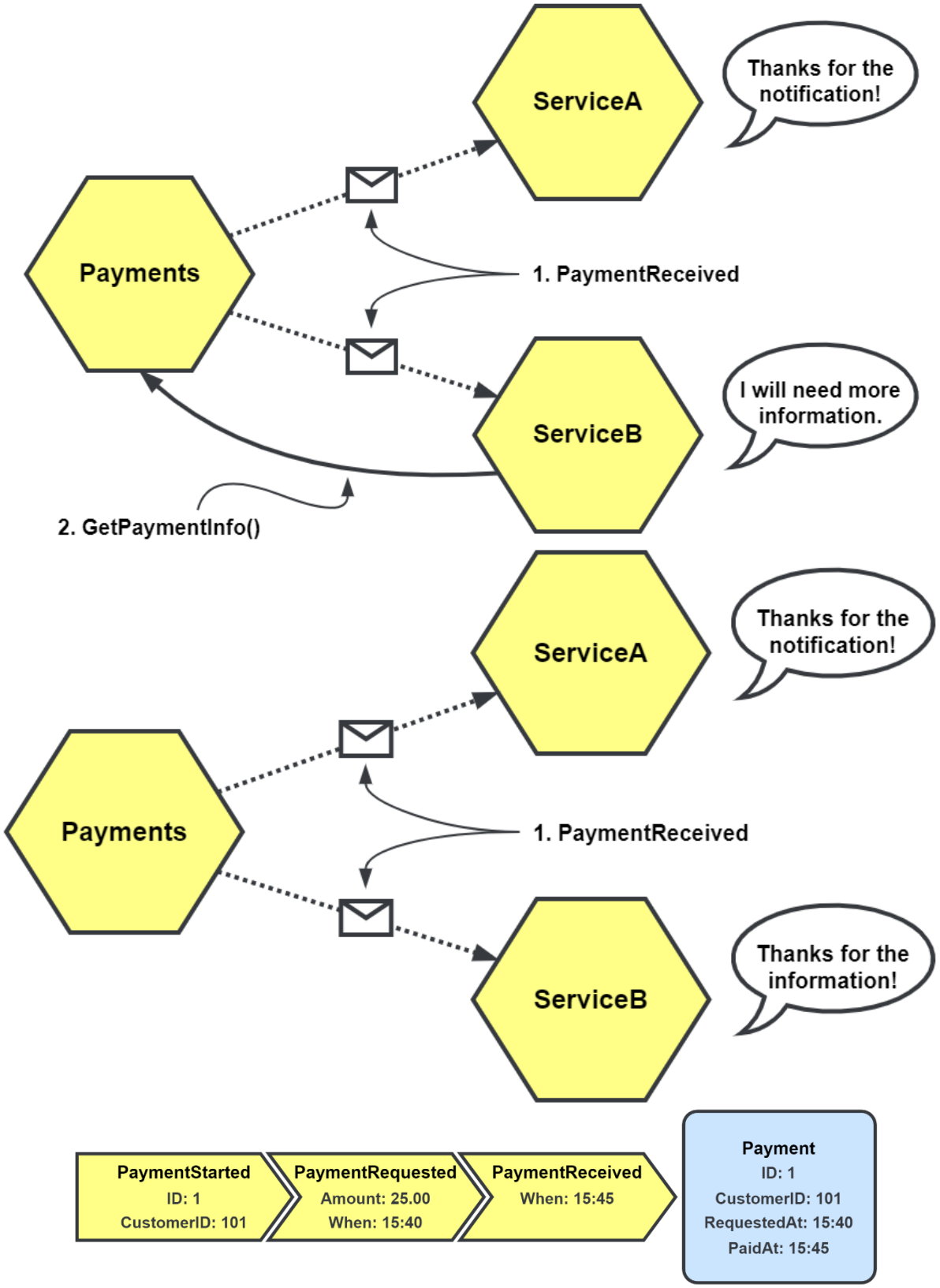
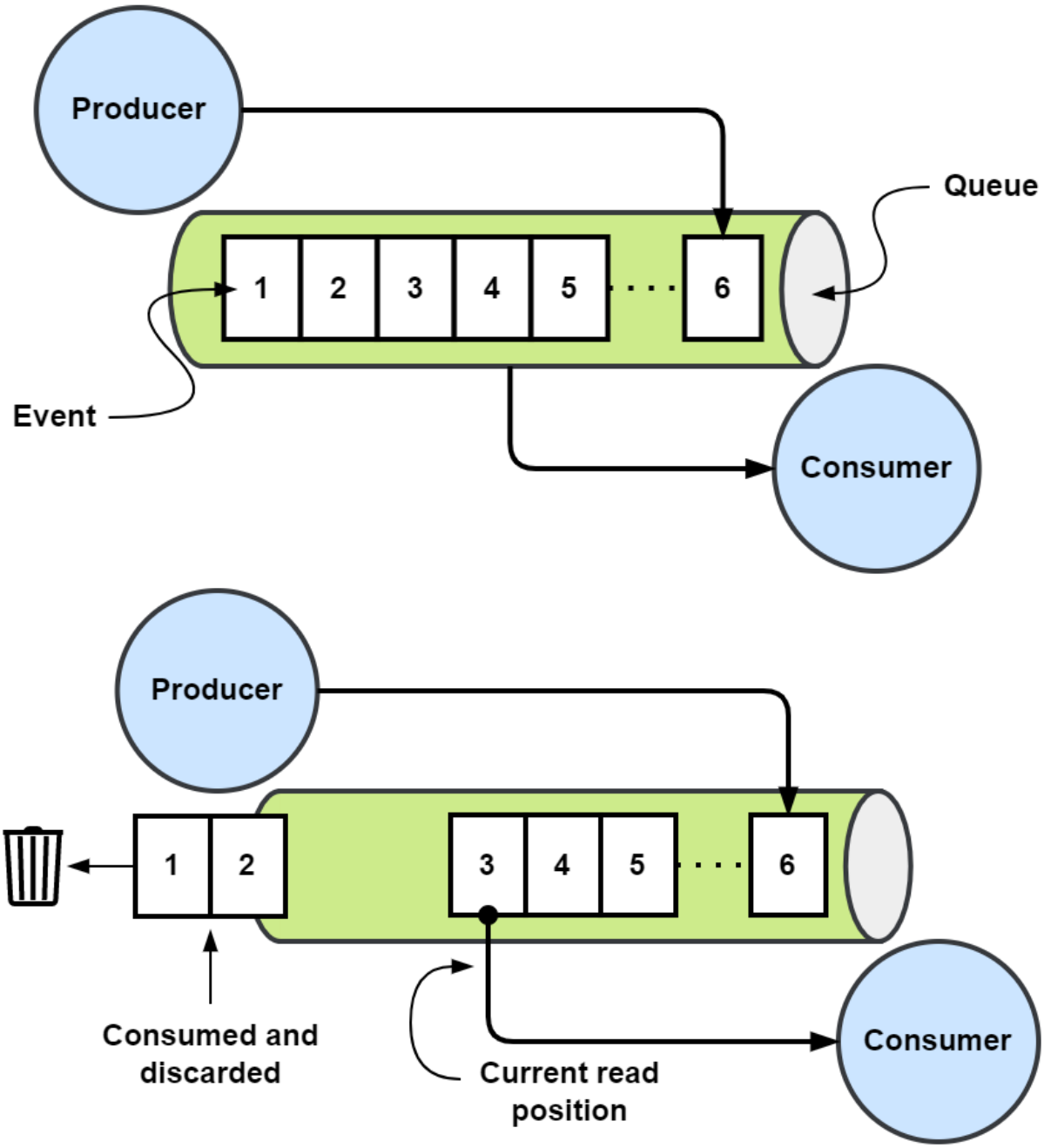
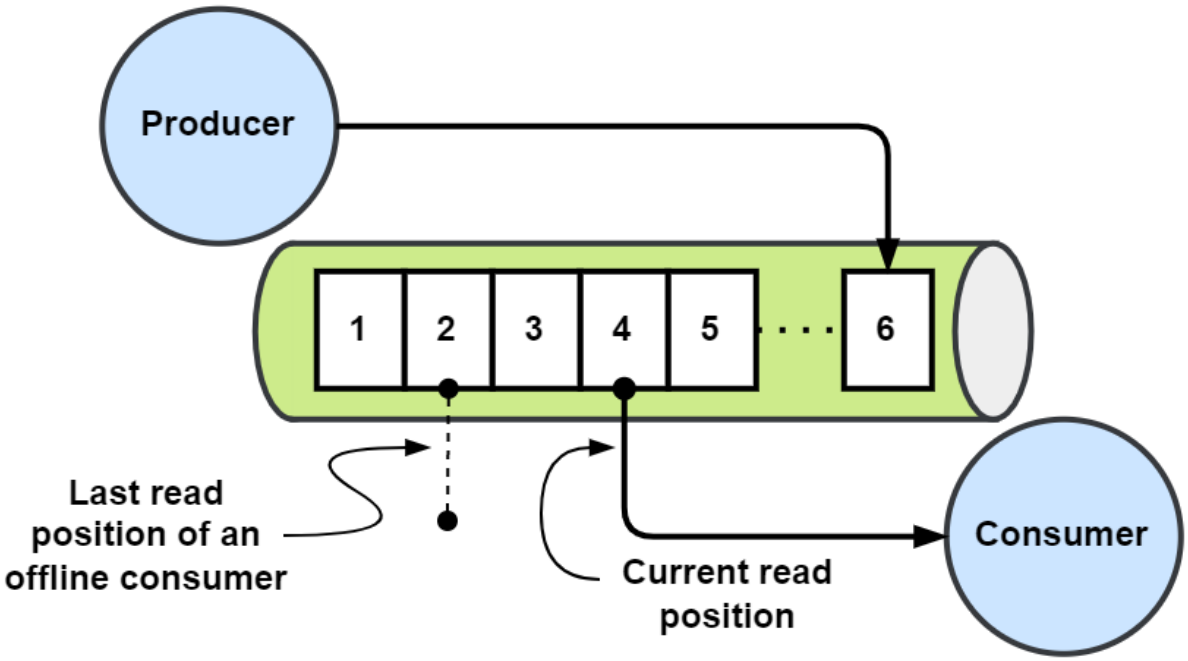
Chapter 1: Introduction to Event-Driven Architectures

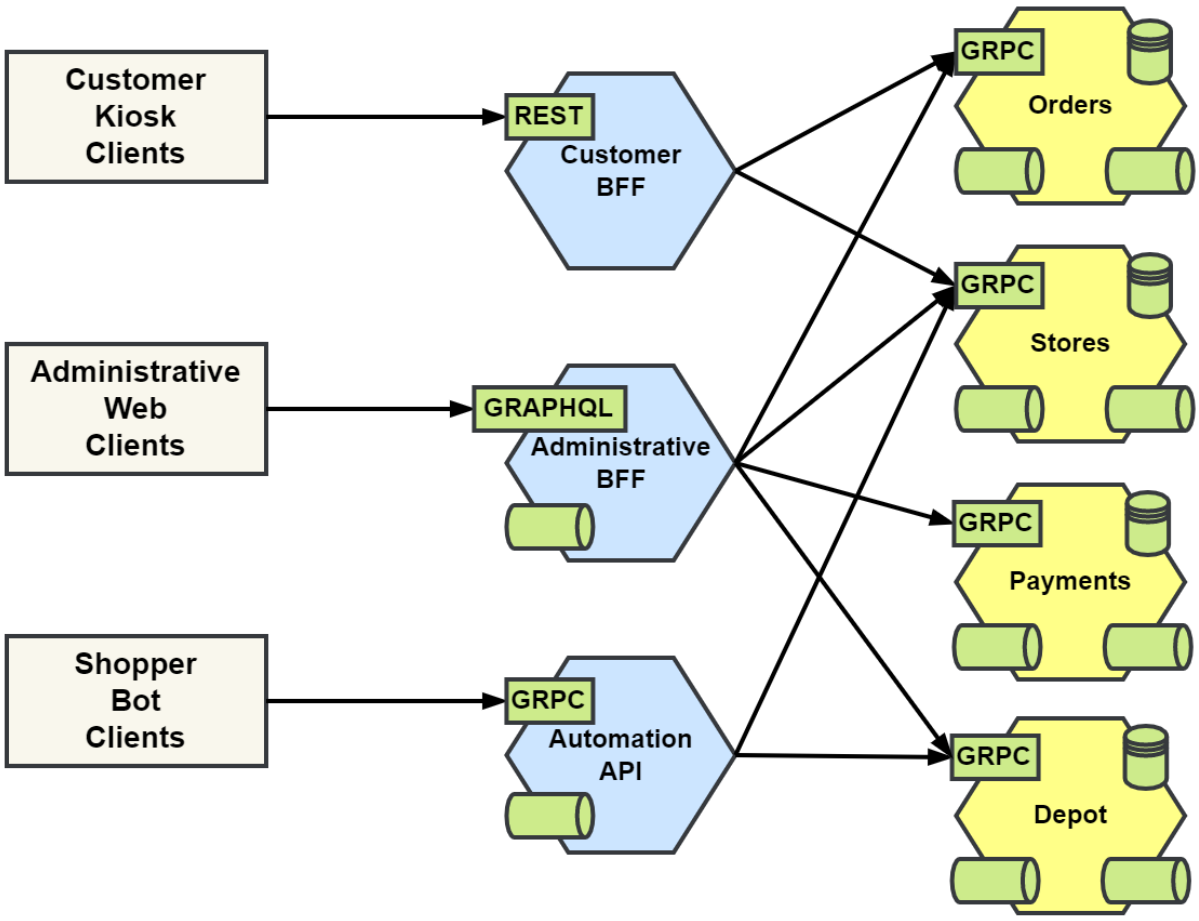


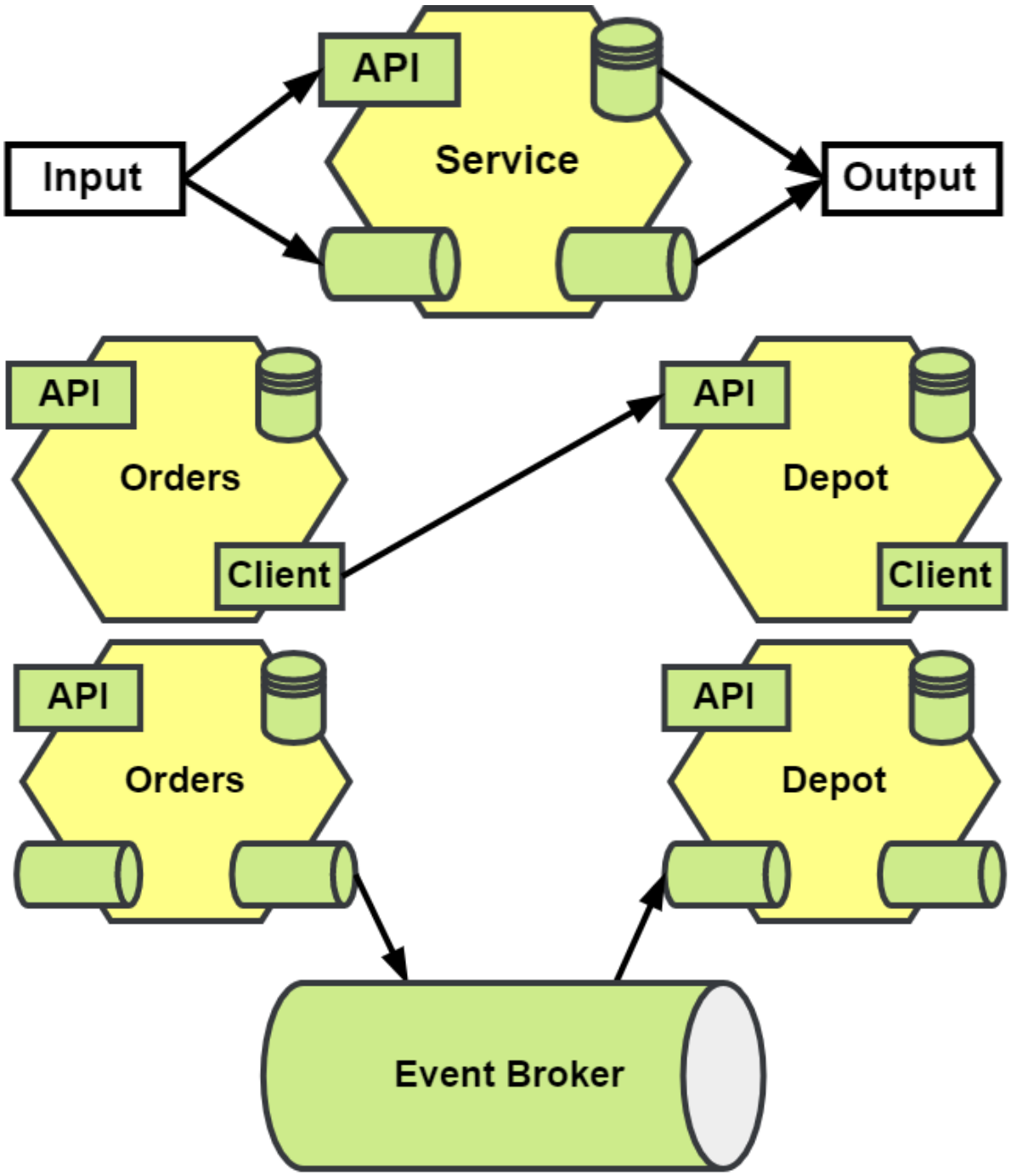

Producer
Queue
1
2
3
4
5
6
Event
Consumer
Producer
1
2
3
4
5
6
Consumed and
discarded
Current read
position
Consumer

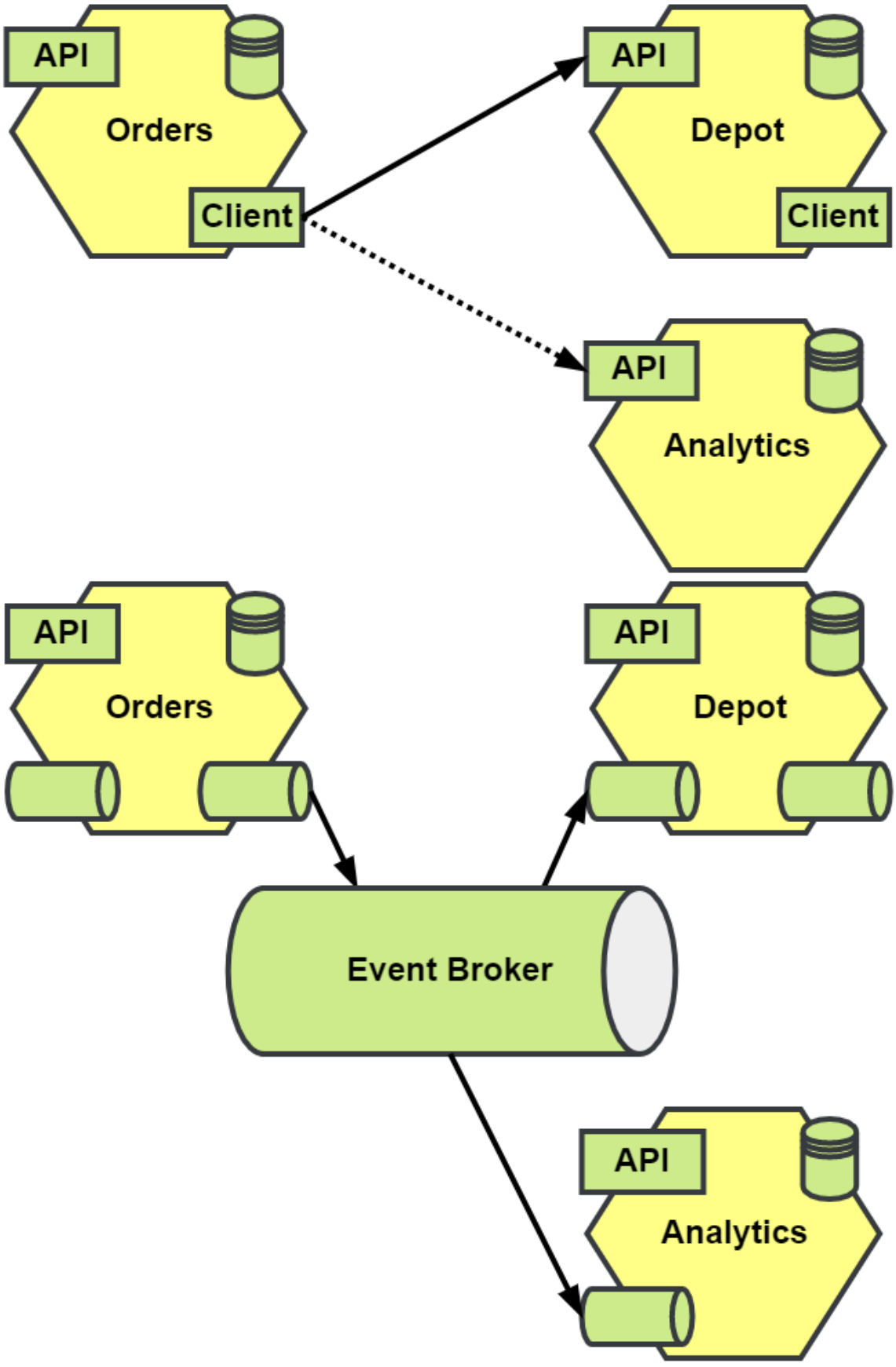


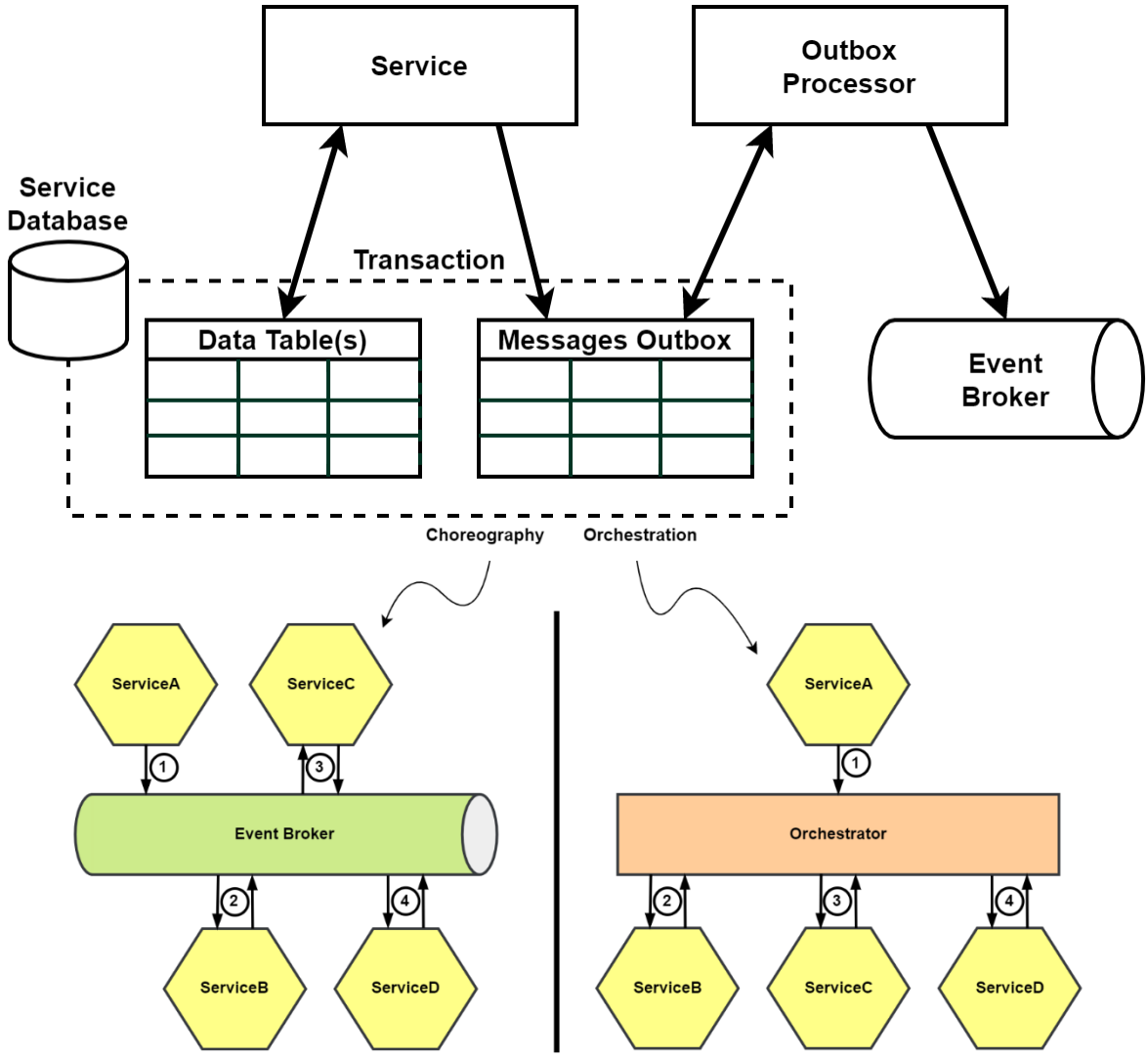
Events Table

ID	ObjectID	ObjectType	Version	Type	Data	
101	1	Payment	1	PaymentStarted	{ ... }	Events for **Payment**: 1
102	1	Payment	2	PaymentRequested	{ ... }	
103	1	Payment	3	PaymentReceived	{ ... }	
104	2	Payment	1	PaymentStarted	{ ... }	Events for **Payment**: 2
...	...	...	...	...	...	

