KubeCon | CloudNativeCon

North America 2019

NATS

Derek Collison, Colin Sullivan, Wally Quevedo, and Jaime Piña

Agenda

KubeCon | CloudNativeCon North America 2019

✓ Overview of the NATS project

✓ New Features and Roadmap

✓ Demonstrations

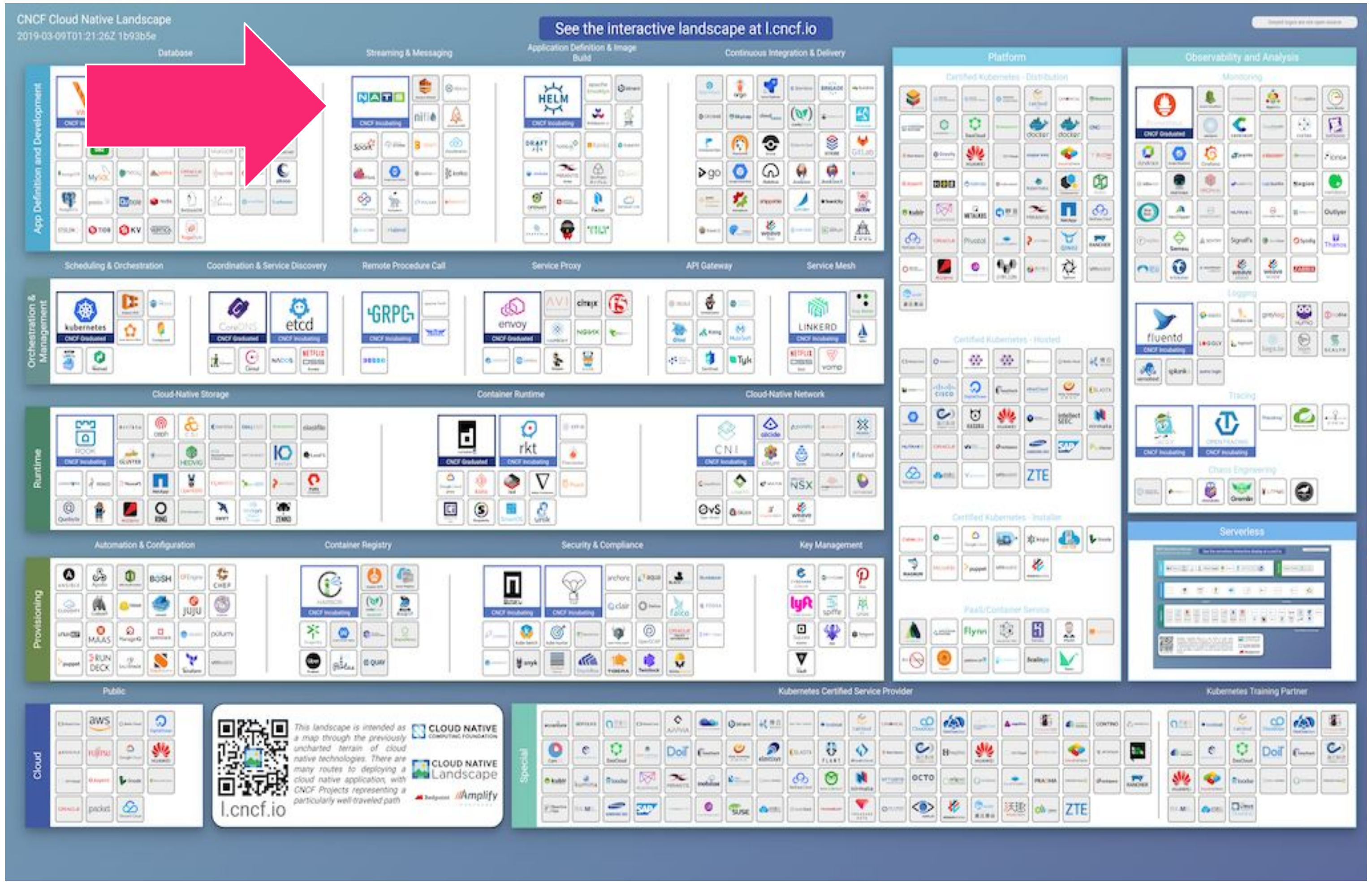
NATS Overview

What is NATS?

NATS is a ten year old, production proven, cloud-native messaging system made for developers and operators who want to spend more time doing their work and less time worrying about how to do messaging.

- ✓ DNA: Performance, simplicity, security, and availability
- ✓ Built from the ground up to be cloud native
- ✓ Multiple qualities of service
- ✓ Support for multiple communication patterns
- ✓ Over 30 client languages

CNCF Landscape


CNCF Cloud Native Landscape
See the interactive landscape at l.cncf.io
App Definition and Development
Orchestration & Management
Runtime
Provisioning
Platform
Observability and Analysis
Serverless
Cloud
Special
l.cncf.io

Joined CNCF as an incubation project in 2018

https://landscape.cncf.io

Contribution Statistics

- Over 1000 contributors, over 100 with more than 10 commits
- 30+ public repos
    - 15,000+ GitHub stars across repos
- ~51M NATS Server Docker Hub pulls
- ~50M NATS Streaming Server pulls
- 1500+ Slack members
- 20+ releases of the NATS server since June 2014, ~= 5/year

https://nats.devstats.cncf.io/d/9/developers-summary

History

Derek Collison
Founder and CEO at Synadia

Founder and former CEO at Apcera
CTO, Chief Architect at VMware
Architected CloudFoundry
Technical Director at Google
SVP and Chief Architect at TIBCO

Created by Derek Collison

Derek has been building messaging systems and solutions > 25 yrs

Maintained by a highly experienced messaging team

Engaged User Community

End Users

mastercard

netlify

Telia Company

htc

GE

vmware

Alibaba Group 阿里巴巴集团

Zephyr™ Project

ERICSSON

tinder

Bai du 百度

Pivotal

NATS Clients

KubeCon | CloudNativeCon
North America 2019

nats.go

Golang client for NATS, the cloud native messaging system.

go · golang · microservices · nats · cloud-native

● Go ★ 2,265 303 Apache-2.0 3 issues need help Updated a day ago

nats.rb

Ruby client for NATS, the cloud native messaging system.

ruby · client · messaging · cncf · pubsub · nats · eventmachine

● Ruby ★ 823 131 Apache-2.0 Updated a day ago

nats.java

Java client for NATS

java · client · middleware · messaging · nats · messaging-library

● Java ★ 194 68 Apache-2.0 Updated a day ago

nats.ex

Elixir client for NATS, the cloud native messaging system. https://nats.io

client · elixir · nats · nats-io

● Elixir ★ 33 11 MIT 1 issue needs help Updated 6 days ago

nats.js

Node.js client for NATS, the cloud native messaging system.

● JavaScript ★ 672 96 Apache-2.0 Updated 8 days ago

nats.net

The official C# Client for NATS

client · visual-studio · csharp · messaging · message-bus · pubsub

● C# ★ 232 63 Apache-2.0 3 issues need help Updated 2 days ago

nats.c

A C client for NATS

c · messaging · message-bus · message-queue · messaging-library

● C ★ 139 45 Apache-2.0 Updated 7 days ago

nats.py

An asyncio based Python 3 client for NATS

aio · nats · python3 · asyncio · cloud-native · aio-nats

● Python ★ 187 34 Apache-2.0 Updated 4 days ago

Use Cases

- Cloud Messaging
    - ✓ Services (microservices)
    - ✓ Event/Data Streaming (observability, analytics)
    - ✓ Command and Control
- IoT and Edge
    - ✓ Telemetry / Sensor Data / Command and Control
- Augmenting or Replacing Legacy Messaging

KubeCon | CloudNativeCon

North America 2019

Messaging Patterns

High Level Patterns

- Streams
  - ✓ A flow of data
  - ✓ Fan out
- Services
  - ✓ Do some work and return a result
  - ✓ Load balanced

Application Patterns

- ✓ Request/Reply
- ✓ Publish/Subscribe
- ✓ Load Balanced Queue Subscribers

Subjects

KubeCon | CloudNativeCon North America 2019

A subject is simply a string representing an interest in data.

- Simple subject: **foo**
- Hierarchically Tokenized: **foo.bar**
- Wildcard subscriptions
  - ✓ **foo.*** matches **foo.bar** and **foo.baz**.
  - ✓ **foo.*.bar** matches **foo.a.bar** and **foo.b.bar**.
  - ✓ **foo.>** matches any of the above
  - ✓ **>** matches <u>everything</u> in NATS
- Unique Subjects for 1:1 addressability

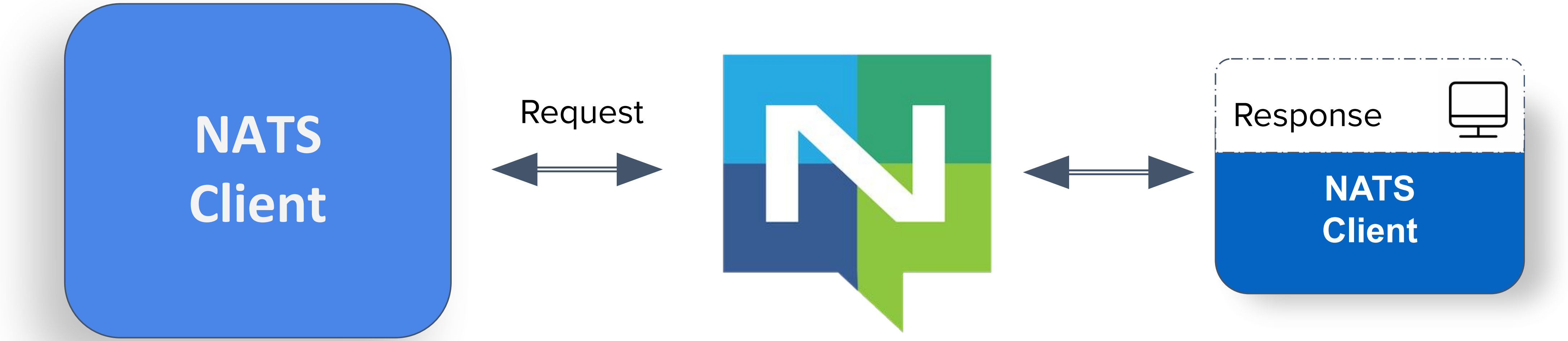
Wildcard Subscribers

- Given sensors publish messages to:
  - ✓ sensors.data.us.ca.sandiego
  - ✓ sensors.errors.us.ca.sandiego
  - ✓ sensors.data.uk.eng.london
  - ✓ sensors.errors.uk.eng.london

- Subscribe to:
  - ✓ sensors.data.us.> → all US data
  - ✓ sensors.data.uk.eng.london → data from London
  - ✓ sensors.errors.> → errors worldwide
  - ✓ sensors.*.uk.> → all errors and data in the UK

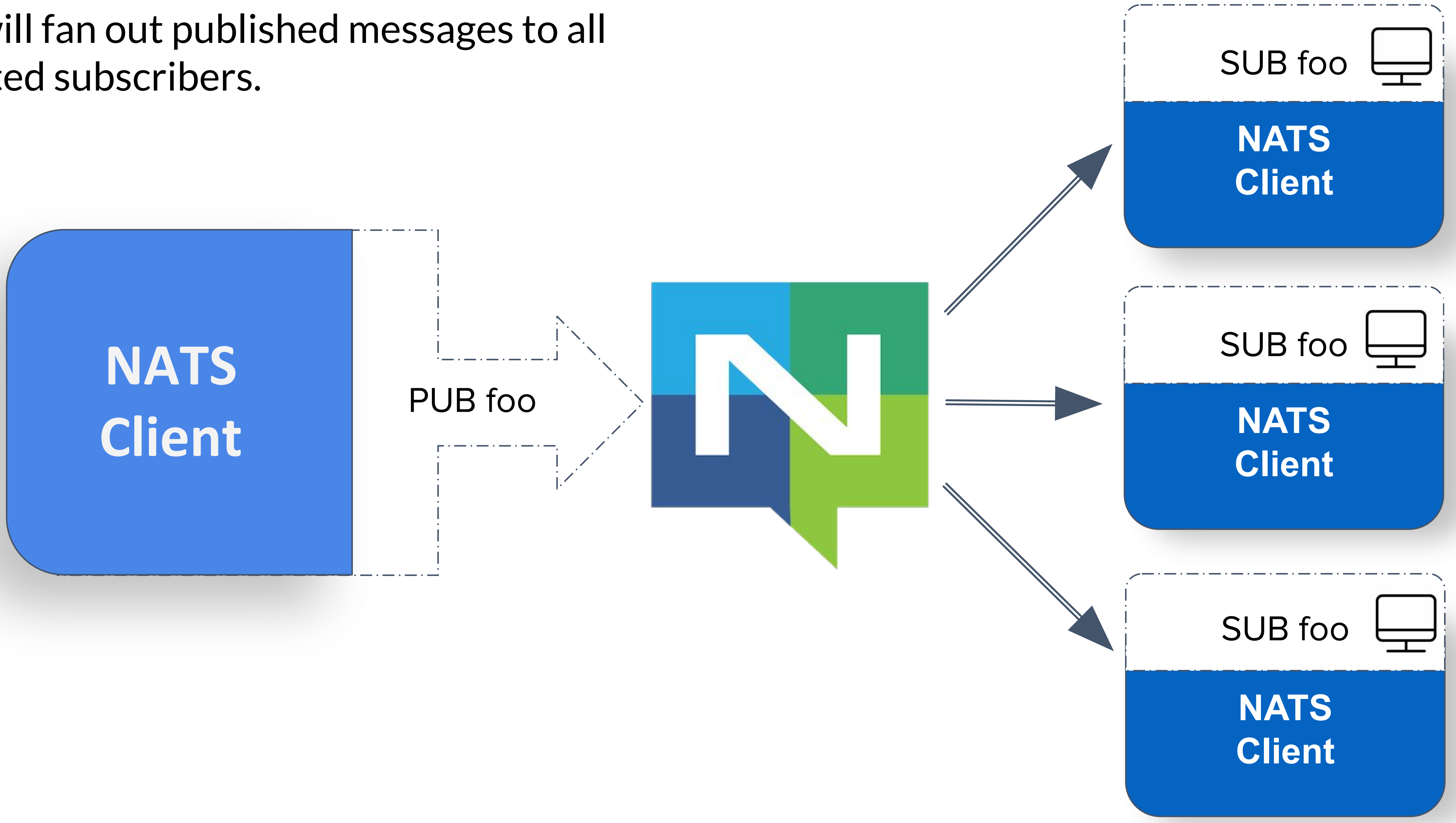
Request/Response (1:1)

Using unique reply subjects, clients can make requests to services that respond only to the request, creating a 1 to 1 relationship.



