Microsoft

Azure

Azure Service Bus

The Architect's Cut

Clemens Vasters
Chief Messenger, @clemensv

Patterns

Features

Architecture

Internals

Agenda

“What is Azure Service Bus?”

Patterns and Features

JMS 2.0, Azure SDKs, product-neutral AMQP stacks

Deep(er) Dive

Clusters and Namespaces

Networking Capabilities

Authorization Models

Log Storage

Message Broker Compute Model

Q&A

What is Azure Service Bus?

Platform-as-a-Service Queue and PubSub Message Broker

Fully managed: You use the features, Azure deals with everything else

JMS 2.0 and AMQP 1.0 standards compliant

Polyglot Azure SDK and cross-platform client support

Industry-leading reliability and availability

SKUs and Pricing

Azure Service Bus *Standard*:

Very cheap consumption-based pricing model

- $9.72 monthly base charge per Azure subscription for up to 50 namespaces
- 13M ops/month free, 13-100M ops/month $0.80 per M, 100-2500 M ops/month $0.50 per M, 2500M+ ops/month $0.20 per M.
- Service Bus *Basic* is a queues-only limited version of *Standard, $0.05 per M ops*.

Shared-resources model

Azure SDK, AMQP clients, supports JMS 1.1/2.0 queue API for send and receive.

Azure Service Bus *Premium*:

Capacity-based pricing model

- $0.928/hour or $677.08/month per **Messaging Unit**
- No further per-operation charges

Isolated-resources model

Azure SDK, AMQP clients, supports the full JMS 2.0 feature set

As you'll learn today, these are different service implementations behind the scenes

You're running a JMS 2.0 broker cluster in your own datacenter?

IBM? TIBCO? Red Hat? VMWare?

Azure Service Bus is

more reliable and dependable with higher 24/365 uptime
(no service windows, no weekend downtime)

far less costly to own
(no initial/recurring licensing fees)

far less costly to operate
(fully managed, no hardware)

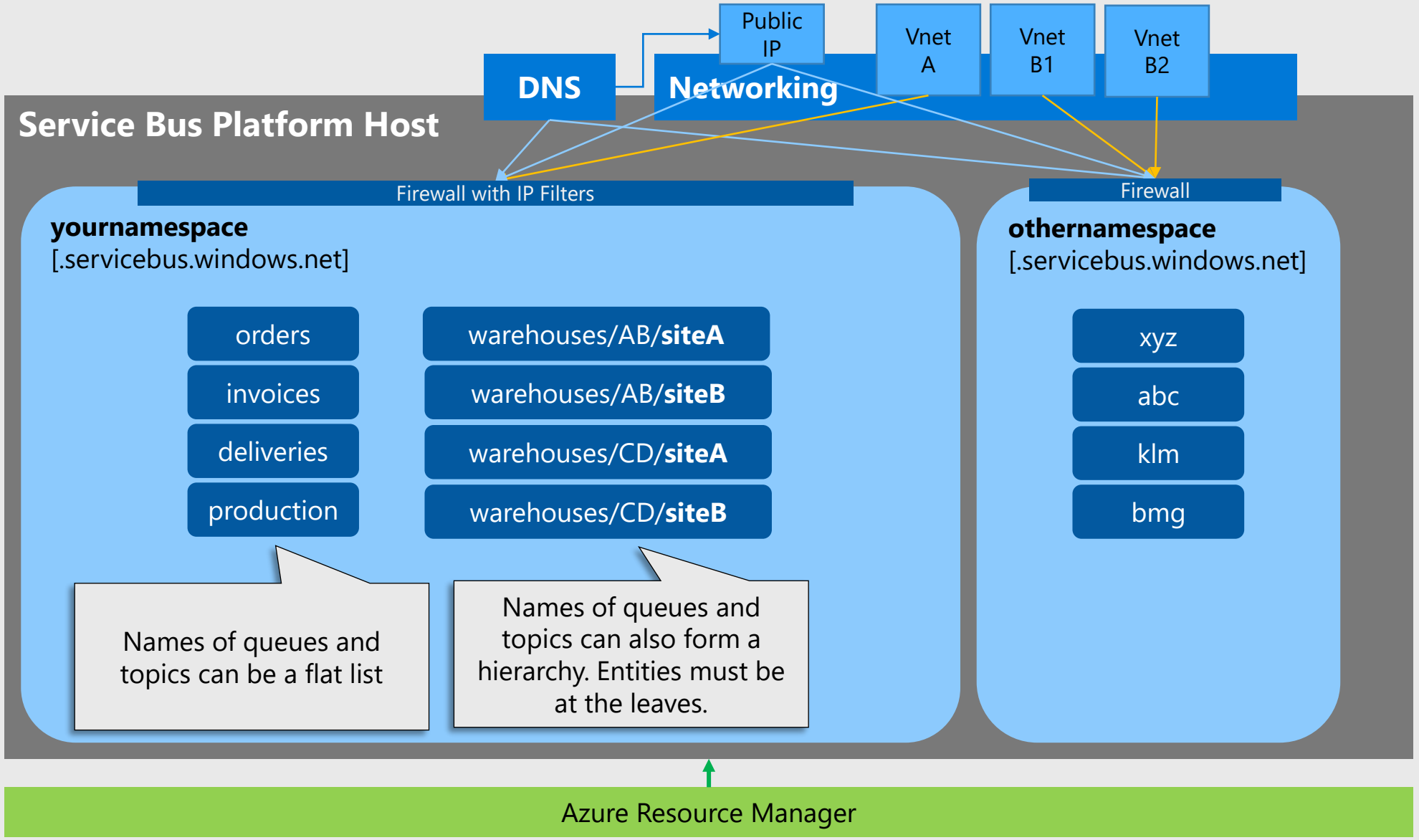
just a quick network hop away from your existing workloads

easy to integrate into your network via VPN Gateway or Express Route

Azure Service Bus

Microsoft Azure

Service Bus Namespaces: Virtual Brokers



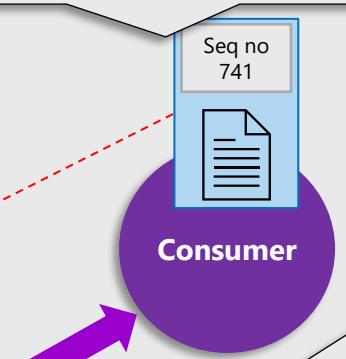
Service Bus Queues (I)

There can be up to 5000 concurrently competing consumers per entity.

First come, first served.

Send: Add sequence number & server timestamp, write/flush to log, then report as accepted

Sequential log, ordered by sequence number (order of arrival)

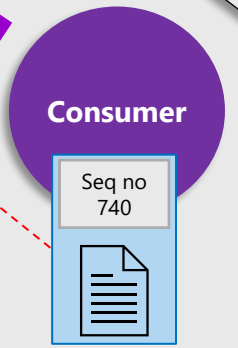


Acquired messages are locked **exclusively** to the acquirer (consumer) until they are settled or the lock expires*

transfer

accept

transfer



Consumer tells the broker the state of the message.

accepted: Accepted and handled. Can be removed, must not be redelivered.

released: Not handled. Should be redelivered.

rejected: Not handled. Considered defective. Must not be redelivered.

