Integration Technical Conference 2019

A03 Application Integration

All things HTTP inside ACEv11

David Coles
dcoles@uk.ibm.com

IBM **Cloud**

IBM

Agenda

- HTTP introduction
- HTTP Input/Reply
- Connectors and listeners
- HTTP Request including asynchronous
- Group nodes
- Demo

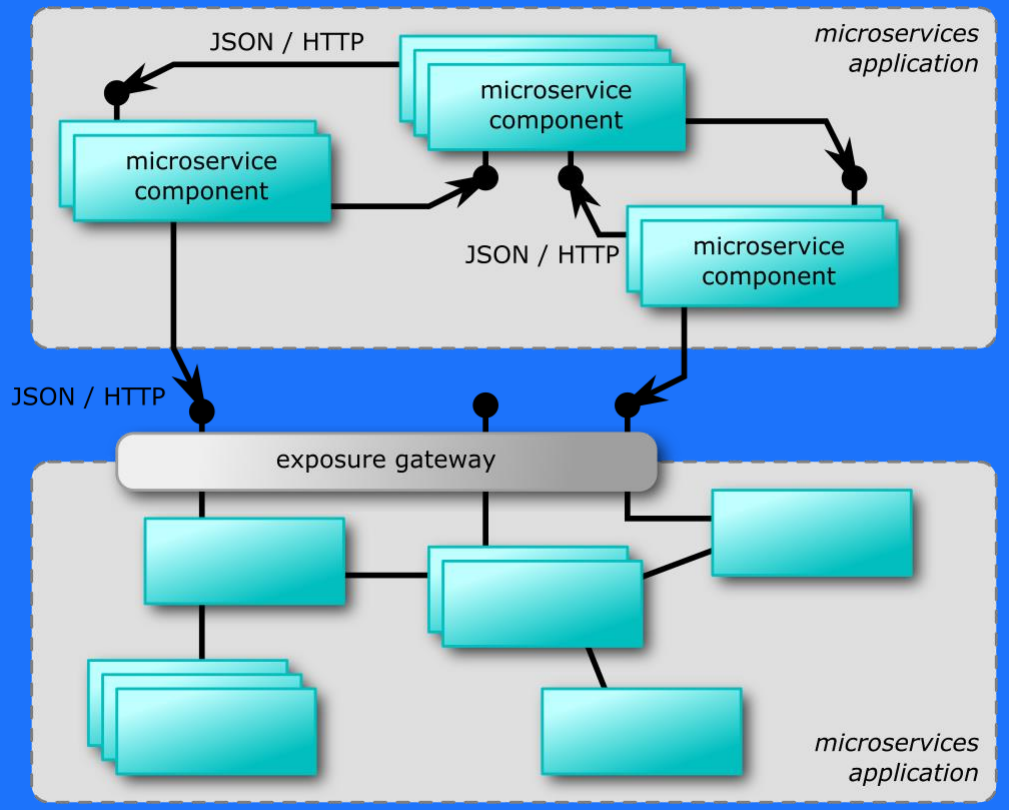
Why do we care about HTTP in 2018?

- Protocol development initiated in 1989
- Standards development by IETF and W3C

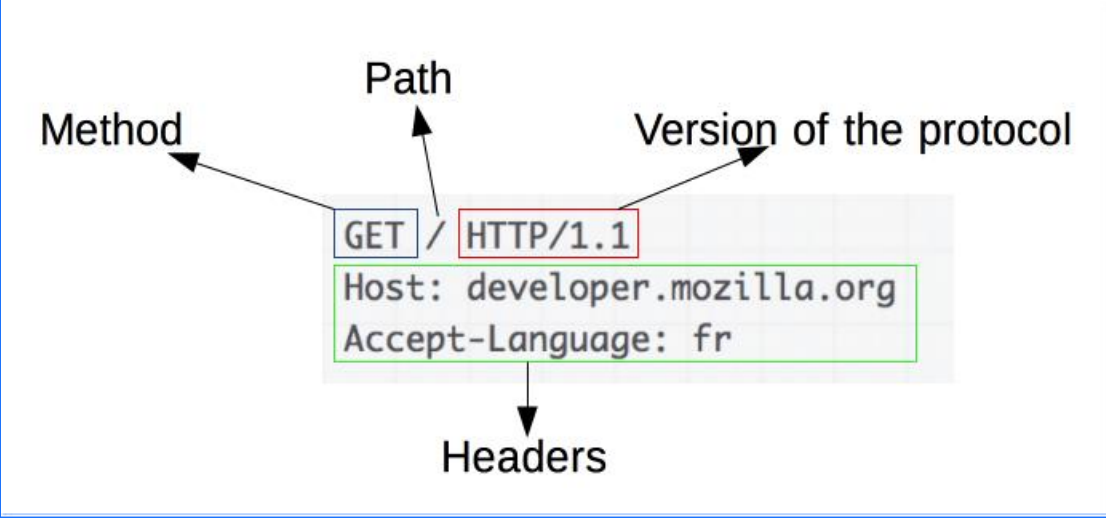
HTTP/1.1
- RFC 2068 (Jan 1997)
- RFC 2616 (Jun 1999)
- RFC 7230 (Jun 2014)

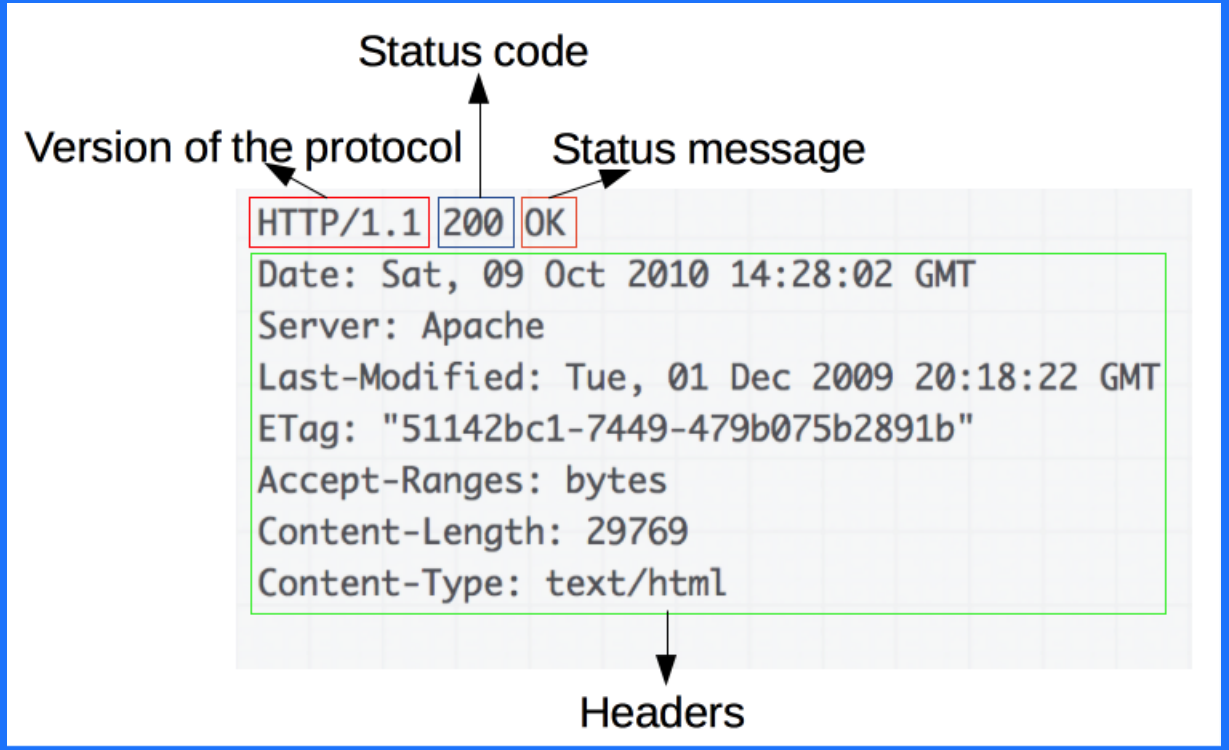
HTTP/2
- RFC 7540 (May 2015)

- Lightweight / low overhead, mature, and accessible protocol (practically) built on TCP/IP
- Near-universal support & wide adoption (WWW, SOAP, REST)
- Ideal fit for cloud and Agile Integration Architecture

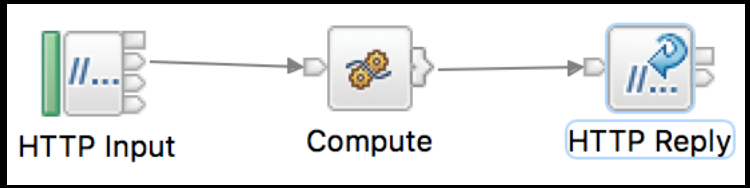


HTTP Protocol Explained





HTTPNodes – HTTPInput/Reply



IBM

Properties | Problems | Outline | Tasks | Deployment Log | Tutorial Steps View | Console | Progress

HTTP Input Node Properties - HTTP Input

- Description
- **Basic**
- Advanced
- Input Message Parsing
- Parser Options
- Error Handling
- Validation
- Security
- Monitoring

Settings for working with the HTTPInput node.

Path suffix for URL* /echo

e.g: /path/to/service, where the full url is htt

Use HTTPS ☐

HTTP Input Node Properties - HTTP Input

- Description
- Basic
- **Advanced**
- Input Message Parsing
- Parser Options

Advanced settings for the HTTPInput node

Set destination list ☐

Label prefix

Parse Query String ☐

Decompress input message ☐

HTTP Reply Node Properties - HTTP Reply

- Description
- **Basic**
- Validation
- Monitoring

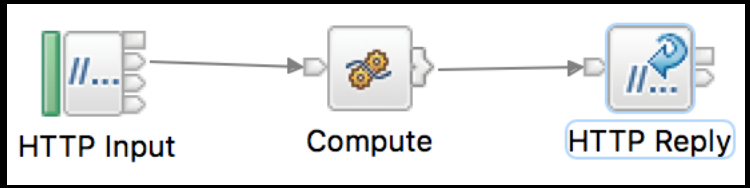
Settings for working with the HTTPReply node.

