Unit 4 - IP Communication Protocols

- HTTP
- AMQP
- MQTT
- STOMP
- protocol definitions
- use cases
- differences

Why set-up protocols?

- In order for proper communication to take place between the client and the server, these applications must agree on a specific application-level protocol

- Each communication protocol is made of rules and regulations clearly defined to form a shared language spoken between different application with the end result of being able to communicate regardless how they might be originally set to work.These protocols have elements such as data formats, definition of parties using the protocol, routings and flow (rate) control

HTTP - Hypertext Transfer Protocol

- Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext
- It is an RESTful service application layer protocol for distributed, collaborative, and hypermedia information systems
- HTTP is the foundation of data communication for the World Wide Web with development initiated by Tim Berners-Lee at CERN in 1989
- HTTP specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these request
- HTTP exchanges happen between -
    - Client - The HTTP client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a TCP/IP connection
    - Server - The HTTP server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta information, and possible entity-body content

HTTP - Features

- HTTP is an asymmetric request-response client-server protocol - An HTTP client sends a request message to an HTTP server and the server, in turn, returns a response message
- HTTP is a pull protocol - the client pulls information from the server
- HTTP is connectionless - The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send a response back.
- HTTP is media independent - any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type, a standardized way to indicate the nature and format of a document
- HTTP is stateless - HTTP is connectionless as a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

HTTP - URI and Messages

- Uniform Resource Identifiers (URI) are simply formatted, case-insensitive string containing name, location, etc. to identify a resource, for example, a website, a web service, etc. A general syntax of URI used for HTTP is as follows:
  - URI = “http: // “<host> ":" <port> <abs_path> "?" <query>
  - if the port is empty or not given, port 80 is assumed for HTTP and an empty abs_path is equivalent to an abs_path of "/".
- HTTP makes use of the URI to identify a given resource and to establish a connection. Once the connection is established, HTTP messages are passed in a Multipurpose Internet Mail Extension (MIME) format
- HTTP requests and HTTP responses use a generic message format -
  - A Start-line - is a Request-Line by the client and a Status-Line by the server
  - Zero or more header fields followed by CRLF
  - An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields
  - Optionally a message-body - carries request or response data

HTTP - Headers

- HTTP header fields provide required information about the request or response, or about the object sent in the message body
- There are four types of HTTP message headers:
  - General-header: These header fields have general applicability for both request and response messages.
  - Request-header: These header fields have applicability only for request messages.
  - Response-header: These header fields have applicability only for response messages.
  - Entity-header: These header fields define meta information about the entity-body or, if no body is present, about the resource identified by the request.
- The above mentioned headers follow the same generic format and each of the header field consists of a name followed by a colon (:) and the field value as follows:
  - message-header = <field-name> : <field-value>

HTTP - Request Methods

- GET - is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
- HEAD - Same as GET, but it transfers the status line and the header section only
- POST - A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms
- PUT - Replaces all the current representations of the target resource with the uploaded content
- DELETE - Removes all the current representations of the target resource given by URI
- CONNECT - Establishes a tunnel to the server identified by a given URI
- OPTIONS - Describe the communication options for the target resource
- TRACE - Performs a message loop back test along with the path to the target resource

The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers.