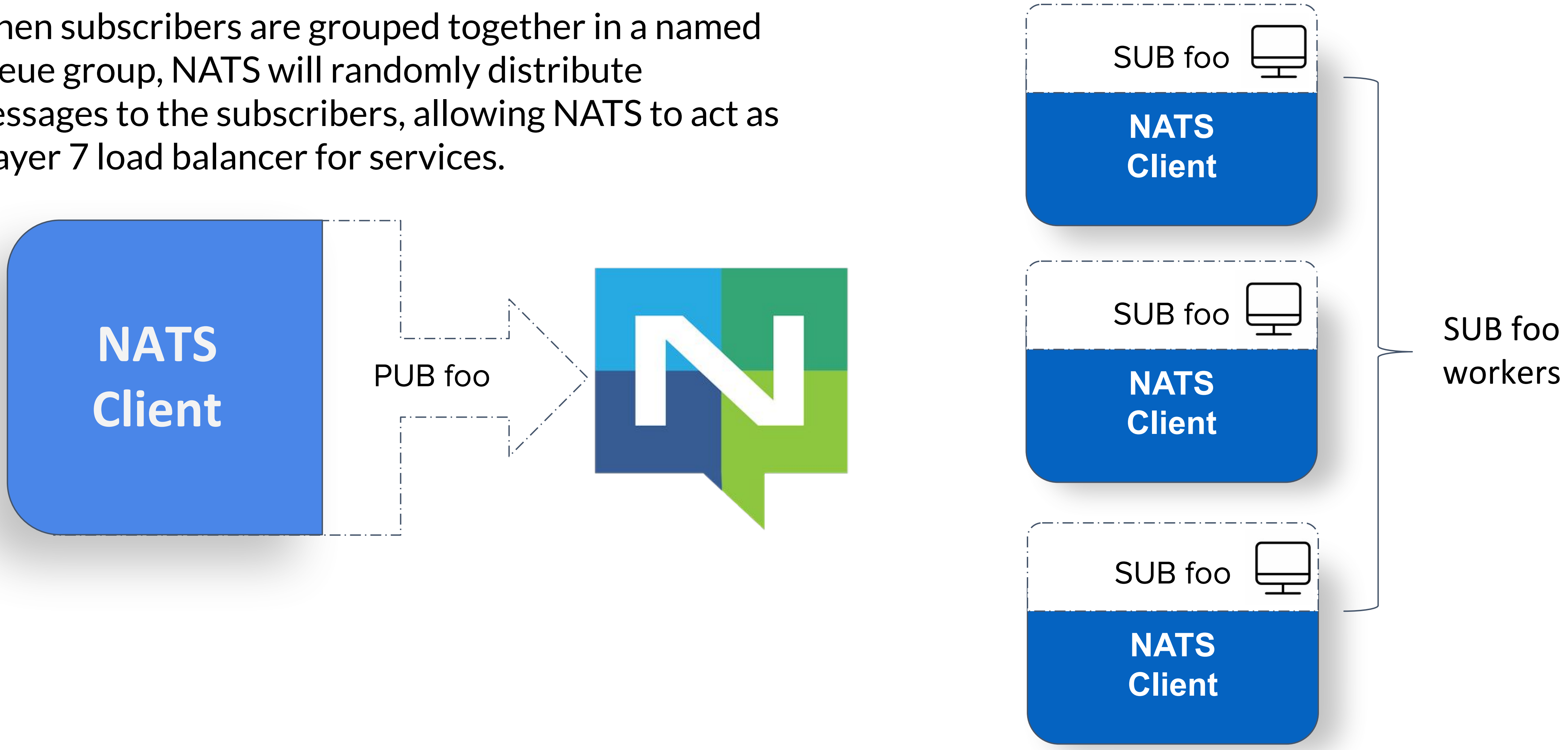
Publish/Subscribe (1:N)

NATS will fan out published messages to all interested subscribers.

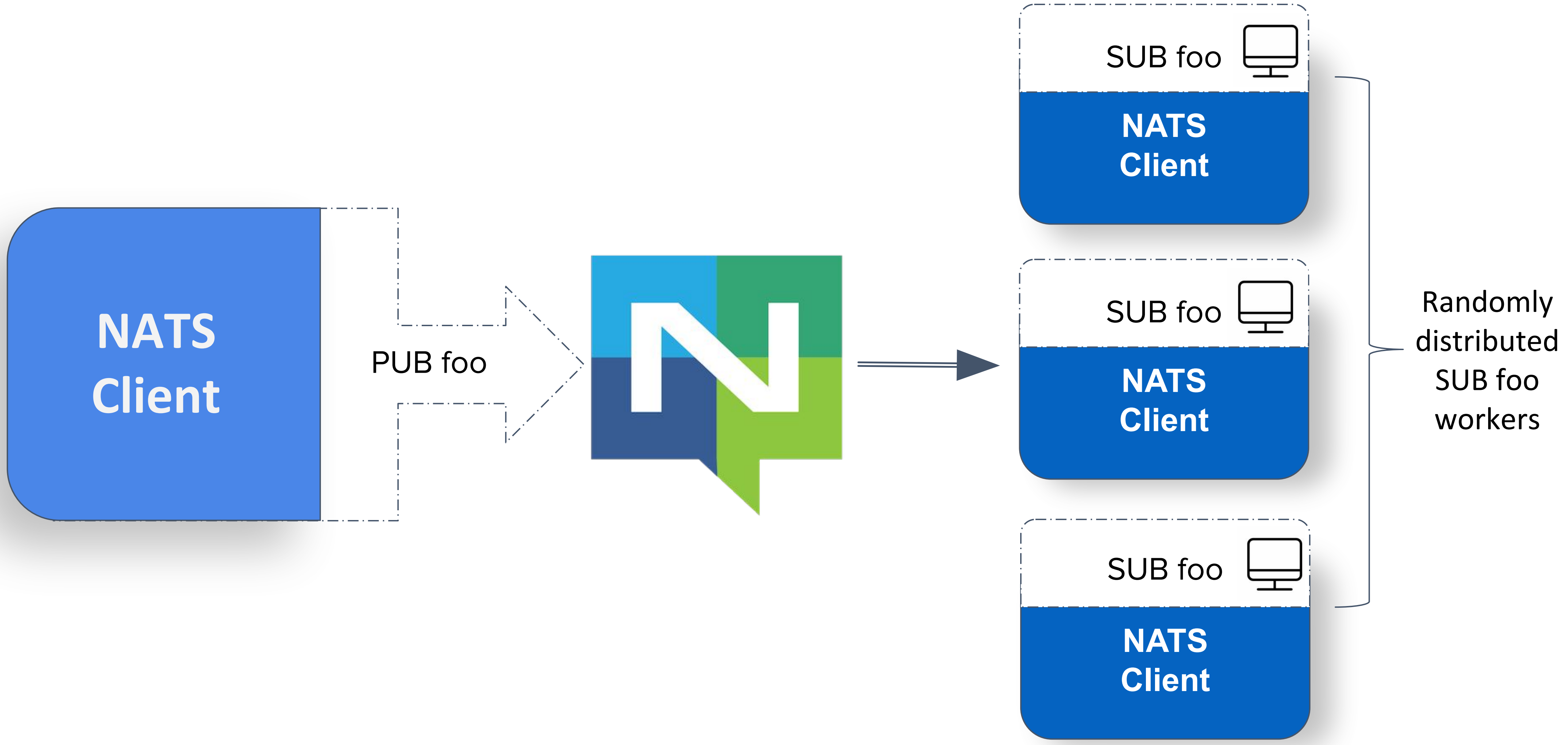


Load Balanced Queues

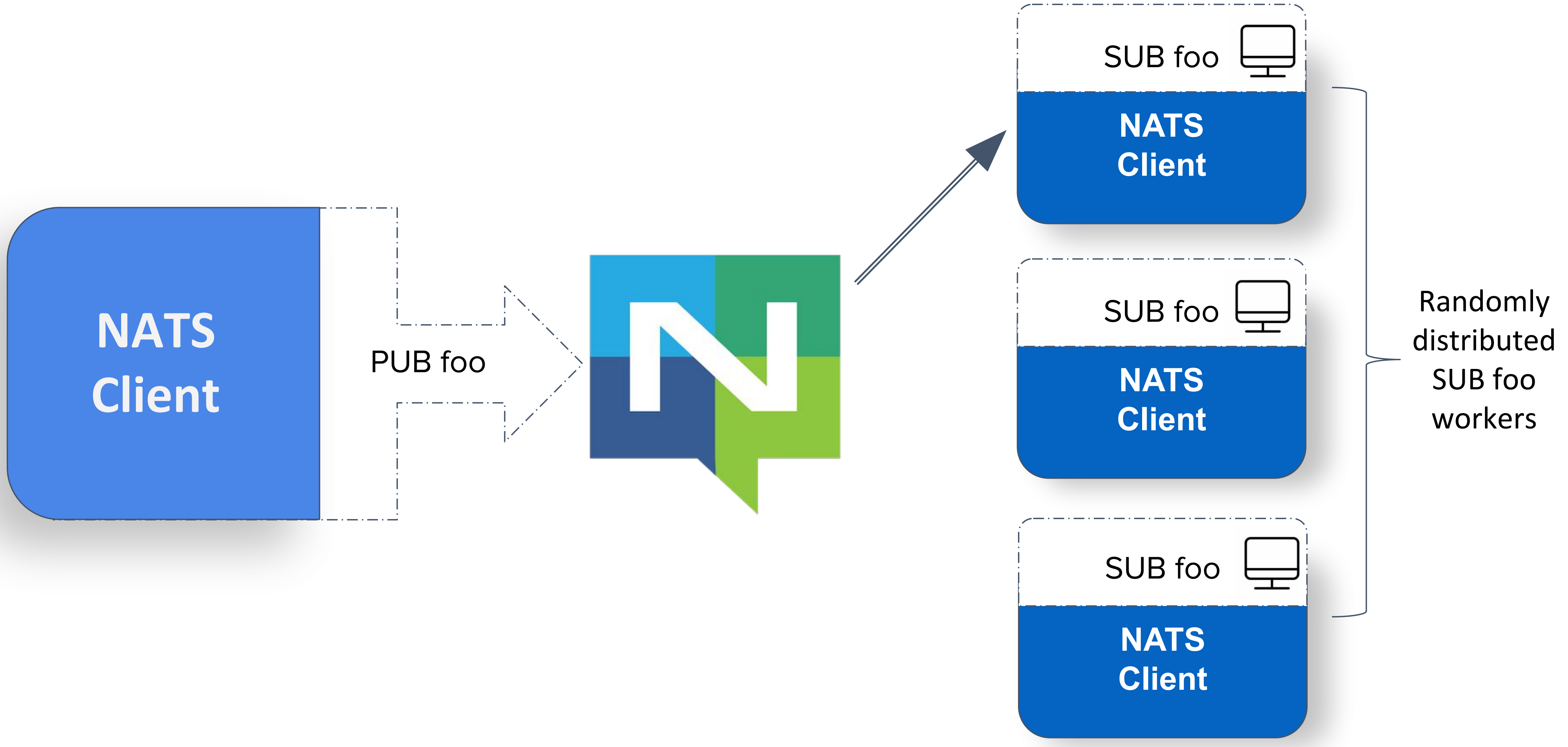
When subscribers are grouped together in a named queue group, NATS will randomly distribute messages to the subscribers, allowing NATS to act as a layer 7 load balancer for services.