Max message size:
Std: **Up to 1MB**
Prm: **Up to 100 MB**

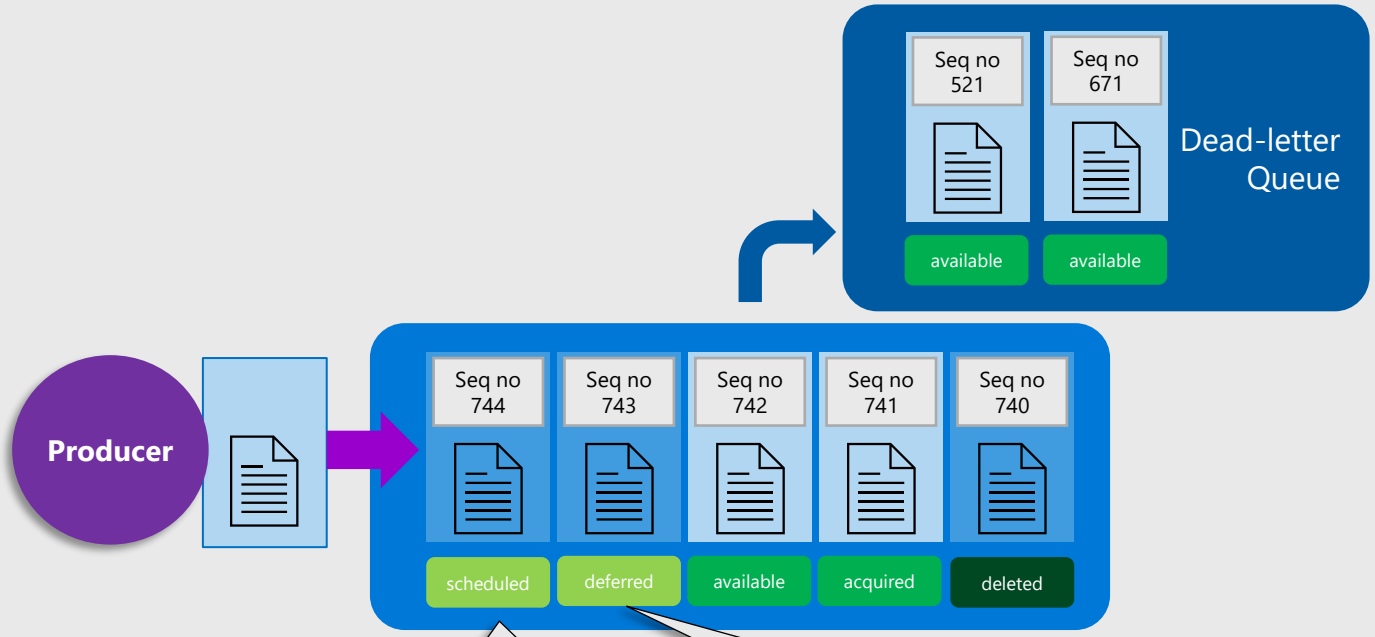
Max queue size:
Up to 80GB

Oldest available message is the next to be acquired and delivered.

In Premium, using JMS, consumers can skip to the first available message that fits a certain criterion (message selector) without head-of-line blocking.

* locks always have a timeout and instantly expire when the connection breaks. (As of 3/22, might change ;)

Service Bus Queues (II)



Every queue and topic subscription has its own **dead-letter subqueue**: myqueue/$deadletterqueue

All messages that cannot be delivered (released more often than permitted) or have been rejected and optionally those that have expired and put into the dead-letter queue and remain there.

The reason for why and from where the message has been moved is remarked in the **DeadLetterReason**, **DeadLetterErrorDescription**, and **DeadLetterSource** properties.

The deadletter queue is otherwise a normal queue and can be used with the normal receiver clients.

Scheduled messages have been accepted and stored in the queue log, but they are made available for delivery only after **ScheduledTimeUtc**.

Scheduled messages *can be canceled* using the sequence number returned by the scheduling API.

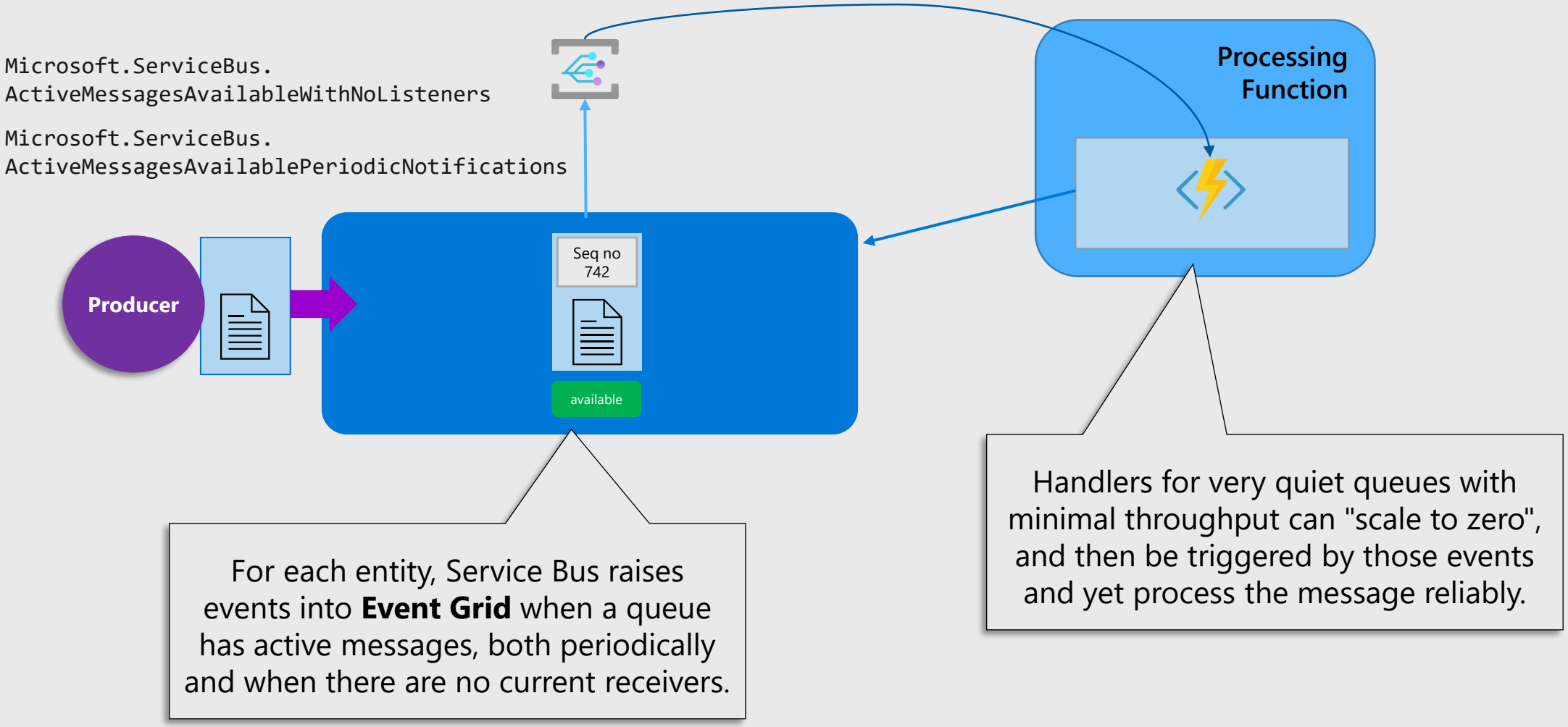
Scheduled messages are re-sequenced and re-timestamped as their state changes to available.

Deferred messages have been delivered to a consumer at least once and the consumer has decided to set them aside instead of settling them.

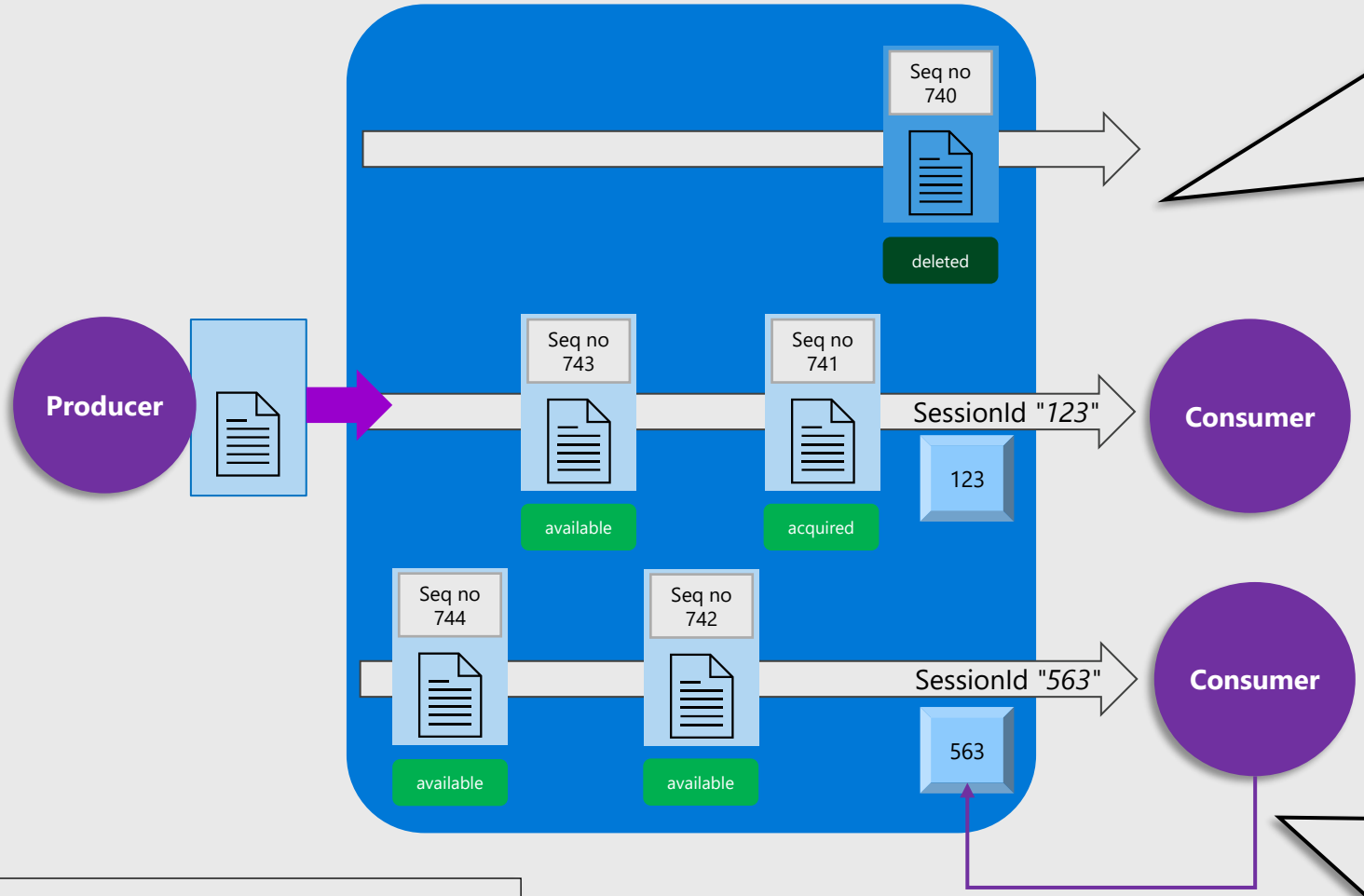
Deferred messages remain in the log but are not eligible for delivery until they are restored into the available state.

