Tuning IG and the ClientHandler/ ReverseProxyHandler

IG ClientHandler and ReverseProxyHandler Configuration

The IG `ClientHandler` and `ReverseProxyHandler` is configured to communicate as the client to the downstream Protected application. For the purposes of conciseness, the term `ClientHandler` will be used to represent both implementations.

Specific configuration that manages this communication are:

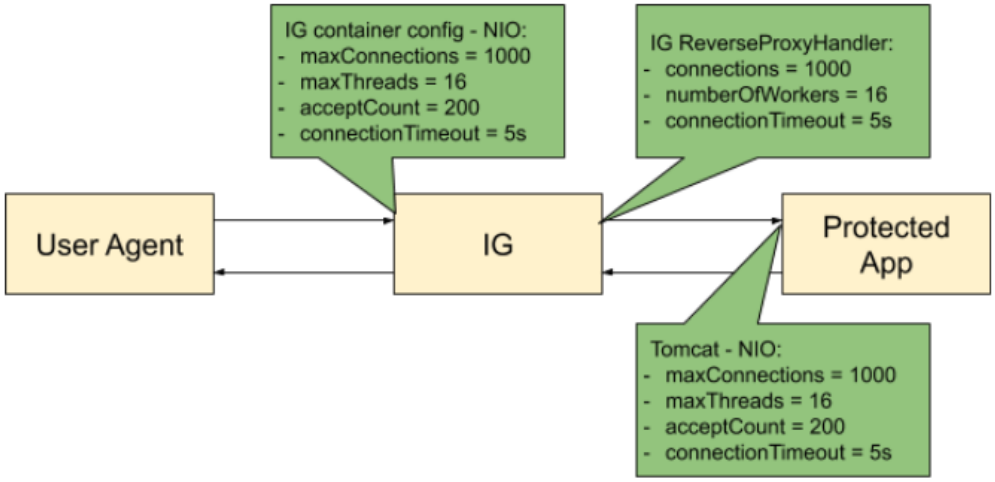
Option	Description
`connections`	The number of available connections to the downstream remote application
`numberOfWorkers`	This is the number of IG worker threads allocated to service inbound requests and manage propagation to the downstream application. Of note, IG has an asynchronous threading-model, so a worker thread is not consumed blocking for a response from the downstream server. By default, this value is set to the number of available cores.
`connectTimeout`	The connection timeout, or maximum time to connect to a server-side socket, before timing out and abandoning the connection attempt.
`soTimeout`	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.

Configuration Strategy

The IG container (Tomcat, Jetty, or Vert.x config) and IG-specific configuration should be done with regard to:

1. the performance goals, together with the capabilities and limitations of the downstream system:
2. expectation of some increase in response time with IG inserted as a proxy in front of the protected application, due to the extra network hop and processing required.
3. IG and its container being constrained by the limitations of the downstream server and the response times of the protected application. This includes the downstream web container configuration, its JVM configuration and tuning, resource types (e.g. compiled resources), etc.

With that in mind, the configuration of IG as a proxy should be conducted as follows:



1. Start with the configuration of the downstream server and protected application:
   a. Ensure that the web container and JVM are tuned and able to achieve performance targets.
   b. Test and confirm in a pre-production environment under expected load and with common use-cases.
   c. Ensure that the web container configuration forms the basis of configuring IG and its web container.
2. Configure IG and its web container, based on the limitations of the downstream server and protected application:
   a. Configure the IG `ClientHandler` based on the downstream server configuration (see below).
   b. Configure the IG web container (e.g. Tomcat) to correspond with the downstream server configuration:
      i. At this stage, IG and its web container should replicate the number of connections and timeouts of the downstream application.
      ii. Test and tune the IG `ClientHandler numberOfThreads` and IG web container threads `maxThreads` to determine the optimum throughput.
   c. Tune the IG web container JVM to support the desired throughput:
      i. Ensure there is sufficient memory to accommodate peak-load for the required connections. See Tuning the JVM].
      ii. Ensure IG and its container timeouts support latency in the protected application.
      iii. This phase should involve an incremental optimisation exercise to settle on the best performing memory and garbage collection settings.

3. Vertical scaling:
    a. Look to increase hardware resources, as required.

Configuring the IG (Tomcat) Web Container Uniformly with the ClientHandler

The relationship between the Tomcat container and IG webapp is that Tomcat's `maxThreads` is the number of Tomcat HTTP request threads. An IG worker thread - `numberOfWorkers` - will pick up from a Tomcat request thread to propagate requests for processing downstream, via the `ClientHandler`.

The Tomcat version and the selection of Tomcat HTTP Connector is very important with regards to configuring IG. Notably, IG should be configured in conjunction with the Tomcat IG container configuration. Notably:

- If using a BIO Connector (Tomcat 3.x to 8.x):
    - the Tomcat maxThreads should be aligned to be close to the number of Tomcat configured connections. IG can be configured a lot lower (using an async threading model). The async IG threads are freed up immediately after the request is propagated and can service another blocking Tomcat request thread.
    - Assumptions should be ratified in a pre-production performance test environment using real-life use cases.
- If using a NIO Connector:
    - Tomcat `maxThreads` config can be a lot lower than it would be using a BIO Connector. The NIO Connector also uses an async threading model, freeing up request threads once the request is handed over to the IG worker threads.
    - Therefore, the config of IG worker threads - `numberOfWorkers` should be closely aligned with your Tomcat request threads - `maxThreads`.
    - It is still necessary to test the IG worker config throughput/ errors incrementally in this deployment to identify optimum throughput.

