Asynchronous messaging

Integration

Application Integration Challenges

* varied and multiple integration points
* need for various complex data relationships
* reliance of application on validation and standardization of input data
* the need for integration process to interact with apps at run-time
* independent applications

Application Integration Styles

* file transfer – two different apps transfer files using a common agreed schema
* shared database- one app insets data into db, other app reads it
* remote method invocation – one app invokes processes that are part of a different app
* messaging
  + point-to-point –producer gives message to single consumer
  + pub/sub – multiple consumers consume same message
  + hub/spoke – share information of processes among ‘spokes’ attached to the hub
    - uses a central message broker which does-
      * routing
      * message transformations
      * application of rules to the content of message
    - detaches coupling between application
    - can change broker subscriber rules without changing the connected apps
    - bottlenecks under a heavy load

Messaging Models

* synchronous – wait, blocking
* asynchronous – continue working

Enterprise Service Bus ESB

* provides patterns and specifications to handle the challenges of integration
* middleware tool
* used to distribute work among connected components of an app
* designed to provide a uniform means of moving work
* provides applications with the ability to connect to the bus
* subscribes to messages based on simple structural and business policy rules
* decouples applications with the help of a messaging server
* essential integration principles of ESB
  + orchestration – how the systems or apps integrate with each other
  + transformation – for differences between the schemas of 2 different apps
  + transportation – identifies correct route for messages
  + mediation –mapping, SLA, and transaction management
* Camel Architecture – contains various components and layers, Camel Endpoint is responsible for integration, Camel processors response for message filtering
* ServiceMix Architecture – components bound using JBI components like SOAP

Enterprise Integration Architecture

* 3 layers
  + IT management services – responsible for handling security
  + enterprise service bus – contains all the artifacts required to manage application integration
    - deployed message flows that identify the endpoints and coordinating them
    - messaging models – aynch or synchronous
    - transport protocols that identifies what protocol will be used to facilitate data exchange or processes
  + business logic– final layer which consumes the integration layer, can contain interactions, processes, and information that are exchanged between apps and generally coordinate with each other through ESB
* Apache Camel Features
  + open source java framework
  + simplifies integration by providing
    - concrete implementations of all widely used EIPs (Enterprise Integration Patterns)
    - connectivity to a great variety of transports and APIs
    - easy to use Domain Specific Languages (DSLs) to wire EIPs and transports together
  + integral part of JBoss Fuse
  + has routing engine and processors and camel context

Enterprise Integration Patterns

* catalog of design patterns
* used for developing systems to integrate new and existing software in a business environment
* creates common integration interface, easing access to a cluster of different apps
* provides the capability to couple applications enough to ensure the changes do not impact functions
* sets data exchange frequency and quantity between apps
* controls how an applications functionality impacts that of others

Patter Classification

* endpoint pattern
* message construction
* routing
* transformation
* message endpoint
* content-based router
* message filter
* dynamic router
* recipient list
* splitter
* aggregator
* content filter

Asynchronous Messaging

* communication method wherein the system puts a message in a message queue
* does not require an immediate response to continue processing
* also known as fire-and-forget information exchange or message-oriented middleware
* solves the problem of intermittent connectivity

Kafka

* $ apt-get update
* use Zookeeper
* topicName, kafkaServer, zooKeeperHost, serializerClass

Essential Patterns

* challenges
  + providing proper content negotiation
  + proper protocol mediation
  + handling various diversified channels for communication
* routing, channel, message construction patterns
* EIP Enterprise Integration Pattern
  + provides standard integration approach
  + messaging is an integral part of integration
  + resolves various identified problems in integration
  + provides solution for integration issues
  + elements
    - message construction
    - end-point
    - messaging channel
    - transformation
* EIP Messaging Channels
  + router
  + CBR content based routing – interprets message to route to correct recipient
  + aggregator – when receiving from multiple sources, combine into 1 message
  + splitter – to split message into small chunks
  + resequencer
  + filter
  + recipient list – maintain a list of receivers to send a message too simultaneously