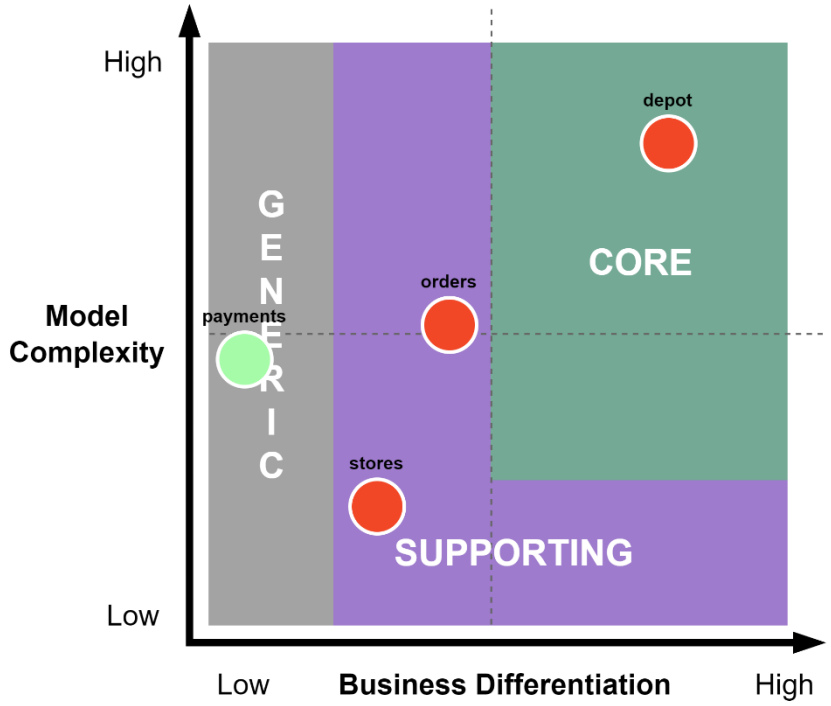


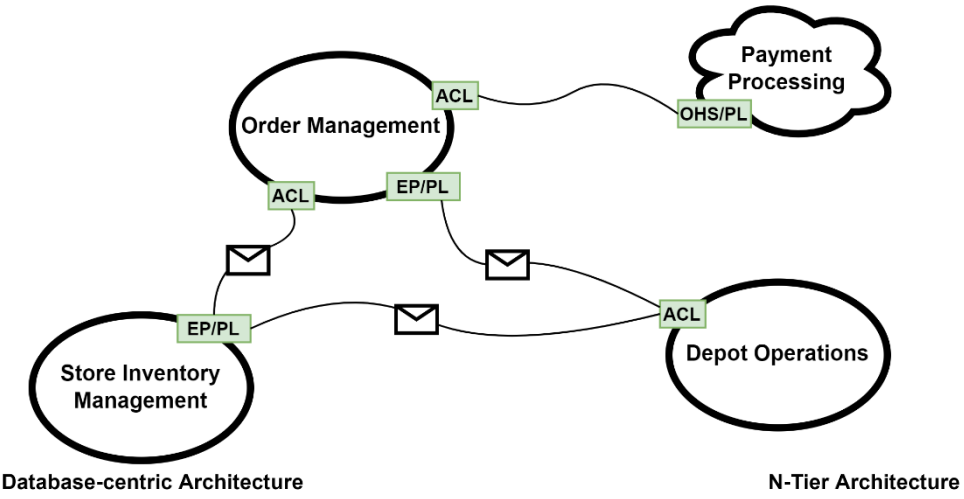

API
Service
Input
Output
API
Orders
Client
API
Depot
Client
API
Orders
API
Depot
Event Broker

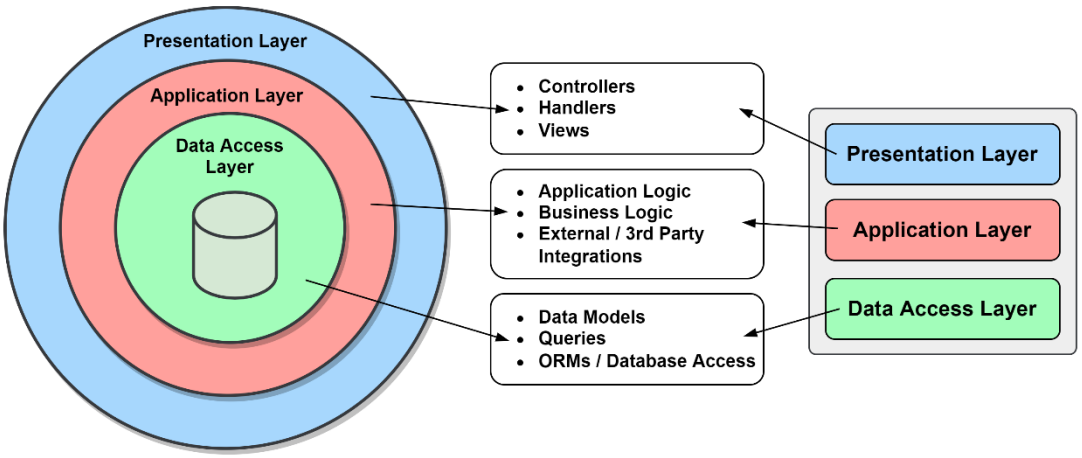

API
Orders
Client
API
Depot
Client
API
Analytics
API
Orders
API
Depot
Event Broker
API
Analytics

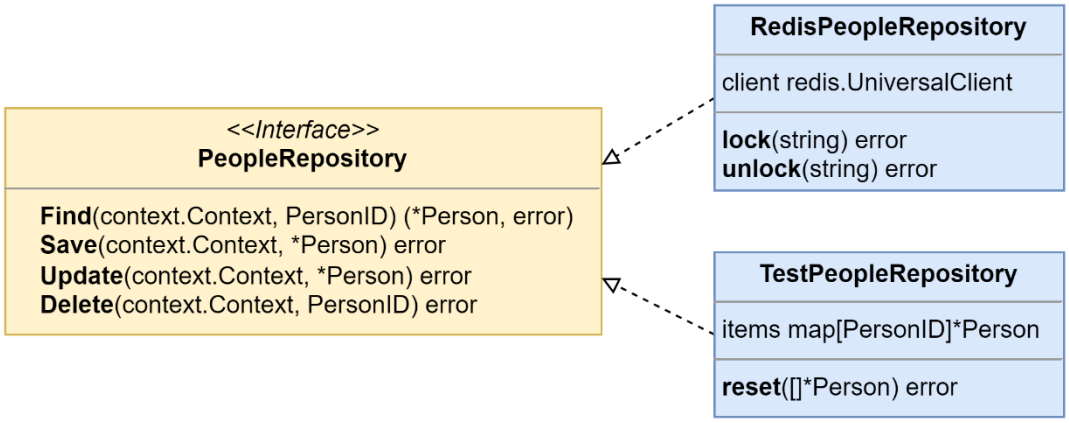

Service
Outbox Processor
Service Database
Transaction
Data Table(s)
Messages Outbox
Event Broker
Choreography
Orchestration
ServiceA
ServiceC
Event Broker
ServiceB
ServiceD
ServiceA
Orchestrator
ServiceB
ServiceC
ServiceD
1
2
3
4

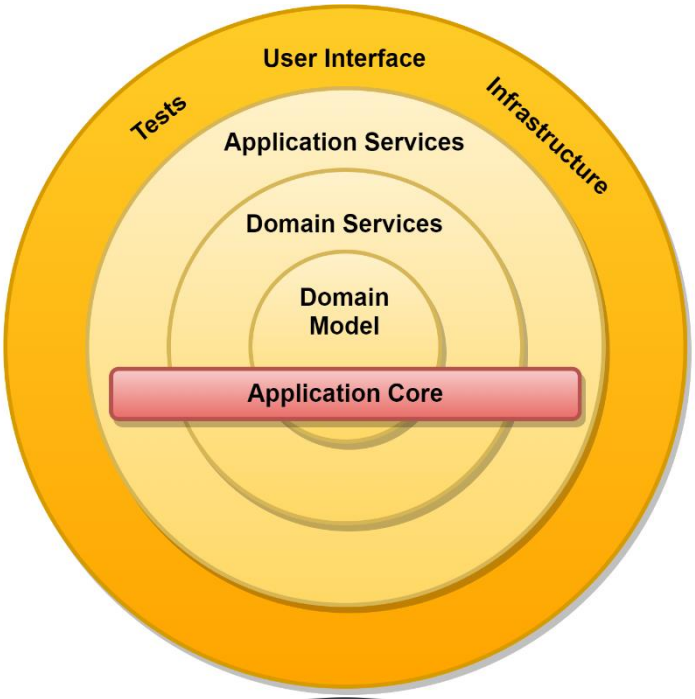
Chapter 2: Supporting Patterns In Brief

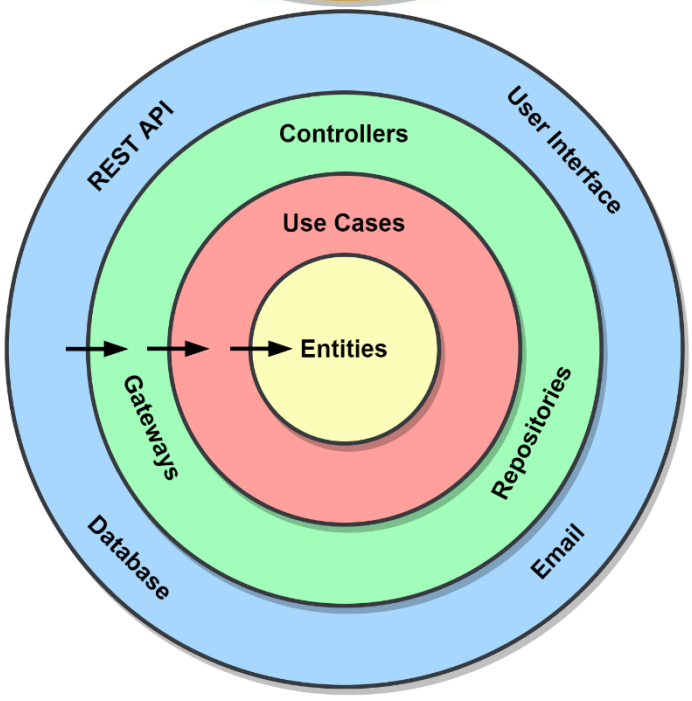


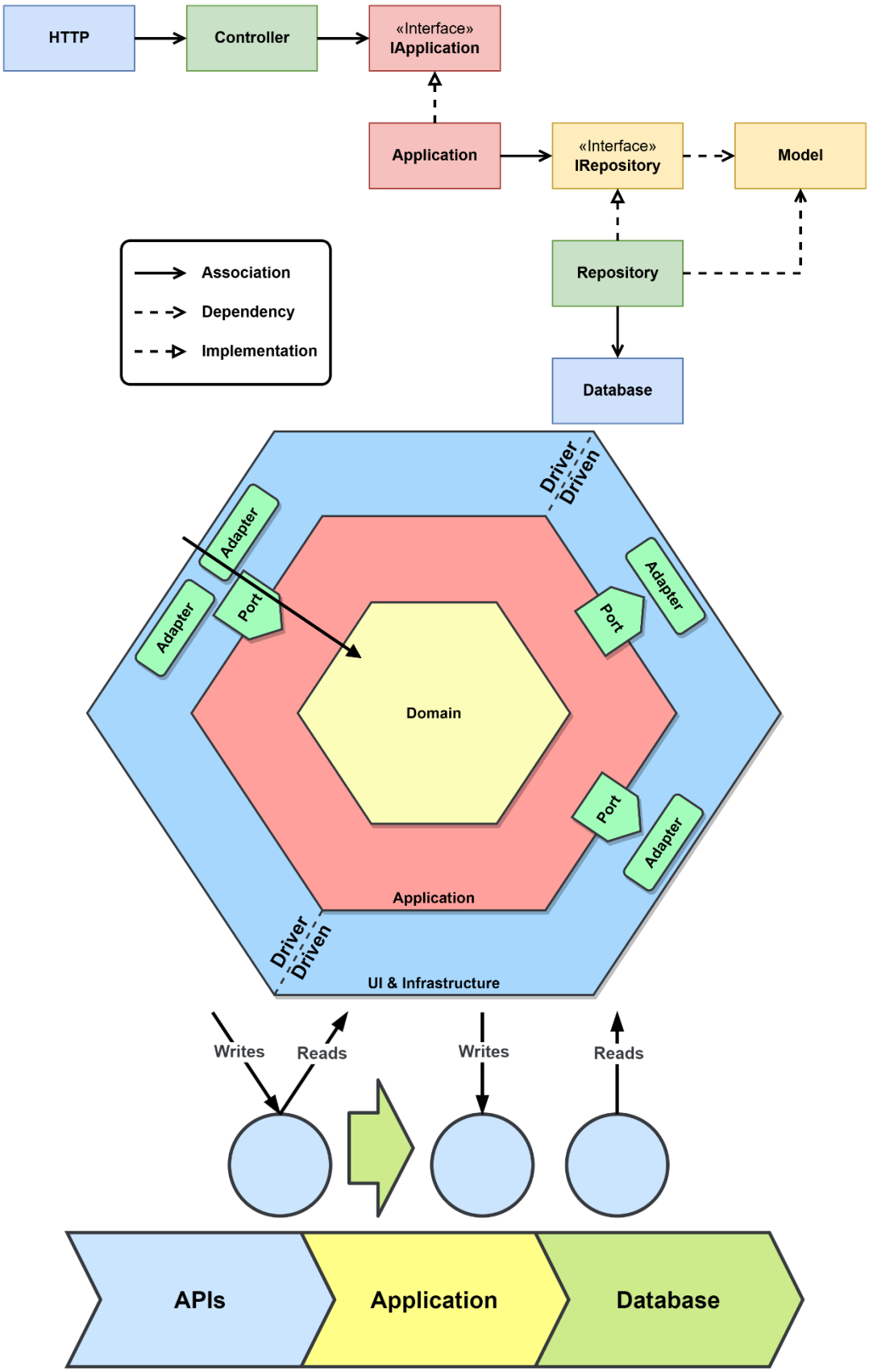


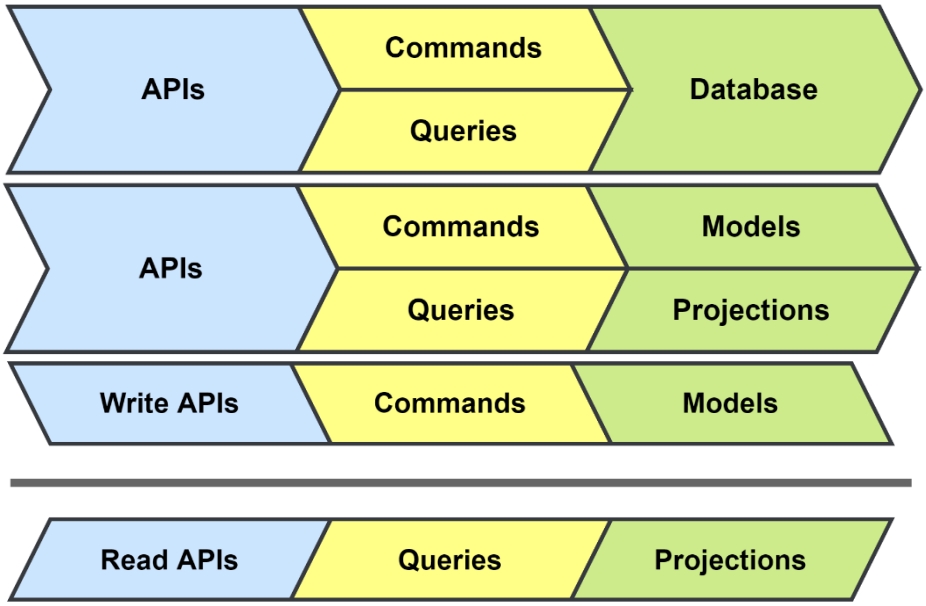


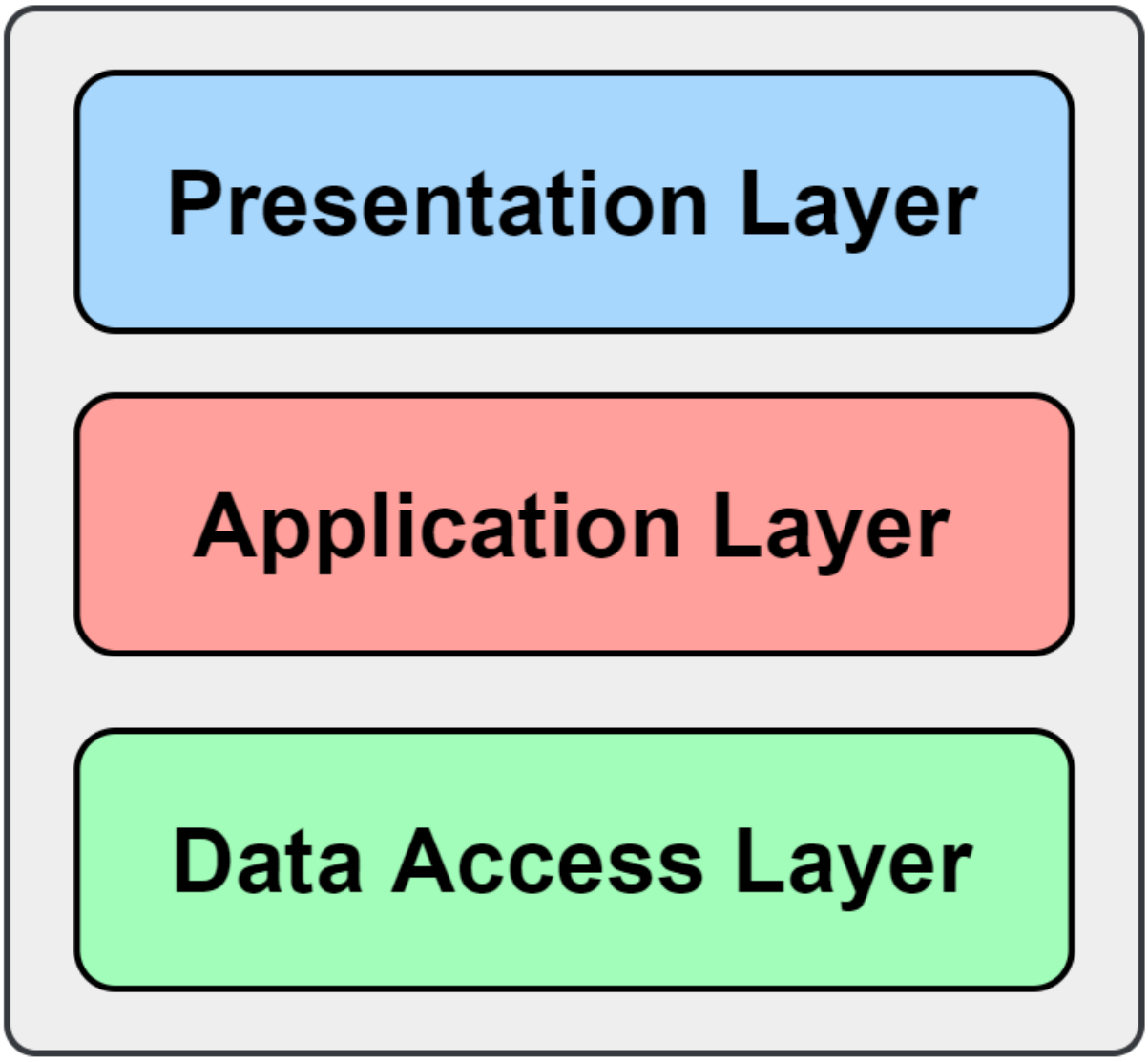

RedisPeopleRepository
client redis.UniversalClient
lock(string) error
unlock(string) error
<<Interface>>
PeopleRepository
Find(context.Context, PersonID) (*Person, error)
Save(context.Context, *Person) error
Update(context.Context, *Person) error
Delete(context.Context, PersonID) error
TestPeopleRepository
items map[PersonID]*Person
reset([]*Person) error