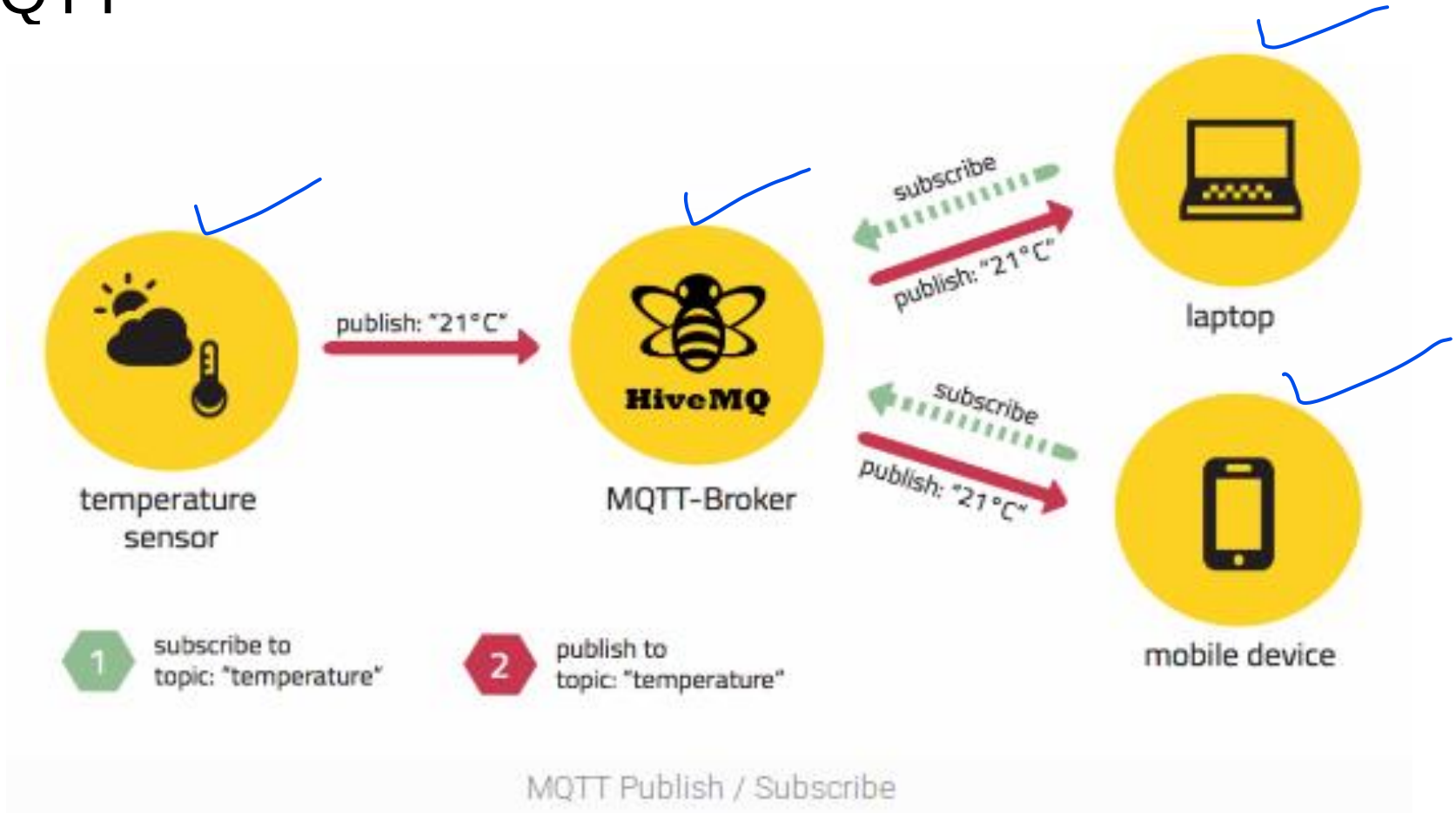
HTTP - Responses

- A Status-Line consists of the protocol version followed by a numeric status code and its associated textual phrase
- Types of Status Codes -
  - 1xx: Informational - It means the request was received and the process is continuing
  - 2xx: Success - It means the action was successfully received, understood, and accepted
  - 3xx: Redirection - It means further action must be taken in order to complete the request
  - 4xx: Client Error - It means the request contains incorrect syntax or cannot be fulfilled
  - 5xx: Server Error - It means the server failed to fulfill an apparently valid request
- The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status- Line. These header fields give information about the server and about further access to the resource identified by the Request-URI

MQTT - Message Queue Telemetry Transport*

- MQTT is a machine-to-machine (M2M) connectivity session layer, binary protocol. It was designed as an extremely lightweight publish/subscribe messaging transport
- It is useful for connections with remote locations where a small code footprint (topic) is required and/or network bandwidth is at a premium. It is also ideal for mobile applications because of its small size, low power usage, minimised data packets, and efficient distribution of information to one or many receivers
- MQTT utilizes many characteristics of the TCP transport, so the minimum requirement for using MQTT is a working TCP stack, which is now available for even the smallest microcontrollers
- MQTT constituents :
  - Publisher - sends the message over MQTT
  - Subscriber - receives messages over MQTT
  - MQTT broker - decouples Pub/Sub pair, meaning the publisher and subscriber don’t know about the existence of one another but are aware of the broker

MQTT


subscribe
publish: "21°C"
laptop
publish: "21°C"
HiveMQ
temperature sensor
MQTT-Broker
subscribe
publish: "21°C"
mobile device
1
subscribe to topic: "temperature"
2
publish to topic: "temperature"
MQTT Publish / Subscribe

MQTT

- The central communication point is the MQTT broker, it is in charge of dispatching all messages between the senders and the rightful receivers. Each client that publishes a message to the broker, includes a topic into the message. The topic is the routing information for the broker. Each client that wants to receive messages subscribes to a certain topic and the broker delivers all messages with the matching topic to the client
- A topic is a simple string that can have more hierarchy levels, which are separated by a forward slash (topic level separator). A sample topic for sending temperature data of the living room could be house/living-room/temperature. On one hand the client can subscribe to the exact topic or on the other hand use a wildcard (cannot be used on the sender end). The subscription to house/+/temperature would result in subscribing to any topic with an arbitrary value in the place of living room, for example house/kitchen/temperature. The plus sign is a single level wildcard and only allows arbitrary values for one hierarchy. If you need to subscribe to more than one level, for example to the entire subtree, there is also a multilevel wildcard (#). It allows to subscribe to all underlying hierarchy levels. For example house/# is subscribing to all topics beginning with house

MQTT

- MQTT does not implement message queues :
  - If a message is not consumed, MQTT broker might drop or deal with it based on the implementation unlike in traditional message queues where messages are stored until consumed
  - MQTT allows for multiple subscribers to receive with the same topic; Message queues allows for only one consumer to take in the message
  - Queues are inflexible - they HAVE to be created and named. MQTT topics just have to be declared and no pipelines to be explicitly created
- The difference to HTTP is that a client doesn't have to pull the information it needs, but the broker pushes the information to the client, in the case there is something new. Therefore each MQTT client has a permanently open TCP connection to the broker. If this connection is interrupted by any circumstances, the MQTT broker can buffer all messages and send them to the client when it is back online

MQTT - Extra

- Decoupling by :
  - Space - Publisher and subscriber do not need to know each other (by ip address and port for example)
  - Time - Publisher and subscriber do not need to run at the same time
  - Synchronisation - Operations on both components are not halted during publish or receiving
- Message filtering by :
  - Subject/topic - The receiving client subscribes on the topics it is interested in with the broker and from there on it gets all message based on the subscribed topics
  - Type - When using object-oriented languages it is a common practice to filter based on the type/class of the message
  - Content - the broker filters the message based on a specific content filter-language. A big downside to this is, that the content of the message must be known beforehand and can not be encrypted or changed easily

