Maintenance Guide

/ ForgeRock Identity Gateway 7

Latest update: 7.0.2

ForgeRock AS.
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Abstract

Guide to maintaining ForgeRock® Identity Gateway in production, for ForgeRock partners, technical consultants, system engineers, and reliability engineers.

Table of Contents

Preface

ForgeRock Identity Platform™ serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com.

About This Guide

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 IG services for their organization.

Chapter 1
Auditing Your Deployment

For information about the IG audit framework, see "*Audit Framework*" in the *Configuration Reference*. The following sections describe how to set up auditing for your deployment:

- "Recording Access Audit Events in CSV"
- "Recording Access Audit Events in JMS"
- "Recording Access Audit Events in JSON"
- "Recording Access Audit Events to Standard Output"
- "Recording Access Audit Events in Splunk"
- "Trusting Transaction IDs From Other Products"
- "Safelisting Audit Event Fields for the Logs"
- "Including or Excluding Audit Event Fields In Logs"

For more information about each event handler, see "*Audit Framework*" in the *Configuration Reference*.

Recording Access Audit Events in CSV

This section describes how to record access audit events in a CSV file. For information about the CSV audit event handler, see "CsvAuditEventHandler" in the *Configuration Reference*.

Important

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

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

Record Audit Events in a CSV File

Before you start, prepare IG and the sample application as described in Getting Started Guide.

1. Add the following route to IG:

Linux

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

Windows

```
%appdata%\OpenIG\config\routes\30-csv.json
```

```
{
  "name": "30-csv",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(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",
              "buffering": {
                "enabled": "true",
                "autoFlush": "true"
              },
              "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.

2. Go to http://openig.example.com:8080/home/csv-audit.

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

Recording Access Audit Events in JMS

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, see "JmsAuditEventHandler" in the *Configuration Reference*.

Record Audit Events With a JMS Audit Event Handler

Before you start, prepare IG as described in Getting Started Guide.

1. Add ActiveMQ client dependencies to IG:

   a. Download the following .jar files from :

      - `geronimo-j2ee-management_1.1_spec-1.0.1.jar`
      - `hawtbuf-1.11.jar`
      - `activemq-client-5.13.3.jar`

   b. Add the .jar files to the configuration:

      - For IG in standalone mode, create the directory `$HOME/.openig/extra`, where `$HOME/.openig` is the instance directory: and add .jar files to the directory.
      - For IG in web container mode, add .jar files to the web container classpath. For example, in Jetty use `/path/to/jetty/webapps/ROOT/WEB-INF/lib`.

2. Download and install the ActiveMQ message broker from http://activemq.apache.org/. For help, see the the ActiveMQ documentation on the same site.

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. Add the following route to IG:

   Linux

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

   Windows

   ```
   %appdata%\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://openam.example.com:61616",
                  "topic.audit" : "audit"
                },
                "topicName" : "audit",
                "connectionFactoryName" : "ConnectionFactory"
              }
            }
          }
        ],
        "config" : { }
      }
    }
  ],
  "auditService": "AuditService",
  "handler" : {
    "type" : "StaticResponseHandler",
    "config" : {
      "status" : 200,
      "headers" : {
        "Content-Type" : [ "text/plain" ]
      },
      "reason" : "found",
      "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. Access the route on http://openig.example.com:8080/activemq_event_handler.

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

Recording Access Audit Events in JSON

This section describes how to record access audit events with a JSON audit event handler. For information about configuring the JSON event handler, see "JsonAuditEventHandler" in the *Configuration Reference*.

Record Audit Events With a JSON Audit Event Handler

1. Add the following route to IG:

   Linux

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

   Windows

   ```
   %appdata%\OpenIG\config\routes\30-json.json
   ```

   ```
   {
     "name": "30-json",
     "baseURI": "http://app.example.com:8081",
     "condition": "${matches(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"
                 ],
                 "fileRetention": {
                   "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. Go to http://openig.example.com:8080/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://openig.example.com:8080/home/json-audit",
      "headers": {
        "accept": ["text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"],
        "host": ["openig.example.com:8080"],
        "user-agent": ["Mozilla/5.0 ... Firefox/66.0"]
      }
    }
  },
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "200",
    "elapsedTime": 212,
    "elapsedTimeUnits": "MILLISECONDS"
  }
}
```

Recording 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, see "JsonStdoutAuditEventHandler" in the *Configuration Reference*.

Record Audit Events to Standard Output

Before you start, prepare IG and the sample application as described in Getting Started Guide.

- Add the following route to IG:

  Linux

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

  Windows

  ```
  %appdata%\OpenIG\config\routes\30-jsonstdout.json
  ```

  ```
  {
    "name": "30-jsonstdout",
    "baseURI": "http://app.example.com:8081",
    "condition": "${matches(request.uri.path, '^/home/jsonstdout-audit')}",
    "heap": [
      {
        "name": "AuditService",
        "type": "AuditService",
        "config": {
          "eventHandlers": [
            {
              "class": "org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEventHandler",
              "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.

Test the Setup

- Go to http://openig.example.com:8080/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://openig.example.com:8080/home/jsonstdout-audit",
      "headers": {
        "accept": ["text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"],
        "host": ["openig.example.com:8080"],
        "user-agent": ["Mozilla/5.0 ... Firefox/66.0"]
      }
    }
  },
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "200",
    "elapsedTime": 10,
    "elapsedTimeUnits": "MILLISECONDS"
  },
  "source": "audit",
  "topic": "access",
  "level": "INFO"
}
```

Recording Access Audit Events in Splunk

This section describes how to set up a Splunk audit event handler to log IG access events to a Splunk system. For information about configuring the Splunk event handler, see "SplunkAuditEventHandler" in the *Configuration Reference*.

