Maintenance

This guide describes tasks and configurations you might repeat throughout the life cycle of a deployment in your organization. It is for people who maintain PingGateway services for their organization.

Audit the deployment

The following sections describe how to set up auditing for your deployment. For information about how to include user ID in audit logs, refer to Recording User ID in Audit Events.

For information about the audit framework and each event handler, refer to Audit framework.

Record access audit events in CSV

This section describes how to record access audit events in a CSV file, using tamper-evident logging. For information about the CSV audit event handler, refer to CsvAuditEventHandler.

IMPORTANT

The CSV handler does not sanitize messages when writing to CSV log files.

Do not open CSV logs in spreadsheets or other applications that treat data as code.

Before you start, prepare PingGateway and the sample application as described in the Quick install.

1. Set up secrets for tamper-evident logging:
   a. Locate a directory for secrets, and go to it:

   ```
   $ cd /path/to/secrets
   ```

   b. Generate a key pair in the keystore.

   The CSV event handler expects a JCEKS-type keystore with a key alias of `signature` for the signing key, where the key is generated with the `RSA` key algorithm and the `SHA256withRSA` signature algorithm:

```
$ keytool \
 -genkeypair \
 -keyalg RSA \
 -sigalg SHA256withRSA \
 -alias "signature" \
 -dname "CN=ig.example.com,O=Example Corp,C=FR" \
 -keystore audit-keystore \
 -storetype JCEKS \
 -storepass password \
 -keypass password
```

NOTE

Because keytool converts all characters in its key aliases to lowercase, use only lowercase in alias definitions of a keystore.

c. Generate a secret key in the keystore.

The CSV event handler expects a JCEKS-type keystore with a key alias of `csv-key-2` for the symmetric key, where the key is generated with the `HmacSHA256` key algorithm and 256-bit key size:

```
$ keytool \
 -genseckey \
 -keyalg HmacSHA256 \
 -keysize 256 \
 -alias "password" \
 -keystore audit-keystore \
 -storetype JCEKS \
 -storepass password \
 -keypass password
```

d. Verify the content of the keystore:

```
$ keytool \
 -list \
 -keystore audit-keystore \
 -storetype JCEKS \
 -storepass password

Keystore type: JCEKS
Keystore provider: SunJCE

Your keystore contains 2 entries
```

```
password, ... SecretKeyEntry,
signature, ... PrivateKeyEntry,
Certificate fingerprint (SHA1): 4D:...:D1
```

2. Set up PingGateway

   a. Set up PingGateway for HTTPS, as described in Configure PingGateway for TLS (server-side).

   b. Add the following route to PingGateway, replacing `/path/to/secrets/audit-keystore` with your path:

Linux | Windows

```
$HOME/.openig/config/routes/30-csv.json
```

```
{
  "name": "30-csv",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/csv-
audit')}",
  "heap": [
    {
      "name": "AuditService",
      "type": "AuditService",
      "config": {
        "eventHandlers": [
          {
            "class":
"org.forgerock.audit.handlers.csv.CsvAuditEventHandler",
            "config": {
              "name": "csv",
              "logDirectory": "/tmp/logs",
              "security": {
                "enabled": "true",
                "filename": "/path/to/secrets/audit-
keystore",
                "password": "password",
                "signatureInterval": "1 day"
              },
              "topics": [
                "access"
              ]
            }
          }
        ],
        "config": { }
      }
    }
  ],
  "auditService": "AuditService",
  "handler": "ForgeRockClientHandler"
}
```

The route calls an audit service configuration for publishing log messages to the CSV file, `/tmp/logs/access.csv`.

When a request matches `audit`, audit events are logged to the CSV file.

The route uses the `ForgeRockClientHandler` as its handler, to send the `X-ForgeRock-TransactionId` header with its requests to external services.

3. Test the setup:

   a. In your browser's privacy or incognito mode, go to https://ig.example.com:8443/home/csv-audit.

      The home page of the sample application is displayed, and the file `/tmp/logs/tamper-evident-access.csv` is updated.

Record access audit events with a JMS audit event handler

IMPORTANT

This procedure is an example of how to record access audit events with a JMS audit event handler configured to use the ActiveMQ message broker. This example is not tested on all configurations, and can be more or less relevant to your configuration.

For information about configuring the JMS event handler, refer to JmsAuditEventHandler.

Before you start, prepare PingGateway as described in the Quick install.

1. Download the following files:
   - ActiveMQ binary. PingGateway is tested with ActiveMQ Classic 5.15.11.
   - ActiveMQ Client. Use a version that corresponds to your ActiveMQ version.
   - Apache Geronimo J2EE management bundle.
   - hawtbuf-1.11 JAR.
2. Add the files to the configuration:

- Create the directory `$HOME/.openig/extra`, where `$HOME/.openig` is the instance directory, and add .jar files to the directory.

3. Create a consumer that subscribes to the `audit` topic.

   From the ActiveMQ installation directory, run the following command:

   ```
   $ ./bin/activemq consumer --destination topic://audit
   ```

4. Set up PingGateway

   a. Set up PingGateway for HTTPS, as described in Configure PingGateway for TLS (server-side).

   b. Add the following route to PingGateway:

   Linux Windows

   ```
   $HOME/.openig/config/routes/30-jms.json
   ```

   ```
   {
     "name": "30-jms",
     "MyCapture" : "all",
     "baseURI": "http://app.example.com:8081",
     "condition" : "${request.uri.path ==
   '/activemq_event_handler'}",
     "heap": [
       {
         "name": "AuditService",
         "type": "AuditService",
         "config": {
           "eventHandlers" : [
             {
               "class" :
   "org.forgerock.audit.handlers.jms.JmsAuditEventHandler",
               "config" : {
                 "name" : "jms",
                 "topics": [ "access" ],
                 "deliveryMode" : "NON_PERSISTENT",
                 "sessionMode" : "AUTO",
                 "jndi" : {
                   "contextProperties" : {
                     "java.naming.factory.initial" :
   "org.apache.activemq.jndi.ActiveMQInitialContextFactory",
                     "java.naming.provider.url" :
   ```

```
"tcp://am.example.com:61616",
                  "topic.audit" : "audit"
                },
                "topicName" : "audit",
                "connectionFactoryName" :
"ConnectionFactory"
              }
            }
          }
        ],
        "config" : { }
      }
    }
  ],
  "auditService": "AuditService",
  "handler" : {
    "type" : "StaticResponseHandler",
    "config" : {
      "status" : 200,
      "headers" : {
        "Content-Type" : [ "text/plain; charset=UTF-8" ]
      },
      "entity" : "Message from audited route"
    }
  }
}
```

When a request matches the `/activemq_event_handler` route, this configuration publishes JMS messages containing audit event data to an ActiveMQ managed JMS topic, and the StaticResponseHandler displays a message.

5. Test the setup:

   a. In your browser's privacy or incognito mode, go to https://ig.example.com:8443/activemq_event_handler.

      Depending on how ActiveMQ is configured, audit events are displayed on the ActiveMQ console or written to file.

Record access audit events with a JSON audit event handler

This section describes how to record access audit events with a JSON audit event handler. For information about configuring the JSON event handler, refer to JsonAuditEventHandler.

1. Set up PingGateway

a. Set up PingGateway for HTTPS, as described in Configure PingGateway for TLS (server-side).

b. Add the following route to PingGateway:

Linux | Windows

```
$HOME/.openig/config/routes/30-json.json
```

```
{
  "name": "30-json",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/json-
audit')}",
  "heap": [
    {
      "name": "AuditService",
      "type": "AuditService",
      "config": {
        "eventHandlers": [
          {
            "class":
"org.forgerock.audit.handlers.json.JsonAuditEventHandler",
            "config": {
              "name": "json",
              "logDirectory": "/tmp/logs",
              "topics": [
                "access"
              ],
              "rotationRetentionCheckInterval": "1
minute",
              "buffering": {
                "maxSize": 100000,
                "writeInterval": "100 ms"
              }
            }
          }
        ]
      }
    }
  ],
  "auditService": "AuditService",
  "handler": "ReverseProxyHandler"
}
```

Notice the following features of the route:

- The route calls an audit service configuration for publishing log messages to the JSON file, `/tmp/audit/access.audit.json`. When a request matches `/home/json-audit`, a single line per audit event is logged to the JSON file.
- The route uses the `ForgeRockClientHandler` as its handler, to send the `X-ForgeRock-TransactionId` header with its requests to external services.

2. Test the setup:

   a. In your browser's privacy or incognito mode, go to https://ig.example.com:8443/home/json-audit.

   The home page of the sample application is displayed and the file `/tmp/logs/access.audit.json` is created or updated with a message. The following example message is formatted for easy reading, but it is produced as a single line for each event:

```
{
  "_id": "830...-41",
  "timestamp": "2019-...540Z",
  "eventName": "OPENIG-HTTP-ACCESS",
  "transactionId": "830...-40",
  "client": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 51666
  },
  "server": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 8080
  },
  "http": {
    "request": {
      "secure": false,
      "method": "GET",
      "path": "http://ig.example.com:8080/home/json-
audit",
      "headers": {
        "accept":
["text/html,application/xhtml+xml,application/xml;q=0.9,/;
q=0.8"],
        "host": ["ig.example.com:8080"],
        "user-agent": ["Mozilla/5.0 ... Firefox/66.0"]
      }
    }
  },
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "200",
    "elapsedTime": 212,
    "elapsedTimeUnits": "MILLISECONDS"
  },
  "ig": {
    "exchangeId": "b3f...-29",
    "routeId": "30-json",
    "routeName": "30-json"
  }
}
```

Record access audit events to standard output

This section describes how to record access audit events to standard output. For more information about the event handler, refer to JsonStdoutAuditEventHandler.

Before you start, prepare PingGateway and the sample application as described in the Quick install.

1. Set up PingGateway

   a. Set up PingGateway for HTTPS, as described in Configure PingGateway for TLS (server-side).

   b. Add the following route to PingGateway:

   Linux | Windows

   ```
   $HOME/.openig/config/routes/30-jsonstdout.json
   ```

   ```
   {
     "name": "30-jsonstdout",
     "baseURI": "http://app.example.com:8081",
     "condition": "${find(request.uri.path,
   '^/home/jsonstdout-audit')}",
     "heap": [
       {
         "name": "AuditService",
   ```

```
        "type": "AuditService",
        "config": {
          "eventHandlers": [
            {
              "class":
"org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditE
ventHandler",
              "config": {
                "name": "jsonstdout",
                "elasticsearchCompatible": false,
                "topics": [
                  "access"
                ]
              }
            }
          ],
          "config": {}
        }
      }
    ],
    "auditService": "AuditService",
    "handler": "ReverseProxyHandler"
}
```

Notice the following features of the route:

- The route matches requests to `/home/jsonstdout-audit`.
- The route calls the audit service configuration for publishing access log messages to standard output. When a request matches `/home/jsonstdout-audit`, a single line per audit event is logged.

2. In your browser's privacy or incognito mode, go to https://ig.example.com:8443/home/jsonstdout-audit.

   The home page of the sample application is displayed, and a message like this is published to standard output:

```
{
  "_id": "830...-61",
  "timestamp": "2019-...89Z",
  "eventName": "OPENIG-HTTP-ACCESS",
  "transactionId": "830...-60",
  "client": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 51876
  },
  "server": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 8080
  },
  "http": {
    "request": {
      "secure": false,
      "method": "GET",
      "path": "http://ig.example.com:8080/home/jsonstdout-
audit",
      "headers": {
        "accept":
["text/html,application/xhtml+xml,application/xml;q=0.9,/;q=0.
8"],
        "host": ["ig.example.com:8080"],
        "user-agent": ["Mozilla/5.0 ... Firefox/66.0"]
      }
    }
  },
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "200",
    "elapsedTime": 10,
    "elapsedTimeUnits": "MILLISECONDS"
  },
  "ig": {
    "exchangeId": "b3f...-41",
    "routeId": "30-jsonstdout",
    "routeName": "30-jsonstdout"
  },
  "source": "audit",
  "topic": "access",
  "level": "INFO"
}
```

Trust transaction IDs from other products

Each audit event is identified by a unique transaction ID that can be communicated across products and recorded for each local event. By using the transaction ID, requests can be tracked as they traverse the platform, making it easier to monitor activity and to enrich reports.

The `X-ForgeRock-TransactionId` header is automatically set in all outgoing HTTP calls from one ForgeRock product to another. Customers can also set this header themselves from their own applications or scripts that call into the Ping Identity Platform.

