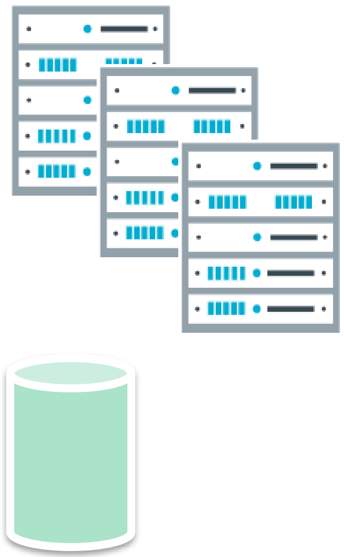
MQ Hybrid Cloud Architectures

Matthew Whitehead
IBM MQ Development
mwhitehead@uk.ibm.com

Agenda

- Topologies
- Connectivity
- Clients & Applications

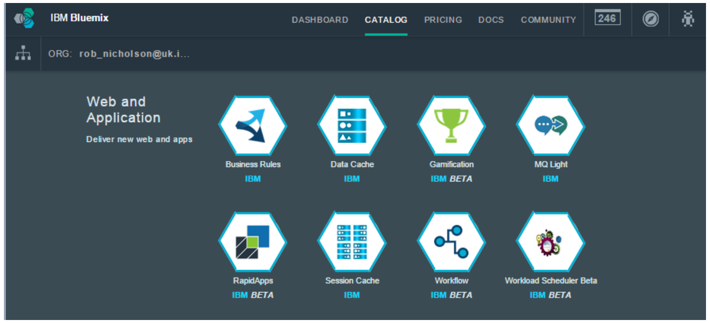
IBM Cloud Hybrid Messaging – Joining the 2 worlds together



- Systems of record
- Enterprise data
- 24 x 7 x 365 applications

On Premise

- Systems of engagement
- Mobile
- Social
- Analytics & Watson
- Rapid development



IBM Bluemix

DASHBOARD CATALOG PRICING DOCS COMMUNITY 246

ORG: rob_nicholson@uk.i...

Web and Application

Deliver new web and apps

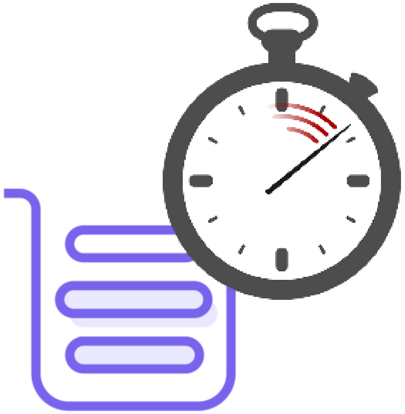
Business Rules IBM	Data Cache IBM	Gamification IBM BETA	MQ Light IBM
RapidApps IBM BETA	Session Cache IBM	Workflow IBM BETA	Workload Scheduler Beta IBM BETA

IBM Cloud

But first – parish notices

- **MQ on Cloud** service now available
- Your queue managers running in the IBM Cloud
- Hosted solution removes hassle of platform and OS maintenance from you
- See session on Wednesday (1pm) this week

- **Hourly Container-Based Pricing** now available
- Requires you to run your queue managers in containers
- Uses IBM Cloud Private (ICP) metering solution to track usage



Why Hybrid Messaging?

“All the benefits of cloud, with access to your enterprise data”

- Doing more with less
- Being more ready to change
- Making the development process less heavyweight
- Paying for what you use
- Integrating with other cloud services
- Rapidly scaling up and down with demand

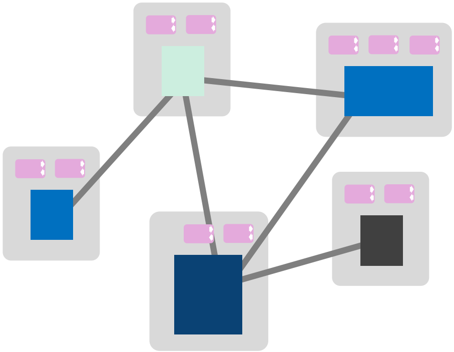
- Customer profiles
- Purchases (online orders)
- Data requests (e.g. insurance quotes)
- Website comments

Why Hybrid Messaging?

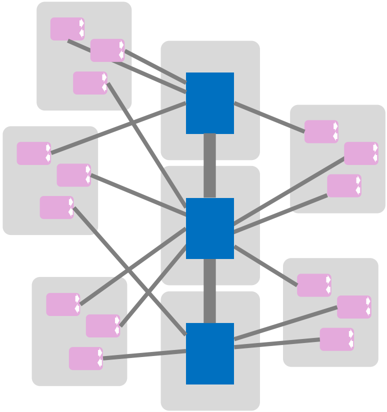
Is it to…

- run you apps, unchanged, in a cheaper environment?
- stage the migration of applications to cloud-native runtimes?
- move to micro-services model?
- enable developers?
- be able to say you’re “in the cloud”?

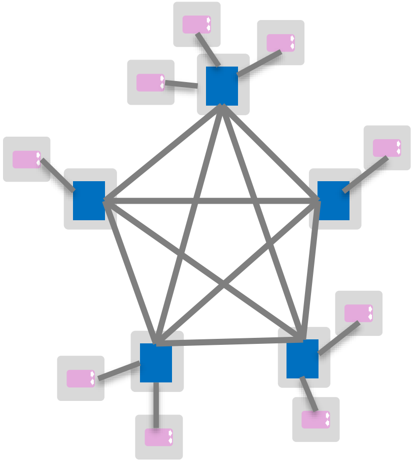
Typical MQ Architectures



Classic



Hub



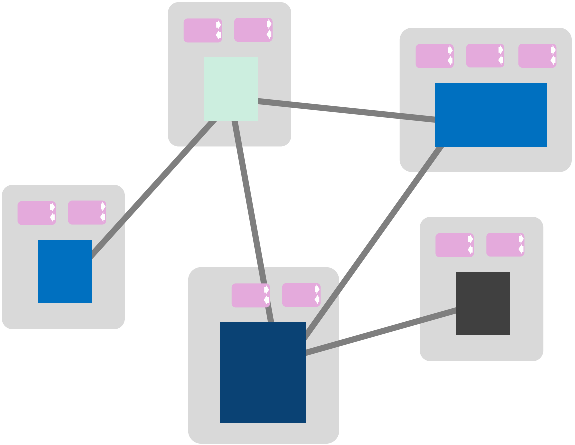
Decentralized

More suited to cloud scenarios

The Classic

Used for connectivity of heterogeneous systems, providing store and forward to overcome system and network outages

Isolation through dedicated queue managers, tightly bound to the application runtimes



This is one of the '*original*' deployment patterns for MQ and has often ended up as bespoke, tuned deployments for individual components

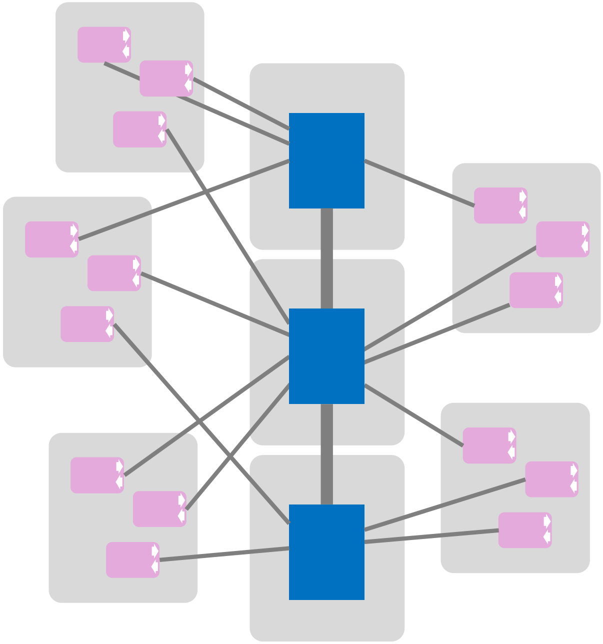
Leads to hard to deploy, manage and maintain systems over time

The Hub

A ‘*hub*’ (or backbone) of systems running multiple queue managers, based on a standard deployment

Applications connecting as clients from remote systems. Loser coupling enables simpler deployments and independent scaling and maintenance

This pattern has gained popularity as networks improve and administration costs go up

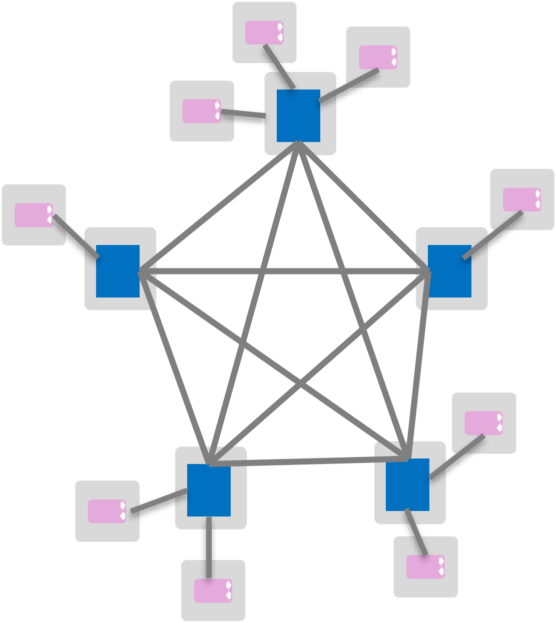


Decentralised

Decentralise the MQ system completely. Each line of business or application has its own infrastructure and therefore own queue managers. Client connections to separate applications from the infrastructure

Remove the central administration as much as possible to reduce bureaucracy and speed up application deployments

Has popularity as a way to satisfy greater autonomy for lines of business



Queue Managers: Pets or Cattle?

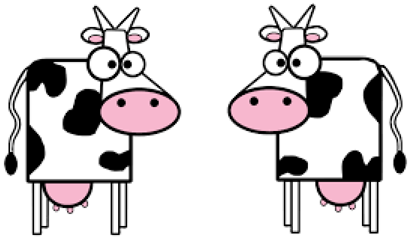
It's best practice to adopt a consistent queue manager configuration and usage pattern to enable full automation and deployment of your system.

It's hard to treat the queue managers as true cattle, the message state associated with each is typically critical. But it is possible to architect your system to minimise any single point of failure through high availability and active/active patterns.

And it's definitely possible to separate the messaging state from the physical/virtual system it is currently running on.

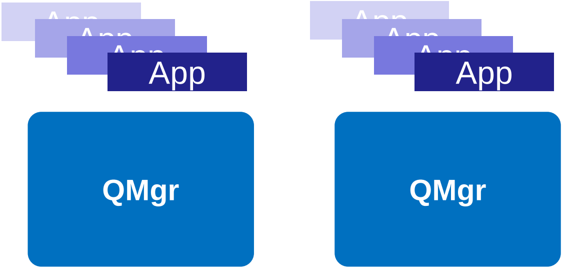
Pets: Each system is unique and indispensable.

Hand-built and customised. Lovingly nurtured.



Cattle: Uniform systems, built using automation. Built for failure. If they go wrong it doesn't matter if another takes its place.

Tenancy



Multi tenant

Potentially lower runtime overheads

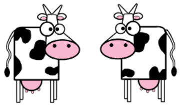
More care needed in configuring to achieve isolation

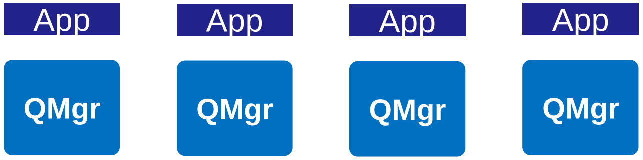
Isolation of machine resources not possible