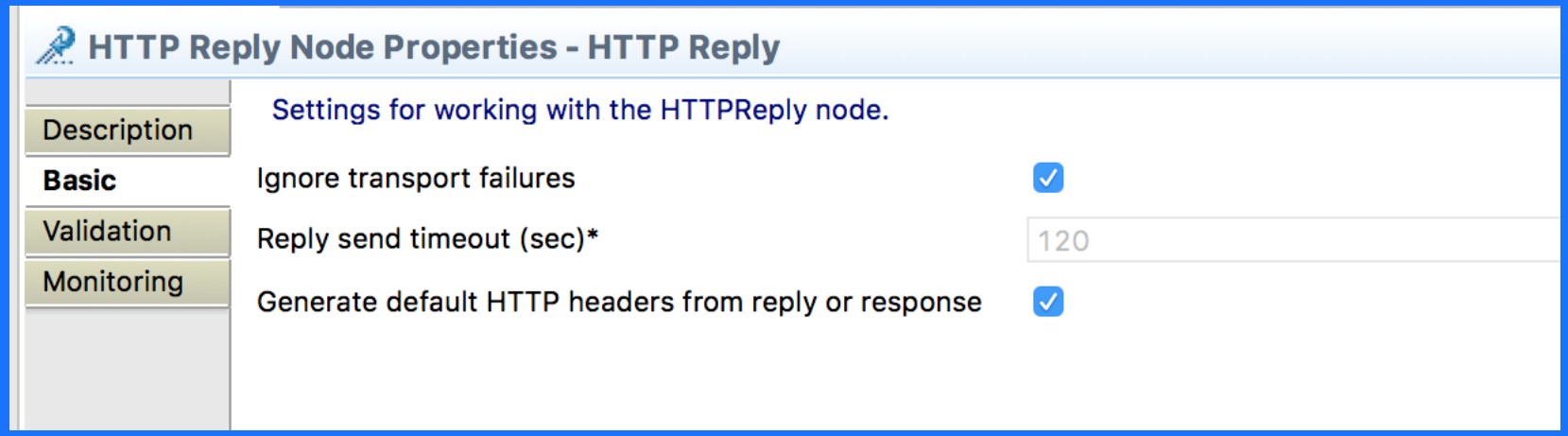
Ignore transport failures ☑

Reply send timeout (sec)* 120

Generate default HTTP headers from reply or response ☑

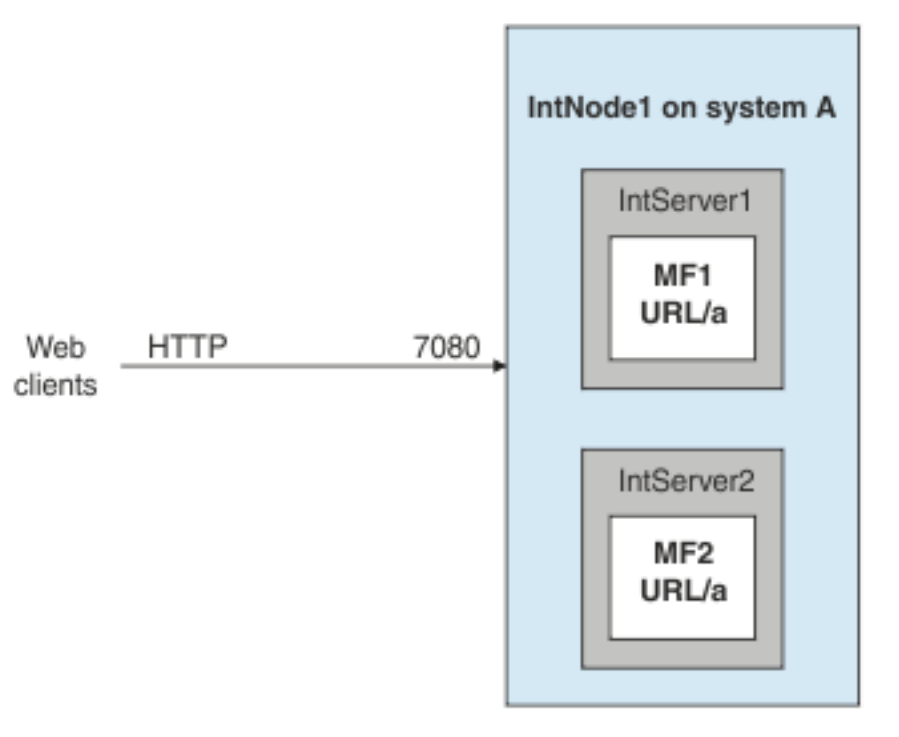
HTTPNodes – HTTPInput/Reply



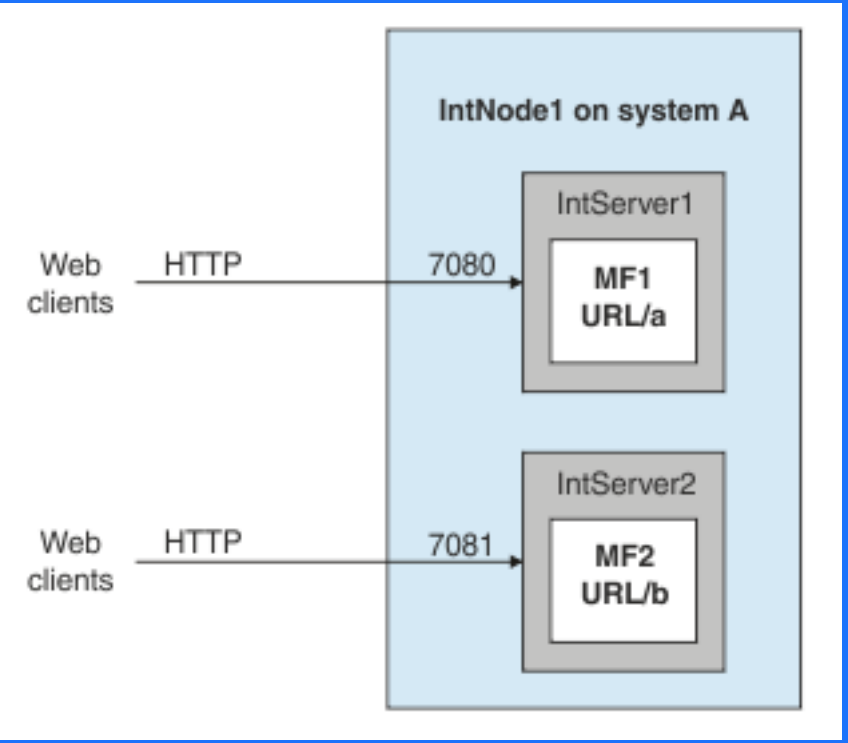


Listeners

- Node wide - biphttplistener

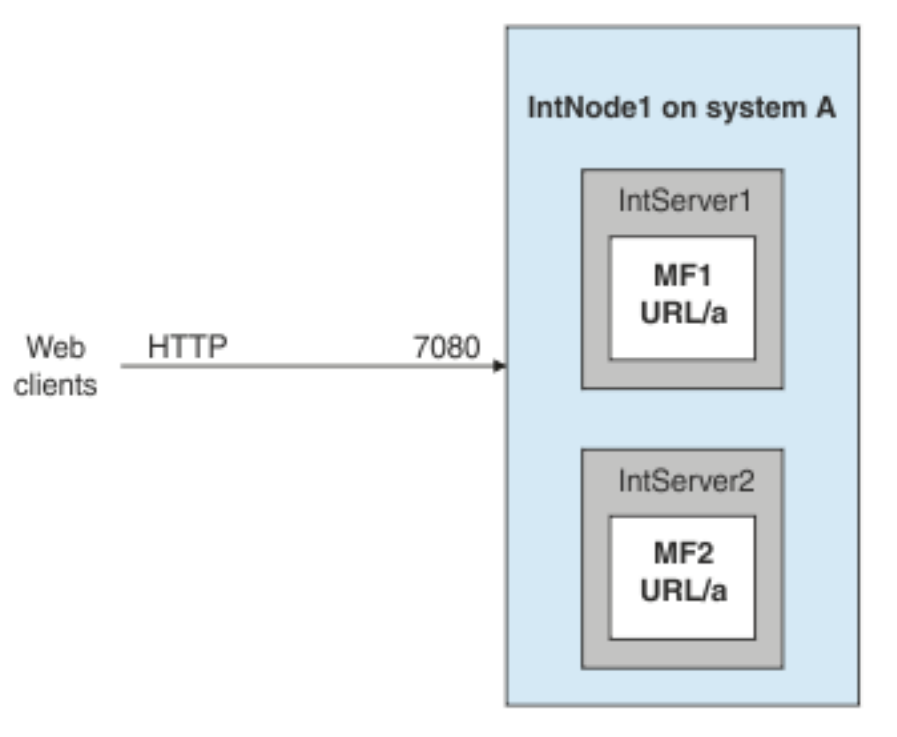


- Server wide - embedded



Listeners

- Node wide - biphttplistener

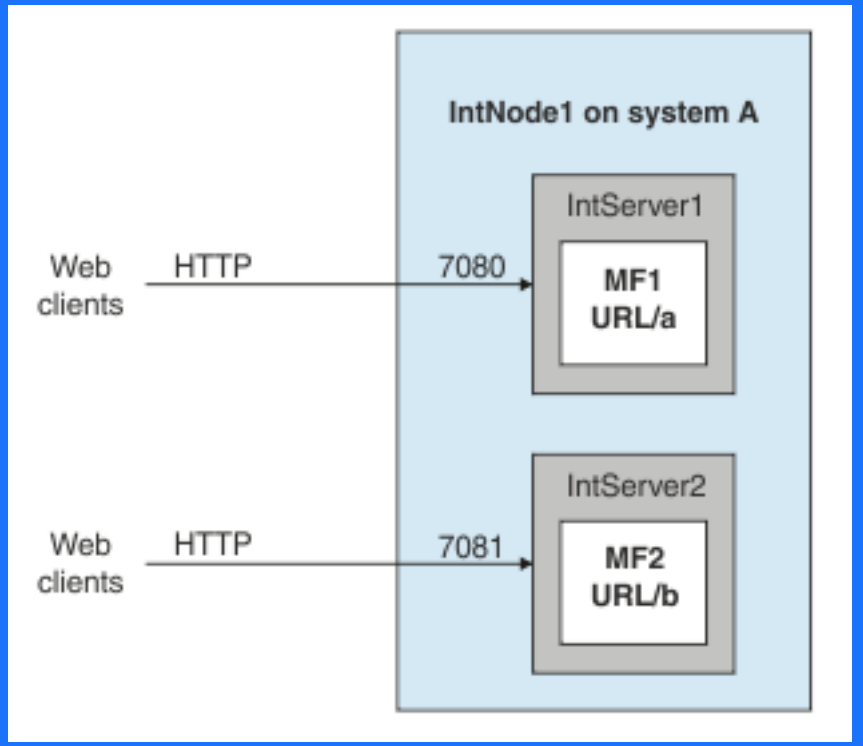


- Strengths
  - Simplicity of administration and web service discovery
  - All inbound requests are routed through a single port for HTTP/HTTPS
  - Load-balancing across message flows and integration servers
  - This configuration offers a scalable load-balanced solution with some degree of failover; if one integration server fails, the others continue to process the workload while the first integration server restarts.
- Weaknesses
  - Failover: there is a single point of failure
  - Activity partitioning: there is no partitioning between activities managed by the integration node.
  - Throughput: a single listener handles all HTTP and all HTTPS messages sent through two ports on the integration node. This single point of processing and error handling can cause bottlenecks if high message throughput is required.

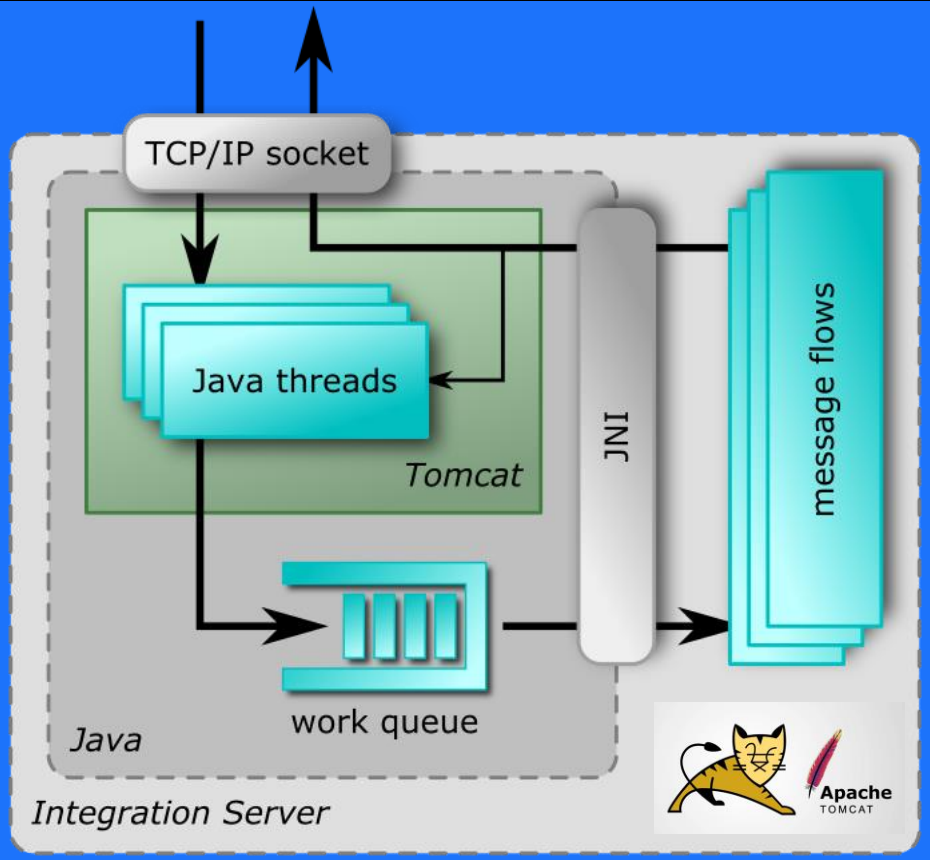
Listeners

- Strengths
  - High throughput: you can deploy message flows to different integration servers so that the HTTP (or HTTPS) messages can be handled by multiple listeners on multiple ports to meet high throughput requirements
  - Simple configuration: these listeners communicate directly with the HTTP transport network; no intermediate queues are required
  - Activity partitioning: activities managed by the integration node are partitioned into separate integration servers
- Weaknesses
  - Failover: there is a single point of failure (integration server, integration node, or machine). If machine failover is implemented, it is complicated, and involves the secondary machine taking over the IP address of the primary machine
  - You must include both the input and the reply nodes in the same message flow, or deploy separate message flows to the same integration server, so that they use the same listener; matching input and reply messages must be processed by the same port

