Red Hat Data Grid 8.5

Data Grid セキュリティーガイド

Data Grid セキュリティーの有効化および設定

Last Updated: 2024-08-13

Red Hat Data Grid 8.5 Data Grid セキュリティーガイド

Data Grid セキュリティーの有効化および設定

法律上の通知

概要

ネットワークから Data Grid デプロイメントを保護します。許可されたユーザーのデータアクセスを制限します。

目次

RED HAT DATA GRID

Data Grid は、高性能の分散型インメモリーデータストアです。

スキーマレスデータ構造
: さまざまなオブジェクトをキーと値のペアとして格納する柔軟性があります。

グリッドベースのデータストレージ
: クラスター間でデータを分散および複製するように設計されています。

エラスティックスケーリング
: サービスを中断することなく、ノードの数を動的に調整して要件を満たします。

データの相互運用性
: さまざまなエンドポイントからグリッド内のデータを保存、取得、およびクエリーします。

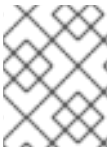
DATA GRID のドキュメント

Data Grid のドキュメントは、Red Hat カスタマーポータルで入手できます。

- Data Grid 8.5 ドキュメント
- Data Grid 8.5 コンポーネントの詳細
- Data Grid 8.5 でサポートされる構成
- Data Grid 8 機能のサポート
- Data Grid で非推奨の機能

DATA GRID のダウンロード

Red Hat カスタマーポータルで Data Grid Software Downloads にアクセスします。



注記

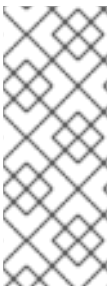
Data Grid ソフトウェアにアクセスしてダウンロードするには、Red Hat アカウントが必要です。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、用語の置き換えは、今後の複数のリリースにわたって段階的に実施されます。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1章 ロールベースアクセス制御によるセキュリティー承認

ロールベースアクセス制御 (RBAC) 機能では、さまざまなパーミッションレベルを使用して、Data Grid とのユーザーの対話を制限します。



注記

リモートキャッシュまたは組み込みキャッシュに固有のユーザーの作成と承認の設定に関する詳細は、以下を参照してください。

- Configuring user roles and permissions with Data Grid Server
- Programmatically configuring user roles and permissions

1.1. DATA GRID のユーザーロールと権限

Data Grid には、キャッシュや Data Grid のリソースにアクセスするための権限をユーザーに提供するロールがいくつかあります。

ロール	パーミッション	説明
admin	ALL	Cache Manager ライフサイクルの制御など、すべてのパーミッションを持つスーパーユーザー。
deployer	ALL_READ、ALL_WRITE、LISTEN、EXEC、MONITOR、CREATE	**application** パーミッションに加えて、Data Grid リソースを作成および削除できます。
application	ALL_READ、ALL_WRITE、LISTEN、EXEC、MONITOR	**observer** パーミッションに加え、Data Grid リソースへの読み取りおよび書き込みアクセスがあります。また、イベントをリッスンし、サーバータスクおよびスクリプトを実行することもできます。
observer	ALL_READ、MONITOR	**monitor** パーミッションに加え、Data Grid リソースへの読み取りアクセスがあります。
monitor	MONITOR	JMX および **metrics** エンドポイント経由で統計を表示できます。

関連情報

- org.infinispan.security.AuthorizationPermission Enum
- Data Grid 設定スキーマ参照

1.1.1. パーミッション

ユーザーロールは、さまざまなアクセスレベルを持つパーミッションのセットです。

表1.1 Cache Manager のパーミッション

Permission	機能	説明
設定	**defineConfiguration**	新しいキャッシュ設定を定義します。
LISTEN	**addListener**	Cache Manager に対してリスナーを登録します。
ライフサイクル	**stop**	Cache Manager を停止します。
CREATE	**createCache**, **removeCache**	キャッシュ、カウンター、スキーマ、スクリプトなどのコンテナーリソースを作成および削除することができます。
MONITOR	**getStats**	JMX 統計および **metrics** エンドポイントへのアクセスを許可します。
ALL	-	すべての Cache Manager のアクセス許可が含まれます。

表1.2 キャッシュ権限

Permission	機能	説明
READ	**get**, **contains**	キャッシュからエントリーを取得します。
WRITE	**put**, **putIfAbsent**, **replace**, **remove**, **evict**	キャッシュ内のデータの書き込み、置換、削除、エビクト。
EXEC	**distexec**, **streams**	キャッシュに対するコードの実行を許可します。
LISTEN	**addListener**	キャッシュに対してリスナーを登録します。
BULK_READ	**keySet**, **values**, **entrySet**, **query**	一括取得操作を実行します。
BULK_WRITE	**clear**, **putAll**	一括書き込み操作を実行します。
ライフサイクル	**start**, **stop**	キャッシュを開始および停止します。

ADMIN	**getVersion**, **addInterceptor***, **removeInterceptor**, **getInterceptorChain**, **getEvictionManager**, **getComponentRegistry**, **getDistributionManager**, **getAuthorizationManager**, **evict**, **getRpcManager**, **getCacheConfiguration**, **getCacheManager**, **getInvocationContextContainer**, **setAvailability**, **getDataContainer**, **getStats**, **getXAResource**	基盤となるコンポーネントと内部構造へのアクセスを許可します。
MONITOR	**getStats**	JMX 統計および **metrics** エンドポイントへのアクセスを許可します。
ALL	-	すべてのキャッシュパーミッションが含まれます。
ALL_READ	-	READ パーミッションと BULK_READ パーミッションを組み合わせます。
ALL_WRITE	-	WRITE パーミッションと BULK_WRITE パーミッションを組み合わせます。

関連情報

- Data Grid Security API

1.1.2. ロールとパーミッションマッパー

Data Grid は、ユーザーをプリンシパルのコレクションとして実装します。プリンシパルは、ユーザー名などの個々のユーザー ID、またはユーザーが属するグループのいずれかを表します。内部的には、これらは **javax.security.auth.Subject** クラスで実装されます。

承認を有効にするには、プリンシパルをロール名にマップし、その後、ロール名を一連のパーミッションに展開する必要があります。

Data Grid には、セキュリティープリンシパルをロールに関連付ける **PrincipalRoleMapper** API と、ロールを特定のパーミッションに関連付ける **RolePermissionMapper** API が含まれています。

Data Grid は以下のロールおよびパーミッションのマッパーの実装を提供します。

クラスターロールマッパー

クラスターレジストリーにロールマッピングのプリンシパルを保存します。

クラスターパーミッションマッパー

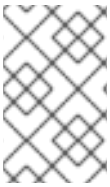
ロールとパーミッションのマッピングをクラスターレジストリーに保存します。ユーザーのロールとパーミッションを動的に変更できます。

ID ロールマッパー

ロール名としてプリンシパル名を使用します。プリンシパル名のタイプまたはフォーマットはソースに依存します。たとえば、LDAP ディレクトリーでは、プリンシパル名を識別名 (DN) にすることができます。

コモンネームロールマッパー

ロール名としてコモンネーム (CN) を使用します。このロールマッパーは、識別名 (DN) を含む LDAP ディレクトリーまたはクライアント証明書で使用できます。たとえば、**cn=managers,ou=people,dc=example,dc=com** は、**managers** ロールにマッピングされます。



注記

デフォルトでは、プリンシパルとロールのマッピングは、グループを表すプリンシパルにのみ適用されます。ユーザープリンシパルのマッピングも実行するように Data Grid を設定できます。

1.1.2.1. Data Grid でのロールとパーミッションへのユーザーのマッピング

LDAP サーバーから DN のコレクションとして取得された次のユーザーを考えてみましょう。

```
CN=myapplication,OU=applications,DC=mycompany
CN=dataprocessors,OU=groups,DC=mycompany
CN=finance,OU=groups,DC=mycompany
```

コモンネームロールマッパー を使用すると、ユーザーは次のロールにマッピングされます。

```
dataprocessors
finance
```

Data Grid には次のロール定義があります。

```
dataprocessors: ALL_WRITE ALL_READ
finance: LISTEN
```

ユーザーには次のパーミッションが与えられます。

```
ALL_WRITE ALL_READ LISTEN
```

関連情報

- Data Grid Security API
- org.infinispan.security.PrincipalRoleMapper
- org.infinispan.security.RolePermissionMapper
- org.infinispan.security.mappers.IdentityRoleMapper
- org.infinispan.security.mappers.CommonNameRoleMapper

1.1.3. ロールマッパーの設定

Data Grid は、デフォルトでクラスターロールマッパーとクラスターパーミッションマッパーを有効にします。ロールマッピングに別の実装を使用するには、ロールマッパーを設定する必要があります。

手順

1. Data Grid 設定を開いて編集します。
2. ロールマッパーを、Cache Manager 設定のセキュリティー許可の一部として宣言します。
3. 変更を設定に保存します。

組み込みキャッシュを使用すると、**principalRoleMapper()** および **rolePermissionMapper()** メソッドを使用して、ロールおよびパーミッションマッパーをプログラムで設定できます。

ロールマッパーの設定

XML

```
<cache-container>
  <security>
    <authorization>
      <common-name-role-mapper />
    </authorization>
  </security>
</cache-container>
```

JSON

```
{
  "infinispan" : {
    "cache-container" : {
      "security" : {
        "authorization" : {
          "common-name-role-mapper": {}
        }
      }
    }
  }
}
```

YAML

```
infinispan:
  cacheContainer:
    security:
      authorization:
        commonNameRoleMapper: ~
```

関連情報

- Data Grid 設定スキーマ参照

1.1.4. クラスターのロールと権限マッパーの設定

クラスターロールマッパーは、プリンシパルとロール間の動的なマッピングを維持します。クラスター権限マッパーは、動的なロール定義セットを維持します。どちらの場合も、マッピングはクラスターレジストリーに保存され、ランタイム時に CLI または REST API を使用して操作できます。

前提条件

- Data Grid に **ADMIN** 権限がある。
- Data Grid CLI を起動している。
- 実行中の Data Grid クラスターに接続している。

1.1.4.1. 新しいロールの作成

新しいロールを作成し、権限を設定します。

手順

- **user roles create** コマンドを使用してロールを作成します。以下に例を示します。

```
user roles create --permissions=ALL_READ,ALL_WRITE simple
```

検証

user roles ls コマンドでユーザーに付与したロールを一覧表示します。

```
user roles ls
["observer","application","admin","monitor","simple","deployer"]
```

user roles describe コマンドを使用してロールを説明します。

```
user roles describe simple
{
  "name" : "simple",
  "permissions" : [ "ALL_READ","ALL_WRITE" ]
}
```

1.1.4.2. ユーザーへのロール付与

ユーザーにロールを割り当て、キャッシュ操作や Data Grid のリソースとのやり取りを行う権限を付与します。

ヒント

同じロールを複数のユーザーに割り当て、それらの権限を一元管理する場合は、ユーザーではなくグループにロールを付与します。

前提条件

- Data Grid に **ADMIN** 権限がある。
- Data Grid ユーザーを作成すること。

手順

1. Data Grid への CLI 接続を作成します。
2. **user roles grant** コマンドでユーザーにロールを割り当てます。以下に例を示します。

```
user roles grant --roles=deployer katie
```

検証

user roles ls コマンドでユーザーに付与したロールを一覧表示します。

```
user roles ls katie
["deployer"]
```

1.1.4.3. クラスターロールマッパー名リライター

デフォルトでは、プリンシパル名とロール間の厳密な文字列等価性を使用してマッピングが実行されます。ルックアップを実行する前に、プリンシパル名にトランスフォーメーションを適用するようにクラスターロールマッパーを設定することができます。

手順

1. Data Grid 設定を開いて編集します。
2. Cache Manager 設定のセキュリティー認可の一環として、クラスターロールマッパーの名前リライターを指定します。
3. 変更を設定に保存します。

プリンシパル名の形式は、セキュリティーレルムの種類に応じて異なります。

- プロパティーとトークンレルムは、単純な文字列を返す場合があります
- 信頼と LDAP レルムは X.500 形式の識別名を返す場合があります
- Kerberos レルムは **user@domain** 形式の名前を返す場合があります

次のいずれかのトランスフォーマーを使用して、名前を共通形式に正規化できます。

1.1.4.3.1. ケースプリンシパルトランスフォーマー

XML

```
<cache-container>
  <security>
    <authorization>
      <cluster-role-mapper>
        <name-rewriter>
          <case-principal-transformer uppercase="false"/>
        </name-rewriter>
      </cluster-role-mapper>
    </authorization>
  </security>
</cache-container>
```

-

JSON

```
{
  "cache-container": {
    "security": {
      "authorization": {
        "cluster-role-mapper": {
          "name-rewriter": {
            "case-principal-transformer": {}
          }
        }
      }
    }
  }
}
```

YAML

```
cacheContainer:
  security:
    authorization:
      clusterRoleMapper:
        nameRewriter:
          casePrincipalTransformer:
            uppercase: false
```

1.1.4.3.2. Regex プリンシパルトランスフォーマー

XML

```
<cache-container>
  <security>
    <authorization>
      <cluster-role-mapper>
        <name-rewriter>
          <regex-principal-transformer pattern="cn=([^,]+),.*" replacement="$1"/>
        </name-rewriter>
      </cluster-role-mapper>
    </authorization>
  </security>
</cache-container>
```

JSON

```
{
  "cache-container": {
    "security": {
      "authorization": {
        "cluster-role-mapper": {
          "name-rewriter": {
            "regex-principal-transformer": {
```

```
            "pattern": "cn=([^,]+),.*",
            "replacement": "$1"
          }
        }
      }
    }
  }
}
```

YAML

```
cacheContainer:
  security:
    authorization:
      clusterRoleMapper:
        nameRewriter:
          regexPrincipalTransformer:
            pattern: "cn=([^,]+),.*"
            replacement: "$1"
```

関連情報

- Data Grid 設定スキーマ参照

1.2. セキュリティー承認によるキャッシュの設定

キャッシュにセキュリティー承認を追加し、ロールベースアクセス制御 (RBAC) を実施することができます。これには、Data Grid ユーザーが、キャッシュ操作を実行するのに十分なレベルのパーミッションを持つロールを持っている必要があります。