Deferral is a special feature for state workflow engines that expect to handle messages in a particular sequence.

Service Bus Queues (III)



Service Bus Queues – Sessions



Sessions are unlimited, unbounded, ordered sequences of messages that are all labeled with the same **SessionId**. Using sessions requires a session-enabled queue or topic subscription.

Consumers acquire exclusive locks on the entire session and all messages belonging to it.

An active session behaves like an independent queue, without messages from other sessions blocking progress (no "head of line blocking")

A single queue can maintain hundreds of thousands of sessions and thus logical sub-queues.

The owning consumer of a session has read/write access to the **session state** that the broker optionally maintains. Session state can hold data up to the max size of a message permitted for the queue.

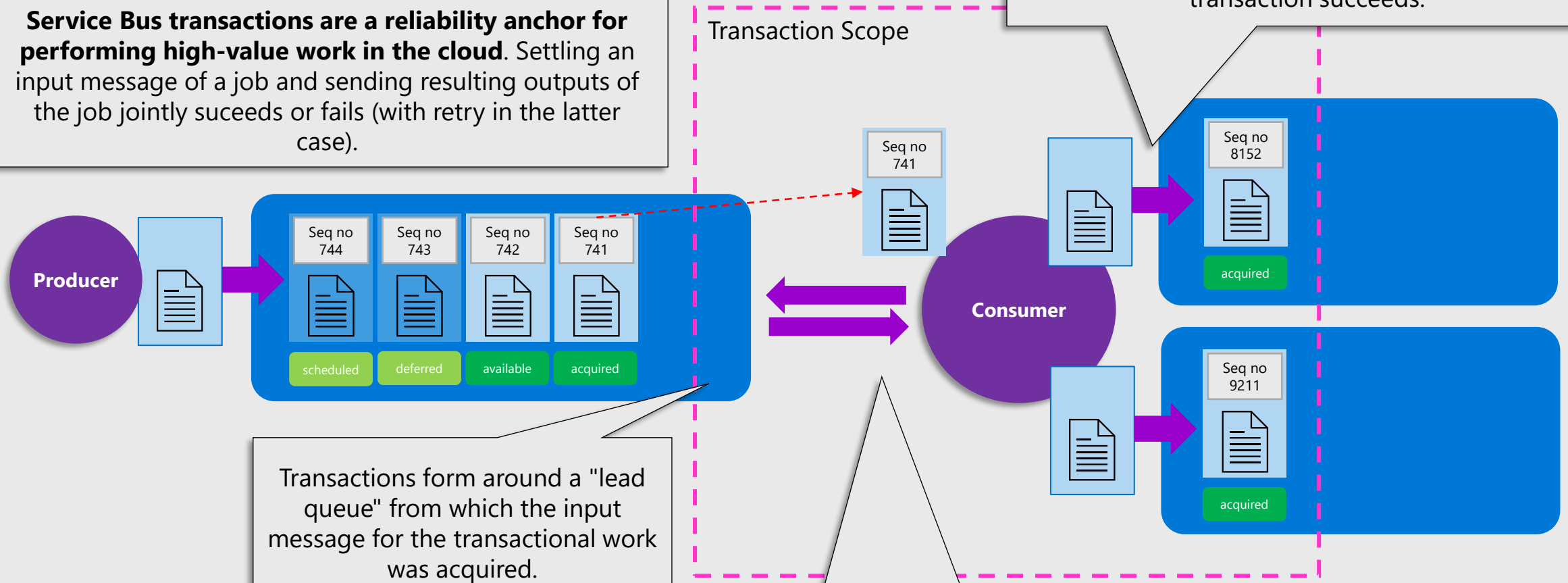
When a new consumer acquires the session, it can therefore access all info left by a prior consumer related to processing the session.

Built for long-running workflow processes.

Service Bus Queues – Tx

Service Bus transactions are a reliability anchor for performing high-value work in the cloud. Settling an input message of a job and sending resulting outputs of the job jointly suceeds or fails (with retry in the latter case).

A transactional operation can send messages to any number of output queues or topics as a result of the transactional work. Sends are held until the transaction succeeds.



Transactions form around a "lead queue" from which the input message for the transactional work was acquired.

Only one receiving queue can be enlisted in a transaction. Sessions are supported.

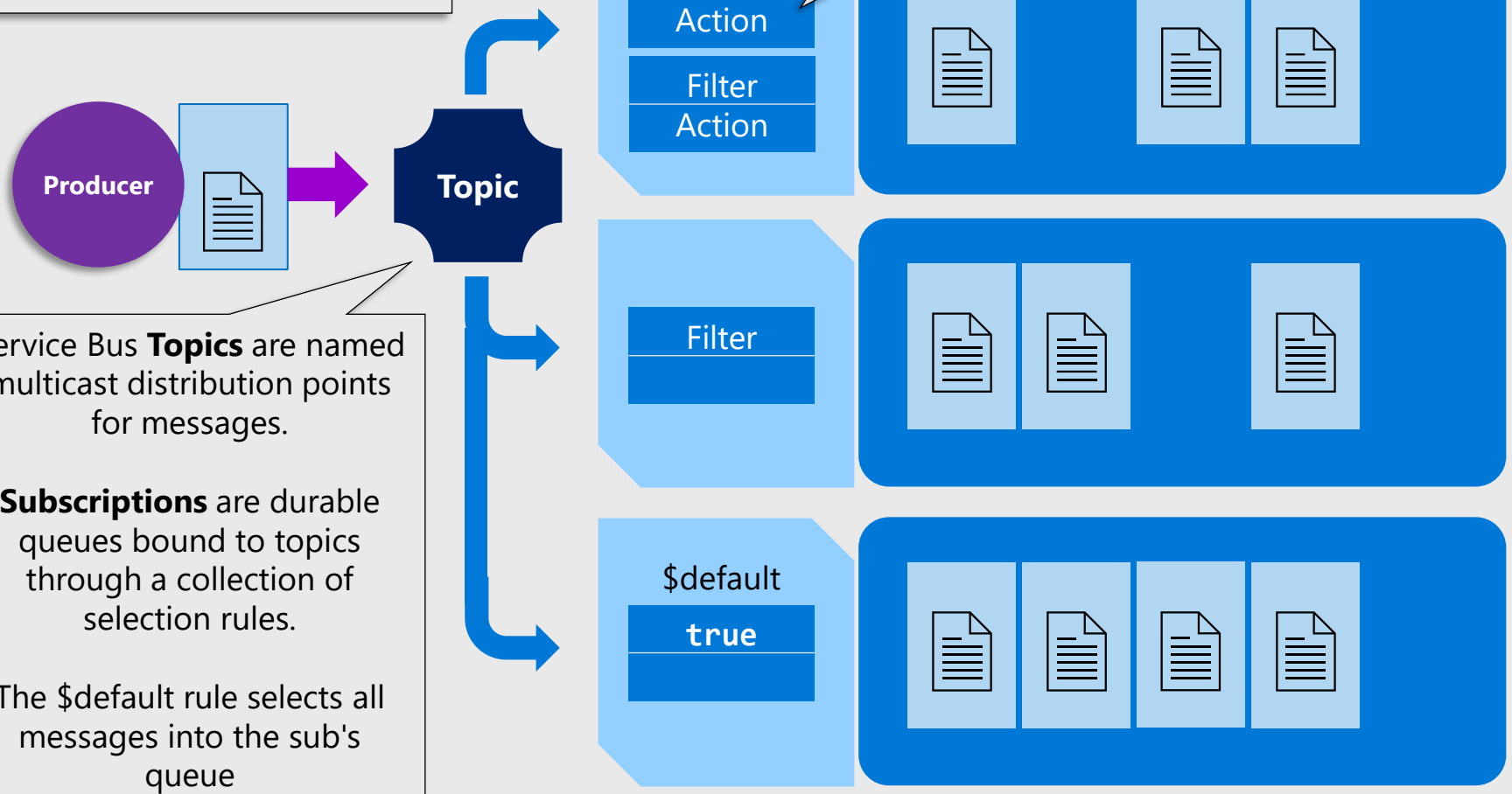
All settlement operations on the lead source queue are transactional: accepting (complete), releasing, rejecting (deadletter), deferral, session state ops.