To reduce the risk of malicious attacks, by default PingGateway does not trust transaction ID headers from client applications.

If you trust the transaction IDs sent by your client applications, consider setting Java system property `org.forgerock.http.TrustTransactionHeader` to `true`.

Add the following system property in `env.sh`:

```
# Specify a JVM option
TX_HEADER_OPT="-Dorg.forgerock.http.TrustTransactionHeader=true"

# Include it into the JAVA_OPTS environment variable
export JAVA_OPTS="${TX_HEADER_OPT}"
```

All incoming `X-ForgeRock-TransactionId` headers are trusted, and monitoring or reporting systems that consume the logs can allow requests to be correlated as they traverse multiple servers.

Safelist audit event fields for the logs

To prevent logging of sensitive data for an audit event, the Common Audit Framework uses a safelist to specify which audit event fields appear in the logs.

By default, only safelisted audit event fields are included in the logs. For information about how to include non-safelisted audit event fields, or exclude safelisted audit event fields, refer to Including or excluding audit event fields in logs.

Audit event fields use JSON pointer notation, and are taken from the JSON schema for the audit event content. The following event fields are safelisted:

- `/_id`
- `/timestamp`
- `/eventName`
- `/transactionId`
- `/trackingIds`
- `/userId`
- `/client`
- `/server`
- `/ig/exchangeId`
- `/ig/routeId`
- `/ig/routeName`
- `/http/request/secure`

- `/http/request/method`
- `/http/request/path`
- `/http/request/headers/accept`
- `/http/request/headers/accept-api-version`
- `/http/request/headers/content-type`
- `/http/request/headers/host`
- `/http/request/headers/user-agent`
- `/http/request/headers/x-forwarded-for`
- `/http/request/headers/x-forwarded-host`
- `/http/request/headers/x-forwarded-port`
- `/http/request/headers/x-forwarded-proto`
- `/http/request/headers/x-original-uri`
- `/http/request/headers/x-real-ip`
- `/http/request/headers/x-request-id`
- `/http/request/headers/x-requested-with`
- `/http/request/headers/x-scheme`
- `/request`
- `/response`

Include or exclude audit event fields in logs

The safelist is designed to prevent logging of sensitive data for audit events by specifying which audit event fields appear in the logs. You can add or remove messages from the logs as follows:

- To include audit event fields in logs that are not safelisted, configure the `includeIf` property of AuditService.

  IMPORTANT

  Before you include non-safelisted audit event fields in the logs, consider the impact on security. Including some headers, query parameters, or cookies in the logs could cause credentials or tokens to be logged, and allow anyone with access to the logs to impersonate the holder of these credentials or tokens.

- To exclude safelisted audit event fields from the logs, configure the `excludeIf` property of AuditService. For an example, refer to Exclude safelisted audit event fields from logs.

Exclude safelisted audit event fields from logs

Before you start, set up and test the example in Recording access audit events in JSON. Note the audit event fields in the log file `access.audit.json`.

1. Replace `30-json.json` with the following route:

   Linux | Windows

   ```
   $HOME/.openig/config/routes/30-json-excludeif.json
   ```

   ```
   {
     "name": "30-json-excludeif",
     "baseURI": "http://app.example.com:8081",
     "condition": "${find(request.uri.path, '^/home/json-audit-
   excludeif$')}",
     "heap": [
       {
         "name": "AuditService",
         "type": "AuditService",
         "config": {
           "config": {
             "filterPolicies": {
               "field": {
                 "excludeIf": [
                   "/access/http/request/headers/host",
                   "/access/http/request/path",
                   "/access/server",
                   "/access/response"
                 ]
               }
             }
           },
           "eventHandlers": [
             {
               "class":
   "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
               "config": {
                 "name": "json",
                 "logDirectory": "/tmp/logs",
                 "topics": [
                   "access"
                 ],
                 "rotationRetentionCheckInterval": "1 minute",
                 "buffering": {
                   "maxSize": 100000,
   ```

```
                    "writeInterval": "100 ms"
                  }
                }
              }
            ]
          }
        }
      ],
      "auditService": "AuditService",
      "handler": "ReverseProxyHandler"
    }
```

Notice that the AuditService is configured with an `excludeIf` property to exclude audit event fields from the logs.

2. In your browser's privacy or incognito mode, go to https://ig.example.com:8443/home/json-audit-excludeif.

   The home page of the sample application is displayed and the file `/tmp/logs/access.audit.json` is updated:

```
{
  "_id": "830...-41",
  "timestamp": "2019-...540Z",
  "eventName": "OPENIG-HTTP-ACCESS",
  "transactionId": "830...-40",
  "client": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 51666
  },
  "http": {
    "request": {
      "secure": false,
      "method": "GET",
      "headers": {
        "accept":
["text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
0.8"],
        "user-agent": ["Mozilla/5.0 ... Firefox/66.0"]
      }
    }
  },
  "ig": {
    "exchangeId": "b3f...-56",
    "routeId": "30-json-excludeif",
    "routeName": "30-json-excludeif"
    }
  }
```

3. Compare the audit event fields in `access.audit.json` with those produced in Recording access audit events in JSON, and note that the audit event fields specified by the `excludeIf` property no longer appear in the logs.

Record user ID in audit events

The following sections provide examples of how to capture the AM user ID in audit logs.

Sample scripts are available in the `openig-samples.jar` file, to capture the user ID after SSO, CDSSO, OpenID, or SAML authentication. The scripts inject the user ID into the RequestAuditContext so that it is available when the audit event is written.

Using the notes in the sample scripts, adapt the script for your deployment. For example, configure which `user_info` field to capture in the audit event.

The audit service in these examples use a JsonStdoutAuditEventHandler, which writes audit events to standard output, but can be any other audit service.

Record user ID in audit logs after SSO authentication

Before you start, set up and test the example in Cross-domain single sign-on (CDSSO).

1. Add the following script to PingGateway:

Linux | Windows

```
$HOME/.openig/scripts/groovy/InjectUserIdSso.groovy
```

```
package scripts.groovy

import org.forgerock.openig.openam.SsoTokenContext
import org.forgerock.services.context.RequestAuditContext

/**
 * Sample ScriptableFilter implementation to capture the user
id from the session
 * and inject it into the RequestAuditContext for later use
when the audit event
 * is written.
 *
 * This ScriptableFilter should be added in the filter chain
```

```
at whatever point the
 * desired user id is available - e.g. on the session after
SSO.
 *
 * "handler": {
 *   "type": "Chain",
 *   "config": {
 *     "filters": [ {
 *        "name": "SingleSignOnFilter-1",
 *         "type": "SingleSignOnFilter",
 *         "config": {
 *           "amService": "AmService-1"
 *         }
 *       }, {
 *         "type" : "ScriptableFilter",
 *         "config" : {
 *           "file" : "InjectUserIdSso.groovy",
 *           "type": "application/x-groovy"
 *         }
 *       }
 *     ],
 *     "handler" : "ReverseProxyHandler",
 * }
 *
 * When using the SSO/ CDSSO flow then the SsoTokenContext is
guaranteed to exist and
 * be populated if there was no error. The RequestAuditContext
is also guaranteed to
 * be available. Note also that if the SessionInfoFilter is
present in the route then
 * a SessionInfoContext would be available in the context
chain and could be queried
 * for user info.
 *
 * Implementors may decide which user id field to capture in
the audit event:
 * - The sessionInfo universalId - 'universalId' - is always
available as
 *   provided by AM and resembles -
 *   e.g.
"id=bonnie,ou=user,o=myrealm,ou=services,dc=openam,dc=forgeroc
k,dc=org".
 * - The sessionInfo username - mapped to 'username')
resembles - e.g. "bonnie".
 *   Field 'username' should be preferred to 'uid', which also
```

```
points to 'username'.
 *
 * Additional error handling may be required.
 *
 * @see RequestAuditContext
 * @see SsoTokenContext
 * @see org.forgerock.openig.openam.SessionInfoContext
 */

def requestAuditContext =
context.asContext(RequestAuditContext.class)
def ssoTokenContext = context.asContext(SsoTokenContext.class)

// The sessionInfo 'universalId' is always available, though
'username' may be unknown
requestAuditContext.setUserId(ssoTokenContext.universalId)

// Propagate the request to the next filter/ handler in the
chain
next.handle(context, request)
```

The script captures the user ID after SSO or CDSSO authentication, and injects it into the RequestAuditContext so that it is available when the audit event is written.

2. Replace `sso.json` with the following route:

Linux Windows

```
$HOME/.openig/config/routes/audit-sso.json
```

```
{
  "name": "audit-sso",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/audit-
sso$')}",
  "heap": [
    {
      "name": "AuditService",
      "type": "AuditService",
      "config": {
        "eventHandlers": [
          {
            "class":
"org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEvent
```

```
Handler",
            "config": {
              "name": "jsonstdout",
              "elasticsearchCompatible": false,
              "topics": [
                "access"
              ]
            }
          }
        ],
        "config": {}
      }
    },
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "agent": {
          "username": "ig_agent",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "url": "http://am.example.com:8088/openam/"
      }
    }
  ],
  "auditService": "AuditService",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "SingleSignOnFilter-1",
          "type": "SingleSignOnFilter",
          "config": {
            "amService": "AmService-1"
          }
        },
        {
          "type" : "ScriptableFilter",
          "config" : {
            "file" : "InjectUserIdSso.groovy",
            "type": "application/x-groovy"
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

Notice the following features of the route compared to `sso.json`:

- The route matches requests to `/home/audit-sso`.
- An audit service is included to publish access log messages to standard output.
- The chain includes a scriptable filter that refers to `InjectUserIdSso.groovy`.

3. Test the setup:

   a. In your browser's privacy or incognito mode, go to https://ig.example.com:8443/home/audit-sso. The SingleSignOnFilter redirects the request to AM for authentication.

   b. Log in to AM as user `demo`, password `Ch4ng31t`, and then allow the application to access user information.

      The profile page of the sample application is displayed. The script captures the user ID from the session, and the audit service includes it with the audit event.

   c. Search the standard output for a message like this, containing the user ID:

```
{
  "_id": "23a...-23",
  "timestamp": "...",
  "eventName": "OPENIG-HTTP-ACCESS",
  "transactionId": "23a...-22",
  "userId":
"id=demo,ou=user,dc=openam,dc=forgerock,dc=org",
  "client": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 57843
  },
  "server": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 8080
  },
  "http": {
    "request": {
      "secure": false,
      "method": "GET",
      "path": "http://ig.example.com/home/audit-sso",
      "headers": {
        "accept":
["text/html,application/xhtml+xml,application/xml;q=0.9,im
age/webp,/;q=0.8"],
        "host": ["ig.example.com:8080"],
        "user-agent": [...]
      }
    }
  },
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "200",
    "elapsedTime": 276,
    "elapsedTimeUnits": "MILLISECONDS"
  },
  "ig": {
    "exchangeId": "1dc...-26",
    "routeId": "audit-sso",
    "routeName": "audit-sso"
  },
  "source": "audit",
  "topic": "access",
  "level": "INFO"
}
```

Record user ID in audit logs after OpenID connect authentication

Before you start, set up and test the example in AM as OIDC provider.

1. Set up the script:
    a. Add the following example script to PingGateway:

Linux | Windows

```
$HOME/.openig/scripts/groovy/InjectUserIdOpenId.groovy
```

```
package scripts.groovy

import org.forgerock.services.context.AttributesContext
import org.forgerock.services.context.RequestAuditContext
```

```
/**
 * Sample script implementation supporting user id
injection in an OpenId scenario.
 * This sample captures the user id and injects it into
the RequestAuditContext for
 * later use when the audit event is written.
 *
 * This ScriptableFilter should be added in the filter
chain at whatever point the
 * desired user id is available - e.g. after OpenId client
authentication (in the
 * OAuth2 authentication filter chain) - as follows:
 *
 * "handler" : {
 *   "type" : "Chain",
 *   "config" : {
 *     "filters" : [ {
 *       "type" : "OAuth2ClientFilter",
 *       "config" : {
 *         ...
 *         "target" : "${attributes.target}",
 *         "registrations" : [
"ClientRegistrationWithOpenIdScope" ],
 *       }
 *     }, {
 *       "type" : "ScriptableFilter",
 *       "config" : {
 *         "file" : "InjectUserIdOpenId.groovy",
 *         "type": "application/x-groovy"
 *       }
 *     } ],
 *     "handler" : "display-user-info-groovy-handler"
 *   }
 * }
 *
 * The ClientRegistration associated with the above
OAuth2ClientFilter config will
 * require the 'openid' scope. The OAuth2SessionContext is
guaranteed to exist and
 * be populated on successful authentication. The userinfo
will then be populated
 * according to the OAuth2ClientFilter OpenId 'target'
configuration (e.g. in this
 * sample, on the AttributesContext). The 'target'
```