前提条件

- Data Grid のユーザーを作成し、ロールを付与するか、グループに割り当てている。

手順

1. Data Grid 設定を開いて編集します。
2. 設定に **security** セクションを追加します。
3. **authorization** 要素を使用して、ユーザーがキャッシュ操作を実行するために必要なロールを指定します。
   Cache Manager で定義されたすべてのロールを暗黙的に追加するか、ロールのサブセットを明示的に定義できます。
4. 変更を設定に保存します。

暗黙的なロール設定
次の設定では、Cache Manager で定義されたすべてのロールが暗黙的に追加されます。

XML

```
<distributed-cache>
  <security>
    <authorization/>
  </security>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "security": {
      "authorization": {
        "enabled": true
      }
    }
  }
}
```

YAML

```
distributedCache:
  security:
    authorization:
      enabled: true
```

明示的なロール設定

次の設定では、Cache Manager で定義されたロールのサブセットを明示的に追加しています。この場合、Data Grid は、設定されたロールのいずれかを持たないユーザーのキャッシュ操作を拒否します。

XML

```
<distributed-cache>
  <security>
    <authorization roles="admin supervisor"/>
  </security>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "security": {
      "authorization": {
        "enabled": true,
        "roles": ["admin","supervisor"]
      }
    }
  }
}
```

YAML

```
distributedCache:
  security:
    authorization:
      enabled: true
      roles: ["admin","supervisor"]
```

第2章 セキュリティーレルム

セキュリティーレルムは、ユーザー ID にアクセスおよび検証する環境内のネットワークプロトコルおよびインフラストラクチャーと Data Grid Server デプロイメントを統合します。

2.1. セキュリティーレルムの作成

セキュリティーレルムを Data Grid Server 設定に追加し、デプロイメントへのアクセスを制御します。設定に 1 つ以上のセキュリティーレルムを追加できます。

注記

設定にセキュリティーレルムを追加すると、Data Grid Server は Hot Rod および REST エンドポイントの一致する認証メカニズムを自動的に有効にします。

前提条件

- 必要に応じて、ソケットバインディングを Data Grid Server 設定に追加します。
- キーストアを作成するか、PEM ファイルがあり、TLS/SSL 暗号化でセキュリティーレルムを設定します。
  Data Grid Server は起動時にキーストアを生成することもできます。
- セキュリティーレルム設定に依存するリソースまたはサービスをプロビジョニングします。
  たとえば、トークンレルムを追加する場合は、OAuth サービスをプロビジョニングする必要があります。

この手順では、複数のプロパティーレルムを設定する方法を説明します。開始する前に、ユーザーを追加し、コマンドラインインターフェイス (CLI) でパーミッションを割り当てるプロパティーファイルを作成する必要があります。**user create** コマンドを使用します。

```
user create <username> -p <changeme> -g <role> \
    --users-file=application-users.properties \
    --groups-file=application-groups.properties

user create <username> -p <changeme> -g <role> \
    --users-file=management-users.properties \
    --groups-file=management-groups.properties
```

ヒント

サンプルおよび詳細情報について **user create --help** を実行します。

注記

CLI を使用してプロパティーレルムに認証情報を追加すると、接続しているサーバーインスタンスにのみユーザーが作成されます。プロパティーレルムの認証情報をクラスター内の各ノードに手動で同期する必要があります。

手順

1. Data Grid Server 設定を開いて編集します。

2. 複数のセキュリティーレルムの作成を含めるには、**security** 設定の **security-realms** 要素を使用します。

3. **security-realm** 要素でセキュリティーレルムを追加し、**name** 属性の一意の名前を付けます。この例に従うには、**application-realm** という名前のセキュリティーレルムと、**management-realm** という名前の 1 つのセキュリティーレルムを作成します。

4. Data Grid Server の TLS/SSL 識別に **server-identities** 要素を指定して、必要に応じてキーストアを設定します。

5. 以下の要素またはフィールド 1 つを追加して、セキュリティーレルムのタイプを指定します。

   - **properties-realm**
   - **ldap-realm**
   - **token-realm**
   - **truststore-realm**

6. 必要に応じて、設定するセキュリティーレルムタイプのプロパティーを指定します。例に従うには、**user-properties** および **group-properties** 要素またはフィールドの **path** 属性を使用して、CLI で作成した ***.properties** ファイルを指定します。

7. 複数の異なるタイプのセキュリティーレルムを設定に追加する場合は、**distributed-realm** 要素またはフィールドを含めて、Data Grid Server がレルムを相互に組み合わせて使用できるようにします。

8. **security-realm** 属性でセキュリティーレルムを使用するように Data Grid Server エンドポイントを設定します。

9. 変更を設定に保存します。

複数のプロパティーレルム

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="application-realm">
        <properties-realm groups-attribute="Roles">
          <user-properties path="application-users.properties"/>
          <group-properties path="application-groups.properties"/>
        </properties-realm>
      </security-realm>
      <security-realm name="management-realm">
        <properties-realm groups-attribute="Roles">
          <user-properties path="management-users.properties"/>
          <group-properties path="management-groups.properties"/>
        </properties-realm>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "management-realm",
        "properties-realm": {
          "groups-attribute": "Roles",
          "user-properties": {
            "digest-realm-name": "management-realm",
            "path": "management-users.properties"
          },
          "group-properties": {
            "path": "management-groups.properties"
          }
        }
      }, {
        "name": "application-realm",
        "properties-realm": {
          "groups-attribute": "Roles",
          "user-properties": {
            "digest-realm-name": "application-realm",
            "path": "application-users.properties"
          },
          "group-properties": {
            "path": "application-groups.properties"
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "management-realm"
        propertiesRealm:
          groupsAttribute: "Roles"
          userProperties:
            digestRealmName: "management-realm"
            path: "management-users.properties"
          groupProperties:
            path: "management-groups.properties"
      - name: "application-realm"
        propertiesRealm:
          groupsAttribute: "Roles"
          userProperties:
            digestRealmName: "application-realm"
            path: "application-users.properties"
          groupProperties:
            path: "application-groups.properties"
```

–

2.2. KERBEROS ID の設定

Data Grid Server 設定のセキュリティーレルムに Kerberos ID を追加して、Kerberos パスワードから派生するサービスプリンシパル名と暗号化されたキーが含まれる **keytab** ファイルを使用します。

前提条件

- Kerberos サービスアカウントプリンシパルがある。

注記

キータブ ファイルには、ユーザーとサービスのアカウントプリンシパルの両方を含めることができます。しかし、Data Grid Server はサービスアカウントプリンシパルのみを使用します。これは、クライアントに ID を提供し、クライアントが Kerberos サーバーで認証できることを意味します。

ほとんどの場合、Hot Rod および REST エンドポイントに固有のプリンシパルを作成します。たとえば、"INFINISPAN.ORG" ドメインに "datagrid" サーバーがある場合は、以下のサービスプリンシパルを作成する必要があります。

- **hotrod/datagrid@INFINISPAN.ORG** は Hot Rod サービスを特定します。
- **HTTP/datagrid@INFINISPAN.ORG** は REST サービスを識別します。

手順

1. Hot Rod および REST サービスのキータブファイルを作成します。

   Linux

   ```
   ktutil
   ktutil:  addent -password -p datagrid@INFINISPAN.ORG -k 1 -e aes256-cts
   Password for datagrid@INFINISPAN.ORG: [enter your password]
   ktutil:  wkt http.keytab
   ktutil:  quit
   ```

   Microsoft Windows

   ```
   ktpass -princ HTTP/datagrid@INFINISPAN.ORG -pass * -mapuser
   INFINISPAN\USER_NAME
   ktab -k http.keytab -a HTTP/datagrid@INFINISPAN.ORG
   ```

2. keytab ファイルを Data Grid Server インストールの **server/conf** ディレクトリーにコピーします。
3. Data Grid Server 設定を開いて編集します。
4. **server-identities** 定義を Data Grid サーバーのセキュリティーレルムに追加します。
5. Hot Rod および REST コネクターにサービスプリンシパルを提供するキータブファイルの場所を指定します。

6. Kerberos サービスプリンシパルに名前を付けます。
7. 変更を設定に保存します。

Kerberos ID の設定

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="kerberos-realm">
        <server-identities>
          <!-- Specifies a keytab file that provides a Kerberos identity. -->
          <!-- Names the Kerberos service principal for the Hot Rod endpoint. -->
          <!-- The required="true" attribute specifies that the keytab file must be present when the server
starts. -->
          <kerberos keytab-path="hotrod.keytab"
                    principal="hotrod/datagrid@INFINISPAN.ORG"
                    required="true"/>
          <!-- Specifies a keytab file and names the Kerberos service principal for the REST endpoint. -->
          <kerberos keytab-path="http.keytab"
                    principal="HTTP/localhost@INFINISPAN.ORG"
                    required="true"/>
        </server-identities>
      </security-realm>
    </security-realms>
  </security>
  <endpoints>
    <endpoint socket-binding="default"
              security-realm="kerberos-realm">
      <hotrod-connector>
        <authentication>
          <sasl server-name="datagrid"
                server-principal="hotrod/datagrid@INFINISPAN.ORG"/>
        </authentication>
      </hotrod-connector>
      <rest-connector>
        <authentication server-principal="HTTP/localhost@INFINISPAN.ORG"/>
      </rest-connector>
    </endpoint>
  </endpoints>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "kerberos-realm",
        "server-identities": [{
          "kerberos": {
            "principal": "hotrod/datagrid@INFINISPAN.ORG",
```

```
            "keytab-path": "hotrod.keytab",
            "required": true
          },
          "kerberos": {
            "principal": "HTTP/localhost@INFINISPAN.ORG",
            "keytab-path": "http.keytab",
            "required": true
          }
        }]
      }]
    },
    "endpoints": {
      "endpoint": {
        "socket-binding": "default",
        "security-realm": "kerberos-realm",
        "hotrod-connector": {
          "authentication": {
            "security-realm": "kerberos-realm",
            "sasl": {
              "server-name": "datagrid",
              "server-principal": "hotrod/datagrid@INFINISPAN.ORG"
            }
          }
        },
        "rest-connector": {
          "authentication": {
            "server-principal": "HTTP/localhost@INFINISPAN.ORG"
          }
        }
      }
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "kerberos-realm"
        serverIdentities:
          - kerberos:
              principal: "hotrod/datagrid@INFINISPAN.ORG"
              keytabPath: "hotrod.keytab"
              required: "true"
          - kerberos:
              principal: "HTTP/localhost@INFINISPAN.ORG"
              keytabPath: "http.keytab"
              required: "true"
  endpoints:
    endpoint:
      socketBinding: "default"
      securityRealm: "kerberos-realm"
      hotrodConnector:
        authentication:
```

```
      sasl:
        serverName: "datagrid"
        serverPrincipal: "hotrod/datagrid@INFINISPAN.ORG"
    restConnector:
      authentication:
        securityRealm: "kerberos-realm"
        serverPrincipal" : "HTTP/localhost@INFINISPAN.ORG"
```

2.3. プロパティーレルム

プロパティーレルムはプロパティーファイルを使用して、ユーザーおよびグループを定義します。

- **users.properties** には Data Grid ユーザーの認証情報が含まれます。パスワードは、 **DIGEST-MD5** および **DIGEST** 認証メカニズムを使用して事前署名できます。
- **groups.properties** は、ユーザーをロールおよびパーミッションに関連付けます。

注記

Data Grid CLI を使用して、ファイルに正しいセキュリティーレルム名を入力することで、プロパティーファイルに関連する認証の問題を回避することができます。**infinispan.xml** ファイルを開き、**<security-realm name>** プロパティーに移動すると、Data Grid Server の正しいセキュリティーレルム名を見つけることができます。Data Grid Server から別の Data Grid Server へプロパティーファイルをコピーする場合、セキュリティーレルム名がターゲットエンドポイントの正しい認証メカニズムに対応していることを確認してください。

users.properties

```
myuser=a_password
user2=another_password
```

groups.properties

```
myuser=supervisor,reader,writer
user2=supervisor
```

プロパティーレルム設定

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="default">
        <!-- groups-attribute configures the "groups.properties" file to contain security authorization roles.
-->
        <properties-realm groups-attribute="Roles">
          <user-properties path="users.properties"
                           relative-to="infinispan.server.config.path"
                           plain-text="true"/>
          <group-properties path="groups.properties"
                            relative-to="infinispan.server.config.path"/>
```