Service Bus Topics

For consumers, subscription queues have all features of queues, including a dead-letter queue. They can also support sessions.

Actions can add, remove, and edit all application message properties and system properties like TimeToLive or SessionId.

One subscription can have up to 2000 (!) rules. Each filter match yields a copy of the tested message.



Correlation filters match properties against values ("equals").

SQL filters allow WHERE-condition-style SQL expressions against the app and platform message properties.

```
sys.To LIKE
'region/DE/shops/%' AND
subject='catalogUpdate'
```

Service Bus **Topics** are named multicast distribution points for messages.

Subscriptions are durable queues bound to topics through a collection of selection rules.

The $default rule selects all messages into the sub's queue

JMS clients can use shared, unshared, durable and volatile subscriptions, with JMS-compliant SQL message selectors.

SDK Support: JMS 2.0 Provider

Azure Service Bus is fully conformant with Java JMS 2.0 (Jakarta Messaging)

```
Topic topic = session.createTopic(dest.toString());

MessageProducer sender = session.createProducer(dest);

TopicSubscriber topicSubscriber1 =
    session.createDurableSubscriber(topic, sub1Name);

TopicSubscriber topicSubscriber2 =
    session.createDurableSubscriber(topic, sub2Name, "JMSCorrelationID='5'", false);

MessageConsumer topicSubscriber3 =
    session.createSharedDurableConsumer(topic, sub3Name, "JMSCorrelationID='5'");
```

```
TextMessage message = session.createTextMessage("Text!");

message.setJMSCorrelationID(Integer.toString(i));

sender.send(message, DELIVERY_MODE, Message.DEFAULT_PRIORITY,
Message.DEFAULT_TIME_TO_LIVE);
```

- Queues
- Topics
- JMS Message Selectors
- Shared Durable Subscriptions
- Unshared Durable Subscriptions
- Shared Volatile Subscriptions
- Unshared Volatile Subscriptions
- Queue Browser
- Queue Message Selectors
- Temporary Queues
- Temporary Topics
- JMS Transactions

SDK Support: Azure SDKs

.NET

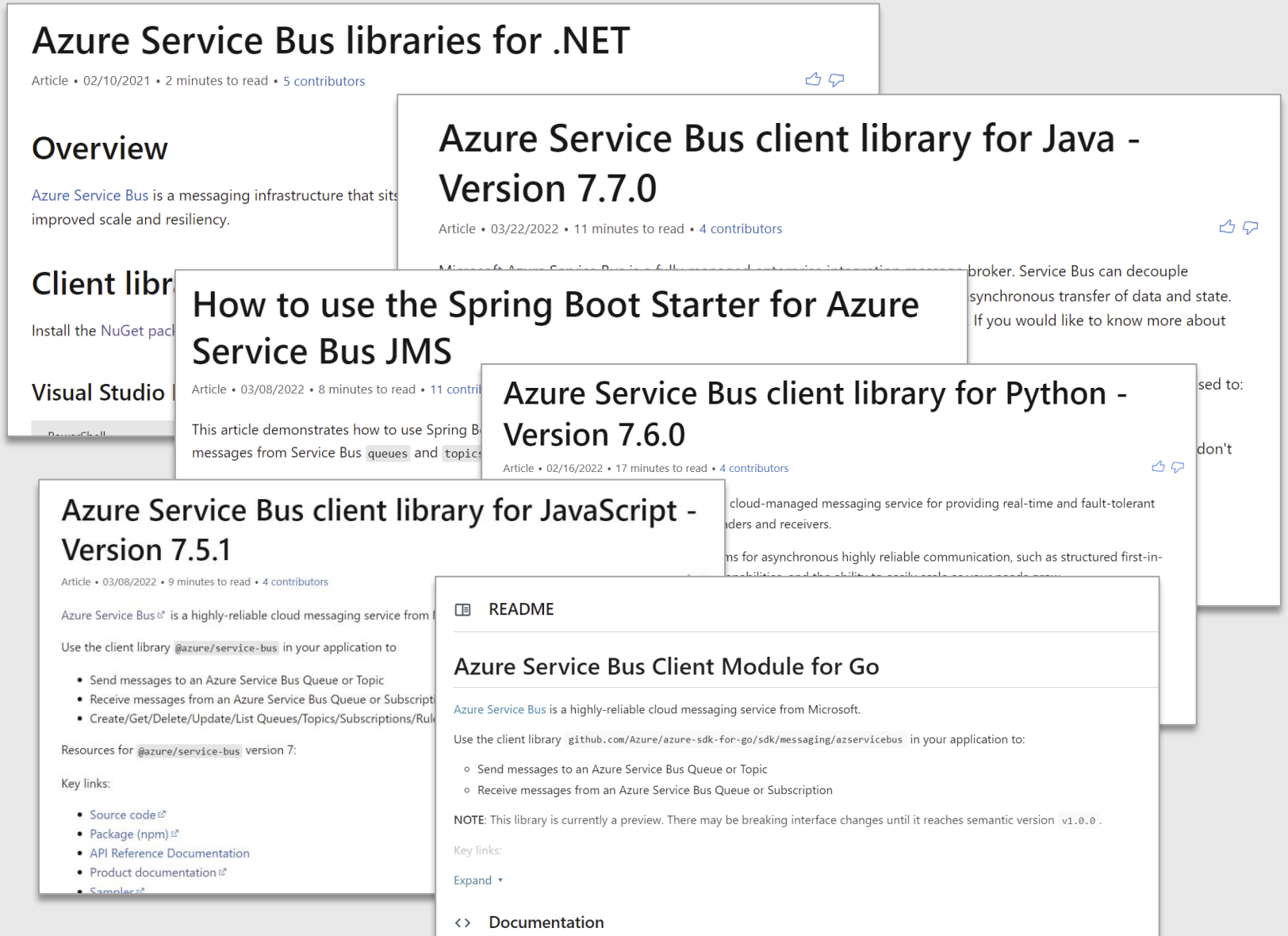
Java

Java Spring Boot

Python

JavaScript/Typescript

Go



Compatible Generic AMQP 1.0 Clients

Client Stack	URL
Apache Qpid Proton-C (Go, Python, Ruby, C++)	https://qpid.apache.org/proton/
Apache Qpid Messaging API	https://qpid.apache.org/components/messaging-api
Apache Qpid Proton-J	https://qpid.apache.org/proton/
Apache NMS AMQP	https://activemq.apache.org/components/nms/providers/amqp/
AMQP .NET Lite (all variants of .NET, incl. Nano & Micro)	https://github.com/Azure/amqpnetlite
Azure AMQP (our own server stack)	https://github.com/Azure/azure-amqp
Azure uAMQP C (Python, PHP)	https://github.com/Azure/azure-uamqp-c https://github.com/Azure/azure-uamqp-python
Rhea (NodeJS)	https://github.com/amqp/rhea
Go AMQP	https://github.com/Azure/go-amqp
Vert.X AMQP Client	https://vertx.io/docs/vertx-amqp-client/java/