Harder/simpler to monitor

Depends on your view of more queue managers

Fine grain security required





Single tenant

Simple to configure, maintain and monitor

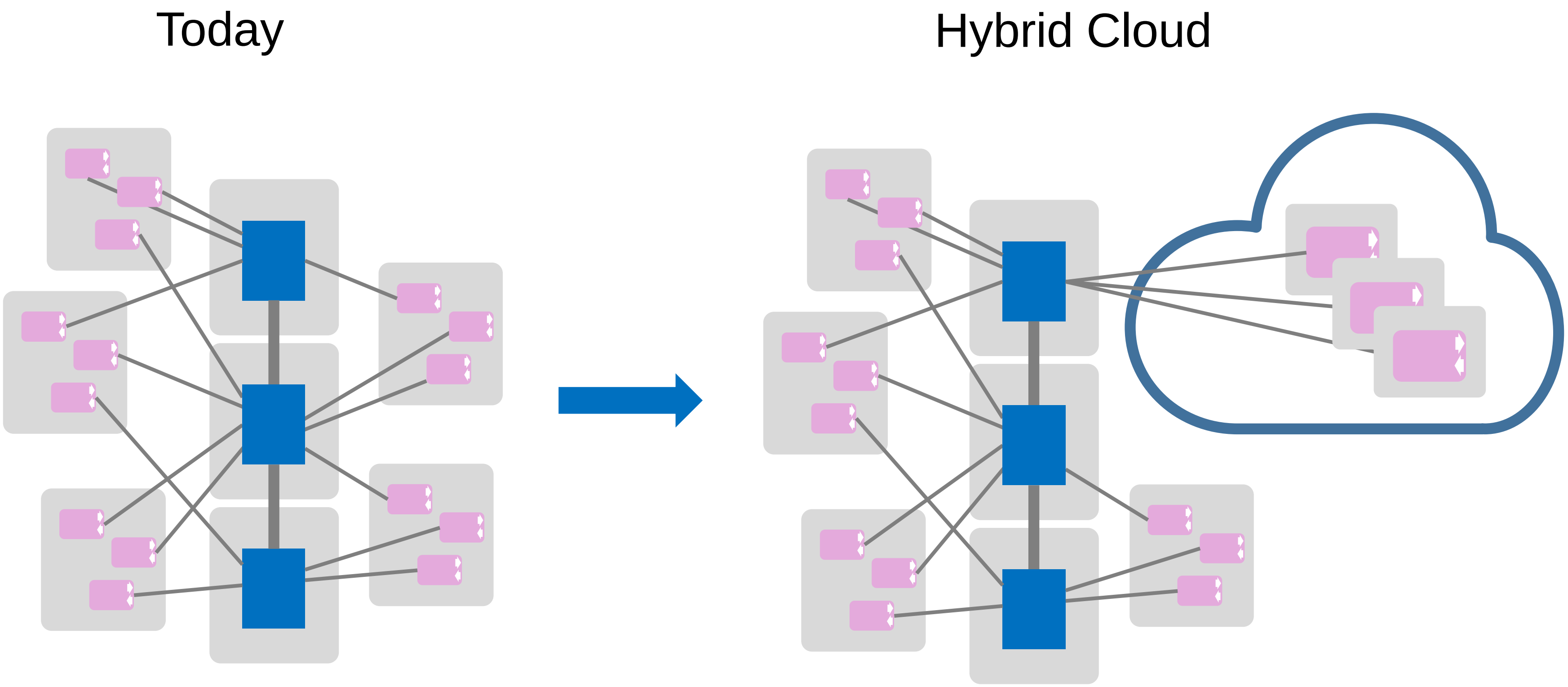
Very good isolation

A proliferation of queue managers

Harder when integration is required

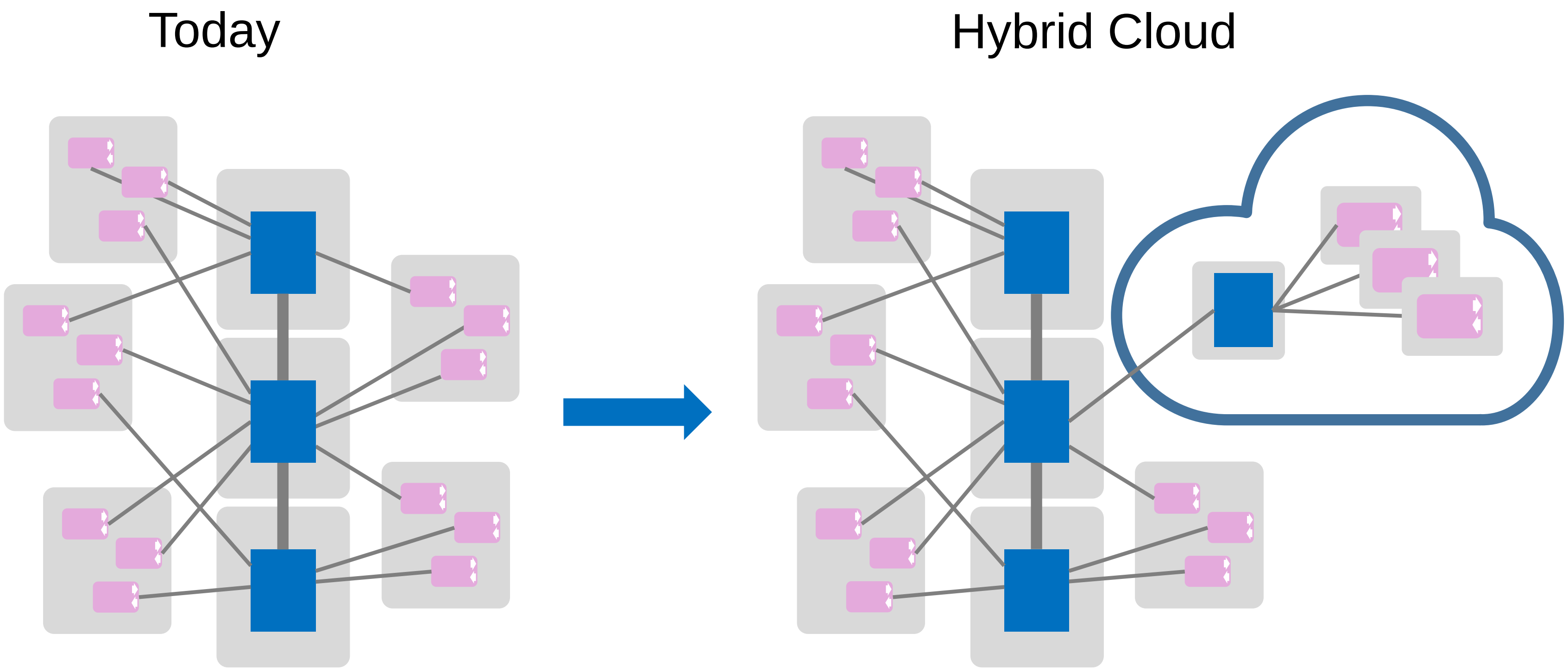
Best suited to scalable, cloud deployments

Hybrid Architectures



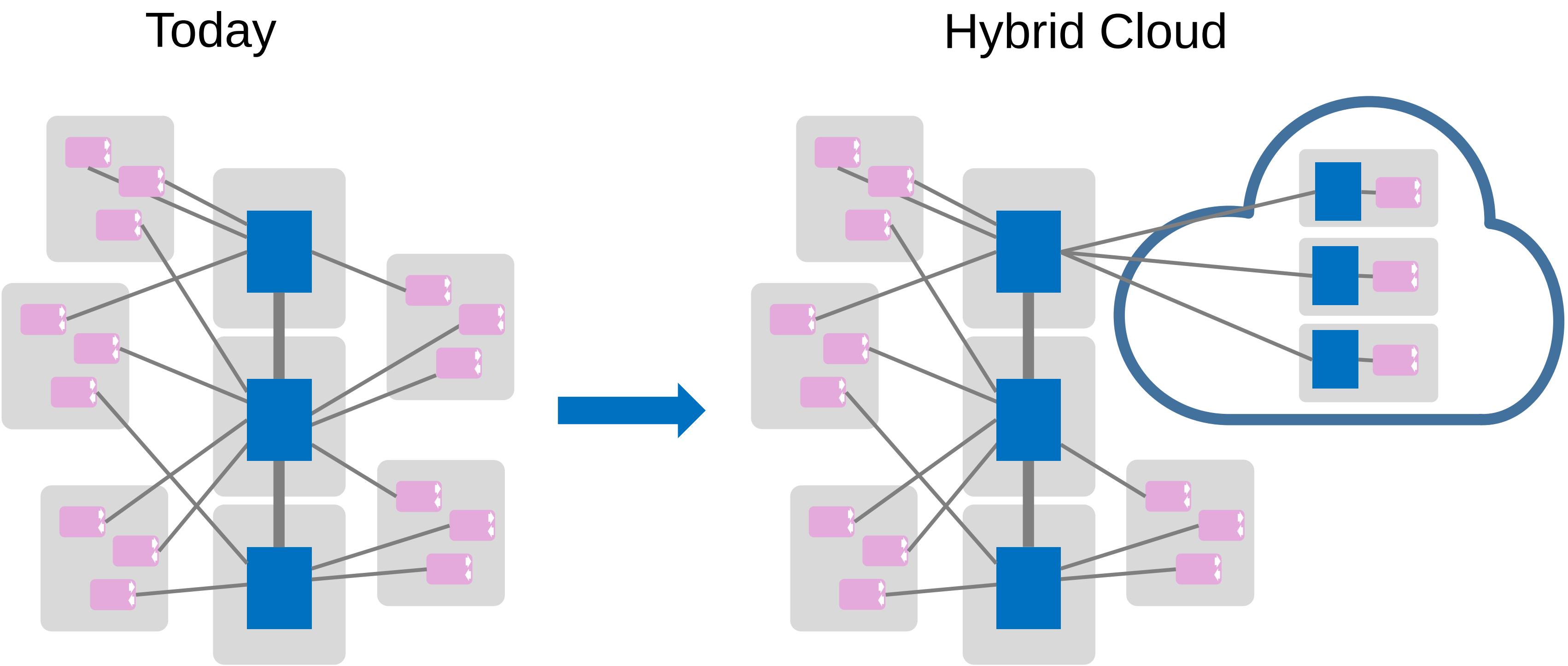
- Run MQ clients in the cloud
- Connect to on-premise hub
- Applications running in container, Cloud Foundry, serverless environment (e.g. Lambda/OpenWhisk), etc…

Hybrid Architectures



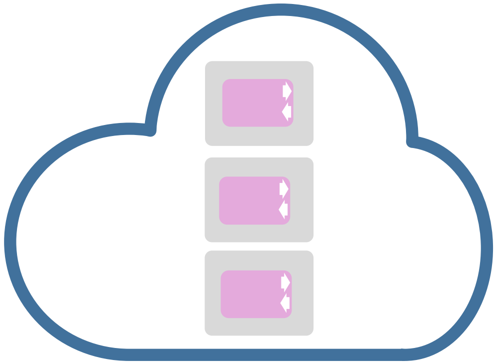
- Single queue manager run in the cloud
- Gateway QM connects to on-premise hub
- Not multi-tenancy - apps are scaled instances
- Allows some communication between cloud apps without going back to on-premise

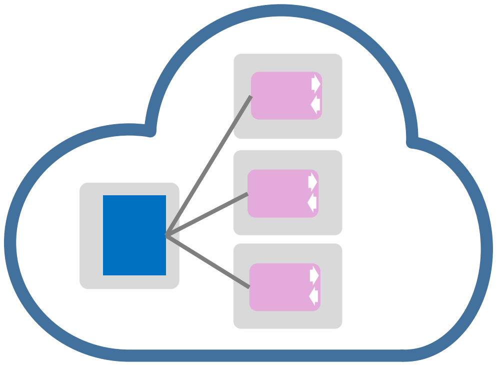
Hybrid Architectures

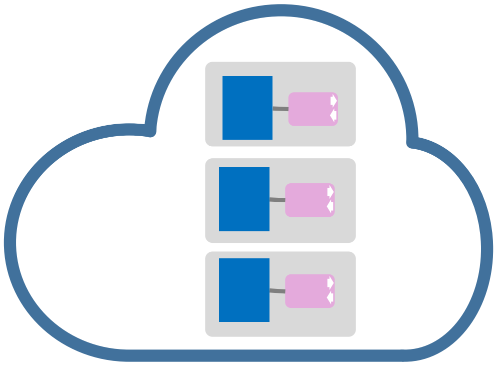


- Queue managers run in the cloud alongside apps
- Connect to on-premise hub
- Run in **VMs** or **containers**
- Unless you have a good reason to run QMs along side apps this may not be the best architecture for cloud

Hybrid Architectures







Clients

- Easier to scale ✓
- Stateless ✓
- Less administration ✓

- Need to discover a QM ✗
- Can’t operate during network partition ✗

Clients & Gateway QM

- Client service discovery simpler ✓
- QM manages discovery and routing ✓
- Single place to configure connectivity back to the enterprise ✓

- Limits app scalability ✗
- Not very cloudy ✗

Clients & QMs

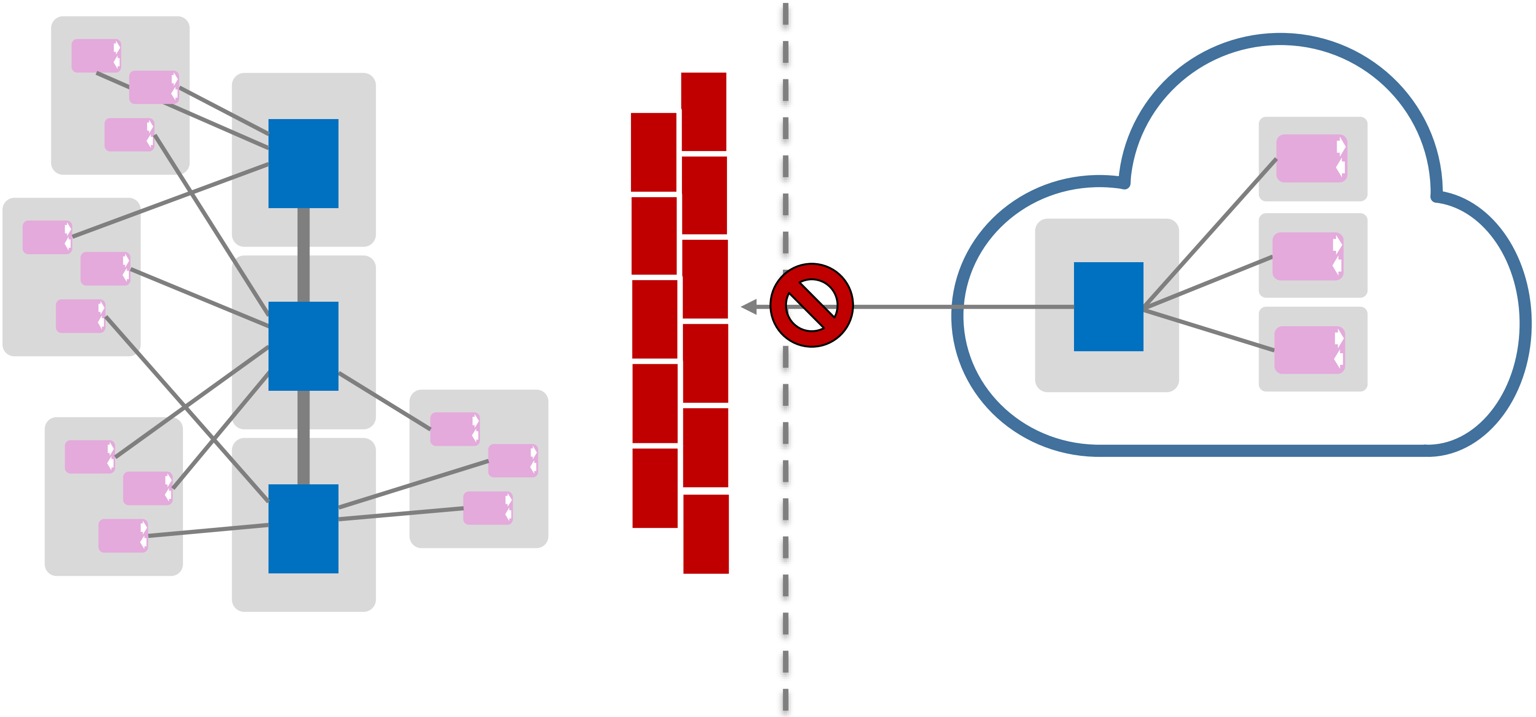
- QM buffers messages between outages ✓
- Client service discovery easier ✓

- More admin required ✗
- Need access to each QMs logs ✗
- Harder to scale down ✗
- Can apps really do anything during an outage anyway? ✗

Agenda

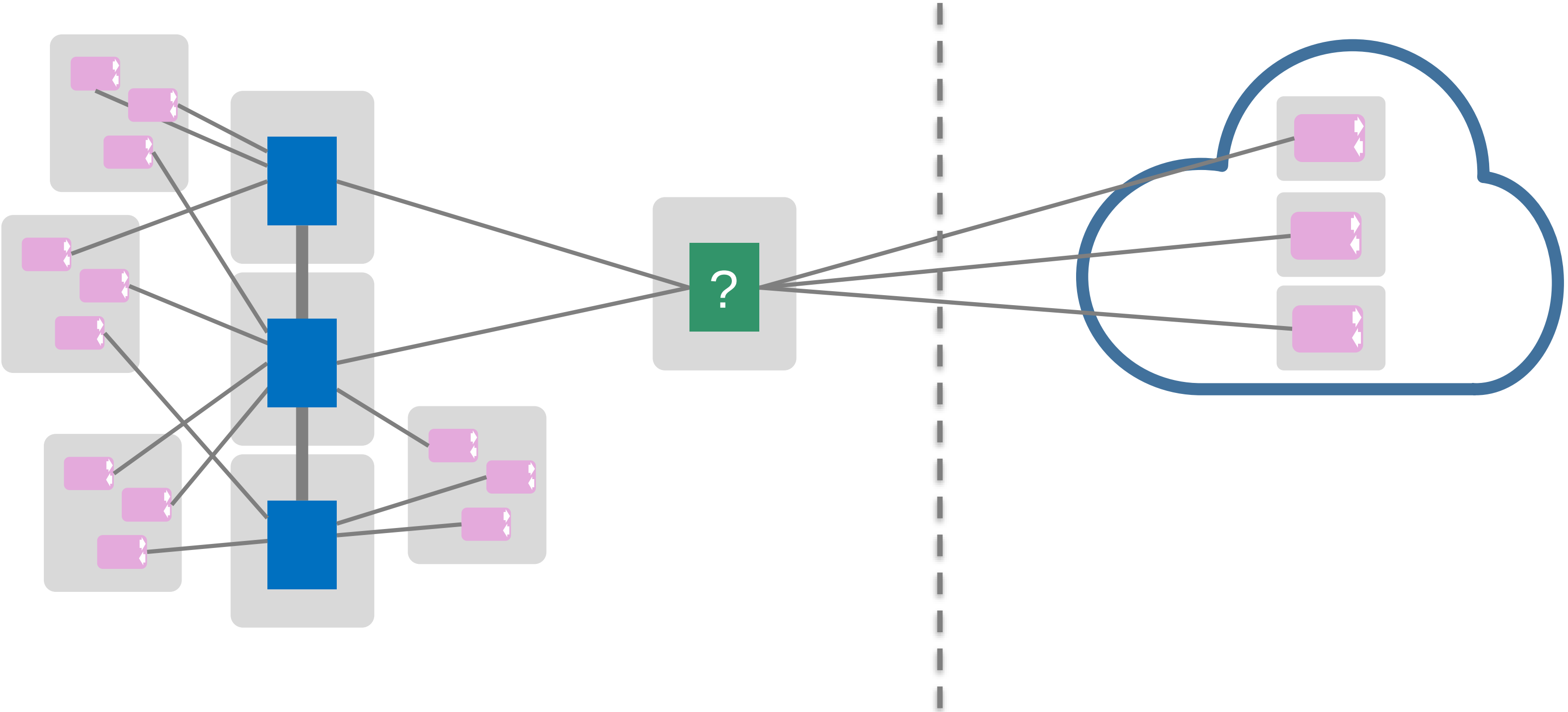
- Topologies
- Connectivity
- Clients & Applications