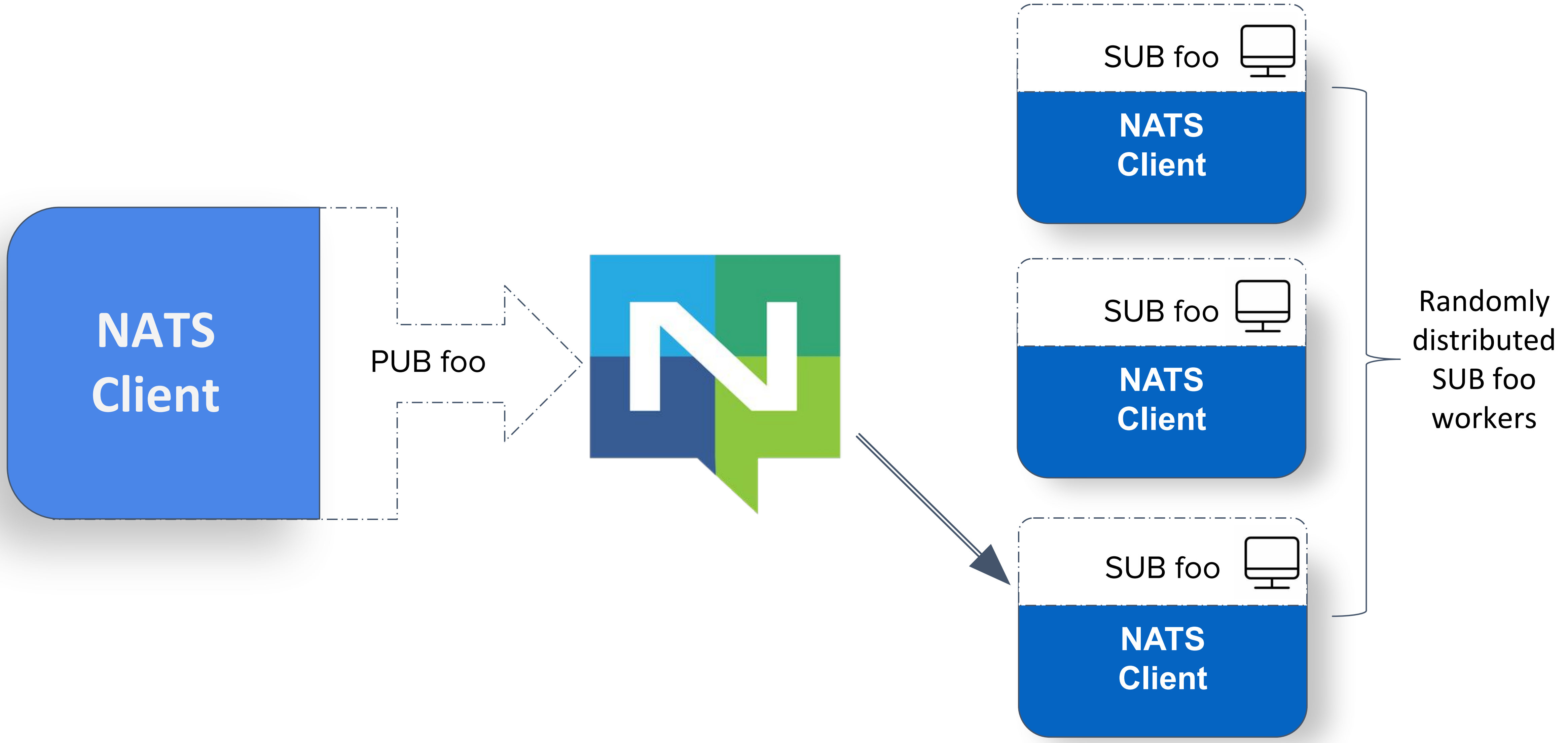
Load Balanced Queues



Load Balanced Queues

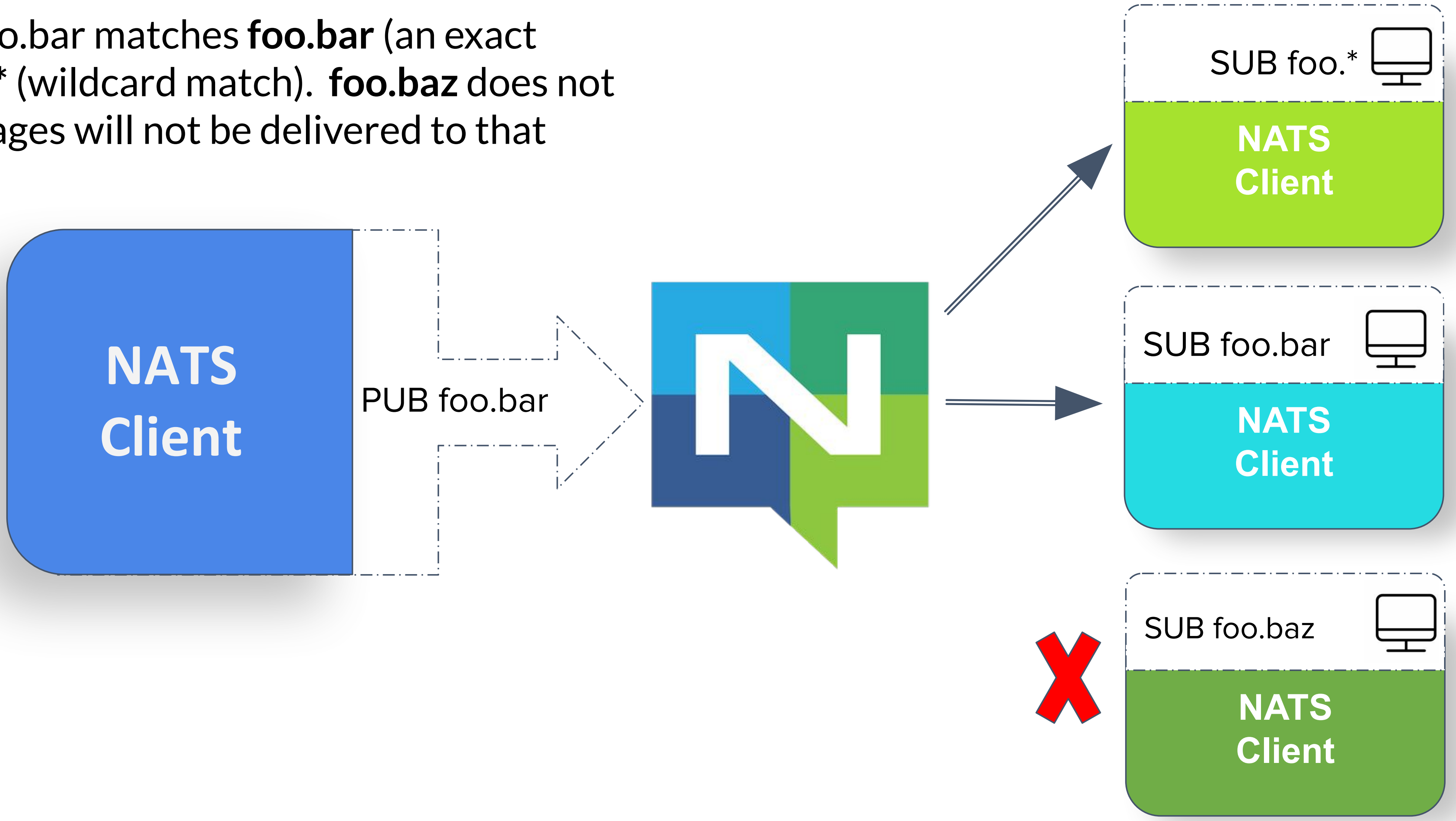


Load Balanced Queues



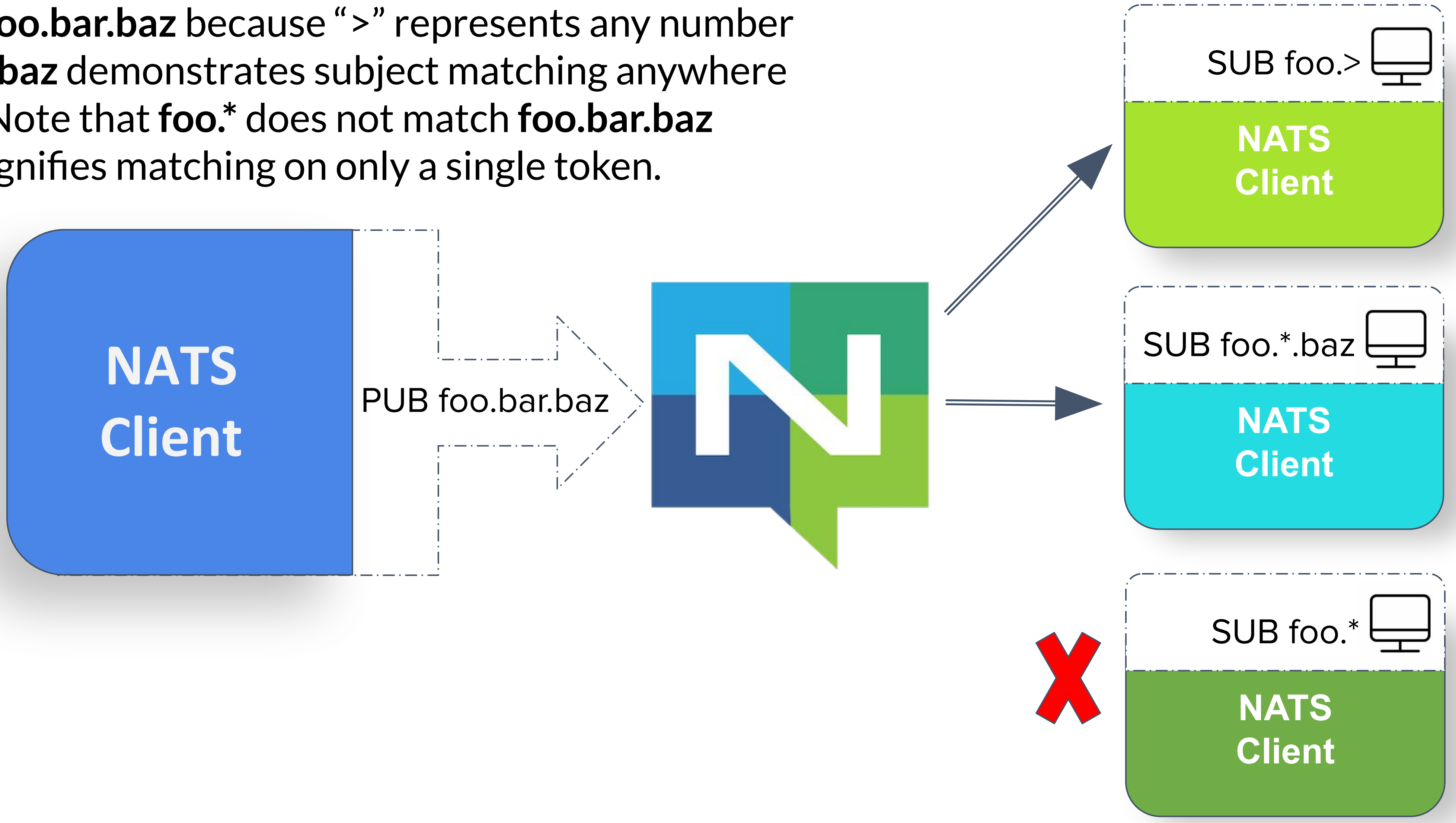
Wildcard Subscribers

Publishing to foo.bar matches **foo.bar** (an exact match) and **foo.*** (wildcard match). **foo.baz** does not match, so messages will not be delivered to that subscriber.



Wildcard Subscribers

foo.> matches **foo.bar.baz** because ">" represents any number of tokens. **foo.*.baz** demonstrates subject matching anywhere in the subject. Note that **foo.*** does not match **foo.bar.baz** because the * signifies matching on only a single token.



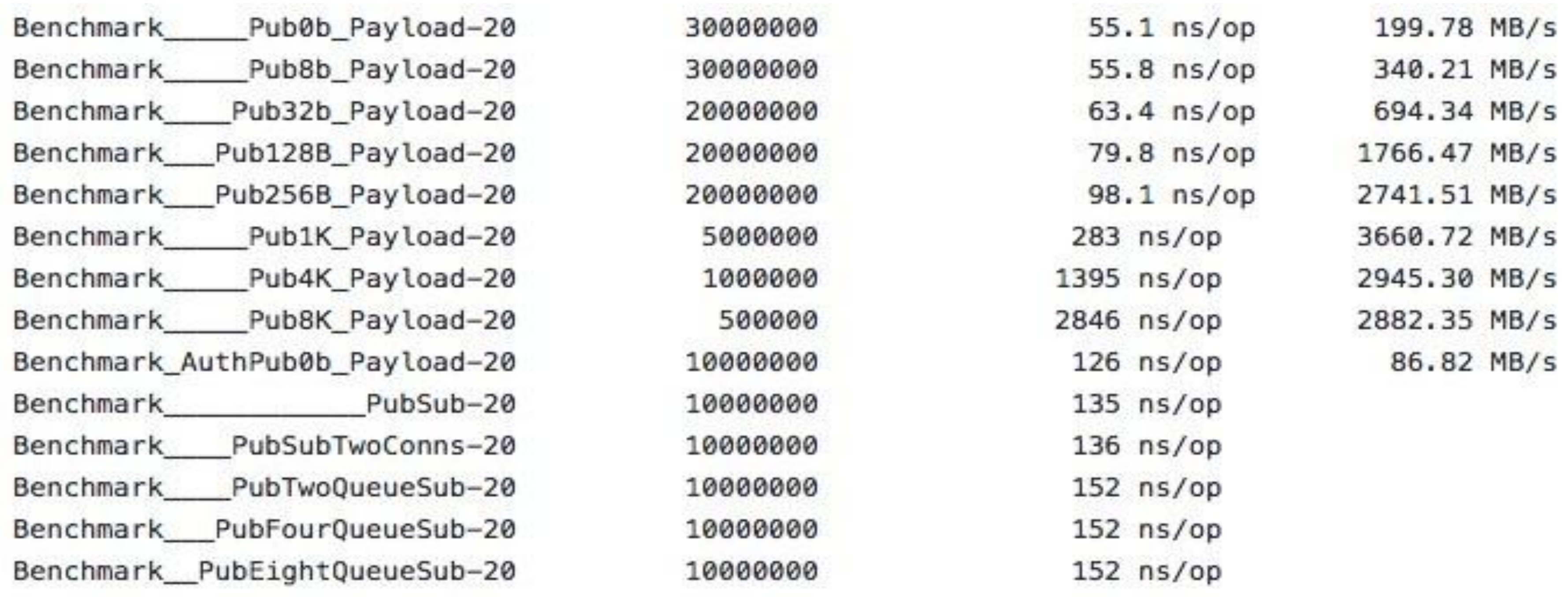
KubeCon | CloudNativeCon

North America 2019

Performance, Scalability, and Resilience

Performance

18 million messages per second with one server, one data stream.
Up to **80 million** messages per second per server with multiple data streams.



```
Benchmark_____Pub0b_Payload-20        30000000         55.1 ns/op     199.78 MB/s
Benchmark_____Pub8b_Payload-20        30000000         55.8 ns/op     340.21 MB/s
Benchmark____Pub32b_Payload-20        20000000         63.4 ns/op     694.34 MB/s
Benchmark___Pub128B_Payload-20        20000000         79.8 ns/op    1766.47 MB/s
Benchmark___Pub256B_Payload-20        20000000         98.1 ns/op    2741.51 MB/s
Benchmark_____Pub1K_Payload-20         5000000        283 ns/op      3660.72 MB/s
Benchmark_____Pub4K_Payload-20         1000000       1395 ns/op      2945.30 MB/s
Benchmark_____Pub8K_Payload-20          500000       2846 ns/op      2882.35 MB/s
Benchmark_AuthPub0b_Payload-20        10000000        126 ns/op        86.82 MB/s
Benchmark____________PubSub-20        10000000        135 ns/op
Benchmark____PubSubTwoConns-20        10000000        136 ns/op
Benchmark____PubTwoQueueSub-20        10000000        152 ns/op
Benchmark___PubFourQueueSub-20        10000000        152 ns/op
Benchmark__PubEightQueueSub-20        10000000        152 ns/op
```

Performance Decisions

Performance is a part of every decision we make…

- ✓ Design for scale
- ✓ Careful analysis of the fastpath

Just as important is what **NOT** to implement…

- ✗ Exactly Once QoS
- ✗ Transactions
- ✗ Message Schemas
- ✗ Message Headers

Availability

The health and availability of the system as a whole is prioritized over servicing any individual client or server…

- ✓ NATS server "selfish optimization"
  - → Protects against *Slow Consumers*
- ✓ Full Mesh clustering of NATS servers
- ✓ Server and client connections self heal

… this creates a NATS dial-tone, always on, always available.

Simplicity

- Single binary
- 7.8 MB docker image with no external dependencies
- “Text-based” protocol with just a handful of verbs

| PUB | SUB | UNSUB | CONNECT | INFO | MSG | -ERR | +OK | PING | PONG |

- Low Configuration
  - ✓ Clients only need a url and credentials
  - ✓ Servers auto-discover
  - ✓ You can share configuration files amongst servers
- Simple and Straightforward API

Auto Discovery

- Auto-Discovery
  - ✓ Automatically Exchange Server Topology
  - ✓ Server ⇆ Server
  - ✓ Server ⟶ Client
- No configuration updates
  - ✓ Failover to auto-discovered servers
- Great for rolling upgrades

Delivery Modes

Delivery Modes

NATS supports two delivery modes:

- At most once *(Core)*
    - ✓ No guarantee of delivery - messages can be lost - applications must detect and handle lost messages
- At least once *(NATS Streaming and JetStream)*
    - ✓ A message will always be delivered, but in certain cases may be delivered more than once

✗ Exactly once is arguably unnecessary, always complex, and inevitably slow

Delivery Modes

Through NATS streaming and **JetStream**, NATS supports:

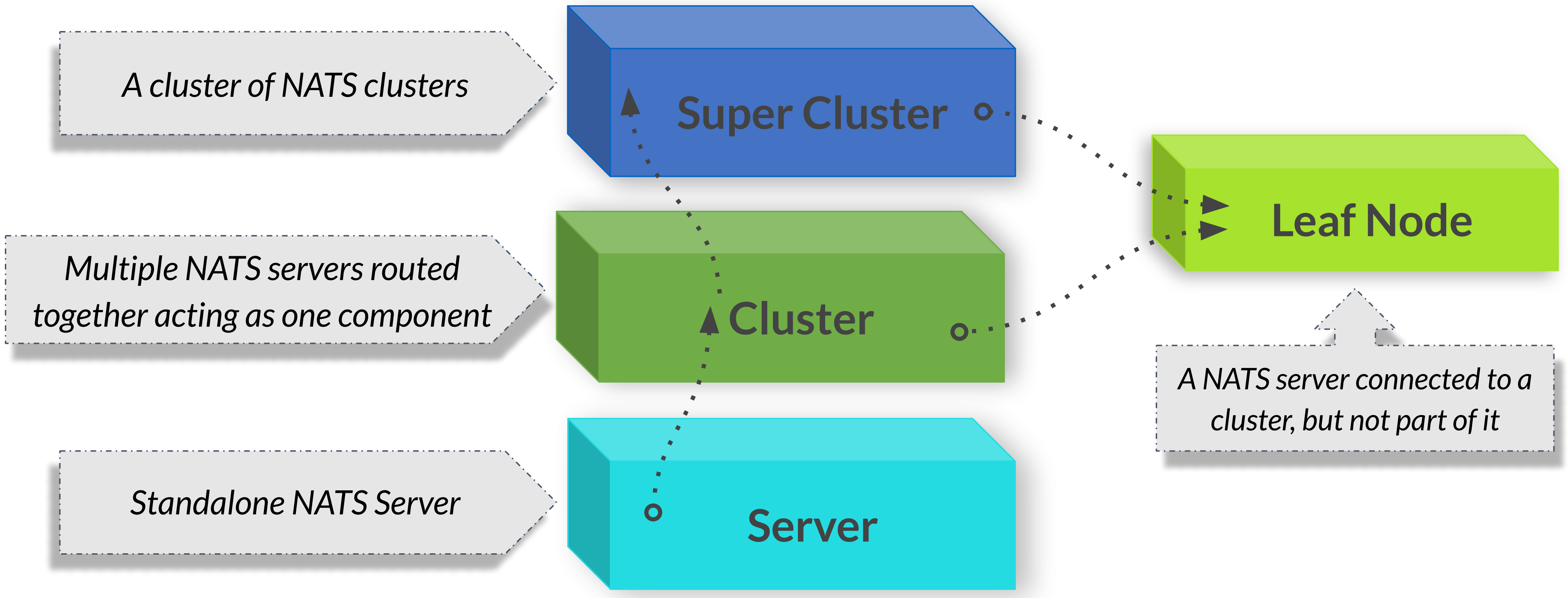
- **At-least-once** delivery
- Replay by time or sequence number
- Last/initial value caching
- Durable subscribers
- Rate matching per subscriber
- Memory, File, or Database storage
- High Availability through fault tolerant or clustered configurations
- Scale through partitioning