Azure Service Bus Deep Dive

Architecture

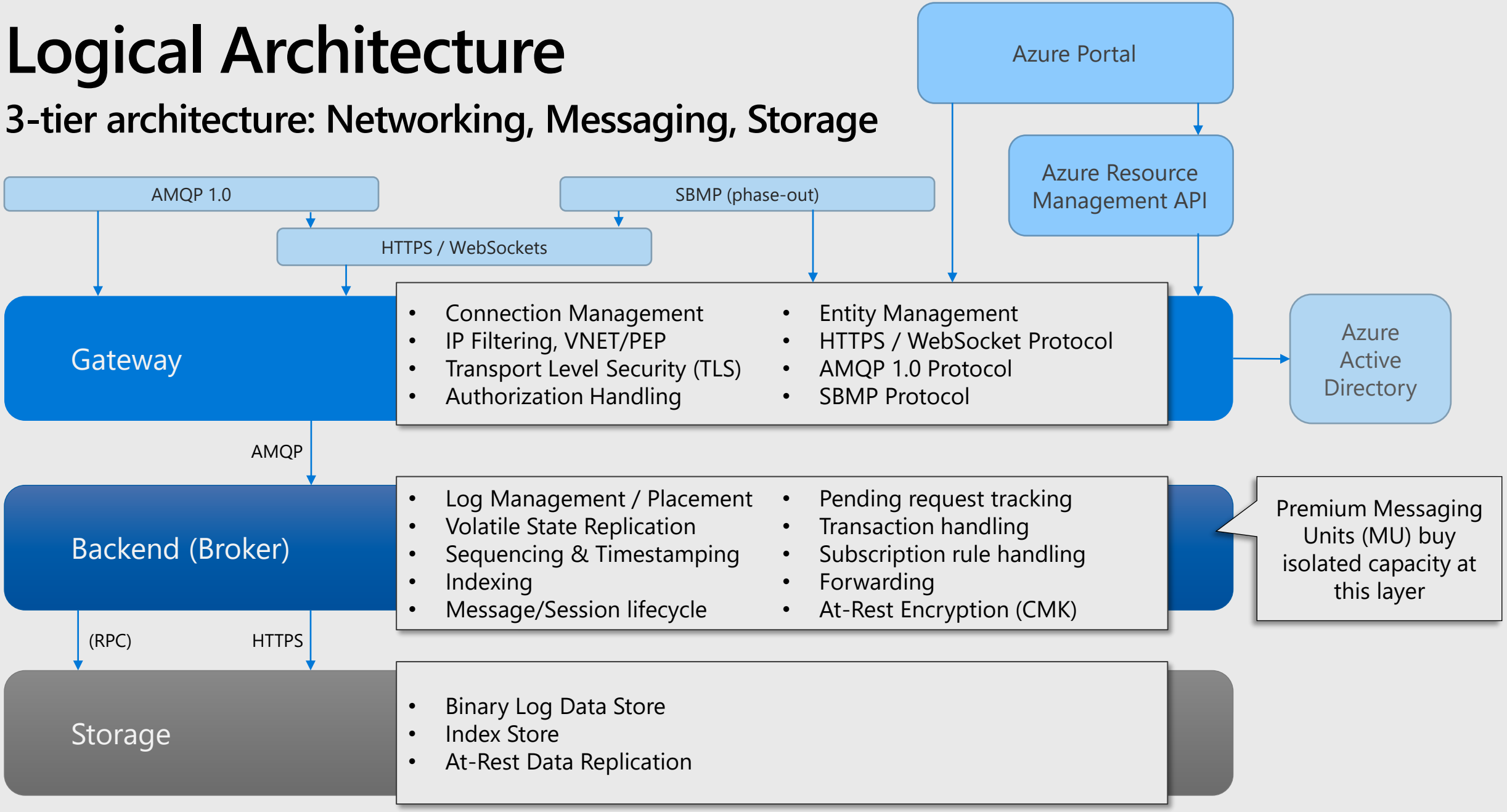
Internal Architecture

Protocols

Performance Metrics

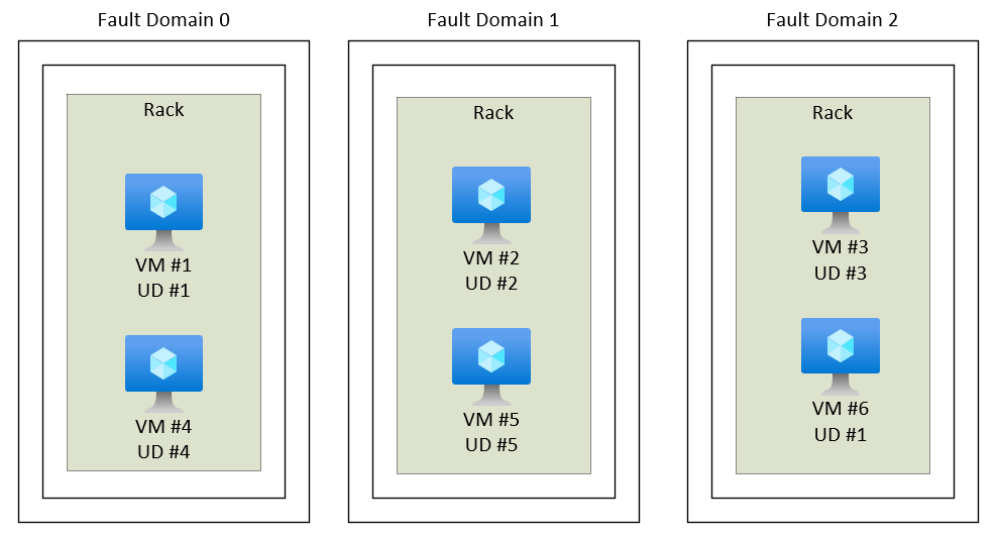
Logical Architecture

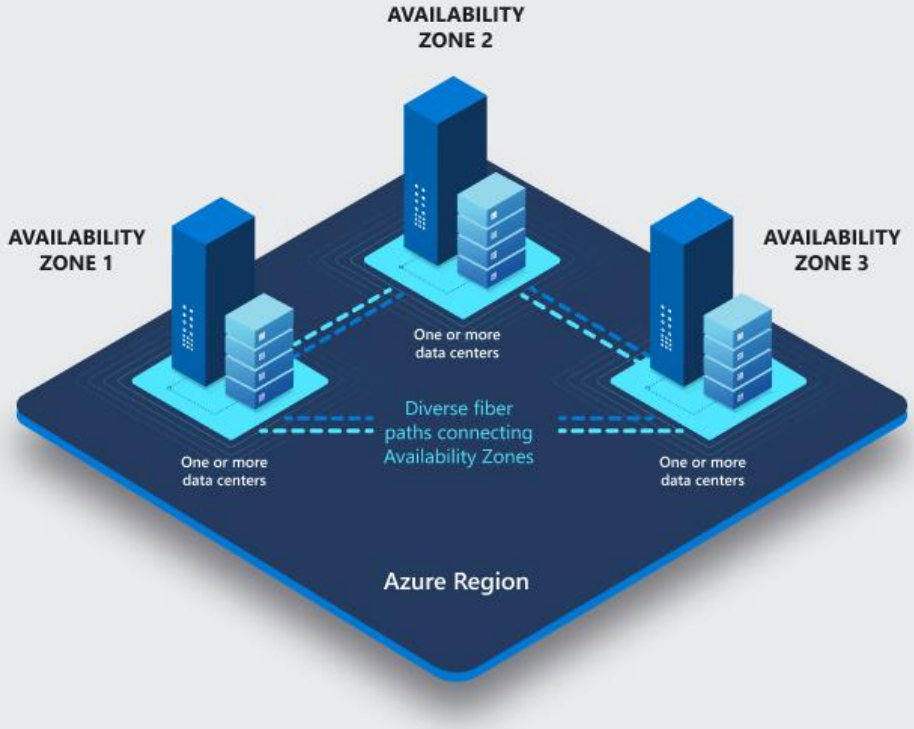
3-tier architecture: Networking, Messaging, Storage