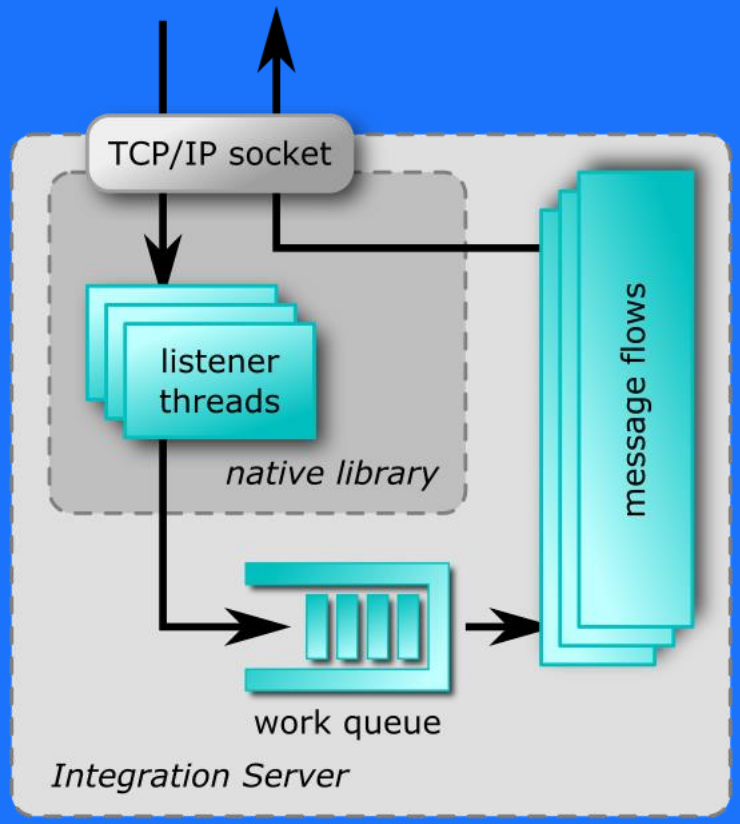
- Server wide - embedded



HTTP server I/O in IIB V10 vs. ACE V11

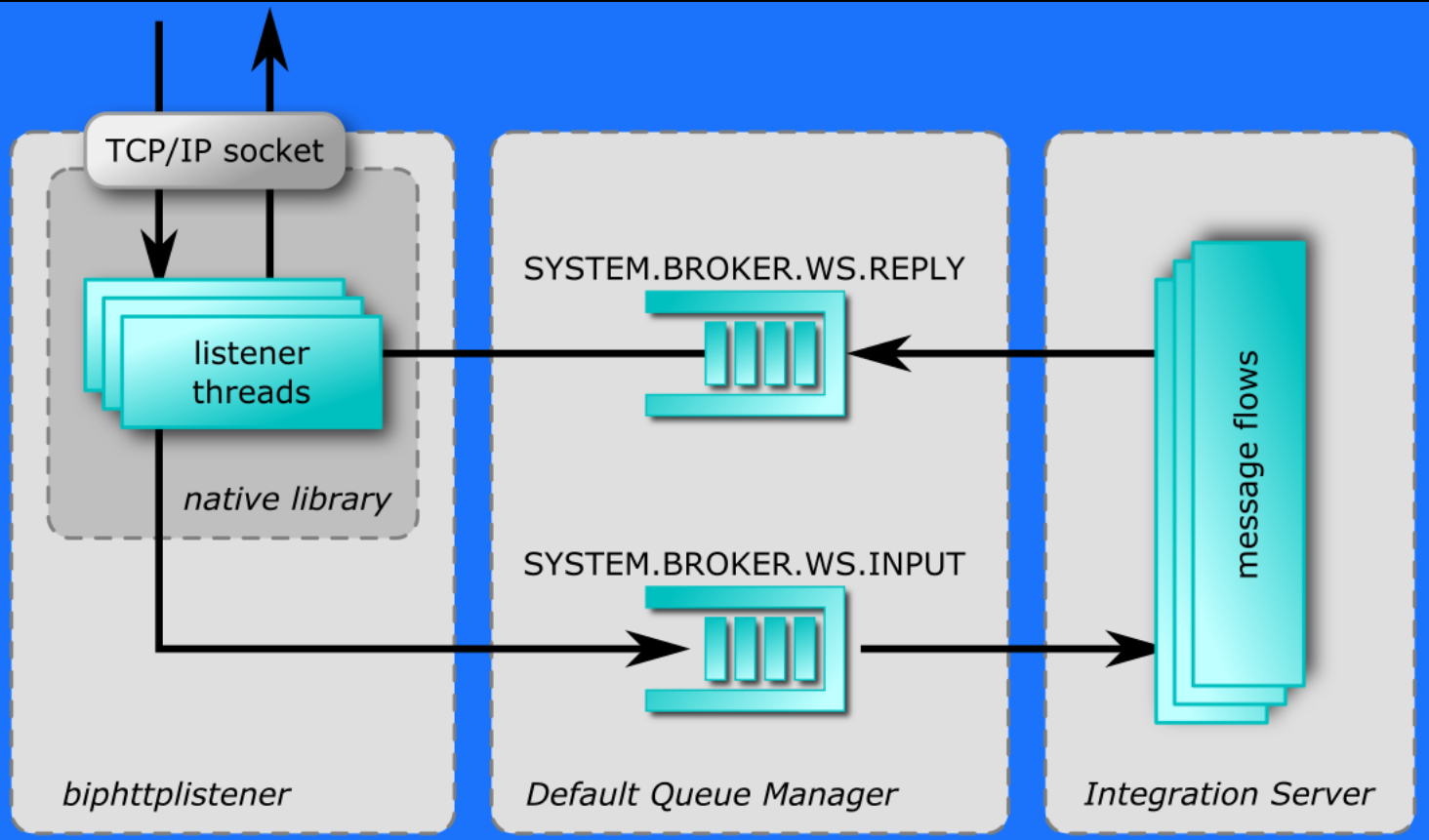


IBM Integration Bus V10



IBM App Connect Enterprise V11

Integration Node Listener



HTTPConnector/HTTPSConnector configuration

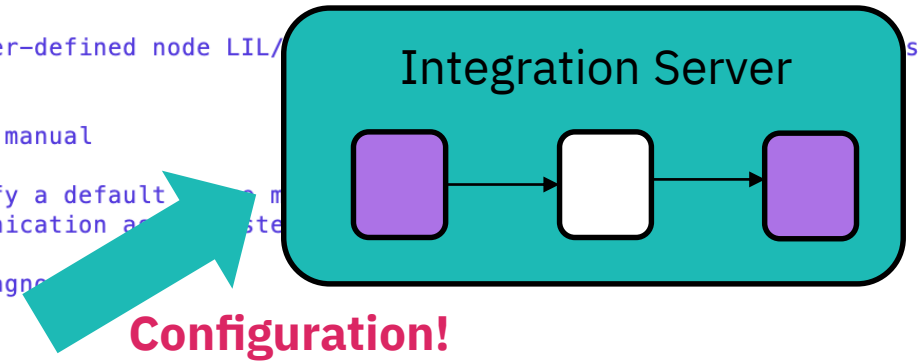
- Configuration of the listeners is through HTTPConnector and HTTPSConnector settings
- There are some common properties between both types of connector
  - Basic properties
    - **ListenerPort**: 7800
      - Port number for the listener to bind to
    - **ListenerAddress**: '0.0.0.0'
      - IP address for the listener to bind to
    - **ServerName**: ''
      - The name of the server to return in the HTTP response 'server' header
  - 6 x Performance tuning properties
  - 7 x CORS related properties
- The HTTPSConnector adds 13 more HTTPS specific ones

How to configure the connectors?

- V10: mqsichangeproperties
- V11: server.conf.yaml or policy
  - You could use mqsichangeproperties to set the properties in the yaml or edit manually

Configuration recap - server.conf.yaml

```
---
# ACE Integration Server configuration file
#
# General notes :
#  - Integration Server will load server.conf.yaml from directory set via --work-dir
#  - To ensure valid YAML avoid any use of TAB characters
#  - File paths may be taken as absolute, or relative to the integration server's work directory
#
# WARNING: Any value specified in the 'overrides/server.conf.yaml' will override values here
#
serverConfVersion: 1

#lilPath: ''                # A list of paths from where User-defined node LIL/
are separated by platform path separator)

#deployMode: 'replace'      # Deploy mode: replace | none | manual

#defaultQueueManager: ''    # Set non-empty string to specify a default
#mqTrustedQueueManager: 'no' # Set to yes to enable MQ communication a

#trace: 'none'              # choose 1 of : none|service|diagn
#traceSize: '1G'            # Set the service trace size
#traceNodeLevel: true       # Enable or disable Trace Nodes

#forceServerHTTPS: true     # force HTTPS on all HTTP/SOAP input nodes

Log:
  #consoleLog: true         # Control writing BIP messages to standard out. Set to true or false, default is true.
  #outputFormat: 'text'     # Set the format for writing BIP messages ibmjson or text, default is text.
  #eventLog: ''             # Control writing BIP messages to file. Set to '/path/log.txt' to enable, or '' to disable.
                            # Default is <workdir>/log/integration_server.<server_name>.events.txt
```



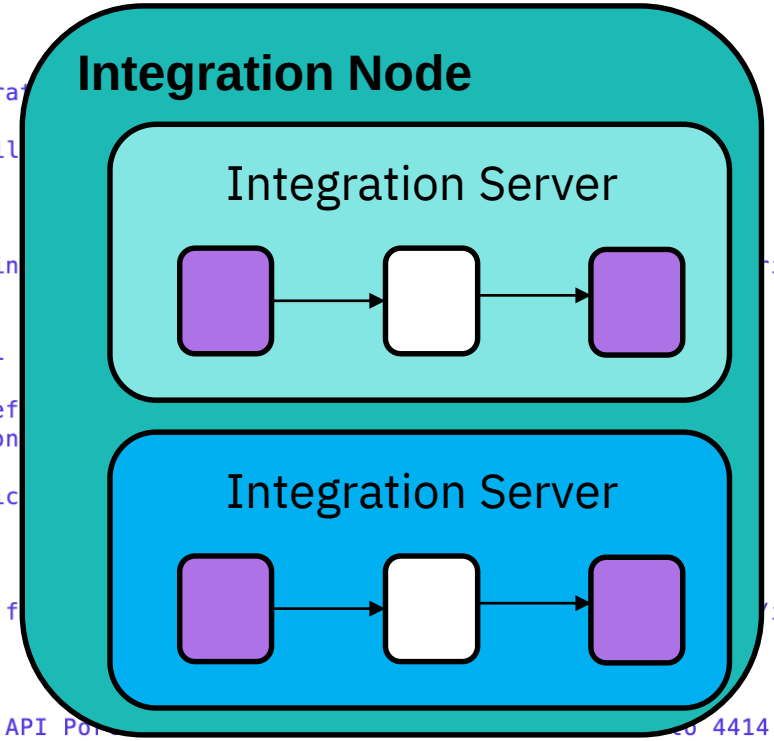
A heavily commented YAML file used for configuring an Integration Server.

All configuration required by the Integration Server for running message flows is taken from here.

No need to run commands aside from mqsisetdbparms.

Configuration recap - node.conf.yaml

```
---
# ACE Integration Node configuration file
#
# General notes :
#  - Integration Node will load node.conf.yaml from directory named after the Integration Node
#    in the components directory in the workpath:
#    <workpath> eg: /var/mqsi/
#     /components
#      /<node-name>
#       node.conf.yaml
#  - To ensure valid YAML avoid any use of TAB characters
#  - File paths may be taken as absolute, or relative to the integrat
#
# WARNING: Any value specified in the 'overrides/node.conf.yaml' wil
#
nodeConfVersion: 1

#lilPath: ''                 # A list of paths from where User-defin                ries
are separated by platform path separator)

#deployMode: 'replace'       # Deploy mode: replace | none          al

#defaultQueueManager: ''     # Set non-empty string          ry a def
#mqTrustedQueueManager: 'no' # Set to yes to enabl          ommunication

#agentTrace: 'none'          # choose 1 of : none|service|diagnostic
#agentTraceSize: '1G'        # Set the service trace size

Log:
  #eventLog: '/path/to/event/log.txt'   # Writes bip messages to a f                /inte
gration_server.<server_name>.events.txt

RestAdminListener:
  port: 4414                          # Set the Admin REST API Por                to 4414
```



Configuration!

A heavily commented YAML file used for configuring an Integration Node.

Integration Servers inherit properties from a node.conf.yaml , but can be overridden in their individual server.conf.yaml.

Resides in: $MQSI_REGISTRY/components/<NODE_NAME>/node.conf.yaml

Embedded listener configuration

Embedded listener connector properties found under '*ResourceManagers*' in server's server.conf.yaml

```
ResourceManagers:
  JVM:

  HTTPConnector:
    #ListenerPort: 0              # Set non-zero to set a specific port, defaults to 7800
    #ListenerAddress: '0.0.0.0'   # Set the IP address for the listener to listen on. Default is to listen on all addresses
    #CORSEnabled: false           # Set the value to true to make the listener respond to valid HTTP CORS requests
    #CORSAllowOrigins: '*'
    #CORSAllowCredentials: false
    #CORSExposeHeaders: 'Content-Type'
    #CORSMaxAge: -1
    #CORSAllowMethods: 'GET,HEAD,POST,PUT,PATCH,DELETE,OPTIONS'
    #CORSAllowHeaders: 'Accept,Accept-Language,Content-Language,Content-Type'

  HTTPSConnector:
    #ListenerPort: 0                # Set non-zero to set a specific port, defaults to 7843
    #ListenerAddress: '0.0.0.0'      # Set the IP address for the listener to listen on. Default is to listen on all addresses
    #ReqClientAuth: true
    #KeyAlias: ''
    #KeyPassword: 'P4s5w0rd'         # Set the password or alias to the password of the key
    #KeystoreFile: '/path/to/keystore.jks'
    #KeystorePassword: 'P4s5w0rd'    # Set the password or alias to the password of the keystore
    #KeystoreType: 'JKS'             # Set the keystore type, can be 'JKS' or 'p12'. Default is JKS.
    #TruststoreFile: /path/tp/truststore.jks
    #TruststorePassword: 'P4s5w0rd'  # Set the password or alias to the password of the keystore
    #TruststoreType: 'JKS'           # Set the truststore type, can be 'JKS' or 'p12'. Default is JKS.
    #CORSEnabled: false   # Set the value to true to make the listener respond to valid HTTP CORS requests
    #CORSAllowOrigins: '*'
    #CORSAllowCredentials: false
    #CORSExposeHeaders: 'Content-Type'
    #CORSMaxAge: -1
    #CORSAllowMethods: 'GET,HEAD,POST,PUT,PATCH,DELETE,OPTIONS'
    #CORSAllowHeaders: 'Accept,Accept-Language,Content-Language,Content-Type'
    #EnableTLSTrace: false           # Enables tracing of TLS handshake messages to the console

  ActivityLogManager:
```

