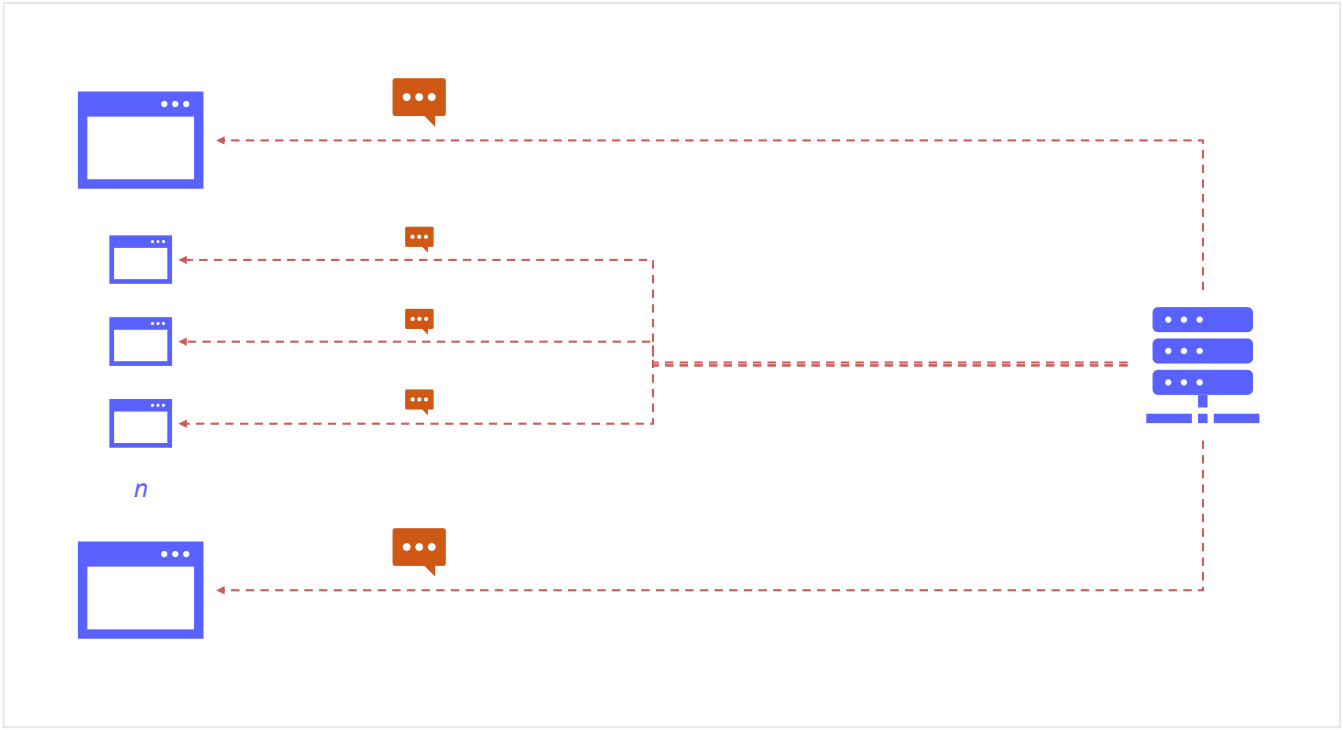
100 XP

Pub/Sub

7 minutes

Distributed microservices often respond to events that occur in other microservices. This pattern is commonly referred to as *event-driven architecture*.

In the retail company scenario, the various microservices must communicate with each other and react to events from other microservices. Today, they manually store the addresses for other microservices, but this technique is challenging to manage, increases technical complexity, and doesn't scale well.



Event aggregation using Pub/Sub

An *event aggregator* is a middleware solution that aggregates events across the entire solution in a scalable and straightforward to manage way.

Here, you learn how the Pub/Sub feature of Azure Cache for Redis can serve as a middleware to simplify the communication between components of your application. Pub/Sub can help your application components subscribe to other's events and publish their own events.

