flink Web UI and `bin/flink run`) is unaffected.
2. Authentication / authorization for the fetch is delegated to the filesystem plugin resolving the URI's scheme, exactly as it already is for Application Mode. No new credential-handling code is introduced. (The credentials used will be the cluster's, configured by the session cluster operator, and per-submission credentials are deliberately not supported. *More about **Credentials** below.*)

Credentials and the submitted URI

The only credentials that should be used to fetch jars are the cluster's own, configured by the owner of the session cluster. Embedding credentials in the submitted jar URI as a workaround for this should be discouraged and unsupported. This includes URLs with a userinfo component (e.g. `https://user:pass@host/flinkapp.jar`) and URLs including an authorization token in a query string. We will not actively try to block such URIs as it would be difficult to do this consistently and reliably.

Submitted URIs will not be treated as sensitive. They will not be masked or added to global configuration's sensitive key list. They may be logged by the Job Manager, and will remain in the submission's configuration after the fetch has completed. (*Note that the jar URI will not reach `/jobs/{jobid}/config` which only exposes pipeline global-job-parameters, so it will not appear in the web UI job configuration tab*)

Handling fetch timeouts

When an application mode cluster is fetching a job jar, this doesn't block anything else as the cluster is not supporting any other jobs during this startup time. For a session mode cluster, a long or slow download needs to be managed in parallel to the Job Manager's other responsibilities.

The new `user.artifacts.fetch-timeout` (Duration, default 5 minutes) option is proposed to be used in two ways:

- Control how long the `HttpArtifactFetcher` can spend downloading a jar. (There was no timeout before, so this is a change to the previously unbounded behaviour.) After this time, the fetch will be stopped at the socket level.
- Control how long the request handler will wait for ArtifactFetchManager before returning a response to the submitter. (This means a REST request is guaranteed to receive at least a `400 Bad Request` and stop waiting after this duration.)

For non-HTTP/HTTPS URI schemes (e.g. `s3://`, `gs://`, `abfs://`, etc.) this only determines how long the request handler will wait for the fetch, not how long the fetch will be allowed to continue. Controlling the timeout for filesystem fetches will be deferred to the filesystem plugin, and users will need to configure this accordingly (e.g. `fs.s3a.connection.timeout`, `gs.http.read-timeout`, etc.)

Compatibility, Deprecation, and Migration Plan

- **No impact on existing users.**
    - With the new configuration option left at its default (`false`), the REST API's behavior, request/response shapes, and existing endpoints are unchanged. A cluster that never enables the option is indistinguishable, from any existing client's point of view, from a cluster running today's code. (*A small exception to this is the introduction of the default fetch timeout. Users who are used to fetches taking longer than 5 minutes would need to modify this config.*)
- **No deprecation.**
    - Nothing existing is removed, renamed, or behaviorally altered. The local-jar-upload-then-run flow remains the default and the only flow when the new option is disabled.
- **No migration tooling is required.**
    - Adopting this feature is opt-in per cluster (set the configuration flag) and per request (set `user.artifacts.job-jar`); there is no state to migrate. Because the mechanism is a configuration key on an existing request rather than a new field or endpoint, no existing REST client needs code changes to become capable of using it.
- **No timeline for removing older behavior** is proposed
    - because no older behavior is being replaced
    - the existing upload-and-run flow remains a first-class, permanently supported way to submit a job to a session cluster

Operators enabling `web.submit.jar-uri.enable` should be aware that doing so means any caller able to reach the REST submission endpoint can cause the Job Manager to fetch an arbitrary URI reachable by the cluster's configured filesystem plugins, using the cluster's own credentials.

Operators should also be aware of the disk usage impact of requests. A `/run` or `/plan` submission would only hold the fetched jar for the duration of the request, so a session cluster supporting multiple long-lived applications submitted in this way will need to size `user.artifacts.base-dir` to support one jar per in-flight request. A `/run-application` submission keeps the jar for the lifetime of the application, so a session cluster supporting multiple long-lived applications submitted in this way will need to size user.artifacts.base-dir for one jar per running application.

This is the same trust boundary that already exists around who may call the REST submission API at all (today's `/jars/upload` endpoint already lets any such caller run arbitrary code on the cluster). The proposed new option changes what that already-trusted caller can point the fetch at, not who is trusted to submit jobs in the first place. This should be called out prominently in the configuration option's documentation.