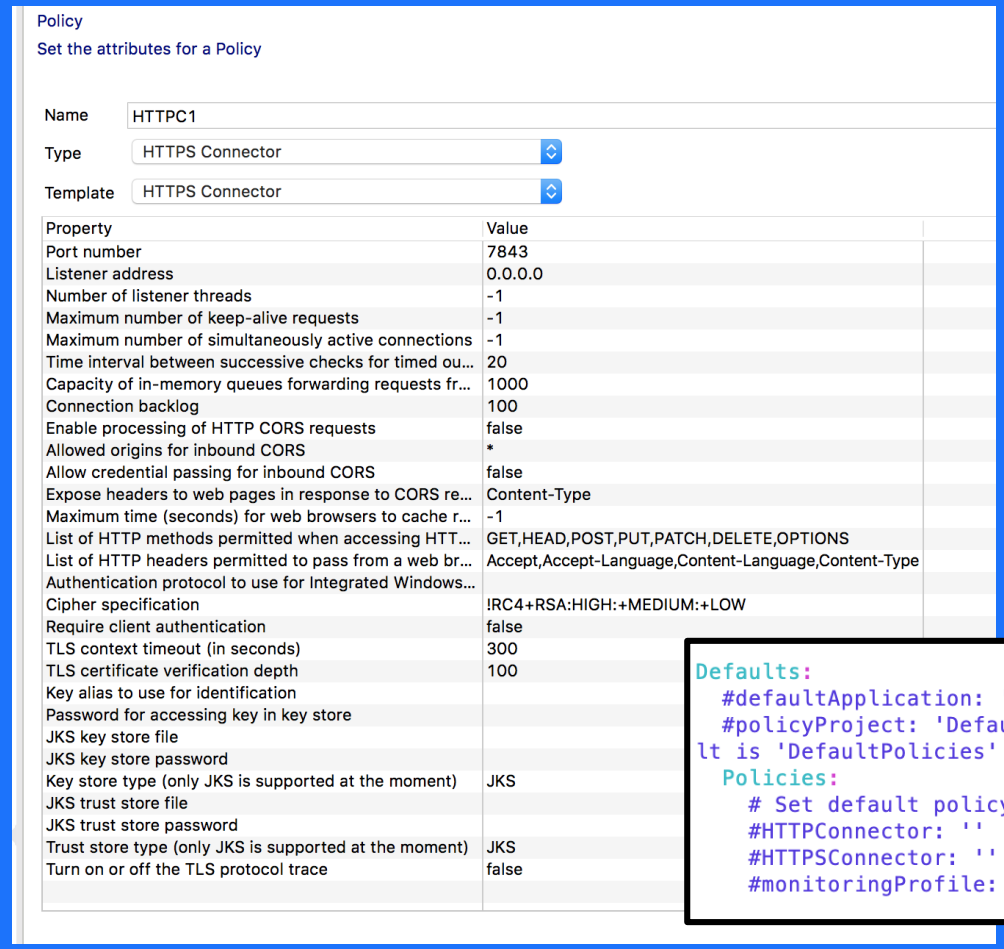
Node-wide listener configuration

Node-wide listener connector properties found under '*NodeHttpListener*' in node's node.conf.yaml

```
NodeHttpListener:
  #startListener: true                  # Enables the Integration Node listener
  #trace: 'none'                        # choose 1 of : none|service|diagnostic
  #traceSize: '1G'                      # Set the service trace size

  HTTPConnector:
    ListenerPort: 7080            # Set non-zero to set a specific port, defaults
    #ListenerAddress: '0.0.0.0'   # Set the IP address for the listener to listen
    #CORSEnabled: false           # Set the value to true to make the listener resp
    #CORSAllowOrigins: '*'
    #CORSAllowCredentials: false
    #CORSExposeHeaders: 'Content-Type'
    #CORSMaxAge: -1
    #CORSAllowMethods: 'GET,HEAD,POST,PUT,PATCH,DELETE,OPTIONS'
    #CORSAllowHeaders: 'Accept,Accept-Language,Content-Language,Content-Type'

  HTTPSConnector:
    ListenerPort: 7083              # Set non-zero to set a specific port, defaults to 7083
    #ListenerAddress: '0.0.0.0'     # Set the IP address for the listener to listen on. Default is to listen on all addresses
    #ReqClientAuth: true
    #KeyAlias: ''
    #KeyPassword: 'P4s5w0rd'        # Set the password or alias to the password of the key
    #KeystoreFile: '/path/to/keystore.jks'
    #KeystorePassword: 'P4s5w0rd'   # Set the password or alias to the password of the keystore
    #KeystoreType: 'JKS'            # Set the keystore type, can be 'JKS' or 'p12'. Default is JKS.
    #TruststoreFile: /path/tp/truststore.jks
    #TruststorePassword: 'P4s5w0rd'  # Set the password or alias to the password of the keystore
    #TruststoreType: 'JKS'           # Set the truststore type, can be 'JKS' or 'p12'. Default is JKS.
    #CORSEnabled: false   # Set the value to true to make the listener respond to valid HTTP CORS requests
    #CORSAllowOrigins: '*'
    #CORSAllowCredentials: false
    #CORSExposeHeaders: 'Content-Type'
    #CORSMaxAge: -1
    #CORSAllowMethods: 'GET,HEAD,POST,PUT,PATCH,DELETE,OPTIONS'
    #CORSAllowHeaders: 'Accept,Accept-Language,Content-Language,Content-Type'
```

'*NodeHttpListener*' section also includes properties to control whether the listener starts and trace enablement

HTTPConnector / HTTPSConnector policy



Policy

Set the attributes for a Policy

Name: HTTPC1

Type: HTTPS Connector

Template: HTTPS Connector

Property	Value
Port number	7843
Listener address	0.0.0.0
Number of listener threads	-1
Maximum number of keep-alive requests	-1
Maximum number of simultaneously active connections	-1
Time interval between successive checks for timed ou...	20
Capacity of in-memory queues forwarding requests fr...	1000
Connection backlog	100
Enable processing of HTTP CORS requests	false
Allowed origins for inbound CORS	*
Allow credential passing for inbound CORS	false
Expose headers to web pages in response to CORS re...	Content-Type
Maximum time (seconds) for web browsers to cache r...	-1
List of HTTP methods permitted when accessing HTT...	GET,HEAD,POST,PUT,PATCH,DELETE,OPTIONS
List of HTTP headers permitted to pass from a web br...	Accept,Accept-Language,Content-Language,Content-Type
Authentication protocol to use for Integrated Windows...	
Cipher specification	!RC4+RSA:HIGH:+MEDIUM:+LOW
Require client authentication	false
TLS context timeout (in seconds)	300
TLS certificate verification depth	100
Key alias to use for identification	
Password for accessing key in key store	
JKS key store file	
JKS key store password	
Key store type (only JKS is supported at the moment)	JKS
JKS trust store file	
JKS trust store password	
Trust store type (only JKS is supported at the moment)	JKS
Turn on or off the TLS protocol trace	false

- Configure integration server embedded listener
- Create, configure & deploy HTTPConnector or HTTPSConnector policy
- Update server.conf.yaml to point at the policy to be used
- Can reference policy using the policy project syntax:
  {PolicyProjectName}:<policyName>

```
Defaults:
  #defaultApplication: ''            # Name a default application under which independent resources will be placed
  #policyProject: 'DefaultPolicies'  # Name of the Policy project that will be used for unqualified Policy references, defau
lt is 'DefaultPolicies'
  Policies:
    # Set default policy names, optionally qualified with a policy project as {policy project}:name
    #HTTPConnector: ''           # Default HTTP connector policy
    #HTTPSConnector: ''          # Default HTTPS connector policy
    #monitoringProfile: ''       # Default Monitoring profile
```

HTTPSConnector capabilities

- Uses OpenSSL to provide the encryption capabilities
  - This is a change over V10 which used IBM Java JSSE security provider
- HTTPSConnector requires a keystore configured which contains a private key
  - The connector uses this private key to initiate the SSL handshake with the client
  - The client must have the public certificate for the handshake to succeed
  - Keystores can be either JKS or PKCS#12 format
- Can enable client authentication which requires a truststore configured
  - Clients are required to provide a key for the handshake to succeed
  - The connector validates the key provided by the client against the certificates in the truststore
  - Truststores can be either JKS or PEM format
  - If a keystore or truststore is not configured then the connector falls back to using the one configured on the JVMManager or BrokerRegistry in that order of precedence
- Only supports TLSv1.2 protocol
- SSL handshake can be restricted to certain ciphers
  - Supports IANA cipher names or OpenSSL format

HTTPSConnector configuration – 1/2

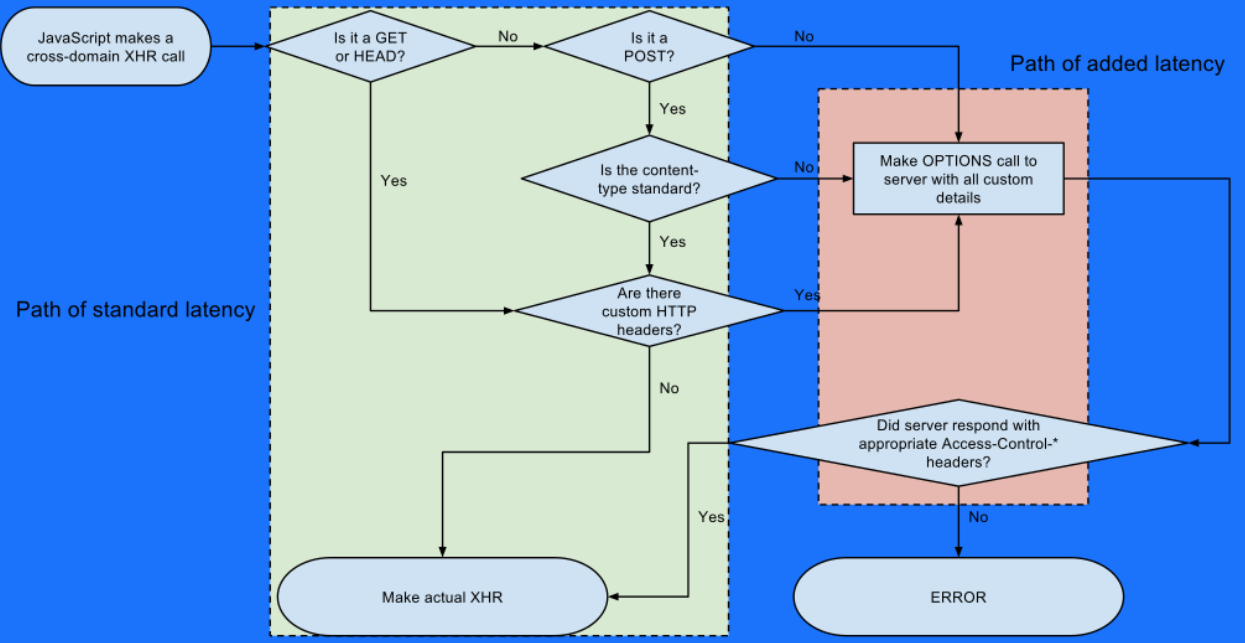
- HTTPSConnector specific properties:
  - **CipherSpec**: '!RC4+RSA:HIGH:+MEDIUM:+LOW'
    - Specifies the ciphers which are allowed to be used for the SSL Handshake
    - Can be in either OpenSSL format, or using IANA cipher names
  - **KeystoreFile** / **KeystorePassword** / **KeystoreType**
    - Specifies the keystore and its credentials, to use to initiate the SSL handshake
    - Password can be plaintext or reference an mqsisetdbparms resource
    - Type can be either JKS or PEM
  - **KeyAlias** / **KeyPassword**
    - Specifies the key and its password if different to the keystore password to use from the configured keystore
    - If a specific key is not specified then the connector will pick one from the available ones in the keystore
  - **ReqClientAuth**: false
    - If set to true, the server will reject TLS connections that don't provide a valid client certificate.
    - Once validated the certificate is propagated to the message flow via the LocalEnvironment message tree

HTTPSConnector configuration – 2/2

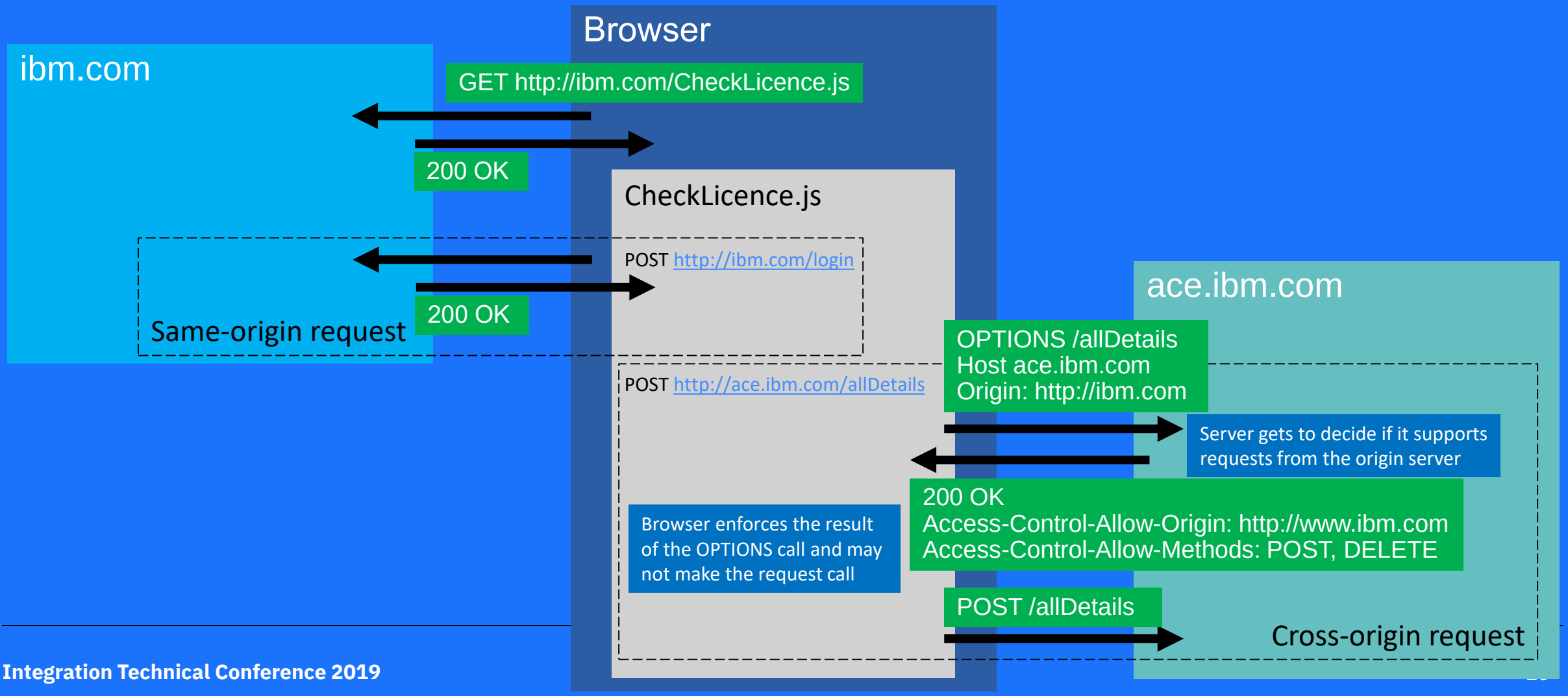
- HTTPSConnector specific properties continued:
  - **TruststoreFile** / **TruststorePassword** / **TruststoreType**
    - Specifies the trustore to use for validating client keys when client authentication is enabled
    - Password can be plaintext or reference an mqsisetdbparms resource
    - Type can be either JKS or PEM
  - **TLSContextTimeout**: 300
    - Sets the timeout for SSL sessions after which the session is not re-used and will renegotiated
  - **TLSCertVerifyDepth**: 100
    - Sets the maximum depth for certificate chain verifications that shall be allowed
  - **EnableTLSTrace**: false
    - Enables ssl handshake trace to ACE service trace

Cross-origin Resource Sharing - CORS

- Allows restricted resources on a web-page to be requested from another domain outside the domain from which the first resource was served (cross-origin)
- A web-page can freely embed cross-origin stylesheets, scripts, i-frames, and videos
- Certain cross-origin requests, notable AJAX requests, are forbidden by the same-origin security policy (default)
- Browser driven and enforced
- Server needs to understand CORS requests and respond appropriately
- Allows more freedom than allowing purely same-origin requests, but is more secure than allowing all cross-origin requests.