The **Pub/Sub** feature of Azure Cache for Redis routes messages between application components. Microservices can use this feature to subscribe to messages or publish messages.

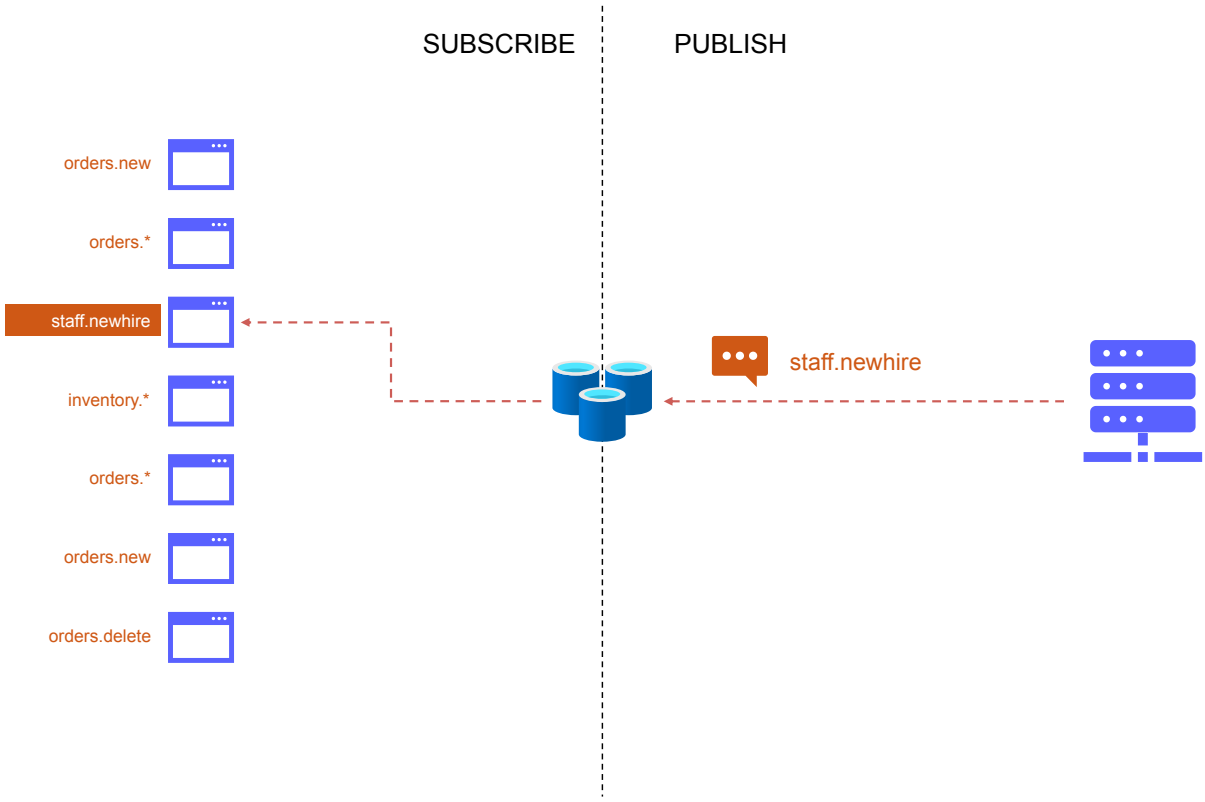
Azure Cache for Redis handles the routing of messages to the appropriate destinations without each microservice knowing where each of their messages should go.

This feature can significantly simplify the requirement of microservices reacting to events across the entire solution.

Subscribing to channels

Clients can subscribe to a topic or range of topics using a string value. In Redis nomenclature, topics are referred to as **channels**. For example, a client that wants to subscribe to the **staff.newhire** channel can use the appropriate command with the `staff.newhire` string value.

Once the client is subscribed, all messages in the **staff.newhire** channel are sent to that specific client.



Redis includes a `SUBSCRIBE` command used to subscribe to one or more channels. This command is flexible enough to subscribe to a space-delimited list of channels.