Kafka

* implementing pub/sub
  + need zookeeper
  + import properties like port where kafka is running, serializer, topic, Kafka producer
* implementing message queue
  + new KafkaProducer(configProperties);
* Kafka API
  + central between producer/consumer/disk
  + broker manages all of Kafka’s services
    - broker creates cluster
  + zookeeper is a dependency of Kafka
* StreamsBuilder KStream for streaming .toStream()
* Message Structure
  + contained in an envelope which contains-
  + header – message attributes
  + body contains the payload - business data
  + only final recipient reads the body
  + root, properties, JMS transport, body, transport folders
    - message metadata
    - header values
    - application properties
    - provider properties
    - standard properties
  + Camel Message
    - contains basic structure of moving data
    - created by message producer
    - message exchange
      * acts as Message container during routing
      * creates communication link between producer and consumer
      * MEP message exchange patter contains InOnly and InOut patterns
    - headers, attachments, body
* Message Destinations
  + Queue – point-to-point
    - local, remote, DLQ, expiry
  + Topic – publish/subscribe
    - subject, subscription type (durable and non durable)
* Message Transformation Pattern
  + message translator – will switch the message format, aka XML to JSON
  + message enricher – adds to message
  + content filter – based on specified criteria
  + envelope wrapper
* AMP Asynchronous Messaging Protocol
  + flexible remoting protocol for sending multiple asynch request/response pairs over the same connection
  + requests and responses are both collections of unordered key/value pairs
  + rich set of intenet applications
    - traditional client-server APIs to custom efficient RPC protocols
    - distributed peer-to-peer messaging topologies
    - can implement more advanced patterns such as a spoke topology
  + basic types
  + compound types
* Message Consumption Essentials

Kafka

* a distributed, replicated service for publish-subscribe messaging system
  + composed of clustered servers
  + messages broken up between cluster servers
  + leaders and followers
* partitioning
  + logs are partitioned to fit on a server
  + enables scalability arbitrary size
  + enables parallelism
  + a topic is split into partitions
  + partitions are strong ordered and immutable
  + they can exist on different servers
  + they enabled scalability
  + producers can assign a message to a partition
  + messages in a given partition are always sent to the same consumer
  + data consumed in the order it was received
* topics
  + a logical group of messages
  + messages are published to a topic
  + producers always publish data to a specific topic
  + consumers always read data from a topic
  + a producer can assign a partition within a topic
  + each message in a topic is sent to one group member
  + can be created at the CLI and can be modified
* replicas
  + prevent data loss
  + are backups of a partition
  + replication factor is configurable in the broker config
  + data exists on multiple servers
  + server can fail and data is still accessible
  + never written to or read from and do not increase parallelism
* Kafka can be used for
  + metrics
  + log aggregation
  + stream processing
  + messaging
  + website activity tracking
  + event sourcing
* architecture
  + cluster takes input from producers and sends them to consumers
  + requires Apache ZooKeeper
  + ZooKeeper stores configuration data for distributed services
  + used primarily by brokers
  + broker is an instance of a Kafka server
* Producers
  + publish data to the cluster
  + can run in sync or async
  + decide on partitioning semantics
  + assign messages to a partition
  + are replication aware
  + messages are sent to the partition leader to be replicated
  + sync mode – blocks until active replicas are written
  + async mode – blocks on the write to the leader only
* Consumers
  + read the data
  + different APIs have different characteristics
  + High Level Consumer API
    - abstract away details about offsets for the consumer – just gives data
    - high level consumers should be threaded
  + SimpleConsumer API
    - API gives greater control over consumption so you can
      * consumer a subset of partitions
      * read messages multiple times
      * transaction management
* Brokers
  + clustered data repositories
  + must have 1 or more brokers
  + each broker in a cluster has a unique broker.id
  + brokers use ZooKeeper which keeps
    - state information
    - leader elections
  + form .9 only brokers use ZooKeeper
  + brokers decouple processing from data producers
  + buffer unprocessed messages
  + broker structure can produce messages independently of consumer messages
  + broker structure is the reason for Kafka’s very high performance