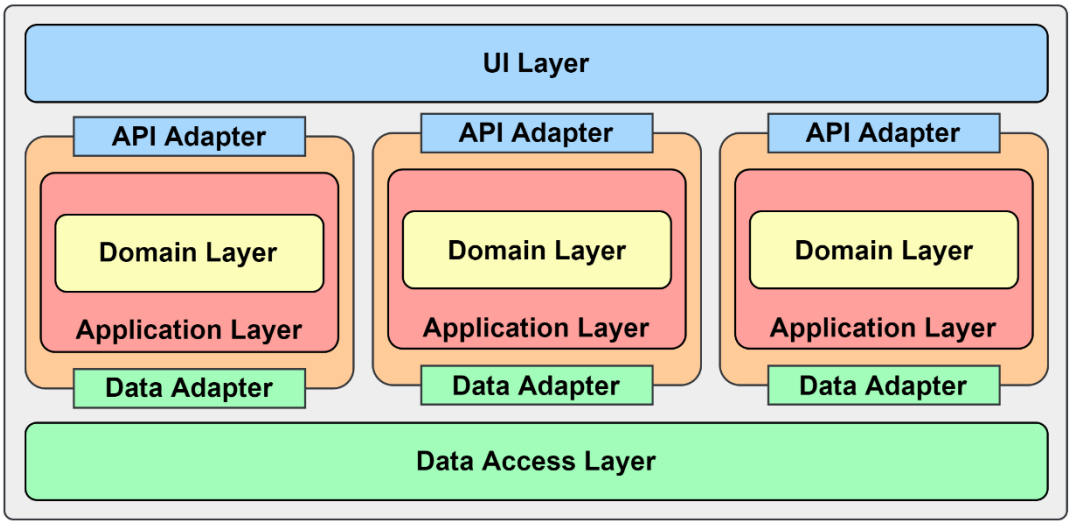

User Interface
Tests
Infrastructure
Application Services
Domain Services
Domain Model
Application Core


REST API
User Interface
Controllers
Use Cases
Entities
Gateways
Repositories
Database
Email

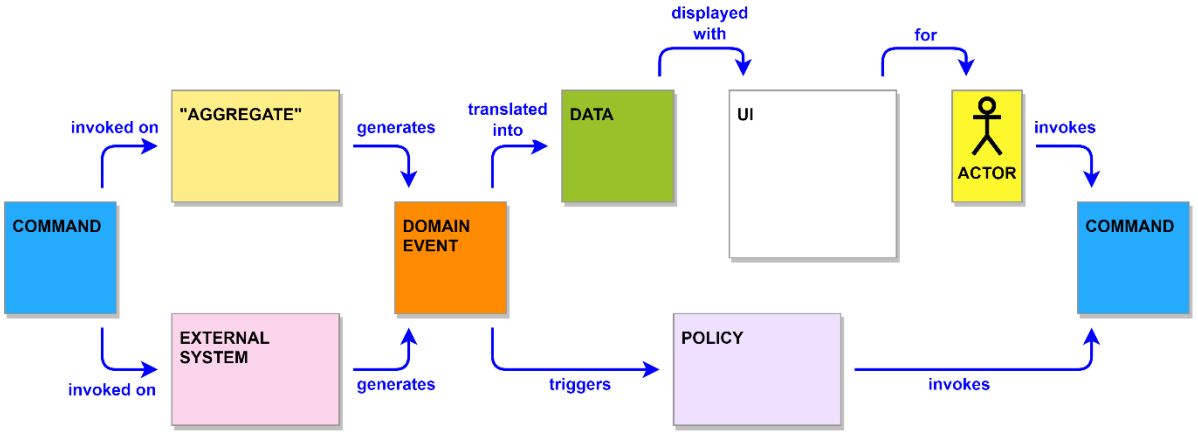

HTTP
Controller
«Interface»
IApplication
Application
«Interface»
IRepository
Model
Repository
Database
Association
Dependency
Implementation
Driver
Driven
Adapter
Port
Adapter
Port
Adapter
Port
Adapter
Port
Domain
Application
Driver
Driven
UI & Infrastructure
Writes
Reads
Writes
Reads
APIs
Application
Database

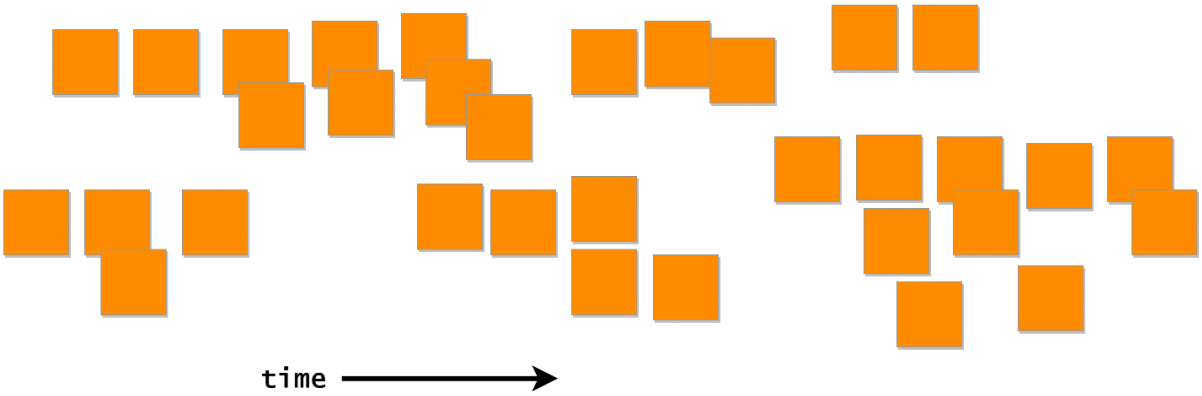

APIs
Commands
Queries
Database
APIs
Commands
Queries
Models
Projections
Write APIs
Commands
Models
Read APIs
Queries
Projections

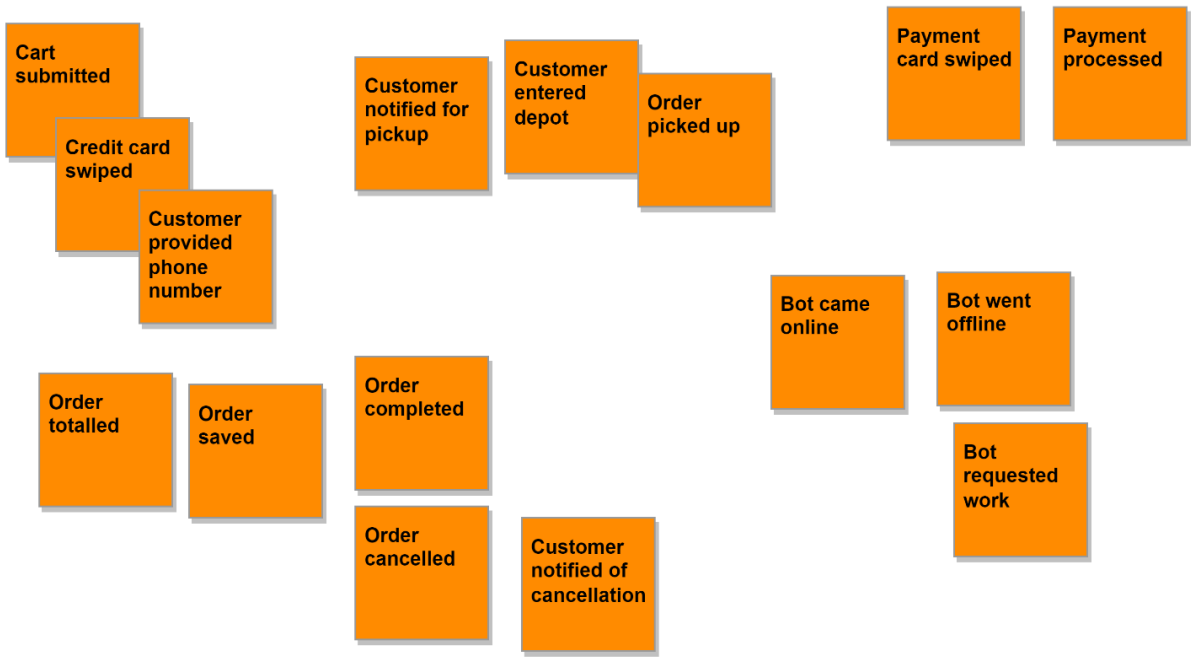

Presentation Layer
Application Layer
Data Access Layer

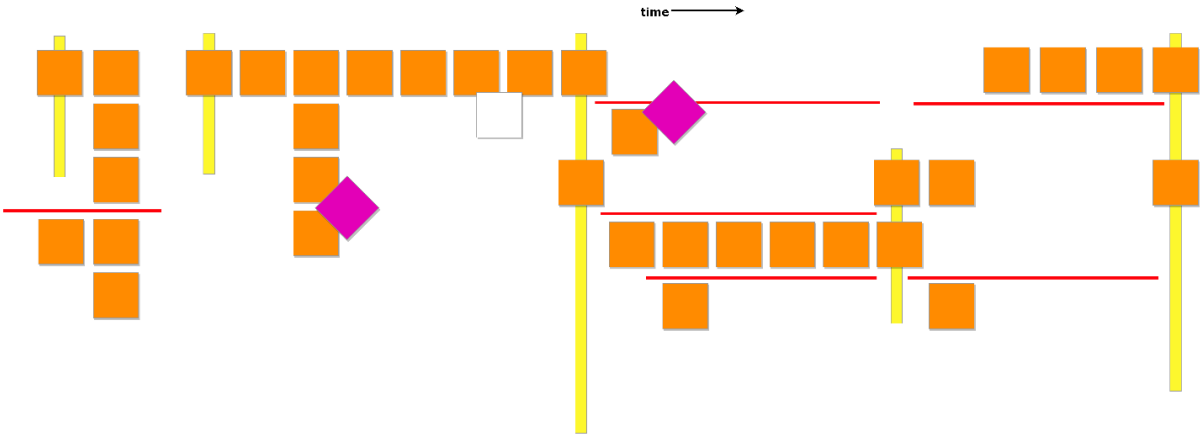

UI Layer
API Adapter
Domain Layer
Application Layer
Data Adapter
API Adapter
Domain Layer
Application Layer
Data Adapter
API Adapter
Domain Layer
Application Layer
Data Adapter
Data Access Layer

Chapter 3: Design and Planning









Customer approached kiosk

Cart started

Item added to the cart

Item removed from the cart

Cart is canceled

Cart is abandoned

Are these different?

Cart totaled

Customer reviewed cart

Credit card swiped

Customer provided phone number

The ordering of these two events does not matter.