Subscribe to a single known channel

The most common use case of the `SUBSCRIBE` command is to subscribe to a single channel. For example, the command can be used to subscribe to the **staff.newhire** channel.

Redis

```
SUBSCRIBE staff.newhire
```

Subscribe to multiple known channels

The `SUBSCRIBE` command can also be used to subscribe to multiple channels simultaneously. To subscribe to multiple channels, separate each channel with a single space in the channels list.

As an example of subscribing to multiple channels, use the `SUBSCRIBE` command to subscribe to the **orders.delete** and **orders.new** channels simultaneously.

Redis

```
SUBSCRIBE orders.new orders.delete
```

Subscribe to a pattern of channels

The `PSUBSCRIBE` command uses glob-style patterns to subscribe a client to any channel that matches the specific pattern. There are three primary operators that you can use in a glob-style pattern:

Expand table

Operator	Description	Example	Matches	Does not match
`?`	Matches any single character	`l?arn`	`learn`, `loarn`	`larn`, `lern`
`*`	Matches any content (including none)	`lear*`	`learn`, `learaeiou`	`larn`, `lern`
`[]`	Matches only characters within the list	`le[ao]rn`	`learn`, `leorn`	`lern`, `leurn`

For example, use the `PSUBSCRIBE` command to subscribe to all channels that begin with a prefix of **inventory.**

Redis

```
PSUBSCRIBE inventory.*
```

Alternatively, as another example, use the `PSUBSCRIBE` command to subscribe to all channels with the suffix of **.new**.

Redis

```
PSUBSCRIBE *.new
```

In this example, the `PSUBSCRIBE` command is used to subscribe to all channels that include **orders** or **staff** as a whole word in the name.

Redis

```
PSUBSCRIBE *orders* *staff*
```

Unsubscribe from channels

Once a client no longer wishes to receive messages from a specific channel, the client should unsubscribe from that channel or pattern of channels.

Redis includes an `UNSUBSCRIBE` command to remove a client's subscription from one or more channels. The command is flexible enough to support unsubscribing from multiple channels, like the `SUBSCRIBE` command.

To unsubscribe from a pattern of channels, Redis includes a `PUNSUBSCRIBE` command that functions similar to the `PSUBSCRIBE` command.

There are three common ways to unsubscribe from a channel or channels.

Unsubscribe from one or more known channels

The client can unsubscribe from specific channel or channels by providing one or more channels to the `UNSUBSCRIBE` command.

Redis

```
UNSUBSCRIBE staff.newhire
UNSUBSCRIBE orders.new orders.delete
```

Unsubscribe from one or more channel patterns

The client can also unsubscribe from one or more patterns by providing each of those patterns to the `PUNSUBSCRIBE` command.

```Redis
PUNSUBSCRIBE inventory.*
PUNSUBSCRIBE inventory.* orders.* staff.*
```

Unsubscribe from all channels or patterns of channels

If the client wants to unsubscribe from all known subscriptions, the client can invoke the `UNSUBSCRIBE` command with no arguments.

```Redis
UNSUBSCRIBE
```