Set Up Splunk

This procedure assumes a Splunk instance running on the same host as IG. Adjust the instructions for your Splunk system.

1. Download Splunk from http://www.splunk.com, and install it with the default configuration. If you don't already have a Splunk account, create one.

   > **Tip**
   >
   > Splunk currently uses the following ports by default: `8000`, `8065`, `8088`, `8089`, and `8091`. Before you install Splunk, make sure that these ports are free. Alternatively, change the Splunk installation and IG route to use other ports.
   >
   > To find port numbers and other settings used by Splunk, select Server settings > General settings in the Splunk web interface.

2. With Splunk running, create a new source type and associate it with log data from IG:

   a. In the Splunk web interface, select Settings > Source Types > New Source Type.

   b. In the Create Source Type window, enter a name for the source type, for example, `openig`.

   c. In the Event Breaks panel of the same window, select Regex... and enter `^{` to indicate how the bulk messages are separated.

   d. Accept all of the other values as default and select Save.

3. Create an HTTP Event Collector to provide an authorization token so that IG can log events to Splunk:

   a. Select Settings > Data Inputs > HTTP Event Collector > New Token.

   b. Enter a Name for the token, for example, `openig`, leave the other fields with their default values, and select Next.

   c. In the Input Settings screen, select Select > Select Source Type > Custom, and then select the source type you created in the previous step.

   d. Select Review and then Submit.

      An authorization token is displayed. Make a note of the value or keep it on the screen so that you use it as the value of `authzToken` in `30-splunk.json`.

4. In the HTTP Event Collector window, check that the Global Settings are configured correctly. For example, make sure that all tokens are enabled and that SSL is not enabled.

The HTTP port number displayed in these global settings is used as the value of `port` in `30-splunk.json`.

5. Add the following route to IG, replacing the value of `authzToken` with the value returned earlier:

   Linux

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

   Windows

   ```
   %appdata%\OpenIG\config\routes\30-splunk.json
   ```

   ```
   {
     "name": "30-splunk",
     "baseURI": "http://app.example.com:8081",
     "condition": "${matches(request.uri.path, '^/home/splunk-audit')}",
     "heap": [
       {
         "name": "AuditService",
         "type": "AuditService",
         "config": {
           "eventHandlers": [
             {
               "class": "org.forgerock.audit.handlers.splunk.SplunkAuditEventHandler",
               "config": {
                 "name": "splunk",
                 "enabled": true,
                 "authzToken": "<splunk-authorization-token>",
                 "connection": {
                   "host": "localhost",
                   "port": 8088,
                   "useSSL": false
                 },
                 "topics": [
                   "access"
                 ],
                 "buffering": {
                   "maxSize": 10000,
                   "maxBatchedEvents": 500,
                   "writeInterval": "100 ms"
                 }
               }
             }
           ]
         }
       }
     ],
     "auditService": "AuditService",
     "handler": "ReverseProxyHandler"
   }
   ```

   The route calls an audit service configuration for publishing log messages to Splunk.

   For information about how to set up the route in Studio, see "Auditing in Structured Editor" in the *Studio User Guide*.

6. Test the setup:

    a. Go to http://openig.example.com:8080/home/splunk-audit.

    The home page of the sample application is displayed and events are logged in Splunk.

    b. Access the Splunk web interface on http://localhost:8000, and select Search & Reporting > Data Summary.

    Depending on how Splunk is configured, audit events are displayed on the web interface.

Trusting 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 ForgeRock Identity Platform.

To reduce the risk of malicious attacks, by default IG 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`. 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:

- When IG is running in standalone mode, add a 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}"
```

- When IG is running in web container mode, set a Java system property. For information, see the container documentation.

Safelisting 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, see "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` (Available in IG logs from IG 7.2)
- `/client`
- `/server`
- `/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`

Including or Excluding 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, see "Exclude Safelisted Audit Event Fields From Logs".

Exclude Safelisted Audit Event Fields From Logs

1. Set up recording for audit events, as described in "Recording Access Audit Events in JSON", and note the audit event fields in the log file `access.audit.json`.
2. Replace the route `30-json.json` with the following route:

```
{
  "name": "30-json-excludeif",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/json-audit-excludeif$')}",
  "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"
              ],
              "fileRetention": {
                "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.

3. Go to http://openig.example.com:8080/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"]
      }
    }
  }
}
```

4. 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.

Chapter 2
Monitoring Services

The following sections describe how to set up and maintain monitoring in your deployment, to ensure appropriate performance and service availability:

- "Accessing the Monitoring Endpoints"
- "Protecting the Monitoring Endpoints"

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

For information about IG monitoring endpoints and available metrics, see "*Monitoring*" in the *Configuration Reference*.

Monitoring at the Prometheus Scrape Endpoint

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

When IG is set up as described in Getting Started Guide, the Prometheus Scrape Endpoint is available at http://openig.example.com:8080/openig/metrics/prometheus.

By default, no special setup or configuration is required to access metrics at this endpoint. 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, see the Grafana website.

Monitor the Prometheus Scrape Endpoint

1. Add the following route to IG:

Linux

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

Windows

```
%appdata%\OpenIG\config\routes\myroute1.json
```

```
{
  "name": "myroute1",
  "handler": {
    "type": "StaticResponseHandler",
    "config": {
      "status": 200,
      "reason": "OK",
      "headers": {
        "Content-Type": [ "text/plain" ]
      },
      "entity": "Hello world, from myroute1!"
    }
  },
  "condition": "${matches(request.uri.path, '^/myroute1')}"
}
```

The route contains a StaticResponseHandler to display a simple message.

2. Access the route a few times, on http://openig.example.com:8080/myroute1.

3. Query the Prometheus Scrape Endpoint:

```
$ curl "http://openig.example.com:8080/openig/metrics/prometheus"
```

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

```
# HELP ig_router_deployed_routes Generated from Dropwizard metric import
 (metric=gateway._router.deployed-routes, type=gauge)
# TYPE ig_router_deployed_routes gauge
ig_router_deployed_routes{fully_qualified_name="gateway._router",heap="gateway",name="_router",} 1.0
# HELP ig_route_request_active Generated from Dropwizard metric import
 (metric=gateway._router.route.default.request.active, type=gauge)
# TYPE ig_route_request_active gauge
ig_route_request_active{name="default",route="default",router="gateway._router",} 0.0
# HELP ig_route_request_active Generated from Dropwizard metric import
 (metric=gateway._router.route.myroute1.request.active, type=gauge)
# TYPE ig_route_request_active gauge
ig_route_request_active{name="myroute1",route="myroute1",router="gateway._router",} 0.0
# HELP ig_route_request_total Generated from Dropwizard metric import
 (metric=gateway._router.route.default.request, type=counter)
# TYPE ig_route_request_total counter
ig_route_request_total{name="default",route="default",router="gateway._router",} 0.0
# HELP ig_route_response_error Generated from Dropwizard metric import
 (metric=gateway._router.route.default.response.error, type=counter)
# TYPE ig_route_response_error counter
ig_route_response_error{name="default",route="default",router="gateway._router",} 0.0
# HELP ig_route_response_null Generated from Dropwizard metric import
 (metric=gateway._router.route.default.response.null, type=counter)
# TYPE ig_route_response_null counter
ig_route_response_null{name="default",route="default",router="gateway._router",} 0.0
# HELP ig_route_response_status_total Generated from Dropwizard metric import
 (metric=gateway._router.route.default.response.status.client_error, type=counter)
# TYPE ig_route_response_status_total counter
ig_route_response_status_total{family="client_error",name="default",route="default",router="gateway._router",}
 0.0
...
```

Monitoring the Common REST Monitoring Endpoint

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

When IG is set up as described in Getting Started Guide, the Common REST Monitoring Endpoint is available at http://openig.example.com:8080/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.

Monitor the Common REST Monitoring Endpoint

Before you start, prepare IG as described in Getting Started Guide.

1. Set up IG 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 "http://openig.example.com:8080/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
}
```

3. Change the query to access metrics only for `myroute1`: http://openig.example.com:8080/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.

Protecting the 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.

Protect the Monitoring Endpoints

1. Add the following script to the IG configuration as `$HOME/.openig/scripts/groovy/BasicAuthResourceServerFilter.groovy` (on Windows, `%appdata%\OpenIG\scripts\groovy\BasicAuthResourceServerFilter.groovy`):

```
/**
 * This script is a simple implementation of HTTP Basic 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
 *  - password: the expected password
 */

import static org.forgerock.util.promise.Promises.newResultPromise;

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

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

String expectedAuthorization = "Basic " + Base64.encode((username + ":" +
 password).getBytes(Charset.defaultCharset()))
if (!expectedAuthorization.equals(authorizationHeader)) {
    return newResultPromise(new Response(Status.FORBIDDEN));
}
// Credentials are as expected, let's continue
return next.handle(context, request);
```

   The script is a simple implementation of the HTTP Basic Authentication mechanism.

   For information about scripting filters and handlers, see "*Extending IG*" in the *Gateway Guide*.

2. Add the following route to IG:

   Linux

   ```
   $HOME/.openig/config/admin.json
   ```

   Windows

   ```
   %appdata%\OpenIG\config\admin.json
   ```

   Web container mode

   ```
   {
     "heap": [{
       "name": "ClientHandler",
       "type": "ClientHandler"
     }, {
       "name": "MetricsProtectionFilter",
       "type": "ScriptableFilter",
       "config": {
         "type": "application/x-groovy",
         "file": "BasicAuthResourceServerFilter.groovy",
         "args": {
           "realm": "/",
           "username": "metric",
           "password": "password"
         }
       }
     }],
     "prefix": "openig"
   }
   ```

   Standalone mode

```
{
  "prefix": "openig",
  "connectors": [
    { "port": 8080 }
  ],
  "heap": [
    {
      "name": "ClientHandler",
      "type": "ClientHandler"
    },
    {
      "name": "MetricsProtectionFilter",
      "type": "ScriptableFilter",
      "config": {
        "type": "application/x-groovy",
        "file": "BasicAuthResourceServerFilter.groovy",
        "args": {
          "realm": "/",
          "username": "metric",
          "password": "password"
        }
      }
    }
  ]
}
```

3. Restart IG to reload the configuration.

Chapter 3
Managing Logs

Log messages in IG and third-party dependencies are recorded using the Logback implementation of the Simple Logging Facade for Java (SLF4J) API. For a full description of the options for logging, see the Logback website.

The following sections describe how to collect log messages, and how to adapt and maintain logs:

Default Logging Behavior

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

- When IG starts, log messages for IG 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 route is accessed, log messages for requests and responses passing through the route are written to a log file in `$HOME/.openig/logs`, and named by the route name or filename, where `$HOME/.openig` is the instance directory.