Connectivity



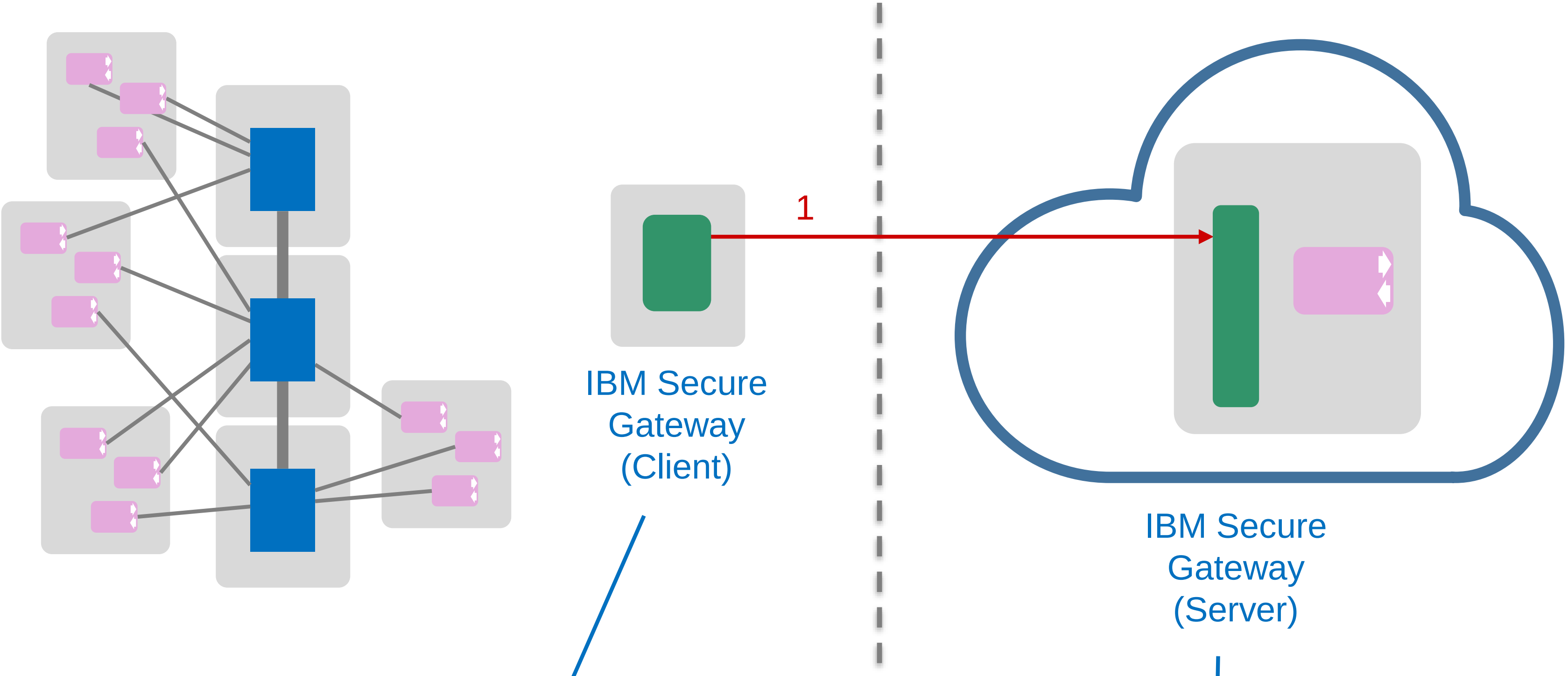
- Enterprise network behind firewall
- Cloud queue manager on public facing IP address
- Cloud can’t connect directly to enterprise QM

Connectivity



- Like connecting from any other external network, need to route connectivity through firewall/DMZ
- All cloud platforms provide ways to connect on-premise and cloud networks (e.g. IBM SecureGateway, DirectConnect, VPN)

Connectivity – IBM Secure Gateway

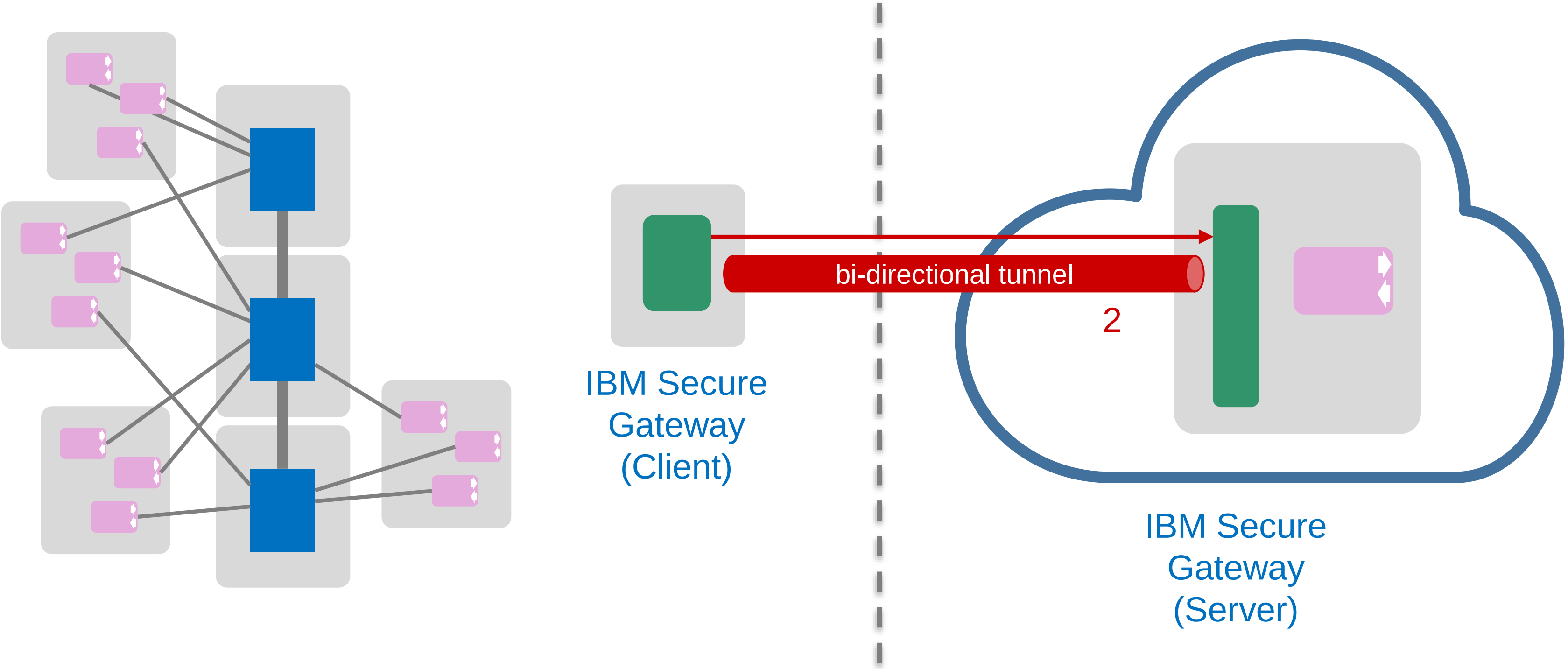


Secure Gateway client runs on-premise

- Native Mac/Linux/Win app
- Docker
- DataPower

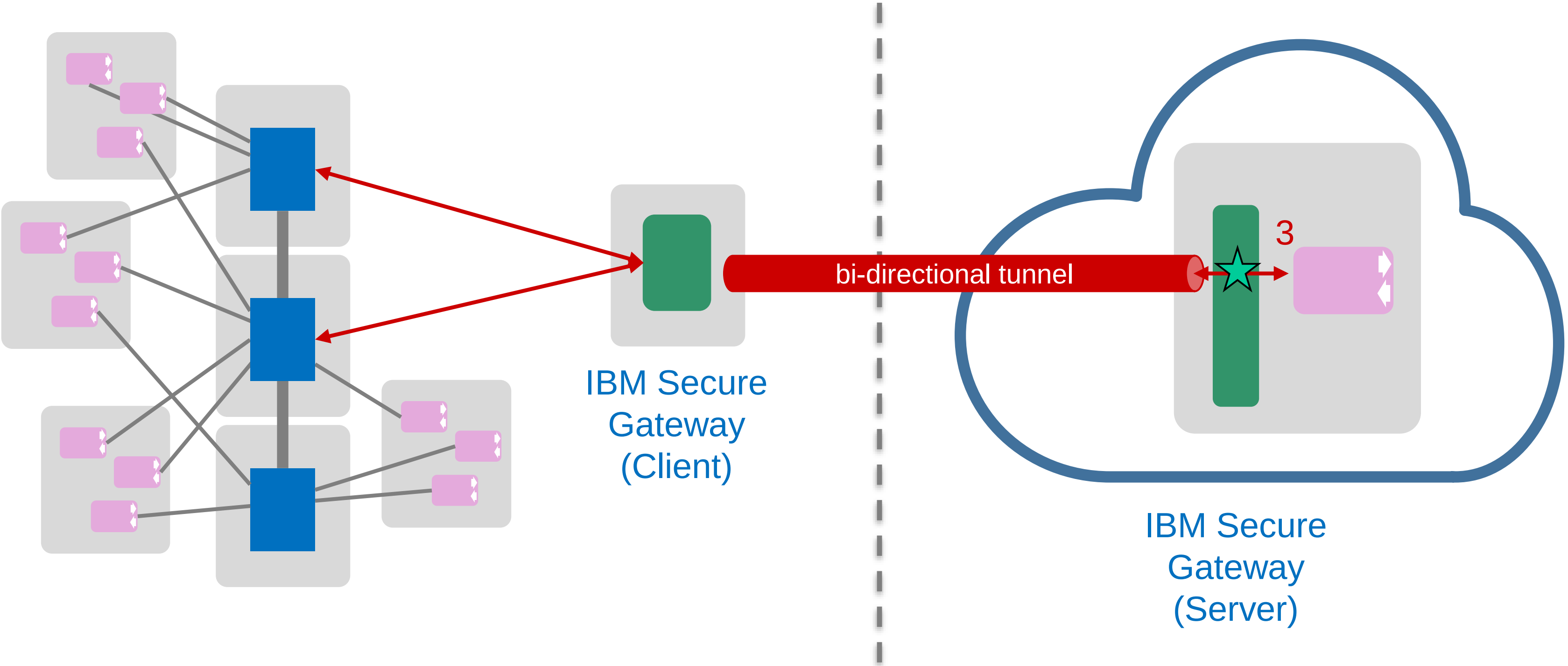
Connects to IBM Cloud Secure Gateway

Connectivity – IBM Secure Gateway



- Secure Gateway sets up a tunnel to on-premise client

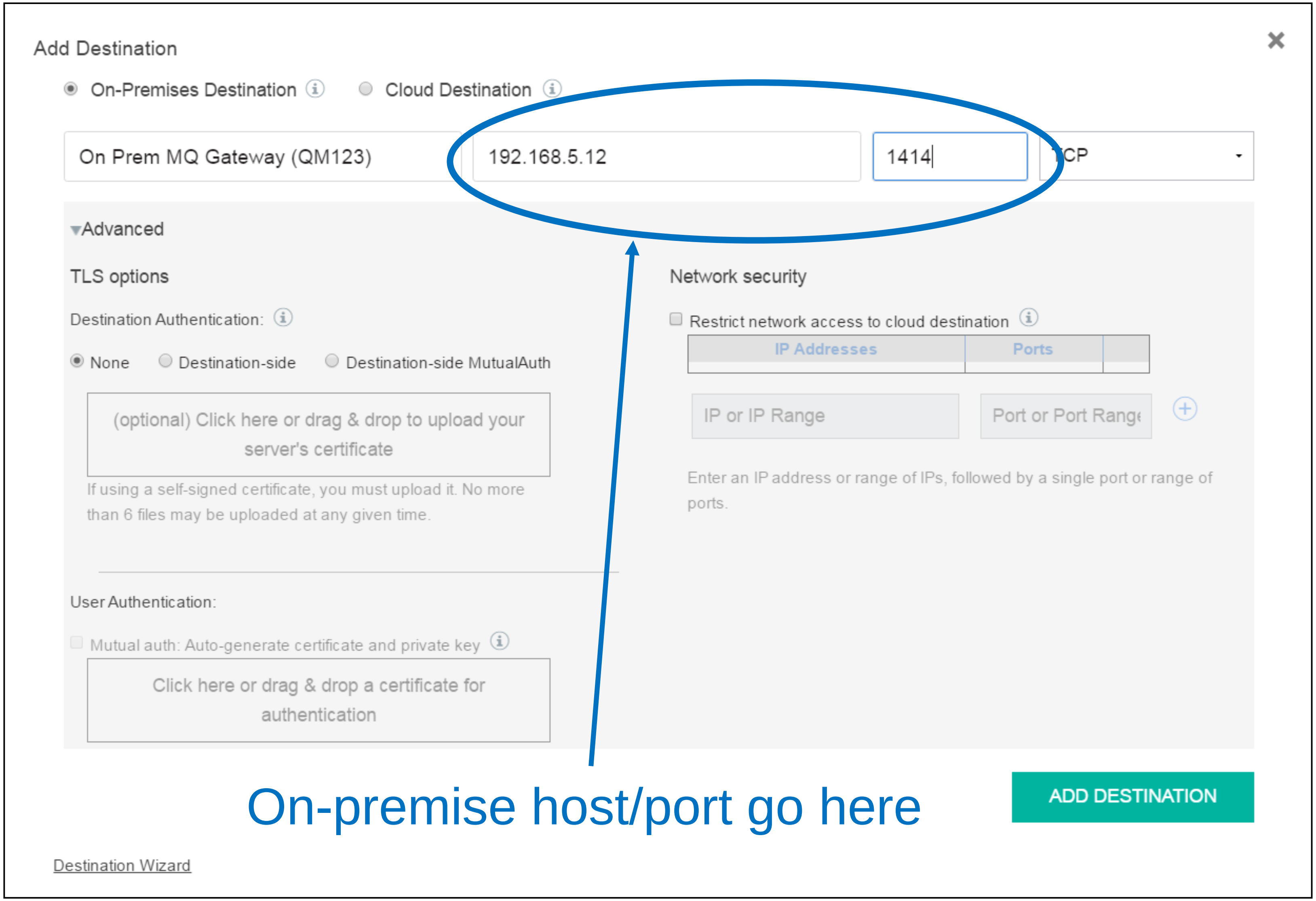
Connectivity – IBM Secure Gateway



- You configure valid routes from Secure Gateway client to on-premise network interfaces
- Cloud application connects to virtual address in cloud e.g. ***cap-sg-prd-1.integration.ibmcloud.com:17036***
- Secure gateway client routes packets to/from on-premise network

☆ Connectivity from app to tunnel secured with TLS and/or restricted IP ranges

Secure Gateway Destinations



Add Destination ✕

● On-Premises Destination ⓘ ○ Cloud Destination ⓘ

On Prem MQ Gateway (QM123) | 192.168.5.12 | 1414 | TCP ▾

▾Advanced

TLS options

Destination Authentication: ⓘ

● None ○ Destination-side ○ Destination-side MutualAuth

(optional) Click here or drag & drop to upload your server's certificate

If using a self-signed certificate, you must upload it. No more than 6 files may be uploaded at any given time.