Notes on Threading

The IG `ClientHandler` `numberOfThreads` option defaults to the number of available cores. This is a sensible start point given the asynchronous threading model used in IG. That is, in theory, as the IG uses an asynchronous threading model, one thread per core should maximise use of available CPU time (e.g. in-between I/O operations). However, in reality, some requests do block due to IG dependencies. For example the `ResourceHandler` serves static resources from disk.

Additionally, performance testing has indicated that some improvement can be seen by increasing `numberOfThreads`. It is therefore advisable to test and optimise in a pre-production environment under load with sensible, realistic use cases to understand if increasing this value leads to better throughput. Test incrementally with binary increases between number of cores and some large maximum, based on the number of concurrent connections. The aim of the exercise is to identify the plateau in throughput.

Configuring IG Standalone and the ClientHandler

IG standalone exists as a server in its own right, rather than being a web container hosted web application. It is implemented on the Vert.x application framework.

IG standalone is configured through its admin.json file, as described in the Gateway Guide (version 7.0+). There are a number of first-class options available, but also, the full set of Vert.x specific options remain available (except where expressly disallowed due to first-class options taking precedence). These can be configured for all the server overall and per connector:

- Server config is configured in the root `vertx` object:
    - This also contains several warning log configuration options: blocked thread, max time in thread.
    - See Vert.x VertxOptions API doc for details.
- Connector config is configured in the `connectors` config - in a contained connector's `vertx` config:
    - This contains various max levels pertaining to header size, data/ chunk size, websocket frame and message size
    - It also contains options supporting message compression
    - See Vert.x HttpServerOptions API doc for details.

In the example below, specific WebSocket configuration is provided to the IG server for the overall server (root `vertx` object) and for a connector `connectors.<connector>.vertx` object:

Example IG standalone admin.json vertx config

```
{
  "vertx": {
    "doc (ignored)": "overall server configuration - see https://vertx.io/docs/apidocs/io/vertx/core/VertxOptions.
html",
    ...
  },
  "connectors": [
    {
      "port": 8080
      "vertx" : {
        "doc (ignored)": "connector configuration - see https://vertx.io/docs/apidocs/io/vertx/core/http
/HttpServerOptions.html",
        ...
      }
    }
  ]
  ...
}
```

As described in Configuration Strategy, and as with IG on Tomcat, there is a relationship between the Vert.x HTTP server configuration and the IG `ClientHandler` configuration. One should expect that the configuration of the downstream server should influence the IG `ClientHandler` and in turn, the overall IG server configuration.

Specific Vert.x options of interest here are:

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 will operate across, so the number of threads available. The default is the number of cores. See also **OPENIG-4256** - Getting issue details... STATUS .
root.vertx	eventLoopPoolSize	The total number of available event-loop threads to be supplied to instances - default 20. This value should be greater than the specified `gatewayUnits` setting supplied.
root.connectors.<connector>.vertx	acceptBacklog	Set the maximum number of connections to queue (before refusing requests).
	sendBufferSize	TCP connection send buffer size - set according to available RAM and number of concurrent connections required.
	receiveBufferSize	TCP receive buffer size - set according to available RAM and number of concurrent connections required.

Standalone (Vert.x) Troubleshooting Options

The following options may be useful for runtime monitoring and troubleshooting - but may affect performance:

Object	Vert.x Option	Description
root.vertx	blockedThreadCheckInterval blockedThreadCheckIntervalUnit	Interval at which Vert.x checks for blocked threads and logs a warning - default is 1 second.
	maxEventLoopExecuteTime maxEventLoopExecuteTimeUnit	Maximum time executing at which Vert.x logs a warning - default 2 seconds
	warningExceptionTime warningExceptionTimeUnit	Threshold at which warning logs are accompanied by a stack trace to identify causes. Default is 5 seconds.
	logActivity	Log network activity.

	metricsOptions.enabled	Turn on Vert.x metrics gathering:
		
		`"metricsOptions": {`
		`  "enabled": true`
		`}`
		
		See Vert.x Dropwizard Metrics for more details on the types of metrics captured. Currently DropWizard only.

References

1. KB FAQ: IG Performance and Tuning
2. KB HowTo: How do I collect data for troubleshooting high CPU utilization or Out of Memory errors on IG/OpenIG (All versions) servers?
3. KB Solutions: 502 Bad Gateway or SocketTimeoutException when using IG (All versions)
4. Vert.x Guide for Java Devs
5. Vertx Core Manual: Writing Verticles
6. Vertx Core Manual: Specifying the Number of Instances
7. Vert.x API: VertxOptions
8. Vert.x API: HttpServerOptions
9. Vert.x Dropwizard Metrics
10. Vertx scaling the number of instances per thread (StackOverflow) by @tsegismont
11. Vertx 2(!): Performance Tuning
12. **OPENIG-4256** - Getting issue details... STATUS

Related articles

- Tuning IG and the ClientHandler/ ReverseProxyHandler
- Tuning the JVM
- Tuning the Tomcat Container