```
referenced will be populated
 * with a 'user_info' JSON value containing the userinfo.
It should be noted that
 * the OAuth2ClientFilter 'target' config is a config-time
expression, and cannot
 * be used in a ScriptableFilter to read runtime data. The
RequestAuditContext is
 * also guaranteed to be available.
 *
 * Implementors may decide which 'user_info' field to
capture in the audit event:
 * - The userinfo 'sub' field is the user's "complex" ID
marked with a type - e.g.
 *   "(usr!bonnie)".
 * - The userinfo 'subName' field is the user's username
(or resource name) - e.g.
 *   "bonnie".
 * - To capture the universalId (consistent with the
session info universalId),
 *   it is necessary to configure AM to provide it as a
claim in the id-token. To
 *   do this, edit the OIDC Claims Script to include the
following line just prior
 *   to the UserInfoClaims creation:
 *       computedClaims["universalId"] =
identity.universalId
 * - This will include 'universalId' in the userinfo which
we can use with audit
 *   e.g.
"id=bonnie,ou=user,o=myrealm,ou=services,dc=openam,dc=forg
erock,dc=org"
 *
 * Additional error handling may be required.
 *
 * @see RequestAuditContext
 * @see AttributesContext
 */

def requestAuditContext =
context.asContext(RequestAuditContext.class)
def attributesContext =
context.asContext(AttributesContext.class)

// The OAuth2ClientFilter captures userinfo based on its
'target' configuration.
```

```
// In this sample 'target' is configured as the
AttributesContext with key "target".
// We can query this for 'user_info' values: 'sub',
'subName' or anything else
// made available via the OIDC Claims Script (see above).
def oauth2UserInfo =
attributesContext.getAttributes().get("target")
requestAuditContext.setUserId(oauth2UserInfo.get("user_inf
o").get("sub"))

// Propagate the request to the next filter/ handler in
the chain
next.handle(context, request)
```

The script captures the user ID from the AuthorizationCodeOAuth2ClientFilter `target` object, by default at `${attributes.openid}`, and injects it into the RequestAuditContext so that it is available when the audit event is written.

b. Edit the script to get the attributes from the `openid` target:

Replace `attributesContext.getAttributes().get("target")`

with `attributesContext.getAttributes().get("openid")`.

2. Replace `07-openid.json` with the following route:

Linux | Windows

```
$HOME/.openig/config/routes/audit-oidc.json
```

```
{
  "name": "audit-oidc",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/id_token')}",
  "heap": [
    {
      "name": "AuditService",
      "type": "AuditService",
      "config": {
        "eventHandlers": [
          {
            "class":
"org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEvent
Handler",
            "config": {
              "name": "jsonstdout",
              "elasticsearchCompatible": false,
              "topics": [
                "access"
              ]
            }
          }
        ],
        "config": {}
      }
    },
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    }
  ],
  "auditService": "AuditService",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "AuthorizationCodeOAuth2ClientFilter-1",
          "type": "AuthorizationCodeOAuth2ClientFilter",
          "config": {
            "clientEndpoint": "/home/id_token",
            "failureHandler": {
              "type": "StaticResponseHandler",
              "config": {
                "status": 500,
                "headers": {
                  "Content-Type": [
                    "text/plain"
                  ]
                },
                "entity": "Error in OAuth 2.0 setup."
              }
            },
            "registrations": [
              {
                "name": "oidc-user-info-client",
                "type": "ClientRegistration",
                "config": {
                  "clientId": "oidc_client",
                  "clientSecretId": "oidc.secret.id",
                  "issuer": {
                    "name": "Issuer",
                    "type": "Issuer",
                    "config": {
                      "wellKnownEndpoint":
"http://am.example.com:8088/openam/oauth2/.well-known/openid-
configuration"
                    }
                  },
                  "scopes": [
                    "openid",
                    "profile",
                    "email"
                  ],
                  "secretsProvider": "SystemAndEnvSecretStore-
1",
                  "tokenEndpointAuthMethod":
"client_secret_basic"
                }
              }
            ],
            "requireHttps": false,
            "cacheExpiration": "disabled"
          }
        },
        {
          "type" : "ScriptableFilter",
          "config" : {
            "file" : "InjectUserIdOpenId.groovy",
            "type": "application/x-groovy"
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

Notice the following features of the route compared to `07-openid.json`:

- An audit service is included to publish access log messages to standard output.
- The chain includes a scriptable filter that refers to `InjectUserIdOpenId.groovy`.

3. Test the setup:

a. In your browser’s privacy or incognito mode, go to https://ig.example.com:8443/home/id_token. The AM login page is displayed.

b. Log in to AM as user `demo`, password `Ch4ng31t`, and then allow the application to access user information.

   The home page of the sample application is displayed. The script captures the user ID from the `openid` target, and the audit service includes it with the audit event.

c. Search the standard output for a message like this, containing the user ID:

```
{
  "_id": "b64...-25",
  "timestamp": "2021...",
  "eventName": "OPENIG-HTTP-ACCESS",
  "transactionId": "b64...-24",
  "userId": "(usr!demo)",
  "client": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 64443
  },
  "server": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 8080
  },
  "http": {
    "request": {
      "secure": false,
      "method": "GET",
      "path": "http://ig.example.com:8080/home/id_token",
      "headers": {
        "accept":
["text/html,application/xhtml+xml,application/xml;q=0.9,im
age/webp,/;q=0.8"],
        "host": ["ig.example.com:8080"],
        "user-agent": [...]
      }
    }
  },
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "200",
    "elapsedTime": 199,
    "elapsedTimeUnits": "MILLISECONDS"
  },
  "ig": {
    "exchangeId": "1dc...-26",
    "routeId": "audit-oidc",
    "routeName": "audit-oidc"
  },
  "source": "audit",
  "topic": "access",
  "level": "INFO"
}
```

Record user ID in audit logs after SAML authentication

IMPORTANT

This example uses the deprecated SamlFederationHandler. The SamlFederationHandler is replaced by the SamlFederationFilter and will be removed in a future release.

Before you start, set up and test the SamlFederationHandler example under SAML.

1. Set up the script:
   a. Add the following example script to PingGateway:

   Linux | Windows

   ```
   $HOME/.openig/scripts/groovy/InjectUserIdSaml.groovy
   ```

   ```
   package scripts.groovy

   import org.forgerock.http.session.SessionContext
   import org.forgerock.services.context.RequestAuditContext

   /**
    * Sample ScriptableFilter implementation to capture the
   user id obtained from a
    * SAML assertion. The IG SamlFederationHandler captures
   this and locates it on
    * the SessionContext with the key as the configured SAML
   2 user id key. We then
    * take this and inject it into the RequestAuditContext
   for later use when the
    * audit event is written.
    *
    * This ScriptableFilter should be added in the filter
   ```

```
chain together with the
 * SamlFederationHandler, as follows. Note that the
InjectUserIdSaml.groovy script
 * operates on the response, injecting the userId as
captured by the handler.
 *
 * {
 *     "condition" :
"${matches(request.uri.path,'^/api/saml')}",
 *     "handler" : {
 *         "type" : "Chain",
 *         "config" : {
 *             "filters" : [ {
 *                 "type" : "ScriptableFilter",
 *                 "config" : {
 *                     "file" : "InjectUserIdSaml.groovy",
 *                     "type": "application/x-groovy"
 *                 }
 *             } ],
 *             "handler" : {
 *                 "name" : "saml_handler_SPOne",
 *                 "type" : "SamlFederationHandler",
 *                 "config" : {
 *                      "assertionMapping" : {
 *                          "SPOne_userName" : "uid",
 *                          "SPOne_password" : "mail"
 *                      },
 *                      "redirectURI" : "/api/home",
 *                      "logoutURI" :
"http://openig.example.com:8082/api/after_logout",
 *                      "subjectMapping" :
"SubjectName_SPOne",
 *                      "authnContext" :
"AuthnContext_SPOne",
 *                      "sessionIndexMapping" :
"SessionIndex_SPOne"
 *                 }
 *             }
 *         }
 *     }
 * }
 *
 * The SessionContext and RequestAuditContext are
guaranteed to be available and the
 * SessionContext will have been populated with userinfo
```

```
on successful authentication.
 *
 * Implementors may decide which user id field to capture
in the audit event:
 * - This should be based on SAML attribute mappings and/
or the subject mapping (if
 *   transient names are not used).
 * - Other attributes are available, such as 'uid' and
'userName', though  it must be
 *   noted that there is an expectation that the IDP makes
available the user id.
 * - In this sample, 'SPOne_userName' maps to the 'uid'.
 *
 * Additional error handling may be required.
 *
 * @see RequestAuditContext
 * @see SessionContext
 */

// Propagate the request to the next filter/ handler in
the chain
next.handle(context, request)
    .then({ response ->
        def requestAuditContext =
context.asContext(RequestAuditContext.class)
        def sessionContext =
context.asContext(SessionContext.class)

        // Inject the user id as captured by the
SamlFederationHandler

requestAuditContext.setUserId(sessionContext.getSession().
get("SPOne_userName"))
        return response
    })
```

The script captures the user ID from the SessionContext subject or attribute mappings, provided by the SamlFederationHandler from the inbound assertions. It injects the user ID into the RequestAuditContext so that it is available when the audit event is written.

b. Replace `get("SPOne_userName"))` with `get("username"))`.

   The script captures the user ID from the assertionMapping `username`, which is mapped in the route to `cn`.

2. Replace `saml.json` with the following route:

Linux Windows

```
$HOME/.openig/config/routes/audit-saml.json
```

```
{
  "name": "audit-saml",
  "condition": "${find(request.uri.path, '^/saml')}",
  "session": "JwtSession",
  "heap": [
    {
      "name": "AuditService",
      "type": "AuditService",
      "config": {
        "eventHandlers": [
          {
            "class":
"org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEvent
Handler",
            "config": {
              "name": "jsonstdout",
              "elasticsearchCompatible": false,
              "topics": [
                "access"
              ]
            }
          }
        ],
        "config": {}
      }
    }
  ],
  "auditService": "AuditService",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type" : "ScriptableFilter",
          "config" : {
            "file" : "InjectUserIdSaml.groovy",
            "type": "application/x-groovy"
          }
        }
      ],
      "handler": {
        "type": "SamlFederationHandler",
        "config": {
          "useOriginalUri": true,
          "assertionMapping": {
            "username": "cn",
            "password": "sn"
          },
          "subjectMapping": "sp-subject-name",
          "redirectURI": "/home/federate"
        }
      }
    }
  }
}
```

Notice the following features of the route compared to `saml.json`:

- An audit service is included to publish access log messages to standard output.
- The main Handler is a Chain, that includes a scriptable filter to refer to `InjectUserIdSaml.groovy`.
- The script uses the assertionMapping `username` to capture the user ID.

3. Test the setup:

   a. In your browser's privacy or incognito mode, go to IDP-initiated SSO.

   b. Log in to AM with username `demo` and password `Ch4ng31t`.

      PingGateway returns the response page showing that the the demo user has logged in. The script captures the user ID from the session, and the audit service includes it with the audit event.

   c. Search the standard output for a message like this, containing the user ID:

```
{
  "_id": "82f...-14",
  "timestamp": "2021-...",
  "eventName": "OPENIG-HTTP-ACCESS",
  "transactionId": "82f...-13",
  "userId": "demo",
  "client": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 60655
  },
  "server": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 8080
  },
  "http": {
    "request": {
      "secure": false,
      "method": "POST",
      "path":
"http://sp.example.com:8080/saml/fedletapplication/metaAli
as/sp",
      "headers": {
        "accept":
["text/html,application/xhtml+xml,application/xml;q=0.9,im
age/webp,/;q=0.8"],
        "content-type": ["application/x-www-form-
urlencoded"],
        "host": ["sp.example.com:8080"],
        "user-agent": [...]
      }
    }
  },
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "302",
    "elapsedTime": 2112,
    "elapsedTimeUnits": "MILLISECONDS"
  },
  "ig": {
    "exchangeId": "1dc...-26",
    "routeId": "audit-saml",
    "routeName": "audit-saml"
  },
  "source": "audit",
  "topic": "access",
  "level": "INFO"
}
```

Monitor services

This page describes how to set up and maintain monitoring in your deployment.