User Authentication:

□ Mutual auth: Auto-generate certificate and private key ⓘ

Click here or drag & drop a certificate for authentication

Network security

□ Restrict network access to cloud destination ⓘ

IP Addresses	Ports

IP or IP Range | Port or Port Range ⊕

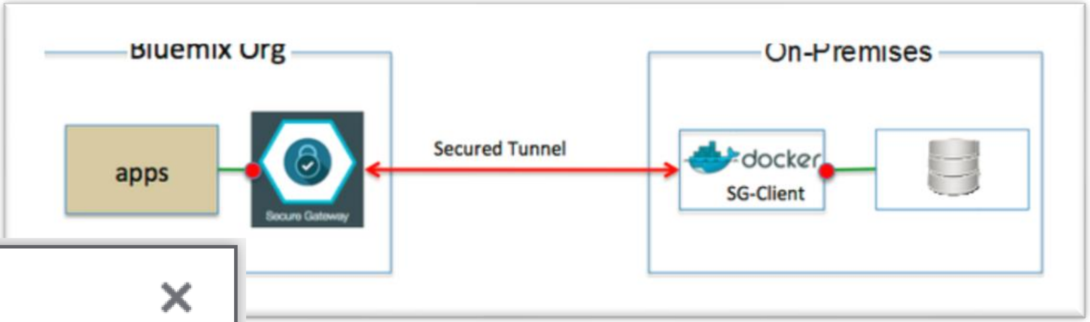
Enter an IP address or range of IPs, followed by a single port or range of ports.

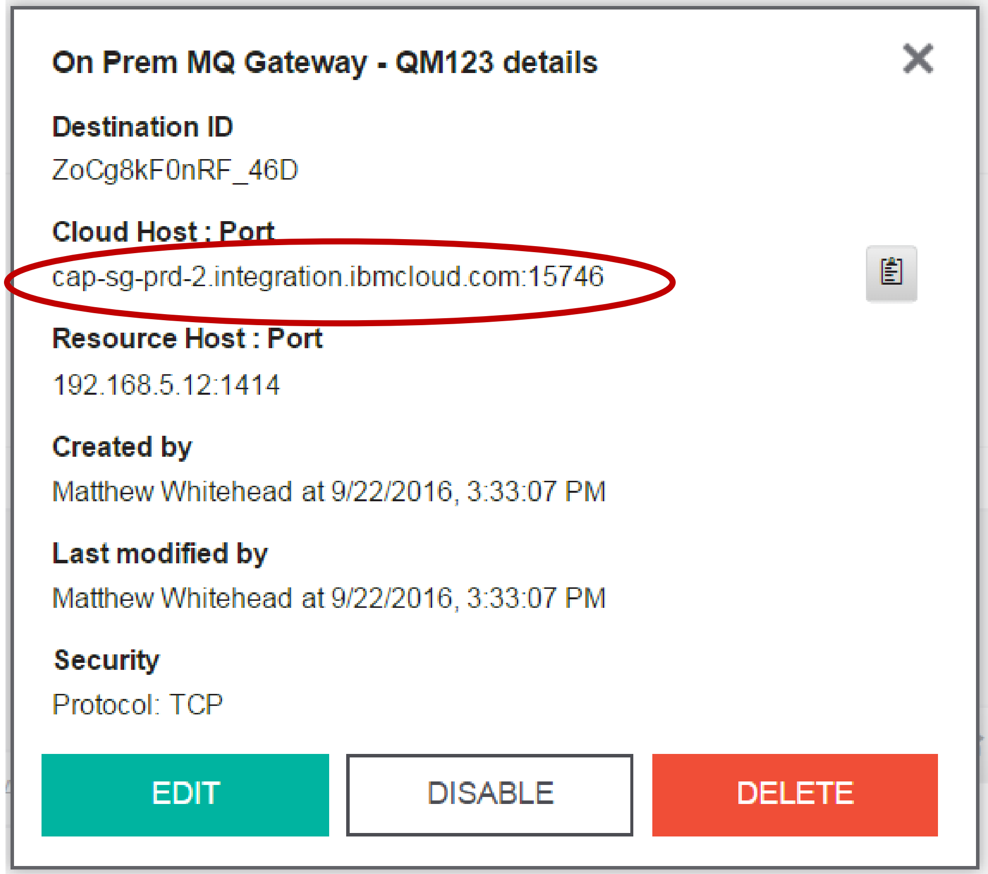
ADD DESTINATION

Destination Wizard

On-premise host/port go here

Secure Gateway Destinations





On Prem MQ Gateway - QM123 details

Destination ID
ZoCg8kF0nRF_46D

Cloud Host : Port
cap-sg-prd-2.integration.ibmcloud.com:15746

Resource Host : Port
192.168.5.12:1414

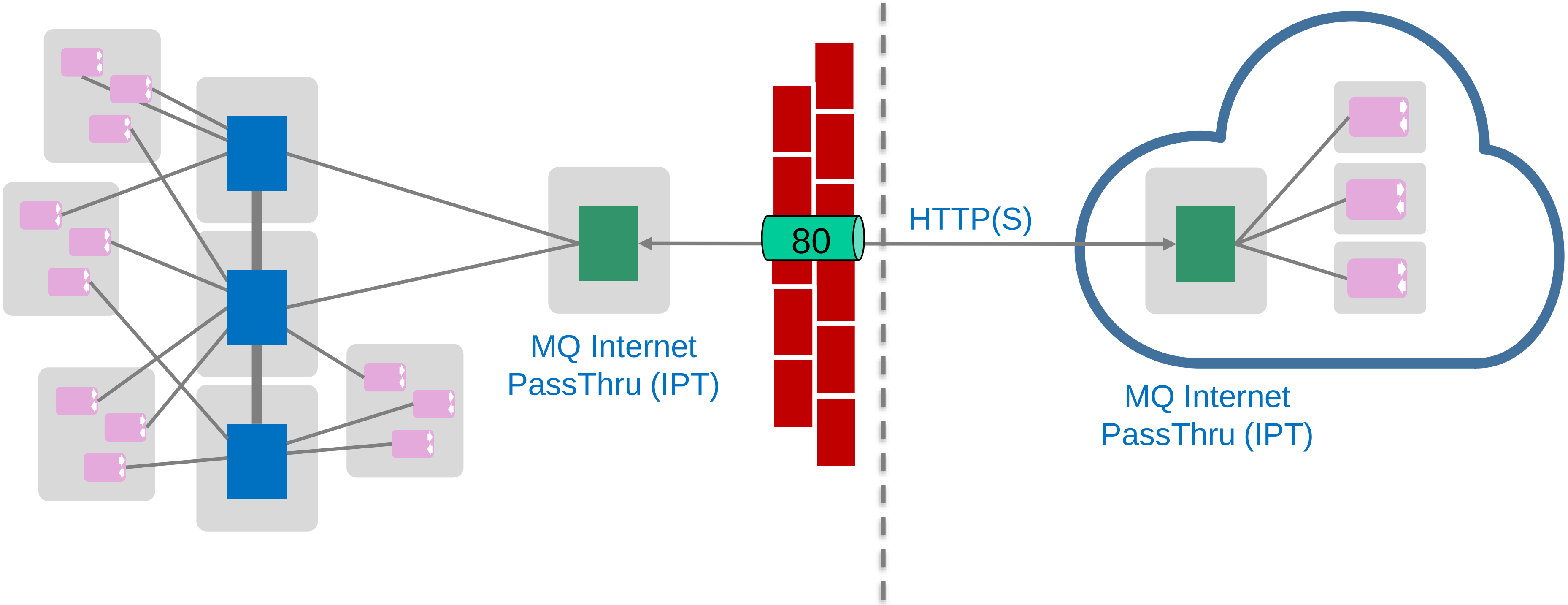
Created by
Matthew Whitehead at 9/22/2016, 3:33:07 PM

Last modified by
Matthew Whitehead at 9/22/2016, 3:33:07 PM

Security
Protocol: TCP

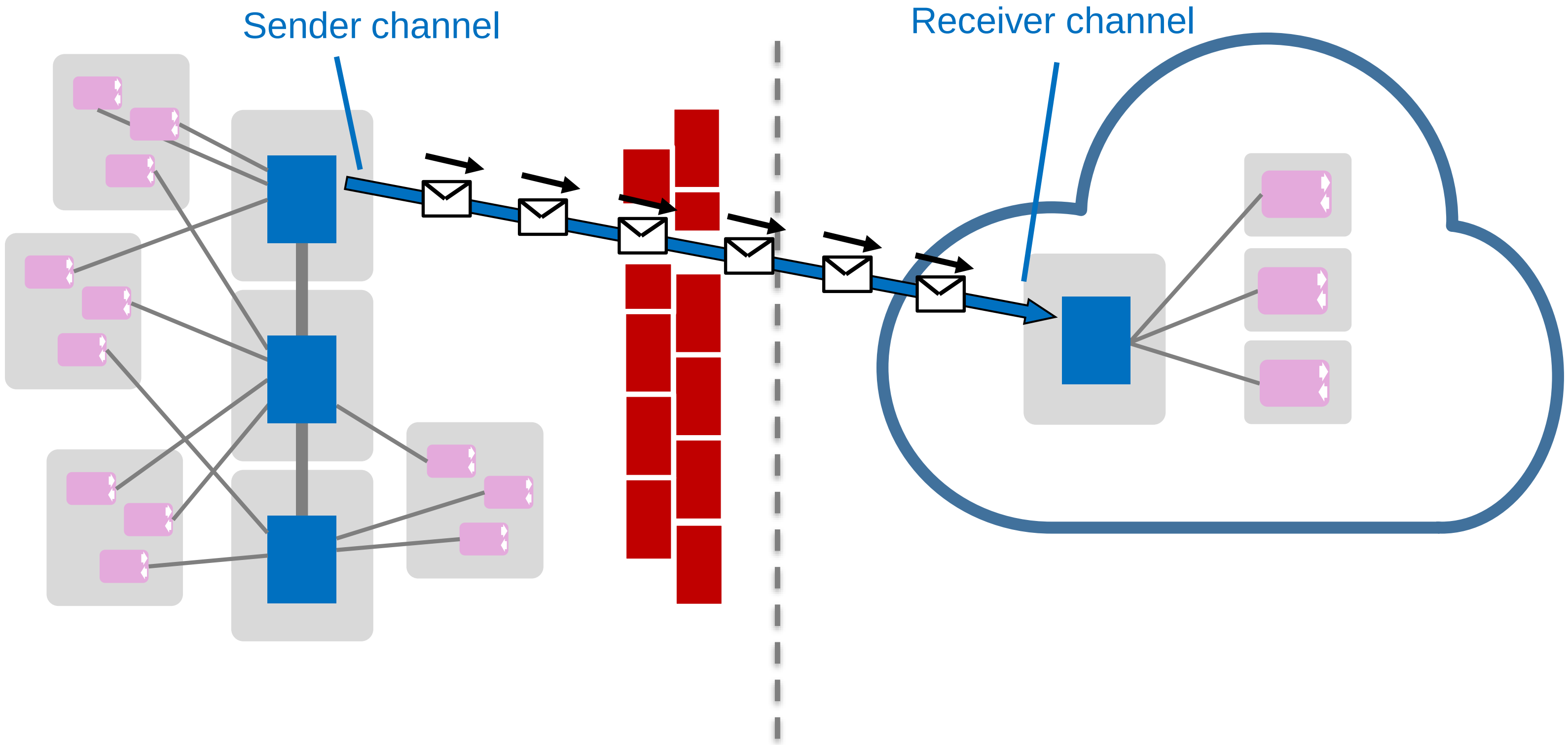
- Once an on-premise destination IP address has been defined, the secure gateway allocates a host name and port
- Your cloud application connects to this virtual host name
- The secure gateway routes traffic to the on-premise address 192.168.5.12:1414

Connectivity – MQ Internet PassThru (MS81)