  For more information, see "Capturing Log Messages for Routes" and "CaptureDecorator" in the *Configuration Reference*.
- 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 IG is defined by the reference `logback.xml` delivered with IG. 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"?>
<!--
  Copyright 2016-2019 ForgeRock AS. All Rights Reserved

  Use of this code requires a commercial software license with ForgeRock AS.
  or with one of its affiliates. All use shall be exclusively subject
  to such license between the licensee and ForgeRock AS.
-->

<configuration>

  <!--
    Avoid logs to be flooded in case of repetitive messages.

    Configuration properties:
     * AllowedRepetitions (int): threshold above which same messages won't be logged anymore
     * CacheSize (int): Remove eldest entry when hitting max size
   -->
  <!--<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">
    <encoder>
      <pattern>%nopex[%thread] %highlight(%-5level) %boldWhite(%logger{35}) @%mdc{routeId:-system} -
 %message%n%highlight(%rootException{short})</pattern>
    </encoder>
  </appender>

  <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>%date{"yyyy-MM-dd'T'HH:mm:ss,SSSXXX", UTC} | %-5level | %thread | %logger{20} | @
%mdc{routeId:-system} | %message%n%xException</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 IG. The custom Logback file overrides the default configuration.

To take into account edits to `logback.xml`, stop and restart IG, 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 ...
```

Changing Log Levels

The Logback implementation in IG supports the following log levels: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `ALL`, and `OFF`. Log levels are case-insensitive in `logback.xml`.

Changing 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}">
```

When IG is running in standalone mode, change the global log level as follows:

- To persist the log level for all future IG 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`:

    ```
    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 IG instances launched from the same shell, add an environment variable in the shell before you start IG:

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

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

  ```
  $ ROOT_LOG_LEVEL=DEBUG /path/to/identity-gateway/bin/start.sh $HOME/.openig
  ```

When IG is running in web container mode, add an environment variable on the command line when you start the web container:

Linux

```
$ export ROOT_LOG_LEVEL=DEBUG
```

Windows

```
C:> set ROOT_LOG_LEVEL=DEBUG
```

Changing 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 areas:

- Expression resolution:

  ```
  org.forgerock.openig.el
  ```

  ```
  org.forgerock.openig.resolver
  ```

- Session management with JwtSession:

  ```
  org.forgerock.openig.jwt
  ```

- OAuth 2.0 and OpenID Connect and token resolution and validation:

  ```
  org.forgerock.openig.filter.oauth2
  ```

- Interaction with AM policies, SSO, CDSSO, and user profiles:

  ```
  org.forgerock.openig.openam
  ```

  ```
  org.forgerock.openig.tools
  ```

- SAML support:

  ```
  org.forgerock.openig.handler.saml
  ```

- UMA support:

  ```
  org.forgerock.openig.uma
  ```

- Web socket 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, where *filter_name* refers to the top-level `name` of the "AllowOnlyFilter" in the *Configuration Reference*.

  ```
  org.forgerock.openig.filter.allow.AllowOnlyFilter.filter_name
  ```

Changing the Format of Log Messages

To change the format of log messages, edit `logback.xml` to change the pattern of the log messages in the `encoder` part of the SIFT appender.

The following lines add the date to log messages:

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

Logging 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
  ```

Logging 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.baseURI" 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
```