Cross-origin Resource Sharing - Example



Cross-origin Resource Sharing in ACE

- If you are making an HTTP request from a web page to a REST API or other HTTP service deployed to ACE, it is likely that a cross-origin request will need to be made
- Built in support for CORS available on embedded and node wide listeners
  - Configured through HTTPConnector/HTTPSConnector properties and disabled by default
- Connector properties which control CORS
  - **CORSEnabled**: false
    - Enable processing of HTTP CORS requests
  - **CORSAllowOrigins**: '*'
    - Allowed origins for inbound CORS
  - **CORSAllowCredentials**: false
    - Allow credential passing for inbound CORS
  - **CORSExposeHeaders**: 'Content-Type'
    - Expose headers to web pages in response to CORS requests
  - **CORSMaxAge**: -1
    - Maximum time (seconds) for web browsers to cache responses to CORS requests
  - **CORSAllowMethods**: 'GET,HEAD,POST,PUT,PATCH,DELETE,OPTIONS'
    - List of HTTP methods permitted when accessing HTTP services
  - **CORSAllowHeaders**: 'Accept,Accept-Language,Content-Language,Content-Type'
    - List of HTTP headers permitted to pass from a web browser to HTTP services

Tuning V11 connectors – 1/2

- For embeddded listeners the threading model is now completely different to before
  - Used to have 1 thread per connection which would be parked to await the response
  - Now a listener thread accepts the incoming connection, performs SSL handshake and initial validation of the request and then puts the connection on a lock-free in-memory queue for the flow to pick from
  - The flow then picks from the in-memory queue, process the request and makes the reply from the flow thread
  - A separate sweeper thread runs monitoring for timeouts
- Node wide listener still has a thread per connection
  - Thread which picks up the request writes it to a queue and then sits and waits for the response
  - Thread sends the timeout response if a response is not received before the timeout deadline

Tuning V11 connectors – 2/2

- Performance tuning parameters:
  - **MaxKeepAliveRequests**: -1
    - The number of HTTP requests that can be sent in a single TCP connection. Values less than 1 are treated as infinity
  - **ListenerThreads**: -1
    - Number of listener threads to use. Will revert to a listener-specific default value if set to less than 1
    - Embedded listener default is #cpu/2
    - Node-wide listener default is 200
  - **TimeoutSweepInterval**: 20
    - Number of milliseconds to wait between successive checks for timed out in-flight messages on the timeout sweeper thread
  - **QueueCapacity**: 1000
    - The number of requests the internal queues connecting the native listeners to message flows can hold
  - **ConnBacklog**: 100
    - Number of inbound TCP connections that can be queued up before accepted by the listener

V10 and earlier to V11 property migration

- Moving from the Tomcat based listener to the native one has mean that a number of Tomcat specific configuration parameters are no longer relevant and have been removed
  - shutdownDelay
  - maxSavePostSize
  - tcpNoDelay
  - allowTrace
  - bufferSize / socketBuffer
  - maxThreads / minSpareThreads / maxSpareThreads
  - noCompressionUserAgents
  - restrictedUserAgents
- Some other properties have been renamed
- If you use mqsiextractcomponents to migrate your V10 and earlier configuration to V11 it will take into account these changes
  - Renamed properties will be renamed
  - Removed properties with non-default values will generate warnings
- sslProtocol gone for now
  - New listener only supports TLSv1.2

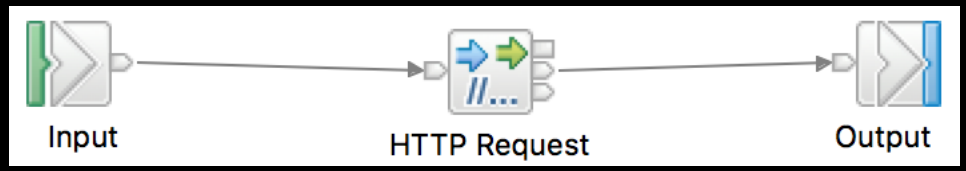
Using the embedded or node-wide listener

- A node owned server's server.conf.yaml contains 2 properties which control which listener HTTP or SOAP nodes use:

```
ResourceManagers:
  ExecutionGroup:
    #httpNodesUseEmbeddedListener: false   # Configures whether the HTTP nodes uses the node wide or embedded listener
    #soapNodesUseEmbeddedListener: true    # Configures whether the SOAP nodes uses the node wide or embedded listener
```

- HTTP nodes default to using the node wide listener if there is a default Queue Manager defined
  - Otherwise they default to using the embedded listener
- SOAP nodes always default to using the embedded listener

HTTPNodes – HTTPRequest



IBM

Properties | Problems | Outline | Tasks | Deployment Log | Tutorial Steps View | Console | Progress

HTTP Request Node Properties - HTTP Request

- Description
- **Basic**
- HTTP Settings
- SSL
- Response Message Parsing
  - Parser Options
- Error Handling
- Advanced
- Validation
- Monitoring

Settings for working with the HTTPRequest node.

Web service URL* http://127.0.0.1:7800/echo_DefaultPropagationOnFlow

e.g. http://server/path/to/service

Request timeout (sec)* 120

HTTP Request Node Properties - HTTP Request

- Description
- Basic
- **HTTP Settings**
- SSL
- Response Message Parsing
  - Parser Options
- Error Handling
- Advanced
- Validation
- Monitoring

HTTP settings for the HTTPRequest node

HTTP(S) proxy location

Follow HTTP(S) redirection ☐

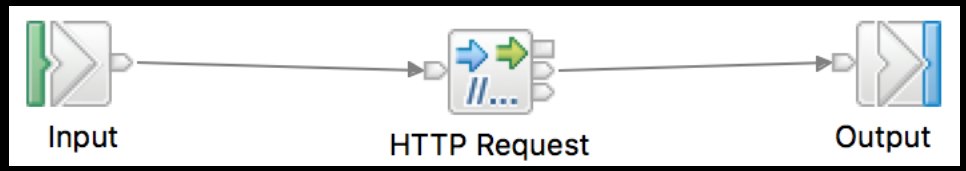
HTTP method* POST

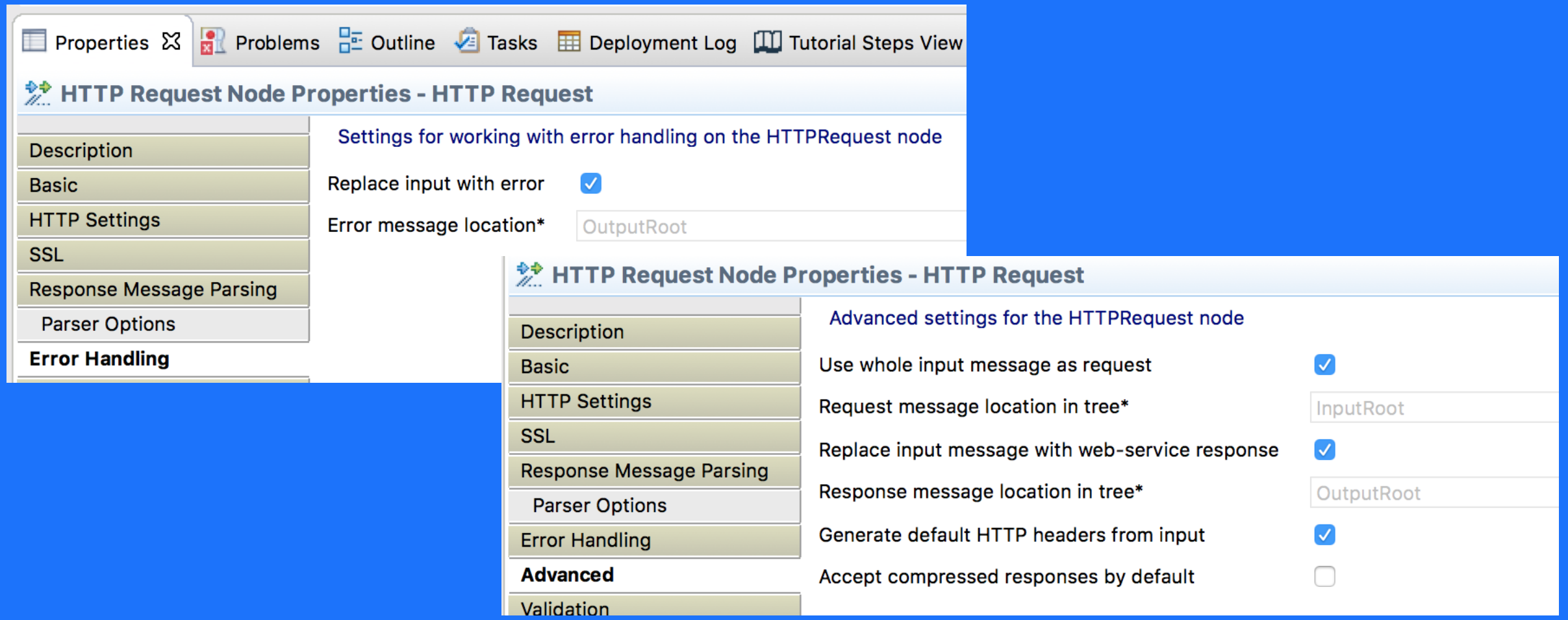
HTTP version* 1.1

Enable HTTP/1.1 keep-alive ☑

Use compression* none

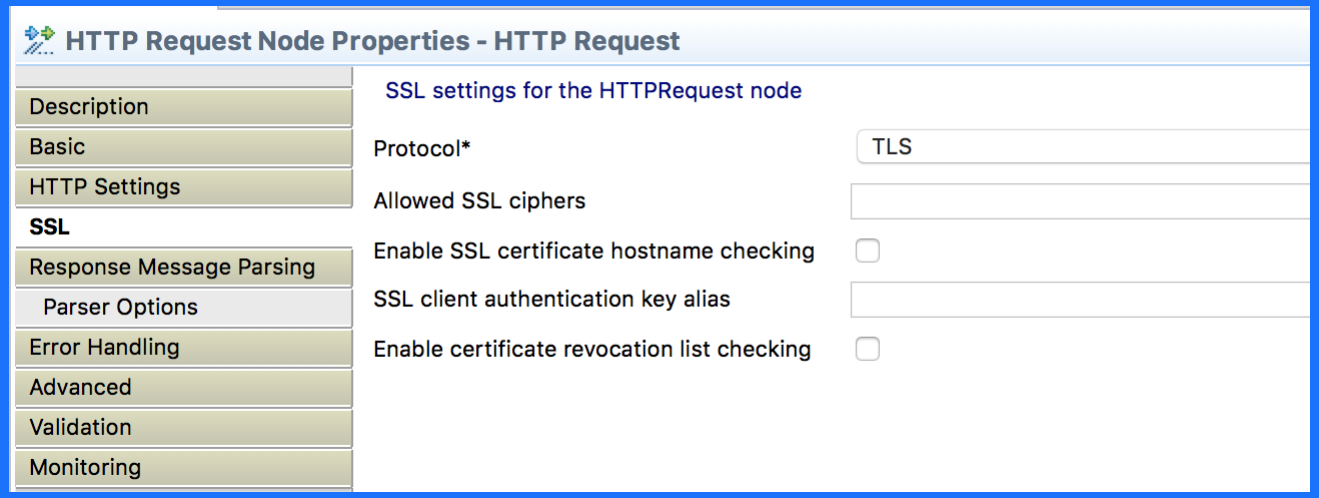
HTTPNodes – HTTPRequest





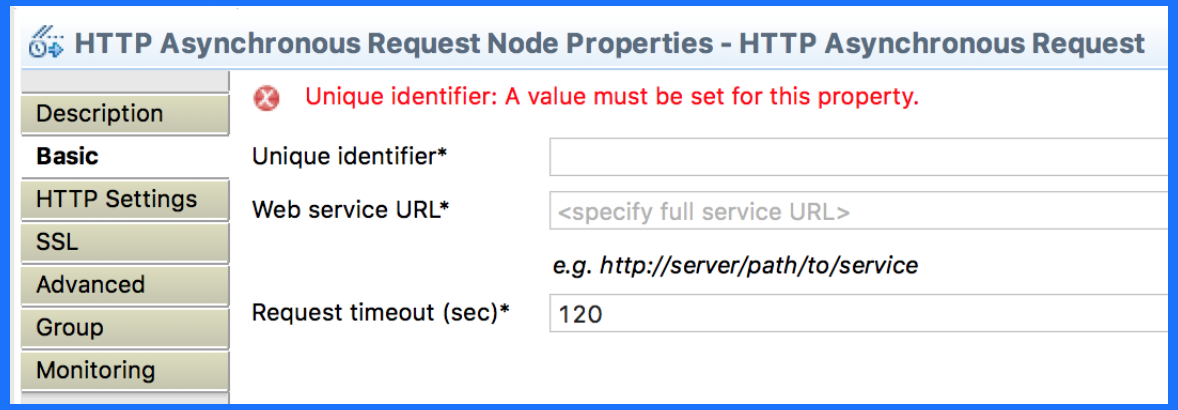
HTTPNodes – HTTPRequest - HTTPS

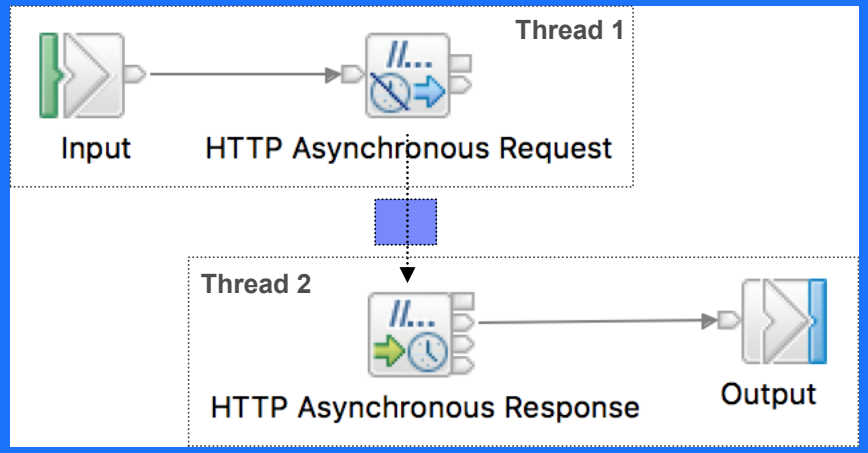
- Configure a 'Web service url' which starts with '*https://*'
- Uses IBM Java JSSE to provide SSL capabilities
- Requires a truststore to be configured to validate the key returned by the server
  - If the server requires client authentication then a configured keystore will also be required
  - The trustore and keystore used comes from those configured on JVMManger or BrokerRegistry in that order of precedence
  - Supports JKS format key/truststores
- Other properties available on the node to set or enable
  - SSL Protocol
  - Allowable ciphers for the SSL handshake
  - The alias of the key to use for client auth
  - Certificate hostname checking
  - Certificate revocation list checking
- Enable SSL trace by setting
  - IBM_JAVA_OPTIONS=-Djavax.net.debug=true
  - Trace is written to process stdout