Push to an OpenTelemetry service

PingGateway can push traces to an OpenTelemetry Protocol (OTLP) endpoint over HTTP. When you use it with other applications with OpenTelemetry support, the service helps you analyze the flows through PingGateway and the other applications to understand performance and system behavior.

> IMPORTANT
>
> This capability is available in Technology preview. It isn't yet supported, may be functionally incomplete, and is subject to change without notice.

OpenTelemetry support is disabled by default. Enable it with the `"openTelemetry"` setting in AdminHttpApplication (`admin.json`). Make sure the OTLP exporter targets the endpoint for the OpenTelemetry service in your deployment. The following example targets the default endpoint:

```
{
    "openTelemetry": {
        "tracing": {
            "enabled": true,
            "exporter": {
                "type": "otlp",
                "config": {
                    "endpoint": "http://localhost:4318/v1/traces"
                }
            }
        }
    }
}
```

When tracing is enabled, traces always span requests entering PingGateway and any requests started by PingGateway; for example, proxied requests to the sample application. The span ends when PingGateway receives or returns a response or when an error arises during processing. If a ClientHandler or ReverseProxyHandler has the retry feature enabled, PingGateway records the number of retries as a `http.request.resend_count` span attribute in the trace.

Trace sampling is `alwaysOn` by default, meaning PingGateway samples every operation. Adjust this in the configuration settings as required for your deployment and traffic.

Use the `"tracing": true` decoration on other configuration objects or in `"globalDecorators"` to activate additional tracing points.

Access the monitoring endpoints

All ForgeRock products automatically expose a monitoring endpoint to expose metrics in a standard Prometheus format, and as a JSON format monitoring resource.

In PingGateway, metrics are available for each router, subrouter, and route in the configuration. When a TimerDecorator is configured, timer metrics are also available.

Learn more about PingGateway monitoring endpoints and available metrics in Monitoring.

Monitor at the Prometheus Scrape Endpoint

NOTE

Prometheus metric names are deprecated and expected to be replaced with names ending in _total. The information provided by the metric is not deprecated. Other Prometheus metrics are not affected.

All ForgeRock products automatically expose a monitoring endpoint where Prometheus can scrape metrics, in a standard Prometheus format.

When PingGateway is set up as described in the Quick install, the Prometheus Scrape Endpoint is available at the following endpoints:

- http://ig.example.com:8080/openig/metrics/prometheus/0.0.4
- http://ig.example.com:8080/openig/metrics/prometheus (deprecated)

By default, no special setup or configuration is required to access metrics at these endpoints. The following example queries the Prometheus Scrape Endpoint for a route.

Tools such as Grafana are available to create customized charts and graphs based on the information collected by Prometheus. For more information on installing and running Grafana, refer to the Grafana website.

1. Set up PingGateway:
   a. Set up PingGateway for HTTPS, as described in Configure PingGateway for TLS (server-side).
   b. Add the following route to PingGateway:

      Linux Windows

      ```
      $HOME/.openig/config/routes/myroute1.json
      ```

      ```
      {
        "name": "myroute1",
        "handler": {
          "type": "StaticResponseHandler",
          "config": {
      ```

```
        "status": 200,
        "headers": {
          "Content-Type": [ "text/plain; charset=UTF-8" ]
        },
        "entity": "Hello world, from myroute1!"
      }
    },
    "condition": "${find(request.uri.path, '^/myroute1')}"
  }
```

The route contains a StaticResponseHandler to display a simple message.

2. Test the setup:

   a. Access the route a few times, on https://ig.example.com:8443/myroute1.

   b. Query the Prometheus Scrape Endpoint:

```
$ curl -v \
--cacert /path/to/secrets/ig.example.com-certificate.pem \
https://ig.example.com:8443/openig/metrics/prometheus/0.0.
4
```

   Metrics for `myroute1` and `_router` are displayed.

   Vert.x monitoring is enabled by default to provide additional metrics for HTTP, TCP, and the internal component pool. The metrics provide low-level information about requests and responses, such as the number of bytes, duration, and the number of concurrent requests.

Monitor the Common REST Monitoring Endpoint (deprecated)

All ForgeRock products expose a monitoring endpoint where metrics are exposed as a JSON format monitoring resource.

When PingGateway is set up as described in Quick install, the Common REST Monitoring Endpoint is available at https://ig.example.com:8443/openig/metrics/api?_prettyPrint=true&_sortKeys=_id&_queryFilter=true

By default, no special setup or configuration is required to access metrics at this endpoint. The following example queries the Common REST Monitoring Endpoint for a route, and restricts the query to specific metrics only.

Before you start, prepare PingGateway as described in the Quick install.

1. Set up PingGateway and some example routes, as described in the first few steps of Monitor the Prometheus Scrape Endpoint.

2. Query the Common REST Monitoring Endpoint:

```
$ curl "https://ig.example.com:8443/openig/metrics/api?
_prettyPrint=true&_sortKeys=_id&_queryFilter=true"
```

Metrics for `myroute1` and `_router` are displayed:

```
{
  "result" : [ {
  "_id" : "gateway._router.deployed-routes",
  "value" : 1.0,
  "_type" : "gauge"
}, {
  "_id" : "gateway._router.route.default.request",
  "count" : 204,
  "_type" : "counter"
}, {
  "_id" : "gateway._router.route.default.request.active",
  "value" : 0.0,
  "_type" : "gauge"
}, {
  "...":  "...",
  "_id" :
"gateway._router.route.myroute1.response.status.unknown",
  "count" : 0,
  "_type" : "counter"
}, {
  "_id" : "gateway._router.route.myroute1.response.time",
  "count" : 204,
  "max" : 0.420135,
  "mean" : 0.08624678327176545,
  "min" : 0.045079999999999995,
  "p50" : 0.070241,
  "p75" : 0.096049,
  "p95" : 0.178534,
  "p98" : 0.227217,
  "p99" : 0.242554,
  "p999" : 0.420135,
  "stddev" : 0.046611762381930474,
  "m15_rate" : 0.2004491450567003,
  "m1_rate" : 2.8726563452698075,
  "m5_rate" : 0.5974045160056258,
  "mean_rate" : 0.010877725092634833,
  "duration_units" : "milliseconds",
  "rate_units" : "calls/second",
  "total" : 17.721825,
  "_type" : "timer"
} ],
  "resultCount" : 11,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "EXACT",
  "totalPagedResults" : 11,
  "remainingPagedResults" : -1
}
```

Vert.x monitoring is enabled by default to provide additional metrics for HTTP, TCP, and the internal component pool. The metrics provide low-level information about requests and responses, such as the number of bytes, duration, the number of concurrent requests, and so on.

3. Change the query to access metrics only for `myroute1`: https://ig.example.com:8443/openig/metrics/api?_prettyPrint=true&_sortKeys=_id&_queryFilter=_id+sw+"gateway._router.route.myroute1";.

   Note that metric for the router, `"_id" : "gateway._router.deployed-routes"`, is no longer displayed.

Monitor Vert.x metrics

NOTE

Vert.x metric names are deprecated and expected to be replaced with names ending in _total. The information provided by the metric is not deprecated. Other Prometheus metrics are not affected.

Vert.x monitoring is enabled by default to provide metrics for HTTP, TCP, and the internal component pool. The metrics provide low-level information about requests and responses, such as the number of bytes, duration, the number of concurrent requests, and so on.

To disable Vert.x monitoring, add the following lines to `admin.json`, and restart PingGateway:

```
{
  "vertx": {
    "metricsEnabled": false
  }
}
```

For more information, refer to AdminHttpApplication (`admin.json`).

Protect monitoring endpoints

By default, no special credentials or privileges are required for read-access to the Prometheus Scrape Endpoint and Common REST Monitoring Endpoint.

To protect the monitoring endpoints, add an `admin.json` file to your configuration, with a filter declared in the heap and named `MetricsProtectionFilter`. The following procedure gives an example of how to manage access to the monitoring endpoints.

1. Set up the procedure in Monitor at the Prometheus Scrape Endpoint, query the Prometheus Scrape Endpoint, and note that metrics for `myroute1` and `_router` are displayed:

```
$ curl -v
"https://ig.example.com:8443/openig/metrics/prometheus"
```

2. Add the following script to the PingGateway configuration:

Linux | Windows

```
$HOME/.openig/scripts/groovy/BasicAuthResourceServerFilter
.groovy
```

```
/*
 * This script is a simple implementation of HTTP basic access
authentication on
 * server side.
 * It expects the following arguments:
 *  - realm: the realm to display when the user agent prompts
for
 *    username and password if none were provided.
 *  - username: the expected username
 *  - passwordSecretId: the secretId to find the password
 *  - secretsProvider: the SecretsProvider to query for the
password
*/
import static
org.forgerock.util.promise.Promises.newResultPromise;

import java.nio.charset.Charset;
import org.forgerock.util.encode.Base64;
import org.forgerock.secrets.Purpose;
import org.forgerock.secrets.GenericSecret;

String authorizationHeader =
request.getHeaders().getFirst("Authorization");
if (authorizationHeader == null) {
    // No credentials provided, return 401 Unauthorized
    Response response = new Response(Status.UNAUTHORIZED);
    response.getHeaders().put("WWW-Authenticate", "Basic
realm=\"" + realm + "\"");
    return newResultPromise(response);
}

return secretsProvider.getNamed(Purpose.PASSWORD,
passwordSecretId)
        .thenAsync(password -> {
            // Build basic authentication string ->
username:password
            StringBuilder basicAuthString = new
StringBuilder(username).append(":");
            password.revealAsUtf8{ p ->
basicAuthString.append(new String(p).trim()) };
            String expectedAuthorization = "Basic " +
Base64.encode(basicAuthString.toString().getBytes(Charset.defa
ultCharset()));
            // Incorrect credentials provided, return 403
forbidden
            if
(!expectedAuthorization.equals(authorizationHeader)) {
                return newResultPromise(new
Response(Status.FORBIDDEN));
            }
            // Correct credentials provided, continue.
            return next.handle(context, request);
        },
                noSuchSecretException -> { throw new
RuntimeException(noSuchSecretException); });
```

The script is a simple implementation of the HTTP basic access authentication scheme. For information about scripting filters and handlers, refer to Extend.

3. Add the following heap configuration for MetricsProtectionFilter to `admin.json`:

```
{
  ...,
  "heap": [
    {
      "name": "ClientHandler",
      "type": "ClientHandler"
    },
    {
      "name": "mySecretsProvider",
      "type": "Base64EncodedSecretStore",
      "config": {
        "secrets": {
          "password.secret.id": "cGFzc3dvcmQ="
        }
      }
    },
    {
      "name": "MetricsProtectionFilter",
      "type": "ScriptableFilter",
      "config": {
        "type": "application/x-groovy",
        "file": "BasicAuthResourceServerFilter.groovy",
        "args": {
          "realm": "/",
          "username": "myUsername",
          "passwordSecretId": "password.secret.id",
          "secretsProvider": "${heap['mySecretsProvider']}"
        }
      }
    }
  ],
  ...
}
```

Notice the following features of the configuration:

- The MetricsProtectionFilter uses the script to protect the monitoring endpoint.
- The MetricsProtectionFilter requires the username `myUsername`, and a password provided by the SecretsProvider in the heap.

4. Restart PingGateway.
5. Query the Prometheus Scrape Endpoint without providing credentials, and note that an HTTP 401 Unauthorized is returned:

```
$ curl -v
"https://ig.example.com:8443/openig/metrics/prometheus"
```

6. Query the Prometheus Scrape Endpoint by providing correct credentials, and note that metrics are displayed:

```
$ curl -v
"https://ig.example.com:8443/openig/metrics/prometheus" -u
myUsername:password
```

7. Query the Prometheus Scrape Endpoint by providing incorrect credentials`, and note that an HTTP 403 Forbidden is returned:

```
$ curl -v
"https://ig.example.com:8443/openig/metrics/prometheus" -u
myUsername:wrong-password
```

Manage sessions

For information about PingGateway sessions, refer to Sessions. Change PingGateway session properties in the following ways:

Mode	To change the session properties
Stateless sessions	Configure the JwtSession object in the route that processes a request, or in its ascending configuration. For example, define the `cookie` property to configure the PingGateway session name. `{ "name": "JwtSession", "type": "JwtSession", "config": { "cookie": { "name": "MY_SESSIONID" } } }`

Mode	To change the session properties

Stateful sessions	Change the `session` property in `admin.json`, and restart PingGateway.
	