Order created

Order totaled

Item selected for pickup

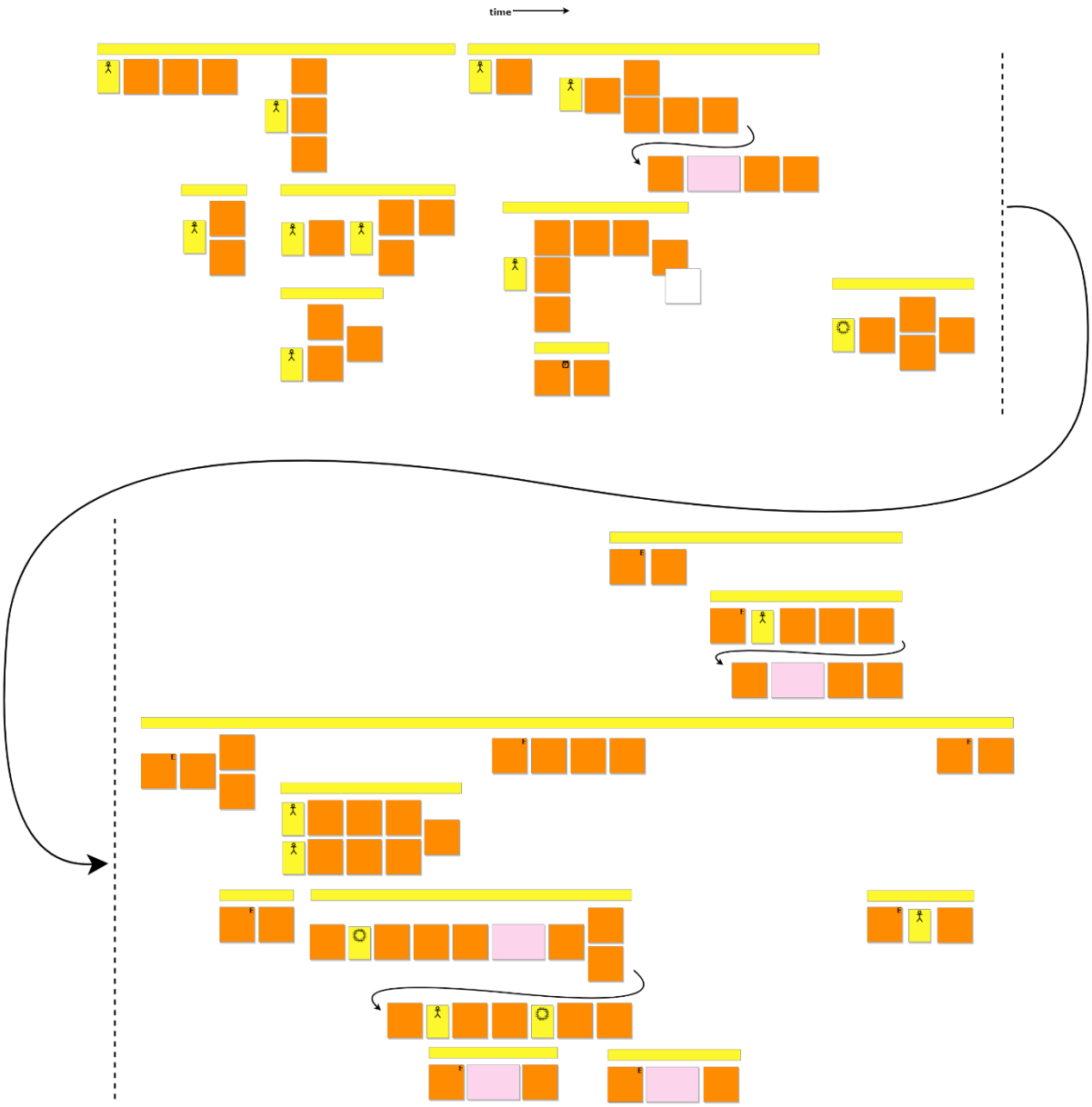
Bot en route to store

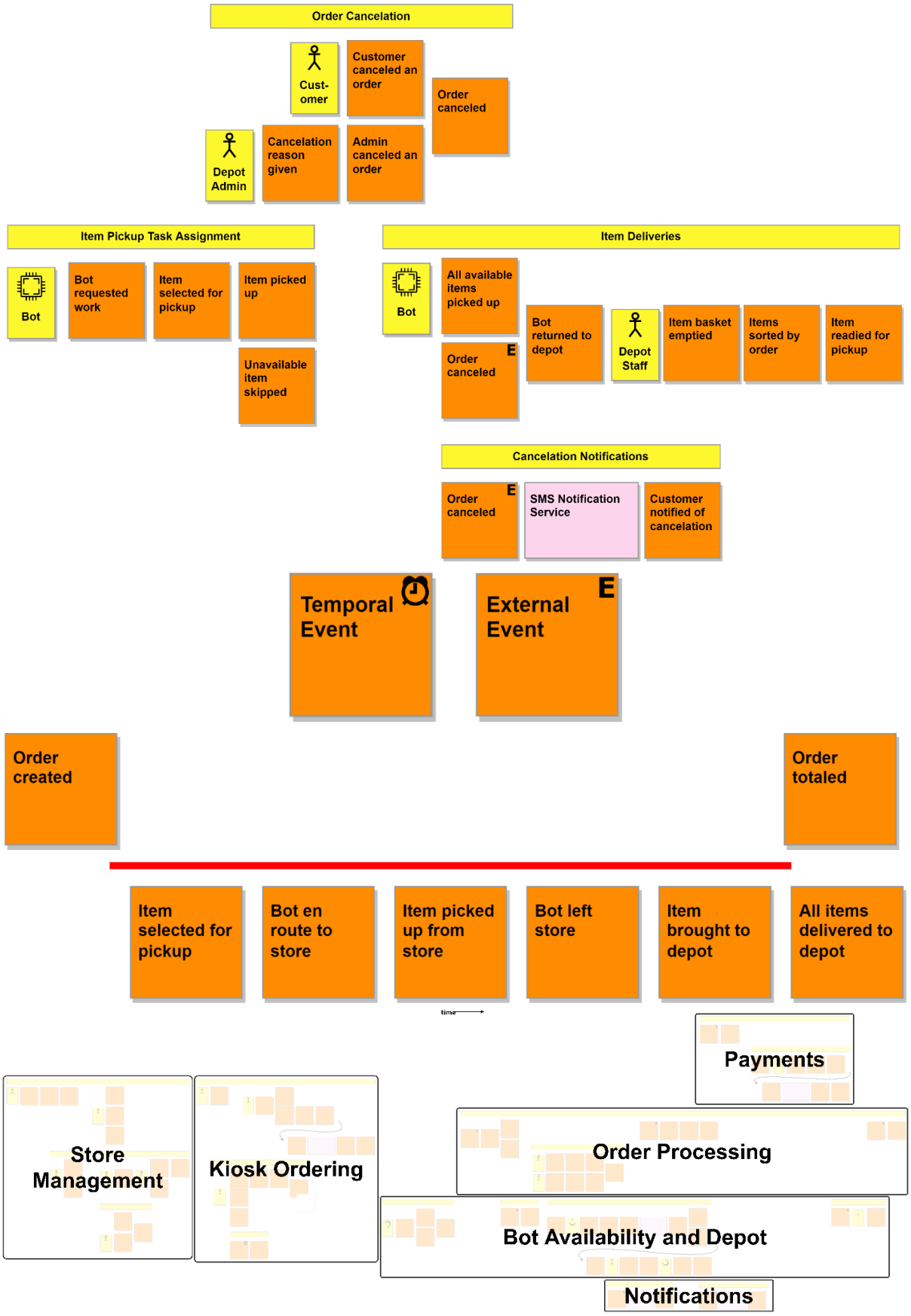
Item picked up from store

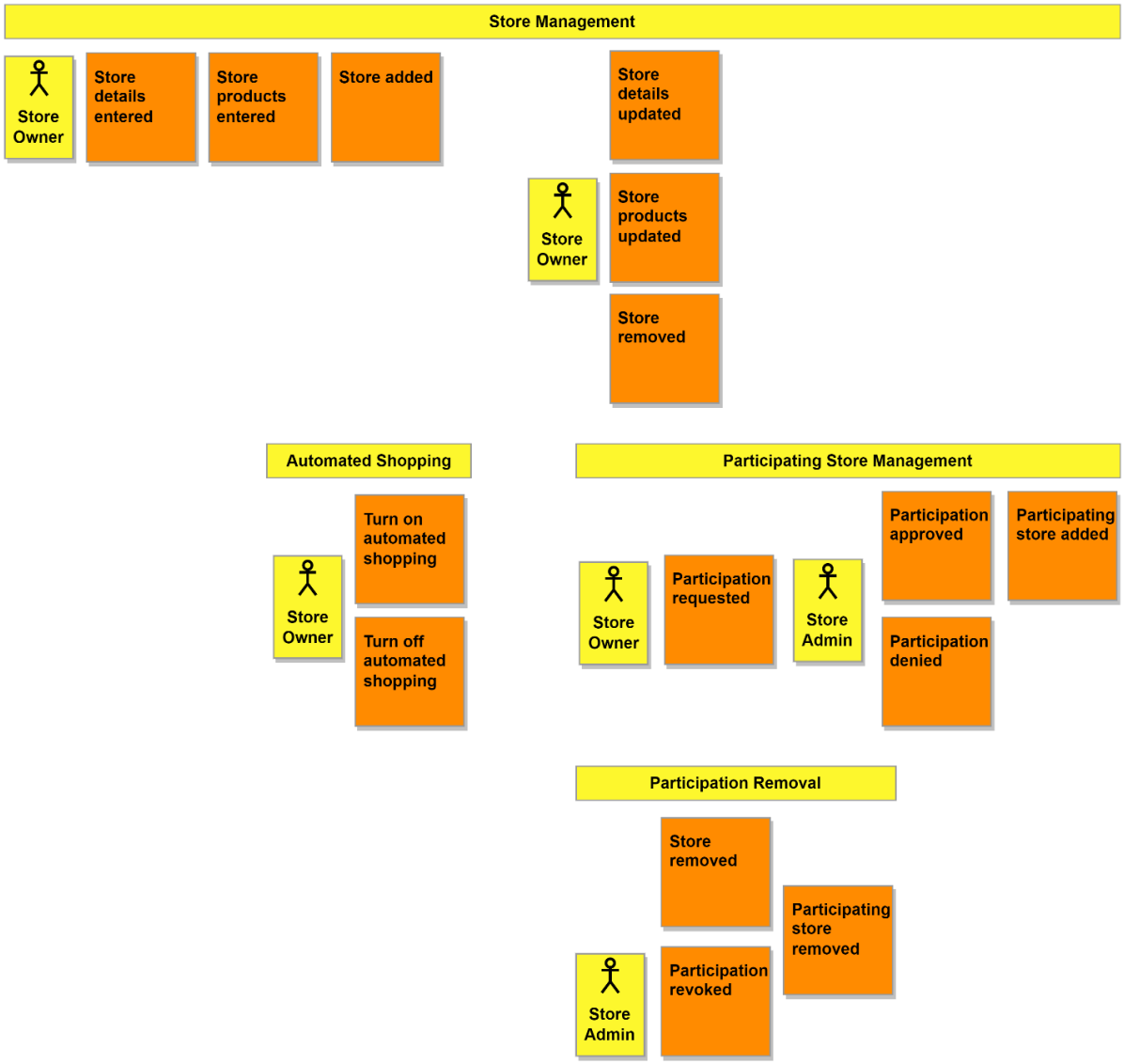
Bot left store

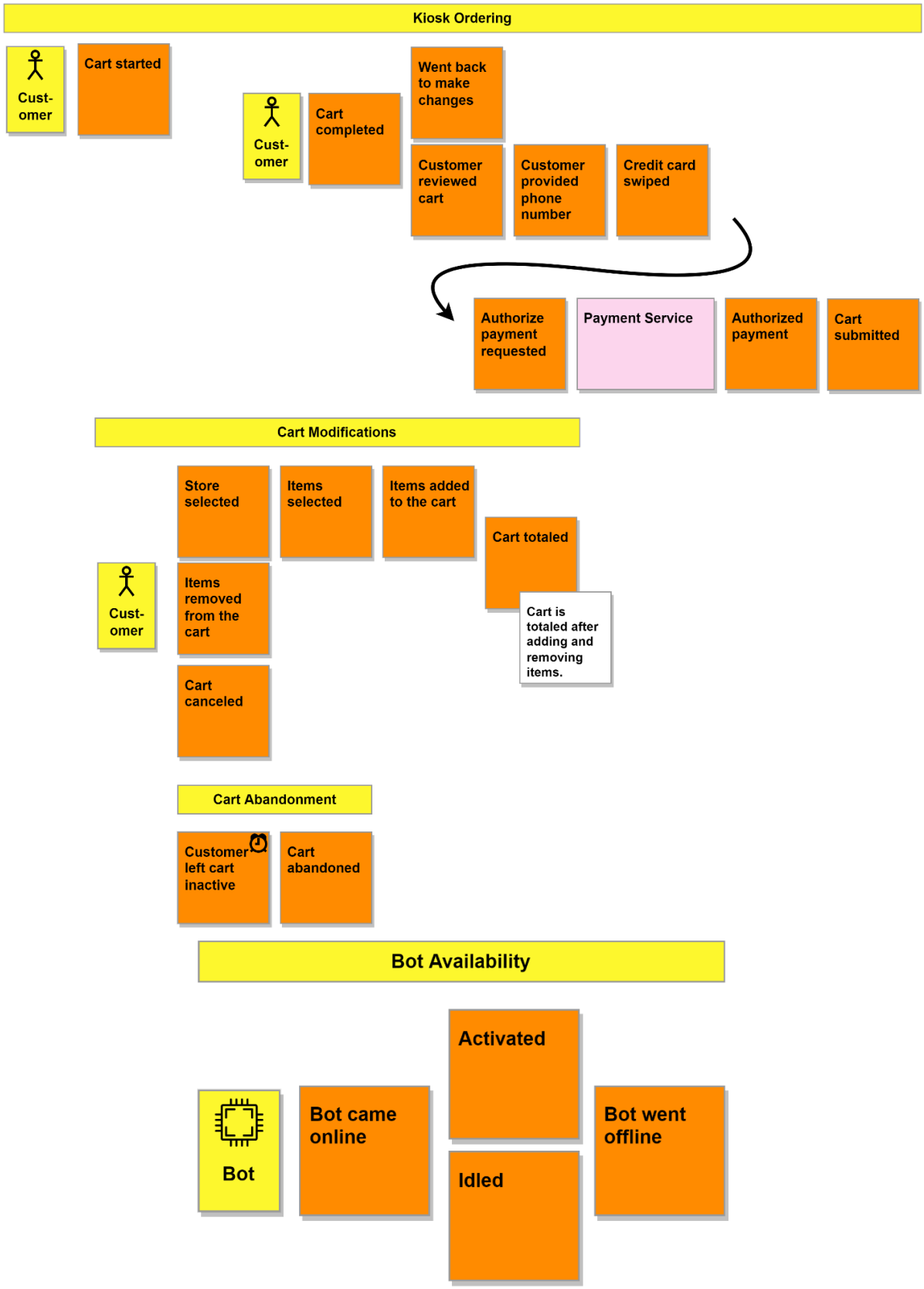
Item brought to depot

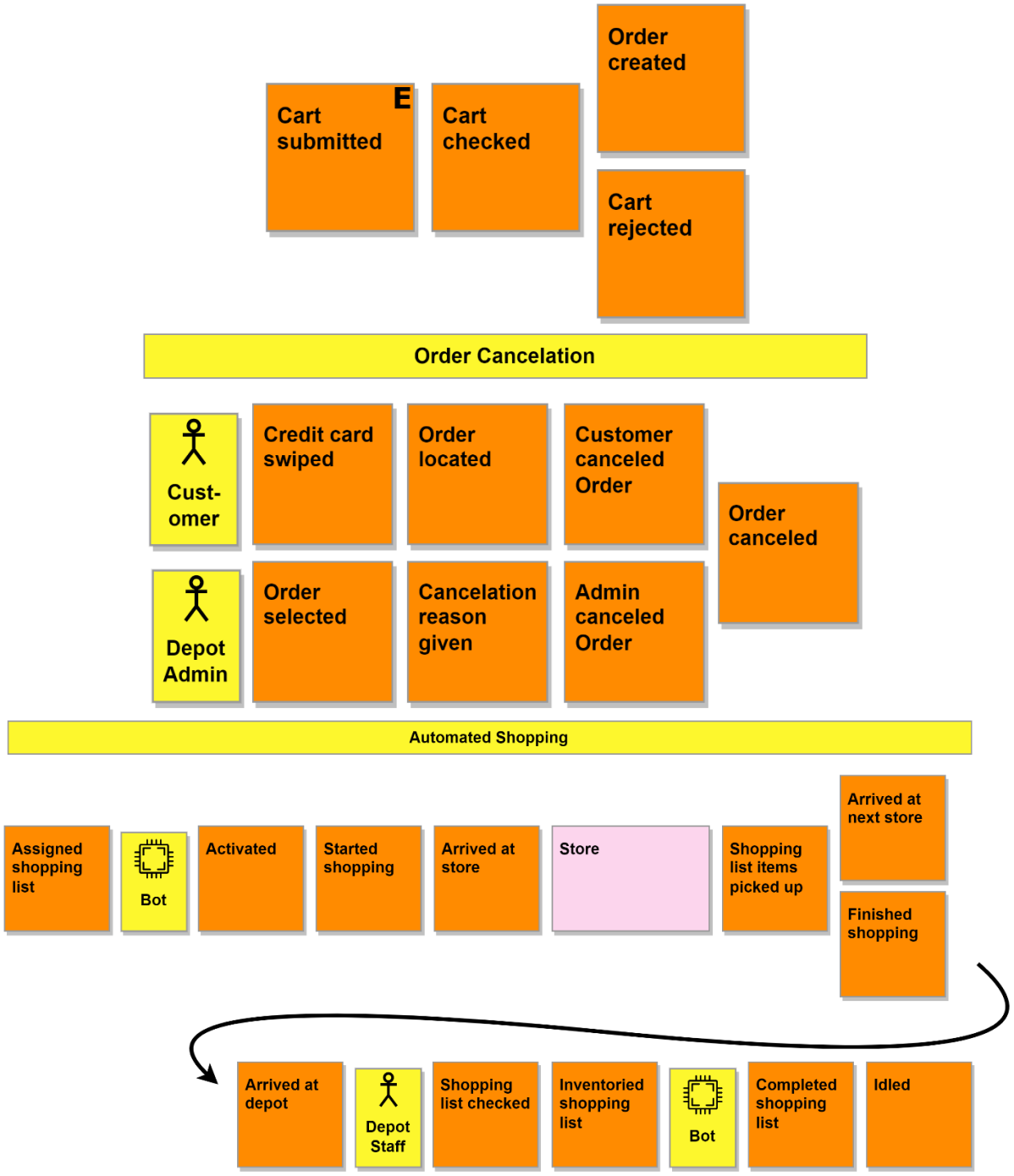
All items delivered to depot

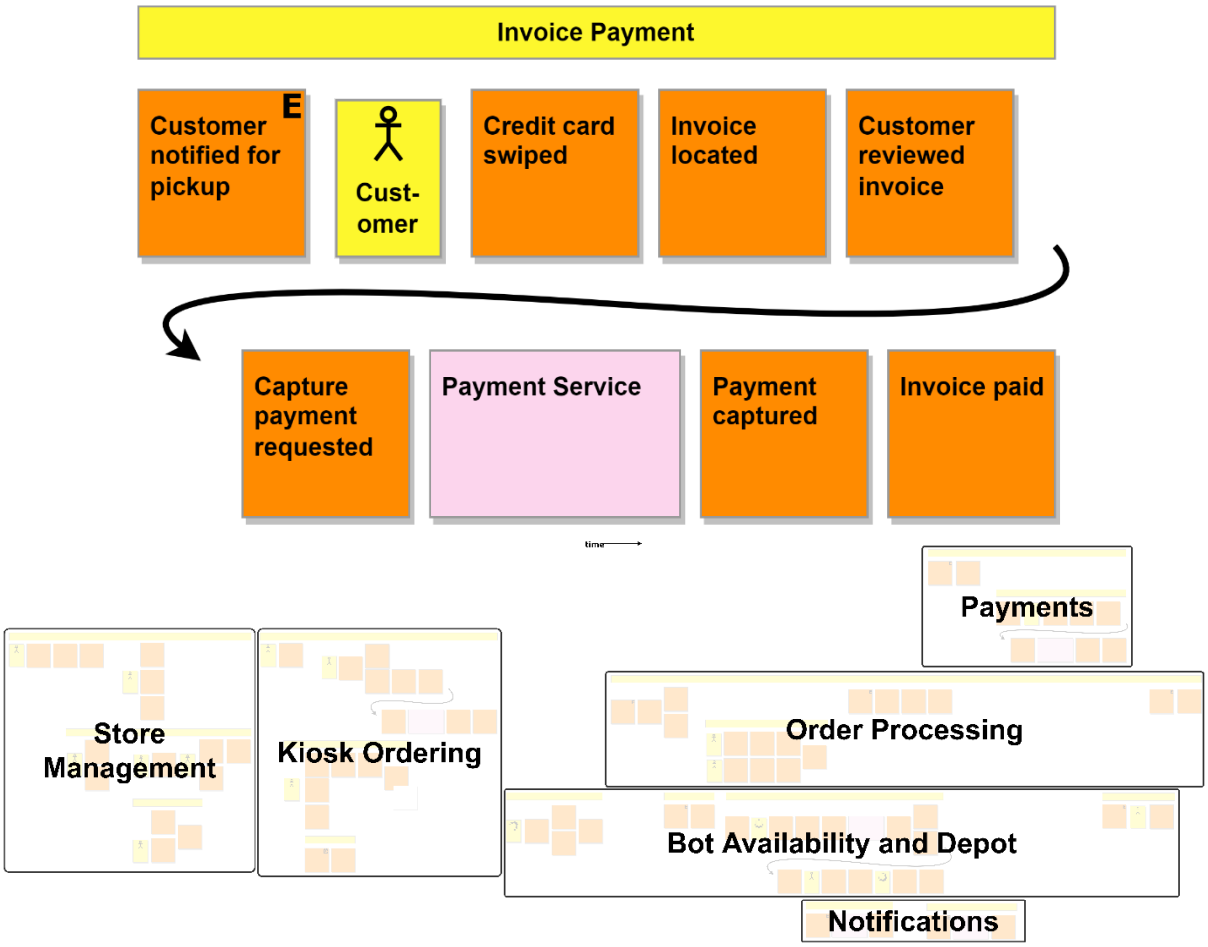

time


Order Cancelation
Cust-omer
Customer canceled an order
Order canceled
Depot Admin
Cancelation reason given
Admin canceled an order
Item Pickup Task Assignment
Bot
Bot requested work
Item selected for pickup
Item picked up
Unavailable item skipped
Item Deliveries
Bot
All available items picked up
Order canceled
E
Bot returned to depot
Depot Staff
Item basket emptied
Items sorted by order
Item readied for pickup
Cancelation Notifications
Order canceled
E
SMS Notification Service
Customer notified of cancelation
Temporal Event
External Event
E
Order created
Order totaled
Item selected for pickup
Bot en route to store
Item picked up from store
Bot left store
Item brought to depot
All items delivered to depot
time
Payments
Store Management
Kiosk Ordering
Order Processing
Bot Availability and Depot
Notifications

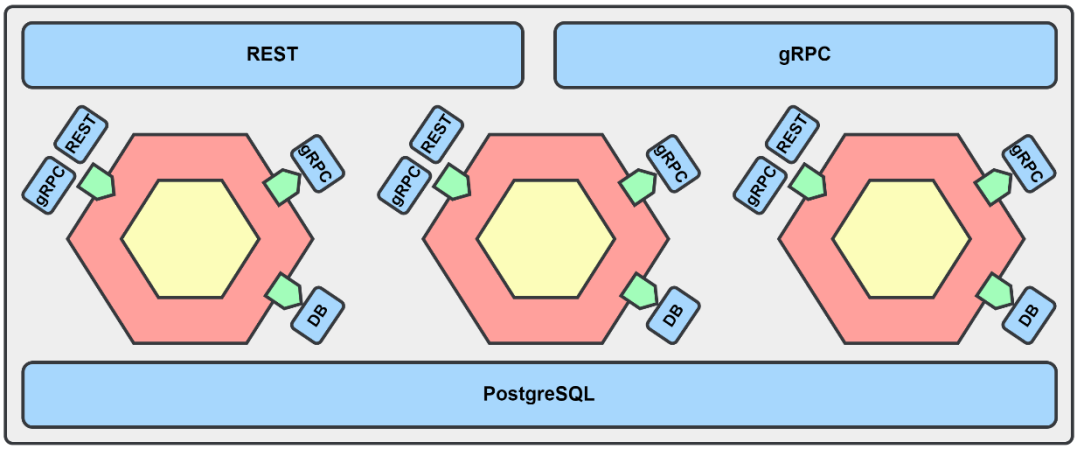

Store Management
Store Owner
Store details entered
Store products entered
Store added
Store details updated
Store Owner
Store products updated
Store removed
Automated Shopping
Store Owner
Turn on automated shopping
Turn off automated shopping
Participating Store Management
Store Owner
Participation requested
Store Admin
Participation approved
Participating store added
Participation denied
Participation Removal
Store removed
Participating store removed
Store Admin
Participation revoked

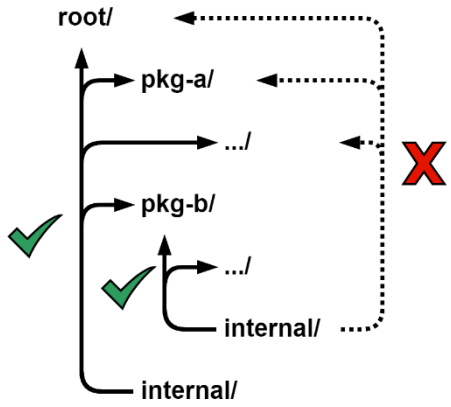

Kiosk Ordering
Cust-omer
Cart started
Cust-omer
Cart completed
Went back to make changes
Customer reviewed cart
Customer provided phone number
Credit card swiped
Authorize payment requested
Payment Service
Authorized payment
Cart submitted
Cart Modifications
Store selected
Items selected
Items added to the cart
Cart totaled
Cust-omer
Items removed from the cart
Cart canceled
Cart is totaled after adding and removing items.
Cart Abandonment
Customer left cart inactive
Cart abandoned
Bot Availability
Bot
Bot came online
Activated
Idled
Bot went offline

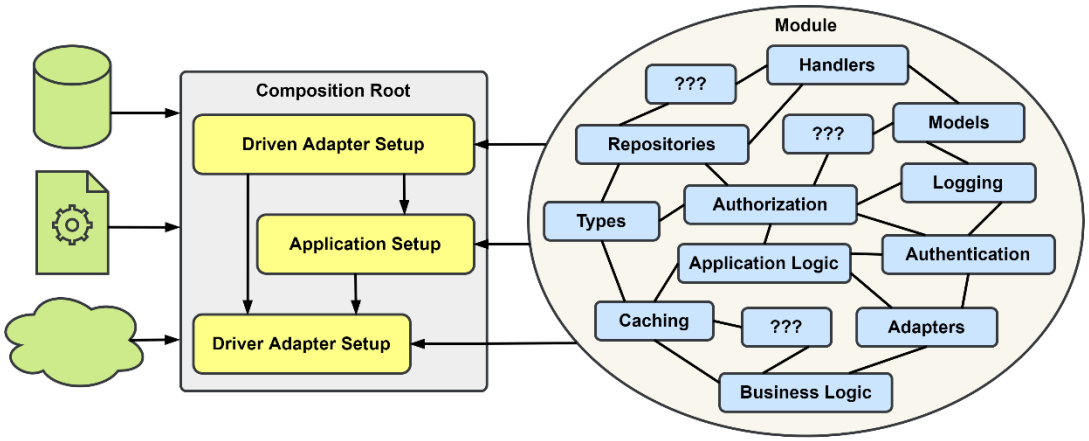

Cart submitted
E
Cart checked
Order created
Cart rejected
Order Cancelation
Cust-omer
Credit card swiped
Order located
Customer canceled Order
Order canceled
Depot Admin
Order selected
Cancelation reason given
Admin canceled Order
Automated Shopping
Assigned shopping list
Bot
Activated
Started shopping
Arrived at store
Store
Shopping list items picked up
Arrived at next store
Finished shopping
Arrived at depot
Depot Staff
Shopping list checked
Inventoried shopping list
Bot
Completed shopping list
Idled

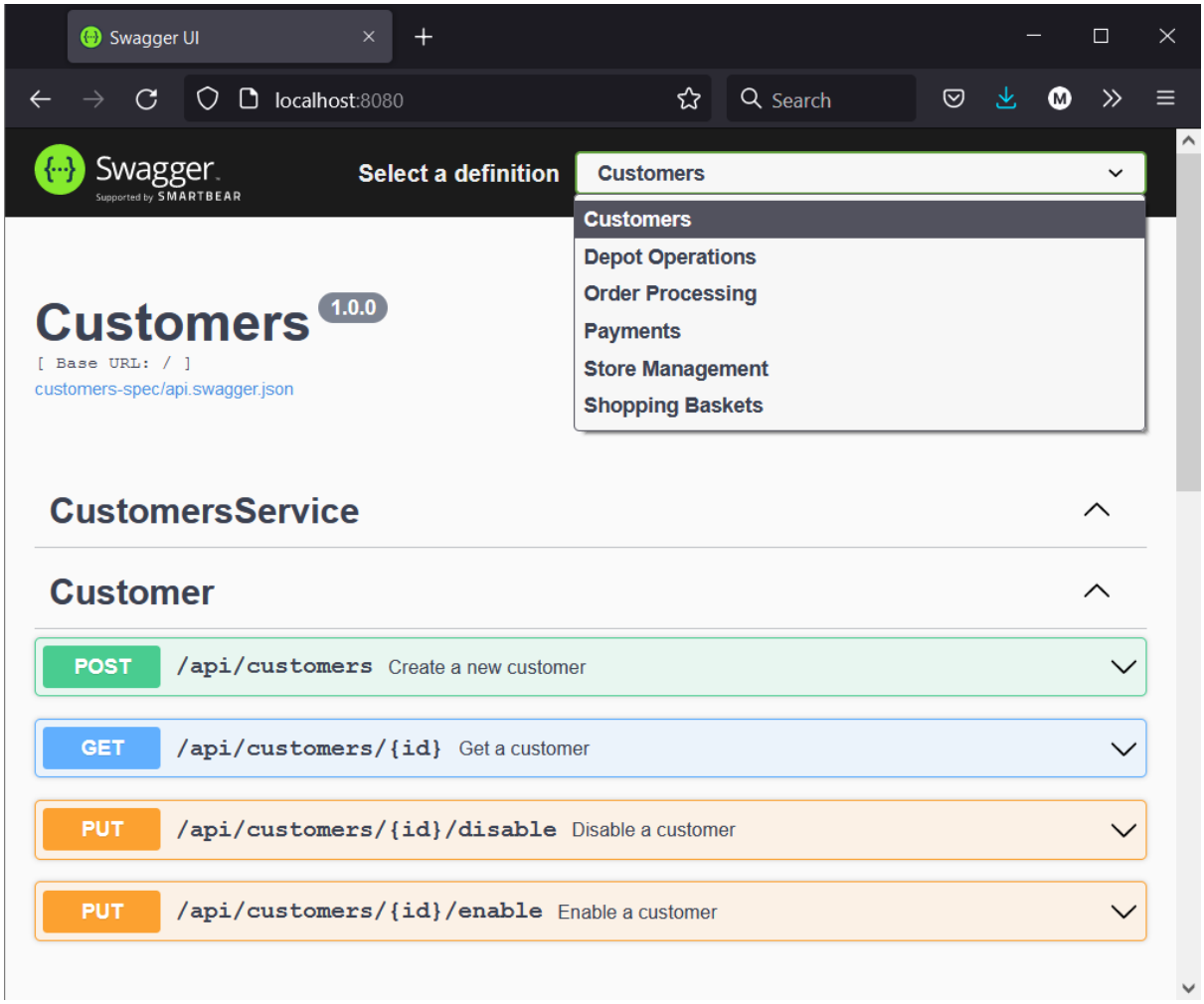

Invoice Payment
Customer notified for pickup
E
Cust-omer
Credit card swiped
Invoice located
Customer reviewed invoice
Capture payment requested
Payment Service
Payment captured
Invoice paid
time
Payments
Store Management
Kiosk Ordering
Order Processing
Bot Availability and Depot
Notifications

