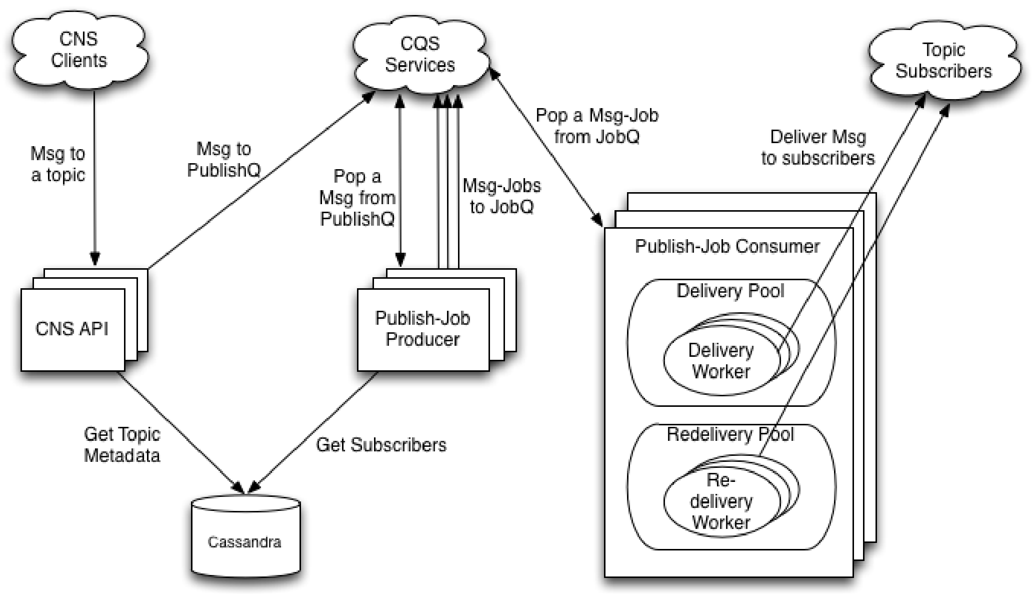
CNS Architecture

Overview

This document gives a high-level architecture for CNS

Description



CNS API Server

The CNS API servers get Publish() requests from clients, does validation of the request (authenticated and correctly formatted) and sends the Message to CQS for handling by the EndpointPublishJobProducers.

EndpointPublishJob Producer

This component is responsible for handling the messages sent to CQS by the CNS API server. It calls RecieveMessage() on the JobQ to get the message, reads a list of subscribers for this topic from Cassandra and partitions the subscriber-list into multiple Endpoint-Publish-Jobs. The Endpoint-Publish-Jobs represent some subset of the subscribers to which notification needs to be sent. These jobs are then sent to CQS to be picked up by the EndpointPublishJobConsumers. Finally, the Publish-Jobs are deleted from CQS.

EndpointPublishJob Consumers

This component is responsible for handling a Endpoint-Publish-Job which contains all the information needed to send a notification out to some subset of the subscribers. It calls RecieveMessage() on the PublishJobQ, gets a

Endpoint-Publish-Job, queues up each notification in its Delivery Handler Pool which has a thread-pool processing it so they are sent in parallel. The notifications which fail are re-queued in the ReDelivery Handler Pool which has a seperate set of threads processing them. The retry logic is as specified by the Amazon SNS spec. Once all notifications for a specific Endpoint-Publish-Job are sent, the job is deleted from CQS.

Availability

Availability is one of the the highest requirement for CNS. With our current design, we can have numerous API servers which make the CNS service available. The availability of CNS assumes availability of CQS and Cassandra.

No Message Loss

Once the CNS API servers accept a message (a successfull call to Publish), the message is persisted in CQS and is only deleted when it is handled. This gaurantees that if any of the components of CNS are down, the job is retained for any available CNS component when it comes up.

Performance

Adding more components to CNS does impact performance but this design choice was made with availability, stability and ease of development in mind. Each component uses in-memory cache to optimize performance.

Linear Scalability

This design lends itself to good linear scalability. Each of the compoents only communicate with each other through the CQS-service and Cassandra both of which are linearly scalable. We can start any number of the three components to add capacity into the system.

Limitations

- This designh does require a lot of resources to send out notofications. It requries a fairly large CQS instance, enough Cassandra capacity and multiple processes representing the multiple components.
- The extra components add latency to the notifications