HTTPAsyncRequest/Response Nodes

- Make asynchronous HTTP Requests
- Both SOAP & HTTP request nodes can request and handle response on different threads
  - Even without WS-Addressing on SOAP nodes
- User Data can be automatically saved from request to response node to simplify processing
- Particularly useful for slow running HTTP servers – reduces concurrent threads & memory demand
- Same properties as normal HTTPRequest node, but need to specify a unique identifier to tie asynchronous request/response nodes together

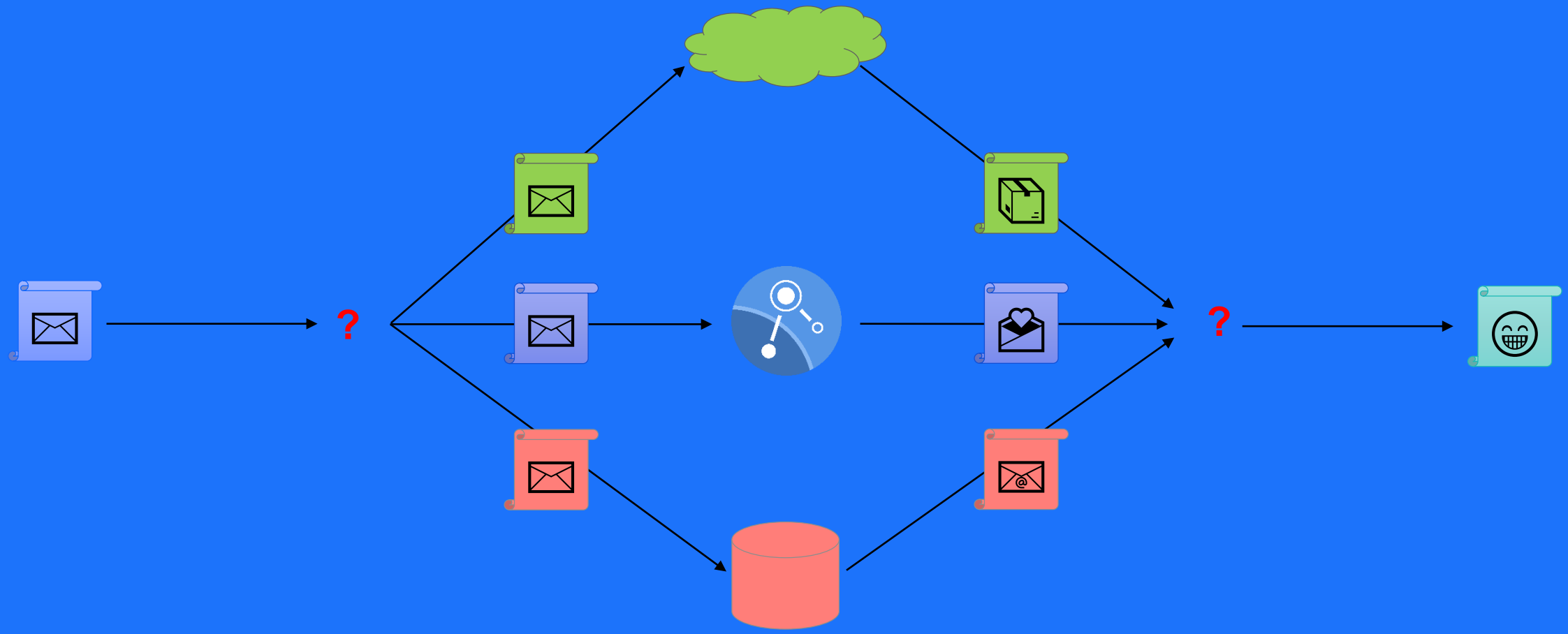




… and now for a brief interlude

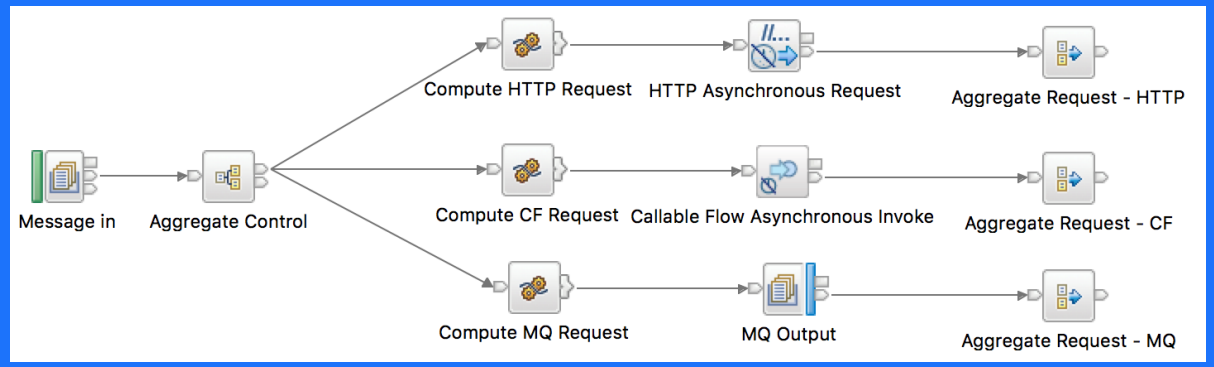
… Group Nodes

Introduction to Aggregation

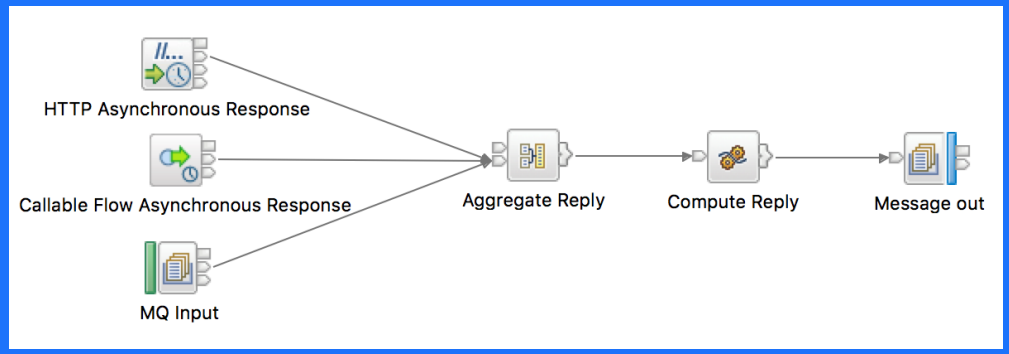


Existing aggregation patterns - Classic aggregation

- Fan-out/Scatter flow
  - Aggregate Control node marks the start of an aggregation
    - Uses LocalEnvironment to send data downstream
  - Aggregate Request nodes record details of the sent requests
    - Reads the ReplyId/CorrelationID from the LocalEnvironment
    - Sends data back upstream using the Environment
  - Control data stored across 5 MQ queues

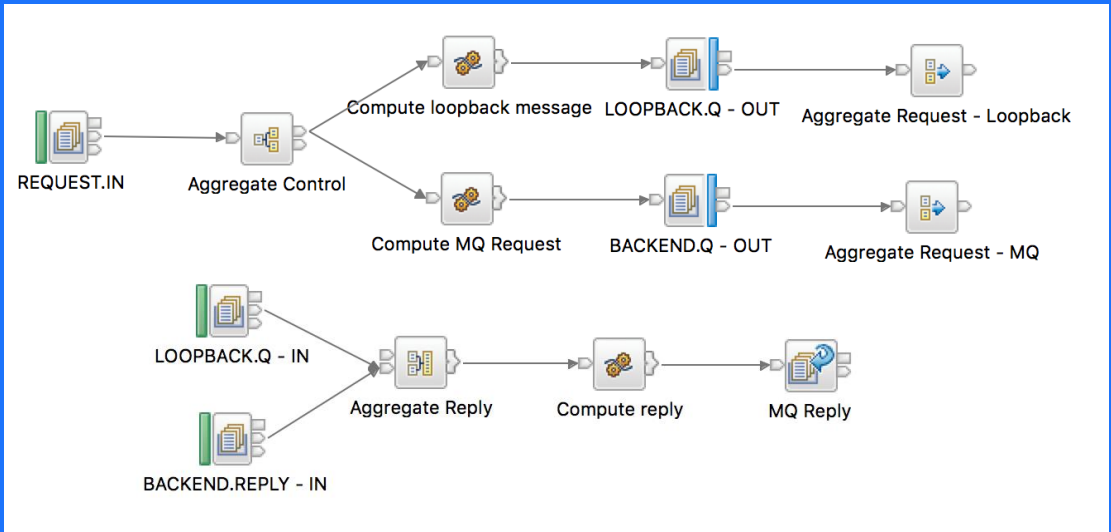


- Fan-in/Gather flow
  - Aggregate Reply matches replies based on their Reply ID/Message ID
  - Replies to incomplete aggregations are stored on an MQ queue
  - Unknown messages are stored on an MQ queue for reconciliation with future aggregations
  - Outputs completed aggregations, timed out aggregations, and timed out unknown messages

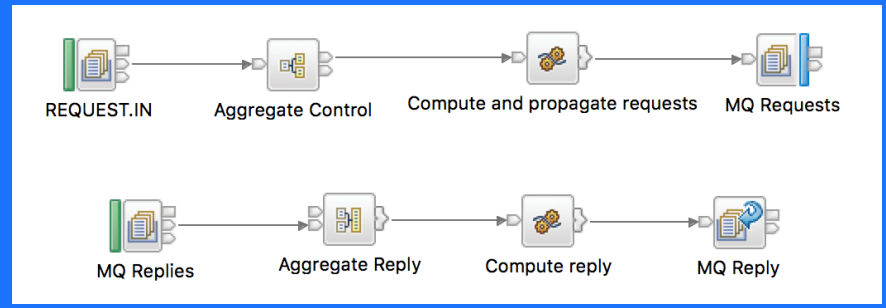


Existing aggregation patterns - Common patterns

- Request-reply
  - Use a loopback queue to store input message or message context
  - Add multiple optional extra requests based on input
  - Loopback queue is a straight passthrough
  - MQ Reply node needs original Properties.ReplyIdentifier to be set

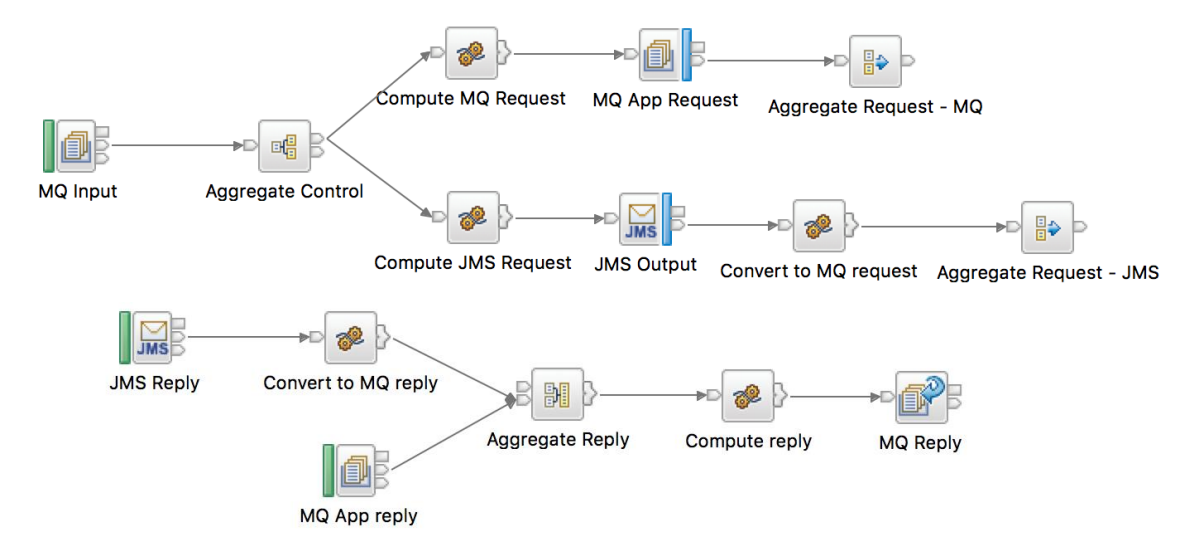


- Dynamic aggregation
  - Use a Compute node and PROPAGATE TO TERMINAL to construct custom MQ requests.
  - Flow designer is responsible for updating Environment with Aggregation data
  - Allows the most control over the aggregation and requests