KubeCon | CloudNativeCon

North America 2019

Deployment Topologies

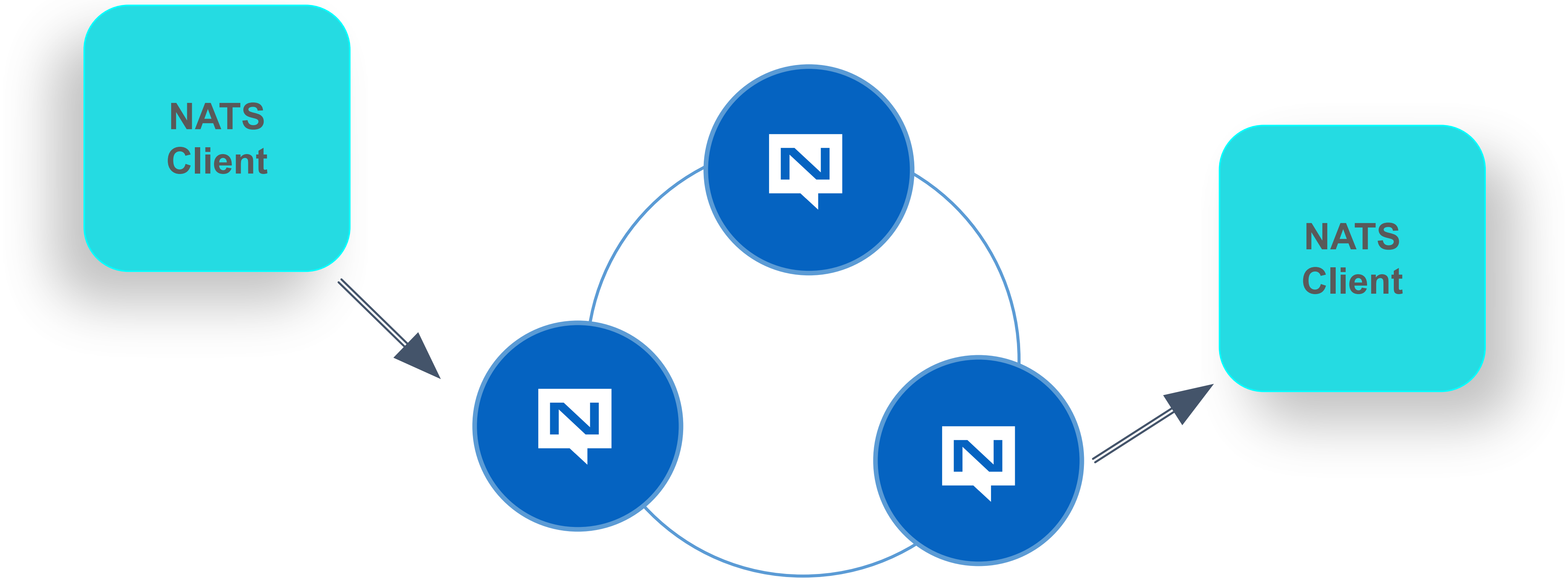
Topology Building Blocks


A cluster of NATS clusters
Super Cluster
Multiple NATS servers routed together acting as one component
Cluster
Standalone NATS Server
Server
Leaf Node
A NATS server connected to a cluster, but not part of it

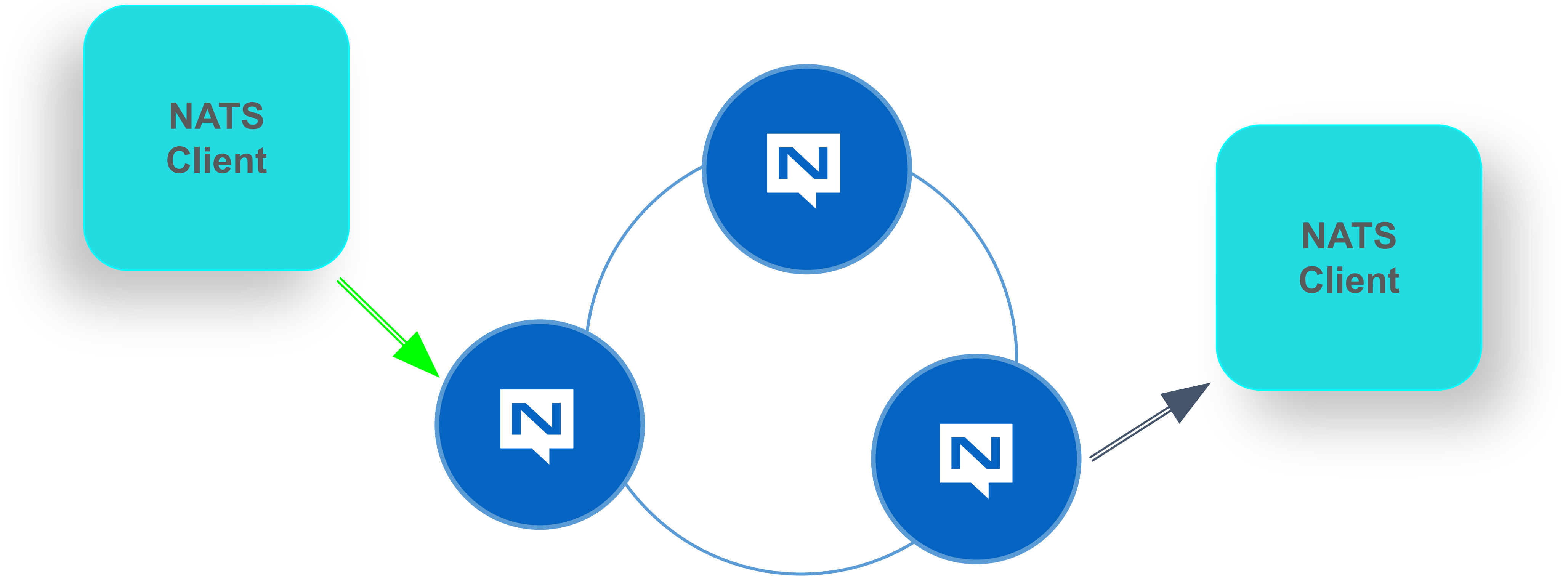
Clients require **no awareness** of server topology beyond a connection URL.

Clusters

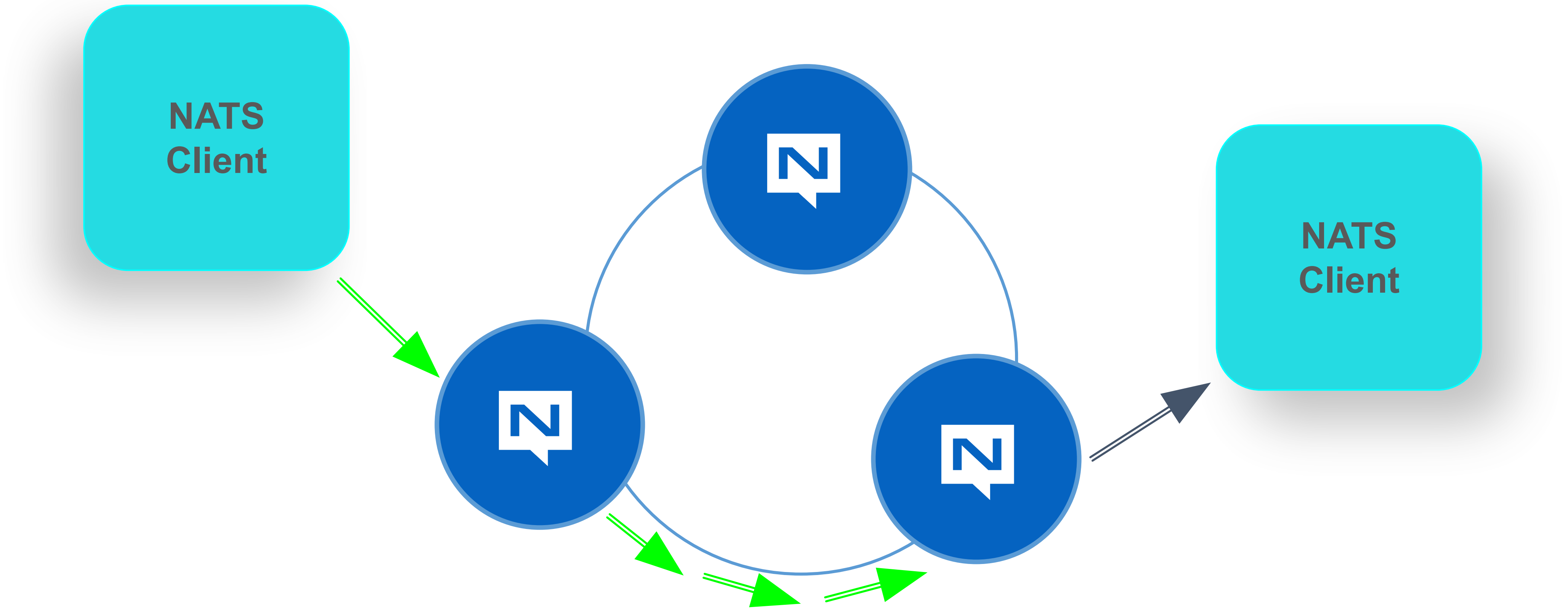
NATS Server clusters are full mesh one hop, and messages only traverse clusters where there is interest.