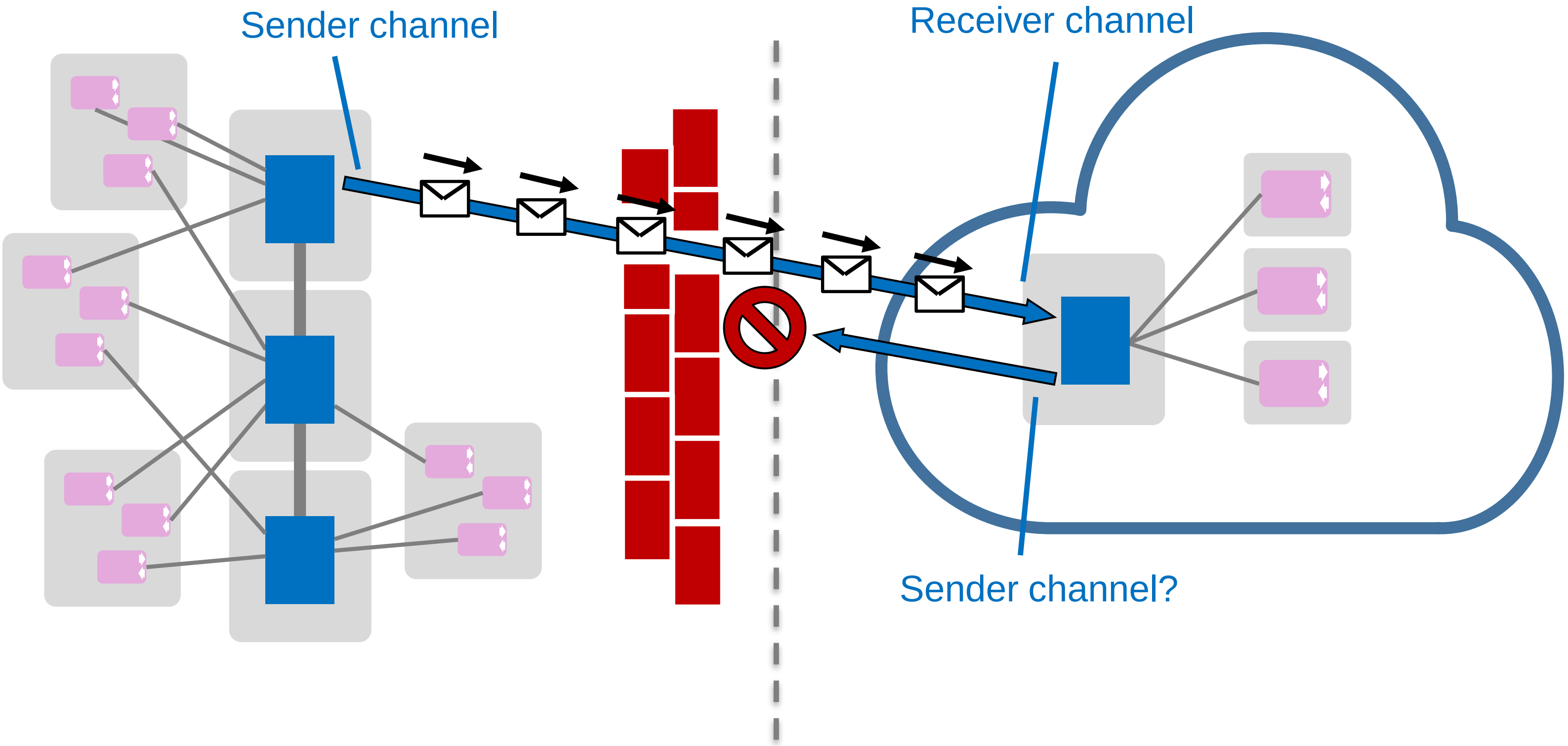
- Avoids the need for a direct TCP connection from cloud to on-prem
- Tunnel MQ traffic over HTTP(S)
- Avoids requirement for more complicated VPN configuration
- Re-use on-prem IPT if you're already using it
- Cloud agnostic

Server/Requester channels



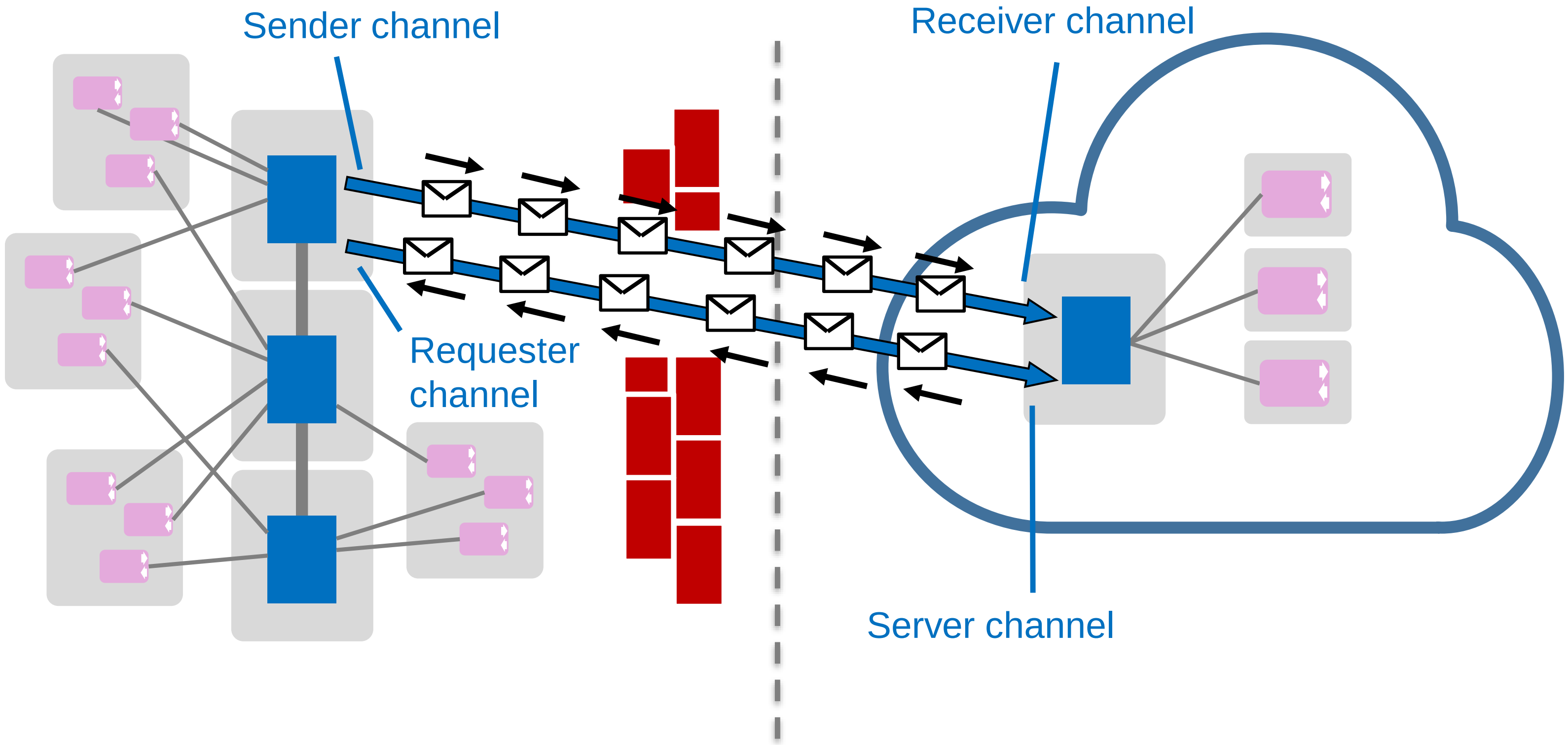
- Initiating channels **TO** the cloud is easy
- **SENDER** channel defined on-premise, **RECEIVER** channel defined in the cloud

Server/Requester channels



- Initiating channels **FROM** the cloud is more difficult
- Typically sender channels won’t be able to connect through the firewall to on-premise listener/receiver channel

Server/Requester channels



- Instead, a **REQUESTER** channel defined on-premise initiates a connection to a **SERVER** channel running the cloud
- Once the connection has been established, it works much like a sender/receiver pair, sending messages from cloud to on-prem

Agenda

- Topologies
- Connectivity
- Clients & Applications

Client Runtimes

- MQ offers a lot of different application runtime options
  - C, C++, JEE, CICS…
- Putting existing applications into cloud-hosted VMs is certainly possible
  - but - are there better runtimes for your new cloud-era applications?
- New concepts like serverless programming suit some runtimes over others
- E.g. AWS Lambda™
  - Node.js
  - Java™
  - Python
  - .NET® C#

Ruby
A PROGRAMMER'S BEST FRIEND

.NET

node JS

Java™

AMQP
Advanced Message Queuing Protocol

python™

Client Runtimes

- **Cloud Foundry™** supported buildpacks
  - Java
  - .NET Core
  - Node.js
  - PHP
  - Python
  - Ruby
  - Go
- but you can still push native MQ apps to Cloud runtimes as we'll see later…

Ruby
A PROGRAMMER'S BEST FRIEND

.NET

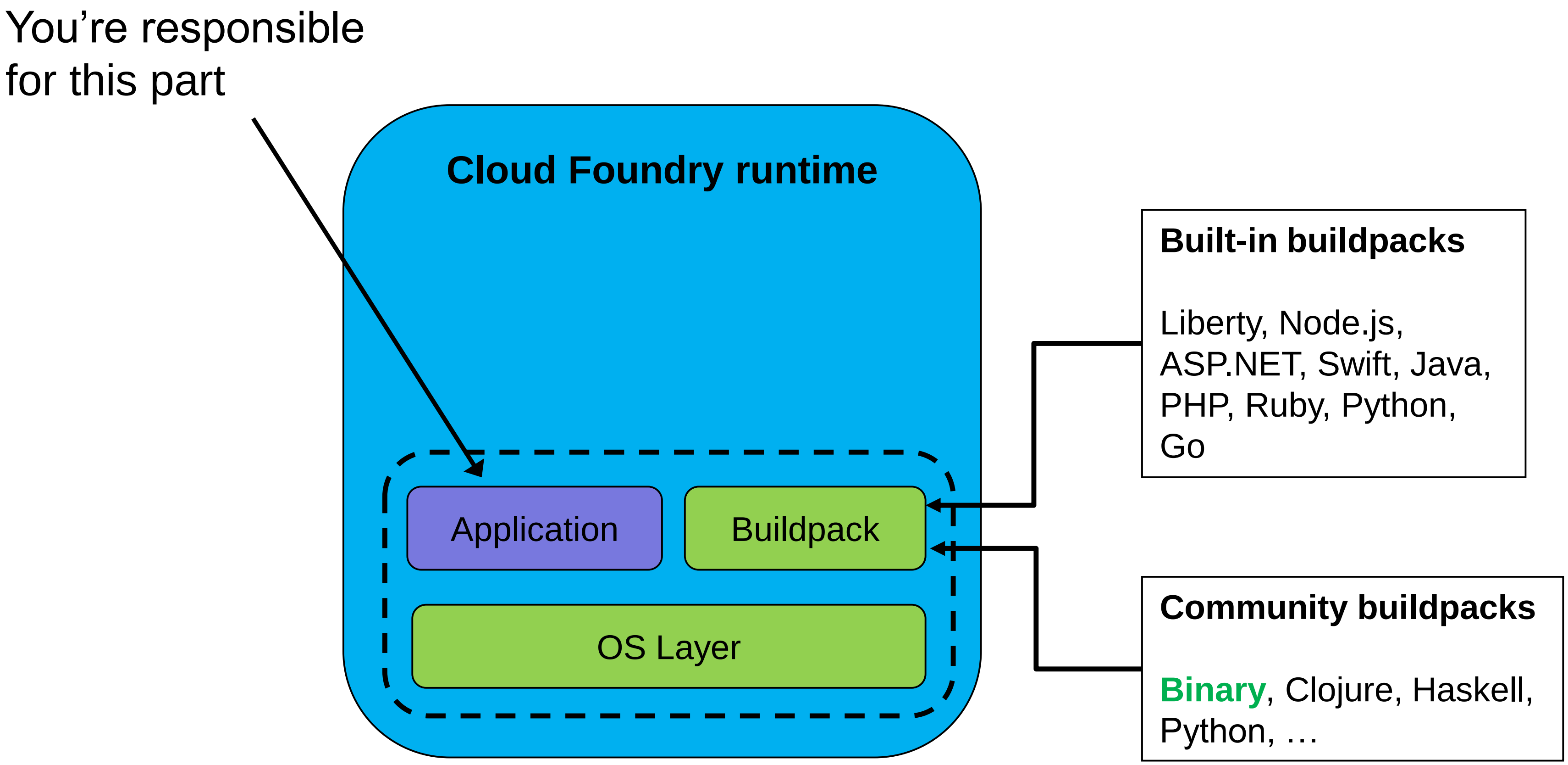
node JS

Java™

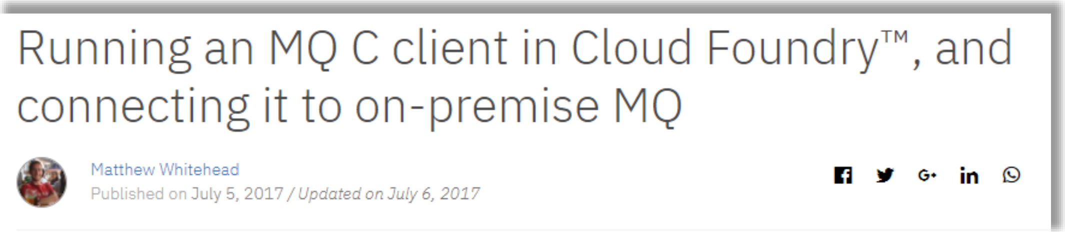
AMQP
Advanced Message Queuing Protocol

python™

Native applications in Cloud Foundry™


You’re responsible for this part
Cloud Foundry runtime
Application
Buildpack
OS Layer
Built-in buildpacks
Liberty, Node.js, ASP.NET, Swift, Java, PHP, Ruby, Python, Go
Community buildpacks
Binary, Clojure, Haskell, Python, …

Cloudifying native apps

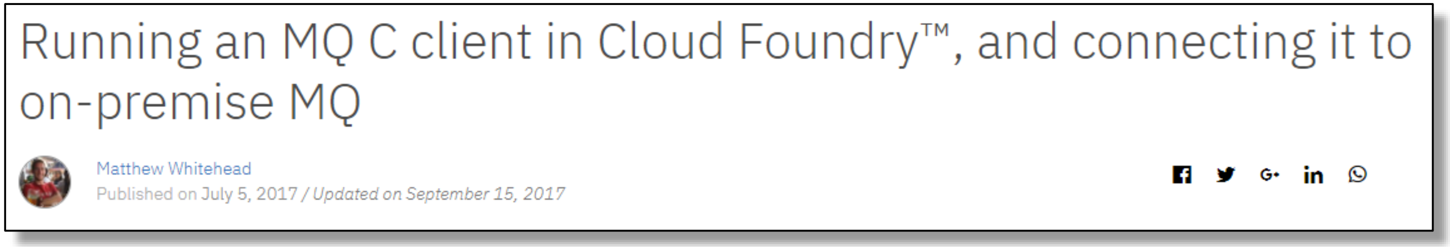


Running an MQ C client in Cloud Foundry™, and connecting it to on-premise MQ

Matthew Whitehead
Published on July 5, 2017 / *Updated on July 6, 2017*

- You can still deploy native applications to cloud platforms
  - See binary buildpack for cloudfoundry…