	For example, add the following lines to `admin.json` to configure the PingGateway session name:
	
	`"session": {`
	`    "cookie": {`
	`      "name": "MY_SESSIONID"`
	`    }`
	`  }`

Manage logs

Log messages in PingGateway and third-party dependencies are recorded using the Logback implementation of the Simple Logging Facade for Java (SLF4J) API. The following log levels are supported: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `ALL`, and `OFF`. For a full description of the options for logging, refer to the Logback website.

Default logging behavior

By default, log messages are recorded with the following configuration:

- When PingGateway starts, log messages for PingGateway and third-party dependencies, such as the ForgeRock Common Audit framework, are displayed on the console and written to `$HOME/.openig/logs/route-system.log`, where `$HOME/.openig` is the instance directory.
- When a capture point for the default CaptureDecorator is defined in a route, for example, when `"capture: "all"` is set as a top-level attribute of the JSON, log messages for requests and responses passing through the route are written to a log file in `$HOME/.openig/logs`.

  When no capture point is defined in a route, only exceptions thrown during request or response processing are logged.

  For more information, refer to Capturing log messages for routes and CaptureDecorator.
- By default, log messages with the level `INFO` or higher are recorded, with the titles and the top line of the stack trace. Messages on the console are highlighted with a color related to their log level.

The content and format of logs in PingGateway is defined by the reference `logback.xml` delivered with PingGateway. This file defines the following configuration items for logs:

- A root logger to set the overall log level, and to write all log messages to the `SIFT` and `STDOUT` appenders.
- A `STDOUT` appender to define the format of log messages on the console.
- A `SIFT` appender to separate log messages according to the key `routeId`, to define when log files are rolled, and to define the format of log messages in the file.
- An exception logger, called `LogAttachedExceptionFilter`, to write log messages for exceptions attached to responses.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

  <!--
    Prevent log flow attacks, by limiting repeated log messages.

    Configuration properties:
     * AllowedRepetitions (int): Threshold above which repeated
messages are no longer logged.
     * CacheSize (int): When CacheSize is reached, remove the
oldest entry.
  -->
  <!--<turboFilter
class="ch.qos.logback.classic.turbo.DuplicateMessageFilter" />-->

  <!-- Allow configuration of JUL loggers within this file,
without performance impact -->
  <contextListener
class="ch.qos.logback.classic.jul.LevelChangePropagator" />

  <appender name="STDOUT"
class="ch.qos.logback.core.ConsoleAppender">
    <withJansi>true</withJansi>
    <encoder>
      <pattern>%nopex[%thread] %highlight(%-5level)
%boldWhite(%logger{35}) @%mdc{routeId:-system} -
%replace(%message){'([\r\n])(.)',
'$1[CONTINUED]$2'}%n%highlight(%replace(%rootException{short})
{'(^|[\r\n])(.)', '$1[CONTINUED]$2'})</pattern>
    </encoder>
  </appender>
```

```xml
  <appender name="SIFT"
class="ch.qos.logback.classic.sift.SiftingAppender">
    <discriminator>
      <key>routeId</key>
      <defaultValue>system</defaultValue>
    </discriminator>
    <sift>
      <!-- Create a separate log file for each <key> -->
      <appender name="FILE-${routeId}"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${instance.dir}/logs/route-${routeId}.log</file>

        <rollingPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
          <!-- Rotate files daily -->
          <fileNamePattern>${instance.dir}/logs/route-${routeId}-
%d{yyyy-MM-dd}.%i.log</fileNamePattern>

          <!-- each file should be at most 100MB, keep 30 days
worth of history, but at most 3GB -->
          <maxFileSize>100MB</maxFileSize>
          <maxHistory>30</maxHistory>
          <totalSizeCap>3GB</totalSizeCap>
        </rollingPolicy>

        <encoder>
          <pattern>%nopex%date{"yyyy-MM-dd'T'HH:mm:ss,SSSXXX",
UTC} | %-5level | %thread | %logger{20} | @%mdc{routeId:-system} |
%replace(%message%n%xException){'([\r\n])(.)', '$1[CONTINUED]$2'}
</pattern>
        </encoder>
      </appender>
    </sift>
  </appender>

  <!-- Disable logs of exceptions attached to responses by
defining 'level' to OFF -->
  <logger
name="org.forgerock.openig.filter.LogAttachedExceptionFilter"
level="INHERITED" />

  <root level="${ROOT_LOG_LEVEL:-INFO}">
    <appender-ref ref="SIFT" />
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

Using a custom Logback file

To change the logging behavior, create a new logback file at `$HOME/.openig/config/logback.xml`, and restart PingGateway. The custom Logback file overrides the default configuration.

To take into account edits to `logback.xml`, stop and restart PingGateway, or edit the `configuration` parameter to add a scan and an interval:

```
<configuration scan="true" scanPeriod="5 seconds">
```

The `logback.xml` file is scanned after both of the following criteria are met:

- The specified number of logging operations have occurred, where the default is 16.
- The `scanPeriod` has elapsed.

If the custom `logback.xml` contains errors, messages like these are displayed on the console but log messages are not recorded:

```
14:38:59,667 |-ERROR in
ch.qos.logback.core.joran.spi.Interpreter@20:72 …
14:38:59,690 |-ERROR in
ch.qos.logback.core.joran.action.AppenderRefAction …
```

Change the global log level

The global log level is set by default to `INFO` by the following line of the default `logback.xml`:

```
<root level="${ROOT_LOG_LEVEL:-INFO}">
```

The log level set in `logback.xml` supercedes the log level set by environment variables. When the global log level is not set in `logback.xml`, set the global log level.

- To persist the log level for all future PingGateway instances:
    - Add an environment variable in `$HOME/.openig/bin/env.sh`, where `$HOME/.openig` is the instance directory:

      ```
      export ROOT_LOG_LEVEL=DEBUG
      ```

- Alternatively, add a system property in `$HOME/.openig/bin/env.sh`, where `$HOME/.openig` is the instance directory:

  ```
  export JAVA_OPTS="-DROOT_LOG_LEVEL=DEBUG"
  ```

  If both an environment variable and system property is set, the system property takes precedence.

- To persist the log level for PingGateway instances launched from the same shell, add an environment variable in the shell before you start PingGateway:

  Linux | Windows

  ```
  $ export ROOT_LOG_LEVEL=DEBUG
  $ /path/to/PingGateway-2024.9.0/bin/start.sh $HOME/.openig
  ```

- To persist the log level for a single PingGateway instance:

  Linux | Windows

  ```
  $ export ROOT_LOG_LEVEL=DEBUG /path/to/PingGateway-
  2024.9.0/bin/start.sh $HOME/.openig
  ```

Change the log level for different object types

To change the log level for a single object type without changing it for the rest of the configuration, edit `logback.xml` to add a logger defined by the fully qualified class name or package name of the object, and set its log level.

The following line in `logback.xml` sets the ClientHandler log level to `ERROR`, but does not change the log level of other classes or packages:

```
<logger name="org.forgerock.openig.handler.ClientHandler"
level="ERROR" />
```

To facilitate debugging, in `logback.xml` add loggers defined by the fully qualified package name or class name of the object. For example, add loggers for the following feature:

Feature	Logger
OAuth 2.0 client authentication: • AuthorizationCodeOAuth2ClientFilter • ClientCredentialsOAuth2ClientFilter • ResourceOwnerOAuth2ClientFilter	`org.forgerock.secrets.oauth2`
Expression resolution	`org.forgerock.openig.el` `org.forgerock.openig.resolver`
WebSocket notifications	`org.forgerock.openig.tools.notifications.ws`
Session management with JwtSession	`org.forgerock.openig.jwt`
OAuth 2.0 and OpenID Connect and token resolution and validation	`org.forgerock.openig.filter.oauth2`
AM policies, SSO, CDSSO, and user profiles	`org.forgerock.openig.openam` `org.forgerock.openig.tools`
SAML	`org.forgerock.openig.handler.saml`
UMA	`org.forgerock.openig.uma`
WebSocket tunnelling	`org.forgerock.openig.websocket`

Secret resolution	`org.forgerock.secrets.propertyresolver`
	`org.forgerock.secrets.jwkset`
	`org.forgerock.secrets.keystore`
	`org.forgerock.secrets.oauth2`
	`org.forgerock.openig.secrets.Base64EncodedSecretStore`

AllowOnlyFilter	`org.forgerock.openig.filter.allow.AllowOnlyFilter.<filter_name>`
Condition of a route	`org.forgerock.openig.handler.router.RouterHandler`
Header field size	`io.vertx.core.http.impl.HttpServerImpl`

Change the character set and format of log messages

By default, logs use the system default character set where PingGateway is running.

TIP

If your logs might contain characters that are not in your system character set, edit `logback.xml` to change the `encoder` part of the SIFT appender.

The following lines add the date to log messages, and change the character set:

```
<encoder>
   <pattern>%d{yyyyMMdd-HH:mm:ss} | %-5level | %thread |
%logger{20} | %message%n%xException</pattern>
   <charset>UTF-8</charset>
</encoder>
```

For more information about what information you can include in the logs, and its format, refer to PatternLayoutEncoder and Layouts in the Logback documentation.

Log in scripts

The logger object provides access to a unique SLF4J logger instance for scripts. Events are logged as defined in by a dedicated logger in `logback.xml`, and are included in the logs with the name of with the scriptable object.

To log events for scripts:

- Add logger objects to the script to enable logging at different levels. For example, add some of the following logger objects:

  ```
  logger.error("ERROR")
  logger.warn("WARN")
  logger.info("INFO")
  logger.debug("DEBUG")
  logger.trace("TRACE")
  ```

- Add a logger to `logback.xml` to reference the scriptable object and set the log level. The logger is defined by the type and name of the scriptable object that references the script, as follows:
  - ScriptableFilter:
    `org.forgerock.openig.filter.ScriptableFilter.filter_name`
  - ScriptableHandler:
    `org.forgerock.openig.handler.ScriptableHandler.handler_name`
  - ScriptableThrottlingPolicy:
    `org.forgerock.openig.filter.throttling.ScriptableThrottlingPolicy.throttling_policy_name`

- ScriptableAccessTokenResolver: `org.forgerock.openig.filter.oauth2.ScriptableAccessTokenResolver.access_token_resolver_name`

For example, the following logger logs trace-level messages for a ScriptableFilter named `cors_filter`:

```
<logger
name="org.forgerock.openig.filter.ScriptableFilter.cors_filter"
level="TRACE" />
```

The resulting messages in the logs contain the name of the scriptable object:

```
14:54:38:307 | TRACE | http-nio-8080-exec-6 |
o.f.o.f.S.cors_filter | TRACE
```

Log the BaseUriDecorator

During setup and configuration, it can be helpful to display log messages from the BaseUriDecorator. To record a log message each time a request URI is rebased , edit `logback.xml` to add a logger defined by the fully qualified class name of the BaseUriDecorator appended by the name of the baseURI decorator:

```
<logger
name="org.forgerock.openig.decoration.baseuri.BaseUriDecorator.bas
eURI" level="TRACE" />
```

Each time a request URI is rebased, a log message similar to this is created:

```
12:27:40| TRACE | http-nio-8080-exec-3 | o.f.o.d.b.B.b.
{Router}/handler| Rebasing request to http://app.example.com:8081
```

Stop exception logging

To stop recording log messages for exceptions, edit `logback.xml` to set the level to `OFF`:

```
<logger
name="org.forgerock.openig.filter.LogAttachedExceptionFilter"
level="OFF" />
```

Capture the context or entity of messages for routes

To capture the context or entity of inbound and outbound messages for a route, or for an individual handler or filter in the route, configure a CaptureDecorator. Captured information is written to SLF4J logs.

IMPORTANT

During debugging, consider using a CaptureDecorator to capture the entity and context of requests and responses. However, increased logging consumes resources, such as disk space, and can cause performance issues. In production, reduce logging by disabling the CaptureDecorator properties `captureEntity` and `captureContext`, or setting `maxEntityLength`.

For more information about the decorator configuration, refer to CaptureDecorator.