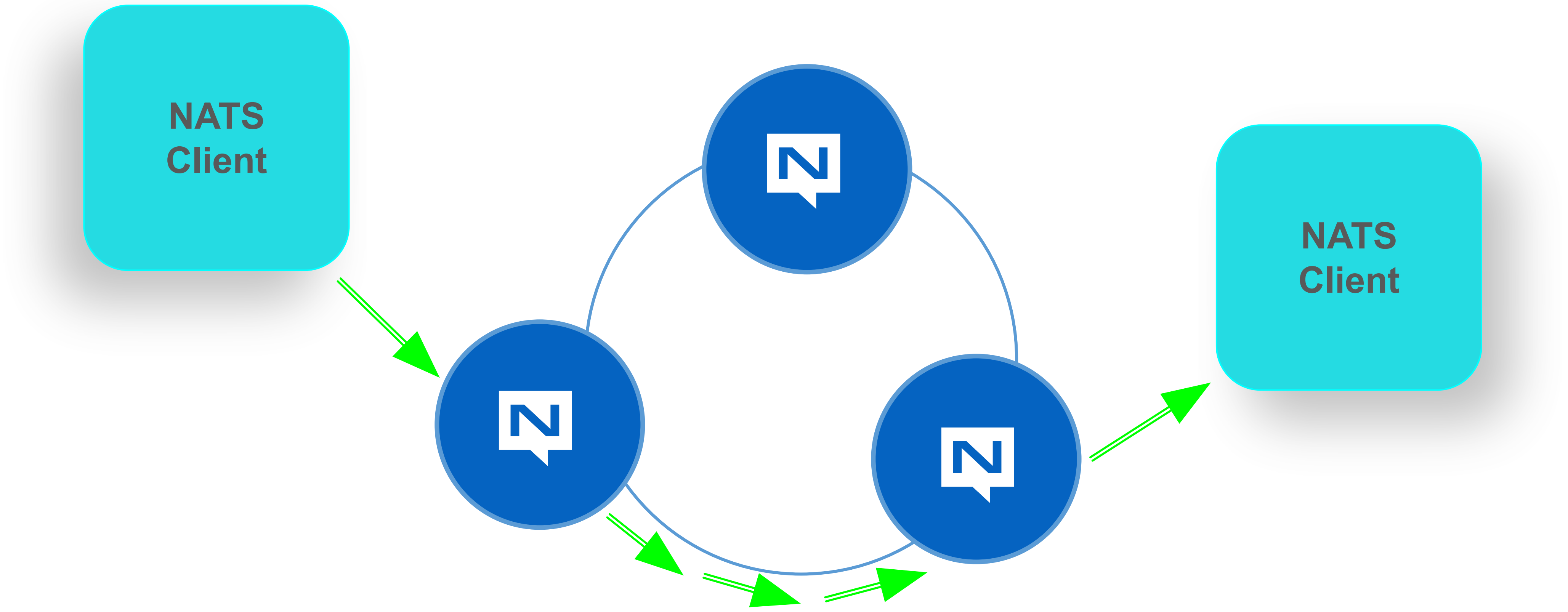
Clusters



Clusters

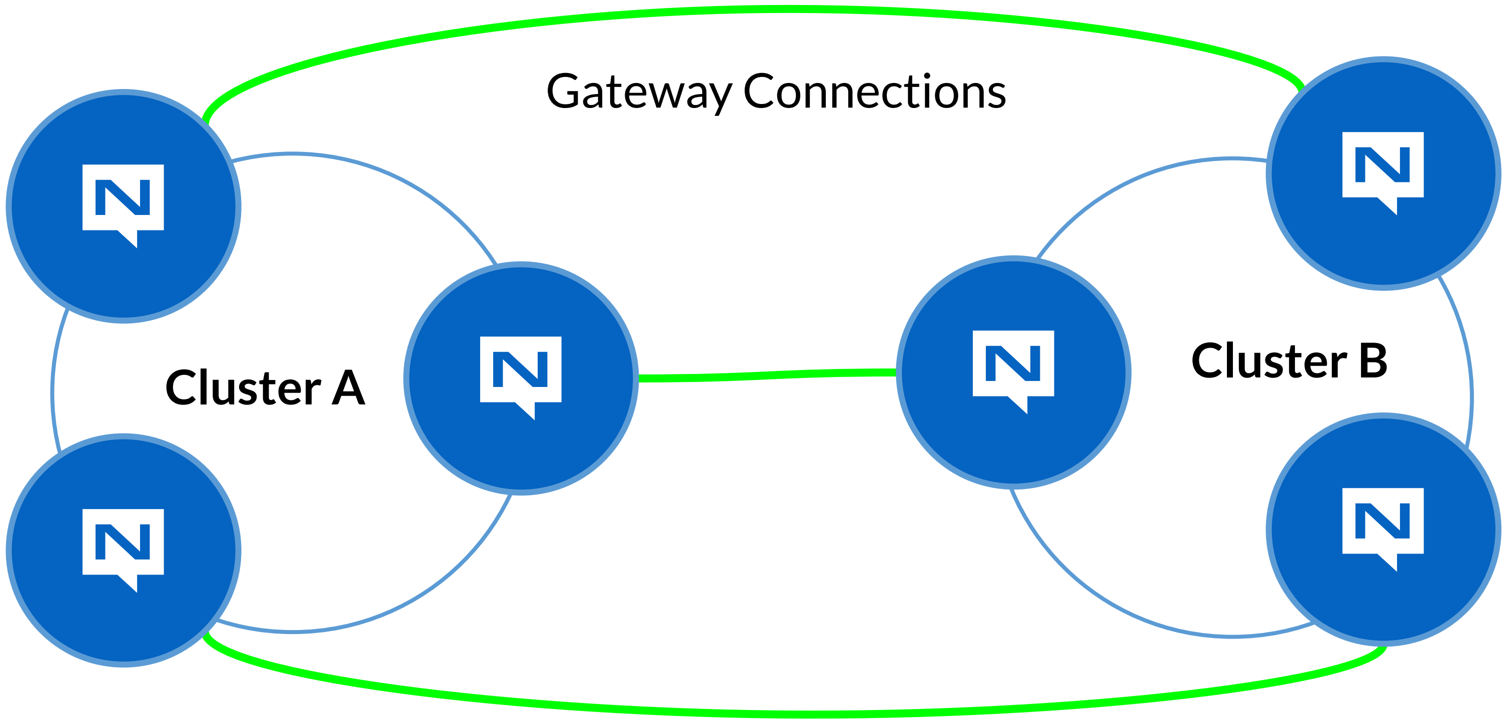


Clusters



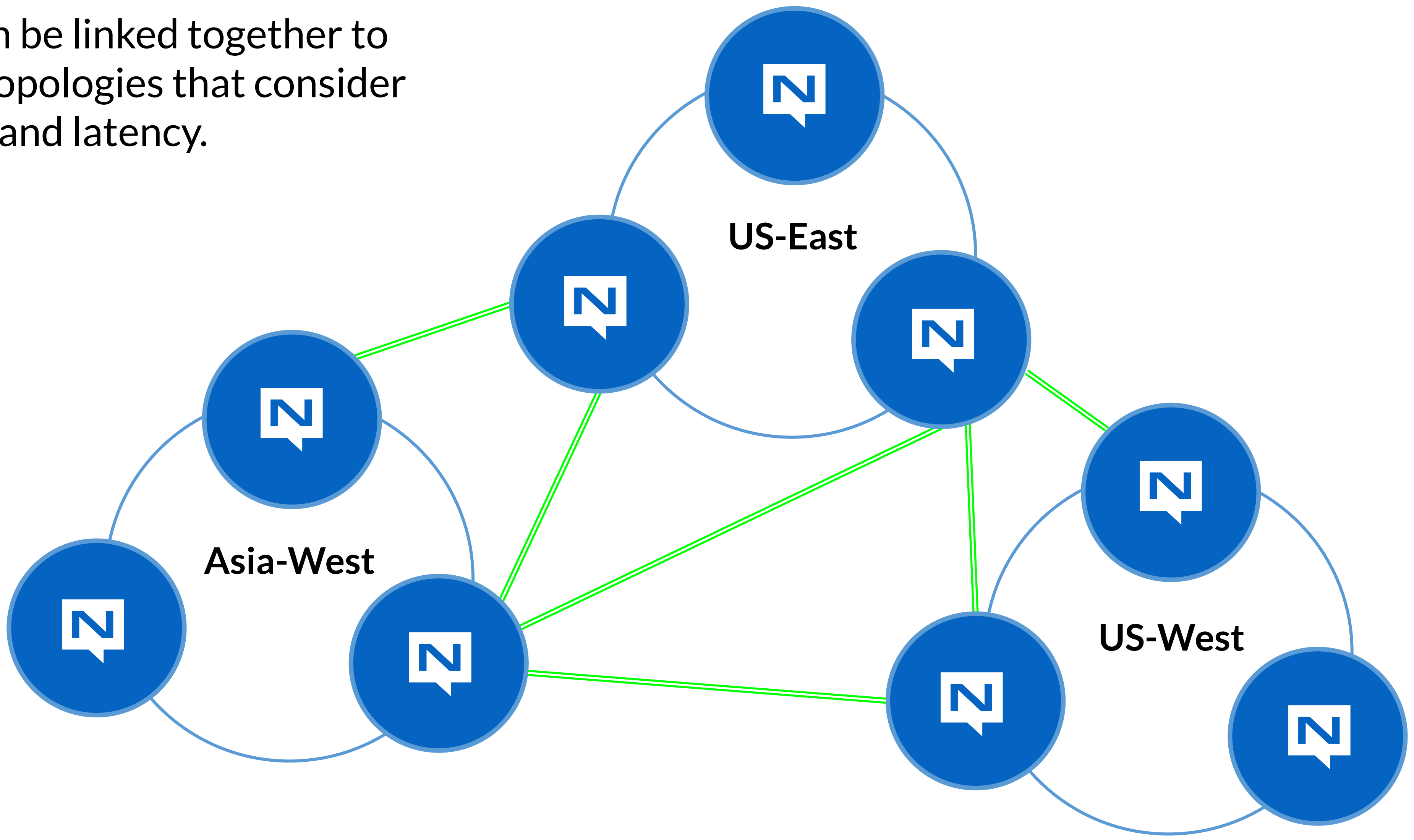
Superclusters

Superclusters are clusters of clusters connected together with gateway connections. They use a spline based technology to ensure resiliency and optimize traffic across clusters.



Superclusters

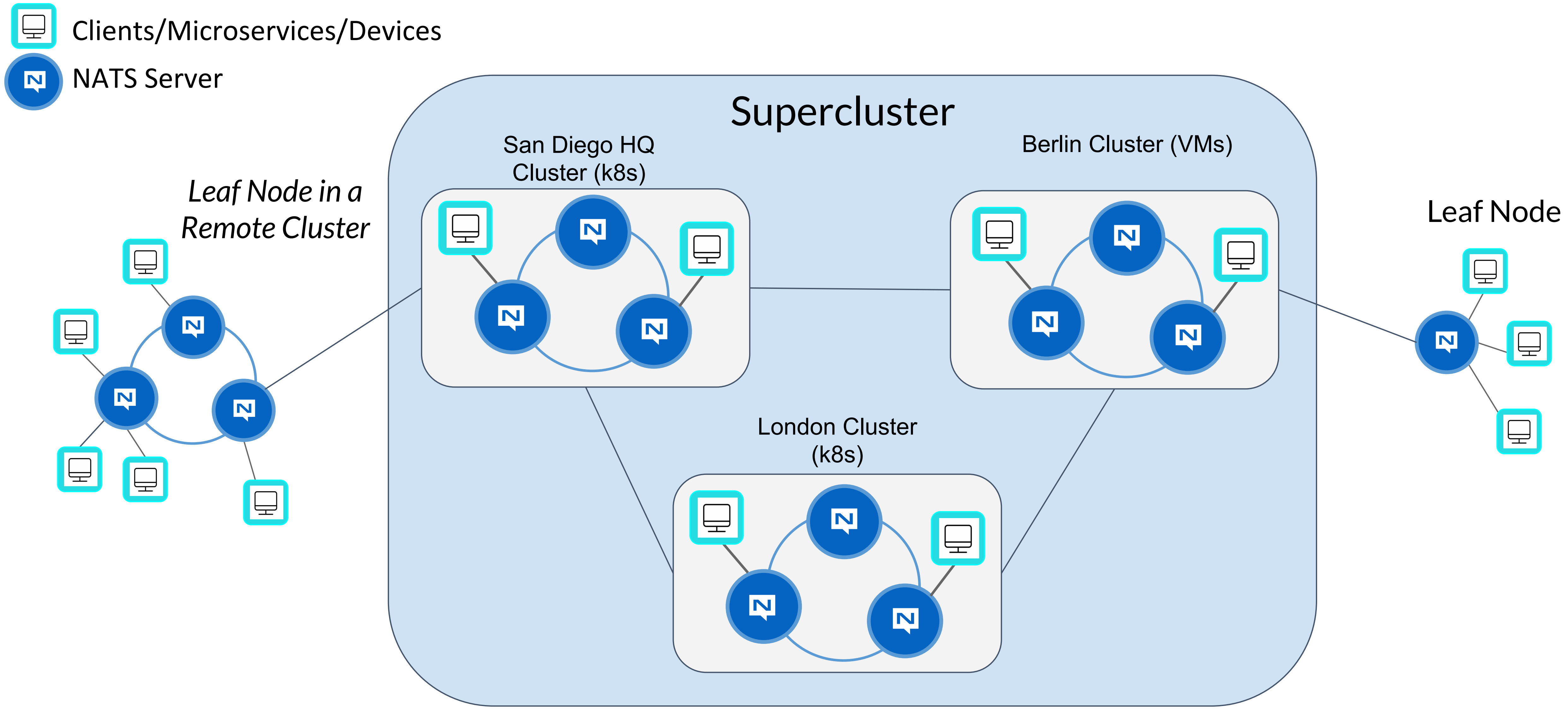
Multiple clusters can be linked together to form vast network topologies that consider WAN network links and latency.



Leaf Nodes

- ✓ A leaf nodes is a single NATS server connected to a cluster or remote server.
- ✓ Leaf nodes extend clusters via a hub and spoke topology
- ✓ Leaf nodes allow you to bridge separate security domains.
- ✓ Ideal for edge computing, IoT hubs, or data centers that need to be connected to a global, regional, or national NATS deployment.
- ✓ Transparently bridge on-premise and cloud deployments.

Hypothetical Global Deployment

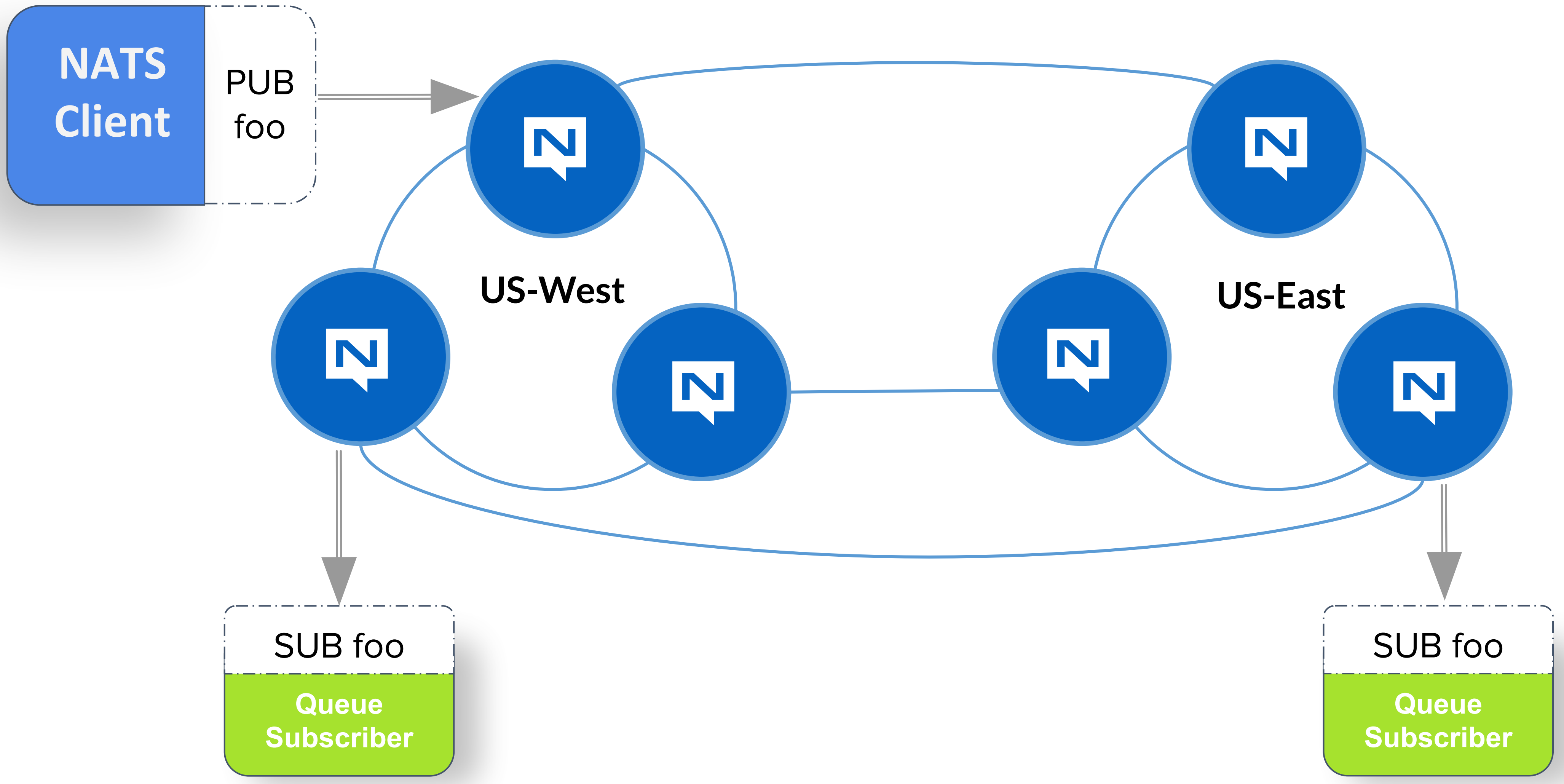

Clients/Microservices/Devices
NATS Server
Supercluster
San Diego HQ Cluster (k8s)
Berlin Cluster (VMs)
London Cluster (k8s)
Leaf Node in a Remote Cluster
Leaf Node