MQTT - Message Types

- Any exchange of messages between broker and client is always one of the 14 message types listed

- Most common are -
  - Connect
  - Publish
  - Subscribe
  - Unsubscribe

- The other message types are mostly for internal mechanisms and flows

MESSAGE TYPE	DESCRIPTION
CONNECT	Client request to connect to Server
CONNACK	Connection Acknowledgement
PUBLISH	A message which represents a new/separate publish
PUBACK	QoS 1 Response to a PUBLISH message
PUBREC	First part of QoS 2 message flow
PUBREL	Second part of QoS 2 message flow
PUBCOMP	Last part of the QoS 2 message flow
SUBSCRIBE	A message used by clients to subscribe to specific topics
SUBACK	Acknowledgement of a SUBSCRIBE message
UNSUBSCRIBE	A message used by clients to unsubscribe from specific topics
UNSUBACK	Acknowledgement of an UNSUBSCRIBE message
PINGREQ	Heartbeat message
PINGRESP	Heartbeat message acknowledgement
DISCONNECT	Graceful disconnect message sent by clients before disconnecting.

MQTT - Advantages

- Ultra-lightweight or Bandwidth efficient
- Massively scalable - no dependency between producer and consumer because of the use of topics and clustered brokers
- Easy-to-implement protocol
- Open and Simple
- Requires very little battery power
- Minimal packet overhead
- Continuous Session Awareness
- Reliable message delivery - MQTT offers 3 levels :
  - 0 - At most once delivery; No retransmission, only best effort
  - 1 - At least once delivery; sent until acknowledgement is received
  - 2 - Exactly once delivery; sent once secure connection is established but creates extra overhead

MQTT - Use cases and Common Brokers

Typical use cases of MQTT include:

- Telemetry
- Automotive
- Smart Home
- Energy Monitoring
- Chat Applications and Notification Services
- Healthcare Applications

Common MQTT Brokers are (Client libraries and tools are separate) -

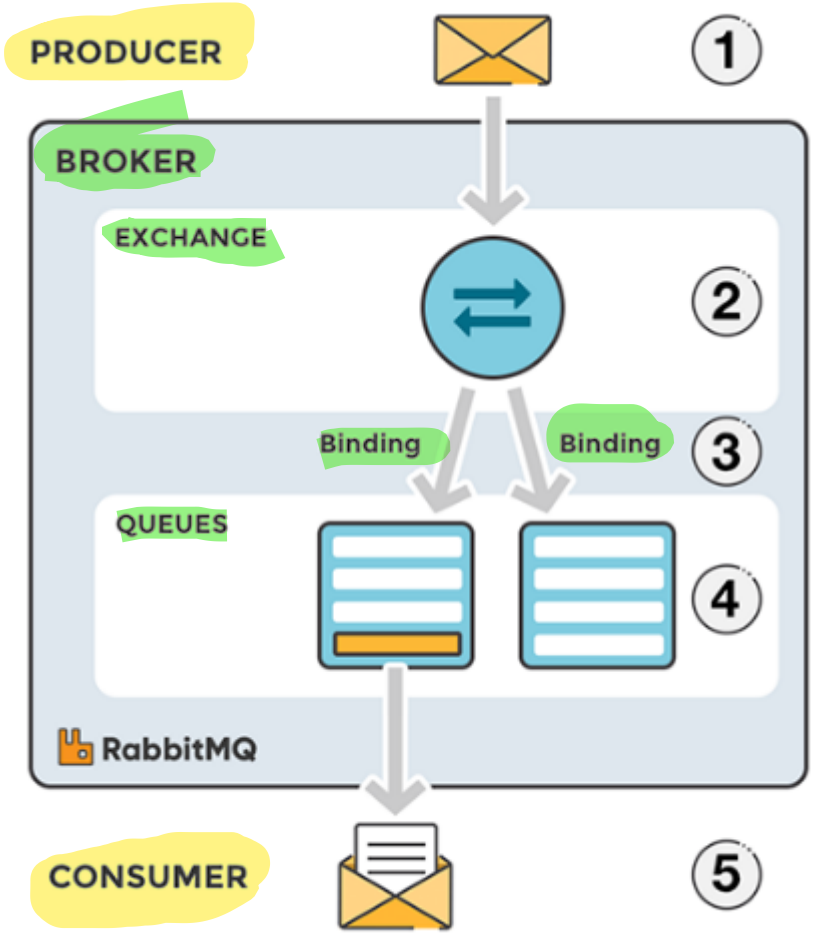
- mosquitto
- HiveMQ
- RabbitMQ
- Apache ActiveMQ
- Mosca
- RSMB
- IBM MQ

AMQP - Advanced Message Queuing Protocol

- It's an open standard and open source designed to allow development of applications tailored to work as middleware in brokering of messages between different processes, applications and unrelated systems that need to talk to each other and pass on messages
- AMQP has 3 main components :
  - Exchanges - Messages to be sent are first published to exchanges
  - Routes - define on which queue(s) to pipe the message. Routes are ensured by an AMQP broker that matches routing and binding keys. Brokers can be imagined to be a part of the exchange.
  - Consumers - subscribe to relevant queue(s) to receive a copy of the message
- Routing and binding keys are often mistaken as the same, but have two different functionalities :
  - Exchanges are linked to queues prior to sending messages through binding keys
  - Messages carry with them routing keys which help exchanges decide which queue to forward the message to, i.e, generally, exchanges send messages to those queues whose routing and binding keys match

AMQP - Flow

1. The producer publishes a message to the exchange
2. The exchange receives the message and is now responsible for the routing of the message
3. A binding has to be set up between the queue and the exchange. In this case, we have bindings to two different queues from the exchange. The exchange routes the message into the queues
4. The messages stay in the queue until they are handled by a consumer. If more than one consumer subscribes to the same queue, the messages are dispensed in a round-robin fashion
5. The consumer handles the message.



