Upgrading S3A to SDKV2

This document explores concerns around upgrading S3A to SDK V2 and possible solutions.

Hadoop JIRA: https://issues.apache.org/jira/browse/HADOOP-18073

Why should we upgrade?

- Moving to SDK V2 will provide performance benefits. For example, the transfer manager for SDKV2 is built using java bindings of the AWS Common Runtime S3 client (CRT). CRT is a set of packages written in C, designed for maximising performance when interacting with AWS services such as S3.
- New features such as additional checksum algorithms which S3A will benefit from are not available in SDKV1.

What are the major concerns around the upgrade?

Credential Providers

What's changed?

1. **Interface change**: `com.amazonaws.auth.AWSCredentialsProvider` has been replaced by `software.amazon.awssdk.auth.credentials.AwsCredentialsProvider`.
2. **Credential provider class changes**: the package and class names of credential providers have changed

How does S3A use credential providers?

- S3A implements custom credential providers, for example see the class TemporaryAWSCredentialsProvider. These implement the V1 credential class `AWSCredentialsProvider`.
- S3A allows users to configure credential providers they want to use with the property `fs.s3a.aws.credentials.provider`. This a list of comma separated classes to use. **Users can refer to credential providers in SDKV1 or to custom providers implementing the SDKV1 API**, as well as the ones Hadoop provides. For example a possible config could look like:

```
org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider, // use a hadoop provider
com.amazonaws.auth.InstanceProfileCredentialsProvider // use a SDKV1 provider
```

- In delegation tokens. Delegation tokens are encrypted byte arrays which hold credential information, eg: `fs.s3a.access.key`, `fs.s3a.secret.key` and are passed around the cluster by the resource manager to allow nodes to authenticate. Their implementations may rely on V1 credential provider interfaces and classes.

How will upgrading to V2 impact these uses?

- Any custom credential providers that implements `AWSCredentialsProvider` will need to be updated. While it's straightforward to update providers implemented in S3A, users can also implement their own providers and reference those in the configuration `fs.s3a.aws.credentials.provider`. Upgrading will mean **user's custom providers implementing V1's `AWSCredentialsProvider` will break**.
- As mentioned above, users can also refer to SDK V1 classes in `fs.s3a.aws.credentials.provider`. Upgrading means these classes will no longer exist (as class names and paths have changed), and thus **references to these SDK V1 credential providers will break**.
- Delegations tokens support allows users to implement custom "binding" classes responsible for extracting credential information from the tokens, and returning credential providers making use of the token's content. These binding

> classes may use custom credential providers implemented in S3A, or in SDKV1, or custom user implemented providers. For example, see the class SessionTokenBinding and how to implement a custom binding class. Again, as we do not have access to these binding classes, we cannot update them.

The problem is then: *S3A provides multiple extension points which are out of our control, and so upgrading the SDK becomes a breaking change.*

How can we solve this?

We can implement an adapter which allows you to use V1 credential providers with SDKV2. This would looking something like:

The `V1V2AwsCredentialProviderAdapter` class:

```
/**
 * Interface for a two-way adapter from the AWS SDK V1 {@link AWSCredentialsProvider}
 * interface and the AWS SDK V2 {@link AwsCredentialsProvider} interface.
 */
public interface V1V2AwsCredentialProviderAdapter
    extends AWSCredentialsProvider, AwsCredentialsProvider {

    /**
     * Creates a two-way adapter from a V1 {@link AWSCredentialsProvider} interface.
     *
     * @param v1CredentialsProvider V1 credentials provider.
     * @return Two-way credential provider adapter.
     */
    static V1V2AwsCredentialProviderAdapter adapt(AWSCredentialsProvider v1Credentials
        return V1ToV2AwsCredentialProviderAdapter.create(v1CredentialsProvider);
    }

  }
```