KubeCon | CloudNativeCon

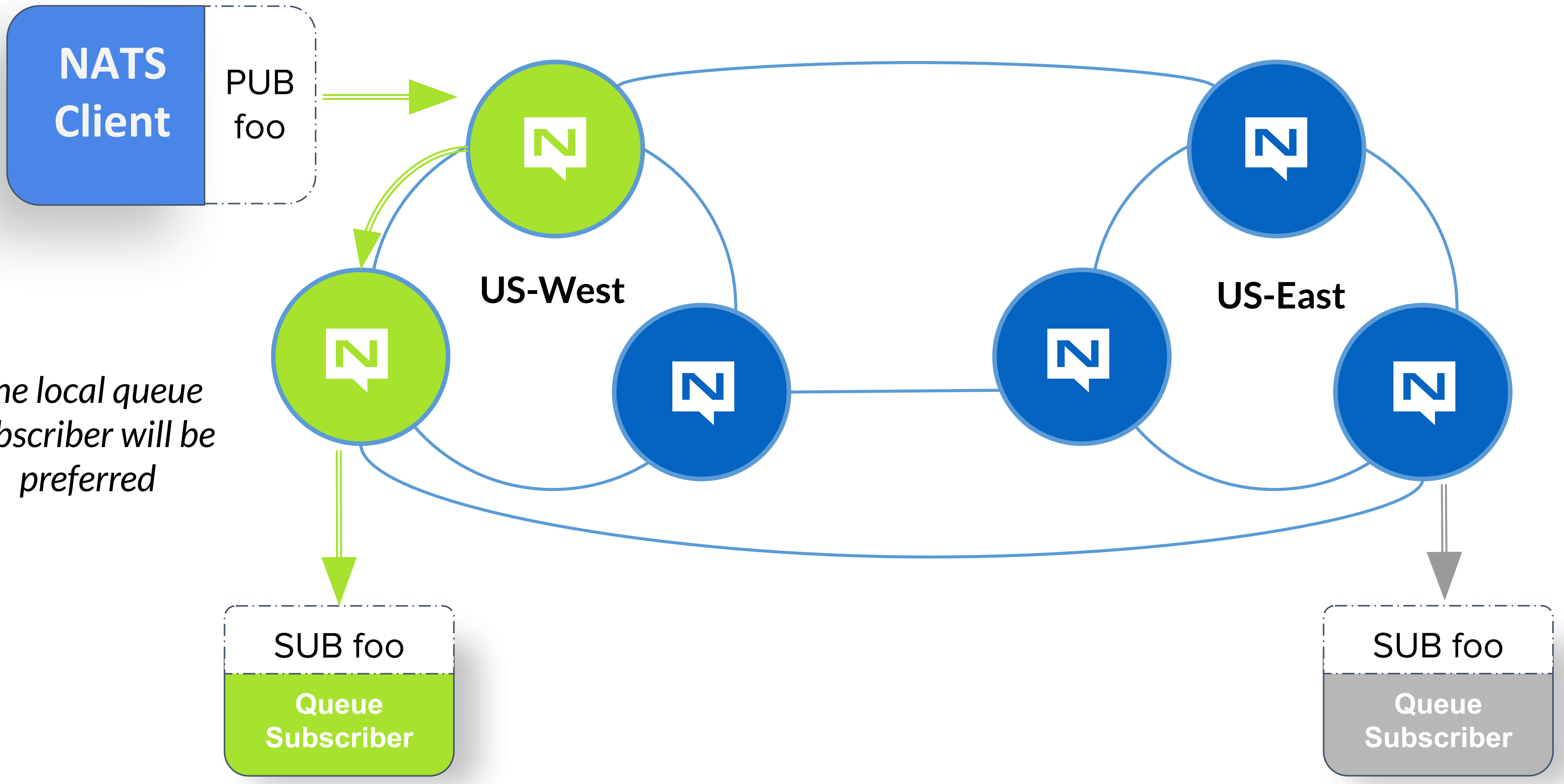
North America 2019

Geo Aware Queue Subscribers

Geo Aware Queue Subscribers

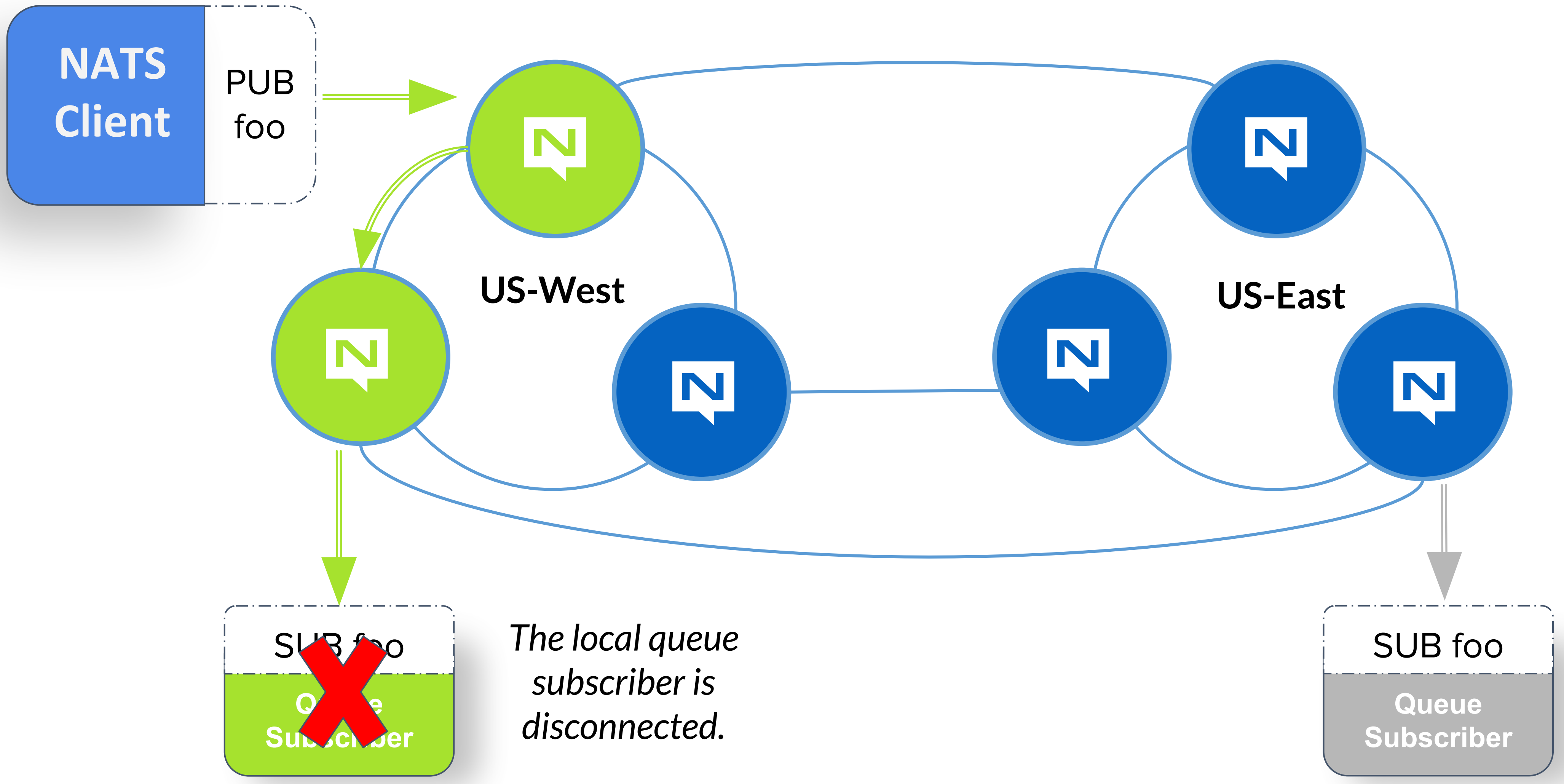


Geo Aware Queue Subscribers



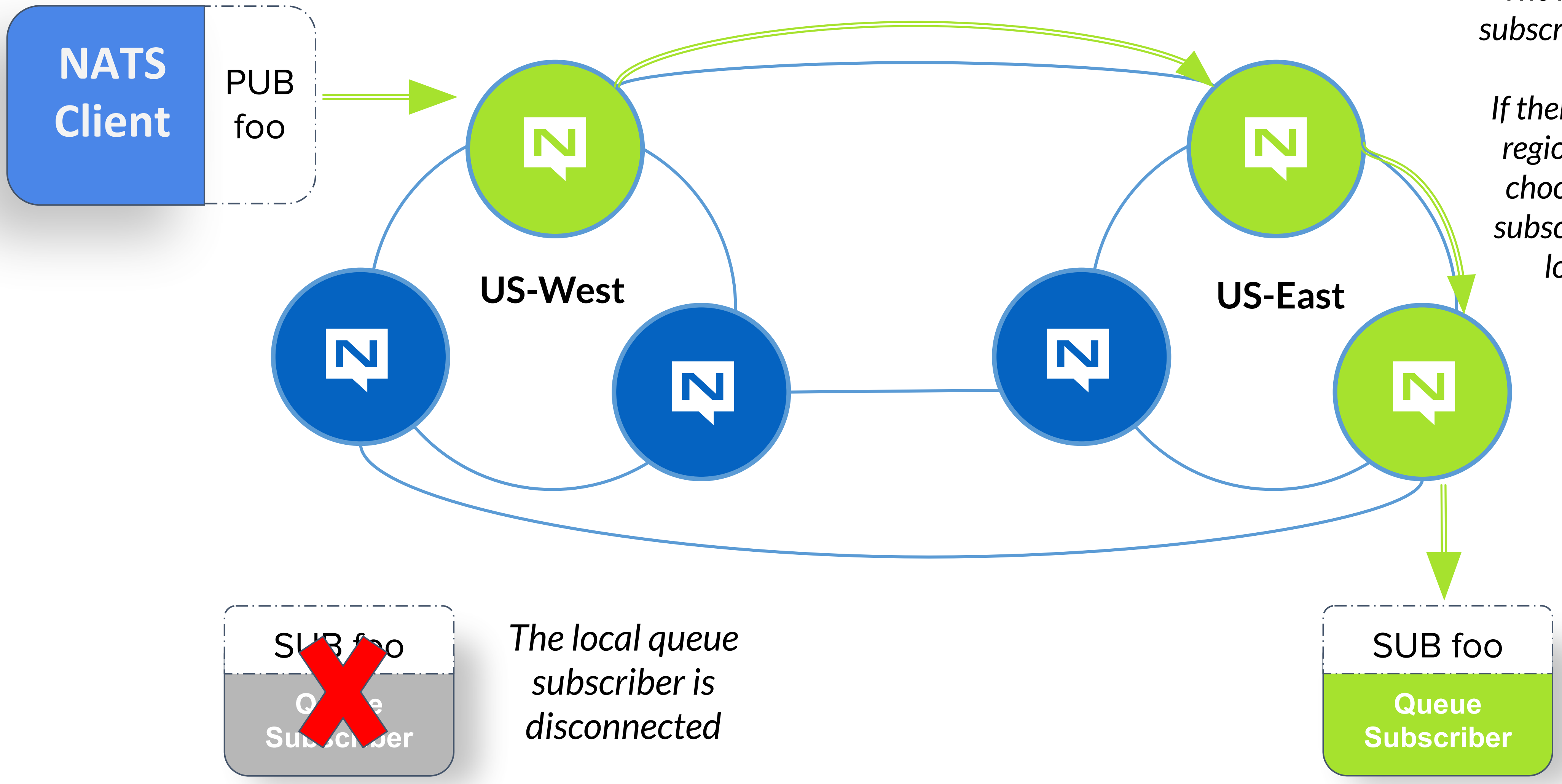
The local queue subscriber will be preferred

Geo Aware Queue Subscribers



The local queue subscriber is disconnected.

Geo Aware Queue Subscribers

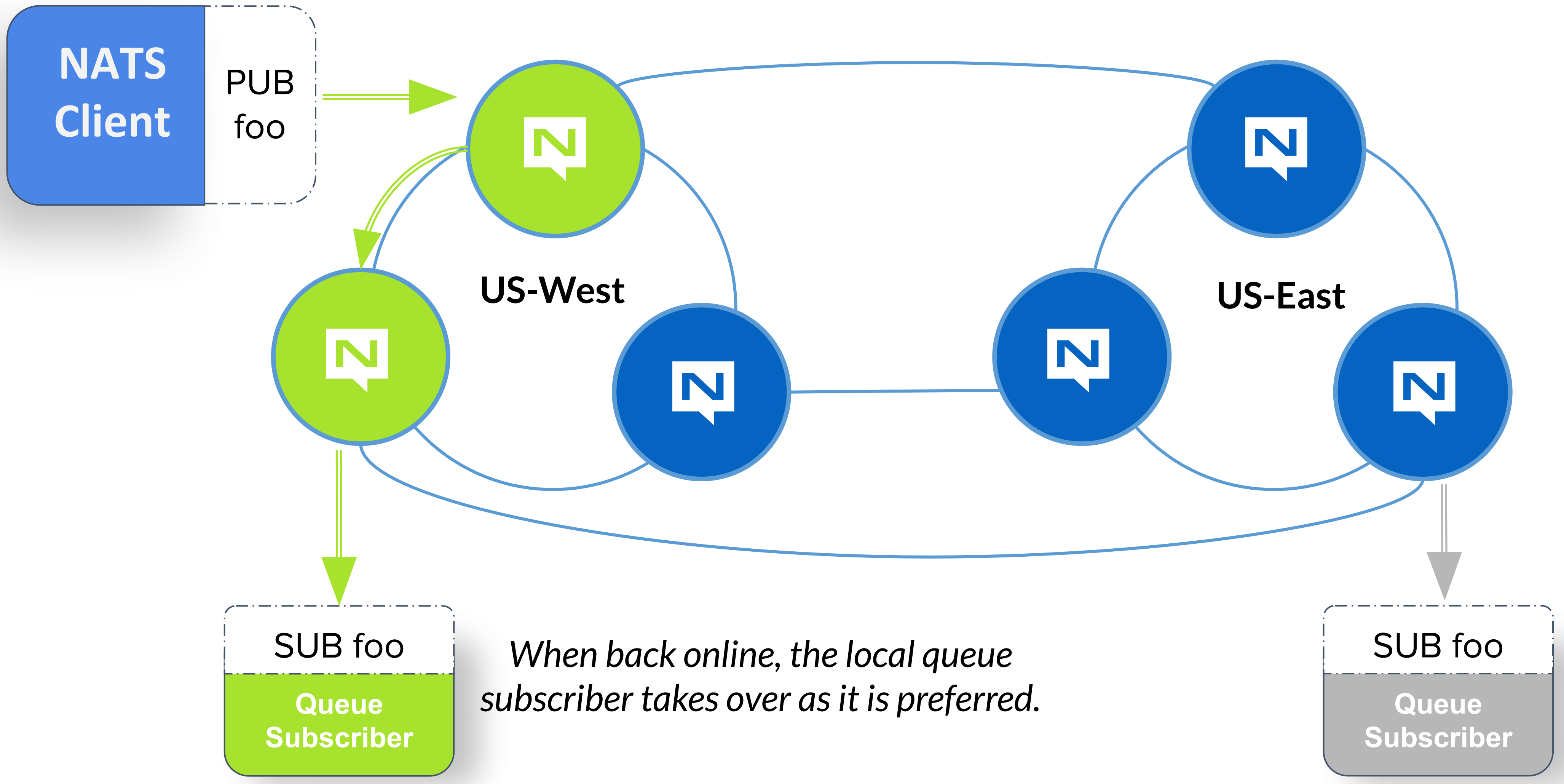


The remote queue subscriber takes over.

If there are multiple regions, NATS will choose the queue subscriber with the lowest RTT.

The local queue subscriber is disconnected

Geo Aware Queue Subscribers



When back online, the local queue subscriber takes over as it is preferred.

Geo Aware Queue Subscribers

What does this mean for you?

You have disaster recovery with….

- ✓ Runtime scalability
- ✓ Zero configuration
- ✓ The best latency

KubeCon | CloudNativeCon

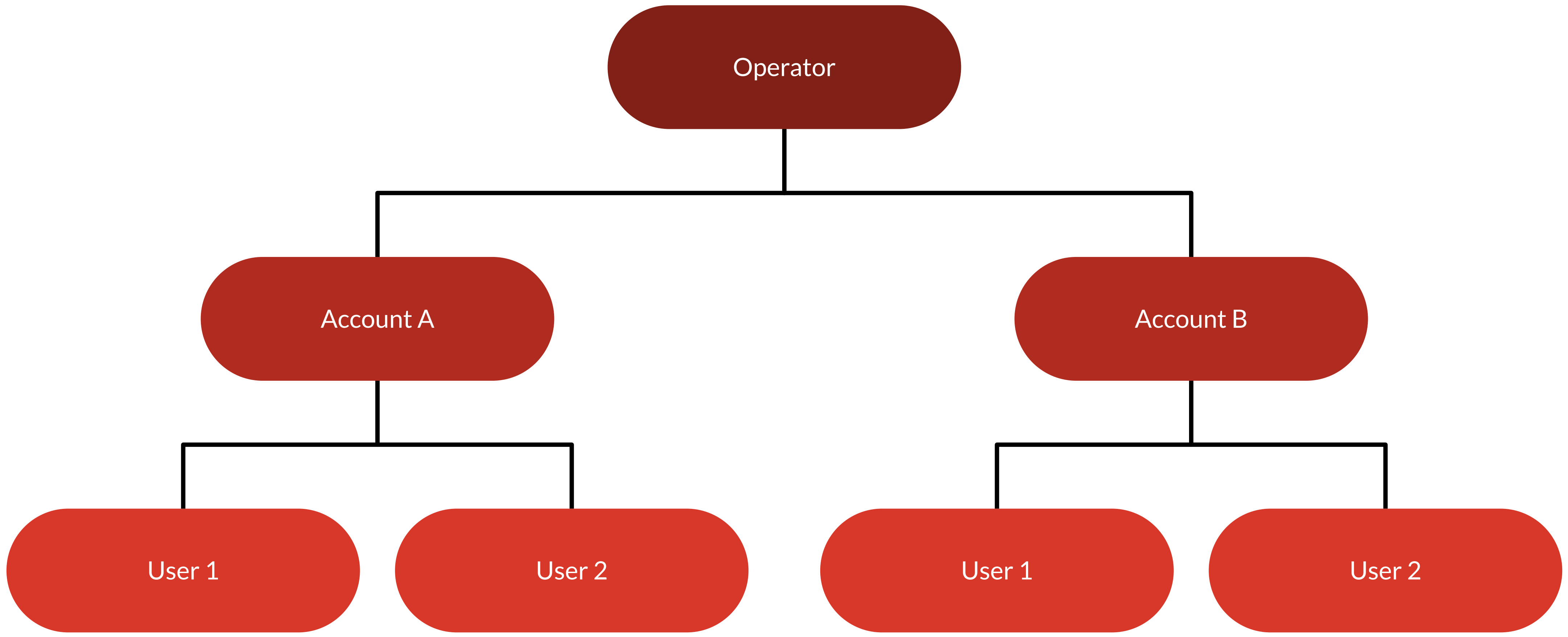
North America 2019

Security

Basic Security

- Full TLS Support: CA certificates, bidirectional support, default to most secure ciphers.
    - ✓ Support for DN or SAN in certificates for NATS user identity
- Support for standard user/password auth
- Permissions restrict who can send and receive on what subjects
- Change these through configuration reload at runtime with **zero downtime**.
- Operator Mode with NATS >= 2.0

Operator Mode



Operator Mode

NATS allows you to define **Operators**, **Accounts**, and **Users** within a NATS deployment.

- **Operator**: Root of trust for the system, e.g. An enterprise operator.
  - Create **Accounts** for account administrators. An account represents an organization with a secure context within the NATS deployment, for example a VAS system, an IT system monitoring group, a set of microservices, etc. Account creation would likely be managed by a central group.
- **Accounts** define limits and may securely expose services and streams
  - Account managers create **Users** with permissions
- **Users** have specific credentials and permissions.

Accounts

- Accounts are isolated communication contexts allowing secure multi-tenancy
- Bifurcate technology from business driven use cases
  - ✓ Data silos are created by design, not software limitations
- Easy, Secure and Cost Effective
  - ✓ One NATS deployment for operators to manage
  - ✓ Decentralized - organizations can self-manage
- Share data between accounts
  - ✓ Secure Streams and Services
  - ✓ Only mutual agreement will permit data flow

NKeys

Used by the NATS Identity authentication and authorization system.