Studio provides an easy way to capture messages while developing your configuration. The following image illustrates the capture points where you can log messages on a route:

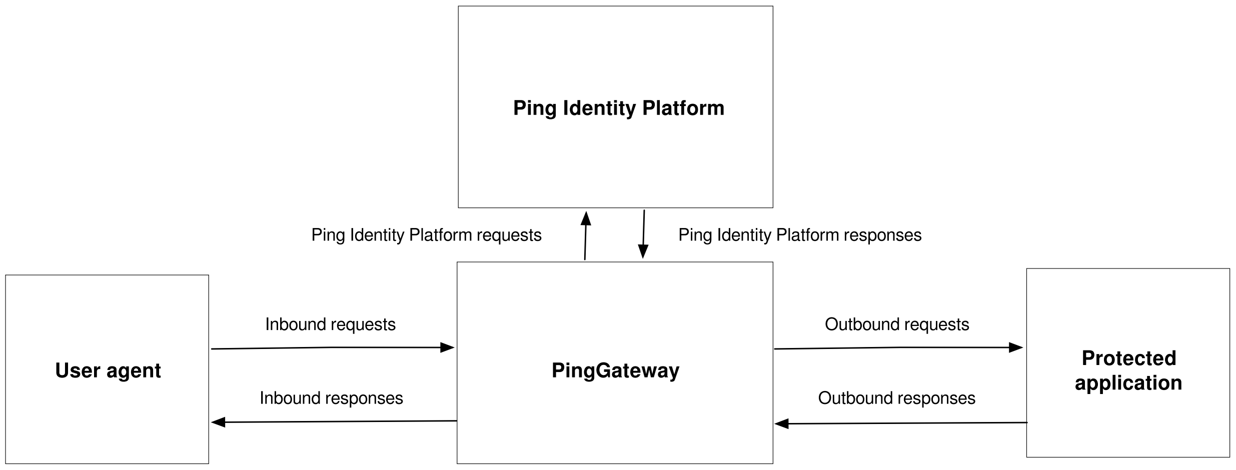

Figure 1. Capturing log messages for routes

Capture messages on a route in Studio

1. In Studio, select **ROUTES**, and then select a route with the icon.
2. On the left side of the screen, select **Capture**, and then select capture options. You can capture the body and context of messages passing to and from the user agent, the protected application, and the Ping Identity Platform.
3. Select **Deploy** to push the route to the PingGateway configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.
4. Access the route, and then check `$HOME/.openig/logs` for a log file named by the route, where `$HOME/.openig` is the instance directory. The log file should contain the messages defined by your capture configuration.

Limit repetitive log messages

To keep log files clean and readable, and to prevent log flow attacks, limit the number of repeat log messages. Add a custom `logback.xml` with a `DuplicateMessageFilter`. This filter detects duplicate messages, and after the specified number of repetitions, drops repeated messages.

The following example allows 5 repetitions of a log message, and holds the last 10 repeated messages in the cache:

```
<turboFilter
class="ch.qos.logback.classic.turbo.DuplicateMessageFilter"
allowedRepetitions="5" CacheSize="10" />
```

The DuplicateMessageFilter has the following limitations:

- Filters out **all** duplicate messages. It does not filter per logger, or logger instance, or logger name.
- Detects repetition of raw messages, meaning that the following example messages are considered as repetition:

  ```
  logger.debug("Hello {}.", name0);
  logger.debug("Hello {}.", name1);
  ```

- Doesn't limit the lifespan of the cache. After the specified number of repetitions is reached, the repeated log messages never appear again, even if they're frequently hit.

Tune performance

Tune deployments in the following steps:

1. Consider the issues that impact the performance of a deployment. See Define requirements and constraints.
2. Tune and test the downstream servers and applications:
   a. Tune the downstream web container and JVM to achieve performance targets.
   b. Test downstream servers and applications in a pre-production environment, under the expected load, and with common use cases.
3. Increase hardware resources as required, and then re-tune the deployment.

Define requirements and constraints

When you consider performance requirements, bear in mind the following points:

- The capabilities and limitations of downstream services or applications on your performance goals.
- The increase in response time due to the extra network hop and processing, when PingGateway is inserted as a proxy in front of a service or application.
- The constraint that downstream limitations and response times place on PingGateway.

Service level objectives

A service level objective (SLO) is a target that you can measure quantitatively. Where possible, define SLOs to set out what performance your users expect. Even if your first version of an SLO consists of guesses, it is a first step towards creating a clear set of measurable goals for your performance tuning.

When you define SLOs, bear in mind that PingGateway can depend on external resources that can impact performance, such as AM’s response time for token validation, policy evaluation, and so on. Consider measuring remote interactions to take dependencies into account.

Consider defining SLOs for the following metrics of a route:

- Average response time for a route.

  The response time is the time to process and forward a request, and then receive, process, and forward the response from the protected application.

  The average response time can range from less than a millisecond, for a low latency connection on the same network, to however long it takes your network to deliver the response.
- Distribution of response times for a route.

  Because applications set timeouts based on worst case scenarios, the distribution of response times can be more important than the average response time.
- Peak throughput.

  The maximum rate at which requests can be processed at peak times. Because applications are limited by their peak throughput, this SLO is arguably more important than an SLO for average throughput.
- Average throughput.

  The average rate at which requests are processed.

Metrics are returned at the monitoring endpoints. For information about monitoring endpoints, refer to Monitoring. For examples of how to set up monitoring in

PingGateway, refer to Monitor services.

Available resources

With your defined SLOs, inventory the server, networks, storage, people, and other resources. Estimate whether it is possible to meet the requirements, with the resources at hand.

Benchmarks

Before you can improve the performance of your deployment, establish an accurate benchmark of its current performance. Consider creating a deployment scenario that you can control, measure, and reproduce.

For information about running Ping Identity Platform benchmark tests, refer to the ForgeOps documentation on benchmarks. Adapt the scenarios as necessary for your PingGateway deployment.

Tune PingGateway

Consider the following recommendations for improving performance, throughput, and response times. Adjust the tuning to your system workload and available resources, and then test suggestions before rolling them out into production.

Logs

Log messages in PingGateway and third-party dependencies are recorded using the Logback implementation of the Simple Logging Facade for Java (SLF4J) API. By default, logging level is INFO.

To reduce the number of log messages, consider setting the logging level to `error`. For information, refer to Manage logs.

Buffering message content

PingGateway creates a TemporaryStorage object to buffer content during processing. For information about this object and its default values, refer to TemporaryStorage.

Messages bigger than the buffer size are written to disk, consuming I/O resources and reducing throughput.

The default size of the buffer is 64 KB. If the number of concurrent messages in your application is generally bigger than the default, consider allocating more heap memory or changing the initial or maximum size of the buffer.

To change the values, add a TemporaryStorage object named `TemporaryStorage`, and use non-default values.

Caches

When caches are enabled, PingGateway can reuse cached information without making additional or repeated queries for the information. This gives the advantage of higher system performance, but the disadvantage of lower trust in results.

When caches are disabled, PingGateway must query a data store each time it needs data. This gives the disadvantage of lower system performance, and the advantage of higher trust in results.

All caches provide similar configuration properties for timeout, defining the duration to cache entries. When the timeout is lower, the cache is evicted more frequently, and consequently, the performance is lower but the trust in results is higher.

When you configure caches in PingGateway, make choices to balance your required performance with your security needs.

Learn more about PingGateway caches in Caches.

WebSocket notifications

By default, PingGateway receives WebSocket notifications from AM for the following events:

- When a user logs out of AM, or when the AM session is modified, closed, or times out. PingGateway can use WebSocket notifications to evict entries from the session cache. For an example of setting up session cache eviction, refer to Session cache eviction.
- When AM creates, deletes, or changes a policy decision. PingGateway can use WebSocket notifications to evict entries from the policy cache. For an example of setting up policy cache eviction, refer to Notifications and the policy cache.
- When PingGateway automatically renews a WebSocket connection to AM. To configure WebSocket renewal, refer to the `notifications.renewalDelay` property of AmService.

If the WebSocket connection is lost, during that time the WebSocket is not connected, PingGateway behaves as follows:

- Responds to session service calls with an empty SessionInfo result.

  When the SingleSignOn filter recieves an empty SessionInfo call, it concludes that the user is not logged in, and triggers a login redirect.
- Responds to policy evaluations with a deny policy result.

By default, PingGateway waits for five seconds before trying to re-establish the WebSocket connection. If it can't re-establish the connection, it keeps trying every five seconds.

To disable WebSocket notifications, or change any of the parameters, configure the `notifications` property in AmService. For information, refer to AmService.

Tune the ClientHandler/ReverseProxyHandler

The ClientHandler/ReverseProxyHandler communicates as a client to a downstream third-party service or protected application. The performance of the communication is determined by the following parameters:

- The number of available connections to the downstream service or application.
- The connection timeout, which is the maximum time to connect to a server-side socket before timing out and abandoning the connection attempt.
- The socket timeout, which is the maximum time a request can take before a response is received after which the request is deemed to have failed.

Configure PingGateway in conjunction with PingGateway's first-class Vert.x configuration, and the `vertx` property of `admin.json`. For more information, refer to AdminHttpApplication (`admin.json`).

Vert.x options for tuning

Object	Vert.x Option	Description
PingGateway (first-class)	`gatewayUnits`	The number of deployed Vert.x Verticles. This setting is effectively the number of cores that PingGateway operates across, or in other words, the number of available threads. Each instance operates on the same port on its own event-loop thread. Default: Number of available cores. (This is the optimal value.)

Object	Vert.x Option	Description
root.vertx	`eventLoopPoolSize`	The size of the pool available to service Verticles for event-loop threads. To guarantee that a single thread handles all I/O events for a single request or response, PingGateway deploys a Verticle onto each event loop. Configure `eventLoopPoolSize` to be greater than or equal to `gatewayUnits`. Default: 2 * number of available cores. For more information, refer to Reactor and Multi-Reactor.
root.connectors.<connector>.vertx	`acceptBacklog`	The maximum number of connections to queue before refusing requests. Default: 1024
	`sendBufferSize`	The TCP connection send buffer size. Set this property according to the available RAM and required number of concurrent connections. Default: Use the Java TCP send buffer size default settings that Java inherits from the operating system.
	`receiveBufferSize`	The TCP receive buffer size. Set this property according to the available RAM and required number of concurrent connections. Default: Use the Java TCP receive buffer size default settings that Java inherits from the operating system.
	`maxHeaderSize`	Set this property when HTTP headers manage large values (such as JWT). Default: 8 KB (8,192 bytes)

Vert.x options for troubleshooting performance

Object	Vert.x Option	Description
root.vertx	`blockedThreadCheckInterval` and `blockedThreadCheckIntervalUnit`	The interval at which Vert.x checks for blocked threads and logs a warning. Default: 1 second
	`maxEventLoopExecuteTime` and `maxEventLoopExecuteTimeUnit`	The maximum execution time before Vert.x logs a warning. Default: 2 seconds
	`warningExceptionTime` and `warningExceptionTimeUnit`	The threshold at which warning logs are accompanied by a stack trace to identify cause. Default: 5 seconds
	`logActivity`	Whether to log network activity. Default: `false`

Set the maximum number of file descriptors and processes per user

Each PingGateway instance in your environment should have access to at least 65,536 file descriptors to handle multiple client connections.

Ensure that every PingGateway instance is allocated enough file descriptors. For example, use the **`ulimit -n`** command to check the limits for a particular user:

```
$ su - iguser
$ ulimit -n
```

It may also be necessary to increase the number of processes available to the user running the PingGateway processes.

For example, use the **`ulimit -u`** command to check the process limits for a user:

```
$ su - iguser
$ ulimit -u
```

IMPORTANT

IMPORTANT

Before increasing the file descriptors for the PingGateway instance, ensure that the total amount of file descriptors configured for the operating system is higher than 65,536.

If the PingGateway instance uses all of the file descriptors, the operating system will run out of file descriptors. This may prevent other services from working, including those required for logging in the system.

Refer to your operating system's documentation for instructions on how to display and increase the file descriptors or process limits for the operating system and for a given user.

Tune PingGateway's JVM

Start tuning the JVM with default values, and monitor the execution, paying particular attention to memory consumption, and GC collection time and frequency. Incrementally adjust the configuration, and retest to find the best settings for memory and garbage collection.

Make sure there is enough memory to accommodate the peak number of required connections, and make sure timeouts in PingGateway and its container support latency in downstream servers and applications.