AMQP - Advanced Message Queuing Protocol

- Exchanges can have attributes such as :
  - Name
  - Durability - Durable exchanges survive broker restart whereas transient exchanges do not and have to be redeclared
  - Auto-delete - if exchange is to be deleted when last queue is unbound from it
  - Arguments - optional, used by plugins and broker-specific features
- A queue can establish connections on both side; on the input side a queue fetches messages from one or more exchanges while on the output side the queue can be connected to one or more consumers
- Before a queue can be used it has to be declared. Declaring a queue will cause it to be created if it does not already exist. The declaration will have no effect if the queue does already exist and its attributes are the same as those in the declaration. When the existing queue attributes are not the same as those in the declaration a channel-level exception with code 406 (PRECONDITION_FAILED) will be raised
- Queue names starting with "amq." are reserved for internal use by the broker. Attempts to declare a queue with a name that violates this rule will result in a channel-level exception with reply code 403 (ACCESS_REFUSED)

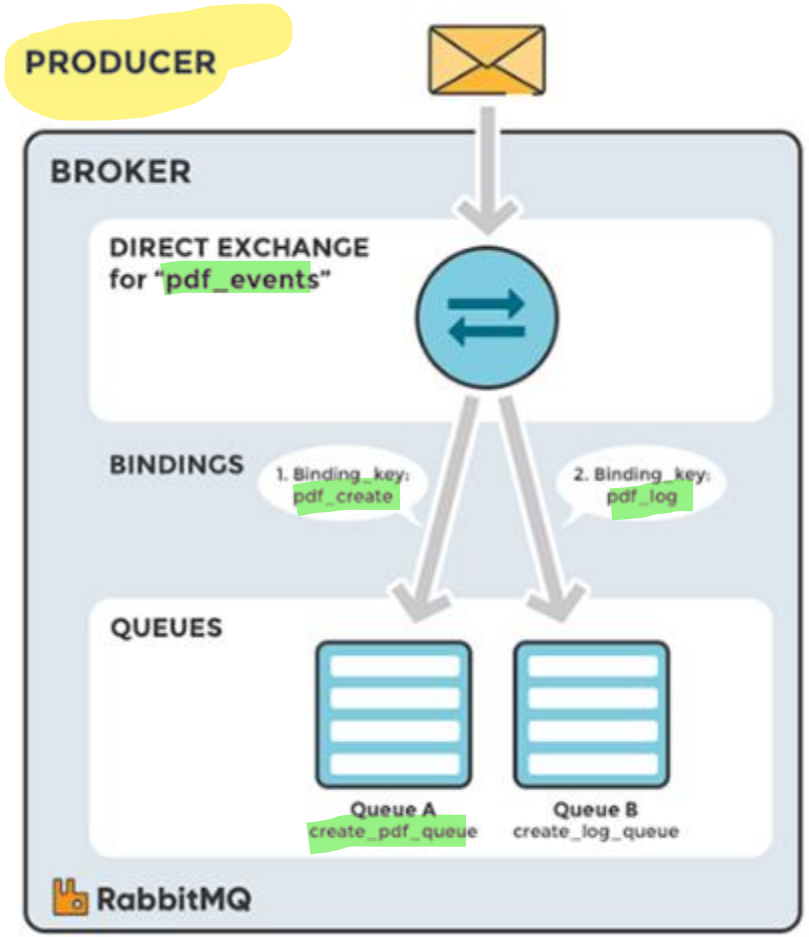
More on AMQP

- A sender always publishes to an exchange, a receiver can only subscribe to a queue - In AMQP, a message is published to an exchange, which through its bindings, the routing technique followed in AMQP, may get sent to one queue or multiple queues. Messages are received on the other end only through subscriptions to queues
- AMQP lives in the Application Layer of an OSI Model and is a programmable protocol
- AMQP provides message header, providing the means to sort and route messages. Messages essentially carry routing keys in these headers
- AMQP brokers are responsible for receiving, routing, and ultimately dispensing messages to consumers
- Dead-Letter-Queue, if implemented, is where all messages that cannot be routed successfully are dropped off