Test Plan

- **Unit tests** (`flink-runtime-web`) for all affected endpoints (`/run`, `/run-application`, `/plan`):
    - Jar URI present, flag enabled, successful fetch, job submitted using the fetched jar; existing local-jar-id path untouched.
    - Jar URI present, flag disabled, request rejected, 400 Bad Request, no fetch attempted.
    - Jar URI absent, behavior identical to current tests for these endpoints.
    - Fetch throws (unreachable URI, filesystem plugin error, bad credential, missing file), request fails cleanly, no job is created, and the temporary directory created for the fetch attempt is verified deleted
    - Successful fetch, successful submission, temporary directory verified deleted after the submission completes (after job completes for /run-application)
    - Submitter cannot override fetch configuration
    - Two submissions for different URIs with paths that end in the same filename do not collide
- **Integration tests** against a session cluster:
    - Submits a job by URI against a local HTTP file server; confirm the job reaches `RUNNING` / `FINISHED`, and confirm that the program's own class is loaded by the ordinary per-job user-code classloader
    - Confirm the fetch uses the cluster's configured credentials, and that a deliberately wrong credential fails the submission cleanly without creating a job.
    - Submit a job with today's existing upload-then-run flow against a cluster with the new option enabled, to confirm the two paths coexist without interference.
    - Cover each program style Flink's client already supports through this submission path (the DataStream API, the Table API, and, on the Flink versions where they exist, the legacy DataSet API and DataStream V2), to confirm the change is submission-path-level and needs no per-API accommodation — the fetched-artifact

substitution happens before any user code runs, so it should be invisible to which client API the program uses.

- **Compatibility test**
    - Confirm a cluster with the option left disabled passes the full existing REST submission test suite unmodified.

Rejected Alternatives

Leave this to Application Mode only.

Session Mode exists specifically to amortize cluster start-up cost across many jobs. Requiring a dedicated, freshly-started cluster per job to get remote-artifact fetch defeats that purpose for any use case that wants both a long-lived cluster and jobs referenced by URI rather than by local upload.

Leave artifact resolution to the submitting client/orchestrator, as today.

This is the status quo so it obviously works. It forces every orchestrator submitting to a session cluster to hold read credentials for every artifact store any job might reference, and to pay for a full fetch-then-reupload of every artifact on every submission and redeploy. The Flink Kubernetes Operator's own uploadJar implementation is a concrete, already-shipping example of this cost being paid today.

Use pipeline.jars and allow this to point at a remote URI

This would need expanding as currently the value is discarded before it could be used, and is currently treated as an URL that would not support filesystem schemes like s3://. Updating how this config option is handled would be more disruptive than introducing a new one.

Build a new artifact-registry subsystem inside Flink

e.g. a jar catalog/registry service with its own storage and resolution semantics. Rejected as unnecessary scope increase.

Enable remote-jar submission unconditionally, with no configuration gate.

Rejected because it silently changes the trust model of the REST submission API. A caller-supplied URI fetched with the cluster's own credentials is a materially larger capability than "upload bytes you already hold," and should be something a cluster operator turns on deliberately, not something that ships as an always-on default change in behaviour.

Allow per-submission fetch credentials

Rejected for an initial proposal to reduce risk

Support a list of artifact URIs from the start.

Deferred. The underlying mechanism (`ArtifactFetchManager.fetchArtifacts(String[])`, `user.artifacts.artifact-list`) already supports it, so extending `pipeline.jar-uri` to a list is possible follow-up. It is left out of this FLIP's initial scope so the first version stays small.

Respond to submission requests immediately, and fetch and run jars asynchronously

This would be a change in behaviour and meaning to the endpoints as they are today. (This is especially true for `/plan` where the response is the plan generated from the fetched jar's Job Graph). It would introduce the need to have a way of learning the outcome of a fetch after the response was sent, likely a new endpoint. This increases the size of the change that would be needed.

A new, typed jarUri field on JarRunRequestBody, with a companion POST /jars/run endpoint that needs no {jarid} at all.

This would result in a clearer self-documenting API, without requiring clients to submit a jarid that will be ignored. It would be a more natural fit for a value that names what job this is, not how Flink is configured.

On balance, requiring a new endpoint will increase the effort needed to adopt this, compared with supporting a configuration key available for all clients already. Given that the goal is to remove work from orchestration layers such as the Flink Kubernetes Operator, requiring a coordinated update feels like it works against that goal.

This could be addressed as a future piece of work, combined with the point above, as the new endpoint could be designed to respond immediately, without waiting for the fetch to complete.

Known limitations

A run-application jar will not be reclaimed if the Job Manager goes away first

A jar fetched for `/run-application` is owned by the application, not the request. The jar is only released once the application reaches a terminal state. If the application stops without reaching a terminal state (such as through cluster shutdown or Job Manager failover) then nothing will delete the jar.

This is only a problem where user.artifacts.base-dir points at persistent storage that will outlive the Job Manager (unlike the default value for this directory, pointing at the JVM's temporary directory). Users who use a non-default `user.artifacts.base-dir` value that specifies a persistent volume **and** use `/run-application` to submit applications will need to clean up jars themselves in the event of Job Manager shutdown or failover.