```
        </properties-realm>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "default",
        "properties-realm": {
          "groups-attribute": "Roles",
          "user-properties": {
            "digest-realm-name": "default",
            "path": "users.properties",
            "relative-to": "infinispan.server.config.path",
            "plain-text": true
          },
          "group-properties": {
            "path": "groups.properties",
            "relative-to": "infinispan.server.config.path"
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "default"
        propertiesRealm:
          # groupsAttribute configures the "groups.properties" file
          # to contain security authorization roles.
          groupsAttribute: "Roles"
          userProperties:
            digestRealmName: "default"
            path: "users.properties"
            relative-to: 'infinispan.server.config.path'
            plainText: "true"
          groupProperties:
            path: "groups.properties"
            relative-to: 'infinispan.server.config.path'
```

2.4. LDAP レルム

LDAP レルムは、OpenLDAP、Red Hat Directory Server、Apache Directory Server、Microsoft Active Directory などの LDAP サーバーに接続して、ユーザーを認証し、メンバーシップ情報を取得します。

注記

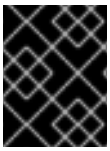
LDAP サーバーは、サーバーのタイプとデプロイメントに応じて、異なるエントリーレイアウトを持つことができます。考えられるすべての設定の例を提供することは、このドキュメントでは扱っていません。

2.4.1. LDAP 接続プロパティー

LDAP レルム設定で LDAP 接続プロパティーを指定します。

次のプロパティーが必要です。

url	LDAP サーバーの URL を指定します。TLS を使用したセキュアな接続のために、URL は **ldap://hostname:port** または **ldaps://hostname:port** の形式である必要があります。
principal	LDAP サーバー内の有効なユーザーの識別名 (DN) を指定します。DN は、LDAP ディレクトリー構造内でユーザーを一意に識別します。
credential	前述のプリンシパルに関連するパスワードに対応します。



重要

LDAP 接続のプリンシパルには、LDAP クエリーを実行し、特定の属性にアクセスするために必要な権限が必要です。

ヒント

connection-pooling を有効にすると、LDAP サーバーに対する認証のパフォーマンスが大幅に向上します。接続プーリングメカニズムは JDK によって提供されます。詳細は、接続プーリングの設定 および Java チュートリアル: プーリング を参照してください。

2.4.2. LDAP レルムのユーザー認証方法

LDAP レルムでのユーザー認証方法を設定します。

LDAP レルムは、次の 2 つの方法でユーザーを認証できます。

ハッシュ化されたパスワードの比較	ユーザーのパスワード属性 (通常は **userPassword**) に保存されているハッシュ化されたパスワードを比較します。
直接検証	提供された認証情報を使用して、LDAP サーバーに対して認証を行います。 Active Directory で機能する唯一の方法は直接検証です。**password** 属性へのアクセスが禁止されているためです。

重要

direct-verification 属性でハッシュ化を実行するエンドポイントの認証メカニズムは使用できません。この方法ではクリアテキストのパスワードが必要であるためです。そのため、Active Directory Server と統合するには、REST エンドポイントでは **BASIC** 認証メカニズムを、Hot Rod エンドポイントでは **PLAIN** を使用する必要があります。より安全な代替方法として、**SPNEGO**、**GSSAPI**、および **GS2-KRB5** 認証メカニズムを可能にする Kerberos を使用することができます。

LDAP レルムはディレクトリーを検索して、認証されたユーザーに対応するエントリーを探し出します。**rdn-identifier** 属性は、指定された識別子 (通常はユーザー名) をもとにユーザーエントリーを検索する LDAP 属性を指定します (例: **uid** または **sAMAccountName** 属性)。**search-recursive="true"** を設定に追加して、ディレクトリーを再帰的に検索します。デフォルトでは、ユーザーエントリーの検索は **(rdn_identifier={0})** フィルターを使用します。**filter-name** 属性を使用して、別のフィルターを指定できます。

2.4.3. ユーザーエントリーの関連グループへのマッピング

LDAP レルム設定で **attribute-mapping** 要素を指定して、ユーザーがメンバーであるすべてのグループを取得して関連付けます。

メンバーシップ情報は通常、次の 2 つの方法で保存されます。

- 通常 **member** 属性にクラス **groupOfNames** または **groupOfUniqueNames** を持つグループエントリーの下。これは、Active Directory を除く、ほとんどの LDAP インストールにおけるデフォルトの動作です。この場合、属性フィルターを使用できます。このフィルターは、提供されたフィルターに一致するエントリーを検索します。フィルターは、ユーザーの DN と等しい **member** 属性を持つグループを検索します。次に、フィルターは、**from** で指定されたグループエントリーの CN を抽出し、それをユーザーの **Roles** に追加します。

- **memberOf** 属性のユーザーエントリー内。これは通常、Active Directory の場合に当てはまります。この場合、以下のような属性参照を使用する必要があります。
  <attribute-reference reference="memberOf" from="cn" to="Roles" />

  この参照は、ユーザーのエントリーからすべての **memberOf** 属性を取得し、**from** で指定された CN を抽出し、それらをユーザーのグループに追加します (**Roles** はグループのマッピングに使用される内部名です)。

2.4.4. LDAP レルム設定リファレンス

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="ldap-realm">
        <!-- Specifies connection properties. -->
        <ldap-realm url="ldap://my-ldap-server:10389"
                    principal="uid=admin,ou=People,dc=infinispan,dc=org"
                    credential="strongPassword"
                    connection-timeout="3000"
                    read-timeout="30000"
                    connection-pooling="true"
                    referral-mode="ignore"
```

```
            page-size="30"
            direct-verification="true">
        <!-- Defines how principals are mapped to LDAP entries. -->
        <identity-mapping rdn-identifier="uid"
                          search-dn="ou=People,dc=infinispan,dc=org"
                          search-recursive="false">
          <!-- Retrieves all the groups of which the user is a member. -->
          <attribute-mapping>
            <attribute from="cn" to="Roles"
                       filter="(&amp;(objectClass=groupOfNames)(member={1}))"
                       filter-dn="ou=Roles,dc=infinispan,dc=org"/>
          </attribute-mapping>
        </identity-mapping>
      </ldap-realm>
    </security-realm>
  </security-realms>
 </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "ldap-realm",
        "ldap-realm": {
          "url": "ldap://my-ldap-server:10389",
          "principal": "uid=admin,ou=People,dc=infinispan,dc=org",
          "credential": "strongPassword",
          "connection-timeout": "3000",
          "read-timeout": "30000",
          "connection-pooling": "true",
          "referral-mode": "ignore",
          "page-size": "30",
          "direct-verification": "true",
          "identity-mapping": {
            "rdn-identifier": "uid",
            "search-dn": "ou=People,dc=infinispan,dc=org",
            "search-recursive": "false",
            "attribute-mapping": [{
              "from": "cn",
              "to": "Roles",
              "filter": "(&(objectClass=groupOfNames)(member={1}))",
              "filter-dn": "ou=Roles,dc=infinispan,dc=org"
            }]
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: ldap-realm
        ldapRealm:
          url: 'ldap://my-ldap-server:10389'
          principal: 'uid=admin,ou=People,dc=infinispan,dc=org'
          credential: strongPassword
          connectionTimeout: '3000'
          readTimeout: '30000'
          connectionPooling: true
          referralMode: ignore
          pageSize: '30'
          directVerification: true
          identityMapping:
            rdnIdentifier: uid
            searchDn: 'ou=People,dc=infinispan,dc=org'
            searchRecursive: false
            attributeMapping:
              - filter: '(&(objectClass=groupOfNames)(member={1}))'
                filterDn: 'ou=Roles,dc=infinispan,dc=org'
                from: cn
                to: Roles
```

2.4.4.1. LDAP レルムプリンシパルの書き換え

GSSAPI、**GS2-KRB5**、**Negotiate** などの SASL 認証メカニズムによって取得されるプリンシパルには、通常、ドメイン名 (例: **myuser@INFINISPAN.ORG**) が含まれます。これらのプリンシパルを LDAP クエリーで使用する前に、プリンシパルを変換して互換性を確保する必要があります。このプロセスは書き換えと呼ばれます。

Data Grid には次のトランスフォーマーが含まれています。

case-principal-transformer	プリンシパルをすべて大文字またはすべて小文字に書き換えます。たとえば **MyUser** は、大文字モードでは **MYUSER** に書き換えられ、小文字モードでは **myuser** に書き換えられます。
common-name-principal-transformer	プリンシパルを LDAP 識別名形式 (RFC 4514 で定義) に書き換えます。タイプ **CN** (commonName) の最初の属性が抽出されます。たとえば、**DN=CN=myuser,OU=myorg,DC=mydomain** は **myuser** に書き換えられます。
regex-principal-transformer	キャプチャーグループを含む正規表現を使用してプリンシパルを書き換え、たとえば、任意の部分文字列の抽出を可能にします。

2.4.4.2. LDAP プリンシパル書き換え設定リファレンス

ケースプリンシパルトランスフォーマー

XML

```xml
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="ldap-realm">
        <ldap-realm url="ldap://${org.infinispan.test.host.address}:10389"
                    principal="uid=admin,ou=People,dc=infinispan,dc=org"
                    credential="strongPassword">
          <name-rewriter>
            <!-- Defines a rewriter that transforms usernames to lowercase -->
            <case-principal-transformer uppercase="false"/>
          </name-rewriter>
          <!-- further configuration omitted -->
        </ldap-realm>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```json
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "ldap-realm",
        "ldap-realm": {
          "principal": "uid=admin,ou=People,dc=infinispan,dc=org",
          "url": "ldap://${org.infinispan.test.host.address}:10389",
          "credential": "strongPassword",
          "name-rewriter": {
            "case-principal-transformer": {
              "uppercase": false
            }
          }
        }
      }]
    }
  }
}
```

YAML

```yaml
server:
  security:
    securityRealms:
      - name: "ldap-realm"
        ldapRealm:
          principal: "uid=admin,ou=People,dc=infinispan,dc=org"
          url: "ldap://${org.infinispan.test.host.address}:10389"
          credential: "strongPassword"
          nameRewriter:
            casePrincipalTransformer:
              uppercase: false
          # further configuration omitted
```

-

コモンネームプリンシパルトランスフォーマー

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="ldap-realm">
        <ldap-realm url="ldap://${org.infinispan.test.host.address}:10389"
                    principal="uid=admin,ou=People,dc=infinispan,dc=org"
                    credential="strongPassword">
          <name-rewriter>
            <!-- Defines a rewriter that obtains the first CN from a DN -->
            <common-name-principal-transformer />
          </name-rewriter>
          <!-- further configuration omitted -->
        </ldap-realm>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "ldap-realm",
        "ldap-realm": {
          "principal": "uid=admin,ou=People,dc=infinispan,dc=org",
          "url": "ldap://${org.infinispan.test.host.address}:10389",
          "credential": "strongPassword",
          "name-rewriter": {
            "common-name-principal-transformer": {}
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "ldap-realm"
        ldapRealm:
          principal: "uid=admin,ou=People,dc=infinispan,dc=org"
          url: "ldap://${org.infinispan.test.host.address}:10389"
          credential: "strongPassword"
```

```
        nameRewriter:
          commonNamePrincipalTransformer: ~
        # further configuration omitted
```

Regex プリンシパルトランスフォーマー

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="ldap-realm">
        <ldap-realm url="ldap://${org.infinispan.test.host.address}:10389"
                    principal="uid=admin,ou=People,dc=infinispan,dc=org"
                    credential="strongPassword">
          <name-rewriter>
            <!-- Defines a rewriter that extracts the username from the principal using a regular
expression. -->
            <regex-principal-transformer pattern="(.*)@INFINISPAN\.ORG"
                                         replacement="$1"/>
          </name-rewriter>
          <!-- further configuration omitted -->
        </ldap-realm>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "ldap-realm",
        "ldap-realm": {
          "principal": "uid=admin,ou=People,dc=infinispan,dc=org",
          "url": "ldap://${org.infinispan.test.host.address}:10389",
          "credential": "strongPassword",
          "name-rewriter": {
            "regex-principal-transformer": {
              "pattern": "(.*)@INFINISPAN\\.ORG",
              "replacement": "$1"
            }
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "ldap-realm"
        ldapRealm:
          principal: "uid=admin,ou=People,dc=infinispan,dc=org"
          url: "ldap://${org.infinispan.test.host.address}:10389"
          credential: "strongPassword"
          nameRewriter:
            regexPrincipalTransformer:
              pattern: (.*)@INFINISPAN\.ORG
              replacement: "$1"
          # further configuration omitted
```

2.4.4.3. Data Grid を使用した LDAP ユーザーとグループのマッピングプロセス

この例では、LDAP ユーザーとグループをロードし、Data Grid サブジェクトに内部マッピングするプロセスを示します。以下は、複数の LDAP エントリーを記述した LDIF (LDAP Data Interchange Format) ファイルです。

LDIF

```
# Users

dn: uid=root,ou=People,dc=infinispan,dc=org
objectclass: top
objectclass: uidObject
objectclass: person
uid: root
cn: root
sn: root
userPassword: strongPassword

# Groups

dn: cn=admin,ou=Roles,dc=infinispan,dc=org
objectClass: top
objectClass: groupOfNames
cn: admin
description: the Infinispan admin group
member: uid=root,ou=People,dc=infinispan,dc=org

dn: cn=monitor,ou=Roles,dc=infinispan,dc=org
objectClass: top
objectClass: groupOfNames
cn: monitor
description: the Infinispan monitor group
member: uid=root,ou=People,dc=infinispan,dc=org
```

root ユーザーは、**admin** および **monitor** グループのメンバーです。

エンドポイントの 1 つでパスワード **strongPassword** を使用してユーザー **root** を認証するよう要求されると、次の操作が実行されます。

- ユーザー名は、選択されたプリンシパルトランスフォーマーを使用して、必要に応じて書き換えられます。
- レルムは、**uid** 属性が **root** に等しいエントリーを **ou=People,dc=infinispan,dc=org** ツリー内で検索し、DN **uid=root,ou=People,dc=infinispan,dc=org** のエントリーを探し出します。このエントリーがユーザープリンシパルになります。
- レルムは、**u=Roles,dc=infinispan,dc=org** ツリー内で、**member** 属性に **uid=root,ou=People,dc=infinispan,dc=org** を含む **objectClass=groupOfNames** のエントリーを検索します。この場合、**cn=admin,ou=Roles,dc=infinispan,dc=org** と **cn=monitor,ou=Roles,dc=infinispan,dc=org** の 2 つのエントリーが見つかります。これらのエントリーから、グループプリンシパルとなる **cn** 属性を抽出します。

したがって、結果として得られるサブジェクトは次のようになります。

- NamePrincipal: **uid=root,ou=People,dc=infinispan,dc=org**
- RolePrincipal: **admin**
- RolePrincipal: **monitor**

この時点で、グローバル承認マッパーが上記のサブジェクトに適用され、プリンシパルがロールに変換されます。その後、ロールが一連のパーミッションに展開され、パーミッションが要求されたキャッシュと操作に対して検証されます。

2.5. トークンレルム

トークンレルムは外部サービスを使用してトークンを検証し、Red Hat SSO などの RFC-7662 (OAuth2 トークンイントロスペクション) と互換性のあるプロバイダーを必要とします。

トークンレルムの設定

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="token-realm">
        <!-- Specifies the URL of the authentication server. -->
        <token-realm name="token"
                     auth-server-url="https://oauth-server/auth/">
          <!-- Specifies the URL of the token introspection endpoint. -->
          <oauth2-introspection introspection-url="https://oauth-
server/auth/realms/infinispan/protocol/openid-connect/token/introspect"
                                client-id="infinispan-server"
                                client-secret="1fdca4ec-c416-47e0-867a-3d471af7050f"/>
        </token-realm>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
```