AMQP - Exchange Types

Direct Exchange

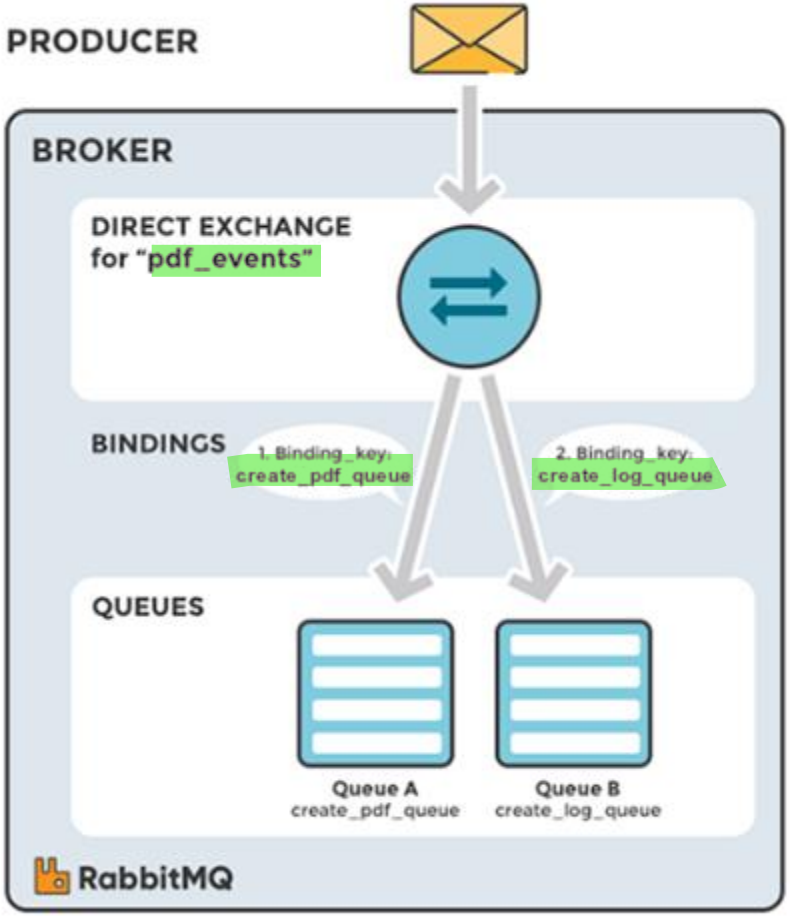
- A direct exchange delivers messages to queues based on a message routing key
- The message goes to the queue(s) whose binding key exactly matches the routing key of the message :
  - When a new message with routing key ‘pdf_create’ arrives at the direct exchange, the exchange routes it to the queue ‘create_pdf_queue’
- The default exchange AMQP brokers must provide for the direct exchange is "amq.direct"
- A direct exchange is ideal for the unicast routing of messages (although they can be used for multicast routing as well)
- Direct exchanges are often used to distribute tasks between multiple workers in a round robin manner



AMQP - Exchange Types

Default Exchange

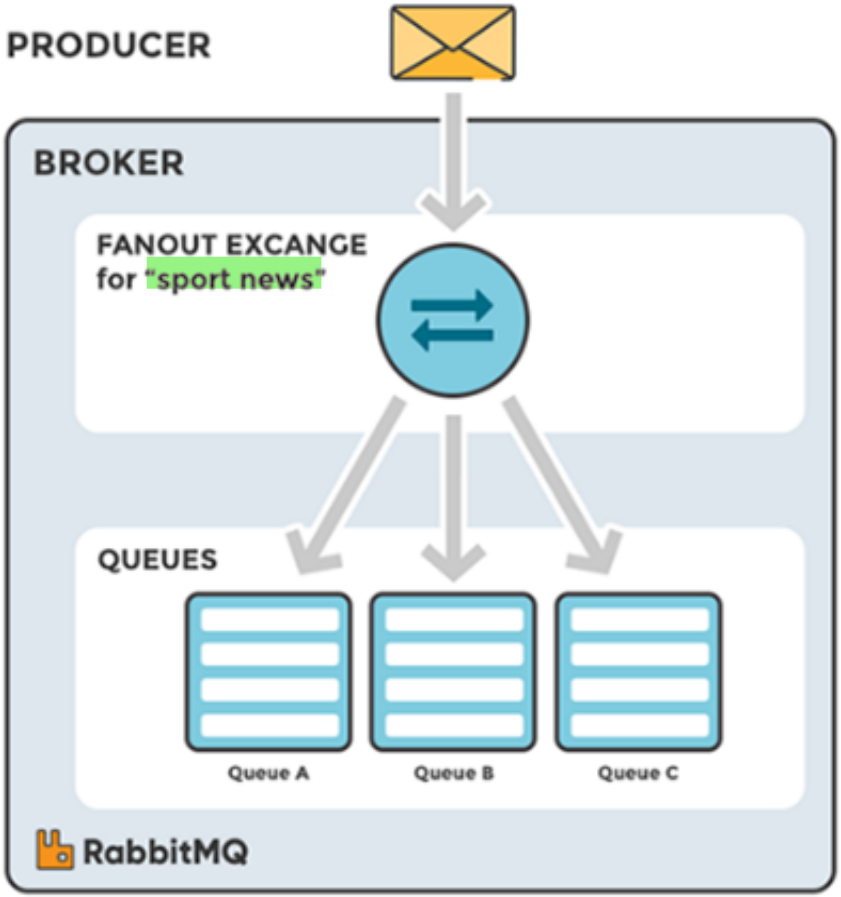
- The default exchange is a direct exchange with no name (empty string) pre-declared by the broker.
- Every queue that is created is automatically bound to it with a routing key which is the same as the queue name
    - For the diagram shown, if the broker declares default exchange, the binding key for queue ‘create_pdf_queue’ is ‘create_pdf_queue’ and for a message to be queued into this should also contain the same routing key
- Default exchanges are used only for simple applications where authentication and security are not vital