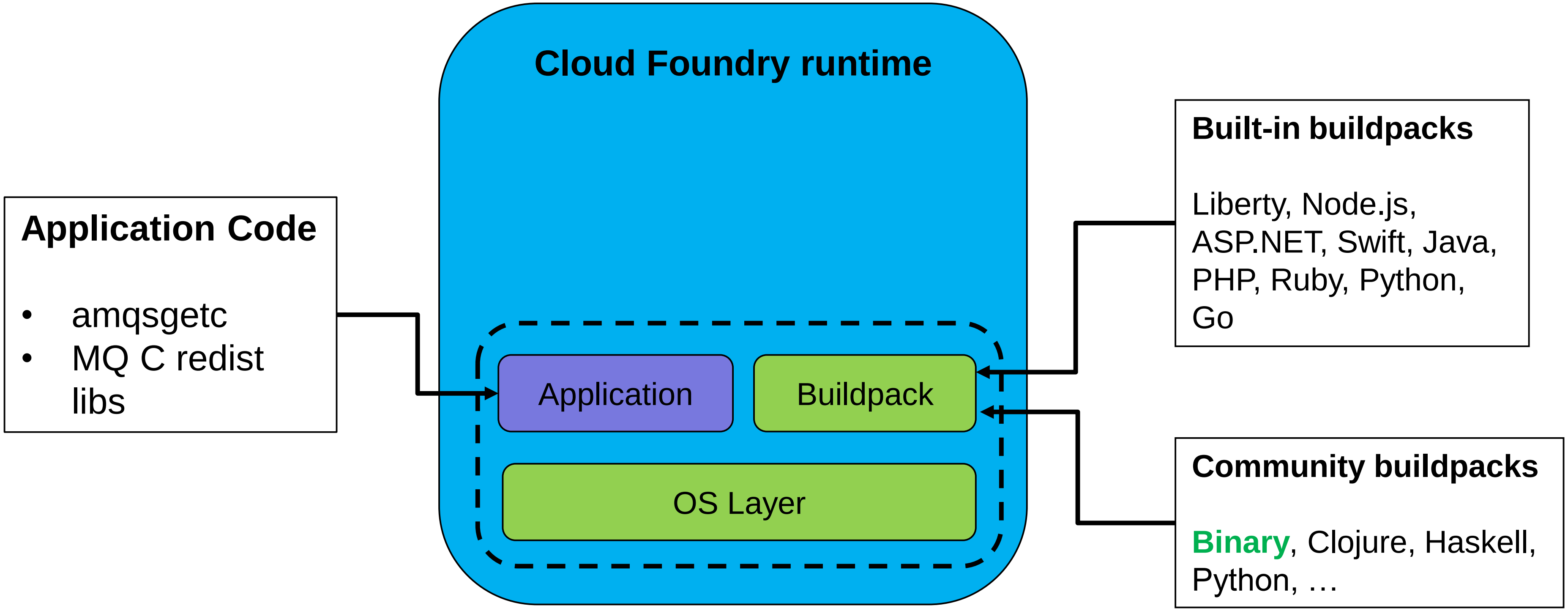
Native applications in Cloud Foundry™



Running an MQ C client in Cloud Foundry™, and connecting it to on-premise MQ

Matthew Whitehead

Published on July 5, 2017 / *Updated on September 15, 2017*



Application Code

- amqsgetc
- MQ C redist libs

Cloud Foundry runtime

Application

Buildpack

OS Layer

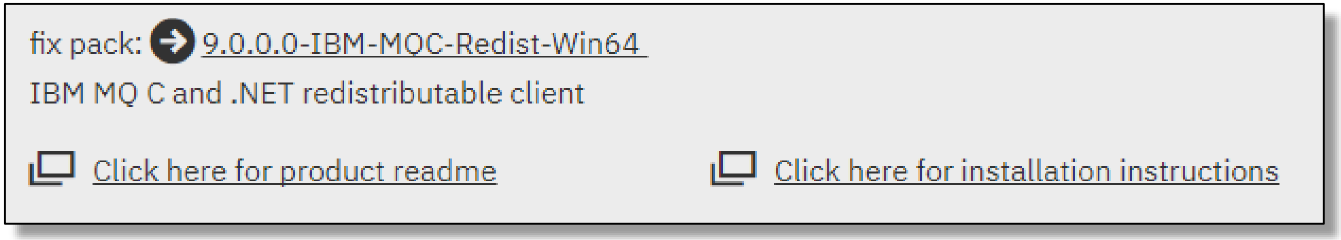
Built-in buildpacks

Liberty, Node.js, ASP.NET, Swift, Java, PHP, Ruby, Python, Go

Community buildpacks

Binary, Clojure, Haskell, Python, …

A Side Note – MQ Redistributable Clients

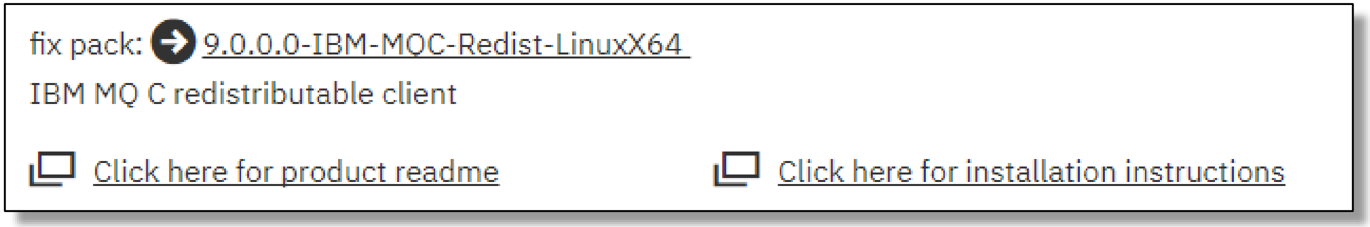


fix pack: 9.0.0.0-IBM-MQC-Redist-Win64

IBM MQ C and .NET redistributable client

Click here for product readme   Click here for installation instructions

Windows
C & .Net

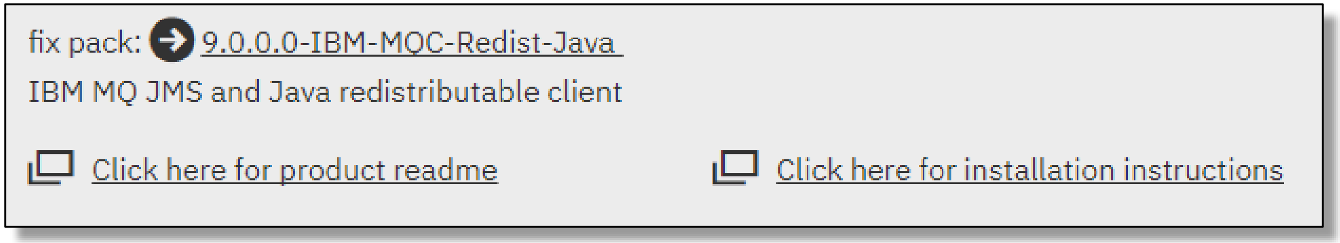


fix pack: 9.0.0.0-IBM-MQC-Redist-LinuxX64

IBM MQ C redistributable client

Click here for product readme   Click here for installation instructions

Linux C



fix pack: 9.0.0.0-IBM-MQC-Redist-Java

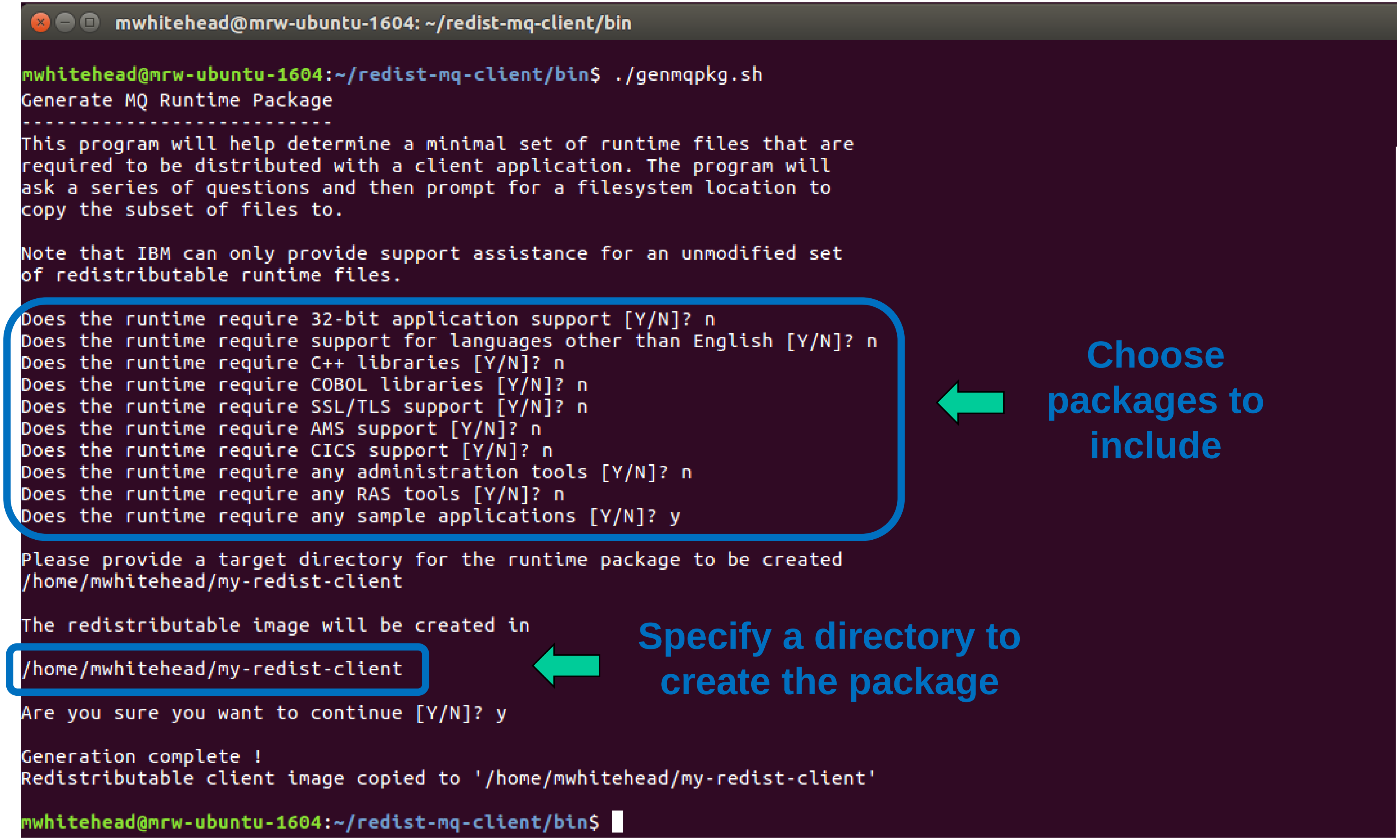
IBM MQ JMS and Java redistributable client

Click here for product readme   Click here for installation instructions

Java/JMS

- Also available for MFT client libraries (create transfers, query agents etc)
- Create your own redistributable packages by stripping out unused libraries
  - See *genmqpkg.sh*

A Side Note – MQ Redistributable Clients

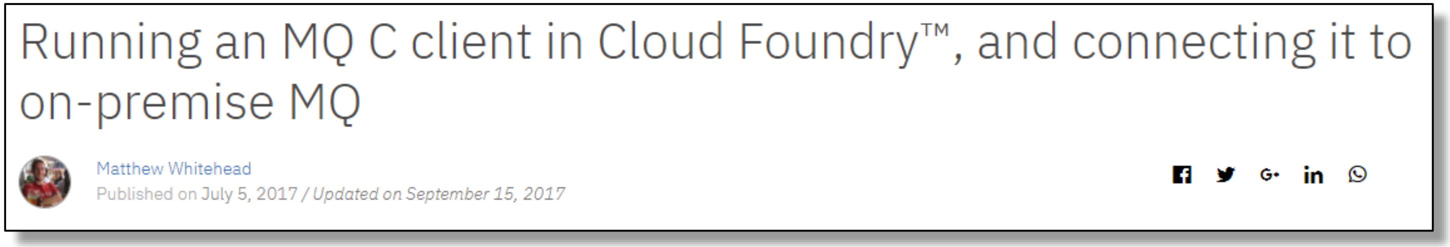


```
mwhitehead@mrw-ubuntu-1604:~/redist-mq-client/bin$ ./genmqpkg.sh
Generate MQ Runtime Package
---------------------------
This program will help determine a minimal set of runtime files that are
required to be distributed with a client application. The program will
ask a series of questions and then prompt for a filesystem location to
copy the subset of files to.

Note that IBM can only provide support assistance for an unmodified set
of redistributable runtime files.

Does the runtime require 32-bit application support [Y/N]? n
Does the runtime require support for languages other than English [Y/N]? n
Does the runtime require C++ libraries [Y/N]? n
Does the runtime require COBOL libraries [Y/N]? n
Does the runtime require SSL/TLS support [Y/N]? n
Does the runtime require AMS support [Y/N]? n
Does the runtime require CICS support [Y/N]? n
Does the runtime require any administration tools [Y/N]? n
Does the runtime require any RAS tools [Y/N]? n
Does the runtime require any sample applications [Y/N]? y

Please provide a target directory for the runtime package to be created
/home/mwhitehead/my-redist-client

The redistributable image will be created in

/home/mwhitehead/my-redist-client

Are you sure you want to continue [Y/N]? y

Generation complete !
Redistributable client image copied to '/home/mwhitehead/my-redist-client'

mwhitehead@mrw-ubuntu-1604:~/redist-mq-client/bin$
```