Fault Domains & Availability Zones

High-Availability starts at the physical reality



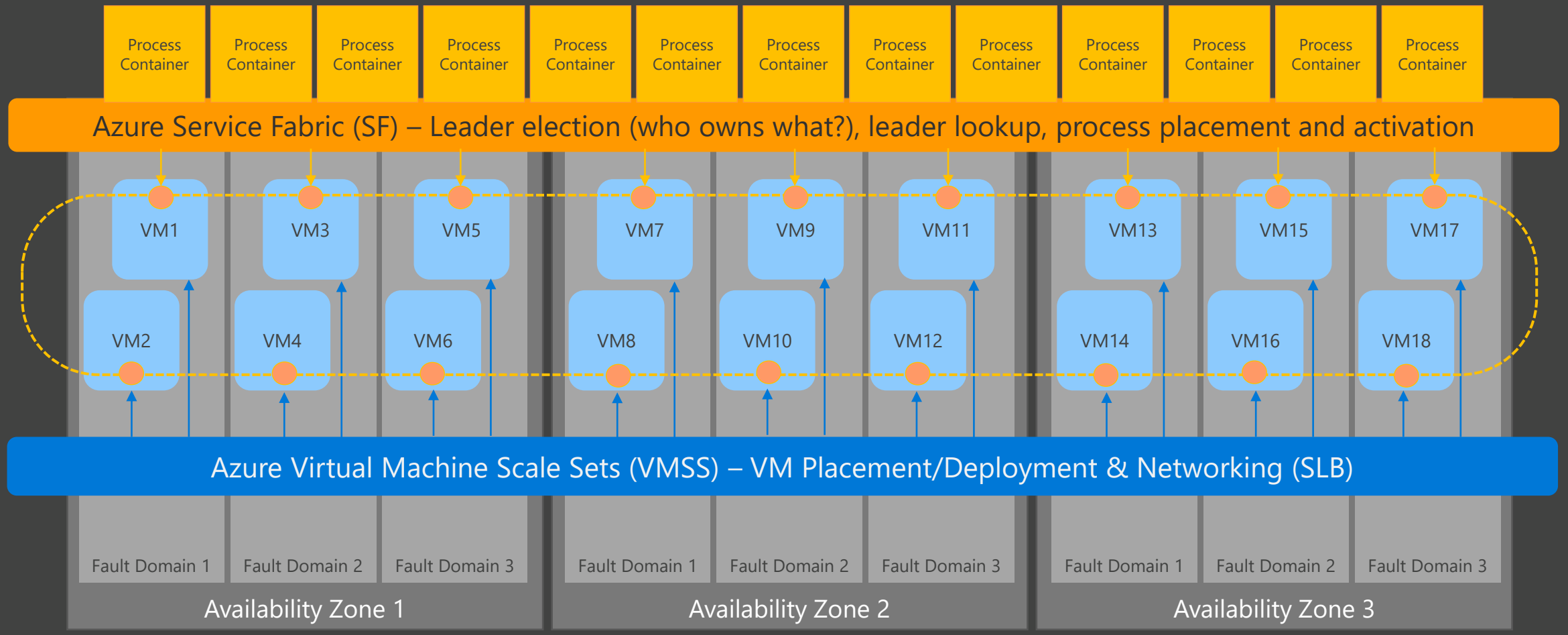


Fault Domain Placement:
Cluster VMs are spread across at least 3 fault domains such that the loss of a rack or network poses no availability risk. Recovery from a fault domain failure is fully automated and the system maintains SLA.

Availability Zones Placement:
Each cluster spans three availability zones and maintains SLA without any tolerance for data loss when one or two zones fail.

Backend and Gateway Clusters

Logical Architecture meets Placement



Backend and Gateway Clusters

Principles

Everything is automated. There's no manual intervention in placement of VMs or processes or workloads.

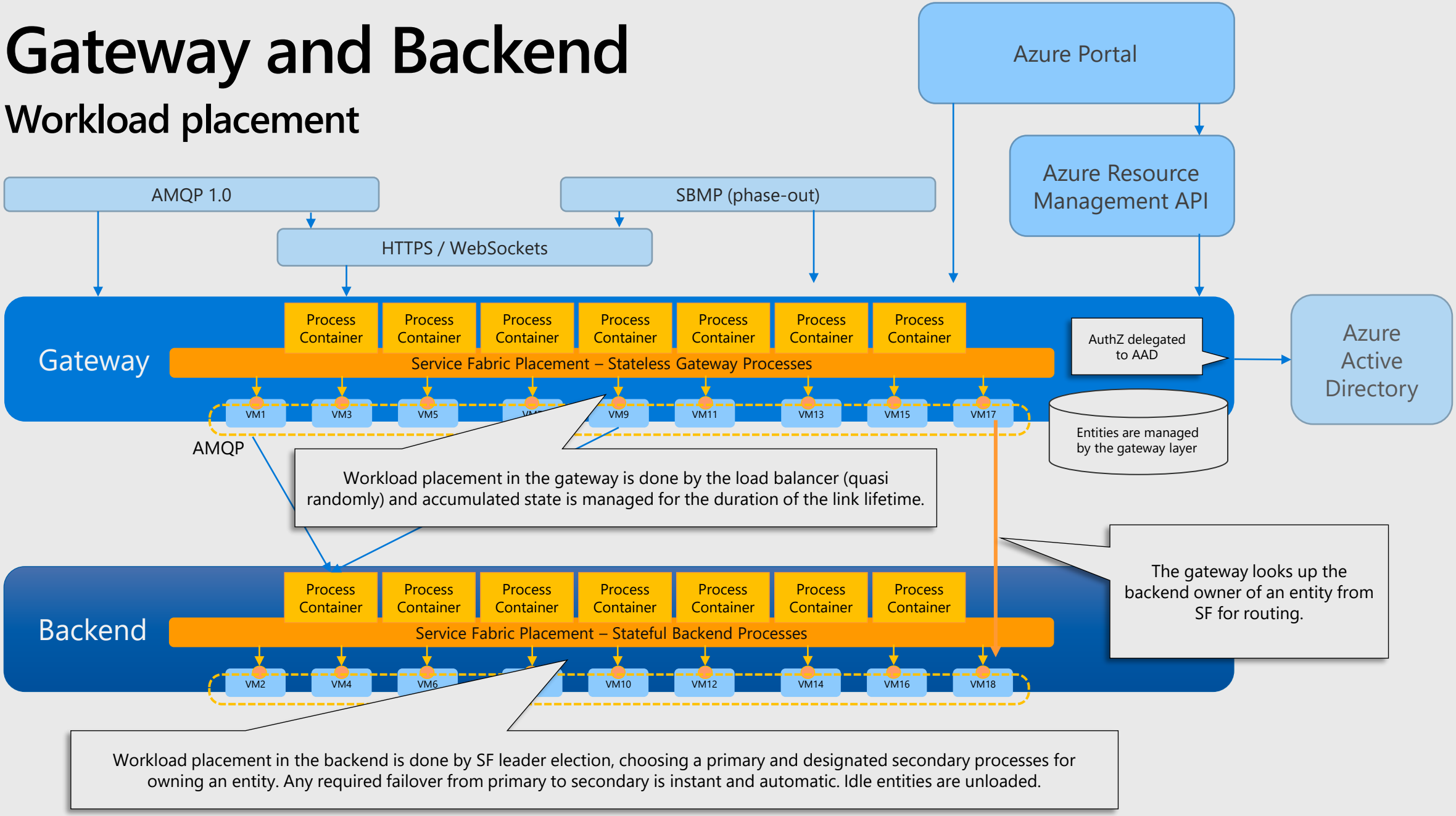
Process placement and servicing/upgrade cycles are controlled by Azure Service Fabric.

Each cluster has a well-known number of virtual machines. We don't use auto-scaling with VMs. We may (rarely) choose to initiate scale up/down to a different level.

Allocation of namespaces to clusters are based on well-established heuristics (Standard) or deterministic capacity limits (Premium).

Gateway and Backend

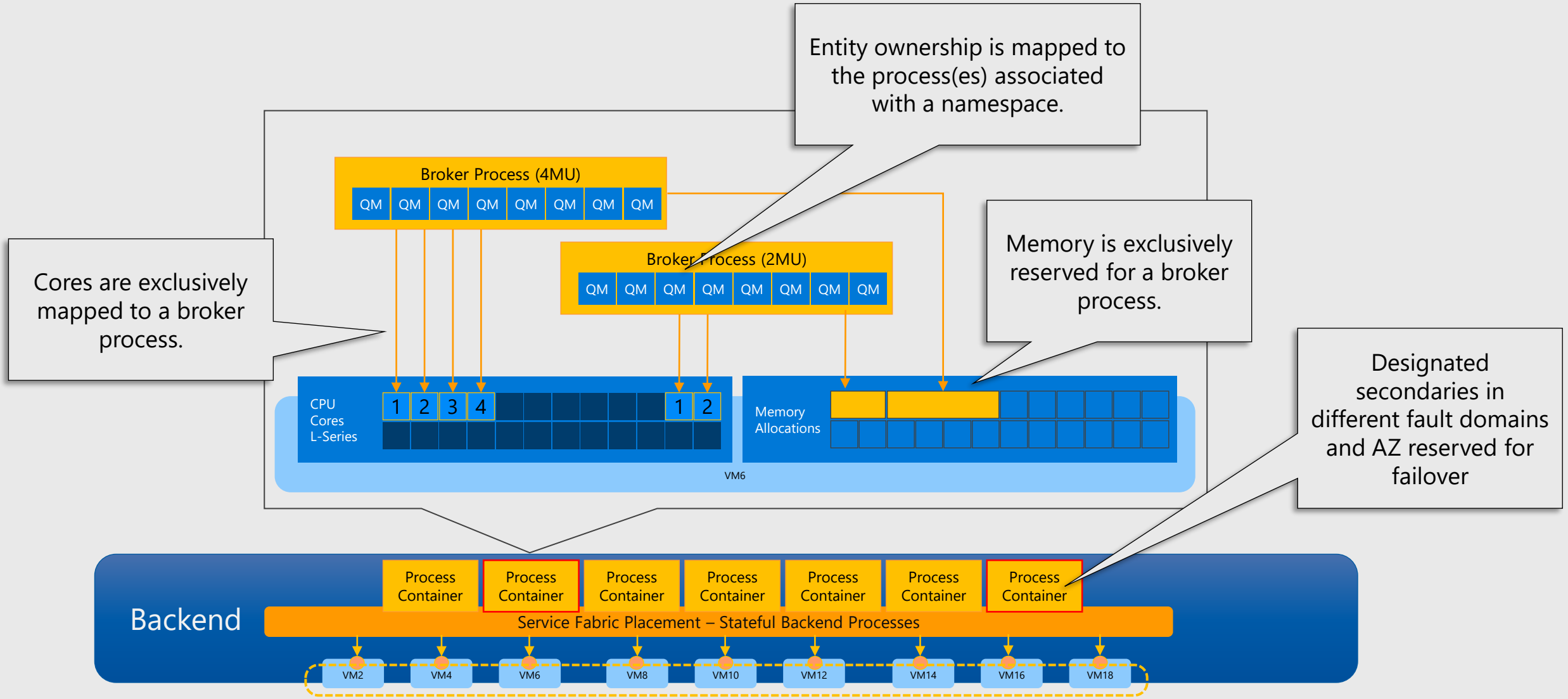
Workload placement



Backend: Service Bus Premium.