AMQP - Exchange Types

Fanout Exchange

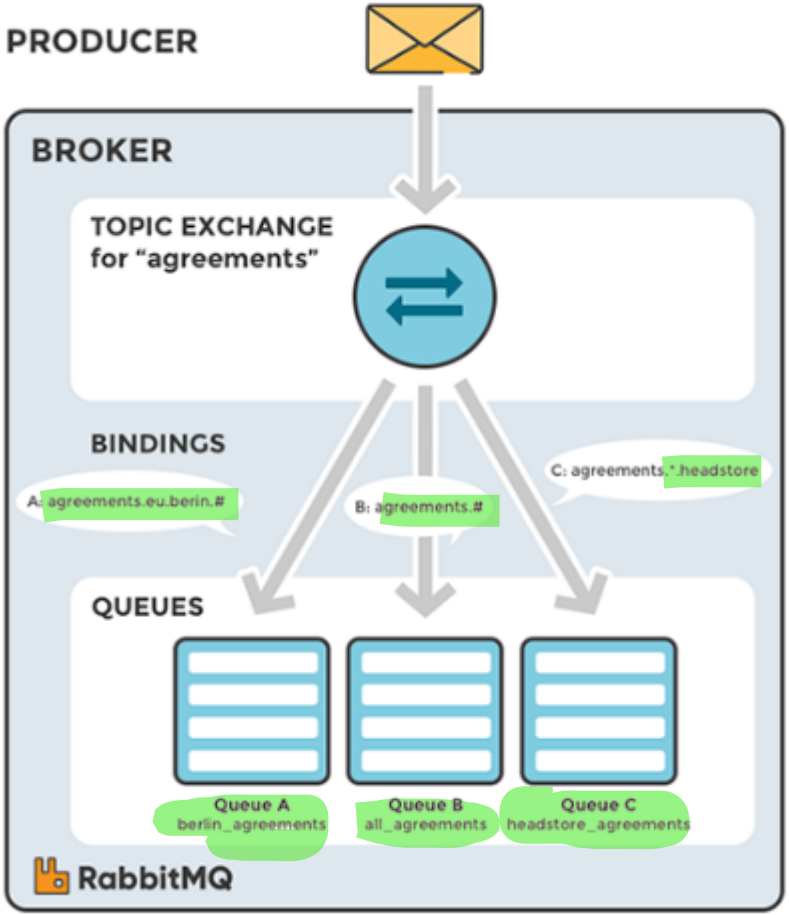
- Fanout exchange copies and routes a received message to all queues that are bound to it regardless of routing keys. Keys provided will simply be ignored.
- Fanout exchanges can be useful when the same message needs to be sent to one or more queues with consumers who may process the same message in different ways
- Fanout exchange is employed in broadcasting messages e.g. games can use it for leaderboard updates
- The default exchange AMQP brokers must provide for the topic exchange is "amq.fanout"



AMQP - Exchange Types

Topic Exchange

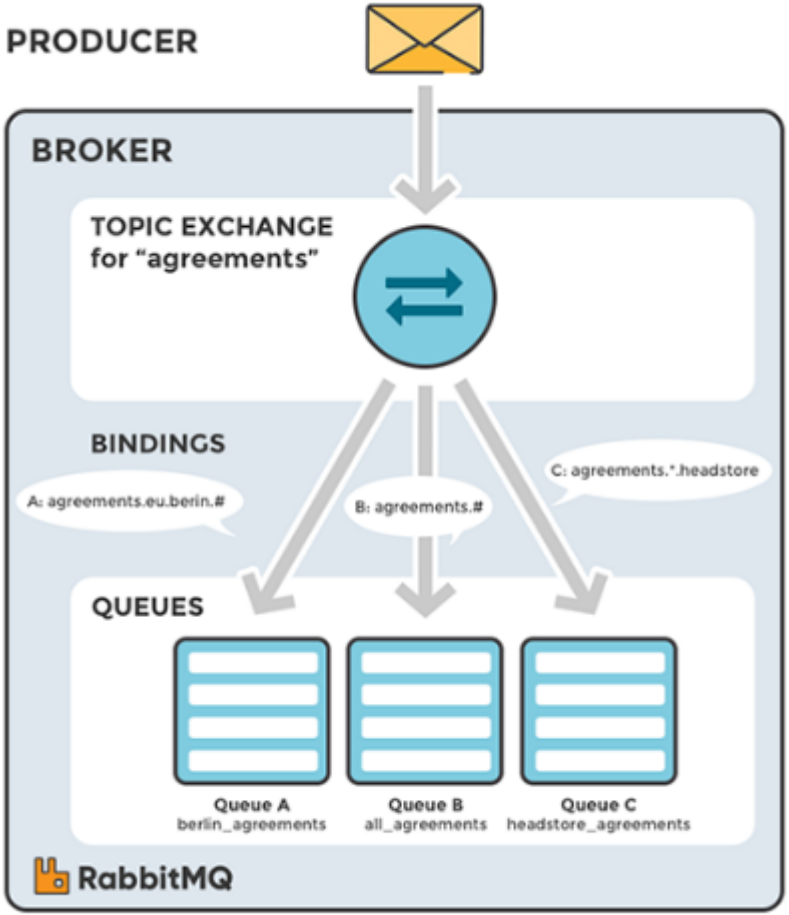
- Topic exchanges route messages to queues based on wildcard matches between the routing key and routing pattern specified by the queue binding. Messages are routed to one or many queues based on a matching between a message routing key and this pattern
- The consumers indicate which topics they are interested in. The consumer creates a queue and sets up a binding with a given routing pattern to the exchange. All messages with a routing key that match the pattern will be routed to the queue and stay there until consumed. e.g. Stock price updates
- The default exchange AMQP brokers must provide for the topic exchange is "amq.topic"