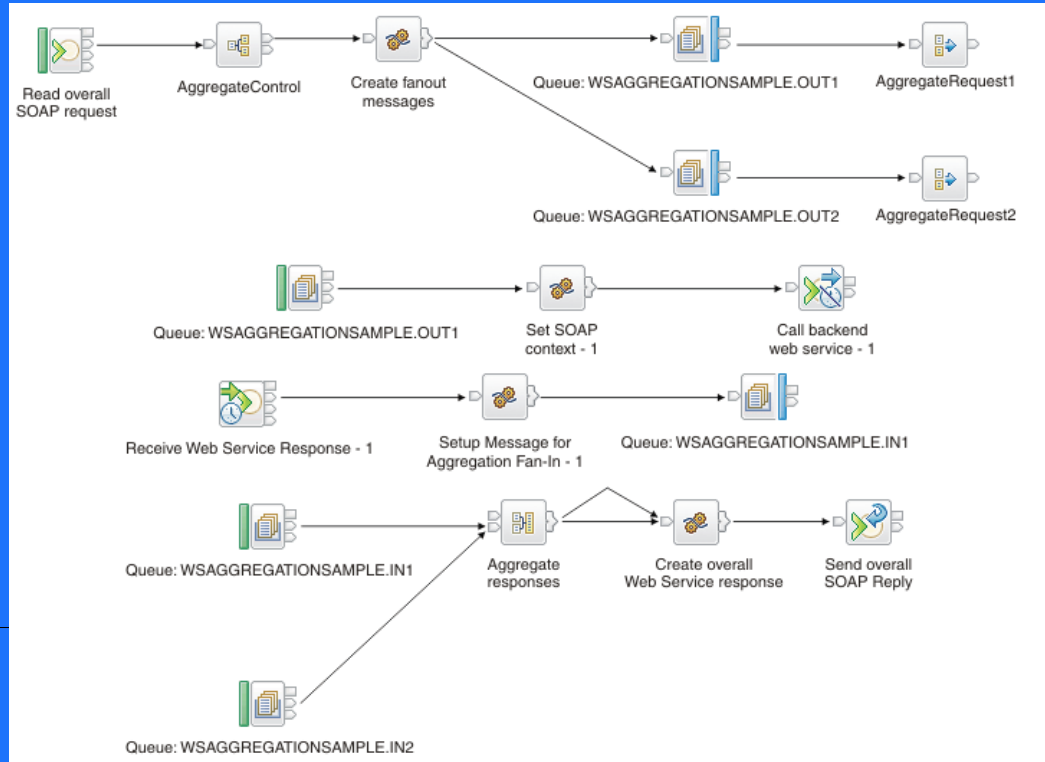


Existing aggregation patterns - Common patterns

- Custom transports
  - Aggregate nodes think everything is MQ
  - Only MQBYTE24 compatible MsgIds and CorrelIds are usable
  - Non-supported transports must use compute nodes to mimic MQ transport data



- Semi-historic pattern - MQ proxied requests
  - More common in earlier versions of WMB before 6.1.0.8
  - Proxy all non-MQ requests behind MQ
  - Aggregation is completely transactional
  - Needs additional flows and queues



Source: IIB v9 Knowledge Centre *Web Service Aggregation* sample

Aggregation Nodes - Limitations and drawbacks

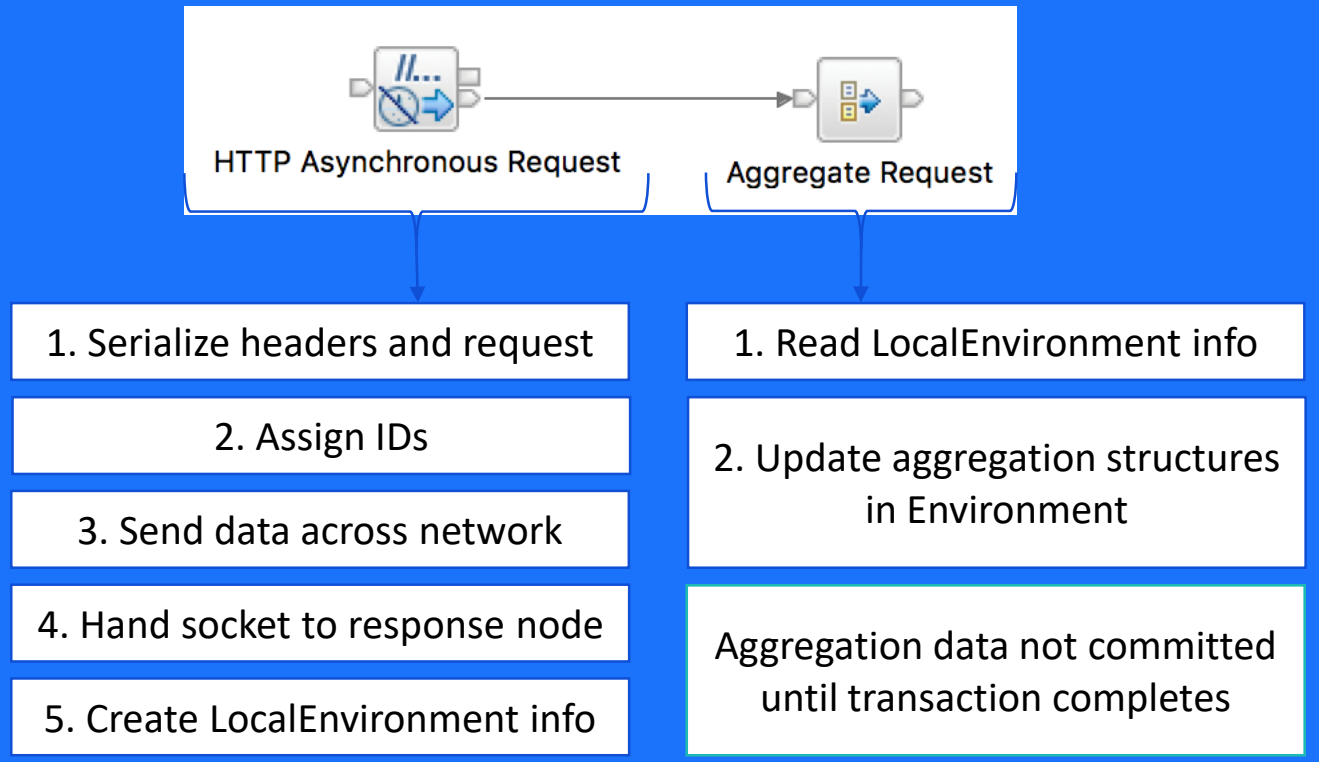
- MQ and persistent state
- Replies outrunning control data
- Unknown message handling
- Unpredictable message trees
- Queue flooding and cascade failures

Aggregation Nodes - Limitations and drawbacks

- **MQ and persistent state**
  - Aggregate nodes rely heavily on the local queue manager
  - Containers running aggregations require MQ
  - No support for remote queue managers
  - No recovery mechanism from queue manager failure
- Replies outrunning control data
- Unknown message handling
- Unpredictable message trees
- Queue flooding and cascade failures

Aggregation Nodes - Limitations and drawbacks

- MQ and persistent state
- **Replies outrunning control data**
- Unknown message handling
- Unpredictable message trees
- Queue flooding and cascade failures

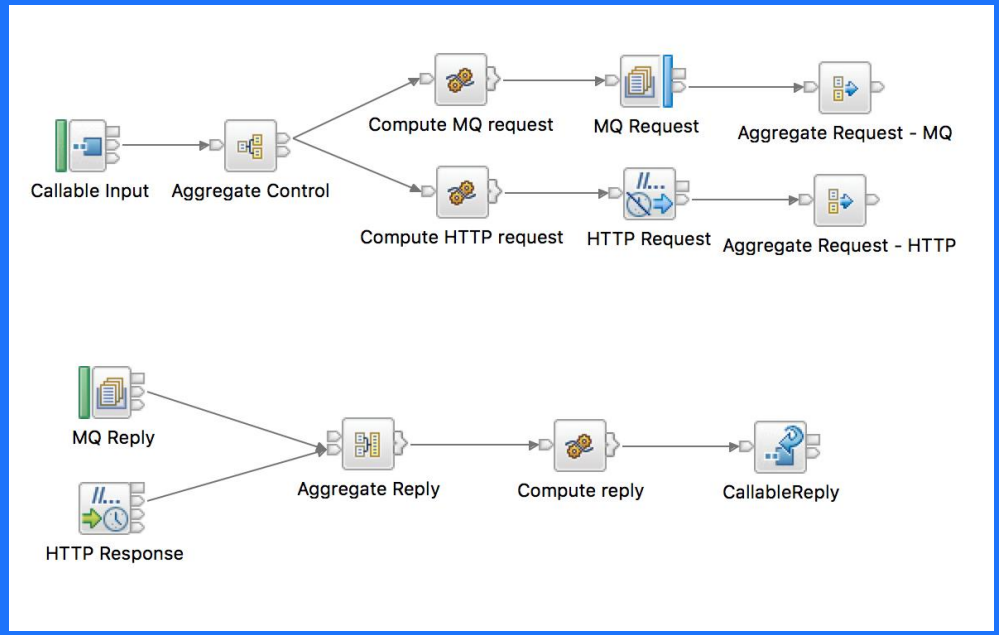


Aggregation Nodes - Limitations and drawbacks

- MQ and persistent state
- Replies outrunning control data
- **Unknown message handling**
  - “Unknown message timeout” attribute is unintuitive
  - **Expected**: Message is held for duration and will be matched against aggregations during that time.
  - **Reality**: Message is put onto an MQ queue with an expiration time. Message is only re-checked against aggregations after this expiry time.
  - Can cause aggregations to time out despite having received all replies
- Unpredictable message trees
- Queue flooding and cascade failures

Aggregation Nodes - Limitations and drawbacks

- MQ and persistent state
- Replies outrunning control data
- Unknown message handling
- **Unpredictable message trees**
  - Callable Input populates LocalEnvironment field with ReplyID
  - Callable Reply needs this ReplyID set
  - MQ nodes do not preserve ReplyIDs in LE
  - HTTP nodes do preserve ReplyIDs in LE
  - Aggregate Reply reuses same thread for the last reply, including LocalEnvironment and Environment trees, and transaction.
  - Last reply message is not serialized
- Queue flooding and cascade failures



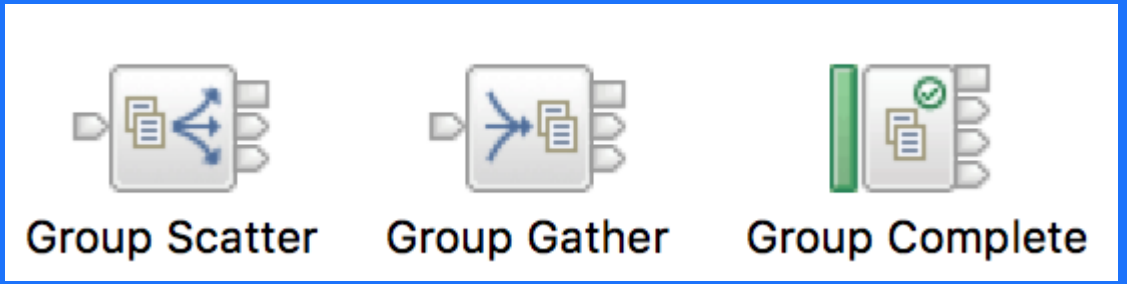
Aggregation Nodes - Limitations and drawbacks

- MQ and persistent state
- Replies outrunning control data
- Unknown message handling
- Unpredictable message trees
- **Queue flooding and cascade failures**
  - Full control queues prevent new aggregations
    - Also slow down timeout processing
  - Slow backend system → timeout processing becomes bottleneck
  - Flood of responses from a resurrected backend can flood replies queue
    - Floods the unknown message queue
  - Can cascade to delay legitimate replies being added to active aggregations
    - False timeouts occur
    - Even worse unknown message queue flooding

Is there another way?