- ED25519 based encoded keys made simple
  - ✓ Fast and resistant to side-channel attacks
  - ✓ Sign and Verify
- NATS servers **never see private keys**
  - ✓ Server sends nonce during connect then verifies the nonce signed by the user's private key, and user JWT signed by an account private key.
- JWT associate users with accounts and permission sets

Operator Mode

JWTs are used to represent identities in NATS

- User, Account, Cluster, or Server

User JWTs Contain

- Account NKey (Issuer)
- Public NKey (Subject)
- Friendly Name
- Permissions, limits, not-before and expiration

Creating JWTs

The `nsc` CLI manages JWTs

- Create the operator, accounts, and users
- Create Import and Exports
- Set account limits
- Set user permissions
- Deploy Account JWTs
    - ✓ Upload to the account server
    - ✓ Create configuration files with embedded JWTs

Managing JWTs

- Servers specify a **resolver**
- Memory Resolver
    - ✓ Embed JWTs in the server configuration
    - ✓ Ideal for deployments that do not often create accounts
- Account Server
    - ✓ Stores JWTs and servers will look them up as needed
    - ✓ Supports mirroring for performance and backup
    - ✓ Use when accounts are frequently created or expired

New Features

JetStream Tech Preview

We built JetStream to be the next-gen streaming system with the following goals:

The System Must
Be easy to configure and operate and be observable

The System Must
Be secure and operate well with NATS 2.0

The System Must
Scale horizontally and be applicable to high ingestion rate

The System Must
Support multiple use cases

The System Must
Self heal and always be available

The System Must
Have an API that is closer to core NATS

The System Must
Allow NATS messages to be part of an NMS as desired

The System Must
Display payload agnostic behavior

The System Must Not
Have third party dependencies

JetStream Tech Preview

JetStream supports

- ✓ **At-least-once** delivery
- ✓ Store messages and replay by time or sequence
- ✓ Embedded NATS server subsystem with an option to enable
- ✓ Wildcard Support
- ✓ NATS 2.0 Security
- ✓ Data at rest encryption
- ✓ Cleanse specific messages (GDPR)
- ✓ Horizontal scalability
- ✓ Persist **Message Sets** and replay via **Observables**

JetStream Sets and Observables

Message Sets are groups of persisted messages in JetStream that are created by applications at runtime, and have various policies set per unique message set.

Observables are application defined and control how message set messages are consumed.

Message Sets or Observables do not need to be configured or provisioned before use.

JetStream Message Sets

Message sets are defined by:

- ✓ Subjects (including wildcards)
- ✓ Retention Policy
- ✓ Limits
- ✓ Replica Count
- ✓ Storage Type

Applications create message sets by sending a specific JSON request.
The NATS clients will be extended to make this easy.

JetStream Message Set Retention

Message Sets support retention policies that determine when the message set's persisted data is rolled off.   These include:

- ✓ **Stream**: Messages are retained until limits are reached
- ✓ **Interest**: Messages are retained until all observables, either durable or ephemeral, have consumed a given message
- ✓ **Work Queue**: A message is retained until the first observable consumes the message. These type of observables most likely form a pull based group for a load balanced system.

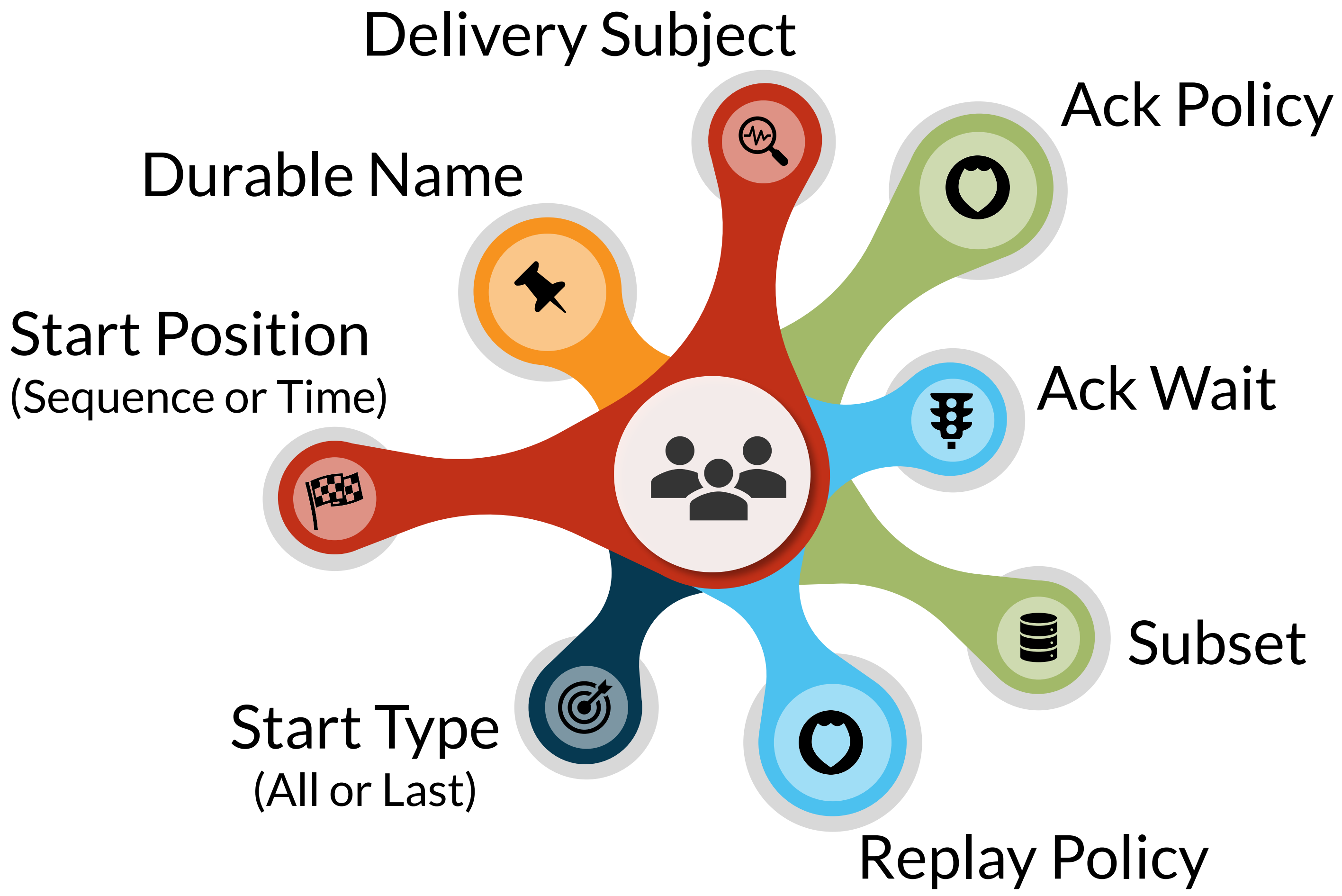
JetStream Message Set Limits

Limits are applied to message sets to determine when to roll off old data when applicable. These include:

- ✓ **Max Messages**: The number of messages the set will retain
- ✓ **Max Bytes**: The number of bytes the set will retain
- ✓ **Max Age**: The oldest message a message set will retain

JetStream Observable

An observable is defined by:



JetStream Observable Ack Policy

Ack policies dictate how an observable behaves when reading messages and indirectly defines what a “lost” message means. These policies also provide options to balance performance with quality of service. Ack Policies include:

- ✓ **None**: Require no acks for delivered messages
- ✓ **All**: A message and all previous messages are ack’d.
- ✓ **Explicit**: Every message requires an ack or nack.

JetStream Observables Replay

Replay policies determine the rate of replay.

Replay policies include:

- ✓ **Instant**: Replay messages as fast as possible.
- ✓ **Original**: Replay messages with the same timing as arrival.

These allow users to accurately replay original data for testing and for applications that need temporal message flow context.

JetStream and NATS Streaming

NATS Streaming will continue to be supported.

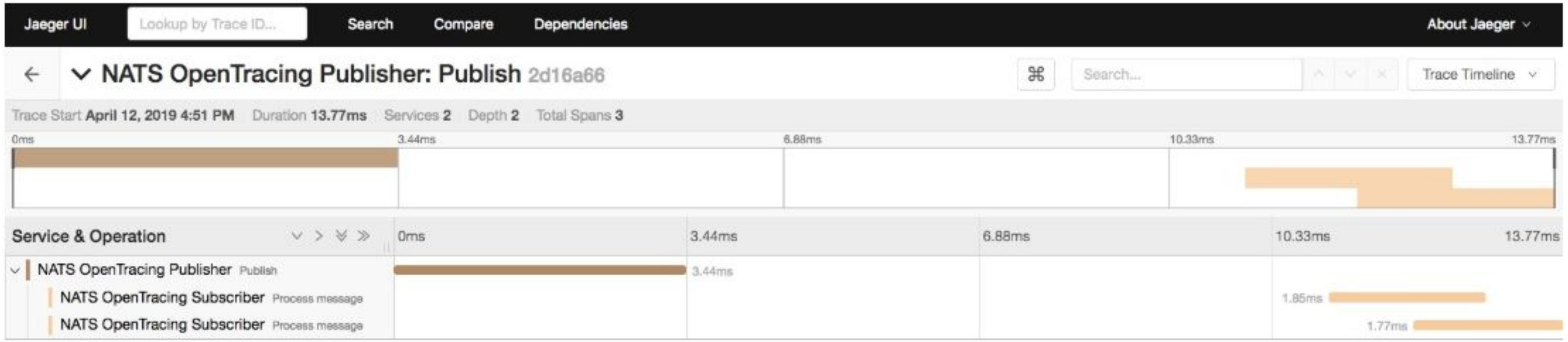
- ✓ 50 million docker downloads
- ✓ Deployed in production globally
- ✓ Bug fixes and Security fixes until June of 2022

Moving forward...

- ✓ New NATS enabled applications should prefer Jetstream
- ✓ We will provide a migration path to use JetStream
- ✓ New NATS streaming development will occur in JetStream

Distributed Tracing

OpenTracing reference implementations are provided for the **java** (not.java repo) and **go** (not.go repo). Using a simple API, encode and decode NATS messages to be traced with **Jaeger**.



NATS OpenTracing Publisher: Publish 2d16a66

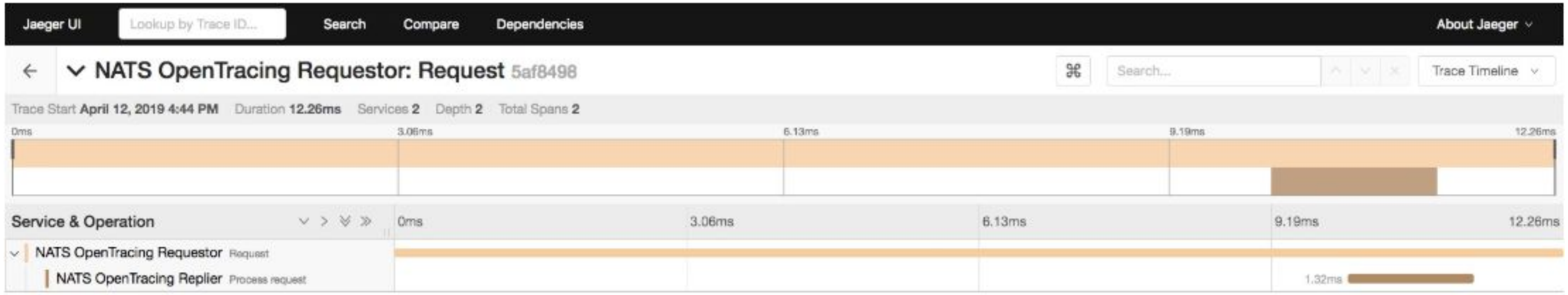
Trace Start April 12, 2019 4:51 PM Duration 13.77ms Services 2 Depth 2 Total Spans 3

Service & Operation

NATS OpenTracing Publisher Publish 3.44ms

NATS OpenTracing Subscriber Process message 1.85ms

NATS OpenTracing Subscriber Process message 1.77ms



NATS OpenTracing Requestor: Request 5af8498

Trace Start April 12, 2019 4:44 PM Duration 12.26ms Services 2 Depth 2 Total Spans 2

Service & Operation

NATS OpenTracing Requestor Request

NATS OpenTracing Replier Process request 1.32ms

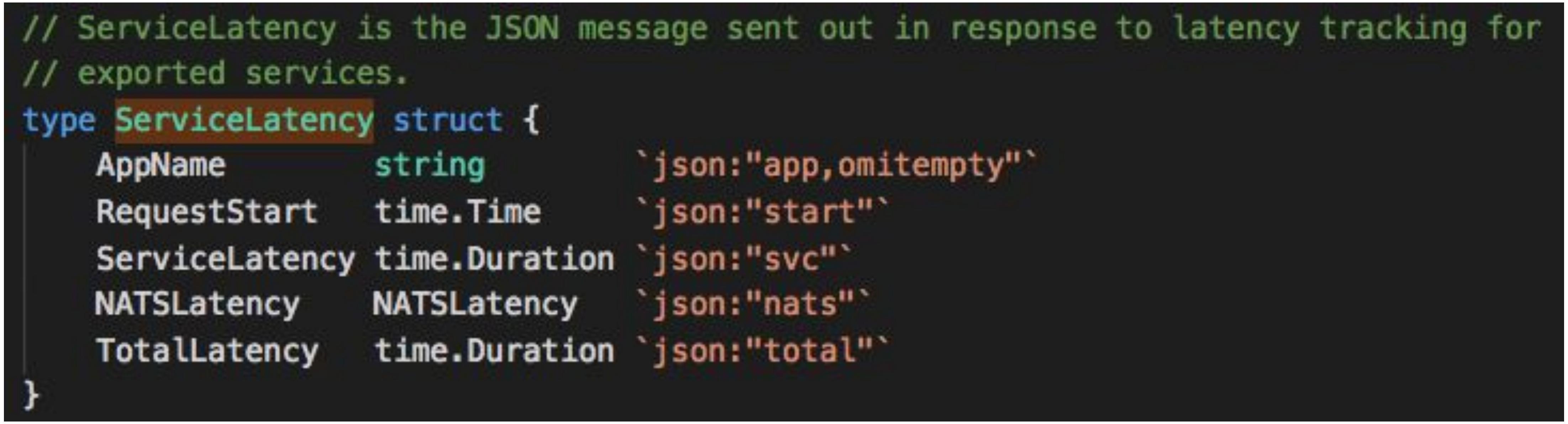
Integrations

We're continuing to integrate NATS with other technologies.

- Spring.io
  - ✓ NATS Spring Boot Starter
  - ✓ NATS Cloud Stream Binder
- NATS Kafka Bridge
  - ✓ Support for bridging to and from Kafka topics
- NATS MQSeries Bridge
  - ✓ Support for bridging to and from IBM MQ series topics

Service Observability

Using the account usage import, operators can now monitor service latency using the usage export.