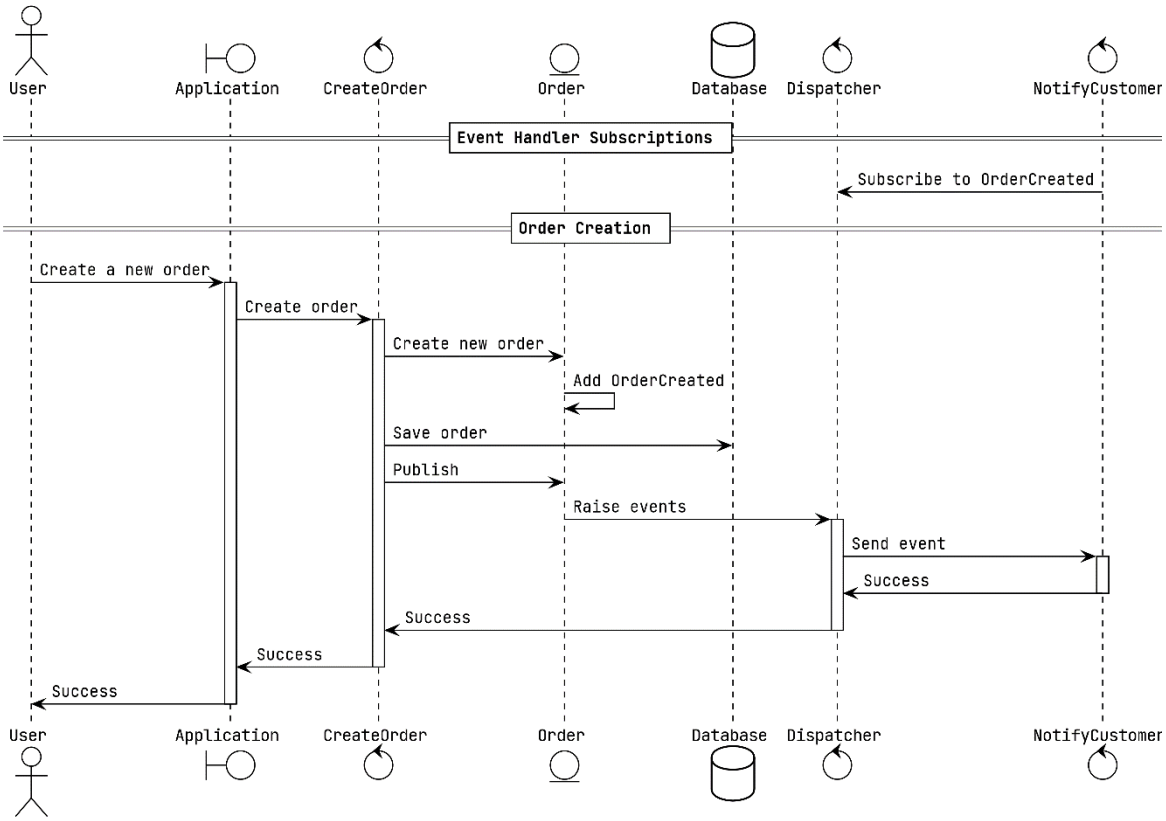
Chapter 4: Event Foundations

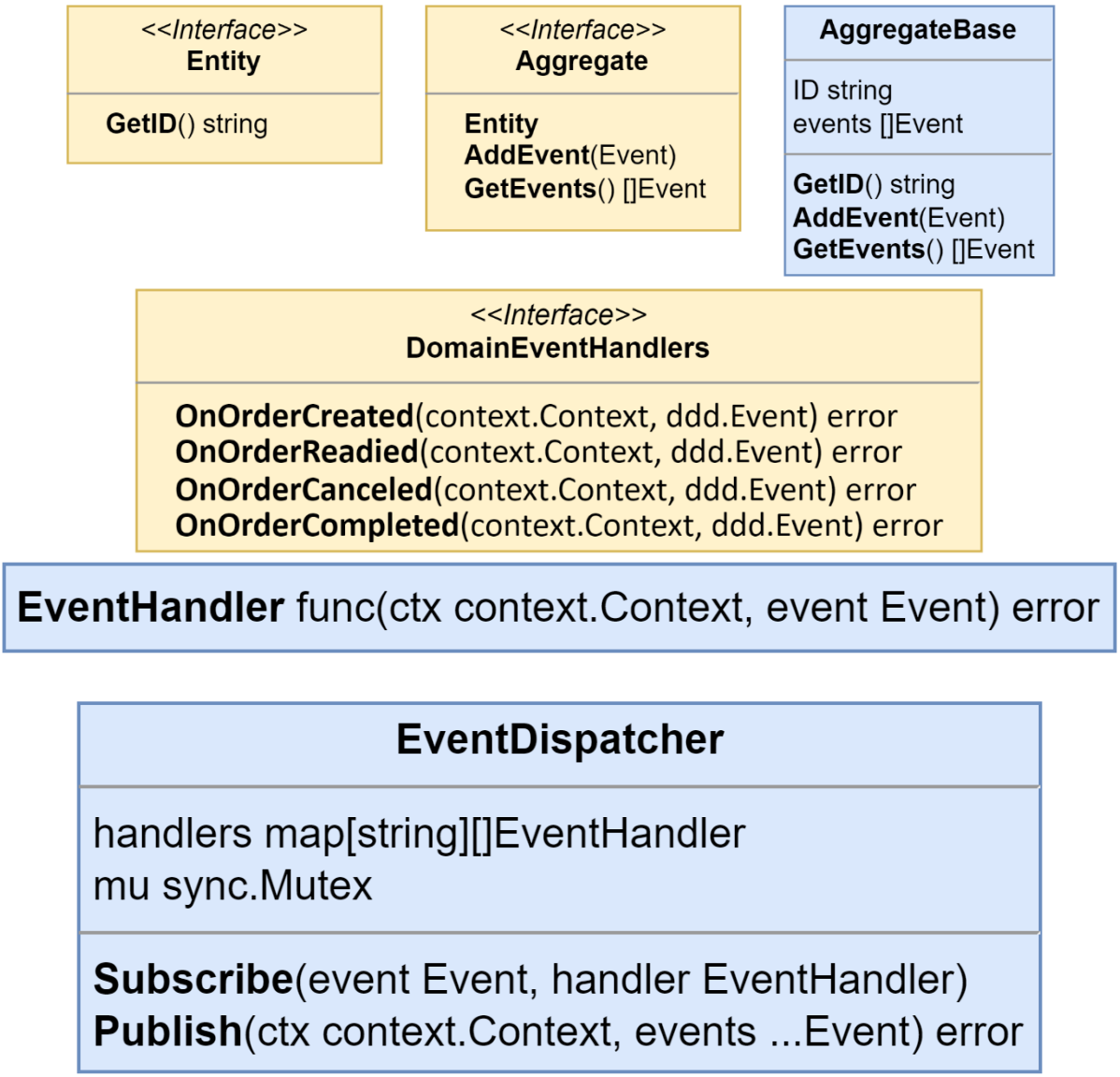







Swagger UI
localhost:8080
Search
Swagger
Supported by SMARTBEAR
Select a definition
Customers
Customers
Depot Operations
Order Processing
Payments
Store Management
Shopping Baskets
Customers
1.0.0
[Base URL: /]
customers-spec/api.swagger.json
CustomersService
Customer
POST /api/customers Create a new customer
GET /api/customers/{id} Get a customer
PUT /api/customers/{id}/disable Disable a customer
PUT /api/customers/{id}/enable Enable a customer

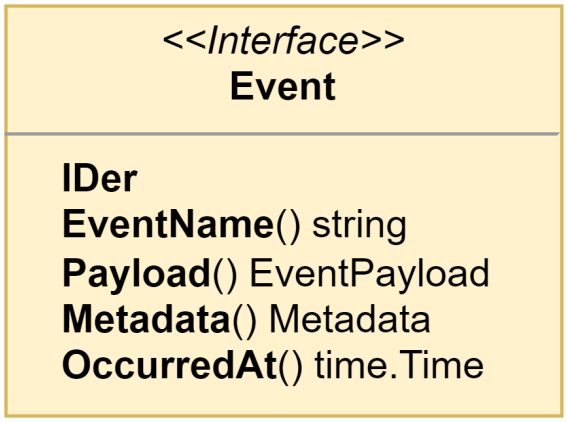

User
Application
CreateOrder
Order
Database
Dispatcher
NotifyCustomer
Event Handler Subscriptions
Subscribe to OrderCreated
Order Creation
Create a new order
Create order
Create new order
Add OrderCreated
Save order
Publish
Raise events
Send event
Success
Success
Success
Success
User
Application
CreateOrder
Order
Database
Dispatcher
NotifyCustomer

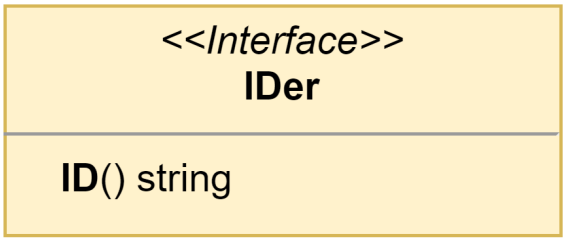

<<Interface>>
Entity
GetID() string
<<Interface>>
Aggregate
Entity
AddEvent(Event)
GetEvents() []Event
AggregateBase
ID string
events []Event
GetID() string
AddEvent(Event)
GetEvents() []Event
<<Interface>>
DomainEventHandlers
OnOrderCreated(context.Context, ddd.Event) error
OnOrderReadied(context.Context, ddd.Event) error
OnOrderCanceled(context.Context, ddd.Event) error
OnOrderCompleted(context.Context, ddd.Event) error
EventHandler func(ctx context.Context, event Event) error
EventDispatcher
handlers map[string][]EventHandler
mu sync.Mutex
Subscribe(event Event, handler EventHandler)
Publish(ctx context.Context, events ...Event) error

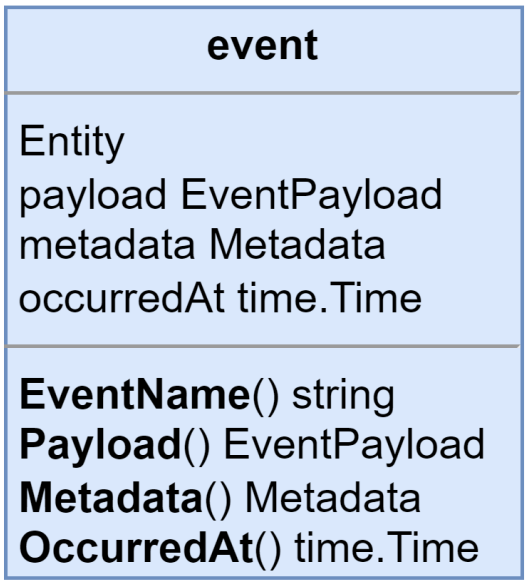
Chapter 5: Tracking Changes with Event Sourcing

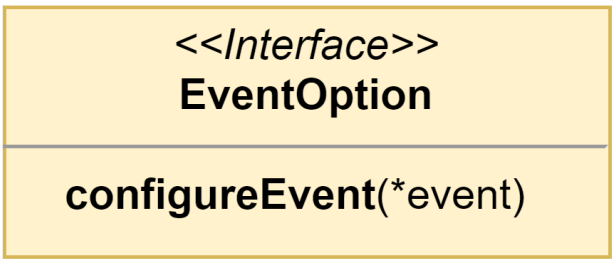
Products					
ID	**StoreID**	**Name**	**SKU**	**Price**	**...**
1	1	Diagrams for Dum...	BS-DD-123	***24.99***	**...**
2	1	5001 Great Exam...	BS-GE-456	34.99	**...**
3	2	Wizard w/ Crystal	wizard	28.99	**...**
...	...	...	...	...	...

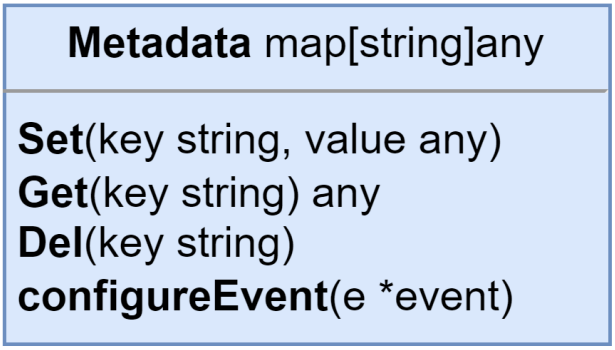
Events					
ID	**Type**	**Version**	**EventID**	**EventType**	**...**
1	Product	1	101	ProductAdded	**...**
1	Product	2	102	ProductRebranded	**...**
1	***Product***	***3***	***103***	***ProductPriceIncreased***	***...***
...	...	...	...	...	...