PingGateway makes low memory demands, and consumes mostly YoungGen memory. However, using caches, or proxying large resources, increases the consumption of OldGen memory. For information about how to optimize JVM memory, refer to the Oracle documentation.

Consider these points when choosing a JVM:

- Find out which version of the JVM is available. More recent JVMs usually contain performance improvements, especially for garbage collection.
- Choose a 64-bit JVM if you need to maximize available memory.

Consider these points when choosing a GC:

- Test GCs in realistic scenarios, and load them into a pre-production environment.
- Choose a GC that is adapted to your requirements and limitations. Consider comparing the *Garbage-First Collector (G1)* and *Parallel GC* in typical business use cases.

  The G1 is targeted for multi-processor environments with large memories. It provides good overall performance without the need for additional options. The G1 is designed to reduce garbage collection, through low-GC latency. It is largely self-tuning, with an adaptive optimization algorithm.

The Parallel GC aims to improve garbage collection by following a high-throughput strategy, but it requires more full garbage collections.

Learn more in Best practice for JVM Tuning with G1 GC.

Rotate keys

The following sections give an overview of how to manage rotation of encryption keys and signing keys, and include examples for key rotation based on use cases from the Gateway guide.

About key rotation

Key rotation is the process of generating a new version of a key, assigning that version as the *active* key to encrypt or sign new messages, or as a *valid* key to decrypt or validate messages, and then deprovisioning the old key.

Why and when to rotate keys

Regular key rotation is a security consideration that is sometimes required for internal business compliance. Regularly rotate keys to:

- Limit the amount of data protected by a single key.
- Reduce dependence on specific keys, making it easier to migrate to stronger algorithms.
- Prepare for when a key is compromised. The first time you try key rotation shouldn’t be during a real-time recovery.

Key revocation is a type of key rotation, done exceptionally if you suspect that a key has been compromised. To decide when to revoke a key, consider the following points:

- If limited use of the old keys can be tolerated, provision the new keys and then deprovision the old keys. Messages produced before the new keys are provisioned are impacted.
- If use of the old keys cannot be tolerated, deprovision the old keys before you provision the new keys. The system is unusable until new keys are provisioned.

Steps for rotating symmetric keys

The following steps outline key rotation and revocation for symmetric keys managed by a KeyStoreSecretStore. For an example, refer to Rotate keys in a shared JWT session.

1. Using OpenSSL, Keytool, or another key creation mechanism, create the new symmetric key. The keystore should contain the old key and the new key.
2. Provision the new key.
    a. In the `mappings` property of KeyStoreSecretStore, add the alias for the new key after the alias for the old key. The new key is now valid. Because the old key is the first key in the list, it is the active key.
    b. Move the new key to be the first key in the list. The new key is now the active key.
3. Deprovision the old key.

    To ensure that no messages or users are impacted, wait until messages encrypted or signed with the old key are out of the system before you deprovision the old key.

    a. In the `mappings` property of KeyStoreSecretStore, delete the alias for the old key. The old key can no longer be used.
    b. Using OpenSSL, Keytool, or another key creation mechanism, delete the old symmetric key.

Steps for rotating asymmetric keys

The following steps outline the process for key rotation and revocation for asymmetric keys managed by a KeyStoreSecretStore or HsmSecretStore. For an example, refer to Rotate keys for stateless access tokens signed with a KeyStoreSecretStore.

1. Create new asymmetric keys for signing and encryption, using OpenSSL, Keytool, or another key creation mechanism.
2. Provision the message consumer with the private portion of the new encryption key, and the public portion of the new signing key.

    The message consumer can now decrypt and verify messages with the old key and the new key.

3. Provision the message producer, with the public portion of the new encryption key, and the private portion of the signing key. The message producer starts encrypting and signing messages with the new key, and stops using the old key.
4. Deprovision the message consumer with the private portion of the old encryption key, and the public portion of the old signing key. The message consumer can no longer decrypt and verify messages with the old key.

    To ensure that no messages or users are impacted, wait until messages encrypted or signed with the corresponding old key are out of the system before you deprovision the old key.

5. Deprovision the message producer, with the public portion of the old encryption key, and the private portion of old signing key.

Key rotation for keys in a JWK set

When keys are provided by a JWK Set from AM, the key rotation is transparent to PingGateway. AM generates a key ID (`kid`) for each key it exposes at the `jwk_uri` . For more information, refer to Mapping and rotating secrets in AM's *Security guide*.

When PingGateway processes a request with a JWT containing a `kid` , PingGateway uses the `kid` to identify the key in the JWK Set. If the `kid` is available at the `jwk_uri` on AM, PingGateway processes the request. Otherwise, PingGateway tries all compatible secrets from the JWK Set. If none of the secrets work, the JWT is rejected.

Rotate keys for stateless access tokens signed with a KeyStoreSecretStore

This example extends the example in Signed tokens with KeyStoreSecretStore to rotate the keys that sign an access token and verify the signature.

Rotate Keys For Stateless Access Tokens Signed With a KeyStoreSecretStore
Before you start, set up and test the example in Signed tokens with KeyStoreSecretStore.

1. Set up the new keys:
    a. Generate a new private key called `signature-key-new` , and a corresponding public certificate called `x509certificate-new.pem` :

    ```
    $ openssl req -x509 \
    -newkey rsa:2048 \
    -nodes \
    -subj
    "/CN=ig.example.com/OU=example/O=com/L=fr/ST=fr/C=fr" \
    -keyout keystore_directory/signature-key-new.key \
    -out keystore_directory/x509certificate-new.pem \
    -days 365

    ... writing new private key to
    'keystore_directory/signature-key-new.key'
    ```

    b. Convert the private key and certificate files into a new PKCS#12 keystore file:

    ```
    $ openssl pkcs12 \
    -export \
    -in keystore_directory/x509certificate-new.pem \
    -inkey keystore_directory/signature-key-new.key \
    -out keystore_directory/keystore-new.p12 \
    ```

```
-passout pass:password \
-name signature-key-new
```

c. List the keys in the new keystore:

```
$ keytool -list \
-keystore "keystore_directory/keystore-new.p12" \
-storepass "password" \
-storetype PKCS12


...
Your keystore contains 1 entry
Alias name: signature-key-new
```

d. Import the new keystore into `keystore.p12`, so that `keystore.p12` contains both keys:

```
$ keytool -importkeystore
-srckeystore keystore_directory/keystore-new.p12
-srcstoretype pkcs12
-srcstorepass password
-destkeystore keystore_directory/keystore.p12
-deststoretype pkcs12
-deststorepass password

Entry for alias signature-key-new successfully imported
...
```

e. List the keys in `keystore.p12`, to make sure it contains the new and old keys:

```
$ keytool -list \
-keystore "keystore_directory/keystore.p12" \
-storepass "password" \
-storetype PKCS12


...
Your keystore contains 2 entries
Alias name: signature-key
Alias name: signature-key-new
```

2. Set up AM:

   a. Copy the updated keystore to AM:

      i. Copy `keystore.p12` to AM:

```
$ cp keystore_directory/keystore.p12
am_keystore_directory/AM_keystore.p12
```

ii. List the keys in the updated AM keystore:

```
$ keytool -list \
-keystore "am_keystore_directory/AM_keystore.p12" \
-storepass "password" \
-storetype PKCS12

...
Your keystore contains 2 entries
Alias name: signature-key
Alias name: signature-key-new
```

iii. Restart AM to update the keystore cache.

b. Update the KeyStoreSecretStore on AM:

i. In AM, select **Secret Stores** > keystoresecretstore.

ii. Select the **Mappings** tab, and in `am.services.oauth2.stateless.signing.RSA` add the alias `signature-key-new`.

The mapping now contains two aliases, but the alias `signature-key` is still the active alias. AM still uses `signature-key` to sign tokens.

iii. Drag `signature-key-new` above `signature-key`.

AM now uses `signature-key-new` to sign tokens.

3. Set up PingGateway:

a. Set up PingGateway for HTTPS, as described in Configure PingGateway for TLS (server-side).

b. Import the public certificate to the PingGateway keystore, with the alias `verification-key-new`:

```
$ keytool -import \
-trustcacerts \
-rfc \
-alias verification-key-new \
-file "keystore_directory/x509certificate-new.pem" \
-keystore "ig_keystore_directory/IG_keystore.p12" \
-storetype PKCS12 \
-storepass "password"

...
```

```
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

c. List the keys in the PingGateway keystore:

```
$ keytool -list \
-keystore "ig_keystore_directory/IG_keystore.p12" \
-storepass "password" \
-storetype PKCS12

...
Your keystore contains 2 entries
Alias name: verification-key
Alias name: verification-key-new
```

d. In `rs-stateless-signed-ksss.json`, edit the KeyStoreSecretStore mapping with the new verification key:

```
"mappings": [
  {
    "secretId": "stateless.access.token.verification.key",
    "aliases": [ "verification-key", "verification-key-
new" ]
  }
]
```

If the Router `scanInterval` is disabled, restart PingGateway to reload the route.

PingGateway can now check the authenticity of access tokens signed with `verification-key`, the old key, and `verification-key-new`, the new key. However, AM signs with the old key.

4. Test the setup:

a. Get an access token for the demo user, using the scope `myscope`:

```
$ mytoken=$(curl -s \
--user "client-application:password" \
--data
"grant_type=password&username=demo&password=Ch4ng31t&scope
=myscope" \
http://am.example.com:8088/openam/oauth2/access_token | jq
-r ".access_token")
```

b. Display the token:

```
$ echo ${mytoken}
```

c. Access the route by providing the token returned in the previous step:

```
$ curl -v \
--cacert /path/to/secrets/ig.example.com-certificate.pem \
--header "Authorization: Bearer ${mytoken}" \
https://ig.example.com:8443/rs-stateless-signed-ksss



...
Decoded access_token: {
sub=demo,
cts=OAUTH2_STATELESS_GRANT,
...
```

Deprovision Old Keys

1. Remove `signature-key` from the AM keystore:

   a. Delete the key from the keystore:

```
$ keytool -delete \
-keystore "am_keystore_directory/AM_keystore.p12" \
-storepass "password" \
-alias signature-key
```

   b. List the keys in the AM keystore to make sure `signature-key` is removed:

```
$ keytool -list \
-keystore "am_keystore_directory/AM_keystore-new.p12" \
-storepass "password" \
-storetype PKCS12
```

   c. Restart AM.

2. Remove `verification-key` from the PingGateway keystore:

   a. Delete the key from the keystore:

```
$ keytool -delete \
-keystore "ig_keystore_directory/IG_keystore.p12" \
-storepass "password" \
-alias verification-key
```

   b. List the keys in the PingGateway keystore to make sure that `verification-key` is removed:

```
$ keytool -list \
-keystore "ig_keystore_directory/IG_keystore.p12" \
-storepass "password" \
-storetype PKCS12
```

3. In AM, delete the mapping for `signature-key` from `keystoresecretstore`.
4. In PingGateway, delete the mapping for `verification-key` from the route `rs-stateless-signed-ksss.json`. If the Router `scanInterval` is disabled, restart PingGateway to reload the route.

Rotate keys in a shared JWT session

This section builds on the example in Share JWT session between multiple instances of PingGateway to rotate a key used in a shared JWT session.

When a JWT session is shared between multiple instances of PingGateway, the instances are able to share the session information for load balancing and failover.

Before you start, set up the example in Set up shared secrets for multiple instances of PingGateway, where three instances of PingGateway share a JwtSession and use the same authenticated encryption key. Instance 1 acts as a load balancer, and generates a session. Instances 2 and 3 access the session information.

1. Test the setup with the existing key, `symmetric-key`:
   a. Access instance 1 to generate a session:

   ```
   $ curl -v http://ig.example.com:8001/log-in-and-generate-
   session

   GET /log-in-and-generate-session HTTP/1.1
   ...