Switching Off 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"/>
```

Capturing Log Messages for Routes

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

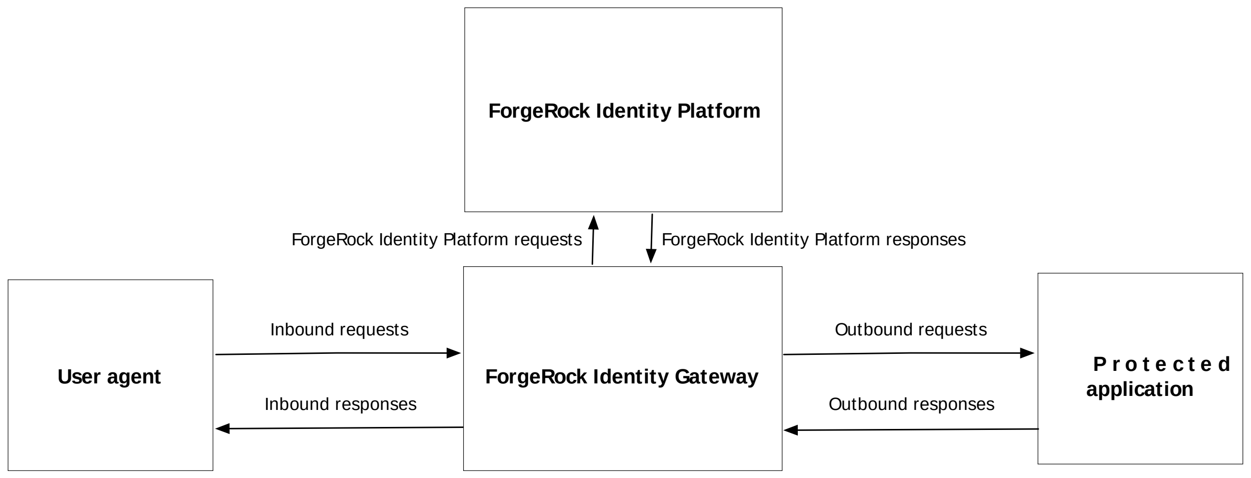
Important

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, see "CaptureDecorator" in the *Configuration Reference*.

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:

Capturing Log Messages for Routes


ForgeRock Identity Platform
ForgeRock Identity Platform requests
ForgeRock Identity Platform responses
User agent
Inbound requests
Inbound responses
ForgeRock Identity Gateway
Outbound requests
Outbound responses
Protected application

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 ForgeRock Identity Platform.
3. Select Deploy to push the route to the IG configuration.

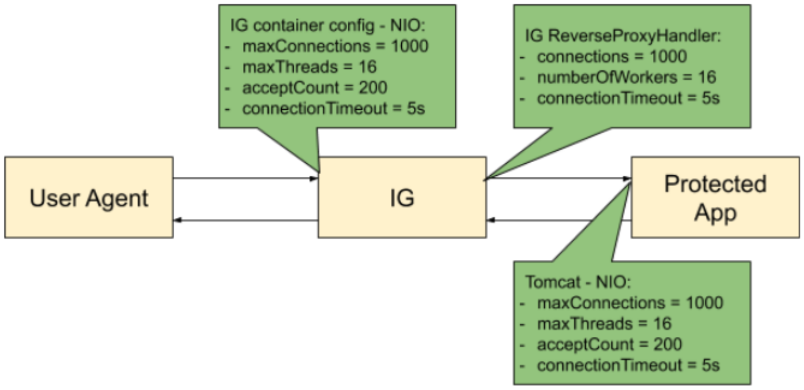
   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.
4. Access the route, and then check `$HOME/.openig/logs` for a log file named by the route, where `$HOME/.openig` is the instance directory. The log file should contain the messages defined by your capture configuration.

Chapter 4
Tuning Performance

Tune deployments in the following steps:

1. Consider the issues that impact the performance of a deployment. See "Defining Performance Requirements and Constraints".
2. Tune and test the downstream servers and applications:
    - Tune the downstream web container and JVM to achieve performance targets.
    - Test downstream servers and applications in a pre-production environment, under the expected load, and with common use cases.
    - Make sure that the configuration of the downstream web container can form the basis for IG and its container.
3. Tune IG and its web container:
    - Optimize IG performance, throughput, and response times. See "Tuning IG".
    - Configure IG connections to downstream services and protected applications. See "Tuning the ClientHandler/ReverseProxyHandler".
    - Configure connections in the IG web container. See "Tuning IG's Tomcat Container".
    - Configure the IG JVM to support the required throughput. See "Tuning IG's JVM".
4. Increase hardware resources as required, and then re-tune the deployment.

The following figure shows an example configuration for IG, its container, and the container for the protected app:



Defining Performance 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 IG is inserted as a proxy in front of a service or application.
- The constraint that downstream limitations and response times places on IG and its container.

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 IG 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, see "*Monitoring*" in the *Configuration Reference*. For examples of how to set up monitoring in IG, see "*Monitoring 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 benchmark tests on IG, see *ForgeOps*' Performance Benchmarks. Benchmark test results are given for throughput and response times in an AM password grant flow, and for IG resource server flows with and without cache.

Tuning IG

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 IG 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, see "*Managing Logs*".

Buffering Message Content

IG creates a TemporaryStorage object to buffer content during processing. For information about this object and its default values, see "TemporaryStorage" in the *Configuration Reference*.

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.

Cache

When caches are enabled, IG 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.

During service downtime, the cache is not updated, and important notifications can be missed, such as for the revocation of tokens or the update of policies, and IG can continue to use outdated tokens or policies.

When caches are disabled, IG 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.

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

IG provides the following caches:

Session cache

After a user authenticates with AM, this cache stores information about the session. IG can reuse the information without asking AM to verify the session token (SSO token or CDSSO token) for each request.

If WebSocket notifications are enabled, the cache evicts entries based on session notifications from AM, making the cache content more accurate (trustable).

By default, the session information is not cached. To increase performance, consider enabling and configuring the cache. For more information, see `sessionCache` in "AmService" in the *Configuration Reference*.

Policy cache

After an AM policy decision, this cache stores the decision. IG can reuse the policy decision without repeatedly asking AM for a new policy decision.

If WebSocket notifications are enabled, the cache evicts entries based on policy notifications from AM, making the cache content more accurate (trustable).

By default, policy decisions are not cached. To increase performance, consider enabling and configuring the cache. For more information, see "PolicyEnforcementFilter" in the *Configuration Reference*.

User profile cache

When the UserProfileFilter retrieves user information, it caches it. IG can reuse the cached data without repeatedly querying AM to retrieve it.

By default, profile attributes are not cached. To increase performance, consider enabling and configuring the cache. For more information, see "UserProfileFilter" in the *Configuration Reference*.

Access token cache

After a user presents an access_token to the OAuth2ResourceServerFilter, this cache stores the token. IG can reuse the token information without repeatedly asking the authorization server to verify the access_token for each request.

By default, access_tokens are not cached. To increase performance by caching access_tokens, consider configuring a cache in one of the following ways:

- Configure a CacheAccessTokenResolver for a cache based on Caffeine. For more information, see "CacheAccessTokenResolver" in the *Configuration Reference*.
- Configure the `cache` property of OAuth2ResourceServerFilter. For more information, see "OAuth2ResourceServerFilter" in the *Configuration Reference*

Open ID Connect user information cache

When a downstream filter or handler needs user information from an OpenID Connect provider, IG fetches it lazily. By default, IG caches the information for 10 minutes to prevent repeated calls over a short time.

For more information, see `cacheExpiration` in "OAuth2ClientFilter" in the *Configuration Reference*.

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. Consider your requirements for performance and security when you configure the timeout properties for each cache.

WebSocket Notifications

By default, IG 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. IG can use WebSocket notifications to evict entries from the session cache.

  For an example of setting up session cache eviction, see "Using WebSocket Notifications to Evict the Session Info Cache" in the *Gateway Guide*.