Choose packages to include

Specify a directory to create the package

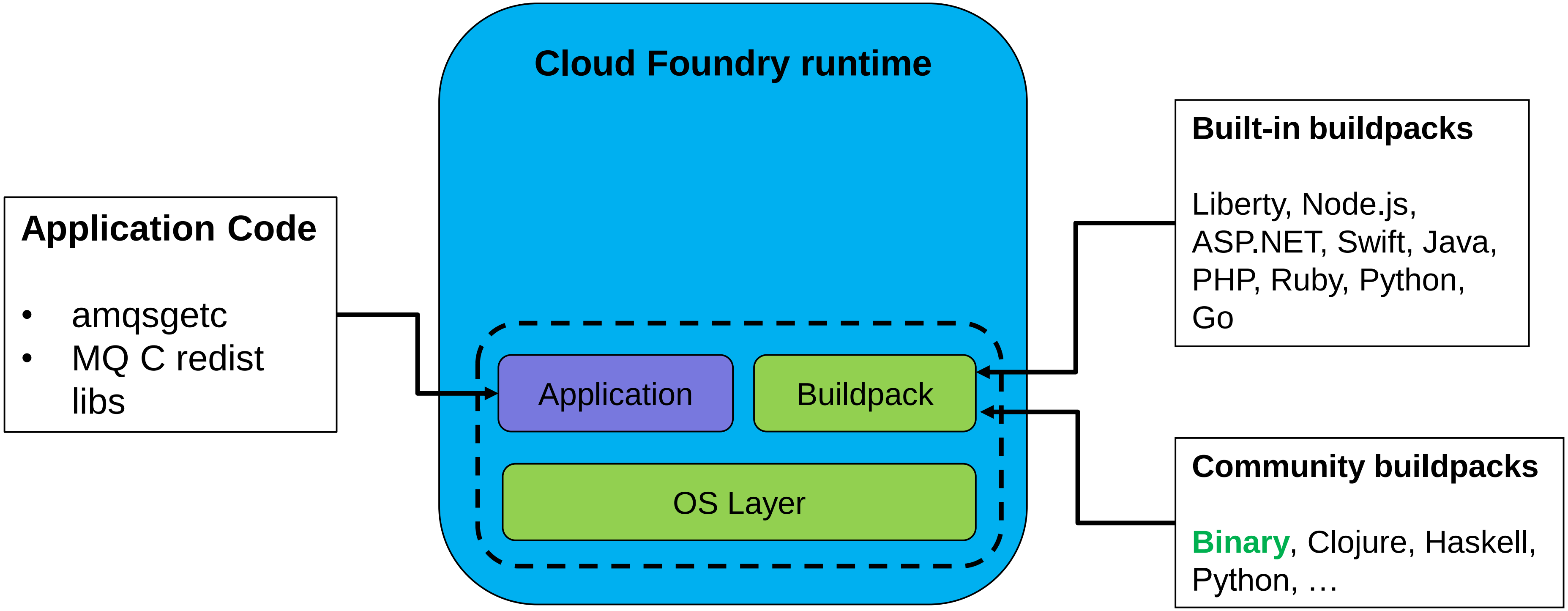
Native applications in Cloud Foundry™



Running an MQ C client in Cloud Foundry™, and connecting it to on-premise MQ

Matthew Whitehead

Published on July 5, 2017 / *Updated on September 15, 2017*



Application Code

- amqsgetc
- MQ C redist libs

Cloud Foundry runtime

Application

Buildpack

OS Layer

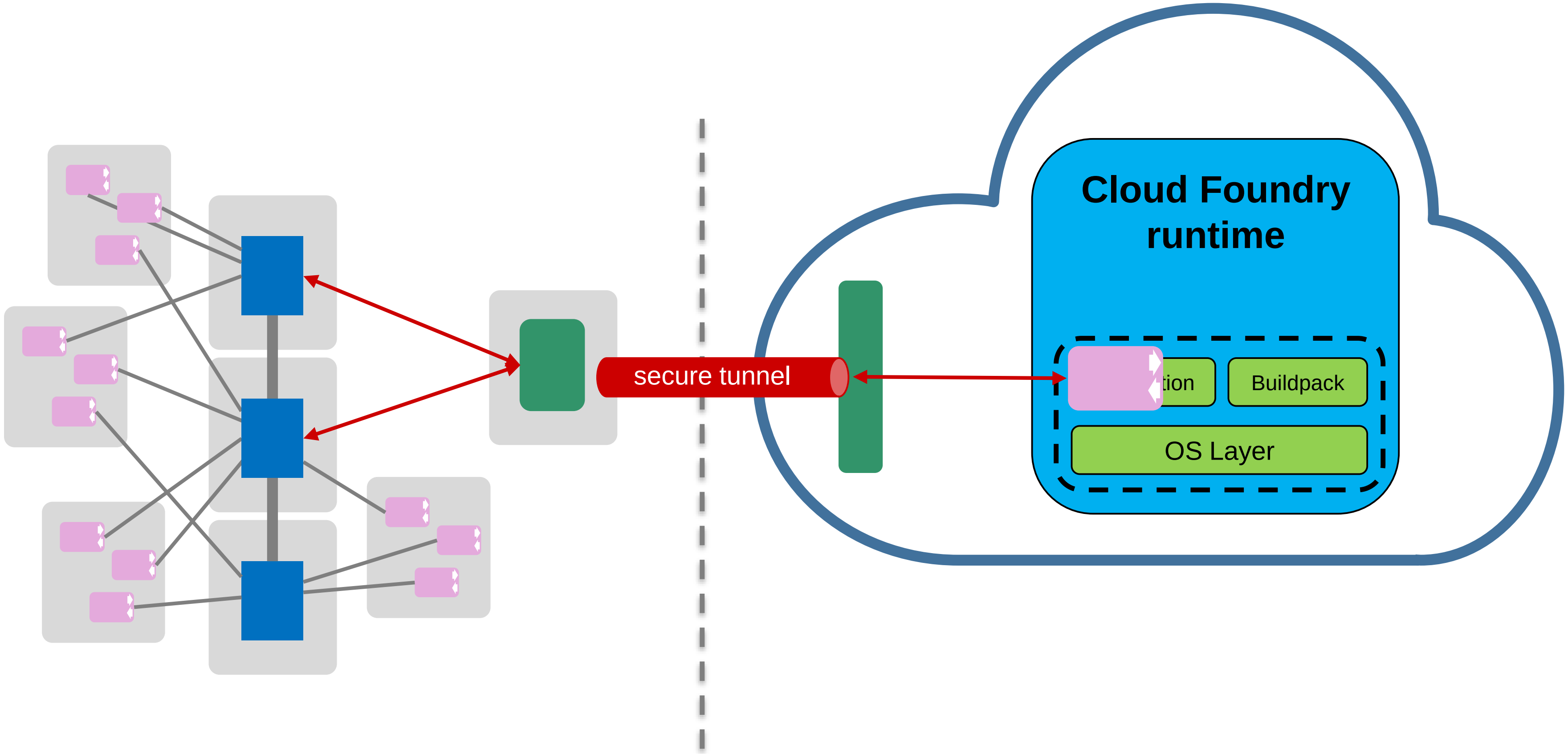
Built-in buildpacks

Liberty, Node.js, ASP.NET, Swift, Java, PHP, Ruby, Python, Go

Community buildpacks

Binary, Clojure, Haskell, Python, …

Native applications in Cloud Foundry™



Client Runtimes

- **IBM Cloud Functions** based on Apache OpenWhisk™
  - Java
  - Node.js
  - Python
  - PHP
  - Docker®

Ruby
A PROGRAMMER'S BEST FRIEND

.NET

node JS

Java™

AMQP
Advanced Message Queuing Protocol

python™

Serverless Functions

IBM Bluemix™

IBM Cloud Functions™

amazon web services

AWS Lambdas™

Microsoft Azure Functions™

Google Cloud Platform

Google Cloud Functions™

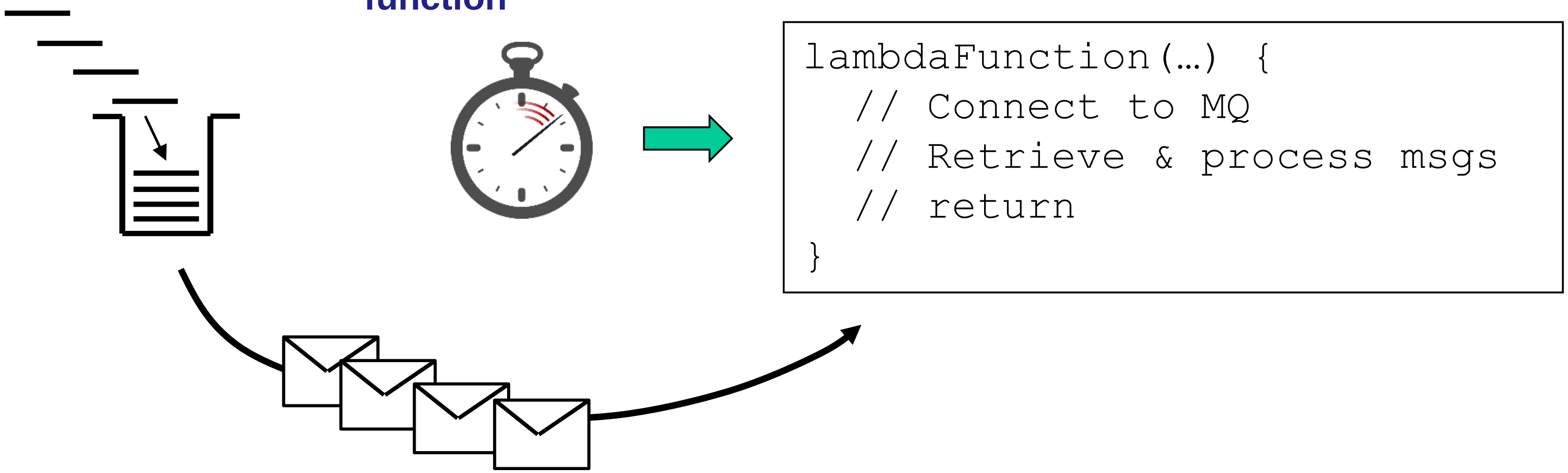
- Ideal for short-lived application logic
- Only pay for the time functions are executing
- Like PaaS, you don’t worry about the OS environment or the application runtime (JVM, nodejs runtime, Pyhon interpreter etc.)
- Just write your function and AWS will invoke it when a defined action occurs
- Scalability and availability is an inherent part of the architecture
  - 1 event = 1 function invocation
  - 10 concurrent events = 10 concurrent function invocations

Serverless Functions

How can you drive MQ serverless applications?

It is difficult since serverless functions don’t generally support long-lived connections

One option - use timer events to invoke functions, e.g.



1. Messages arrive for a subscription

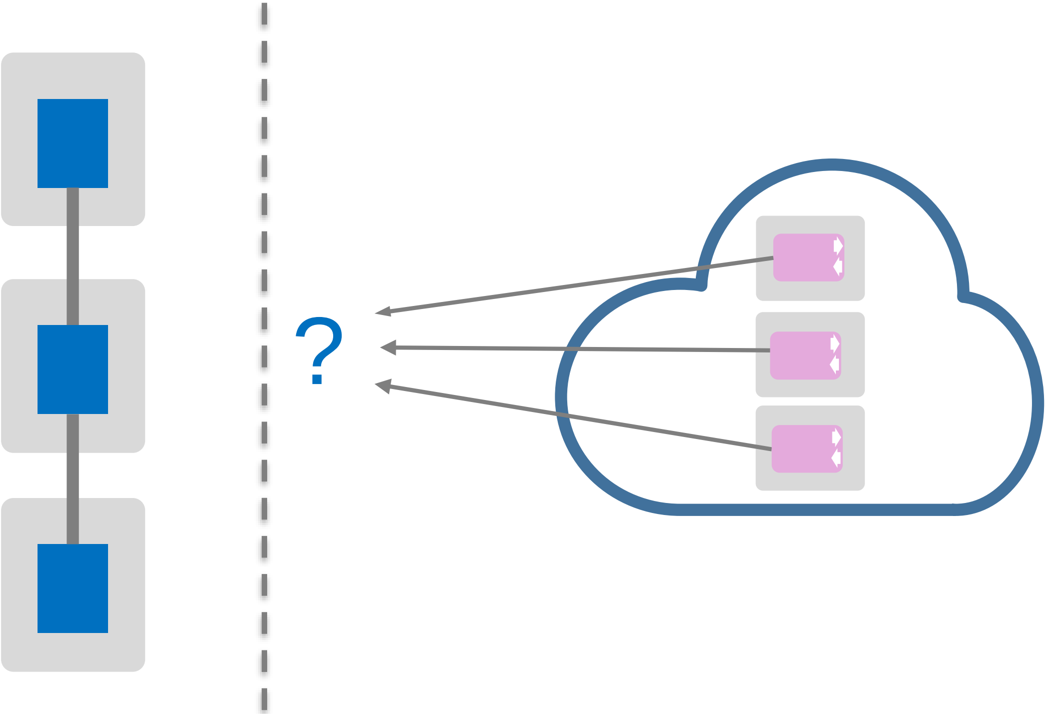
2. Timer periodically invokes lambda function

3. Function connects to QM, consumes messages, returns.

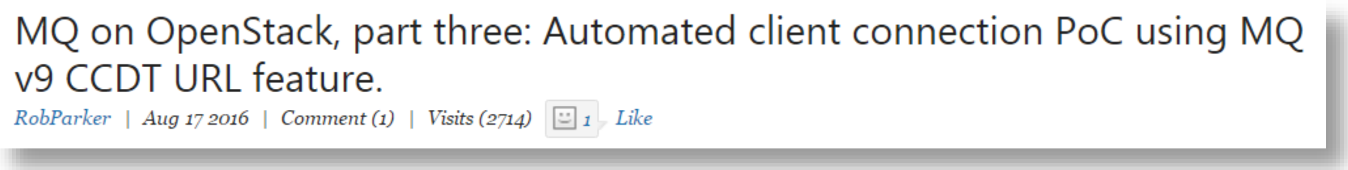
```
lambdaFunction(…) {
  // Connect to MQ
  // Retrieve & process msgs
  // return
}
```

Service Discovery

- Clients need to discover where to connect
- Can be done a number of different ways
  - MQSERVER env
  - CCDT (MQCHLLIB & MQCHLTAB, MQCCDTURL)
  - mqclient.ini
  - JNDI
- But also…
  - MQ Light client service lookup (JSON)
  - DNS
  - Key/value store

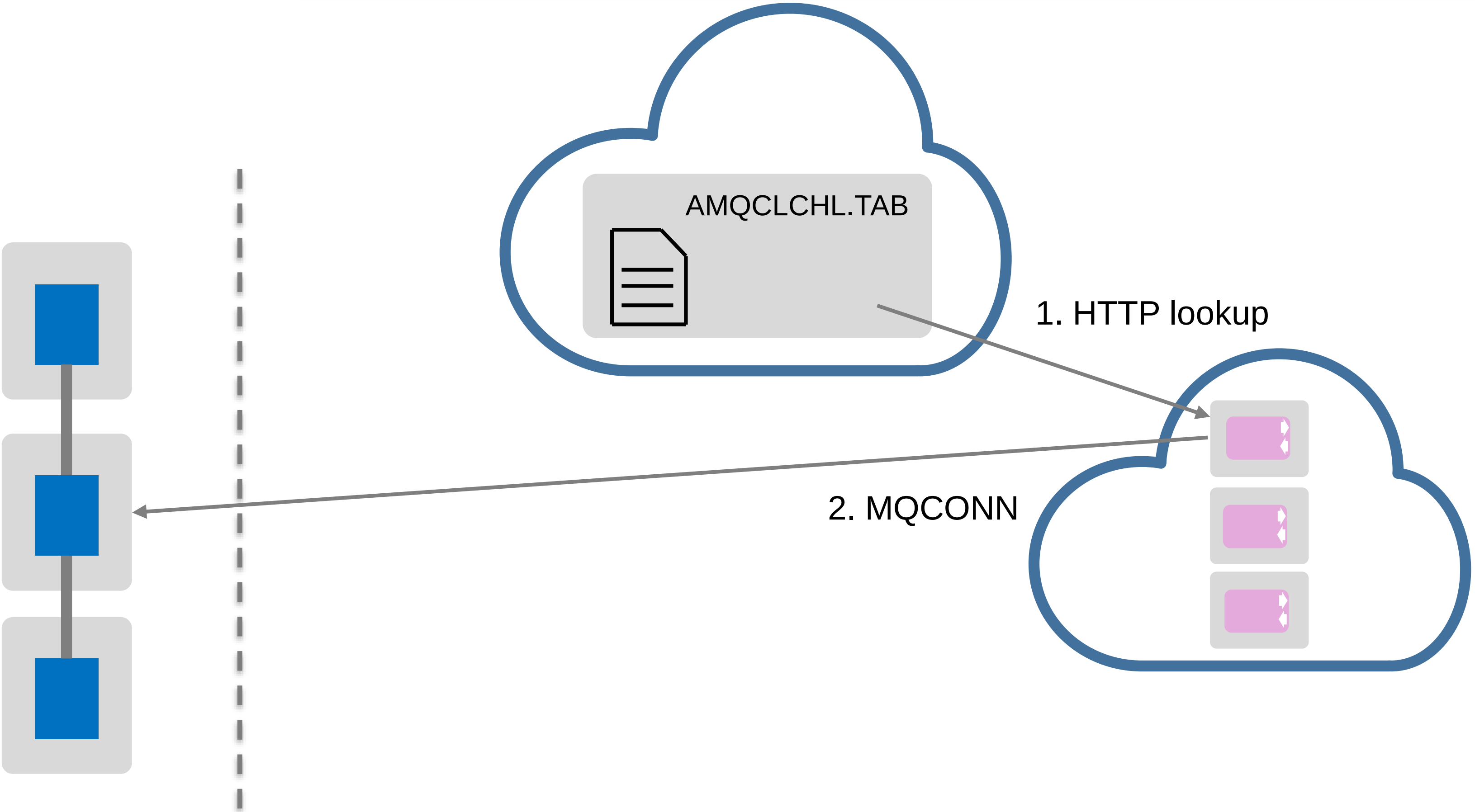


CCDT retrieval over HTTP

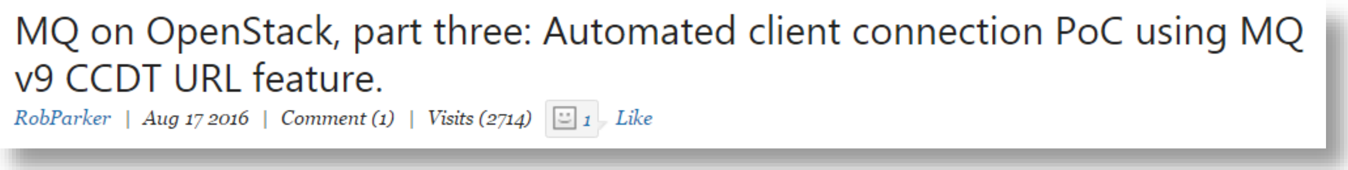


MQ on OpenStack, part three: Automated client connection PoC using MQ v9 CCDT URL feature.