AMQP - Exchange Types

Topic Exchange

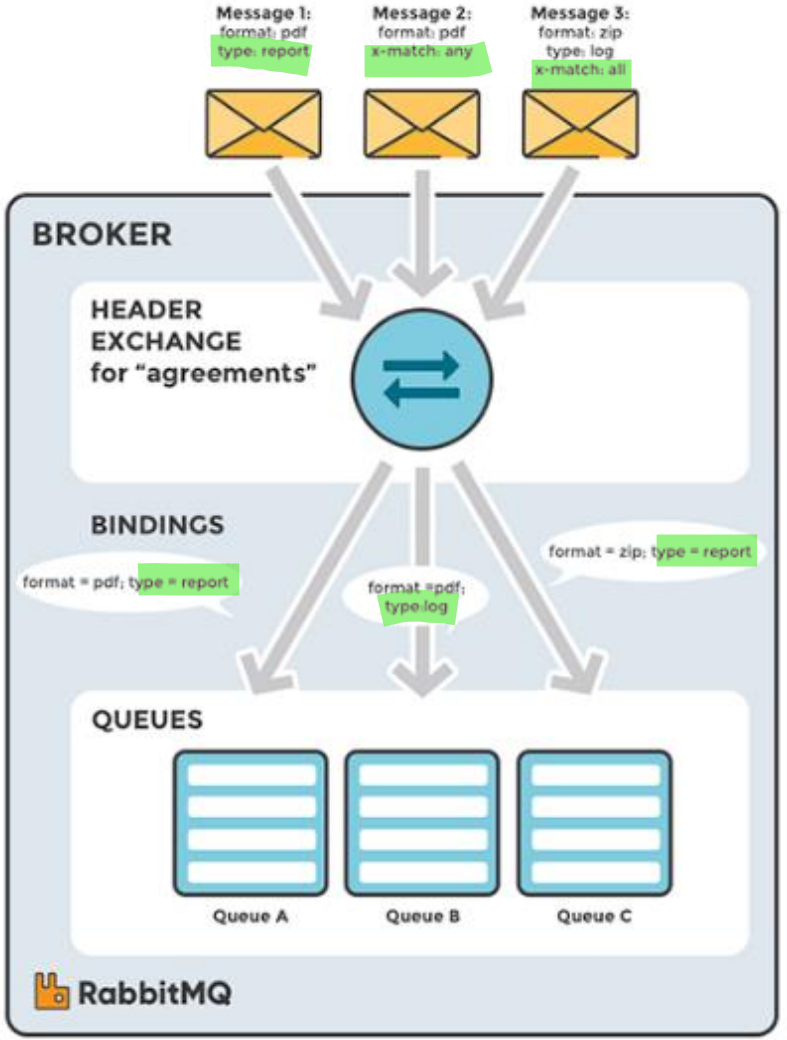
- The routing key must be a list of words, delimited by a period (.), examples are *agreements.us* and *agreements.eu.stockholm* which in this case identifies agreements that are set up for a company with offices in lots of different locations.
- The routing patterns may contain an asterisk (“*”) to match a word in a specific position of the routing key e.g. a routing pattern of "agreements.*.*.b.*" will only match routing keys where the first word is "agreements" and the fourth word is "b"
- A pound symbol (“#”) indicates match on zero or more words, e.g. a routing pattern of "agreements.eu.berlin.#" matches any routing keys beginning with "agreements.eu.berlin"



AMQP - Exchange Types

Headers Exchange

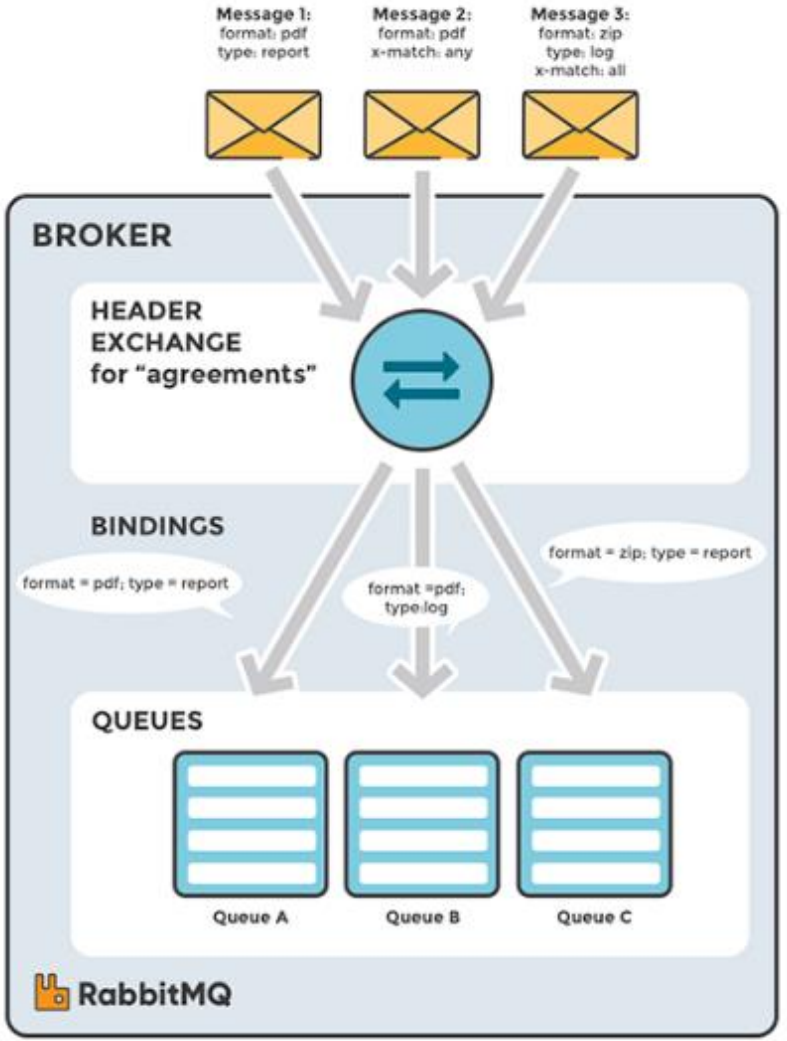
- A headers exchange is designed for routing on multiple attributes that are more easily expressed as message headers than a routing key
- Headers exchanges ignore the routing key attribute. Instead, the attributes used for routing are taken from the headers attribute
- A message is considered matching if the value of the header of the message equals the value specified upon binding
- The default exchange AMQP brokers must provide for the topic exchange is "amq.match"