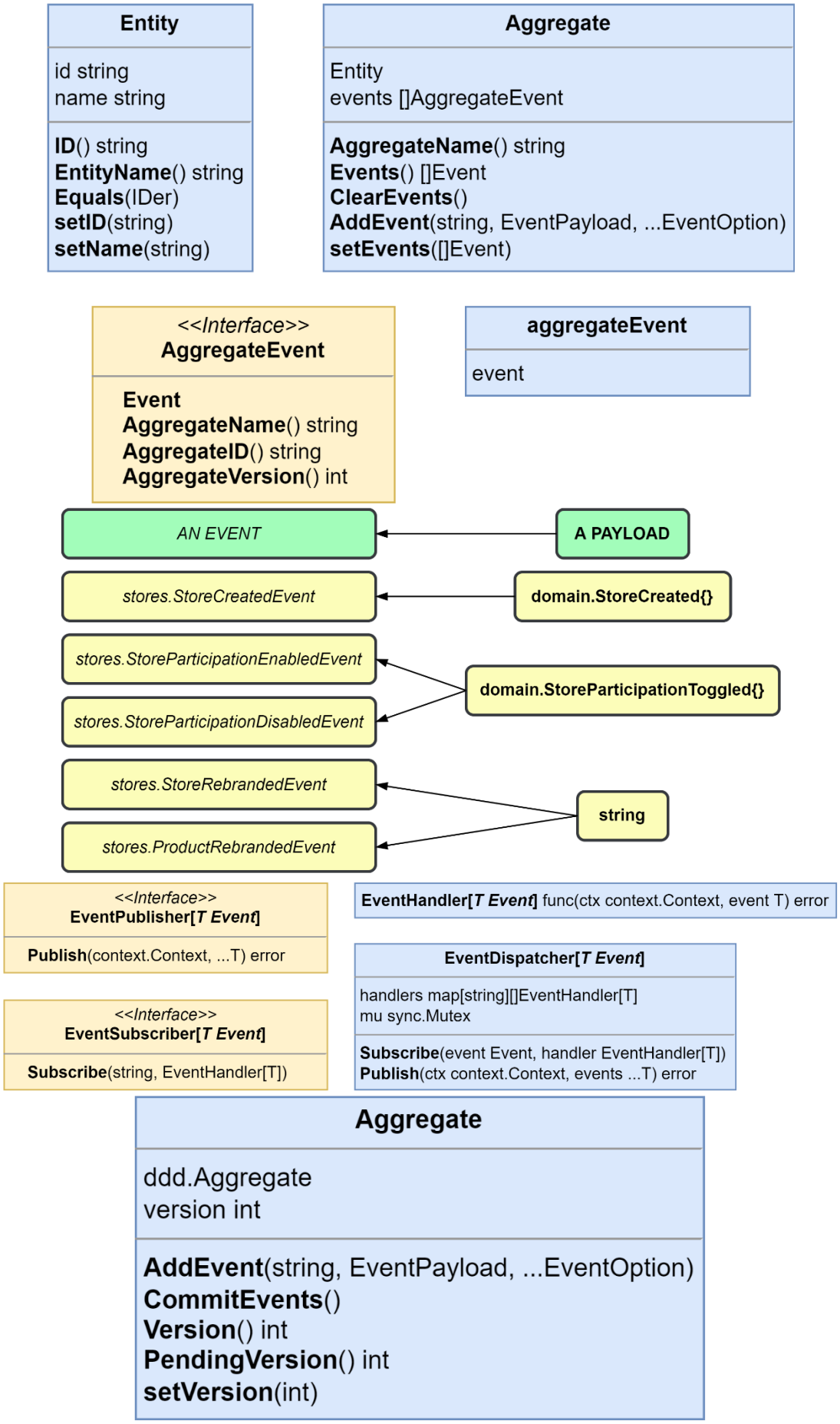


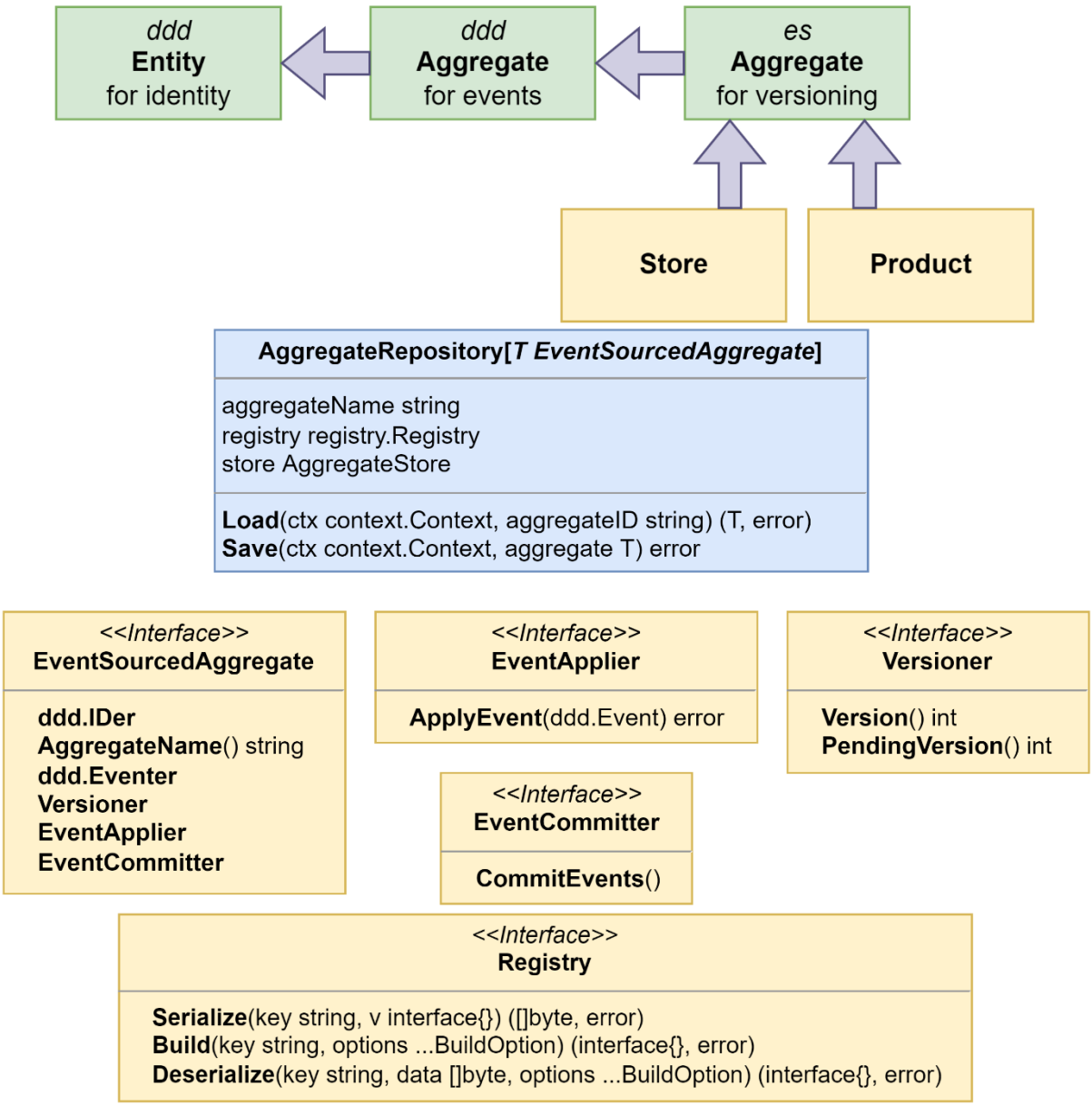


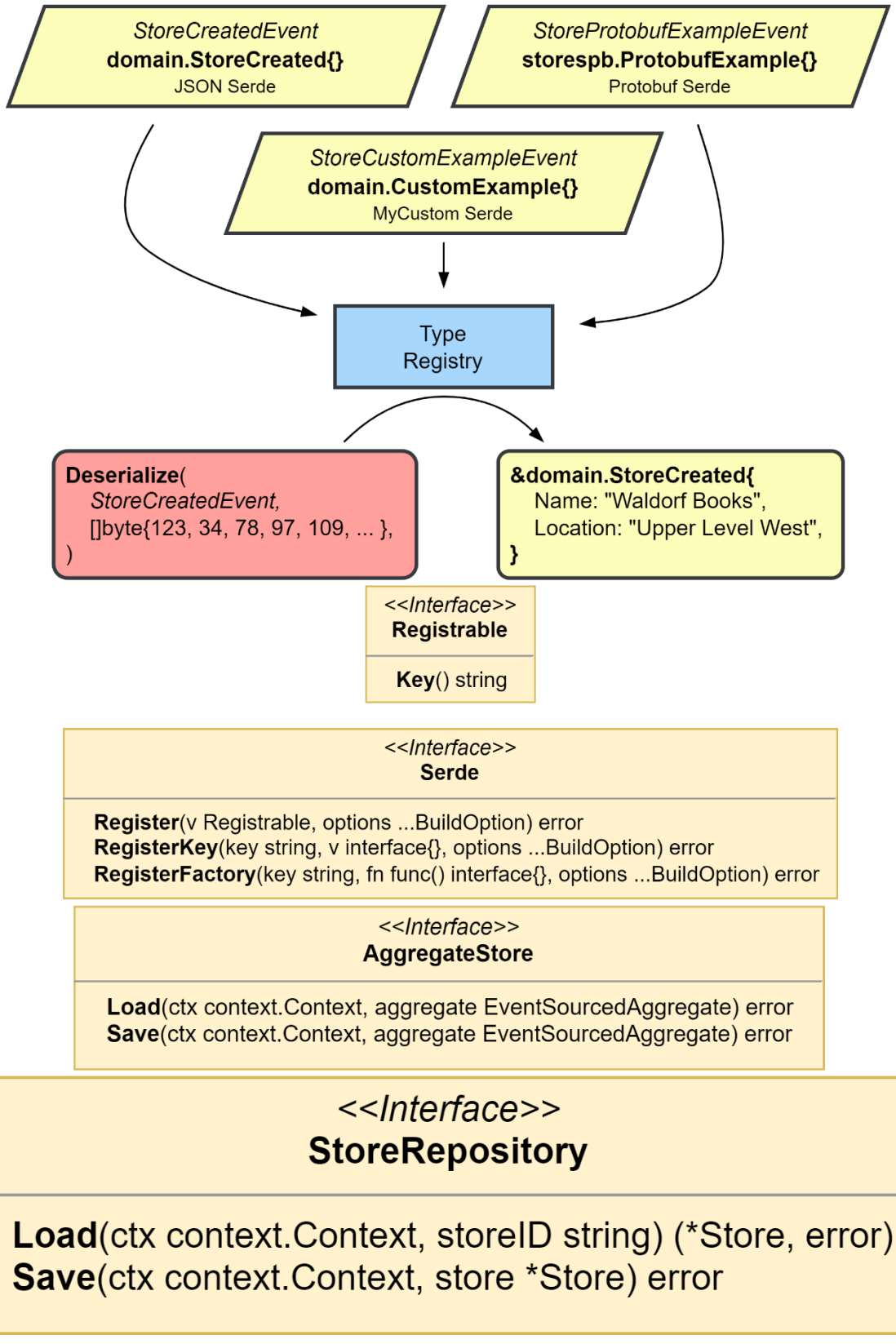


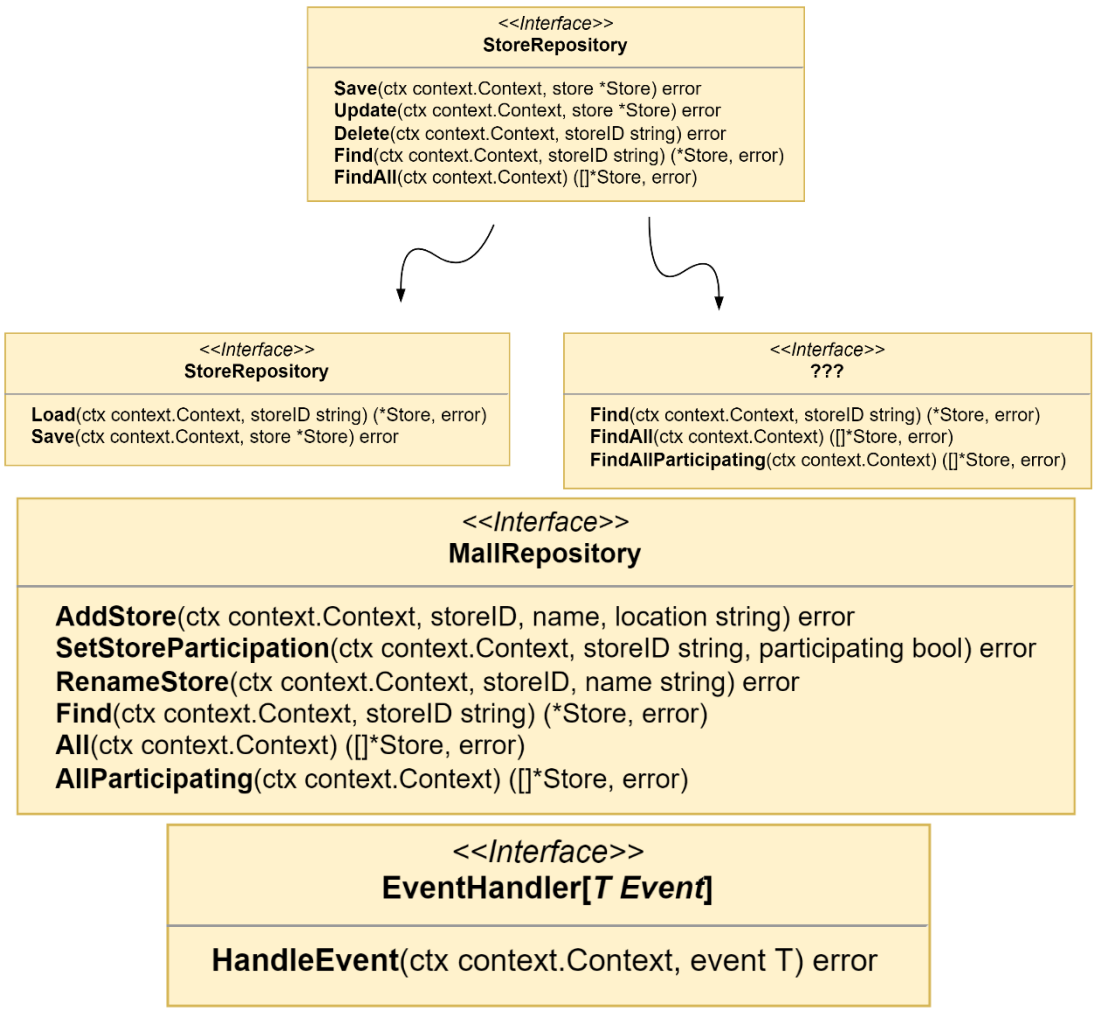


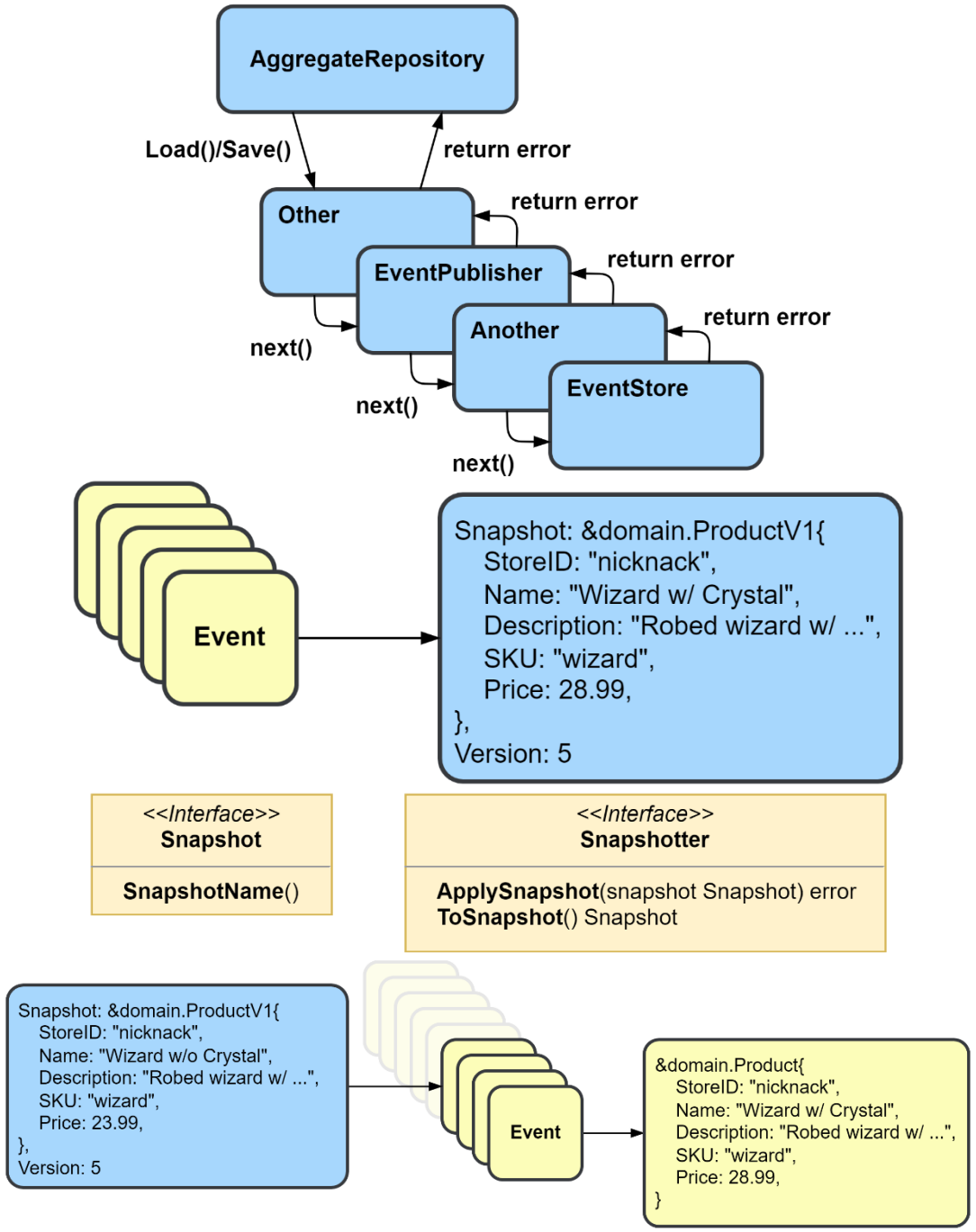



Entity
id string
name string
ID() string
EntityName() string
Equals(IDer)
setID(string)
setName(string)
Aggregate
Entity
events []AggregateEvent
AggregateName() string
Events() []Event
ClearEvents()
AddEvent(string, EventPayload, ...EventOption)
setEvents([]Event)
<<Interface>>
AggregateEvent
Event
AggregateName() string
AggregateID() string
AggregateVersion() int
aggregateEvent
event
AN EVENT
A PAYLOAD
stores.StoreCreatedEvent
domain.StoreCreated{}
stores.StoreParticipationEnabledEvent
stores.StoreParticipationDisabledEvent
domain.StoreParticipationToggled{}
stores.StoreRebrandedEvent
stores.ProductRebrandedEvent
string
<<Interface>>
EventPublisher[T Event]
Publish(context.Context, ...T) error
EventHandler[T Event] func(ctx context.Context, event T) error
<<Interface>>
EventSubscriber[T Event]
Subscribe(string, EventHandler[T])
EventDispatcher[T Event]
handlers map[string][]EventHandler[T]
mu sync.Mutex
Subscribe(event Event, handler EventHandler[T])
Publish(ctx context.Context, events ...T) error
Aggregate
ddd.Aggregate
version int
AddEvent(string, EventPayload, ...EventOption)
CommitEvents()
Version() int
PendingVersion() int
setVersion(int)

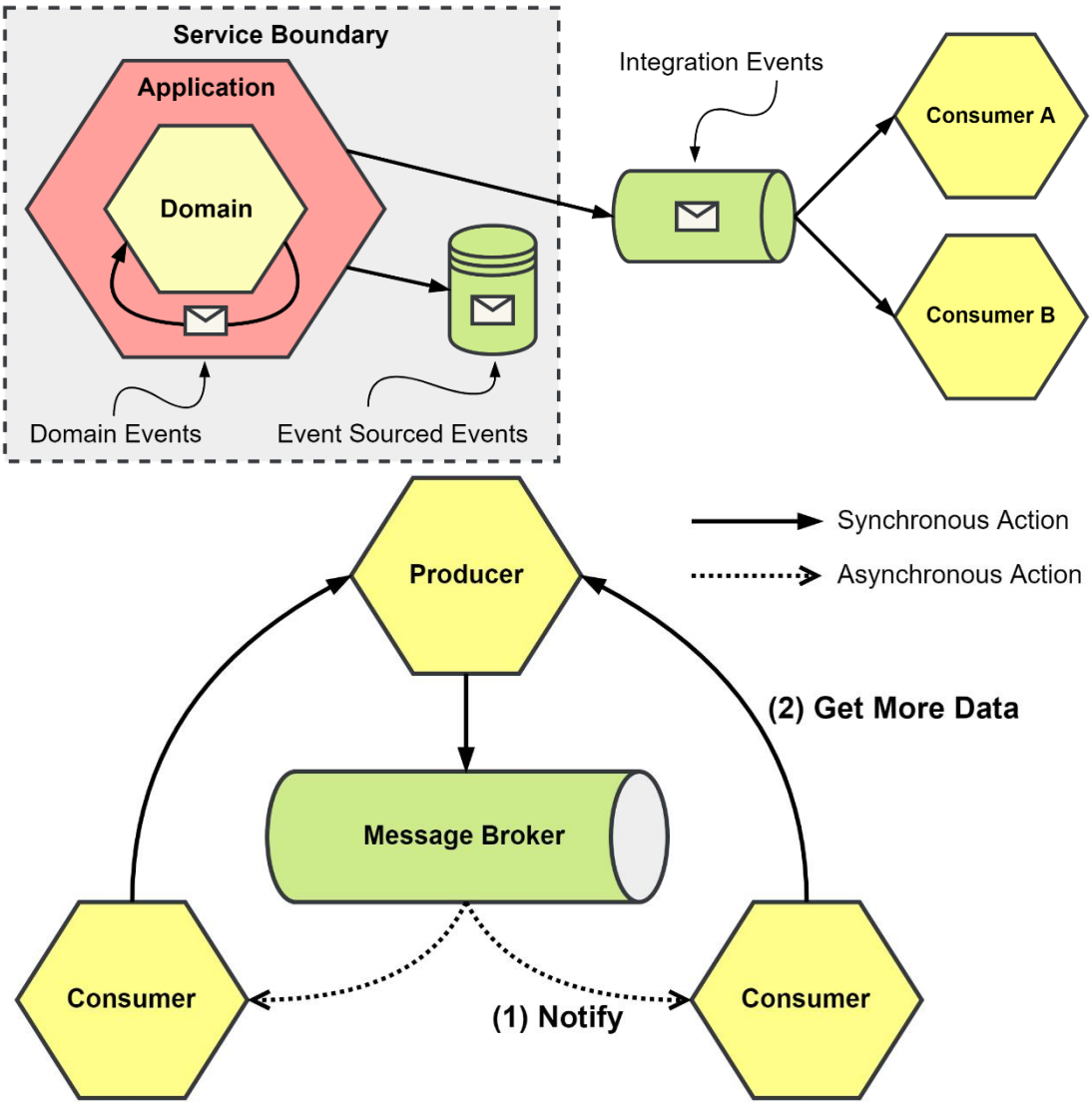

ddd
Entity
for identity
ddd
Aggregate
for events
es
Aggregate
for versioning
Store
Product
AggregateRepository[T EventSourcedAggregate]
aggregateName string
registry registry.Registry
store AggregateStore
Load(ctx context.Context, aggregateID string) (T, error)
Save(ctx context.Context, aggregate T) error
<<Interface>>
EventSourcedAggregate
ddd.IDer
AggregateName() string
ddd.Eventer
Versioner
EventApplier
EventCommitter
<<Interface>>
EventApplier
ApplyEvent(ddd.Event) error
<<Interface>>
Versioner
Version() int
PendingVersion() int
<<Interface>>
EventCommitter
CommitEvents()
<<Interface>>
Registry
Serialize(key string, v interface{}) ([]byte, error)
Build(key string, options ...BuildOption) (interface{}, error)
Deserialize(key string, data []byte, options ...BuildOption) (interface{}, error)

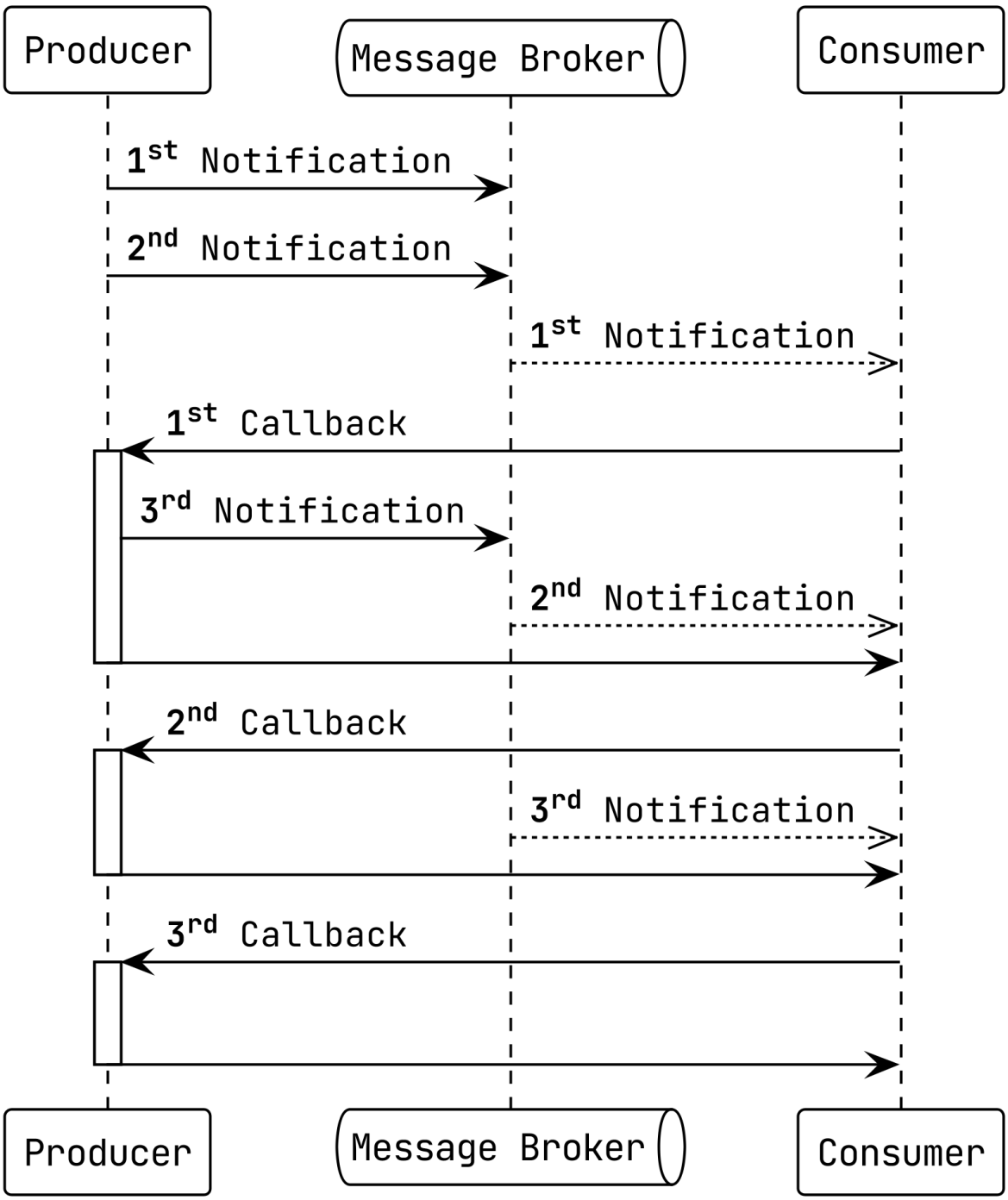

StoreCreatedEvent
domain.StoreCreated{}
JSON Serde
StoreProtobufExampleEvent
storespb.ProtobufExample{}
Protobuf Serde
StoreCustomExampleEvent
domain.CustomExample{}
MyCustom Serde
Type
Registry
Deserialize(
StoreCreatedEvent,
[]byte{123, 34, 78, 97, 109, ... },
)
&domain.StoreCreated{
Name: "Waldorf Books",
Location: "Upper Level West",
}
<<Interface>>
Registrable
Key() string
<<Interface>>
Serde
Register(v Registrable, options ...BuildOption) error
RegisterKey(key string, v interface{}, options ...BuildOption) error
RegisterFactory(key string, fn func() interface{}, options ...BuildOption) error
<<Interface>>
AggregateStore
Load(ctx context.Context, aggregate EventSourcedAggregate) error
Save(ctx context.Context, aggregate EventSourcedAggregate) error
<<Interface>>
StoreRepository
Load(ctx context.Context, storeID string) (*Store, error)
Save(ctx context.Context, store *Store) error

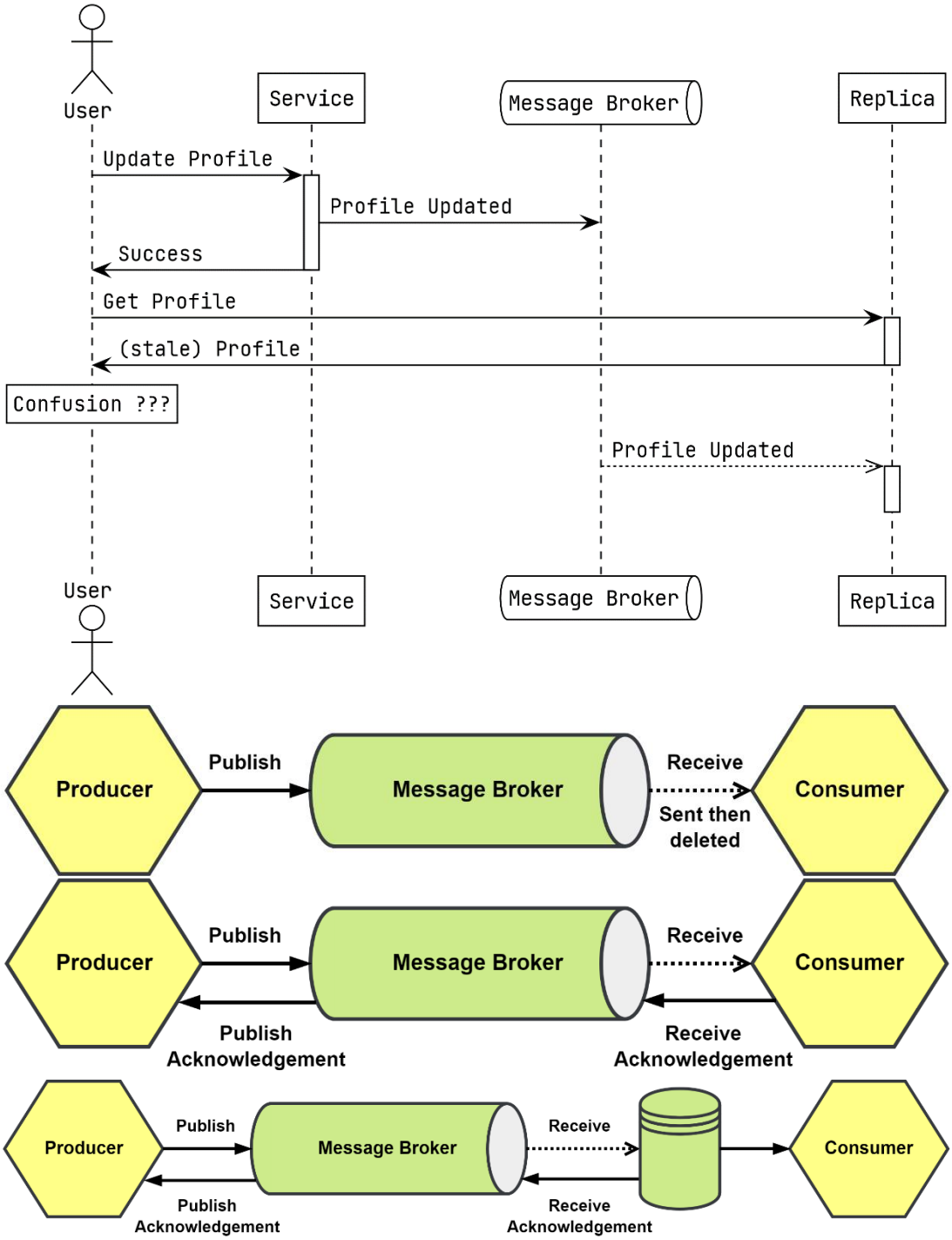

<<Interface>>
StoreRepository
Save(ctx context.Context, store *Store) error
Update(ctx context.Context, store *Store) error
Delete(ctx context.Context, storeID string) error
Find(ctx context.Context, storeID string) (*Store, error)
FindAll(ctx context.Context) ([]*Store, error)
<<Interface>>
StoreRepository
Load(ctx context.Context, storeID string) (*Store, error)
Save(ctx context.Context, store *Store) error
<<Interface>>
???
Find(ctx context.Context, storeID string) (*Store, error)
FindAll(ctx context.Context) ([]*Store, error)
FindAllParticipating(ctx context.Context) ([]*Store, error)
<<Interface>>
MallRepository
AddStore(ctx context.Context, storeID, name, location string) error
SetStoreParticipation(ctx context.Context, storeID string, participating bool) error
RenameStore(ctx context.Context, storeID, name string) error
Find(ctx context.Context, storeID string) (*Store, error)
All(ctx context.Context) ([]*Store, error)
AllParticipating(ctx context.Context) ([]*Store, error)
<<Interface>>
EventHandler[T Event]
HandleEvent(ctx context.Context, event T) error
EventHandler[T Event] func(ctx context.Context, event T) error
<<Interface>>
CatalogRepository
AddProduct(ctx context.Context, productID, storeID, name, description, sku string, price float64) error
Rebrand(ctx context.Context, productID, name, description string) error
UpdatePrice(ctx context.Context, productID string, price float64) error
RemoveProduct(ctx context.Context, productID string) error
Find(ctx context.Context, productID string) (*Product, error)
GetCatalog(ctx context.Context, storeID string) ([]*Product, error)