RobParker | Aug 17 2016 | Comment (1) | Visits (2714) | 1 Like

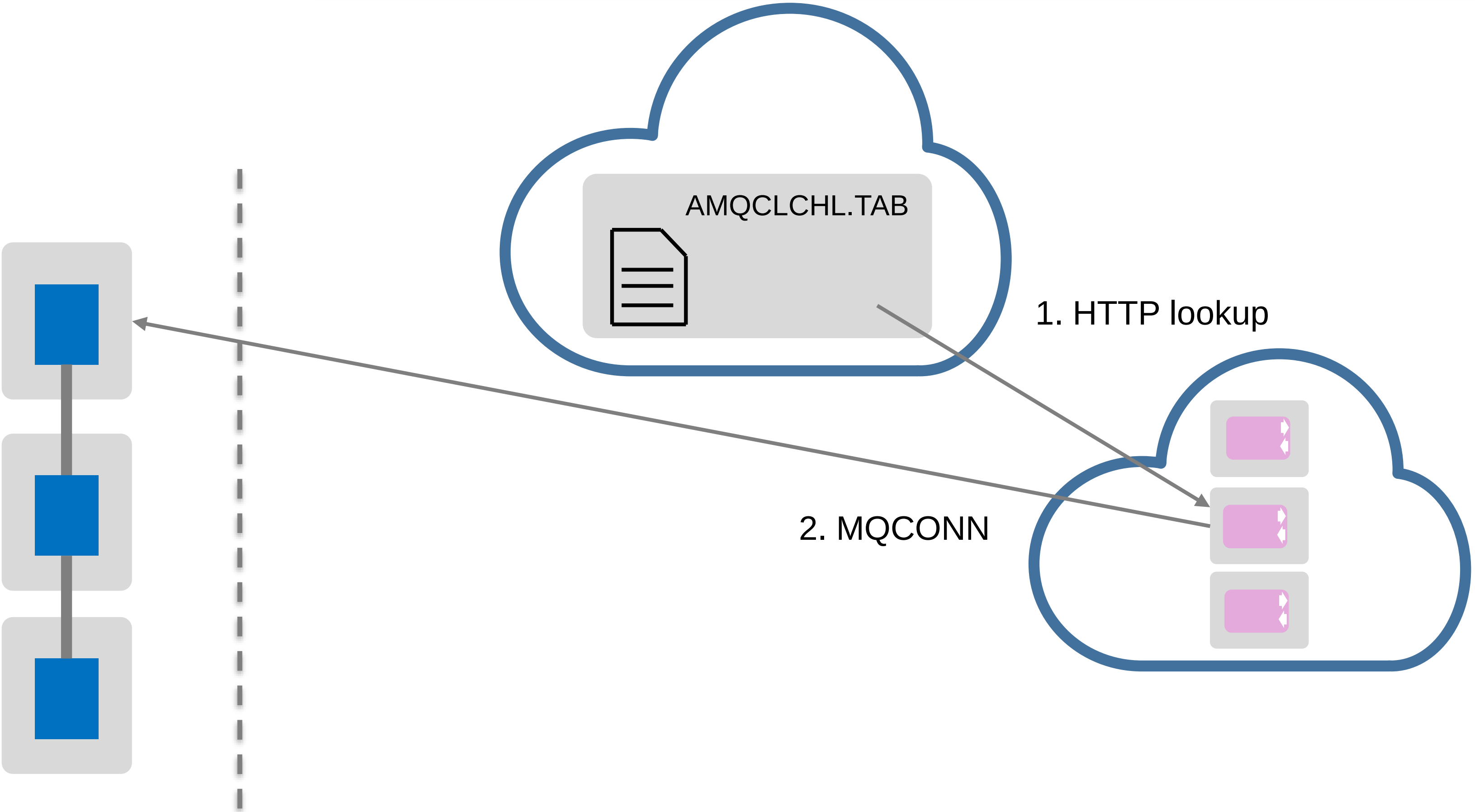


CCDT retrieval over HTTP

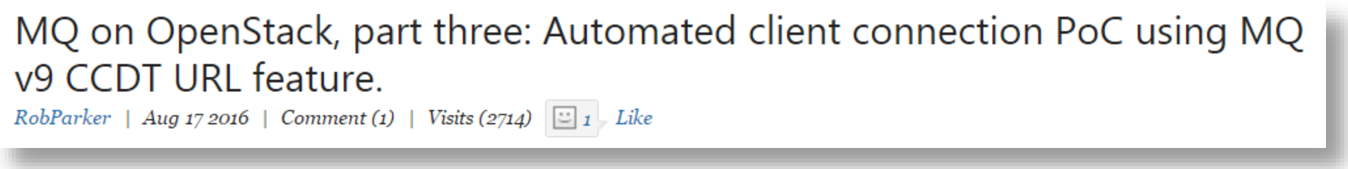


MQ on OpenStack, part three: Automated client connection PoC using MQ v9 CCDT URL feature.

RobParker | Aug 17 2016 | Comment (1) | Visits (2714) | 1 Like

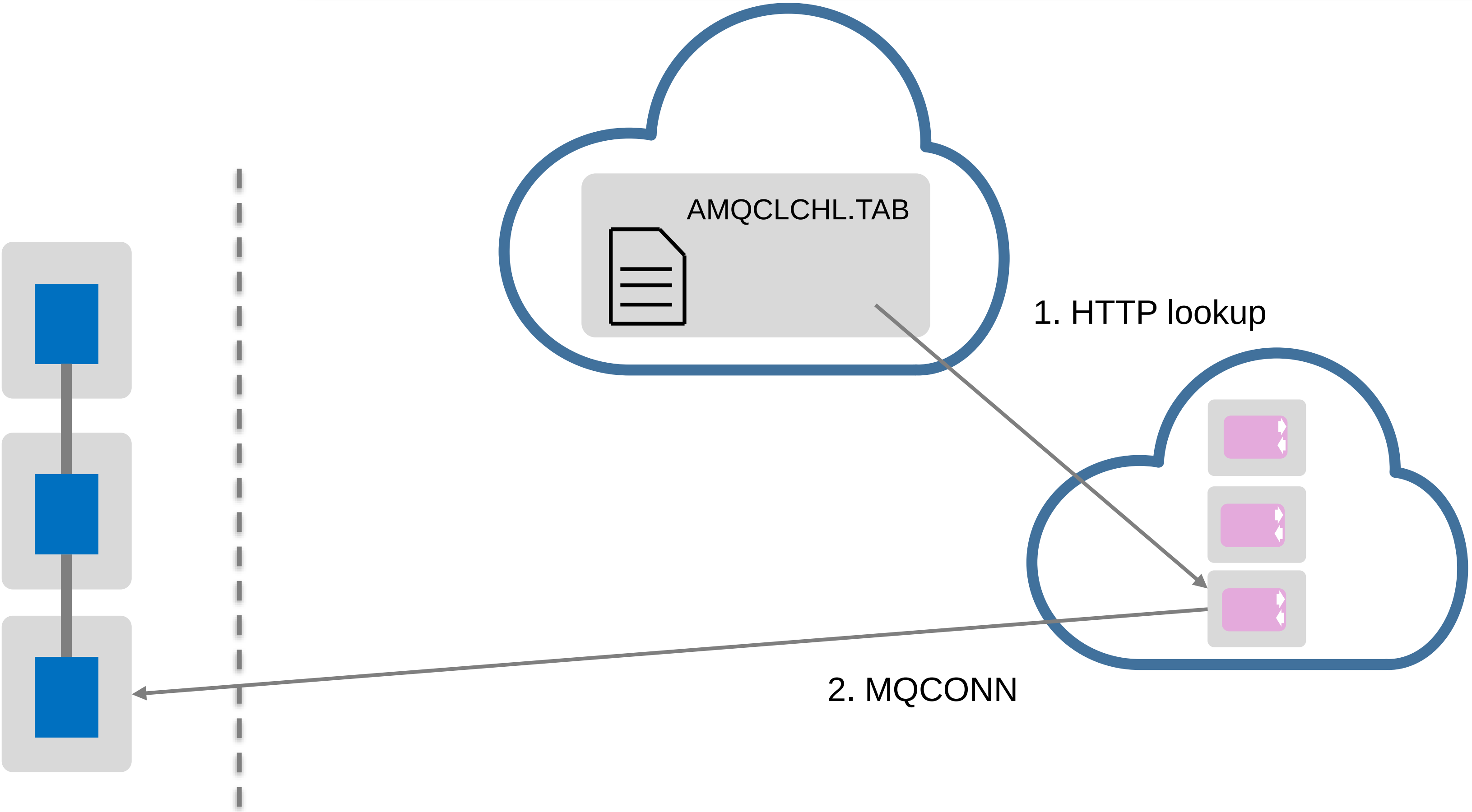


CCDT retrieval over HTTP



MQ on OpenStack, part three: Automated client connection PoC using MQ v9 CCDT URL feature.

RobParker | Aug 17 2016 | Comment (1) | Visits (2714) 1 Like

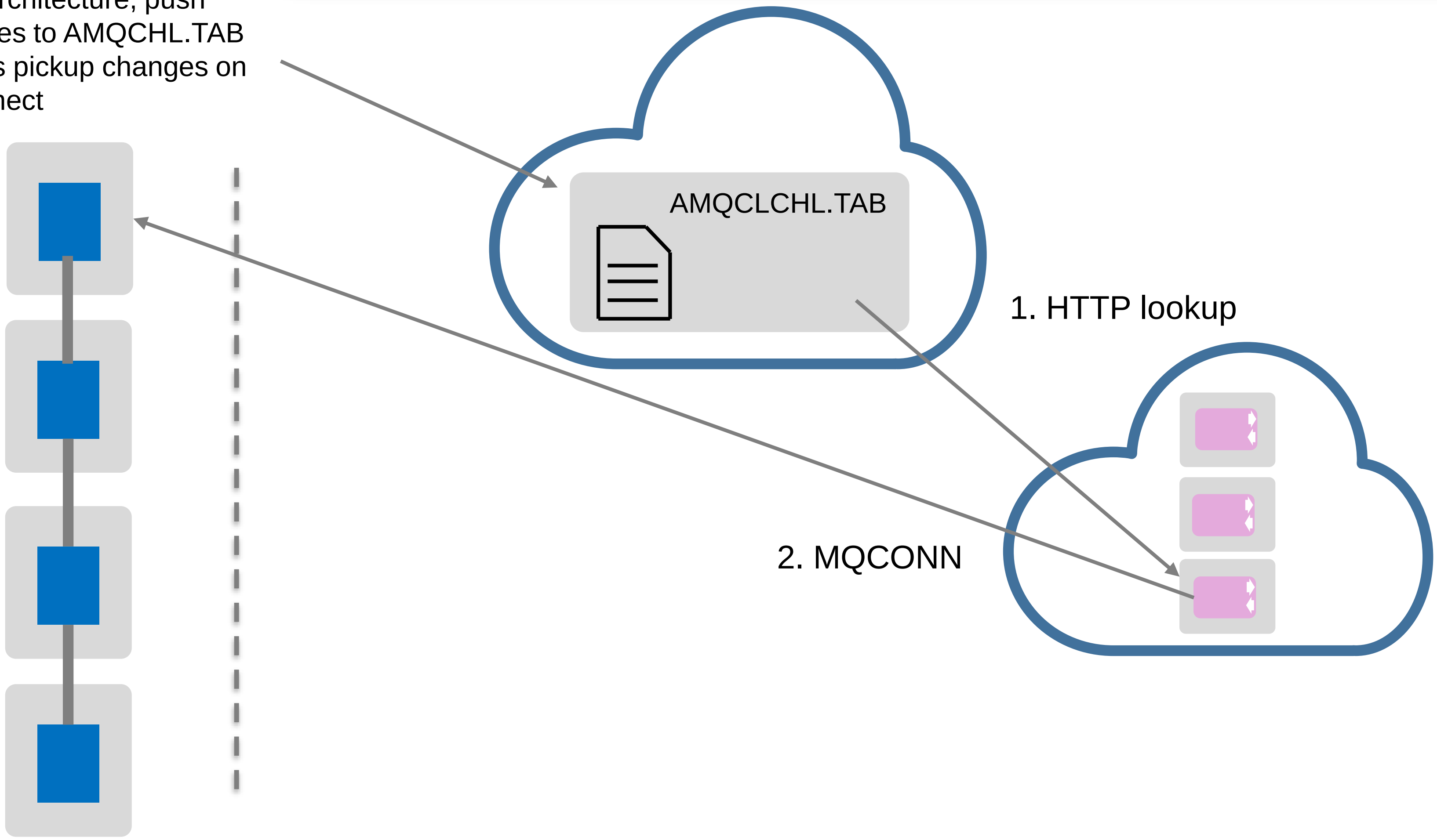


CCDT retrieval over HTTP

MQ on OpenStack, part three: Automated client connection PoC using MQ v9 CCDT URL feature.

RobParker | Aug 17 2016 | Comment (1) | Visits (2714) | 1 Like

- When you need to change your architecture, push changes to AMQCHL.TAB
- Clients pickup changes on reconnect



Thank You - Questions?

Related sessions:

- Running MQ in Containers
  - Tuesday 1.00pm (Leopardwood)

- The MQ on Cloud Service
  - Wednesday 1.00pm (Sagewood)

Please Note

IBM’s statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM’s sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user’s job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Trademark Statement

- IBM and the IBM logo are trademarks of International Business Machines Corporation, registered in many jurisdictions. Other marks may be trademarks or registered trademarks of their respective owners.
- Microsoft, Windows, Windows NT, and the Windows logo, Microsoft Azure, Microsoft Azure Functions, and .NET are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
- Python is a registered trademark of the Python Software Foundation. The Python logo is a trademark of the Python Software Foundation.
- Amazon Web Services, the “Powered by AWS” logo, AWS Lambda, AWS EC2, and the Amazon Web Services logo are trademarks of Amazon.com, Inc. or its affiliates in the United States and/or other countries.
- Google Cloud Platform, Google Functions, and the Google Cloud Platform logo are registered trademarks of Google Inc., U.S.
- Cloud Foundry is a registered trademark of The CloudFoundry.org Foundation, Inc.
- Other company, product and service names may be trademarks, registered marks or service marks of their respective owners.
- Docker and the Docker logo are trademarks or registered trademarks of Docker, Inc. in the United States and/or other countries. Docker, Inc. and other parties may also have trademark rights in other terms used herein.
- References in this publication to IBM products and services do not imply that IBM intends to make them available in all countries in which IBM operates.