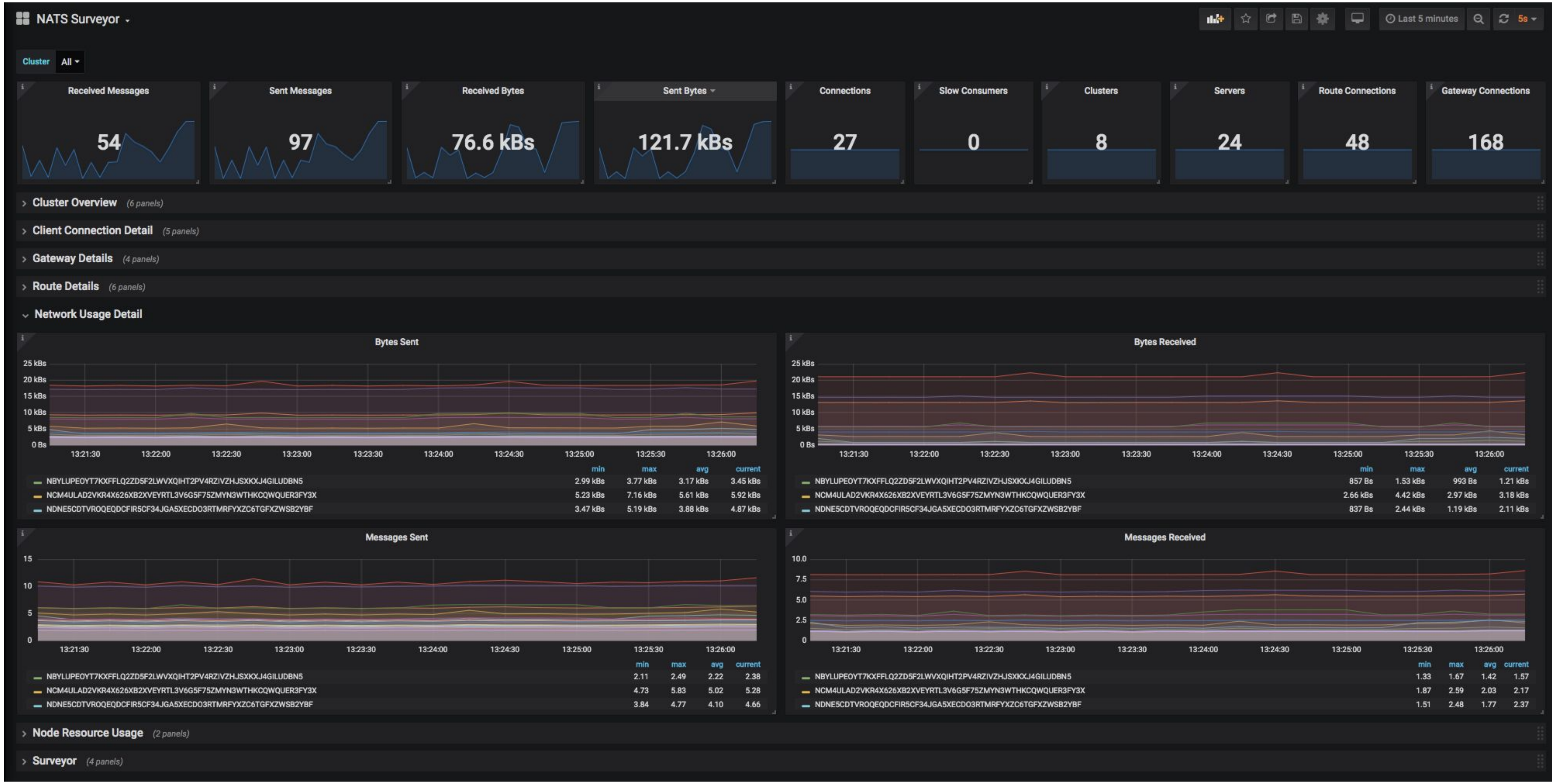
```go
// ServiceLatency is the JSON message sent out in response to latency tracking for
// exported services.
type ServiceLatency struct {
    AppName        string        `json:"app,omitempty"`
    RequestStart   time.Time     `json:"start"`
    ServiceLatency time.Duration `json:"svc"`
    NATSLatency    NATSLatency   `json:"nats"`
    TotalLatency   time.Duration `json:"total"`
}
```

NATS Surveyor

Surveyor can monitor your entire deployment from a single container or process paired with Prometheus and Grafana.

- ✓ Provides a comprehensive view of entire NATS deployment
- ✓ No sidecars to deploy
- ✓ K8s, docker compose, or bare metal deployments
- ✓ Run using Docker Compose
- ✓ Requires NATS 2.0 Security and System Credentials

NATS Surveyor



Received Messages	Sent Messages	Received Bytes	Sent Bytes	Connections	Slow Consumers	Clusters	Servers	Route Connections	Gateway Connections
54	97	76.6 kBs	121.7 kBs	27	0	8	24	48	168

Cluster Overview (6 panels)
Client Connection Detail (5 panels)
Gateway Details (4 panels)
Route Details (6 panels)
Network Usage Detail

Bytes Sent

	min	max	avg	current
NBYLUPEOYT7KXFFLQ2ZD5F2LWVXQIHT2PV4RZIVZHJSXKXJ4GILUDBN5	2.99 kBs	3.77 kBs	3.17 kBs	3.45 kBs
NCM4ULAD2VKR4X626XB2XVEYRTL3V6G5F75ZMYN3WTHKCQWQUER3FY3X	5.23 kBs	7.16 kBs	5.61 kBs	5.92 kBs
NDNE5CDTVROQEQDCFIR5CF34JGA5XECDO3RTMRFYXZC6TGFXZWSB2YBF	3.47 kBs	5.19 kBs	3.88 kBs	4.87 kBs

Bytes Received

	min	max	avg	current
NBYLUPEOYT7KXFFLQ2ZD5F2LWVXQIHT2PV4RZIVZHJSXKXJ4GILUDBN5	857 Bs	1.53 kBs	993 Bs	1.21 kBs
NCM4ULAD2VKR4X626XB2XVEYRTL3V6G5F75ZMYN3WTHKCQWQUER3FY3X	2.66 kBs	4.42 kBs	2.97 kBs	3.18 kBs
NDNE5CDTVROQEQDCFIR5CF34JGA5XECDO3RTMRFYXZC6TGFXZWSB2YBF	837 Bs	2.44 kBs	1.19 kBs	2.11 kBs

Messages Sent

	min	max	avg	current
NBYLUPEOYT7KXFFLQ2ZD5F2LWVXQIHT2PV4RZIVZHJSXKXJ4GILUDBN5	2.11	2.49	2.22	2.38
NCM4ULAD2VKR4X626XB2XVEYRTL3V6G5F75ZMYN3WTHKCQWQUER3FY3X	4.73	5.83	5.02	5.28
NDNE5CDTVROQEQDCFIR5CF34JGA5XECDO3RTMRFYXZC6TGFXZWSB2YBF	3.84	4.77	4.10	4.66

Messages Received

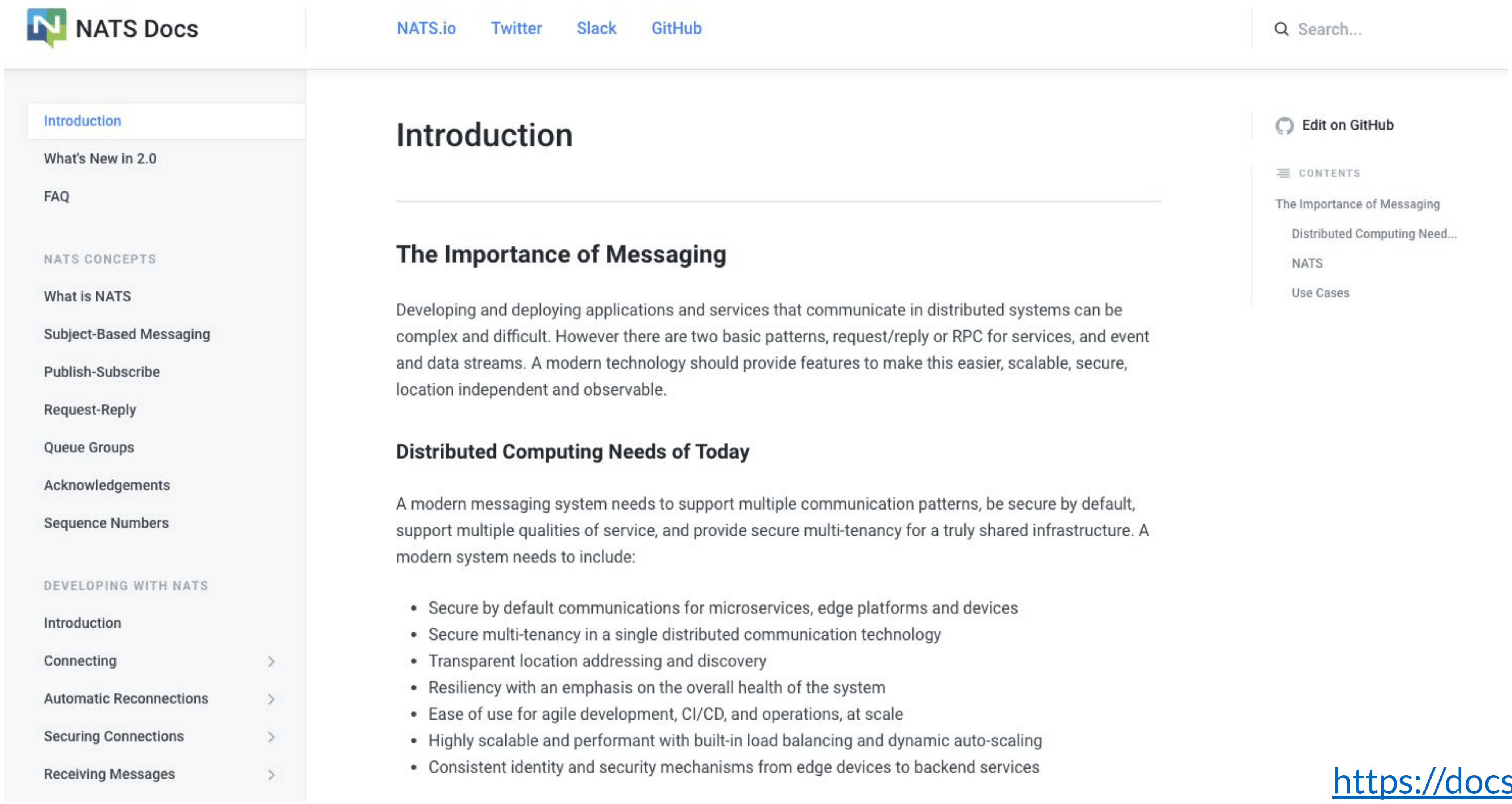
	min	max	avg	current
NBYLUPEOYT7KXFFLQ2ZD5F2LWVXQIHT2PV4RZIVZHJSXKXJ4GILUDBN5	1.33	1.67	1.42	1.57
NCM4ULAD2VKR4X626XB2XVEYRTL3V6G5F75ZMYN3WTHKCQWQUER3FY3X	1.87	2.59	2.03	2.17
NDNE5CDTVROQEQDCFIR5CF34JGA5XECDO3RTMRFYXZC6TGFXZWSB2YBF	1.51	2.48	1.77	2.37

Node Resource Usage (2 panels)
Surveyor (4 panels)

Kubernetes Deployments

- A single command line to install (NATS v2 auth included)
  - ✓ curl -sSL https://nats-io.github.io/k8s/setup.sh | sh
- Stateful Sets (used via installer)
  - ✓ NATS Server / NATS Streaming Server official examples
  - ✓ NATS Operator also changing to use StatefulSets internally
- Monitoring
  - ✓ Surveyor Installation

Extensive Documentation



NATS Docs

NATS.io Twitter Slack GitHub

Search...

Introduction

What's New in 2.0

FAQ

NATS CONCEPTS

What is NATS

Subject-Based Messaging

Publish-Subscribe

Request-Reply

Queue Groups

Acknowledgements

Sequence Numbers

DEVELOPING WITH NATS

Introduction

Connecting

Automatic Reconnections

Securing Connections

Receiving Messages

Introduction

The Importance of Messaging

Developing and deploying applications and services that communicate in distributed systems can be complex and difficult. However there are two basic patterns, request/reply or RPC for services, and event and data streams. A modern technology should provide features to make this easier, scalable, secure, location independent and observable.

Distributed Computing Needs of Today

A modern messaging system needs to support multiple communication patterns, be secure by default, support multiple qualities of service, and provide secure multi-tenancy for a truly shared infrastructure. A modern system needs to include:

- Secure by default communications for microservices, edge platforms and devices
- Secure multi-tenancy in a single distributed communication technology
- Transparent location addressing and discovery
- Resiliency with an emphasis on the overall health of the system
- Ease of use for agile development, CI/CD, and operations, at scale
- Highly scalable and performant with built-in load balancing and dynamic auto-scaling
- Consistent identity and security mechanisms from edge devices to backend services

Edit on GitHub

CONTENTS

The Importance of Messaging

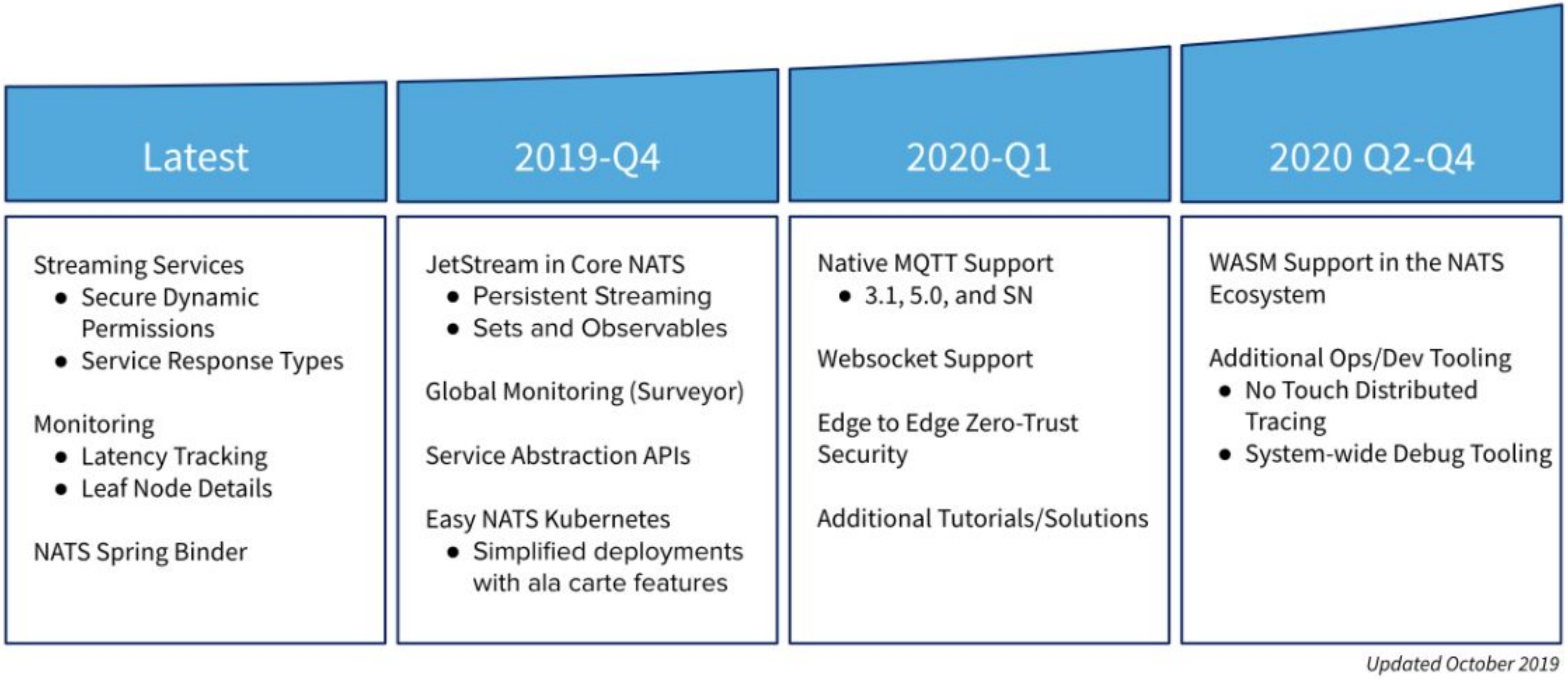
Distributed Computing Need...

NATS

Use Cases

https://docs.nats.io

Roadmap



Latest	2019-Q4	2020-Q1	2020 Q2-Q4
Streaming Services • Secure Dynamic Permissions • Service Response Types Monitoring • Latency Tracking • Leaf Node Details NATS Spring Binder	JetStream in Core NATS • Persistent Streaming • Sets and Observables Global Monitoring (Surveyor) Service Abstraction APIs Easy NATS Kubernetes • Simplified deployments with ala carte features	Native MQTT Support • 3.1, 5.0, and SN Websocket Support Edge to Edge Zero-Trust Security Additional Tutorials/Solutions	WASM Support in the NATS Ecosystem Additional Ops/Dev Tooling • No Touch Distributed Tracing • System-wide Debug Tooling

Updated October 2019

Demos

Questions

Thank you!

KubeCon | CloudNativeCon

North America 2019