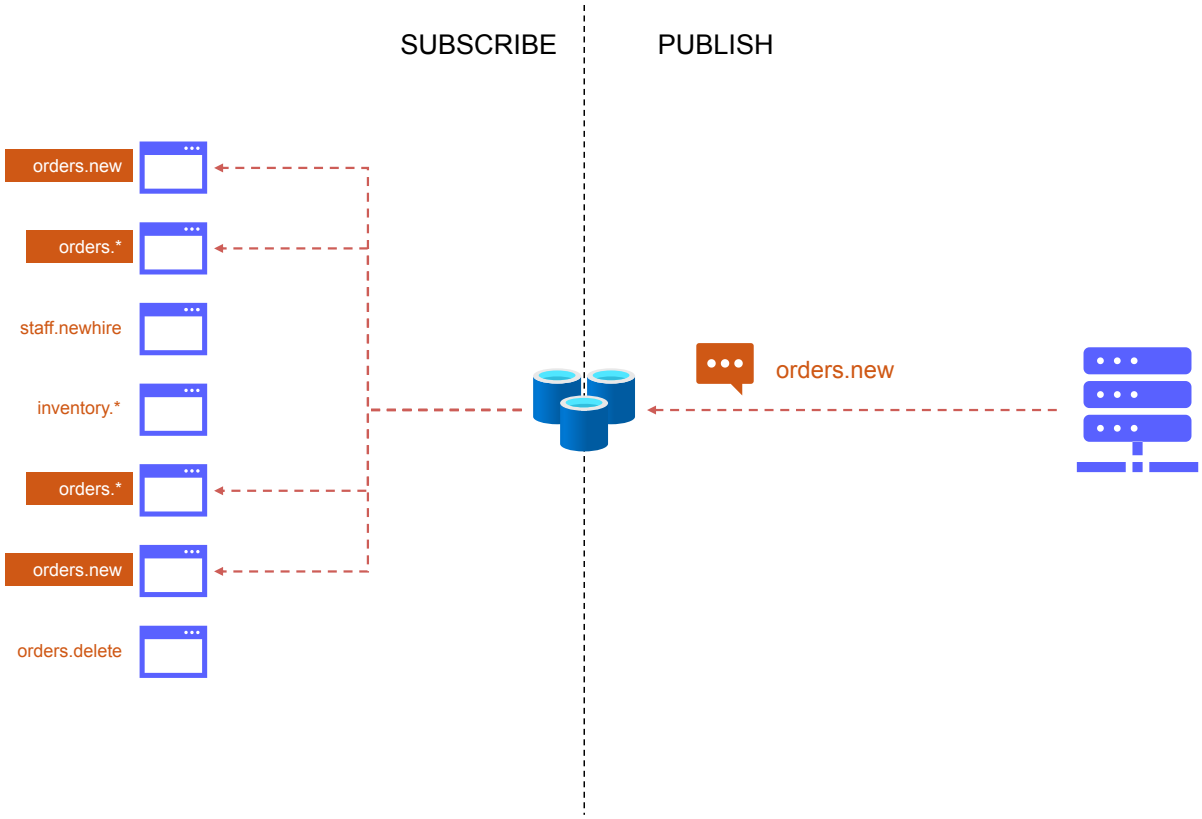
The `UNSUBSCRIBE` command unsubscribes the client from all channels the client specifically subscribed to. The command doesn't unsubscribe the client from pattern-based subscriptions. The `PUNSUBSCRIBE` command is used to unsubscribe the client from pattern-based subscriptions. Invoke the `PUNSUBSCRIBE` command with no arguments to unsubscribe the client from all pattern-based subscriptions.

```Redis
PUNSUBSCRIBE
```

Publishing a message to a channel

Any client can send messages by publishing the message to a channel. Azure Cache for Redis automatically routes the message to clients with matching subscriptions. For example, if a client publishes a message to the **orders.new** channel, Azure Cache for Redis routes the message to:

- Clients subscribed explicitly to the **orders.new** channel
- Clients subscribed to a pattern that matches **orders.new** (Example: **orders.***)



Redis includes a PUBLISH command that takes in two arguments. The first argument is the channel's name where the message is published. The second argument is the string content of the message.

Consider these two examples. The PUBLISH command is used to send a message with the string content **sad348957298s534gh** to the **orders.delete** channel. The PUBLISH command is also used to send a message with content **02a67b49-9da1-487e-8b49-d5aad3f514ae** to the **staff.newhire** channel.

Redis

```
PUBLISH orders.delete sad348957298s534gh

PUBLISH staff.newhire 02a67b49-9da1-487e-8b49-d5aad3f514ae
```

Tip

If a client subscribes to a pattern and a known channel with overlap, the client could potentially receive multiple messages. For example, consider a client subscribed to **staff.*** and **staff.retire**. If a message is published to the **staff.retire** channel; the client will receive the message twice. One receipt will be for the known channel name subscription, and the other will be for the pattern match subscription.

Next unit: Exercise - Create an Azure Cache for Redis instance

Continue >