- When AM creates, deletes, or changes a policy decision. IG can use WebSocket notifications to evict entries from the policy cache.

  For an example of setting up cache eviction, see "Using WebSocket Notifications to Evict the Policy Cache" in the *Gateway Guide*.

To disable WebSocket notifications, or change any of the parameters, configure the `notifications` property in AmService. For information, see "AmService" in the *Configuration Reference*.

If the WebSocket connection is lost for a time, notifications that occur during that time are lost. If a session ends or a policy decision changes while the WebSocket connection is lost, IG is not notified, and can continue to operate on outdated data.

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

Tuning 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.
- Number of IG worker threads allocated to service inbound requests, and manage propagation to the downstream service or application.
- The connection timeout, or maximum time to connect to a server-side socket, before timing out and abandoning the connection attempt.
- The socket timeout, or the maximum time a request is expected to take before a response is received, after which the request is deemed to have failed.

ClientHandler/ReverseProxyHandler Tuning in Web Container Mode

Configure IG in conjunction with the Tomcat container, as follows:

- For BIO Connector (Tomcat 3.x to 8.x), configure `maxThreads` in Tomcat to be close to the number of configured Tomcat connections.

  Because IG uses an asynchronous threading model, the `numberOfWorkers` in ClientHandler/ReverseProxyHandler can be much lower. The asynchronous threads are freed up immediately after the request is propagated, and can service another blocking Tomcat request thread.

  To take advantage of IG's asynchronous thread model, configure Tomcat to use a non-blocking, NIO or NIO2 connector, instead of a BIO connector.

- For NIO connectors, align `numberOfWorkers` in IG with `maxThreads` in Tomcat.

  Because NIO connectors use an asynchronous threading model, the `maxThreads` in Tomcat can be much lower than for a BIO connector.

To identify the throughput plateau, test in a pre-production performance environment, with realistic use cases. Increment `numberOfWorkers` from its default value of one thread per JVM core, up to a large maximum value based on the number of concurrent connections.

ClientHandler/ReverseProxyHandler Tuning in Standalone Mode

Configure IG in conjunction with IG's first-class Vert.x configuration, and the `vertx` property of `admin.json`. For more information, see "AdminHttpApplication (`admin.json`)" in the *Configuration Reference*.

Vert.x Options for Tuning

Object	**Vert.x Option**	**Description**
IG (first-class)	`gatewayUnits`	The number of Vert.x Verticle instances to deploy. Each instance operates on the same port on its own event-loop thread. This setting effectively determines the number of cores that IG operates across, and therefore, the number of available threads. Default: The number of cores.
root.vertx	`eventLoopPoolSize`	The number of available event-loop threads to be supplied to instances. Specify a value greater than that for `gatewayUnits`. Default: 20

Object	Vert.x Option	Description
root.connectors.<connector>.vertx	`acceptBacklog`	The maximum number of connections to queue before refusing requests.
	`sendBufferSize`	TCP connection send buffer size. Set this property according to the available RAM and required number of concurrent connections.
	`receiveBufferSize`	TCP receive buffer size. Set this property according to the available RAM and required number of concurrent connections.

Vert.x Options for Troubleshooting Performance

Object	Vert.x Option	Description
root.vertx	`blockedThreadCheckInterval` and `blockedThreadCheckIntervalUnit`	Interval at which Vert.x checks for blocked threads and logs a warning. Default: One second.
	`maxEventLoopExecuteTime` and `maxEventLoopExecuteTimeUnit`	Maximum time executing before Vert.x logs a warning. Default: Two seconds.
	`warningExceptionTime` and `warningExceptionTimeUnit`	Threshold at which warning logs are accompanied by a stack trace to identify causes. Default: Five seconds.
	`logActivity`	Log network activity.

Tuning IG's Tomcat Container

Configure the Tomcat container in conjunction with IG, as described in "Tuning the ClientHandler/ReverseProxyHandler".

To take advantage of IG's asynchronous thread model, configure Tomcat to use a non-blocking, NIO or NIO2 connector. Consider configuring the following connector attributes:

- `maxConnections`
- `connectionTimeout`
- `soTimeout`
- `acceptCount`

- `executor`
- `maxThreads`
- `minSpareThreads`

For more information, see Apache Tomcat 9 Configuration Reference and Apache Tomcat 8 Configuration Reference.

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

IG 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, see 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. For information about options to suite protected web applications, see Garbage-First Garbage Collector and Garbage-First Garbage Collector Tuning

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

Chapter 5
Securing and Hardening

The following sections describe options for securing and hardening your deployment:

- "General Security Considerations"
- "Switching From Development Mode to Production Mode"
- "Rotating Keys"

General Security Considerations

Consider the following settings in IG to secure your deployment:

Switch to production mode.

Prevent changes to your configuration by setting the run mode to `production`. For information, see "Switching From Development Mode to Production Mode".

Protect endpoints.

Restrict access to your monitoring data by protecting the Prometheus Scrape Endpoint and Common REST Monitoring Endpoint. For information, see "Protecting the Monitoring Endpoints".

Exclude sensitive data from logs.

- Prevent logging of sensitive data for audit events by excluding fields from the audit logs. For information, see "Including or Excluding Audit Event Fields In Logs".
- When using a CaptureDecorator, mask captured header and attribute values to avoid disclosing information, such as token values or passwords. For information, see "CaptureDecorator" in the *Configuration Reference*.

Limit the amount of information available to attackers during reconnaissance, the initial phase of an attack sequence.

- Reduce the global log level from `TRACE` or `DEBUG` to `INFO`. For information, see "Changing Log Levels".
- Avoid using words that help to identify IG in error messages, such as those produced by the entity in a StaticResponseHandler. For information, see "StaticResponseHandler" in the *Configuration Reference*.

Secure responses from cross-site scripting.

- Sanitize any external input, such as the request, before incorporating it in the response.
- Specify `Content-Type` in the `headers` property of StaticResponseHandler when an entity is used. (Required by default, from IG 7.)