AMQP - Exchange Types

Headers Exchange

- A special argument named "x-match" tells if all headers must match or just one. The "x-match" property can the have two different values: "any" or "all", where "all" is the default value
- A value of "all" means all header pairs (key, value) must match and a value of "any" means at least one of the header pairs must match
- The headers exchange type used with "any" is useful for directing messages which may contain a subset of known (unordered) criteria
- Headers can be constructed using a wider range of data types - integer or hash table for example instead of a string



AMQP - Advantages

- Wire specification for asynchronous messaging where every byte of transmitted data is specified. This characteristic allows libraries to be written in many languages, and to run on multiple operating systems and CPU architectures, which makes for a truly interoperable, cross-platform messaging standard
- AMQP publishes its specifications in a downloadable XML format. This availability makes it easy for library maintainers to generate APIs driven by the specs while also automating construction of algorithms to marshal and de-marshal messages
- AMQP, by offering the clearly defined rules and instructions, creates a common ground which can be used for all message queuing and brokering applications to work and interoperate
- It achieves its goal of enabling a wide range of different applications and systems to be able to work together, regardless of their internal designs, standardizing enterprise messaging on industrial scale

AMQP - Advanced Message Queuing Protocol

- **AMQP ensures**
  - Reliability of message deliveries
  - Rapid and ensured delivery of messages
  - Deletion of a message from a queue only after acknowledgements
- **These capabilities make it ideal for**
  - Monitoring and globally sharing updates
  - Connecting different systems to talks to each other
  - Allowing servers to respond to immediate requests quickly and delegate time consuming tasks for later processing
  - Distributing a message to multiple recipients for consumption
  - Enabling offline clients to fetch data at a later time
  - Introducing fully asynchronous functionality for systems
  - Increasing reliability and uptime of application deployments

STOMP - Simple/Streaming Text Orientated Messaging Protocol

- STOMP is a simple interoperable protocol designed for asynchronous message passing between clients via mediating servers. It defines a text based wire-format for messages passed between these clients and servers
- A STOMP server is modelled as a set of destinations to which messages can be sent. The STOMP protocol treats destinations as opaque strings and their syntax is server implementation specific
- Since the protocol is text-based, it is analogous to HTTP in terms of how it looks under the covers
- Like AMQP, STOMP provides a message (or frame) header with properties, and a frame body while aiming for interoperability. The frames are modelled on HTTP and consists of a command, a set of optional headers and an optional body
- STOMP is easy to implement and gives you flexibility since it is language-agnostic, meaning clients and brokers developed in different languages can send and receive messages to and from each other

STOMP - Simple/Streaming Text Orientated Messaging Protocol

- It distinguishes itself by covering a small subset of commonly used messaging operations rather than providing a comprehensive messaging API
- Communication between server and client is through a MESSAGE, RECEIPT or ERROR frame with a similar format of headers and body content
- The message commands within the frames are similar to HTTP and works over TCP. Few of them are - CONNECT, SEND, SUBSCRIBE, UNSUBSCRIBE, BEGIN, COMMIT, ABORT, ACK, NACK, DISCONNECT
- The delivery, or “message exchange”, semantics of destinations can vary from server to server and even from destination to destination. This allows servers to be creative with the semantics that they can support with STOMP
- A STOMP client is a user-agent which can act in two (possibly simultaneous) modes:
  - as a producer, sending messages to a destination on the server via a SEND frame
  - as a consumer, sending a SUBSCRIBE frame for a given destination and receiving messages from the server as MESSAGE frames

Differences between MQTT and HTTP

MQTT

- Publish /Subscribe (push) protocol
- Binary protocol
- Asynchronous - does not block the client while it waits for the message
- Requires lesser bandwidth and battery power
- Quality of Service
- Inherently, has no security only reliability

HTTP

- Request/Response (pull) protocol
- Plain text protocol
- Synchronous
- Requires more bandwidth
- Consumes more battery power
- No Quality of Service
- Can promise security

Differences between MQTT and ~~HTTP~~ AMQP

MQTT

- Lightweight because topics create minimal overhead
- Uses MQTT brokers for decoupling the publisher and the subscriber
- Messages sent to topics that have not been subscribed by any client is typically discarded
- Any number of subscribers can receive the same message as long as the topic matches
- Considered as a basic messaging protocol
- Vendor-driven

AMQP

- Message queue creation and management adds to the overhead
- A exchange-queue pair is employed for decoupling
- Messages remain in the queues until consumed by a subscriber
- Only one subscriber can consume a sent message (however, copies of the message can be sent by round-robin)
- AMQP is considered as a superset of MQTT, with additional functionalities such as secured transmission
- Customer-driven