The `V1ToV2AwsCredentialProviderAdapter` class:

```
/**
 * Adapts a V1 {@link AWSCredentialsProvider} to the V2 {@link AwsCredentialsProvider}
 * Implements both interfaces so can be used with either the V1 or V2 AWS SDK.
 */
final class V1ToV2AwsCredentialProviderAdapter implements V1V2AwsCredentialProviderAda

    private final AWSCredentialsProvider v1CredentialsProvider;

    private V1ToV2AwsCredentialProviderAdapter(AWSCredentialsProvider v1CredentialsPro
        this.v1CredentialsProvider = v1CredentialsProvider;
    }

    @Override
    public AwsCredentials resolveCredentials() {
        AWSCredentials toAdapt = v1CredentialsProvider.getCredentials();
        if (toAdapt instanceof AWSSessionCredentials) {
            return AwsSessionCredentials.create(toAdapt.getAWSAccessKeyId(),
                                                toAdapt.getAWSSecretKey(),
```

```
                                                ((AWSSessionCredentials) toAdapt).getS
        } else {
            return AwsBasicCredentials.create(toAdapt.getAWSAccessKeyId(), toAdapt.get
        }
    }

    @Override
    public AWSCredentials getCredentials() {
        return v1CredentialsProvider.getCredentials();
    }

    @Override
    public void refresh() {
        v1CredentialsProvider.refresh();
    }

    /**
     * @param v1CredentialsProvider V1 credential provider to adapt.
     * @return A new instance of the credentials provider adapter.
     */
    static V1ToV2AwsCredentialProviderAdapter create(AWSCredentialsProvider v1Credenti
        return new V1ToV2AwsCredentialProviderAdapter(v1CredentialsProvider);
    }
}
```

When configuring the S3 V2 client in S3A, this adapter can be used as:

```
s3ClientBuilder.credentialsProvider(
    V1V2AwsCredentialProviderAdapter.adapt(parameters.getCredentialSet()));
```

`parameters.getCredentialSet()` returns V1 credential providers configured to be used.

`V1V2AwsCredentialProviderAdapter.adapt()` creates the adapted credential provider to be used by the SDKV2. When SDKV2 calls `resolveCredentials()`, the adapter will call `getCredentials()` on the V1 credential provider and convert those credentials to an instance of the V2 `AwsCredentials` class and return.

Using this adapter will prevent this from becoming a breaking change, as users can continue to use their existing credential providers and configuration, and the adapter will take care of the rest.

What about directly referenced SDKV1 credential provider classes?

As mentioned above, users can directly reference SDKV1 classes in `fs.s3a.aws.credentials.provider`. S3A will use the adapter class to convert the provider to a V2-compatible interface seamlessly.

S3A currently uses the `aws-java-sdk-bundle` for SDKV1. As part of the upgrade, we are also adding a dependency to the SDKV2 bundle. We can remove the bundle dependency for V1, and add in a dependency for `aws-java-sdk-core`. This will ensure we continue to have access to V1 credential providers.

Signers

What's changed?

1. **Interface change**: `com.amazonaws.auth.Signer` has been replaced by `software.amazon.awssdk.core.signer.Signer`
2. **Some signers are removed in SDKV2, notably SigV2.**

How does S3A use signers?

- S3A allows users to configure which signer to use via the property `fs.s3a.signing-algorithm`. The default signing algorithm is SigV4 and all new AWS regions since 2013 have supported only SigV4. However, not all third-party object stores support this algorithm.
- Users can also implement and use custom signers. However, this is not that common.

How will upgrading to V2 impact these uses?

- Some signers (SigV2, AWS3Signer) are no longer supported. This will impact users who rely on these signers to interact with third party stores.
- Custom signers implementing the V1 signer interface will no longer work.

How can we solve this?

- We can implement our own signers for the ones that are not supported. There is also a related GitHub issue aws/aws-sdk-java-v2#1024 for SigV2 support.
- We would also like to explore implementing an adapter for signers, similar to the one we have for credentials.

Cross region support

What's changed?

Cross-region access is no longer supported. A client may now only access buckets in the region with which the client has been configured.

How does S3A use regions?

- You can configure the region using `fs.s3a.endpoint.region` and the endpoint with `fs.s3a.endpoint`. If the endpoint and region isn't specified, S3A will set the region to `us-east-1` and the SDK will use endpoint `s3.amazonaws.com`.
- When configuring the client, S3A sets `withForceGlobalBucketAccessEnabled(true)`, so even if you set the region as `us-west-2` or endpoint as `s3.us-east-2.amazonaws.com`, you could still access a bucket in a different region.

How will upgrading to V2 impact these uses?

Since `withForceGlobalBucketAccessEnabled(true)` is always set, it didn't matter if the bucket you were accessing was not in the same region as the endpoint being used, or the region configured. With SDKV2, accessing a bucket outside the configured region will not work.

How can we solve this?

Upon file system initialisation, make a call to `getBucketLocation` and configure client accordingly if no region was provided.

GitHub issue aws/aws-sdk-java-v2#52 tracks potential support for this feature, however we do not believe this will be introduced in the near future.