```
  "server": {
    "security": {
      "security-realms": [{
        "name": "token-realm",
        "token-realm": {
          "auth-server-url": "https://oauth-server/auth/",
          "oauth2-introspection": {
            "client-id": "infinispan-server",
            "client-secret": "1fdca4ec-c416-47e0-867a-3d471af7050f",
            "introspection-url": "https://oauth-server/auth/realms/infinispan/protocol/openid-
connect/token/introspect"
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: token-realm
        tokenRealm:
          authServerUrl: 'https://oauth-server/auth/'
          oauth2Introspection:
            clientId: infinispan-server
            clientSecret: '1fdca4ec-c416-47e0-867a-3d471af7050f'
            introspectionUrl: 'https://oauth-server/auth/realms/infinispan/protocol/openid-
connect/token/introspect'
```

2.6. トラストストアレルム

トラストストアレルムは、接続のネゴシエート時に Data Grid Server およびクライアント ID を検証する証明書または証明書チェーンを使用します。

キーストア

Data Grid Server アイデンティティーをクライアントに提供するサーバー証明書が含まれます。サーバー証明書でキーストアを設定する場合、Data Grid Server は業界標準の SSL/TLS プロトコルを使用してトラフィックを暗号化します。

トラストストア

クライアントが Data Grid Server に提示するクライアント証明書または証明書チェーンが含まれます。クライアントのトラストストアはオプションで、Data Grid Server がクライアント証明書認証を実行できるようになっています。

クライアント証明書認証

Data Grid Server でクライアント証明書を検証または認証する場合は、**require-ssl-client-auth="true"** 属性をエンドポイント設定に追加する必要があります。

トラストストアレルムの設定

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="trust-store-realm">
        <server-identities>
          <ssl>
            <!-- Provides an SSL/TLS identity with a keystore that contains server certificates. -->
            <keystore path="server.p12"
                      relative-to="infinispan.server.config.path"
                      keystore-password="secret"
                      alias="server"/>
            <!-- Configures a trust store that contains client certificates or part of a certificate chain. -->
            <truststore path="trust.p12"
                        relative-to="infinispan.server.config.path"
                        password="secret"/>
          </ssl>
        </server-identities>
        <!-- Authenticates client certificates against the trust store. If you configure this, the trust store
must contain the public certificates for all clients. -->
        <truststore-realm/>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "trust-store-realm",
        "server-identities": {
          "ssl": {
            "keystore": {
              "path": "server.p12",
              "relative-to": "infinispan.server.config.path",
              "keystore-password": "secret",
              "alias": "server"
            },
            "truststore": {
              "path": "trust.p12",
              "relative-to": "infinispan.server.config.path",
              "password": "secret"
            }
          }
        },
        "truststore-realm": {}
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "trust-store-realm"
        serverIdentities:
          ssl:
            keystore:
              path: "server.p12"
              relative-to: "infinispan.server.config.path"
              keystore-password: "secret"
              alias: "server"
            truststore:
              path: "trust.p12"
              relative-to: "infinispan.server.config.path"
              password: "secret"
        trustStoreRealm: ~
```

2.7. 分散セキュリティーレルム

分散レルムは、複数のタイプのセキュリティーレルムを組み合わせます。ユーザーが Hot Rod または REST エンドポイントにアクセスしようとすると、認証を実行できるものを見つけるまで、Data Grid Server は各セキュリティーレルムを順番に使用します。

分散レルムの設定

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="distributed-realm">
        <ldap-realm url="ldap://my-ldap-server:10389"
                    principal="uid=admin,ou=People,dc=infinispan,dc=org"
                    credential="strongPassword">
          <identity-mapping rdn-identifier="uid"
                            search-dn="ou=People,dc=infinispan,dc=org"
                            search-recursive="false">
            <attribute-mapping>
              <attribute from="cn" to="Roles"
                         filter="(&amp;(objectClass=groupOfNames)(member={1}))"
                         filter-dn="ou=Roles,dc=infinispan,dc=org"/>
            </attribute-mapping>
          </identity-mapping>
        </ldap-realm>
        <properties-realm groups-attribute="Roles">
          <user-properties path="users.properties"
                           relative-to="infinispan.server.config.path"/>
          <group-properties path="groups.properties"
                            relative-to="infinispan.server.config.path"/>
        </properties-realm>
        <distributed-realm/>
      </security-realm>
```

```
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "distributed-realm",
        "ldap-realm": {
          "principal": "uid=admin,ou=People,dc=infinispan,dc=org",
          "url": "ldap://my-ldap-server:10389",
          "credential": "strongPassword",
          "identity-mapping": {
            "rdn-identifier": "uid",
            "search-dn": "ou=People,dc=infinispan,dc=org",
            "search-recursive": false,
            "attribute-mapping": {
              "attribute": {
                "filter": "(&(objectClass=groupOfNames)(member={1}))",
                "filter-dn": "ou=Roles,dc=infinispan,dc=org",
                "from": "cn",
                "to": "Roles"
              }
            }
          }
        },
        "properties-realm": {
          "groups-attribute": "Roles",
          "user-properties": {
            "digest-realm-name": "distributed-realm",
            "path": "users.properties"
          },
          "group-properties": {
            "path": "groups.properties"
          }
        },
        "distributed-realm": {}
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "distributed-realm"
        ldapRealm:
          principal: "uid=admin,ou=People,dc=infinispan,dc=org"
          url: "ldap://my-ldap-server:10389"
```

```
        credential: "strongPassword"
        identityMapping:
          rdnIdentifier: "uid"
          searchDn: "ou=People,dc=infinispan,dc=org"
          searchRecursive: "false"
          attributeMapping:
            attribute:
              filter: "(&(objectClass=groupOfNames)(member={1}))"
              filterDn: "ou=Roles,dc=infinispan,dc=org"
              from: "cn"
              to: "Roles"
      propertiesRealm:
        groupsAttribute: "Roles"
        userProperties:
          digestRealmName: "distributed-realm"
          path: "users.properties"
        groupProperties:
          path: "groups.properties"
      distributedRealm: ~
```

2.8. 集約セキュリティーレルム

集約レルムは複数のレルムを組み合わせます。最初のレルムは認証手順用で、他のレルムは認可手順のアイデンティティーをロードするためのものです。たとえば、これを使用して、クライアント証明書を介してユーザーを認証し、プロパティーまたは LDAP レルムからアイデンティティーを取得できます。

集約レルムの設定

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="default" default-realm="aggregate">
        <server-identities>
          <ssl>
            <keystore path="server.pfx" password="secret" alias="server"/>
            <truststore path="trust.pfx" password="secret"/>
          </ssl>
        </server-identities>
        <properties-realm name="properties" groups-attribute="Roles">
          <user-properties path="users.properties" relative-to="infinispan.server.config.path"/>
          <group-properties path="groups.properties" relative-to="infinispan.server.config.path"/>
        </properties-realm>
        <truststore-realm name="trust"/>
        <aggregate-realm authentication-realm="trust" authorization-realms="properties">
          <name-rewriter>
            <common-name-principal-transformer/>
          </name-rewriter>
        </aggregate-realm>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [
        {
          "name": "aggregate-realm",
          "default-realm": "aggregate",
          "server-identities": {
            "ssl": {
              "keystore": {
                "path": "server.p12",
                "relative-to": "infinispan.server.config.path",
                "keystore-password": "secret",
                "alias": "server"
              },
              "truststore": {
                "path": "trust.p12",
                "relative-to": "infinispan.server.config.path",
                "password": "secret"
              }
            }
          },
          "properties-realm": {
            "name": "properties",
            "groups-attribute": "Roles",
            "user-properties": {
              "digest-realm-name": "distributed-realm",
              "path": "users.properties"
            },
            "group-properties": {
              "path": "groups.properties"
            }
          },
          "truststore-realm": {
            "name": "trust"
          },
          "aggregate-realm": {
            "authentication-realm": "trust",
            "authorization-realms": ["properties"],
            "name-rewriter": {
              "common-name-principal-transformer": {}
            }
          }
        }
      ]
    }
  }
}
```

YAML

```
server:
  security:
```

```
securityRealms:
  - name: "aggregate-realm"
    defaultRealm: "aggregate"
    serverIdentities:
      ssl:
        keystore:
          path: "server.p12"
          relative-to: "infinispan.server.config.path"
          keystore-password: "secret"
          alias: "server"
        truststore:
          path: "trust.p12"
          relative-to: "infinispan.server.config.path"
          password: "secret"
    truststoreRealm:
      name: "trust"
    propertiesRealm:
      name: "properties"
      groupsAttribute: "Roles"
      userProperties:
        digestRealmName: "distributed-realm"
        path: "users.properties"
      groupProperties:
        path: "groups.properties"
    aggregateRealm:
      authenticationRealm: "trust"
      authorizationRealms:
        - "properties"
      nameRewriter:
        common-name-principal-transformer: ~
```

2.8.1. 名前リライター

プリンシパル名の形式は、セキュリティーレルムの種類に応じて異なります。

- プロパティーとトークンレルムは、単純な文字列を返す場合があります
- 信頼と LDAP レルムは X.500 形式の識別名を返す場合があります
- Kerberos レルムは **user@domain** 形式の名前を返す場合があります

次のいずれかのトランスフォーマーを使用して集約レルムを使用する場合は、名前を共通形式に正規化する必要があります。

2.8.1.1. ケースプリンシパルトランスフォーマー

case-principal-transformer は、名前をすべて大文字またはすべて小文字に変換します。

XML

```
<aggregate-realm authentication-realm="trust" authorization-realms="properties">
  <name-rewriter>
    <case-principal-transformer uppercase="false"/>
  </name-rewriter>
</aggregate-realm>
```

–

JSON

```
{
  "aggregate-realm": {
    "authentication-realm": "trust",
    "authorization-realms": [
      "properties"
    ],
    "name-rewriter": {
      "case-principal-transformer": {
        "uppercase": "false"
      }
    }
  }
}
```

YAML

```
aggregateRealm:
  authenticationRealm: "trust"
  authorizationRealms:
    - "properties"
  nameRewriter:
    casePrincipalTransformer:
      uppercase: false
```

2.8.1.2. コモンネームプリンシパルトランスフォーマー

common-name-principal-transformer は、LDAP または証明書によって使用される **DN** から最初の **CN** 要素を抽出します。たとえば、**CN=app1,CN=serviceA,OU=applications,DC=infinispan,DC=org** 形式のプリンシパルが指定されている場合、次の設定では **app1** がプリンシパルとして抽出されます。

XML

```
<aggregate-realm authentication-realm="trust" authorization-realms="properties">
  <name-rewriter>
    <common-name-principal-transformer/>
  </name-rewriter>
</aggregate-realm>
```

JSON

```
{
  "aggregate-realm": {
    "authentication-realm": "trust",
    "authorization-realms": [
      "properties"
    ],
    "name-rewriter": {
      "common-name-principal-transformer": {}
```

```
    }
  }
}
```

YAML

```
aggregateRealm:
  authenticationRealm: "trust"
  authorizationRealms:
    - "properties"
  nameRewriter:
    commonNamePrincipalTransformer: ~
```

2.8.1.3. Regex プリンシパルトランスフォーマー

regex-principal-transformer は、 正規表現を使用して検索と置換を実行できます。この例では、**user@domain.com** 識別子からローカル部分を抽出する方法を示します。

XML

```
<aggregate-realm authentication-realm="trust" authorization-realms="properties">
  <name-rewriter>
    <regex-principal-transformer pattern="([^@]+)@.*" replacement="$1" replace-all="false"/>
  </name-rewriter>
</aggregate-realm>
```

JSON

```
{
  "aggregate-realm": {
    "authentication-realm": "trust",
    "authorization-realms": [
      "properties"
    ],
    "name-rewriter": {
      "regex-principal-transformer": {
        "pattern" : "([^@]+)@.*",
        "replacement": "$1",
        "replace-all": false
      }
    }
  }
}
```

YAML

```
aggregateRealm:
  authenticationRealm: "trust"
  authorizationRealms:
    - "properties"
  nameRewriter:
    regexPrincipalTransformer:
```

```
      pattern: "([^@]+)@.*"
      replacement: "$1"
      replaceAll: false
```

2.9. セキュリティーレルムのキャッシュ

セキュリティーレルムは、通常は非常にまれに変更されるデータを繰り返し取得する必要を回避するためにキャッシュを実装します。デフォルトでは、以下のようになります。

レルムキャッシュレルム設定

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="default" cache-max-size="1024" cache-lifespan="120000">
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "default",
        "cache-max-size": 1024,
        "cache-lifespan": 120000
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "default"
        cache-max-size: 1024
        cache-lifespan: 120000
```

2.9.1. レルムキャッシュのフラッシュ

CLI を使用して、クラスター全体のセキュリティーレルムキャッシュをフラッシュします。

```
[node-1@mycluster//containers/default]> server aclcache flush
```

第3章 エンドポイント認証メカニズム

Data Grid Server は、Hot Rod および REST エンドポイントにカスタム SASL および HTTP 認証メカニズムを使用できます。

3.1. DATA GRID SERVER の認証

認証は、エンドポイントへのユーザーアクセスと、Data Grid Console およびコマンドラインインターフェイス (CLI) を制限します。

Data Grid Server には、ユーザー認証を強制するデフォルトのセキュリティーレルムが含まれます。デフォルトの認証は、**server/conf/users.properties** ファイルに保存されているユーザー認証情報とともにプロパティーレルムを使用します。Data Grid Server はデフォルトでセキュリティー認証も有効にするため、**server/conf/groups.properties** ファイルに保存されているパーミッションを持つユーザーを割り当てる必要があります。

ヒント

コマンドラインインターフェイス (CLI) で **user create** コマンドを使用して、ユーザーを追加し、パーミッションを割り当てます。サンプルおよび詳細情報について **user create --help** を実行します。

3.2. DATA GRID SERVER の認証メカニズムの設定

特定の認証メカニズムを使用するように Hot Rod および REST エンドポイントを明示的に設定することができます。認証メカニズムの設定は、セキュリティーレルムのデフォルトメカニズムを明示的に上書きする必要がある場合にのみ必要です。

注記

設定の各 **endpoint** セクションには、**hotrod-connector** および **rest-connector** 要素またはフィールドが含まれている必要があります。たとえば、**hotrod-connector** を明示的に宣言する場合は、認証メカニズムを設定しない場合でも **rest-connector** も宣言する必要があります。

前提条件

- 必要に応じて、Data Grid Server 設定にセキュリティーレルムを追加します。

手順

1. Data Grid Server 設定を開いて編集します。
2. **endpoint** 要素またはフィールドを追加し、**security-realm** 属性で使用するセキュリティーレルムを指定します。
3. **hotrod-connector** 要素またはフィールドを追加して、Hot Rod エンドポイントを設定します。
   a. **authentication** 要素またはフィールドを追加します。
   b. **sasl mechanism** 属性で使用する Hot Rod エンドポイントの SASL 認証メカニズムを指定します。
   c. 該当する場合は、**qop** 属性で SASL 品質の保護設定を指定します。

d. 必要に応じて **server-name** 属性を使用して Data Grid Server アイデンティティーを指定します。

4. **rest-connector** 要素またはフィールドを追加して REST エンドポイントを設定します。

a. **authentication** 要素またはフィールドを追加します。

b. **mechanism** 属性で使用する REST エンドポイントの HTTP 認証メカニズムを指定します。

5. 変更を設定に保存します。

認証メカニズムの設定

以下の設定では、Hot Rod エンドポイントが認証に使用する SASL メカニズムを指定します。

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <endpoints>
    <endpoint socket-binding="default"
              security-realm="my-realm">
      <hotrod-connector>
        <authentication>
          <sasl mechanisms="SCRAM-SHA-512 SCRAM-SHA-384 SCRAM-SHA-256
                            SCRAM-SHA-1 DIGEST-SHA-512 DIGEST-SHA-384
                            DIGEST-SHA-256 DIGEST-SHA DIGEST-MD5 PLAIN"
                server-name="infinispan"
                qop="auth"/>
        </authentication>
      </hotrod-connector>
      <rest-connector>
        <authentication mechanisms="DIGEST BASIC"/>
      </rest-connector>
    </endpoint>
  </endpoints>
</server>
```