   HTTP/1.1 200 OK
   Content-Length: 84
   Set-Cookie: IG=eyJ...HyI; Path=/; Domain=.example.com;
   HttpOnly
   ...
   Sam Carter logged IN. (JWT session generated)
   ```

   b. Using the JWT cookie returned in the previous step, access instance 2:

   ```
   $ curl -v http://ig.example.com:8001/webapp/browsing?one -
   -header "cookie:IG=<JWT cookie>"
   ```

```
GET /webapp/browsing?one HTTP/1.1
...
cookie: IG=eyJ...QHyI
...
HTTP/1.1 200 OK
...
Hello, Sam Carter !! (instance2)
```

Note that instance 2 can access the session info.

c. Using the JWT cookie again, access instance 3:

```
$ curl -v http://ig.example.com:8001/webapp/browsing?two -
-header "Cookie:IG=<JWT cookie>"

GET /webapp/browsing?two HTTP/1.1
...
cookie: IG=eyJ...QHyI
...
HTTP/1.1 200 OK
...
Hello, Sam Carter !! (instance3)
```

Note that instance 3 can access the session info.

2. Commission a new key:

   a. Generate a new encryption key, called `symmetric-key-new`, in the existing keystore:

```
$ keytool \
-genseckey \
-alias symmetric-key-new
-keystore /path/to/secrets/jwtsessionkeystore.pkcs12 \
-storepass password \
-storetype PKCS12 \
-keyalg HmacSHA512 \
-keysize 512
```

   b. Make sure the keystore contains the old key and the new key:

```
$ keytool \
-list \
-keystore /path/to/secrets/jwtsessionkeystore.pkcs12 \
-storepass password \
-storetype PKCS12
```

```
...
Your keystore contains 2 entries
symmetric-key, ...
symmetric-key-new ...
```

c. Add the key alias to `instance1-loadbalancer.json`, `instance2-retrieve-session-username.json`, and `instance3-retrieve-session-username.json`, for each PingGateway instance, as follows:

```
"mappings": [{
  "secretId": "jwtsession.encryption.secret.id",
  "aliases": ["symmetric-key", "symmetric-key-new"]
}]
```

If the Router `scanInterval` is disabled, restart PingGateway to reload the route.

The active key is `symmetric-key`, and the valid key is `symmetric-key-new`.

d. Test the setup again, as described in step 1, and make sure instances 2 and 3 can still access the session information.

3. Make the new key the active key for generating sessions:

a. In `instance1-loadbalancer.json`, change the order of the keys to make `symmetric-key-new` the active key, and `symmetric-key` the valid key:

```
"mappings": [{
  "secretId": "jwtsession.encryption.secret.id",
  "aliases": ["symmetric-key-new", "symmetric-key"]
}]
```

Don’t change `instance2-retrieve-session-username.json` or `instance3-retrieve-session-username.json`.

b. Test the setup again, as described in step 1, and make sure instances 2 and 3 can still access the session information.

Instance 1 creates the session using the new active key, `symmetric-key-new`.

Because `symmetric-key-new` is declared as a valid key in instances 2 and 3, the instances can still access the session. It isn’t necessary to make `symmetric-key-new` the active key.

4. Decommission the old key:

a. Remove the old key from all of the routes, as follows:

```
"mappings": [{
  "secretId": "jwtsession.encryption.secret.id",
  "aliases": ["symmetric-key-new"]
}]
```

Key `symmetric-key-new` is the only key in the routes.

b. Remove the old key, `symmetric-key`, from the keystore:

   i. Delete `symmetric-key`:

   ```
   $ keytool \
   -delete \
   -alias symmetric-key \
   -keystore /path/to/secrets/jwtsessionkeystore.pkcs12 \
   -storepass password \
   -storetype PKCS12 \
   -keypass password
   ```

   ii. Make sure the keystore contains only `symmetric-key-new`:

   ```
   $ keytool \
   -list \
   -keystore /path/to/secrets/jwtsessionkeystore.pkcs12 \
   -storepass password \
   -storetype PKCS12

   ...
   Your keystore contains 1 entry
   symmetric-key-new ...
   ```

c. Test the setup again, as described in step 1, and make sure instances 2 and 3 can still access the session information.

Troubleshoot

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to help you set up and maintain your deployments.

Getting support

Ping Identity provides support services, professional services, training, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see https://www.pingidentity.com.

Ping Identity has staff members around the globe who support our international customers and partners. For details on Ping Identity's support offering, visit https://www.pingidentity.com/support.

Ping Identity publishes comprehensive documentation online:

- The Ping Identity Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage Ping Identity Platform software.

  While many articles are visible to everyone, Ping Identity customers have access to much more, including advanced information for customers using Ping Identity Platform software in a mission-critical capacity.
- Ping Identity product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

Getting info about the problem

When trying to solve a problem, save time by asking the following questions:

- How do you reproduce the problem?
- What behavior do you expect, and what behavior do you have?
- When did the problem start occurring?
- Are their circumstances in which the problem does not occur?
- Is the problem permanent, intermittent, getting better, getting worse, or staying the same?

If you contact ForgeRock for help, include the following information with your request:

- The product version and build information. This information is included in the logs when PingGateway starts up. If PingGateway is running in development mode, and set up as described in the Quick install, access the information at http://ig.example.com:8080/openig/api/info or https://ig.example.com:8443/openig/api/info.
- Description of the problem, including when the problem occurs and its impact on your operation.
- Steps you took to reproduce the problem.
- Relevant access and error logs, stack traces, and core dumps.
- Description of the environment, including the following information:
  - Machine type
  - Operating system and version

- Web server or container and version
- Java version
- Patches or other software that might affect the problem

Start up

After a first startup, PingGateway doesn’t restart or load routes.

If PingGateway doesn’t restart or load routes after a first startup, search `route-system.log` for lines containing `Error while starting…`or `Unable to start …` and use the error message to debug the issue.

PID file already exists

If PingGateway shuts down without using the `stop.sh` or `stop.bat` script, the PID file isn’t removed and PingGateway can’t restart. This can happen when you use the PingGateway service to stop or restart PingGateway, or when PingGateway is deployed in Docker.

Remove the PID file or change the configuration as described in Allow startup when there is an existing PID file.

Resources

Requests redirected to AM instead of to the resource

By default, AM 5 and later writes cookies to the fully qualified domain name of the server; for example, `am.example.com`. Therefore, a host-based cookie, rather than a domain-based cookie, is set.

Consequently, after authentication through PingAM, requests can be redirected to PingAM instead of to the resource.

To resolve this issue, add a cookie domain to the PingAM configuration. For example, in the AM admin UI, go to **Configure** > **Global Services** > **Platform**, and add the domain `example.com`.

Sample application not displayed correctly

When the sample application is used with PingGateway in the documentation examples, the sample application must serve static resources, such as the .css. Add the following route to the PingGateway configuration:

```
{
  "name" : "00-static-resources",
  "baseURI" : "http://app.example.com:8081",
  "condition": "${find(request.uri.path,'^/css') or
matchesWithRegex(request.uri.path, '^/.*\\\\.ico$') or
matchesWithRegex(request.uri.path, '^/.*\\\\.gif$')}",
  "handler": "ReverseProxyHandler"
}
```

▼ StaticResponseHandler results in a blank page

Define an entity for the response, as in the following example:

```
{
  "name": "AccessDeniedHandler",
  "type": "StaticResponseHandler",
  "config": {
    "status": 403,
    "headers": {
      "Content-Type": [ "text/html; charset=UTF-8" ]
    },
    "entity": "<html><body><p>User does not have permission</p>
</body></html>"
  }
}
```

Routes

▼ No handler to dispatch to

Symptom

The following errors are in `route-system.log`:

```
... | ERROR | main | o.f.o.h.r.RouterHandler | no handler to
dispatch to
```

```
08:22:54:974 | ERROR | http-... | o.f.o.h.DispatchHandler |
no handler to dispatch to for URI
'http://ig.example.com/demo'
```

Cause

PingGateway is not configured to handle the incoming request or the request to the specified URI:

- "no handler to dispatch to": the router cannot find a route that accepts the incoming request. This error happens when none of the route conditions match the incoming request and there is no default route.
- "no handler to dispatch to for URI": the router cannot find a route that can handle the request to the specified URI because none of the route conditions

match the request path (URI).

Solution

If the errors occur during the startup, they are safe to ignore. If the errors occur after the startup, do the following:

- Identify why the request matched none of the Route conditions, and adapt the conditions. For examples, refer to Example conditions and requests.
- Add a default handler to the Router.
- Add a default route for when no condition is met.

▼ Object not found in heap

If you have the following error, you have specified `"handler": "Router2"` in `config.json` or in the route, but no handler configuration object named Router2 exists:

```
org.forgerock.json.fluent.JsonValueException: /handler:
     object Router2 not found in heap
     at
org.forgerock.openig.heap.HeapImpl.resolve(HeapImpl.java:351)
     at
org.forgerock.openig.heap.HeapImpl.resolve(HeapImpl.java:334)
     at
org.forgerock.openig.heap.HeapImpl.getHandler(HeapImpl.java:538)
```

Make sure you have added an entry for the handler, and that you have correctly spelled its name.

▼ Extra or missing character / invalid JSON

When the JSON for a route is not valid, PingGateway does not load the route. Instead, a description of the error appears in the log.

Use a JSON editor or JSON validation tool such as JSONLint to make sure your JSON is valid.

▼ Route not used

PingGateway loads all configurations at startup, and, by default, periodically reloads changed route configurations.

If you make changes to a route that result in an invalid configuration, PingGateway logs errors, but it keeps the previous, correct configuration, and continues to use the old route.

PingGateway only uses the new configuration after you save a valid version or when you restart PingGateway.

Of course, if you restart PingGateway with an invalid route configuration, then PingGateway tries to load the invalid route at startup and logs an error. In that case, if there is no default handler to accept any incoming request for the invalid route, then you have an error, `No handler to dispatch to`.

▼ Skipped routes

PingGateway returns an exception if it loads a route for which it can't resolve a requirement. For example, when you load a route that uses an AmService object, the object must be available in the AM configuration.

If you add routes to a configuration when the environment is not ready, rename the route to prevent PingGateway from loading it. For example, rename a route as follows:

```
$ mv $HOME/.openig/config/routes/03-sql.json
$HOME/.openig/config/routes/03-sql.inactive
```

If necessary, restart PingGateway to reload the configuration. When you have configured the environment, change the file extension back to `.json`.

Studio

▼ Can't deploy routes in Studio

Studio deploys and undeploys routes through a main router named `_router`, which is the name of the main router in the default configuration. If you use a custom `config.json`, make sure it contains a main router named `_router`.

For information about creating routes in Studio, refer to the Studio guide.

Timeout errors

▼ Log is flushed with timeout exception warnings on sending a request

Problem: After a request is sent to PingGateway, PingGateway seems to hang. An HTTP 502 Bad Gateway error is produced, and the PingGateway log is flushed with SocketTimeoutException warnings.

Possible cause: The `baseURI` configuration is missing or causes the request to return to PingGateway, so PingGateway can't produce a response to the request.

Possible solution: Configure the `baseURI` to use a different host and port to PingGateway.

Other problems

▼ Incorrect values in the flat files

Make sure the user running PingGateway can read the flat file. Remember that values include spaces and tabs between the separator, so make sure the values are not padded with spaces.

▼ Problem accessing URLs

The following error can be encountered when using an `AssignmentFilter` as described in AssignmentFilter and setting a string value for one of the headers.

```
HTTP ERROR 500
      Problem accessing /myURL . Reason:
      java.lang.String cannot be cast to java.util.List
      Caused by:
      java.lang.ClassCastException: java.lang.String cannot be
cast to java.util.List
```

All headers are stored in lists so the header must be addressed with a subscript. For example, rather than trying to set `request.headers['Location']` for a redirect in the response object, you should instead set `request.headers['Location'][0]`. A header without a subscript leads to the error above.

▼ URI Too Long error

When a request is longer than 4096 bytes, it can cause an HTTP 414 URI Too Long response.

The default limit for request length is set by the Vert.x configuration `DEFAULT_MAX_INITIAL_LINE_LENGTH`. This default acts on the `connectors` property of admin.json.

When working with requests constructed with parameters and query strings, such as for SAML or token transformation, where the request can become long consider setting the Vert.x property `getMaxInitialLineLength` to increase the limit.

The following example configuration in `admin.json` increases the request length limit to 9999 bytes:

```
"connectors": [
  {
    "vertx": {
      "maxInitialLineLength": 9999,
      ...
    }
  }
]
```

▼ "Ignored" message logged

The following log message indicates that the client or server side has disconnected and PingGateway has ignored the event.

```
[vert.x-eventloop-thread-2] DEBUG ... @system - Connection
error. Ignored.
[CONTINUED]java.nio.channels.ClosedChannelException: null
```

This type of error occurs when a network component closes the connection. This can occur when:

- A load balancer or firewall terminates or times out connections
- Third-party network changes prevent successful connections

Increase logging to provide more information, as described in Manage logs.

Was this helpful?