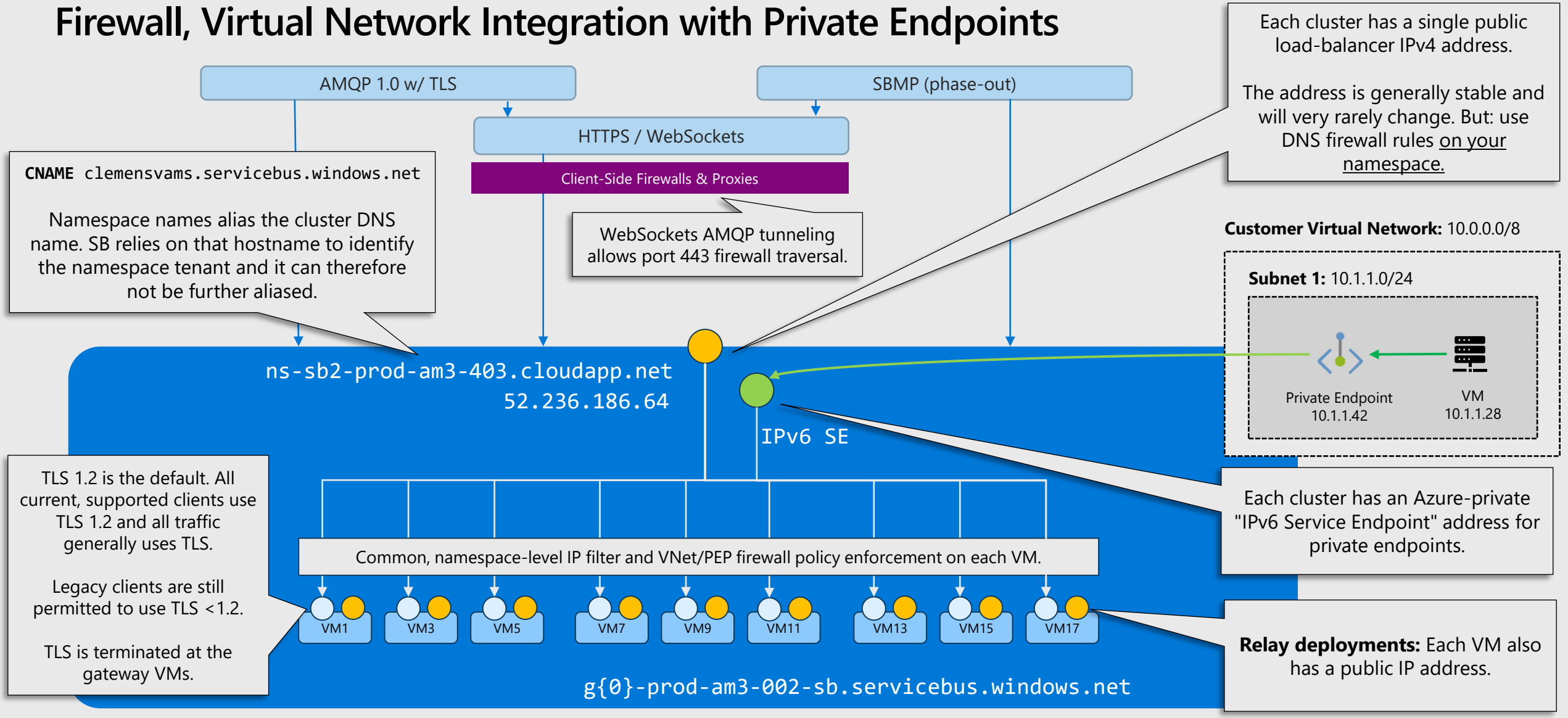
Workload placement

Log Management / Placement	Pending request tracking
Volatile State Replication	Transaction handling
Sequencing & Timestamping	Subscription rule handling
Indexing	Forwarding
Message/Session lifecycle	At-Rest Encryption (CMK)



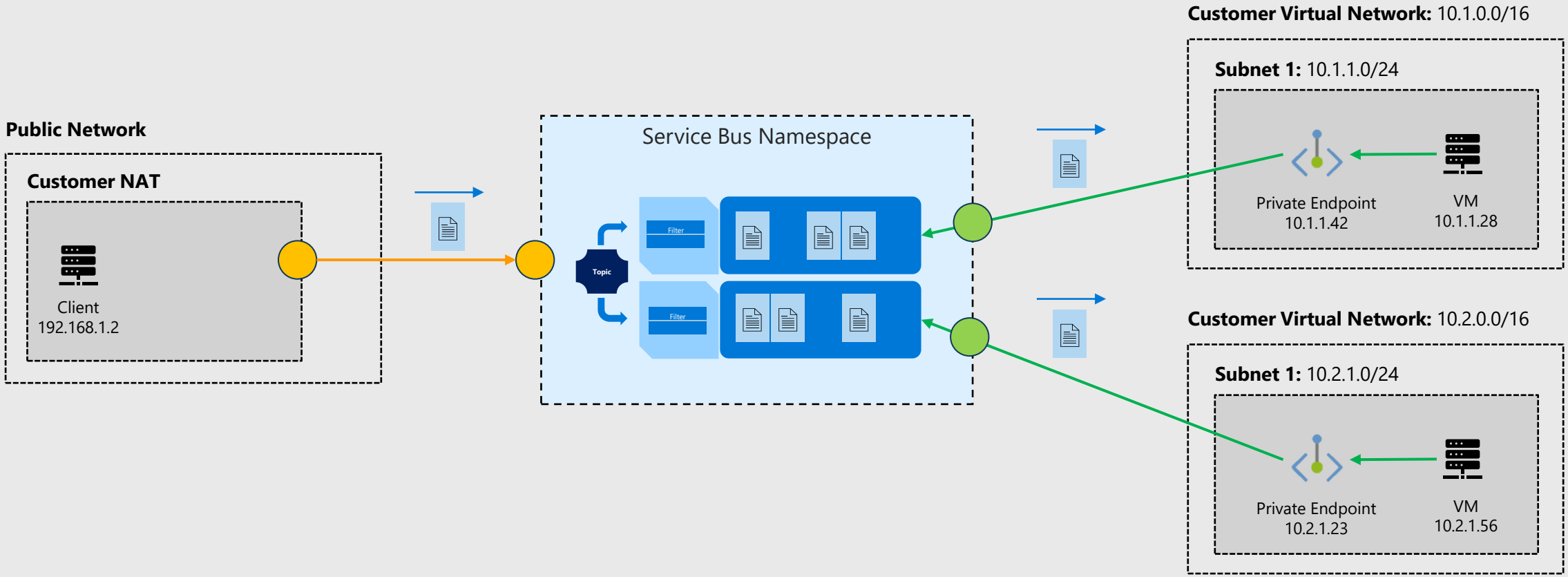
Networking Features

Firewall, Virtual Network Integration with Private Endpoints



Networking Features

Service Bus namespaces can be attached to one or more virtual networks and the public IP address space concurrently and act as safe "Layer 7" (app-level) routers.



Authorization

Azure Active Directory RBAC and Shared Access Signatures (SAS)

Service Bus has two authorization models.

SAS (Local): Simple model for external clients that cannot use Azure Active Directory for any reason.