JSON

```
{
  "server": {
    "endpoints": {
      "endpoint": {
        "socket-binding": "default",
        "security-realm": "my-realm",
        "hotrod-connector": {
          "authentication": {
            "security-realm": "default",
            "sasl": {
              "server-name": "infinispan",
              "mechanisms": ["SCRAM-SHA-512", "SCRAM-SHA-384", "SCRAM-SHA-256", "SCRAM-
SHA-1", "DIGEST-SHA-512", "DIGEST-SHA-384", "DIGEST-SHA-256", "DIGEST-SHA", "DIGEST-
MD5", "PLAIN"],
              "qop": ["auth"]
            }
          }
```

```
        },
        "rest-connector": {
          "authentication": {
            "mechanisms": ["DIGEST", "BASIC"],
            "security-realm": "default"
          }
        }
      }
    }
  }
}
```

YAML

```
server:
  endpoints:
    endpoint:
      socketBinding: "default"
      securityRealm: "my-realm"
      hotrodConnector:
        authentication:
          securityRealm: "default"
          sasl:
            serverName: "infinispan"
            mechanisms:
              - "SCRAM-SHA-512"
              - "SCRAM-SHA-384"
              - "SCRAM-SHA-256"
              - "SCRAM-SHA-1"
              - "DIGEST-SHA-512"
              - "DIGEST-SHA-384"
              - "DIGEST-SHA-256"
              - "DIGEST-SHA"
              - "DIGEST-MD5"
              - "PLAIN"
            qop:
              - "auth"
      restConnector:
        authentication:
          mechanisms:
            - "DIGEST"
            - "BASIC"
          securityRealm: "default"
```

3.2.1. 認証の無効化

ローカル開発環境または分離されたネットワークでは、認証されていないクライアント要求を許可するように Data Grid を設定できます。ユーザー認証を無効にする場合は、Data Grid セキュリティー設定で承認も無効にする必要があります。

手順

1. Data Grid Server 設定を開いて編集します。

2. **endpoints** 要素またはフィールドから **security-realm** 属性を削除します。
3. **cache-container** および各キャッシュ設定の **security** 設定から、**authorization** 要素をすべて削除します。
4. 変更を設定に保存します。

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <endpoints socket-binding="default"/>
</server>
```

JSON

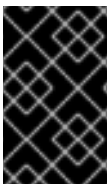
```
{
  "server": {
    "endpoints": {
      "endpoint": {
        "socket-binding": "default"
      }
    }
  }
}
```

YAML

```
server:
  endpoints:
    endpoint:
      socketBinding: "default"
```

3.3. DATA GRID SERVER の認証メカニズム

Data Grid Server は、セキュリティーレルム設定に一致する認証メカニズムでエンドポイントを自動的に設定します。たとえば、Kerberos セキュリティーレルムを追加すると、Data Grid Server は Hot Rod エンドポイントの **GSSAPI** および **GS2-KRB5** 認証メカニズムを有効にします。



重要

現在、Lightweight Directory Access Protocol (LDAP) プロトコルと **DIGEST** または **SCRAM** 認証メカニズムは、特定のハッシュ化されたパスワードにアクセスする必要があるため、使用できません。

ホットローテーションエンドポイント

Data Grid Server は、設定に対応するセキュリティーレルムが含まれている場合に Hot Rod エンドポイントの以下の SASL 認証メカニズムを有効にします。

セキュリティーレルム	SASL 認証メカニズム
プロパティーレルムおよび LDAP レルム	**SCRAM**、**DIGEST**
トークンレルム	**OAUTHBEARER**
信頼レルム	**EXTERNAL**
Kerberos ID	**GSSAPI**、**GS2-KRB5**
SSL/TLS ID	**PLAIN**

REST エンドポイント

Data Grid Server は、設定に対応するセキュリティーレルムが含まれている場合に REST エンドポイントの以下の HTTP 認証メカニズムを有効にします。

セキュリティーレルム	HTTP 認証メカニズム
プロパティーレルムおよび LDAP レルム	**DIGEST**
トークンレルム	**BEARER_TOKEN**
信頼レルム	**CLIENT_CERT**
Kerberos ID	**SPNEGO**
SSL/TLS ID	**BASIC**

Memcached エンドポイント

Data Grid Server は、設定に対応するセキュリティーレルムが含まれている場合、Memcached バイナリープロトコルエンドポイントに対して次の SASL 認証メカニズムを有効にします。

セキュリティーレルム	SASL 認証メカニズム
プロパティーレルムおよび LDAP レルム	**SCRAM**、**DIGEST**
トークンレルム	**OAUTHBEARER**
信頼レルム	**EXTERNAL**
Kerberos ID	**GSSAPI**、**GS2-KRB5**
SSL/TLS ID	**PLAIN**

Data Grid Server は、パスワードベースの認証をサポートするセキュリティーレルムでのみ、Memcached テキストプロトコルエンドポイントでの認証を有効にします。

セキュリティーレルム	Memcached テキスト認証
プロパティーレルムおよび LDAP レルム	はい
トークンレルム	いいえ
信頼レルム	いいえ
Kerberos ID	いいえ
SSL/TLS ID	いいえ

RESP エンドポイント

Data Grid Server は、パスワードベースの認証をサポートするセキュリティーレルムでのみ、RESP エンドポイントでの認証を有効にします。

セキュリティーレルム	RESP 認証
プロパティーレルムおよび LDAP レルム	はい
トークンレルム	いいえ
信頼レルム	いいえ
Kerberos ID	いいえ
SSL/TLS ID	いいえ

3.3.1. SASL 認証メカニズム

Data Grid Server は、Hot Rod および Memcached バイナリープロトコルエンドポイントを使用して、次の SASL 認証メカニズムをサポートします。

認証メカニズム	説明	セキュリティーレルムタイプ	関連する詳細
PLAIN	プレーンテキスト形式の認証情報を使用します。**PLAIN** 認証は、暗号化された接続でのみ使用する必要があります。	プロパティーレルムおよび LDAP レルム	**BASIC** HTTP メカニズムと同様です。

認証メカニズム	説明	セキュリティーレルムタイプ	関連する詳細
DIGEST-*	ハッシュアルゴリズムとナンス値を使用します。ホットロッドコネクターは、強度の順に、**DIGEST-MD5**、**DIGEST-SHA**、**DIGEST-SHA-256**、**DIGEST-SHA-384**、および **DIGEST-SHA-512** ハッシュアルゴリズムをサポートします。	プロパティーレルムおよび LDAP レルム	**Digest** HTTP メカニズムに似ています。
SCRAM-*	ハッシュアルゴリズムとナンス値に加えて**ソルト**値を使用します。ホットロッドコネクターは、**SCRAM-SHA**、**SCRAM-SHA-256**、**SCRAM-SHA-384**、および **SCRAM-SHA-512** ハッシュアルゴリズムを強度順にサポートします。	プロパティーレルムおよび LDAP レルム	**Digest** HTTP メカニズムに似ています。
GSSAPI	Kerberos チケットを使用し、Kerberos ドメインコントローラーが必要です。対応する **Kerberos** サーバー ID をレルム設定に追加する必要があります。ほとんどの場合、ユーザーメンバーシップ情報を提供するために **ldap-realm** も指定します。	Kerberos レルム	**SPNEGO** HTTP メカニズムに似ています。
GS2-KRB5	Kerberos チケットを使用し、Kerberos ドメインコントローラーが必要です。対応する **Kerberos** サーバー ID をレルム設定に追加する必要があります。ほとんどの場合、ユーザーメンバーシップ情報を提供するために **ldap-realm** も指定します。	Kerberos レルム	**SPNEGO** HTTP メカニズムに似ています。

認証メカニズム	説明	セキュリティーレルムタイプ	関連する詳細
EXTERNAL	クライアント証明書を使用します。	トラストストアレルム	**CLIENT_CERTHTTP** メカニズムに似ています。
OAUTHBEARER	OAuth トークンを使用し、**token-realm** 設定が必要です。	トークンレルム	**BEARER_TOKEN** HTTP メカニズムに似ています。

3.3.2. SASL Quality of Protection (QoP)

SASL メカニズムが整合性およびプライバシー保護 (QoP) 設定をサポートする場合は、**qop** 属性を使用して Hot Rod および Memcached エンドポイント設定に追加できます。

QoP 設定	説明
auth	認証のみ。
auth-int	整合性保護による認証。
auth-conf	整合性とプライバシー保護による認証。

3.3.3. SASL ポリシー

SASL ポリシーは、Hot Rod および Memcached 認証メカニズムをきめ細かく制御します。

ヒント

Data Grid のキャッシュ承認では、ロールおよびパーミッションに基づいてキャッシュへのアクセスを制限します。キャッシュ認証を設定し、**<no-anonymous value=false />** を設定して匿名ログインを許可し、アクセスロジックをキャッシュ承認に委譲します。

ポリシー	説明	デフォルト値
forward-secrecy	セッション間の forward secrecy をサポートする SASL メカニズムのみを使用します。これは、1 つのセッションに分割しても、将来のセッションに分割するための情報が自動的に提供されないことを意味します。	false
pass-credentials	クライアント認証情報が必要な SASL メカニズムのみを使用してください。	false

ポリシー	説明	デフォルト値
no-plain-text	単純な受動的攻撃の影響を受けやすい SASL メカニズムは使用しないでください。	false
no-active	アクティブな非辞書攻撃の影響を受けやすい SASL メカニズムは使用しないでください。	false
no-dictionary	受動的な辞書攻撃の影響を受けやすい SASL メカニズムは使用しないでください。	false
no-anonymous	匿名ログインを許可する SASL メカニズムは使用しないでください。	true

SASL ポリシーの設定
以下の設定では、Hot Rod エンドポイントは、すべての SASL ポリシーに準拠する唯一のメカニズムであるため、認証に **GSSAPI** メカニズムを使用します。

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <endpoints>
    <endpoint socket-binding="default"
              security-realm="default">
      <hotrod-connector>
        <authentication>
          <sasl mechanisms="PLAIN DIGEST-MD5 GSSAPI EXTERNAL"
                server-name="infinispan"
                qop="auth"
                policy="no-active no-plain-text"/>
        </authentication>
      </hotrod-connector>
      <rest-connector/>
    </endpoint>
  </endpoints>
</server>
```

JSON

```
{
  "server": {
    "endpoints" : {
      "endpoint" : {
        "socket-binding" : "default",
        "security-realm" : "default",
        "hotrod-connector" : {
```

```
          "authentication" : {
            "sasl" : {
              "server-name" : "infinispan",
              "mechanisms" : [ "PLAIN","DIGEST-MD5","GSSAPI","EXTERNAL" ],
              "qop" : [ "auth" ],
              "policy" : [ "no-active","no-plain-text" ]
            }
          }
        },
        "rest-connector" : ""
      }
    }
  }
}
```

YAML

```
server:
  endpoints:
    endpoint:
      socketBinding: "default"
      securityRealm: "default"
      hotrodConnector:
        authentication:
          sasl:
            serverName: "infinispan"
            mechanisms:
              - "PLAIN"
              - "DIGEST-MD5"
              - "GSSAPI"
              - "EXTERNAL"
            qop:
              - "auth"
            policy:
              - "no-active"
              - "no-plain-text"
      restConnector: ~
```

3.3.4. HTTP 認証メカニズム

Data Grid Server は、REST エンドポイントに以下の HTTP 認証メカニズムをサポートします。

認証メカニズム	説明	セキュリティーレルムタイプ	関連する詳細
BASIC	プレーンテキスト形式の認証情報を使用します。暗号化された接続でのみ **BASIC** 認証を使用する必要があります。	プロパティーレルムおよび LDAP レルム	HTTP **Basic** HTTP 認証方式に対応し、**PLAIN** SASL メカニズムと同様です。

認証メカニズム	説明	セキュリティーレルムタイプ	関連する詳細
DIGEST	ハッシュアルゴリズムとナンス値を使用します。REST コネクターは、**SHA-512**、**SHA-256**、および **MD5** ハッシュアルゴリズムをサポートします。	プロパティーレルムおよび LDAP レルム	**Digest** HTTP 認証スキームに対応し、**DIGEST-*** SASL メカニズムに似ています。
SPNEGO	Kerberos チケットを使用し、Kerberos ドメインコントローラーが必要です。対応する **Kerberos** サーバー ID をレルム設定に追加する必要があります。ほとんどの場合、ユーザーメンバーシップ情報を提供するために **ldap-realm** も指定します。	Kerberos レルム	**Negotiate** HTTP 認証スキームに対応し、**GSSAPI** および **GS2-KRB5SASL** メカニズムに類似しています。
BEARER_TOKEN	OAuth トークンを使用し、**token-realm** 設定が必要です。	トークンレルム	**Bearer** HTTP 認証スキームに対応し、**OAUTHBEARER SASL** メカニズムに似ています。
CLIENT_CERT	クライアント証明書を使用します。	トラストストアレルム	**EXTERNAL** SASL メカニズムに似ています。

第4章 TLS/SSL 暗号化の設定

Data Grid の公開鍵と秘密鍵が含まれるキーストアを設定することにより、SSL/TLS 暗号化を使用して Data Grid Server の接続をセキュアにすることができます。相互 TLS が必要な場合、クライアント証明書認証を設定することもできます。