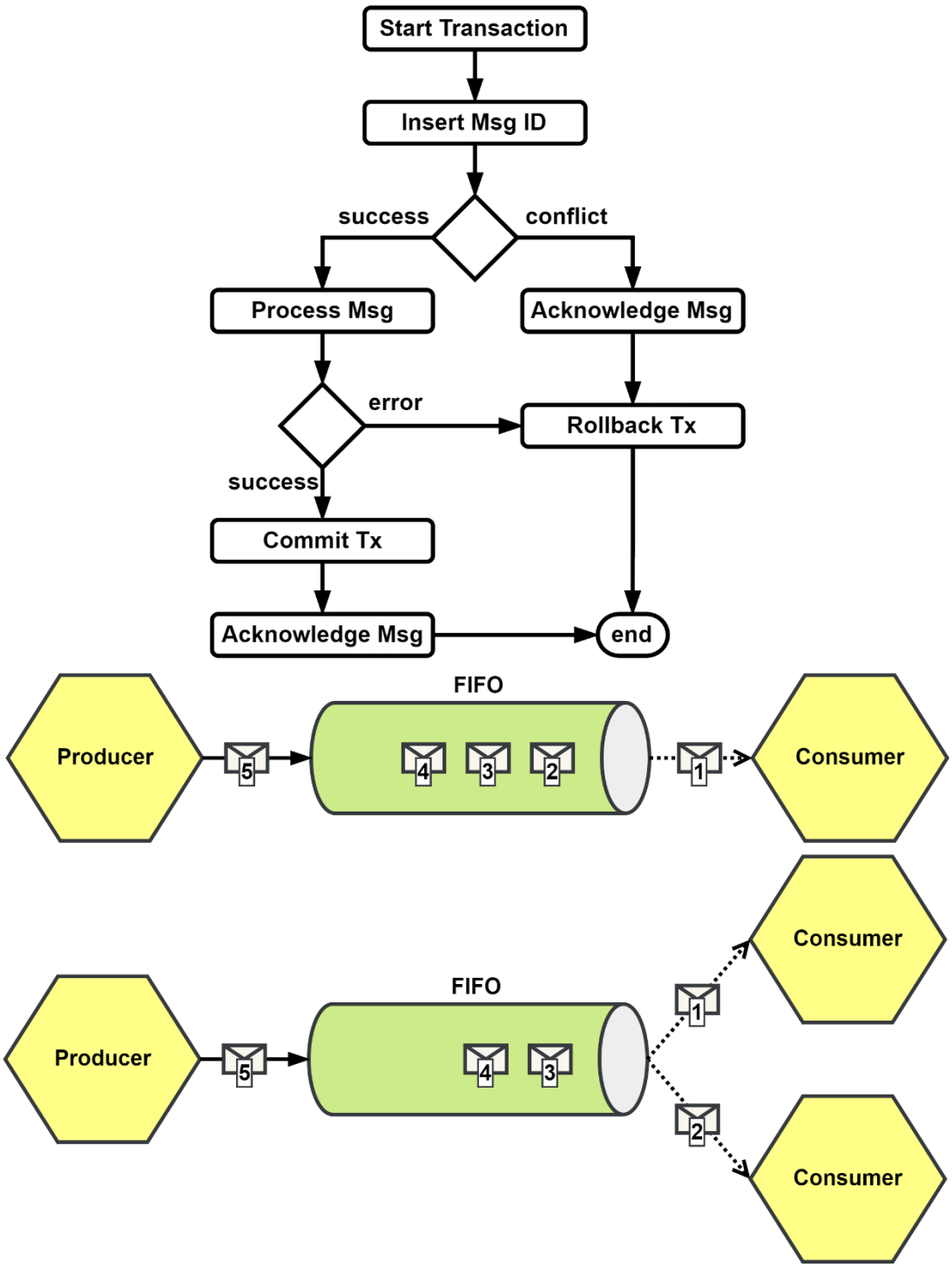

AggregateRepository
Load()/Save()
return error
Other
return error
EventPublisher
return error
Another
return error
EventStore
next()
next()
next()
Event
Snapshot: &domain.ProductV1{
StoreID: "nicknack",
Name: "Wizard w/ Crystal",
Description: "Robed wizard w/ ...",
SKU: "wizard",
Price: 28.99,
},
Version: 5
<<Interface>>
Snapshot
SnapshotName()
<<Interface>>
Snapshotter
ApplySnapshot(snapshot Snapshot) error
ToSnapshot() Snapshot
Snapshot: &domain.ProductV1{
StoreID: "nicknack",
Name: "Wizard w/o Crystal",
Description: "Robed wizard w/ ...",
SKU: "wizard",
Price: 23.99,
},
Version: 5
Event
&domain.Product{
StoreID: "nicknack",
Name: "Wizard w/ Crystal",
Description: "Robed wizard w/ ...",
SKU: "wizard",
Price: 28.99,
}

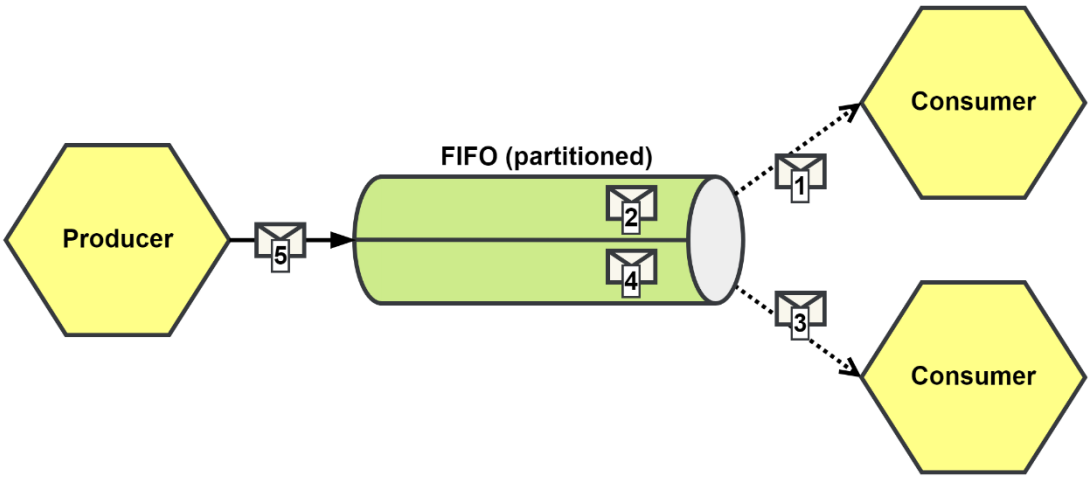
Chapter 6: Asynchronous Connections

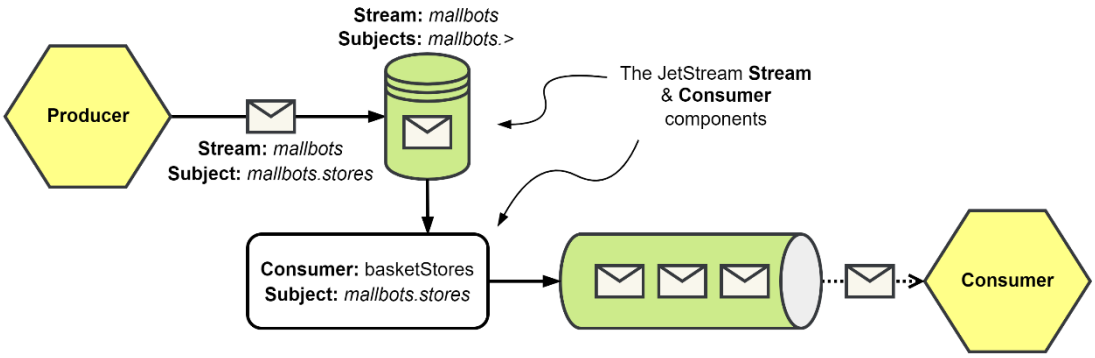


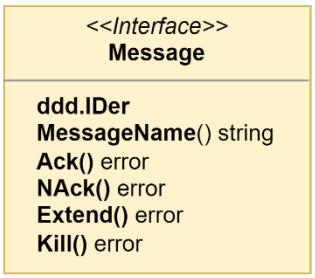

Producer
Message Broker
Consumer
1st Notification
2nd Notification
1st Notification
1st Callback
3rd Notification
2nd Notification
2nd Callback
3rd Notification
3rd Callback
Producer
Message Broker
Consumer

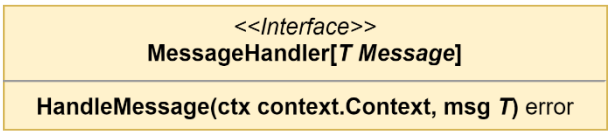

User
Service
Message Broker
Replica
Update Profile
Profile Updated
Success
Get Profile
(stale) Profile
Confusion ???
Profile Updated
User
Service
Message Broker
Replica
Producer
Publish
Message Broker
Receive
Sent then deleted
Consumer
Producer
Publish
Message Broker
Receive
Consumer
Publish Acknowledgement
Receive Acknowledgement
Producer
Publish
Message Broker
Receive
Consumer
Publish Acknowledgement
Receive Acknowledgement

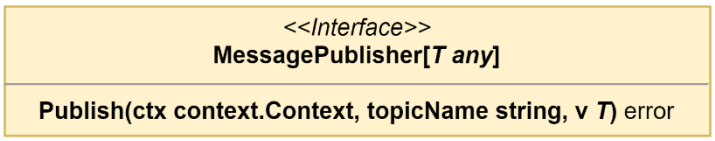

Start Transaction
Insert Msg ID
success
conflict
Process Msg
Acknowledge Msg
error
Rollback Tx
success
Commit Tx
Acknowledge Msg
end
FIFO
Producer
5
4
3
2
1
Consumer
Consumer
FIFO
Producer
5
4
3
1
2
Consumer


Producer
FIFO (partitioned)
Consumer
Consumer
1
2
3
4
5

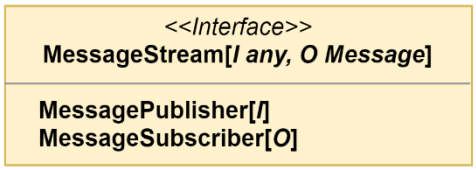

Stream: mallbots
Subjects: mallbots.>
Producer
Stream: mallbots
Subject: mallbots.stores
The JetStream Stream
& Consumer
components
Consumer: basketStores
Subject: mallbots.stores
Consumer


<<Interface>>
Message
ddd.IDer
MessageName() string
Ack() error
NAck() error
Extend() error
Kill() error


<<Interface>>
MessageHandler[T Message]
HandleMessage(ctx context.Context, msg T) error

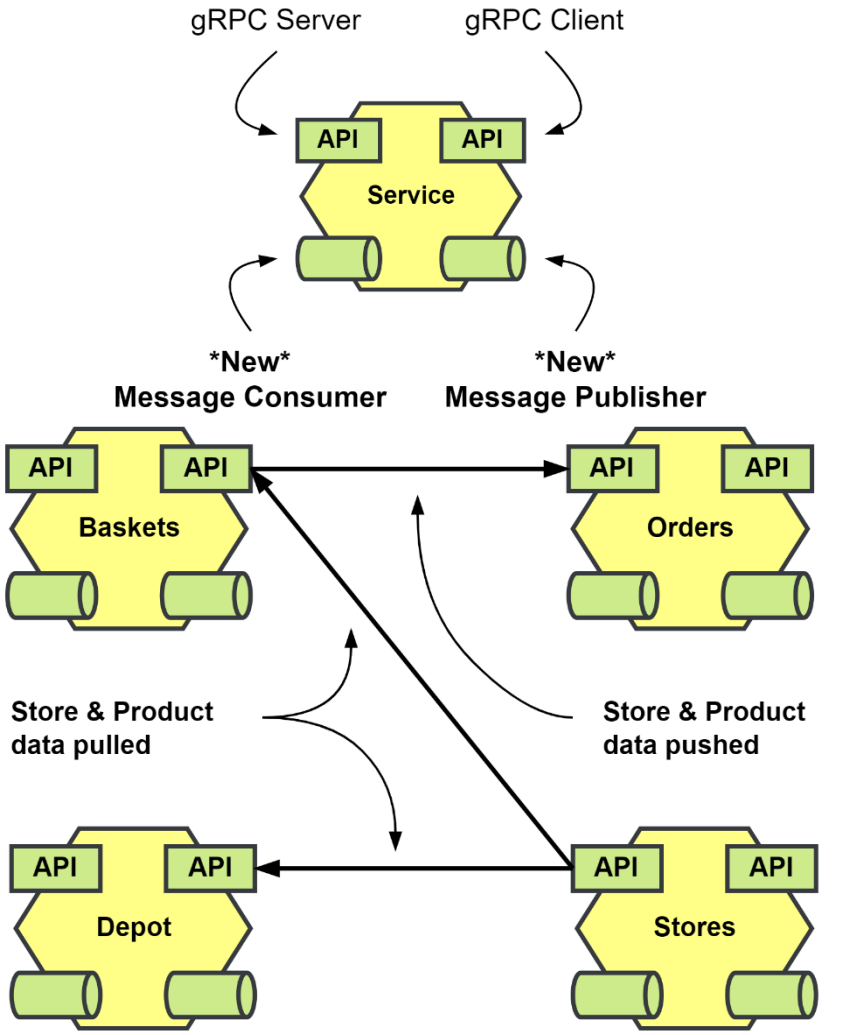

<<Interface>>
MessagePublisher[T any]
Publish(ctx context.Context, topicName string, v T) error

<<Interface>>
MessageSubscriber[T Message]
Subscribe(topicName string, handler MessagHandler[T], options ...SubscriberOption) error


<<Interface>>
MessageStream[I any, O Message]
MessagePublisher[I]
MessageSubscriber[O]

<<Interface>>
RawMessage
Message
Data() []byte
rawMessage
id string
name string
data []byte
ID() string
MessageName() string
Data() []byte
Ack() error
NAck() error
Extend() error
Kill() error
eventStream
reg registry.Registry
stream MessageStream[RawMessage, RawMessage]
Publish(ctx context.Context, topicName string, event ddd.Event) error
Subscribe(topicName string, handler MessageHandler[EventMessage], options ...SubscriberOptions) error
<<Interface>>
EventMessage
Message
ddd.Event
eventMessage
id string
name string
payload ddd.EventPayload
metadata ddd.Metadata
occurredAt time.Time
msg RawMessage
ID() string
EventName() string
MessageName() string
Payload() ddd.EventPayload
Metadata() ddd.Metadata
OccurredAt() time.Time
Ack() error
NAck() error
Extend() error
Kill() error

Chapter 7: Event-carried State Transfer

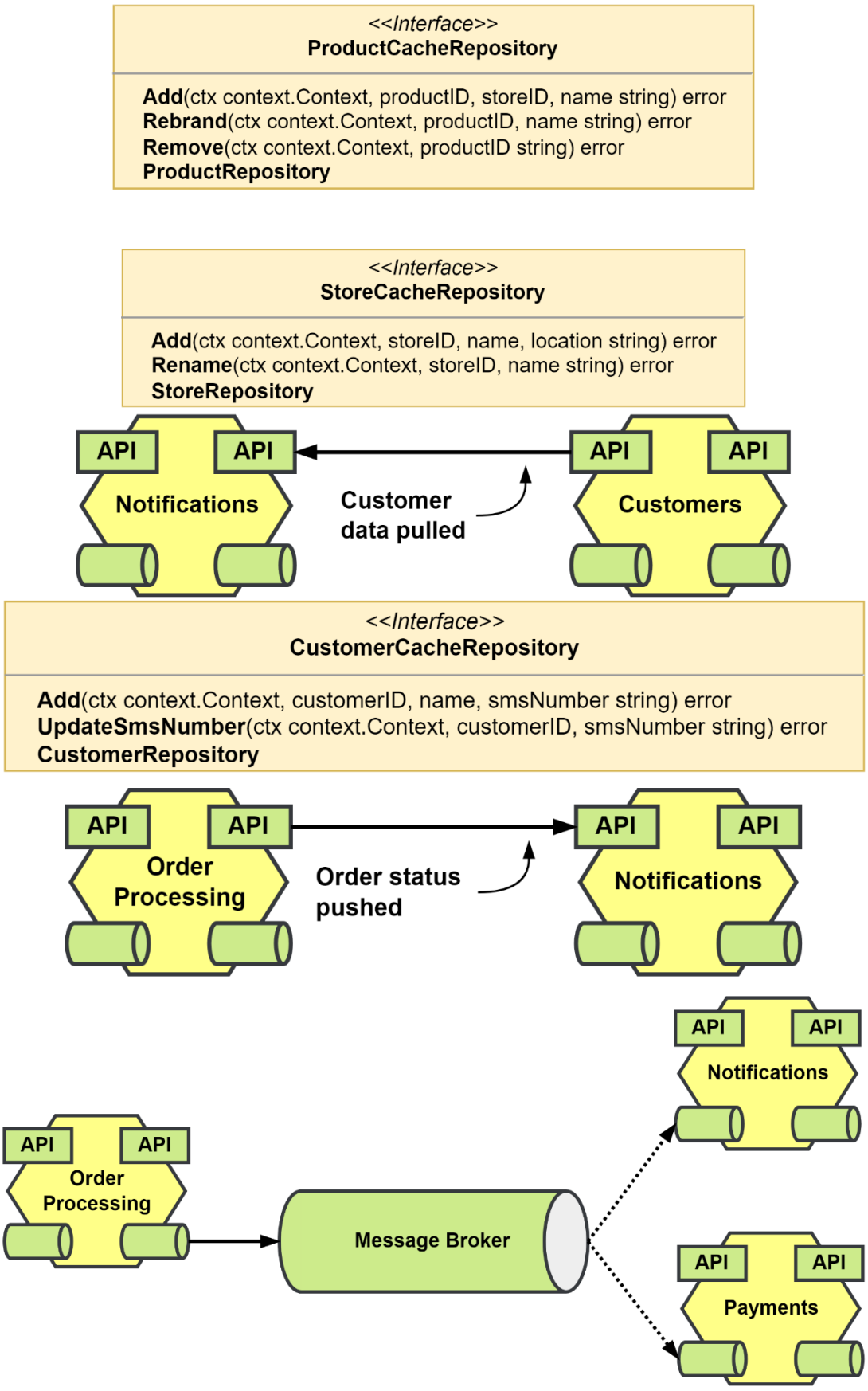


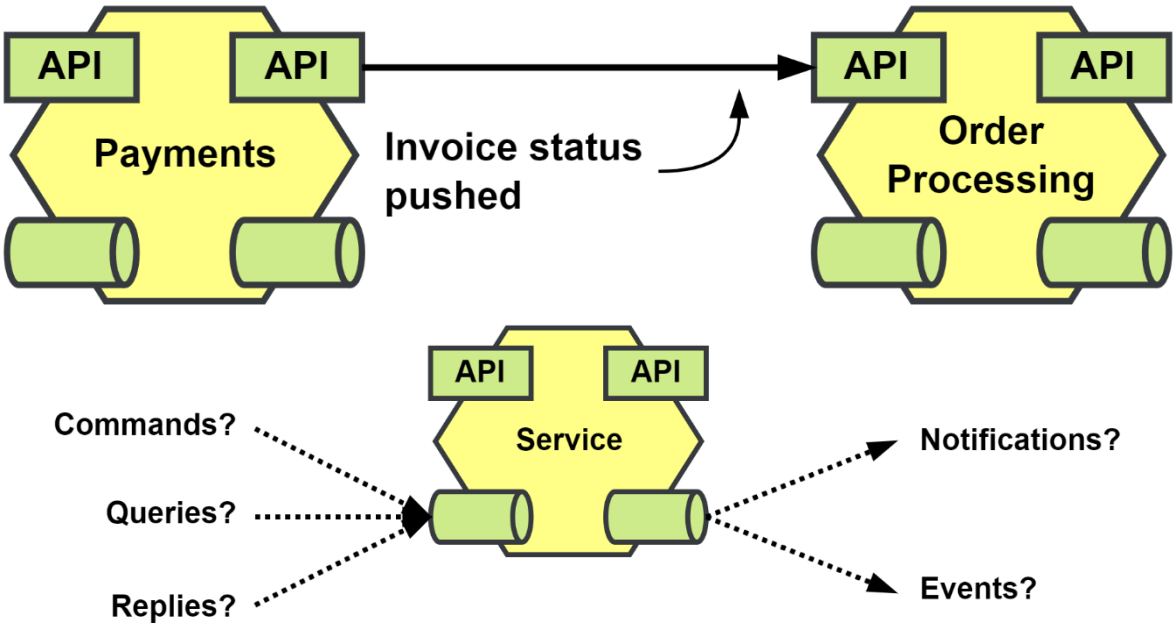
<<*Interface*>>
ProductCacheRepository

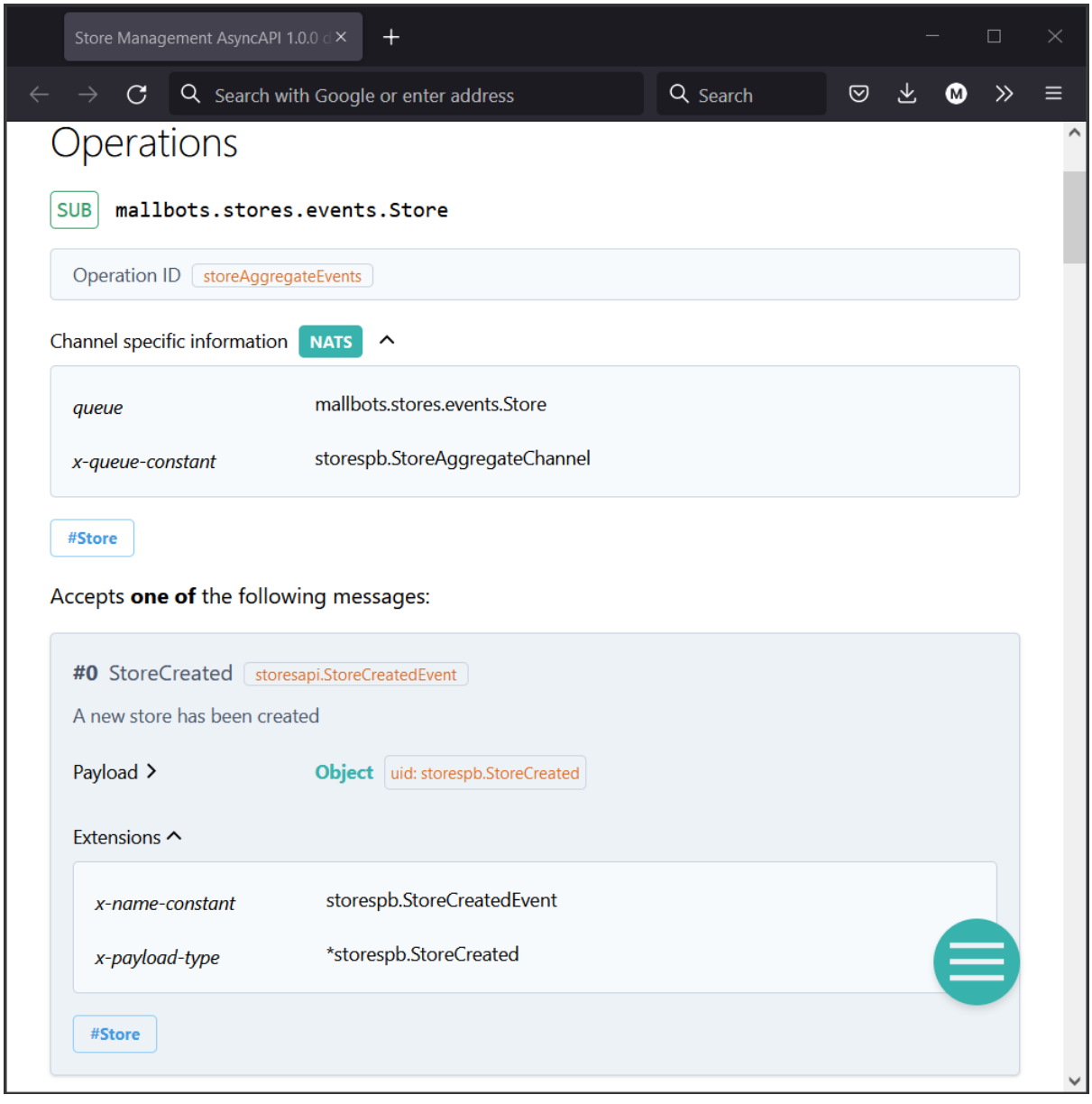
Add(ctx context.Context, productID, storeID, name string, price float64) error
Rebrand(ctx context.Context, productID, name string) error
UpdatePrice(ctx context.Context, productID string, price float64) error
Remove(ctx context.Context, productID string) error
ProductRepository