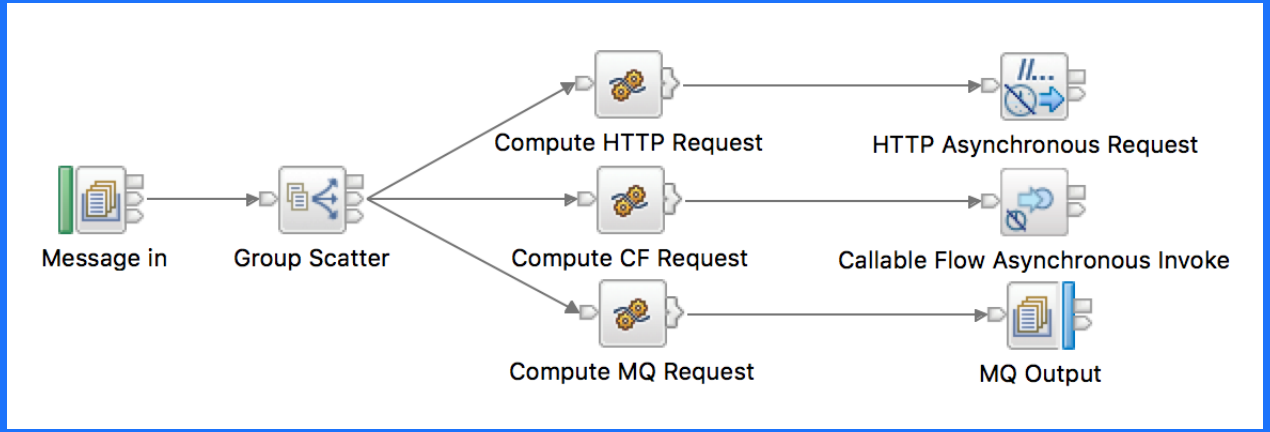
- Introducing the **Group nodes**

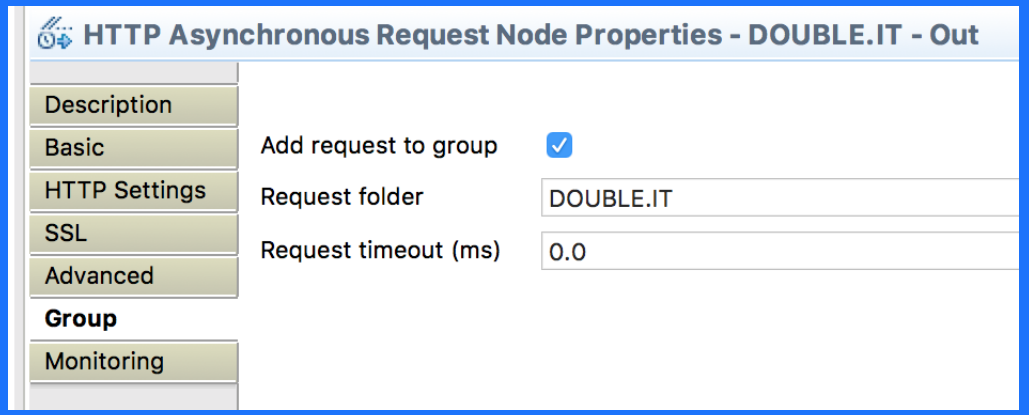
- Stateless alternative to Aggregate nodes
- Available in
  - IBM App Connect Enterprise
  - IBM App Connect Enterprise on Cloud
  - IBM Integration Bus V10 on Windows, LinuxX64, AIX, z/OS
- Aggregate nodes are not being deprecated



Classic aggregation scenario revisited

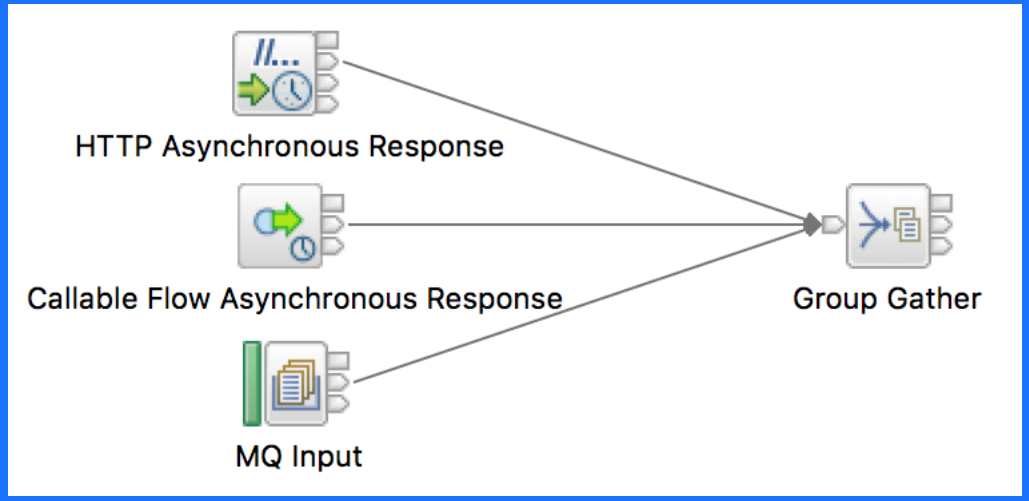
- Scatter flow
- **Group Scatter** node marks the start of a Group
- **Output nodes** record details of the sent requests directly
  - Opt-in via new node attributes
- No Aggregate Request-like node
- Group data stored in memory
- Group Created terminal
- Non-transactional





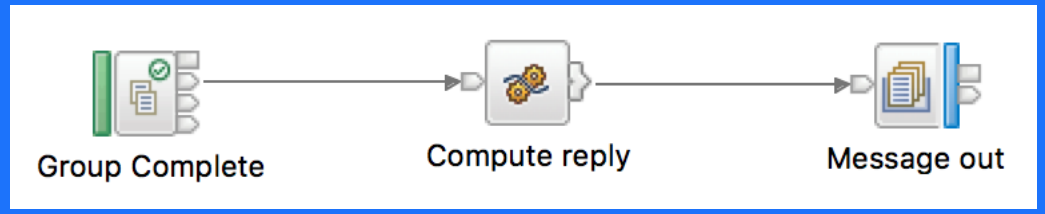
Classic aggregation scenario revisited

- Gather flow
- **Group Gather** node matches replies to Groups based on ReplyID/MessageID
- Unknown messages stored in memory
- Out and Unknown terminals sort replies
- Non-transactional
- Separates gather and complete logic



Classic aggregation scenario revisited

- Complete flow
- **Group Complete** node is a new “input” node for:
  - Completed Groups
  - Timed-out Groups
  - Timed-out unknown messages
- Intuitive timeout and unknown handling
- Timeout threads decoupled
- Predictable message trees
- New transaction per output



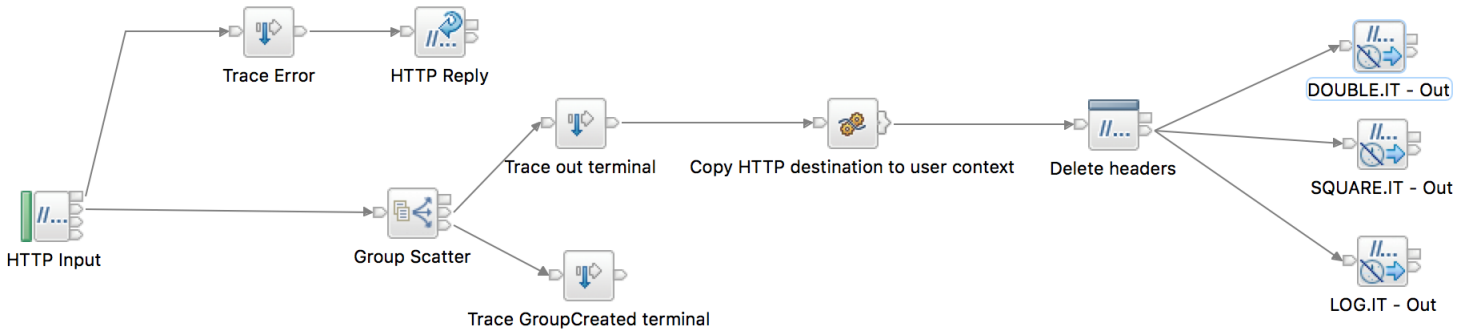
Aggregation vs Groups

- Qualitative comparison

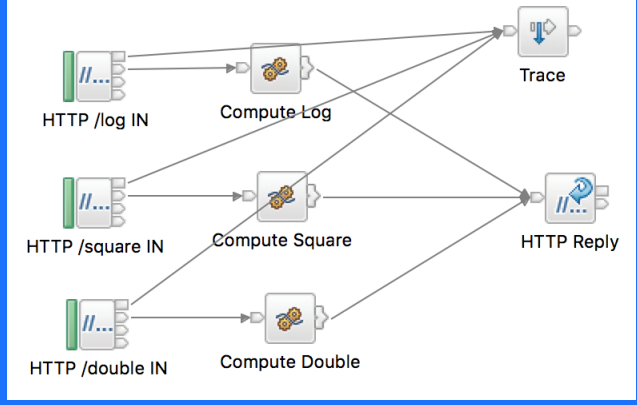
	Aggregate nodes	Group nodes
Control data stored	Using MQ queues	In EG process memory
Control data committed	On transaction commit	On unwind to GroupScatter node
Unknown message handling	Check one, wait, check again	Check periodically
Custom MsgId/ReplyId type	MQBYTE24	Any unique byte string
Platform availability	All platforms	AIX, z/OS, Linux x64, Windows
Message tree predictable?	No	Yes
Distributed processing across EGs?	Yes	No
Can be used in ACE on Cloud?	No	Yes

Group nodes + HTTP Async Demo

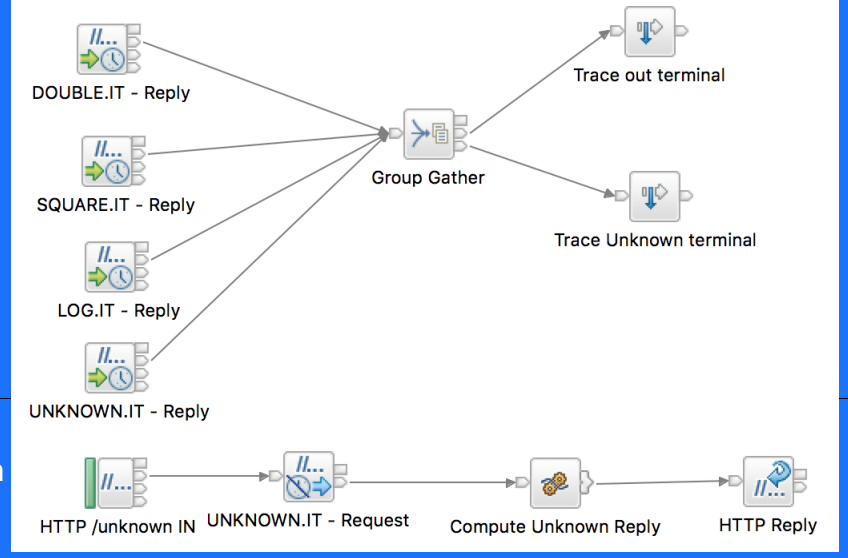
Scatter Flow



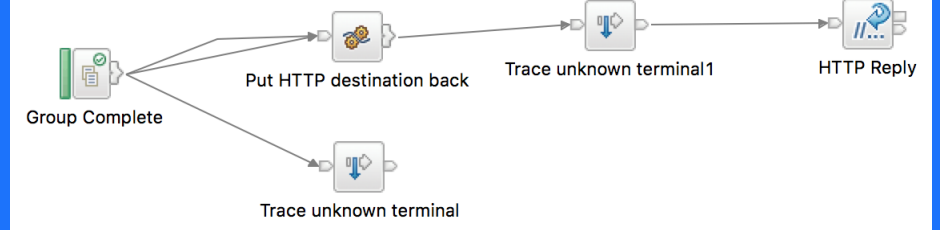
Processing Flow



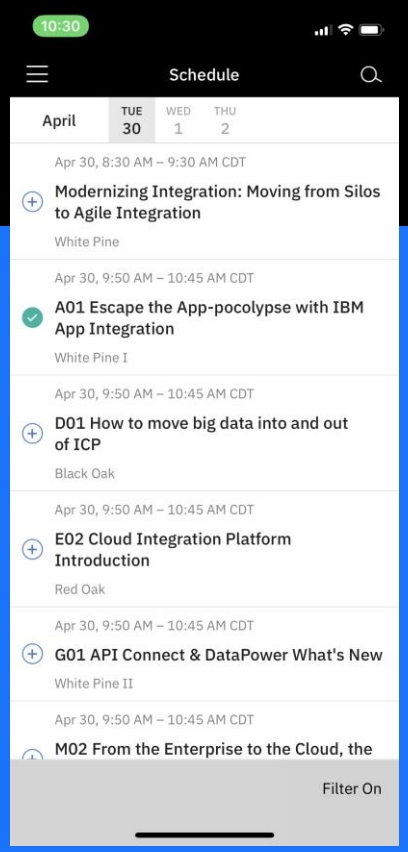
Gather Flow

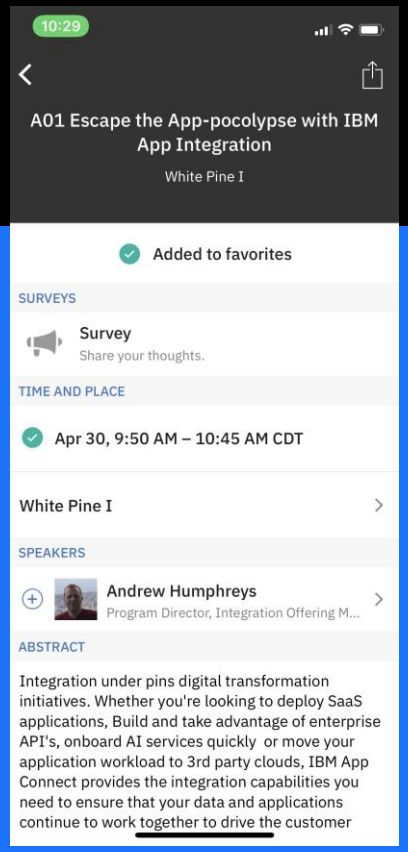


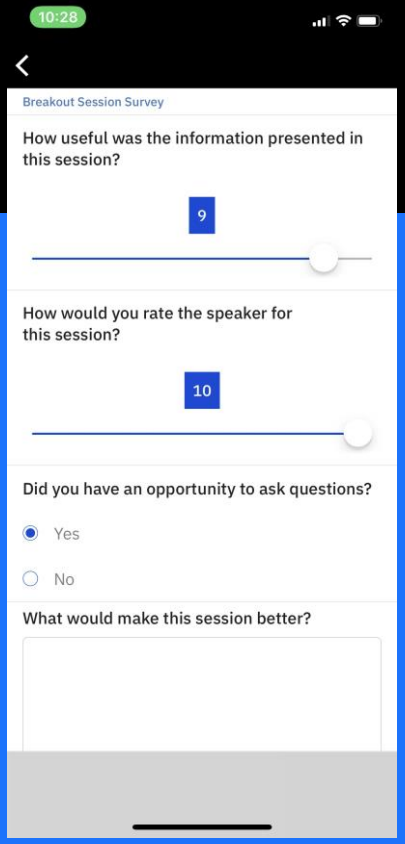
Complete Flow



Questions ?







IBM

Don’t forget to fill out the survey!

Select your session, select survey, rate the session and submit!

Thank You