4.1. DATA GRID SERVER キーストアの設定

キーストアを Data Grid Server に追加し、その ID をクライアントに対して検証する SSL/TLS 証明書を提示します。セキュリティーレルムに TLS/SSL アイデンティティーが含まれる場合は、そのセキュリティーレルムを使用する Data Grid Server エンドポイントへの接続を暗号化します。

前提条件

- Data Grid Server の証明書または証明書チェーンが含まれるキーストアを作成します。

Data Grid Server は、JKS、JCEKS、PKCS12/PFX、および PEM のキーストア形式をサポートします。Bouncy Castle ライブラリーが存在する場合は、BKS、BCFKS、および UBER もサポートされます。

証明書には、RFC 2818 仕様で定義されたルールに従って、クライアントがホスト名の検証を正しく実行できるように、**dNSName** および/または **iPAddress** タイプの **subjectAltName** 拡張が含まれている必要があります。このような拡張子を含まない証明書を使用してサーバーを起動すると、警告が発行されます。

重要

実稼働環境では、サーバー証明書は Root または Intermediate CA のいずれかの信頼される認証局によって署名される必要があります。

ヒント

以下のいずれかが含まれる場合には、PEM ファイルをキーストアとして使用できます。

- PKCS#1 または PKCS#8 形式の秘密鍵。
- 1 つ以上の証明書。

PEM ファイルキーストアを空のパスワード (**password=""**) で設定する必要があります。

手順

1. Data Grid Server 設定を開いて編集します。
2. Data Grid Server の SSL/TLS アイデンティティーが含まれるキーストアを **$RHDG_HOME/server/conf** ディレクトリーに追加します。
3. **server-identities** 定義を Data Grid Server セキュリティーレルムに追加します。
4. **path** 属性でキーストアファイル名を指定します。
5. キーストアパスワードと証明書エイリアスに **keystore-password** および **alias** 属性を指定します。
6. 変更を設定に保存します。

次のステップ

クライアントが Data Grid Server の SSL/TLS ID を確認できるように、トラストストアを使用してクライアントを設定します。

キーストアの設定

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="default">
        <server-identities>
          <ssl>
            <!-- Adds a keystore that contains server certificates that provide SSL/TLS identities to clients.
-->
            <keystore path="server.p12"
                      relative-to="infinispan.server.config.path"
                      password="secret"
                      alias="my-server"/>
          </ssl>
        </server-identities>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "default",
        "server-identities": {
          "ssl": {
            "keystore": {
              "alias": "my-server",
              "path": "server.p12",
              "password": "secret"
            }
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "default"
```

```
    serverIdentities:
      ssl:
        keystore:
          alias: "my-server"
          path: "server.p12"
          password: "secret"
```

関連情報

- Hot Rod クライアントの暗号化の設定

4.1.1. Data Grid Server キーストアの生成

起動時にキーストアを自動的に生成するように Data Grid Server を設定します。

重要

自動生成されたキーストア:

- 実稼働環境では使用しないでください。
- 必要に応じて生成されます。たとえば、クライアントから最初の接続を取得する際などに生成されます。
- Hot Rod クライアントで直接使用可能な証明書が含まれます。

手順

1. Data Grid Server 設定を開いて編集します。
2. サーバー設定に **keystore** 要素の **generate-self-signed-certificate-host** 属性を含めます。
3. サーバー証明書のホスト名を値として指定します。
4. 変更を設定に保存します。

生成されたキーストアの設定

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="generated-keystore">
        <server-identities>
          <ssl>
            <!-- Generates a keystore that includes a self-signed certificate with the specified hostname. --
>
            <keystore path="server.p12"
                      relative-to="infinispan.server.config.path"
                      password="secret"
                      alias="server"
                      generate-self-signed-certificate-host="localhost"/>
          </ssl>
        </server-identities>
```

```
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "generated-keystore",
        "server-identities": {
          "ssl": {
            "keystore": {
              "alias": "server",
              "generate-self-signed-certificate-host": "localhost",
              "path": "server.p12",
              "password": "secret"
            }
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "generated-keystore"
        serverIdentities:
          ssl:
            keystore:
              alias: "server"
              generateSelfSignedCertificateHost: "localhost"
              path: "server.p12"
              password: "secret"
```

4.1.2. TLS バージョンおよび暗号スイートの設定

SSL/TLS 暗号化を使用してデプロイメントのセキュリティーを保護する場合は、特定のバージョンの TLS プロトコルと、プロトコル内の特定の暗号スイートを使用するように Data Grid Server を設定できます。

手順

1. Data Grid Server 設定を開いて編集します。
2. **engine** 要素を Data Grid Server の SSL 設定に追加します。

3. **enabled-protocols** 属性を持つ 1 つ以上の TLS バージョンを使用するように Data Grid を設定します。
   Data Grid Server は、デフォルトで TLS バージョン 1.2 および 1.3 をサポートします。該当する場合は、クライアント接続のセキュリティープロトコルを制限するために、**TLSv1.3** のみを設定できます。Data Grid は、**TLSv1.1** の有効化を推奨していません。これは、サポートが制限された古いプロトコルで、セキュリティー保護が弱いからです。1.1 より古いバージョンの TLS を有効にすることはできません。

警告

Data Grid Server の SSL **engine** 設定を変更する場合は、**enabled-protocols** 属性を使用して TLS バージョンを明示的に設定する必要があります。**enabled-protocols** 属性を省略すると、すべての TLS バージョンが許可されます。

```
<engine enabled-protocols="TLSv1.3 TLSv1.2" />
```

4. **enabled-ciphersuites** 属性 (TLSv1.2 以下) および **enabled-ciphersuites-tls13** 属性 (TLSv1.3) を使用して、1 つまたは複数の暗号スイートを使用するように Data Grid を設定します。
   使用する予定のプロトコル機能 (例: **HTTP/2 ALPN**) をサポートする暗号スイートを設定していることを確認する必要があります。

5. 変更を設定に保存します。

SSL エンジンの設定

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="default">
        <server-identities>
          <ssl>
            <keystore path="server.p12"
                      relative-to="infinispan.server.config.path"
                      password="secret"
                      alias="server"/>
            <!-- Configures Data Grid Server to use specific TLS versions and cipher suites. -->
            <engine enabled-protocols="TLSv1.3 TLSv1.2"
                    enabled-
ciphersuites="TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_128_CBC_
SHA256"
                    enabled-ciphersuites-
tls13="TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_S
HA256"/>
          </ssl>
        </server-identities>
      </security-realm>
```

```
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "default",
        "server-identities": {
          "ssl": {
            "keystore": {
              "alias": "server",
              "path": "server.p12",
              "password": "secret"
            },
            "engine": {
              "enabled-protocols": ["TLSv1.3"],
              "enabled-ciphersuites":
"TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_128_CBC_SHA256",
              "enabled-ciphersuites-tls13":
"TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA25
6"
            }
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "default"
        serverIdentities:
          ssl:
            keystore:
              alias: "server"
              path: "server.p12"
              password: "secret"
            engine:
              enabledProtocols:
                - "TLSv1.3"
              enabledCiphersuites: "TLS_AES_256_GCM_SHA384,TLS_AES_128_GCM_SHA256"
              enabledCiphersuitesTls13: "TLS_AES_256_GCM_SHA384"
```

4.2. FIPS 140-2 準拠の暗号を使用するシステムでの DATA GRID SERVER の設定

FIPS (Federal Information Processing Standards) とは、米国連邦政府のコンピューターシステムの標準およびガイドラインです。FIPS は米国連邦政府が使用するために開発されたものですが、民間部門の多くは自発的にこれらの標準を使用しています。

FIPS 140-2 は、暗号モジュールに対するセキュリティー要件を定義しています。代替の JDK セキュリティープロバイダーを使用することで、FIPS 140-2 仕様に準拠する暗号化方式を使用するように Data Grid Server を設定することができます。

関連情報

- Java PKCS#11 暗号化プロバイダー
- The Legion of the Bouncy Castle cryptographic provider

4.2.1. PKCS11 暗号プロバイダーの設定

SunPKCS11-NSS-FIPS プロバイダーで PKCS11 キーストアを指定すると、PKCS11 暗号化プロバイダーを設定できます。

前提条件

- FIPS モード用にシステムを設定する。システムが FIPS モードを有効にしているかどうかは、Data Grid のコマンドラインインターフェイス (CLI) で **fips-mode-setup --check** コマンドを発行することで確認できます。
- **certutil** ツールを使用して、システム全体の NSS データベースを初期化します。
- **SunPKCS11** プロバイダーを有効にするように **java.security** ファイルを設定した JDK をインストールします。このプロバイダーは、NSS データベースと SSL プロバイダーを指します。
- NSS データベースに証明書をインストールします。

手順

1. Data Grid Server 設定を開いて編集します。
2. **server-identities** 定義を Data Grid Server セキュリティーレルムに追加します。
3. **SunPKCS11-NSS-FIPS** プロバイダーで PKCS11 キーストアを指定します。
4. 変更を設定に保存します。

キーストアの設定

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="default">
        <server-identities>
          <ssl>
            <!-- Adds a keystore that reads certificates from the NSS database. -->
            <keystore provider="SunPKCS11-NSS-FIPS" type="PKCS11"/>
          </ssl>
```

```
            </server-identities>
          </security-realm>
        </security-realms>
      </security>
    </server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "default",
        "server-identities": {
          "ssl": {
            "keystore": {
              "provider": "SunPKCS11-NSS-FIPS",
              "type": "PKCS11"
            }
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "default"
        serverIdentities:
          ssl:
            keystore:
              provider: "SunPKCS11-NSS-FIPS"
              type: "PKCS11"
```

4.2.2. Bouncy Castle FIPS 暗号プロバイダーの設定

Bouncy Castle FIPS (Federal Information Processing Standards) 暗号化プロバイダーは、Data Grid サーバーの設定で設定することができます。

前提条件

- FIPS モード用にシステムを設定する。システムが FIPS モードを有効にしているかどうかは、Data Grid のコマンドラインインターフェイス (CLI) で **fips-mode-setup --check** コマンドを発行することで確認できます。
- 証明書を含む BCFKS 形式のキーストアを作成します。

手順

1. Bouncy Castle FIPS JAR ファイルをダウンロードし、Data Grid Server のインストール先の **server/lib** ディレクトリーにファイルを追加してください。
2. Bouncy Castle をインストールするには、**install** コマンドを実行します。

```
[disconnected]> install org.bouncycastle:bc-fips:1.0.2.3
```

3. Data Grid Server 設定を開いて編集します。
4. **server-identities** 定義を Data Grid Server セキュリティーレルムに追加します。
5. **BCFIPS** プロバイダーで BCFKS キーストアを指定します。
6. 変更を設定に保存します。

キーストアの設定

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="default">
        <server-identities>
          <ssl>
            <!-- Adds a keystore that reads certificates from the BCFKS keystore. -->
            <keystore path="server.bcfks" password="secret" alias="server" provider="BCFIPS"
type="BCFKS"/>
          </ssl>
        </server-identities>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "default",
        "server-identities": {
          "ssl": {
            "keystore": {
              "path": "server.bcfks",
              "password": "secret",
              "alias": "server",
              "provider": "BCFIPS",
              "type": "BCFKS"
            }
          }
        }
      }]
```

```
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "default"
        serverIdentities:
          ssl:
            keystore:
              path: "server.bcfks"
              password: "secret"
              alias: "server"
              provider: "BCFIPS"
              type: "BCFKS"
```

4.3. クライアント証明書認証の設定

Data Grid Server が相互 TLS を使用してクライアント接続のセキュリティーを保護するように設定します。

トラストストアの証明書からクライアント ID を検証するように Data Grid を設定するには、以下の 2 つの方法があります。

- 通常は認証局 (CA) である署名証明書のみが含まれるトラストストアが必要です。CA によって署名された証明書を提示するクライアントは、Data Grid に接続できます。
- 署名証明書に加えて、すべてのクライアント証明書が含まれるトラストストアが必要です。トラストストアに存在する署名済み証明書を提示するクライアントのみが Data Grid に接続できます。

ヒント

トラストストアを提供する代わりに、共有システム証明書を使用できます。

前提条件

- CA 証明書またはすべての公開証明書のいずれかを含むクライアントトラストストアを作成します。
- Data Grid Server のキーストアを作成し、SSL/TLS アイデンティティーを設定します。

注記

PEM ファイルは、1 つ以上の証明書が含まれるトラストストアとして使用できます。これらのトラクトストアは、空のパスワード **password=""** で設定する必要があります。

手順

1. Data Grid Server 設定を開いて編集します。

2. **require-ssl-client-auth="true"** パラメーターを **endpoints** 設定に追加します。
3. クライアントトラストストアを **$RHDG_HOME/server/conf** ディレクトリーに追加します。
4. Data Grid Server セキュリティーレルム設定で、**truststore** 要素の **path** および **password** 属性を指定します。
5. Data Grid Server で各クライアント証明書を認証する場合は、**<truststore-realm/>** 要素をセキュリティーレルムに追加します。
6. 変更を設定に保存します。

次のステップ

- セキュリティーロールおよびパーミッションでアクセスを制御する場合は、Data Grid Server 設定で、クライアント証明書を使用して承認を設定します。
- クライアントを設定し、Data Grid Server と SSL/TLS 接続をネゴシエートします。

クライアント証明書認証設定

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="trust-store-realm">
        <server-identities>
          <ssl>
            <!-- Provides an SSL/TLS identity with a keystore that
                 contains server certificates. -->
            <keystore path="server.p12"
                      relative-to="infinispan.server.config.path"
                      keystore-password="secret"
                      alias="server"/>
            <!-- Configures a trust store that contains client certificates
                 or part of a certificate chain. -->
            <truststore path="trust.p12"
                        relative-to="infinispan.server.config.path"
                        password="secret"/>
          </ssl>
        </server-identities>
        <!-- Authenticates client certificates against the trust store. If you configure this, the trust store
must contain the public certificates for all clients. -->
        <truststore-realm/>
      </security-realm>
    </security-realms>
  </security>
  <endpoints>
    <endpoint socket-binding="default"
              security-realm="trust-store-realm"
              require-ssl-client-auth="true">
      <hotrod-connector>
        <authentication>
          <sasl mechanisms="EXTERNAL"
                server-name="infinispan"
```

```
            qop="auth"/>
        </authentication>
      </hotrod-connector>
      <rest-connector>
        <authentication mechanisms="CLIENT_CERT"/>
      </rest-connector>
    </endpoint>
  </endpoints>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "trust-store-realm",
        "server-identities": {
          "ssl": {
            "keystore": {
              "path": "server.p12",
              "relative-to": "infinispan.server.config.path",
              "keystore-password": "secret",
              "alias": "server"
            },
            "truststore": {
              "path": "trust.p12",
              "relative-to": "infinispan.server.config.path",
              "password": "secret"
            }
          }
        },
        "truststore-realm": {}
      }]
    },
    "endpoints": [{
      "socket-binding": "default",
      "security-realm": "trust-store-realm",
      "require-ssl-client-auth": "true",
      "connectors": {
        "hotrod": {
          "hotrod-connector": {
            "authentication": {
              "sasl": {
                "mechanisms": "EXTERNAL",
                "server-name": "infinispan",
                "qop": "auth"
              }
            }
          },
          "rest": {
            "rest-connector": {
              "authentication": {
                "mechanisms": "CLIENT_CERT"
              }
```

```
          }
        }
      }
    }
  }]
 }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "trust-store-realm"
        serverIdentities:
          ssl:
            keystore:
              path: "server.p12"
              relative-to: "infinispan.server.config.path"
              keystore-password: "secret"
              alias: "server"
            truststore:
              path: "trust.p12"
              relative-to: "infinispan.server.config.path"
              password: "secret"
        truststoreRealm: ~
  endpoints:
    socketBinding: "default"
    securityRealm: "trust-store-realm"
    requireSslClientAuth: "true"
    connectors:
      - hotrod:
          hotrodConnector:
            authentication:
              sasl:
                mechanisms: "EXTERNAL"
                serverName: "infinispan"
                qop: "auth"
      - rest:
          restConnector:
            authentication:
              mechanisms: "CLIENT_CERT"
```