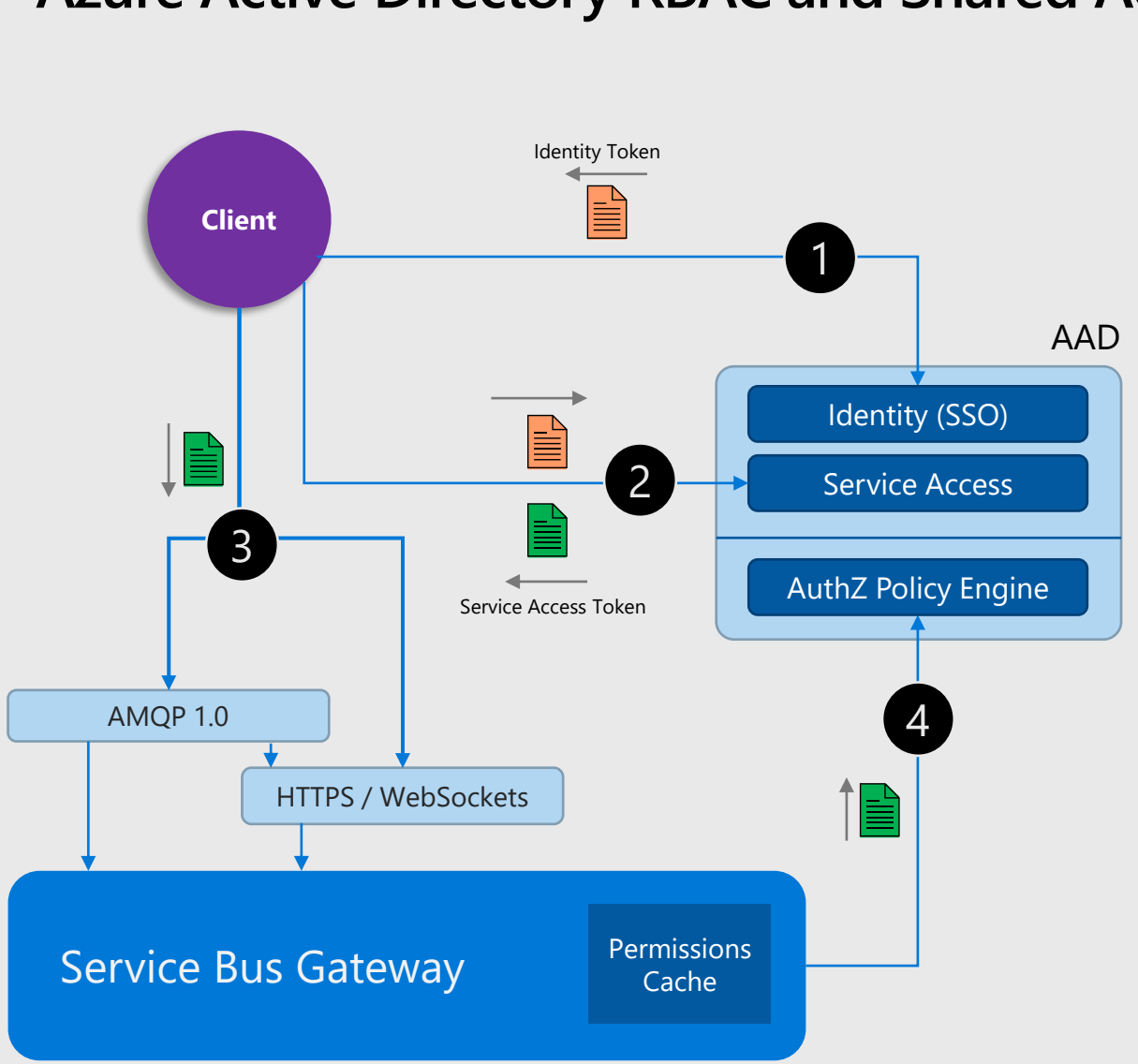
- Very simple. Rights at namespace or entity scope: "Send", "Listen" (Receive/Subscribe), "Manage"
- No accounts. 12 named rules per scope, each rule has a key. Rules combine a set of rights.
- HMACSHA256 signed **tokens** are issued using name and key and passed to the service.
- Tokens can be issued in the client or by some security token service holding the key
- *Can be turned off via the portal on the overview page under "Local Authentication"*

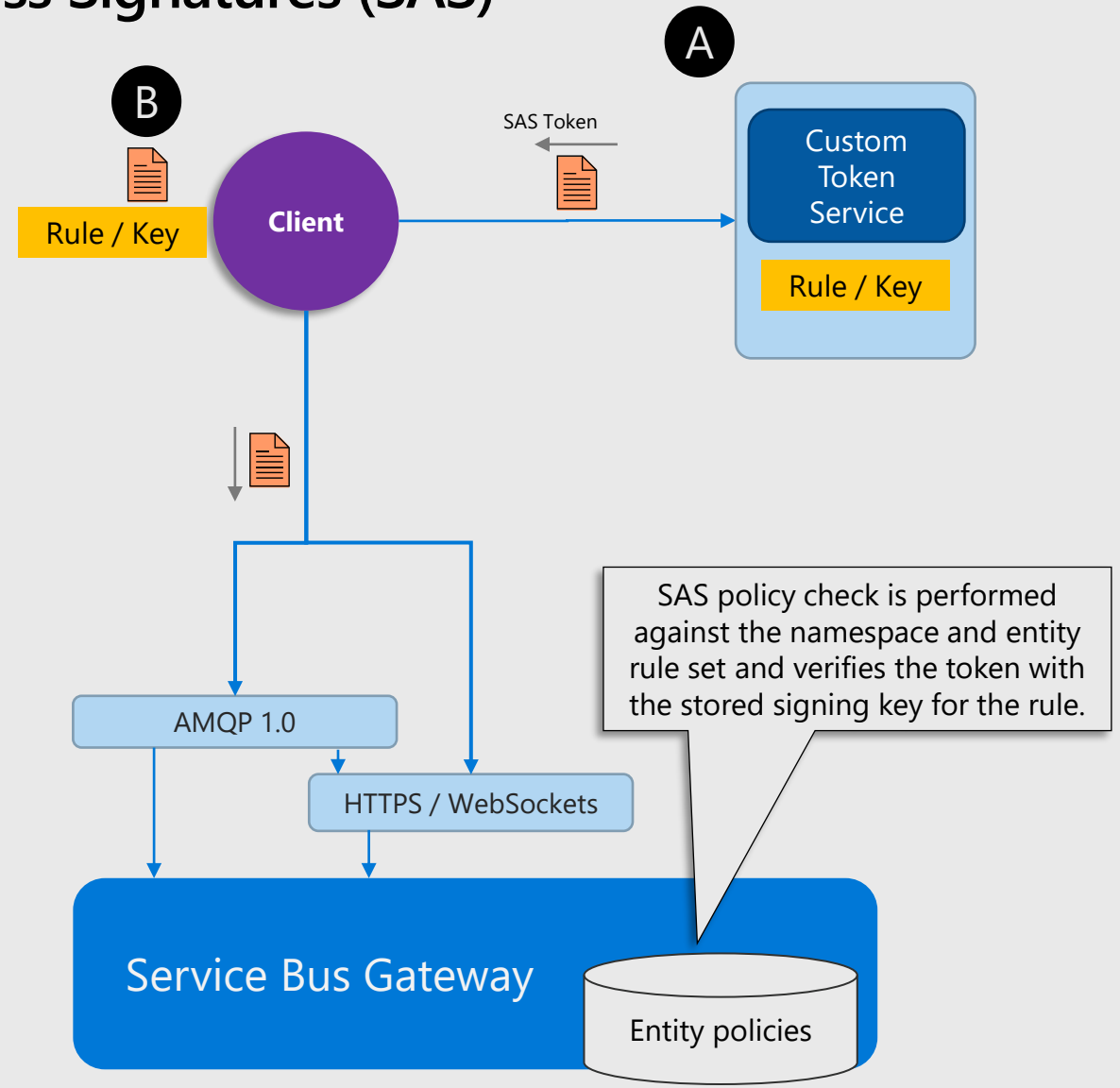
AAD RBAC: Integrated Azure identity model, also for service-level integration (managed identity)

- Very detailed, operation-level permission set that can be assigned to roles
- Several standard control- and data-plane roles (like Data Owner, Data Sender, Data Receiver)
- Roles can be assigned at the namespace and entity level

Authorization

Azure Active Directory RBAC and Shared Access Signatures (SAS)





Performance and Reliability

Throughput:

Service Bus Standard has a soft throttle at about 500 msg/sec per namespace

- 1000 credits per second, each send and receive operation counts one credit.

Service Bus Premium is only limited by compute and memory (MU) as well at I/O caps

- One log (a single queue) can handle about 10 MB/sec data I/O combined (5000 msg/sec @ 1kB)
- More features, more CPU and memory use, less throughput.

Reliability:

Monthly global uptime (are endpoints reachable?): **100%**

Monthly global reliability (are operations succeeding?): **> 99.995%**

Azure Service Bus

Platform-as-a-Service Queue and PubSub Message Broker

Fully managed: You use the features, Azure deals with everything else

JMS 2.0 and AMQP 1.0 standards compliant

Polyglot Azure SDK and cross-platform client support

Industry-leading reliability and availability

That was a lot, wasn't it?

Q&A