- Set the response header `X-Content-Type-Options: nosniff` to prevent the user agent from interpreting the response entity as a different content type. (Set by default, from IG 7.)
- Set a restrictive value in the `Cache-Control` response header. For example, setting `Cache-Control: private` indicates that all or part of the response message is intended for a single user and MUST NOT be cached by a shared cache.

Secure sessions.

Protect network traffic by using HTTPS where possible, and secure communications during stateless sessions by signing and encrypting JWTs. For information, see "JwtSession" in the *Configuration Reference*.

Secure cookies.

- Store a minimum amount of sensitive data in cookies, and only when necessary.
- Change the default name of cookies to prevent them from being easily associated with an application.
- Where possible, configure cookie properties to restrict how and where they can be used. For more information, see `cookie` in "JwtSession" in the *Configuration Reference*, and `authCookie` in "CrossDomainSingleSignOnFilter" in the *Configuration Reference*.

Protect against weak DH keys.

In JVM, the default ephemeral Diffie-Hellman (DH) key size is 1024 bits. To support stronger ephemeral DH keys, and protect against weak keys, installations in Tomcat 8.5.37 and later versions use the Tomcat default DH key size of 2048-bit.

For IG installations in Jetty or JBoss, consider setting the following system property to increase the DH key size: `jdk.tls.ephemeralDHKeySize=2048`.

Consider the following general points to secure your deployment:

Keep up to date on patches.

Security vulnerabilities are the reason why you should keep your operating systems, web and application servers, and any other application in your environment up to date. Knowledge of vulnerabilities spreads fast across malicious users, who would not hesitate in trying to exploit them.

ForgeRock maintains a list of security advisories that you should follow. You should also follow similar lists from all of your vendors.

Keep up to date on cryptographic methods and algorithms.

Different algorithms and methods are discovered and tested over time, and communities of experts decide which are the most secure for different uses. Do not use outdated algorithms for generating your keys.

Turn off unnecessary features.

The more features you have turned on, the more features you need to secure, patch, and audit. If something is not being used, disable it or uninstall it.

Limit access to the servers hosting IG.

Make sure only authorized people can access your servers and applications through the appropriate network, using the appropriate ports, and presenting strong-enough credentials.

Ensure users connect to systems through the latest versions of TLS, and audit system access periodically.

Enforce security.

Do not expect your users to follow security practices on their own; enforce security when possible by requiring secure connections, password resets, and strong authentication methods.

Audit access and changes.

Audit logs record all events that have happened. Some applications store them with their engine logs, others use specific files or send the information to a different server for archiving. Operating systems have audit logs as well, to detect unauthorized login attempts and changes to the software.

IG has its own audit logging service that adheres to the log structure common across the ForgeRock Identity Platform. For information, see "*Auditing Your Deployment*".

Switching From Development Mode to Production Mode

IG operates in development mode and production mode, as defined in "Development Mode and Production Mode" in the *Gateway Guide*.

After installation, IG is by default in production mode. While you evaluate IG or develop routes, it can be helpful to switch to development mode as described in "Switching from Production Mode to Development Mode" in the *Getting Started Guide*. However, after deployment it is essential to switch back to production mode to prevent unwanted changes to the configuration.

Switch From Development Mode to Production Mode

1. In `$HOME/.openig/config/admin.json` (on Windows, `%appdata%\OpenIG\config`), change the value of `mode` from `DEVELOPMENT` to `PRODUCTION`:

   ```
   {
     "mode": "PRODUCTION"
   }
   ```

   The file changes the operating mode from development mode to production mode. For more information about the `admin.json` file, see "AdminHttpApplication (`admin.json`)" in the *Configuration Reference*.

   The value set in `admin.json` overrides any value set by the configuration token `ig.run.mode` when it is used in an environment variable or system property. For information about `ig.run.mode`, see "Configuration Tokens" in the *Configuration Reference*.

2. (Optional) Prevent routes from being reloaded after startup:

   - To prevent all routes in the configuration from being reloaded, add a `config.json` as described in Getting Started Guide, and configure the `scanInterval` of the main router.
   - To prevent individual routes from being reloaded, configure the `scanInterval` of the routers in those routes.

   ```
   {
     "type": "Router",
     "config": {
       "scanInterval": "disabled"
     }
   }
   ```

   For more information, see "Router" in the *Configuration Reference*.

3. Restart IG.

   When IG starts up, the route endpoints are not displayed in the logs, and are not available. You can't access Studio on http://openig.example.com:8080/openig/studio.

Rotating 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 some use cases from the Gateway Guide:

- "About Key Rotation"
- "Rotating Keys For Stateless Access_Tokens Signed With a KeyStoreSecretStore"
- "Rotating Keys In a Shared JWT Session"

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 Rotate Keys

Regular key rotation is a security best-practice, 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.

Key Rotation Steps

The following steps outline the process for key rotation and revocation for keys managed by a KeyStoreSecretStore or HsmSecretStore:

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.

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 before you provision the new keys. The system is unusable until new keys are provisioned.

Key Rotation for JwkSetSecretStore

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

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

Rotating Keys For Stateless Access_Tokens Signed With a KeyStoreSecretStore

This example extends the example in "Validating Signed Access_Tokens With the StatelessAccessTokenResolver and KeyStoreSecretStore" in the *Gateway Guide* 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 "Validating Signed Access_Tokens With the StatelessAccessTokenResolver and KeyStoreSecretStore" in the *Gateway Guide*.

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=openig.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 PKCS12 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 that it contains the new and old keys:

```
$ keytool -list \
-keystore "keystore_directory/keystore-new.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 IG:

a. Import the public certificate to the IG 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
```

b. List the keys in the IG 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
```

c. 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 IG to reload the route.

IG 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://openam.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 http://openig.example.com:8080/rs-stateless-signed-ksss --header "Authorization: Bearer
    ${mytoken}"
   ...
   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 -signature-key
   ```

   b. List the keys in the AM keystore to make sure that `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 IG keystore:

   a. Delete the key from the keystore:

   ```
   $ keytool -delete -verification-key
   ```

   b. List the keys in the IG 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 IG, delete the mapping for `verification-key` from the route `rs-stateless-signed-ksss.json`. If the Router `scanInterval` is disabled, restart IG to reload the route.

Rotating Keys In a Shared JWT Session

This section builds on the example in "Sharing JWT Session Between Multiple Instances of IG" in the *Gateway Guide* to rotate a key used in a shared JWT session.

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

Rotate Keys in a Shared JWT Session

Before you start, set up the example in "Set Up Shared Secrets for Multiple Instances of IG" in the *Gateway Guide*, where three instances of IG 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://openig.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://openig.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://openig.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 that 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 IG instance, as follows:

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

   If the Router `scanInterval` is disabled, restart IG 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 that 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 that 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 that 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 that instances 2 and 3 can still access the session information.

Chapter 6

Troubleshooting

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to help you set up and maintain your deployments. For information about getting support, see "*Getting Support*" in the *Release Notes*.

When you are 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 see?
- 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. If IG is running in development mode, and set up as described in Getting Started Guide, access the information at http://openig.example.com:8080/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

Troubleshooting

Displaying resources	+ *Requests redirected to Access Management instead of to the resource*
	
	By default, ForgeRock Access Management 5 and later writes cookies to the fully qualified domain name of the server; for example, `openam.example.com`. Therefore, a host-based cookie, rather than a domain-based cookie, is set.
	
	Consequently, after authentication through Access Management, requests can be redirected to Access Management instead of to the resource.
	
	To resolve this issue, add a cookie domain to the Access Management configuration. For example, in the Access Management console, go to Configure > Global Services > Platform, and add the domain `example.com`.
	
	+ *Sample application not displayed correctly*
	
	When the sample application is used with IG in the documentation examples, the sample application must serve static resources, such as the .css. Add the following route to the IG configuration, as:
	
	Linux
	`$HOME/.openig/config/routes/static-resources.json`
	
	Windows
	`%appdata%\OpenIG\config\routes\static-resources.json`
	
	`{`
	`  "name" : "sampleapp_resources",`
	`  "baseURI" : "http://app.example.com:8081",`
	`  "condition": "${matches(request.uri.path,'^/css')}",`
	`  "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,
    "reason": "Forbidden",
    "headers": {
      "Content-Type": [ "text/html" ]
    },
    "entity": "<html><body><p>User does not have permission</p></body></
html>"
  }
}
```

Using routes

+ *No handler to dispatch to*

If you get the message `no handler to dispatch to`, consider the following points:

- Make sure that your routes include a `condition` configuration to match the request. For more information, see "Setting Route Conditions" in the *Gateway Guide*.
- If requests might not match any condition, consider adding a default route to provide a default handler when no condition is met. For more information, see "Adding a Default Route" in the *Getting Started Guide*.

+ *Object not found in heap*

```
       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)
```

You have specified `"handler": "Router2"` in `config.json` or in the route, but no handler configuration object named Router2 exists. 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, IG does not load the route. Instead, a description of the error appears in the log:

```
       16:09:50 | ERROR | openig.example.com-startStop-1 |
o.f.o.h.r.RouterHandler |
       The file '/Users/me/.openig/config/routes/zz-default.json' is not
a valid route configuration.
```

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

+ *Route not used*

IG 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, IG logs errors, but it keeps the previous, correct configuration, and continues to use the old route.

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

Of course, if you restart IG with an invalid route configuration, then IG 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 see an error, `No handler to dispatch to`.

+ *Skip routes*

IG 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 IG 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 IG to reload the configuration. When you have configured the environment, change the file extension back to `.json`.

Using 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 that it contains a main router named `_router`.

	For information about creating routes in Studio, see the Studio User Guide.
Understanding timeout errors	+ *Timeout downloading large files* (Not supported for IG in standalone mode.) If `SocketTimeoutException` errors occur in the logs when you try to download large files, in your ReverseProxyHandler or ClientHandler, increase `soTimeout` and set `asyncBehavior` to `streaming`. + *Log is flushed with timeout exception warnings on sending a request* **Problem**: After a request is sent to IG, IG seems to hang. An HTTP 502 Bad Gateway error is produced, and the IG log is flushed with SocketTimeoutException warnings. **Possible cause**: The `baseURI` configuration is missing or causes the request to return to IG, so IG can't produce a response to the request. **Possible solution**: Configure the `baseURI` to use a different host and port to IG.

Other problems	+ *Incorrect values in the flat files*
	
	Make sure that the user running IG 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*
	
	`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`
	
	This error is typically encountered when using an `AssignmentFilter` as described in "AssignmentFilter" in the *Configuration Reference* and setting a string value for one of the headers. 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.