関連情報

- Hot Rod クライアントの暗号化の設定
- Using Shared System Certificates (Red Hat Enterprise Linux 7 Security Guide)

4.4. クライアント証明書を使用した承認の設定

クライアント証明書認証を有効にすると、クライアント設定で Data Grid ユーザー認証情報を指定する必要がなくなります。つまり、ロールをクライアント証明書の Common Name (CN) フィールドに関連付ける必要があります。

前提条件

- クライアントに、公開証明書または証明書チェーンの一部 (通常は公開 CA 証明書) のいずれかが含まれる Java キーストアを提供します。
- クライアント証明書認証を実行するように Data Grid Server を設定します。

手順

1. Data Grid Server 設定を開いて編集します。
2. セキュリティー承認設定で **common-name-role-mapper** を有効にします。
3. クライアント証明書から Common Name (**CN**) に、適切な権限を持つロールを割り当てます。
4. 変更を設定に保存します。

クライアント証明書承認設定

XML

```
<infinispan>
  <cache-container name="certificate-authentication" statistics="true">
    <security>
      <authorization>
        <!-- Declare a role mapper that associates the common name (CN) field in client certificate trust stores with authorization roles. -->
        <common-name-role-mapper/>
        <!-- In this example, if a client certificate contains `CN=Client1` then clients with matching certificates get ALL permissions. -->
        <role name="Client1" permissions="ALL"/>
      </authorization>
    </security>
  </cache-container>
</infinispan>
```

JSON

```
{
  "infinispan": {
    "cache-container": {
      "name": "certificate-authentication",
      "security": {
        "authorization": {
          "common-name-role-mapper": null,
          "roles": {
            "Client1": {
              "role": {
                "permissions": "ALL"
              }
            }
          }
        }
      }
```

```
    }
  }
}
```

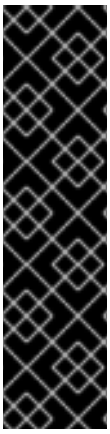
YAML

```
infinispan:
  cacheContainer:
    name: "certificate-authentication"
    security:
      authorization:
        commonNameRoleMapper: ~
        roles:
          Client1:
            role:
              permissions:
                - "ALL"
```

第5章 キーストアへの DATA GRID SERVER 認証情報の保存

外部サービスには、Data Grid Server での認証に認証情報が必要です。パスワードなどの機密なテキスト文字列を保護するには、これらを Data Grid Server 設定ファイルに直接追加するのではなく、認証情報キーストアに追加します。

次に、データベースや LDAP ディレクトリーなどのサービスと接続を確立するためのパスワードを復号化するように、Data Grid Server を設定することができます。



重要

$RHDG_HOME/server/conf のプレーンテキストのパスワードは暗号化されません。ホストファイルシステムへの読み取りアクセス権を持つすべてのユーザーアカウントは、プレーンテキストのパスワードを表示できます。

認証情報キーストアはパスワードで保護されたストア暗号化パスワードですが、ホストファイルシステムへの書き込みアクセス権を持つユーザーアカウントは、キーストア自体を改ざんすることが可能です。

Data Grid Server の認証情報を完全に保護するには、Data Grid Server を設定および実行できるユーザーアカウントにのみ読み書きアクセスを付与する必要があります。

5.1. 認証情報キーストアのセットアップ

Data Grid Server アクセスの認証情報を暗号化するキーストアを作成します。

認証情報キーストアには、暗号化されたパスワードに関連するエイリアスが少なくとも 1 つ含まれます。キーストアの作成後に、データベース接続プールなどの接続設定にエイリアスを指定します。その後、Data Grid Server は、サービスが認証を試行するときに、キーストアからそのエイリアスのパスワードを復号化します。

必要な数のエイリアスを使用して、必要な数の認証情報キーストアを作成できます。

注記

セキュリティーのベストプラクティスとして、キーストアは Data Grid Server のプロセスを実行するユーザーのみが読み取れるようにする必要があります。

手順

1. **$RHDG_HOME** でターミナルを開きます。
2. キーストアを作成し、**credentials** コマンドを使用して認証情報を追加します。

ヒント

デフォルトでは、キーストアのタイプは PKCS12 です。キーストアのデフォルトの変更に関する詳細は、**help credentials** を実行します。

次の例は、パスワード "changeme" 用に "dbpassword" のエイリアスを含むキーストアを作成する方法を示しています。キーストアの作成時に、**-p** 引数を使用してキーストアにアクセスするためのパスワードも指定します。

Linux

```
bin/cli.sh credentials add dbpassword -c changeme -p "secret1234!"
```

Microsoft Windows

```
bin\cli.bat credentials add dbpassword -c changeme -p "secret1234!"
```

3. エイリアスがキーストアに追加されていることを確認します。

```
bin/cli.sh credentials ls -p "secret1234!"
dbpassword
```

4. Data Grid Server 設定を開いて編集します。
5. 認証情報キーストアを使用するように Data Grid を設定します。
   a. **security** 設定に **credential-stores** セクションを追加します。
   b. 認証情報キーストアの名前と場所を指定します。
   c. **clear-text-credential** の設定で、認証情報キーストアにアクセスするためのパスワードを指定します。

注記

Data Grid Server 設定に認証情報キーストアのクリアテキストパスワードを追加する代わりに、外部コマンドまたはマスクされたパスワードを使用して、セキュリティーを強化することができます。

また、ある認証情報ストアのパスワードを、別の認証情報ストアのマスターパスワードとして使用することもできます。

6. Data Grid Server がデータソースや LDAP サーバーなどの外部システムとの接続に使用する設定において、認証情報キーストアを参照します。
   a. **credential-reference** セクションを追加します。
   b. **store** 属性で、認証情報キーストアの名前を指定します。
   c. **alias** 属性でパスワードのエイリアスを指定します。

ヒント

credential-reference 設定の属性はオプションです。

- **store** は、複数のキーストアがある場合にのみ必要です。
- **alias** は、キーストアに複数のパスワードエイリアスが含まれている場合にのみ必要です。

7. 変更を設定に保存します。

5.2. 認証情報キーストアのパスワードの保護

Data Grid Server では、認証情報キーストアにアクセスするためにパスワードが必要です。そのパスワードをキリアテキストで Data Grid Server の設定に追加するか、セキュリティー強化として、パスワードに外部コマンドを使用したり、パスワードをマスクしたりすることができます。

前提条件

- Data Grid Server に認証情報キーストアを設定している。

手順

次のいずれかを行います。

- **credentials mask** コマンドを使用して、パスワードが明らかにならないようにします。以下に例を示します。

  ```
  bin/cli.sh credentials mask -i 100 -s pepper99 "secret1234!"
  ```

  Masked パスワードは Password Based Encryption (PBE) を使用し、Data Grid Server 設定では次の形式である必要があります: <MASKED_VALUE;SALT;ITERATION>

- 標準出力としてパスワードを提供する外部コマンドを使用します。
  外部コマンドは、シェルスクリプトやバイナリーなど、**java.lang.Runtime#exec(java.lang.String)** を使用する任意の実行可能ファイルにすることができます。
  コマンドにパラメーターが必要な場合は、スペースで区切られた文字列のリストで指定します。

5.3. 認証情報キーストアの設定

Data Grid Server の設定に認証情報キーストアを追加し、クリアテキストのパスワード、マスクされたパスワード、またはパスワードを供給する外部コマンドを使用することができます。

クリアテキストパスワードを持つ認証情報キーストア

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <credential-stores>
      <credential-store name="credentials" path="credentials.pfx">
        <clear-text-credential clear-text="secret1234!"/>
      </credential-store>
    </credential-stores>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "credential-stores": [{
        "name": "credentials",
        "path": "credentials.pfx",
```

```
        "clear-text-credential": {
          "clear-text": "secret1234!"
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    credentialStores:
      - name: credentials
        path: credentials.pfx
        clearTextCredential:
          clearText: "secret1234!"
```

パスワードがマスクされた認証情報キーストア

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <credential-stores>
      <credential-store name="credentials"
                        path="credentials.pfx">
        <masked-credential masked="1oTMDZ5JQj6DVepJviXMnX;pepper99;100"/>
      </credential-store>
    </credential-stores>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "credential-stores": [{
        "name": "credentials",
        "path": "credentials.pfx",
        "masked-credential": {
          "masked": "1oTMDZ5JQj6DVepJviXMnX;pepper99;100"
        }
      }]
    }
  }
}
```

YAML

```
server:
```

```
  security:
    credentialStores:
      - name: credentials
        path: credentials.pfx
        maskedCredential:
          masked: "1oTMDZ5JQj6DVepJviXMnX;pepper99;100"
```

外部コマンドのパスワード

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <credential-stores>
      <credential-store name="credentials"
                        path="credentials.pfx">
        <command-credential command="/path/to/executable.sh arg1 arg2"/>
      </credential-store>
    </credential-stores>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "credential-stores": [{
        "name": "credentials",
        "path": "credentials.pfx",
        "command-credential": {
          "command": "/path/to/executable.sh arg1 arg2"
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    credentialStores:
      - name: credentials
        path: credentials.pfx
        commandCredential:
          command: "/path/to/executable.sh arg1 arg2"
```

5.4. 認証情報キーストア参照

Data Grid Server に認証情報キーストアを追加すると、接続設定でそれらを参照することができます。

データソース接続

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <credential-stores>
      <credential-store name="credentials"
                        path="credentials.pfx">
        <clear-text-credential clear-text="secret1234!"/>
      </credential-store>
    </credential-stores>
  </security>
  <data-sources>
    <data-source name="postgres"
                 jndi-name="jdbc/postgres">
      <!-- Specifies the database username in the connection factory. -->
      <connection-factory driver="org.postgresql.Driver"
                          username="dbuser"
                          url="${org.infinispan.server.test.postgres.jdbcUrl}">
        <!-- Specifies the credential keystore that contains an encrypted password and the alias for it. --
>
        <credential-reference store="credentials"
                              alias="dbpassword"/>
      </connection-factory>
      <connection-pool max-size="10"
                       min-size="1"
                       background-validation="1000"
                       idle-removal="1"
                       initial-size="1"
                       leak-detection="10000"/>
    </data-source>
  </data-sources>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "credential-stores": [{
        "name": "credentials",
        "path": "credentials.pfx",
        "clear-text-credential": {
          "clear-text": "secret1234!"
        }
      }],
      "data-sources": [{
        "name": "postgres",
        "jndi-name": "jdbc/postgres",
        "connection-factory": {
          "driver": "org.postgresql.Driver",
          "username": "dbuser",
          "url": "${org.infinispan.server.test.postgres.jdbcUrl}",
```

```
        "credential-reference": {
          "store": "credentials",
          "alias": "dbpassword"
        }
      }
    }]
  }
 }
}
```

YAML

```
server:
 security:
  credentialStores:
   - name: credentials
     path: credentials.pfx
     clearTextCredential:
       clearText: "secret1234!"
 dataSources:
  - name: postgres
    jndiName: jdbc/postgres
    connectionFactory:
      driver: org.postgresql.Driver
      username: dbuser
      url: '${org.infinispan.server.test.postgres.jdbcUrl}'
      credentialReference:
        store: credentials
        alias: dbpassword
```

LDAP 接続

XML

```
<server xmlns="urn:infinispan:server:14.0">
 <security>
  <credential-stores>
   <credential-store name="credentials"
                     path="credentials.pfx">
    <clear-text-credential clear-text="secret1234!"/>
   </credential-store>
  </credential-stores>
  <security-realms>
   <security-realm name="default">
    <!-- Specifies the LDAP principal in the connection factory. -->
    <ldap-realm name="ldap"
                url="ldap://my-ldap-server:10389"
                principal="uid=admin,ou=People,dc=infinispan,dc=org">
     <!-- Specifies the credential keystore that contains an encrypted password and the alias for it. -
->
     <credential-reference store="credentials"
                           alias="ldappassword"/>
    </ldap-realm>
   </security-realm>
```

```
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "credential-stores": [{
        "name": "credentials",
        "path": "credentials.pfx",
        "clear-text-credential": {
          "clear-text": "secret1234!"
        }
      }],
      "security-realms": [{
        "name": "default",
        "ldap-realm": {
          "name": "ldap",
          "url": "ldap://my-ldap-server:10389",
          "principal": "uid=admin,ou=People,dc=infinispan,dc=org",
          "credential-reference": {
            "store": "credentials",
            "alias": "ldappassword"
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    credentialStores:
      - name: credentials
        path: credentials.pfx
        clearTextCredential:
          clearText: "secret1234!"
    securityRealms:
      - name: "default"
        ldapRealm:
          name: ldap
          url: 'ldap://my-ldap-server:10389'
          principal: 'uid=admin,ou=People,dc=infinispan,dc=org'
          credentialReference:
            store: credentials
            alias: ldappassword
```

第6章 クラスタートランスポートの暗号化

ノードが暗号化されたメッセージと通信できるように、クラスタートランスポートを保護します。また、有効なアイデンティティーを持つノードのみが参加できるように、証明書認証を実行するように Data Grid クラスターを設定することもできます。

6.1. TLS アイデンティティーを使用したクラスタートランスポートのセキュア化

SSL/TLS アイデンティティーを Data Grid Server セキュリティーレルムに追加し、これを使用してクラスタートランスポートをセキュア化します。Data Grid Server クラスターのノードは、SSL/TLS 証明書を交換して、クロスサイトレプリケーションを設定する場合の RELAY メッセージを含む JGroups メッセージを暗号化します。

前提条件

- Data Grid Server クラスターをインストールします。

手順

1. 1 つの証明書が含まれる TLS キーストアを作成し、Data Grid Server を特定します。PKCS#1 または PKCS#8 形式の秘密鍵、証明書、および空のパスワードである **password=""** が含まれている場合は、PEM ファイルを使用することもできます。

注記

キーストアの証明書が公開認証局 (CA) で署名されていない場合は、署名証明書または公開鍵のいずれかが含まれるトラストストアを作成する必要もあります。

2. キーストアを **$RHDG_HOME/server/conf** ディレクトリーに追加します。
3. キーストアを Data Grid Server 設定の新しいセキュリティーレルムに追加します。

重要

Data Grid Server エンドポイントがクラスタートランスポートと同じセキュリティーレルムを使用しないように、専用のキーストアとセキュリティーレルムを作成する必要があります。

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="cluster-transport">
        <server-identities>
          <ssl>
            <!-- Adds a keystore that contains a certificate that provides SSL/TLS identity to
encrypt cluster transport. -->
            <keystore path="server.pfx"
                      relative-to="infinispan.server.config.path"
                      password="secret"
                      alias="server"/>
          </ssl>
```

```
      </server-identities>
    </security-realm>
  </security-realms>
 </security>
</server>
```

4. **server:security-realm** 属性でセキュリティーレルムの名前を指定して、セキュリティーレルムを使用するようにクラスタートランスポートを設定します。

```
<infinispan>
 <cache-container>
  <transport server:security-realm="cluster-transport"/>
 </cache-container>
</infinispan>
```

検証

Data Grid Server を起動すると、以下のログメッセージはクラスターがクラスタートランスポートにセキュリティーレルムを使用していることを示します。

```
[org.infinispan.SERVER] ISPN080060: SSL Transport using realm <security_realm_name>
```

6.2. JGROUPS 暗号化プロトコル

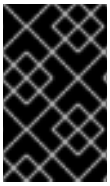
クラスタートラフィックのセキュリティーを保護するには、Data Grid ノードを設定し、シークレットキーで JGroups メッセージペイロードを暗号化します。

Data Grid ノードは、以下のいずれかから秘密鍵を取得できます。

- コーディネーターノード (非対称暗号化)
- 共有キーストア (対称暗号化)

コーディネーターノードからの秘密鍵の取得

非対称暗号化は、Data Grid 設定の JGroups スタックに **ASYM_ENCRYPT** プロトコルを追加して対称暗号化を設定します。これにより、Data Grid クラスターはシークレットキーを生成して配布できます。



重要

非対称暗号化を使用する場合は、ノードが証明書認証を実行し、シークレットキーを安全に交換できるようにキーストアを提供する必要もあります。これにより、中間者 (MitM) 攻撃からクラスターが保護されます。

非対称暗号化は、以下のようにクラスタートラフィックのセキュリティーを保護します。

1. Data Grid クラスターの最初のノードであるコーディネーターノードは、秘密鍵を生成します。
2. 参加ノードは、コーディネーターとの証明書認証を実行して、相互に ID を検証します。
3. 参加ノードは、コーディネーターノードに秘密鍵を要求します。その要求には、参加ノードの公開鍵が含まれています。
4. コーディネーターノードは、秘密鍵を公開鍵で暗号化し、参加ノードに返します。

5. 参加ノードは秘密鍵を復号してインストールします。

6. ノードはクラスターに参加し、秘密鍵でメッセージを暗号化および復号化します。

共有キーストアからの秘密鍵の取得

対称暗号化は、Data Grid 設定の JGroups スタックに **SYM_ENCRYPT** プロトコルを追加して対称暗号化を設定します。これにより、Data Grid クラスターは、指定したキーストアから秘密鍵を取得できます。

1. ノードは、起動時に Data Grid クラスパスのキーストアから秘密鍵をインストールします。

2. ノードはクラスターに参加し、秘密鍵でメッセージを暗号化および復号化します。

非対称暗号化と対称暗号化の比較

証明書認証を持つ **ASYM_ENCRYPT** は、**SYM_ENCRYPT** と比較して、暗号化の追加の層を提供します。秘密鍵のコーディネーターノードへのリクエストを暗号化するキーストアを提供します。Data Grid は、そのシークレットキーを自動的に生成し、クラスタートラフィックを処理し、秘密鍵の生成時に指定します。たとえば、ノードが離れる場合に新規のシークレットキーを生成するようにクラスターを設定できます。これにより、ノードが証明書認証を回避して古いキーで参加できなくなります。

一方、**SYM_ENCRYPT** は **ASYM_ENCRYPT** よりも高速です。ノードがクラスターコーディネーターとキーを交換する必要がないためです。**SYM_ENCRYPT** への潜在的な欠点は、クラスターのメンバーシップの変更時に新規シークレットキーを自動的に生成するための設定がないことです。ユーザーは、ノードがクラスタートラフィックを暗号化するのに使用するシークレットキーを生成して配布する必要があります。

6.3. 非対称暗号化を使用したクラスタートランスポートのセキュア化

Data Grid クラスターを設定し、JGroups メッセージを暗号化するシークレットキーを生成して配布します。

手順

1. Data Grid がノードの ID を検証できるようにする証明書チェーンでキーストアを作成します。

2. クラスター内の各ノードのクラスパスにキーストアを配置します。
   Data Grid Server の場合は、$RHDG_HOME ディレクトリーにキーストアを配置します。

3. 以下の例のように、**SSL_KEY_EXCHANGE** プロトコルおよび **ASYM_ENCRYPT** プロトコルを Data Grid 設定の JGroups スタックに追加します。

```
<infinispan>
  <jgroups>
    <!-- Creates a secure JGroups stack named "encrypt-tcp" that extends the default TCP
stack. -->
    <stack name="encrypt-tcp" extends="tcp">
      <!-- Adds a keystore that nodes use to perform certificate authentication. -->
      <!-- Uses the stack.combine and stack.position attributes to insert
SSL_KEY_EXCHANGE into the default TCP stack after VERIFY_SUSPECT2. -->
      <SSL_KEY_EXCHANGE keystore_name="mykeystore.jks"
                        keystore_password="changeit"
                        stack.combine="INSERT_AFTER"
                        stack.position="VERIFY_SUSPECT2"/>
      <!-- Configures ASYM_ENCRYPT -->
```

```
    <!-- Uses the stack.combine and stack.position attributes to insert ASYM_ENCRYPT into
the default TCP stack before pbcast.NAKACK2. -->
    <!-- The use_external_key_exchange = "true" attribute configures nodes to use the
`SSL_KEY_EXCHANGE` protocol for certificate authentication. -->
    <ASYM_ENCRYPT asym_keylength="2048"
                  asym_algorithm="RSA"
                  change_key_on_coord_leave = "false"
                  change_key_on_leave = "false"
                  use_external_key_exchange = "true"
                  stack.combine="INSERT_BEFORE"
                  stack.position="pbcast.NAKACK2"/>
  </stack>
 </jgroups>
 <cache-container name="default" statistics="true">
  <!-- Configures the cluster to use the JGroups stack. -->
  <transport cluster="${infinispan.cluster.name}"
             stack="encrypt-tcp"
             node-name="${infinispan.node.name:}"/>
 </cache-container>
</infinispan>
```

検証

Data Grid クラスターを起動した際、以下のログメッセージは、クラスターがセキュアな JGroups スタックを使用していることを示しています。

```
[org.infinispan.CLUSTER] ISPN000078: Starting JGroups channel cluster with stack
<encrypted_stack_name>
```

Data Grid ノードは **ASYM_ENCRYPT** を使用している場合のみクラスターに参加でき、コーディネーターノードからシークレットキーを取得できます。それ以外の場合は、次のメッセージが Data Grid ログに書き込まれます。

```
[org.jgroups.protocols.ASYM_ENCRYPT] <hostname>: received message without encrypt header
from <hostname>; dropping it
```

関連情報

- JGroups 4 Manual
- JGroups 4.2 Schema

6.4. 対称暗号化を使用したクラストートランスポートのセキュア化

指定したキーストアからの秘密鍵を使用して JGroups メッセージを暗号化するように Data Grid クラスターを設定します。

手順

1. シークレットキーが含まれるキーストアを作成します。
2. クラスター内の各ノードのクラスパスにキーストアを配置します。
   Data Grid Server の場合は、$RHDG_HOME ディレクトリーにキーストアを配置します。

3. Data Grid 設定の JGroups スタックに **SYM_ENCRYPT** プロトコルを追加します。

```
<infinispan>
 <jgroups>
  <!-- Creates a secure JGroups stack named "encrypt-tcp" that extends the default TCP stack. -->
  <stack name="encrypt-tcp" extends="tcp">
   <!-- Adds a keystore from which nodes obtain secret keys. -->
   <!-- Uses the stack.combine and stack.position attributes to insert SYM_ENCRYPT into the
default TCP stack after VERIFY_SUSPECT2. -->
   <SYM_ENCRYPT keystore_name="myKeystore.p12"
                keystore_type="PKCS12"
                store_password="changeit"
                key_password="changeit"
                alias="myKey"
                stack.combine="INSERT_AFTER"
                stack.position="VERIFY_SUSPECT2"/>
  </stack>
 </jgroups>
 <cache-container name="default" statistics="true">
  <!-- Configures the cluster to use the JGroups stack. -->
  <transport cluster="${infinispan.cluster.name}"
             stack="encrypt-tcp"
             node-name="${infinispan.node.name:}"/>
 </cache-container>
</infinispan>
```

検証

Data Grid クラスターを起動した際、以下のログメッセージは、クラスターがセキュアな JGroups スタックを使用していることを示しています。

```
[org.infinispan.CLUSTER] ISPN000078: Starting JGroups channel cluster with stack
<encrypted_stack_name>
```

Data Grid ノードは、**SYM_ENCRYPT** を使用し、共有キーストアからシークレットキーを取得できる場合に限りクラスターに参加できます。それ以外の場合は、次のメッセージが Data Grid ログに書き込まれます。

```
[org.jgroups.protocols.SYM_ENCRYPT] <hostname>: received message without encrypt header from
<hostname>; dropping it
```

関連情報

- JGroups 4 Manual
- JGroups 4.2 Schema

第7章 DATA GRID ポートおよびプロトコル

Data Grid は、ネットワークにデータを分散し、外部クライアント要求の接続を確立できるため、Data Grid がネットワークトラフィックを処理するために使用するポートおよびプロトコルを認識する必要があります。

Data Grid をリモートサーバーとして実行する場合は、ファイアウォールを介してリモートクライアントを許可する必要がある場合があります。同様に、競合やネットワークの問題を防ぐために、Data Grid ノードがクラスター通信に使用するポートを調整する必要があります。

7.1. DATA GRID SERVER ポートおよびプロトコル

Data Grid Server は、異なるプロトコルでのクライアントアクセスを許可するネットワークエンドポイントを提供します。

ポート	プロトコル	説明
11222	TCP	Hot Rod および REST
11221	TCP	Memcached(デフォルトでは無効)

単一ポート

Data Grid Server は、1 つの TCP ポート **11222** で複数のプロトコルを公開します。1 つのポートで複数のプロトコルを処理すると、設定が簡素化され、Data Grid クラスターをデプロイする際の管理の複雑さが軽減されます。また、1 つのポートを使用すると、ネットワーク上の攻撃対象領域が最小限に抑えられるため、セキュリティーも強化されます。

Data Grid Server は、クライアントからの HTTP/1.1、HTTP/2、および Hot Rod プロトコル要求を、さまざまな方法で単一のポートを介して処理します。

HTTP/1.1 アップグレードヘッダー

クライアントリクエストには、**HTTP/1.1 upgrade** ヘッダーフィールドを追加して、Data Grid Server と HTTP/1.1 の接続を開始できます。続いて、クライアントアプリケーションは **Upgrade: protocol** ヘッダーフィールドを送信できます。ここで、**protocol** はサーバーエンドポイントになります。

Application-Layer Protocol Negotiation (ALPN)/Transport Layer Security (TLS)

クライアント要求には、TLS 接続を介してプロトコルをネゴシエートするための Data Grid Server エンドポイントの Server Name Indication (SNI) マッピングが含まれます。

Hot Rod の自動検出

Hot Rod ヘッダーを含むクライアントリクエストは、自動的に Hot Rod エンドポイントにルーティングされます。

7.1.1. Data Grid トラフィック用のネットワークファイアウォールの設定

ファイアウォールルールを調整して、Data Grid Server とクライアントアプリケーションの間のトラフィックを許可します。

手順

Red Hat Enterprise Linux (RHEL) ワークステーションでは、たとえば、以下のように firewalld を使用してポート **11222** へのトラフィックを許可できます。

```
# firewall-cmd --add-port=11222/tcp --permanent
success
# firewall-cmd --list-ports | grep 11222
11222/tcp
```

ネットワーク全体に適用されるファイアウォールルールを設定するには、nftables ユーティリティーを使用できます。

参照

- firewalld の使用および設定
- nftables の使用

7.2. クラスタートラフィックの TCP および UDP ポート

Data Grid は、クラスタートランスポートメッセージに以下のポートを使用します。

デフォルトのポート	プロトコル	説明
7800	TCP/UDP	JGroups クラスターバインドポート
46655	UDP	JGroups マルチキャスト

クロスサイトレプリケーション
Data Grid は、JGroups RELAY2 プロトコルに以下のポートを使用します。

7900
OpenShift で実行している Data Grid クラスターの向け。

7800
ノード間のトラフィックに UDP を使用し、クラスター間のトラフィックに TCP を使用する場合。

7801
ノード間のトラフィックに TCP を使用し、クラスター間のトラフィックに TCP を使用する場合。