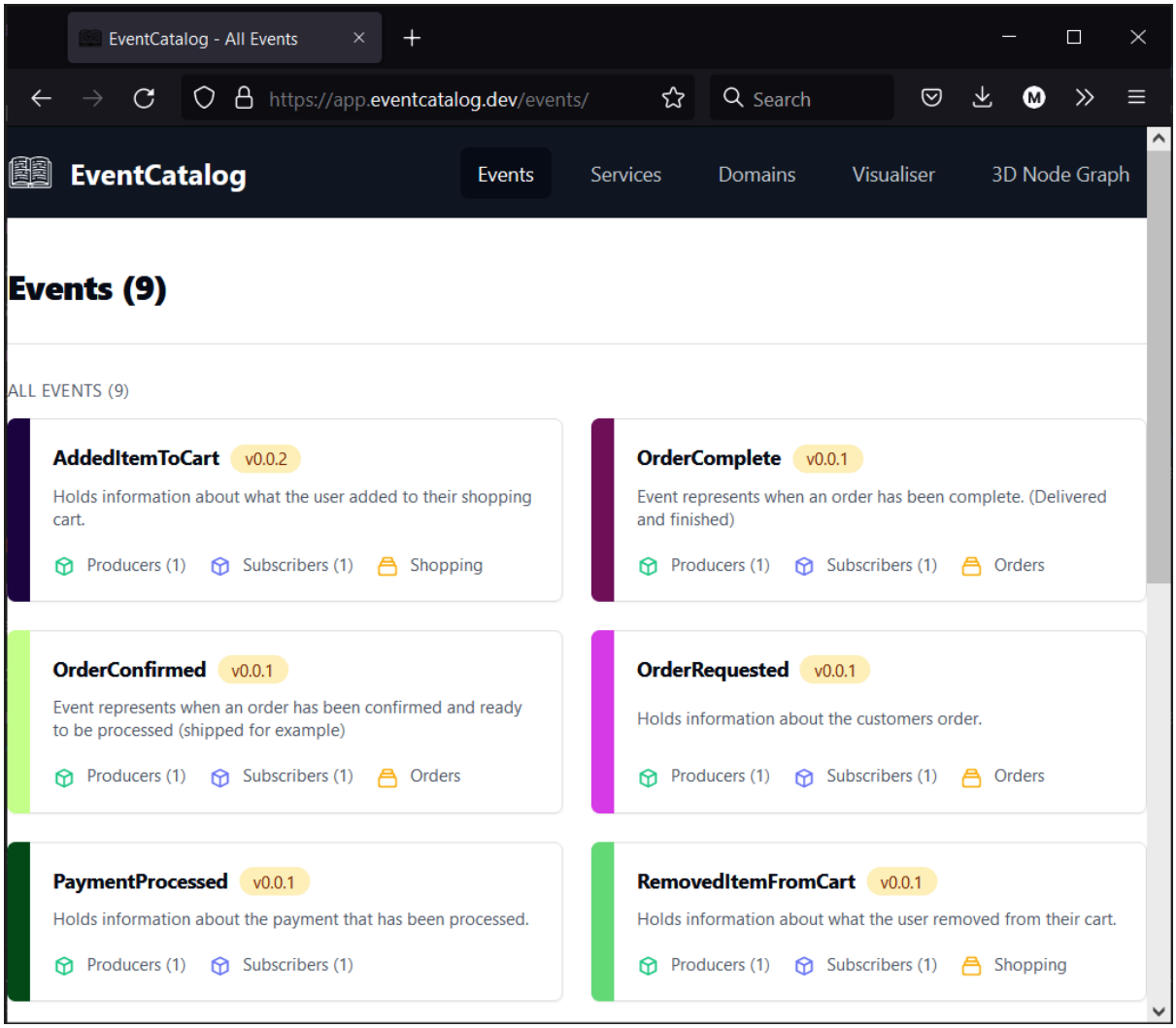
<<*Interface*>>
StoreCacheRepository

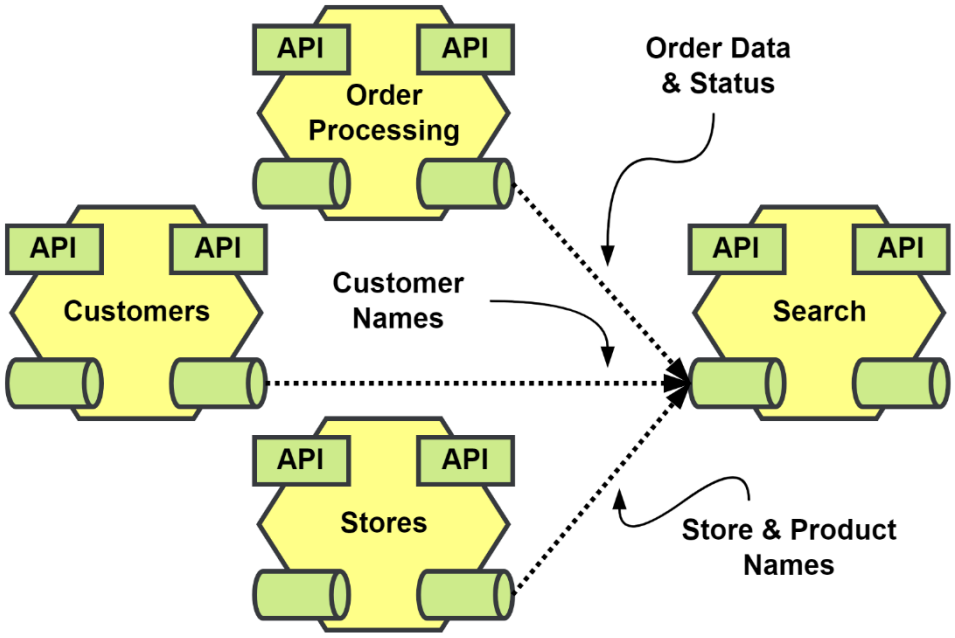
Add(ctx context.Context, storeID, name string) error
Rename(ctx context.Context, storeID, name string) error
StoreRepository


<<Interface>>
ProductCacheRepository
Add(ctx context.Context, productID, storeID, name string) error
Rebrand(ctx context.Context, productID, name string) error
Remove(ctx context.Context, productID string) error
ProductRepository
<<Interface>>
StoreCacheRepository
Add(ctx context.Context, storeID, name, location string) error
Rename(ctx context.Context, storeID, name string) error
StoreRepository
API
API
Notifications
Customer
data pulled
API
API
Customers
<<Interface>>
CustomerCacheRepository
Add(ctx context.Context, customerID, name, smsNumber string) error
UpdateSmsNumber(ctx context.Context, customerID, smsNumber string) error
CustomerRepository
API
API
Order
Processing
Order status
pushed
API
API
Notifications
API
API
Order
Processing
Message Broker
API
API
Notifications
API
API
Payments


API
API
Payments
Invoice status pushed
API
API
Order Processing
API
API
Service
Commands?
Queries?
Replies?
Notifications?
Events?

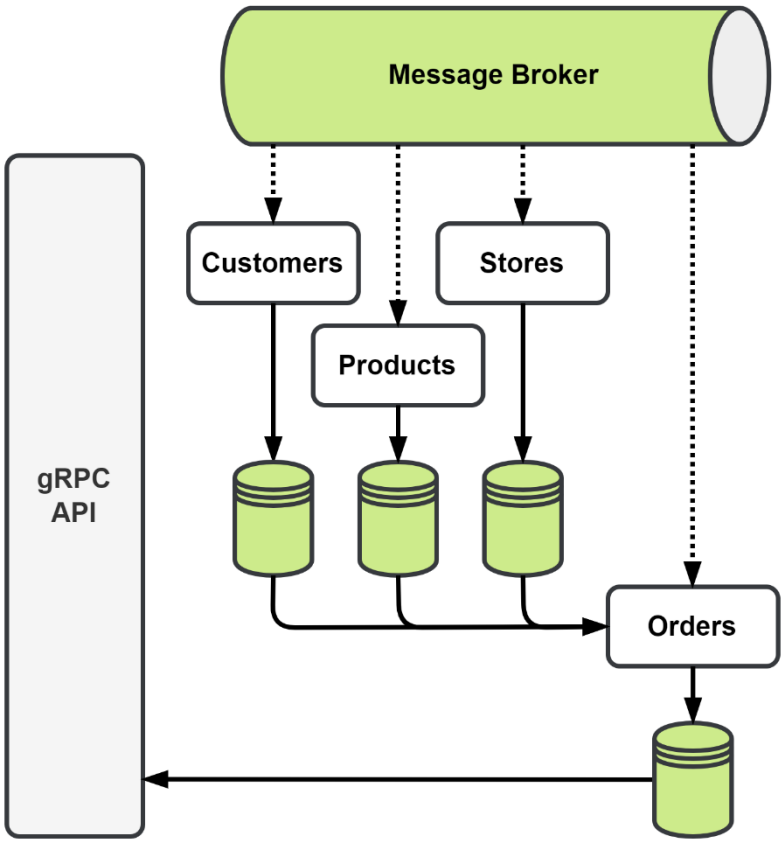

Store Management AsyncAPI 1.0.0
Search with Google or enter address
Search
Operations
SUB
mallbots.stores.events.Store
Operation ID
storeAggregateEvents
Channel specific information
NATS
queue
mallbots.stores.events.Store
x-queue-constant
storespb.StoreAggregateChannel
#Store
Accepts one of the following messages:
#0 StoreCreated
storesapi.StoreCreatedEvent
A new store has been created
Payload
Object
uid: storespb.StoreCreated
Extensions
x-name-constant
storespb.StoreCreatedEvent
x-payload-type
*storespb.StoreCreated
#Store


EventCatalog - All Events
https://app.eventcatalog.dev/events/
Search
EventCatalog
Events
Services
Domains
Visualiser
3D Node Graph
Events (9)
ALL EVENTS (9)
AddedItemToCart v0.0.2
Holds information about what the user added to their shopping cart.
Producers (1) Subscribers (1) Shopping
OrderComplete v0.0.1
Event represents when an order has been complete. (Delivered and finished)
Producers (1) Subscribers (1) Orders
OrderConfirmed v0.0.1
Event represents when an order has been confirmed and ready to be processed (shipped for example)
Producers (1) Subscribers (1) Orders
OrderRequested v0.0.1
Holds information about the customers order.
Producers (1) Subscribers (1) Orders
PaymentProcessed v0.0.1
Holds information about the payment that has been processed.
Producers (1) Subscribers (1)
RemovedItemFromCart v0.0.1
Holds information about what the user removed from their cart.
Producers (1) Subscribers (1) Shopping

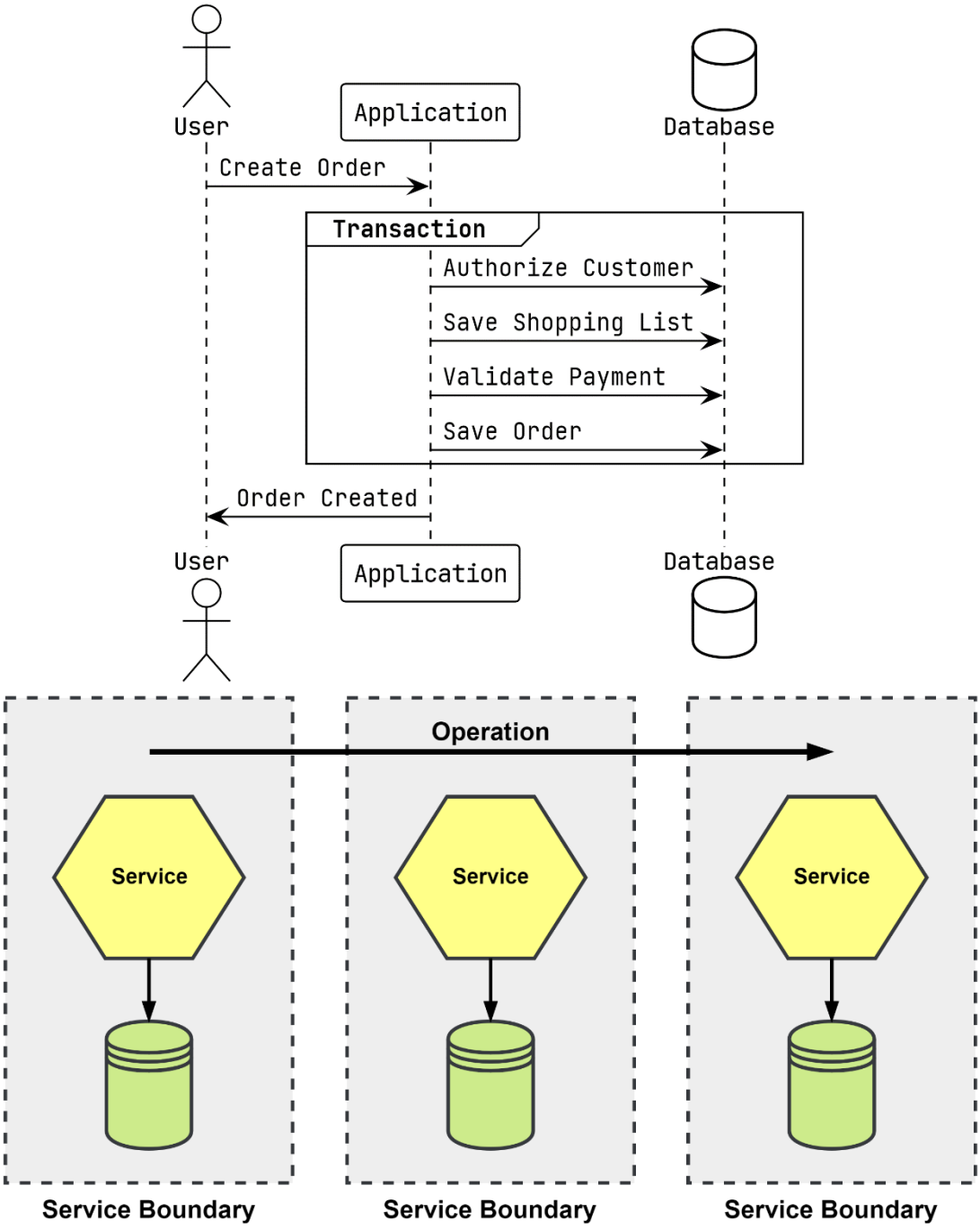

API
API
Order Processing
Order Data & Status
API
API
Customers
Customer Names
API
API
Search
API
API
Stores
Store & Product Names

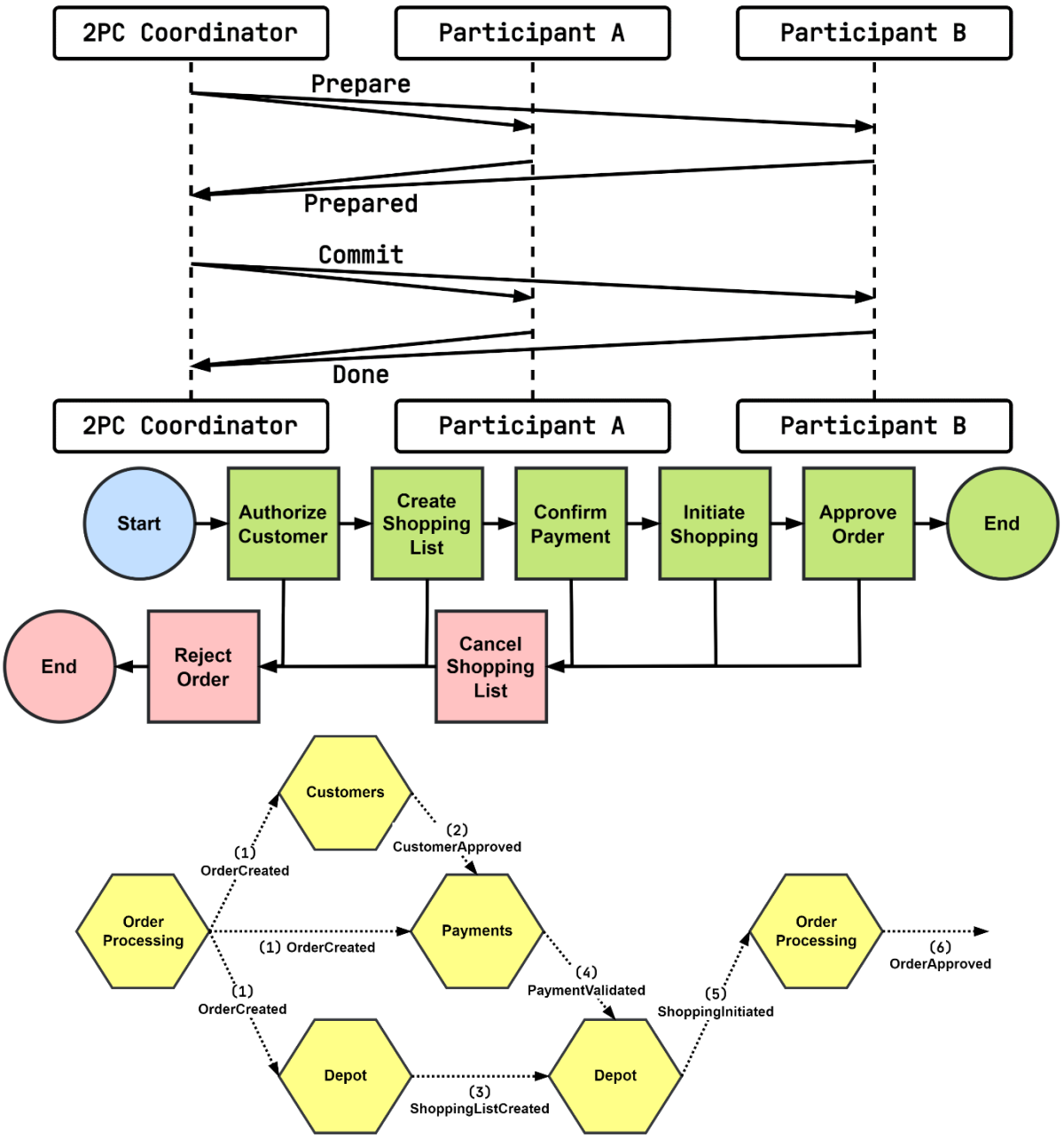
Order	
OrderID	string
CustomerID	string
CustomerName	string
Items	[]Item
Total	float64
Status	string
CreatedAt	time.Time

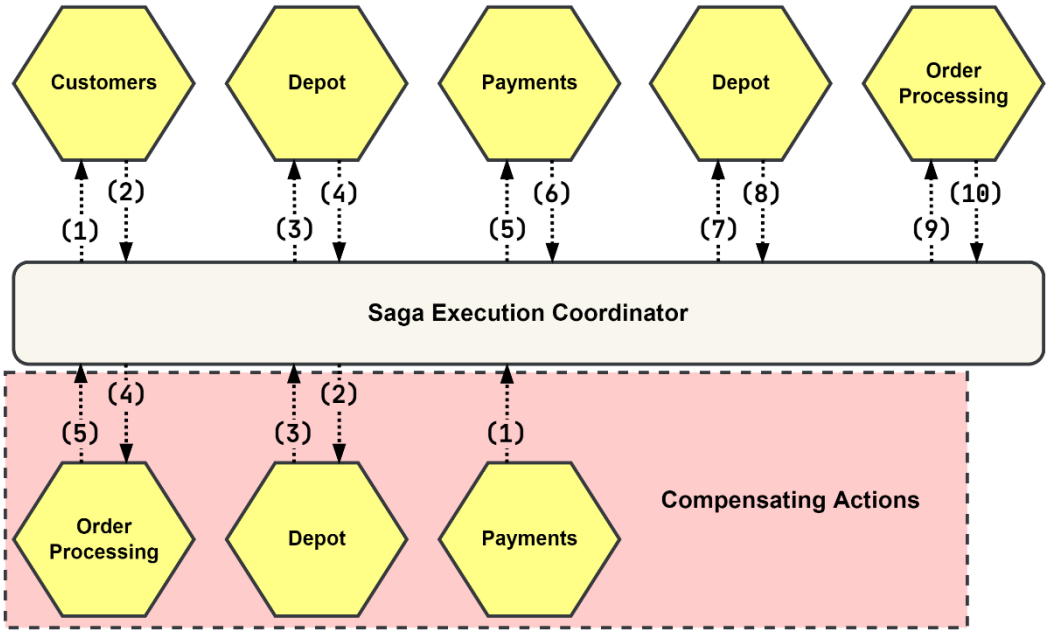
Item	
ProductID	string
StoreID	string
ProductName	string
StoreName	string
Price	float64
Quantity	int



Chapter 8: Event Workflows




2PC Coordinator
Participant A
Participant B
Prepare
Prepared
Commit
Done
2PC Coordinator
Participant A
Participant B
Start
Authorize Customer
Create Shopping List
Confirm Payment
Initiate Shopping
Approve Order
End
End
Reject Order
Cancel Shopping List
Customers
(1) OrderCreated
(2) CustomerApproved
Order Processing
(1) OrderCreated
Payments
(1) OrderCreated
(4) PaymentValidated
Order Processing
(6) OrderApproved
(5) ShoppingInitiated
Depot
(3) ShoppingListCreated
Depot



reply

Entity
payload ReplyPayload
metadata Metadata
occurredAt time.Time

ReplyName() string
Payload() ReplyPayload
Metadata() Metadata
OccurredAt() time.Time

<<Interface>>
Reply

IDer
ReplyName() string
Payload() ReplyPayload
Metadata() Metadata
OccurredAt() time.Time

<<Interface>>
ReplyOption

configureReply(*reply)

<<Interface>>
ReplyPayload

ReplyHandler[*T Reply*] func(ctx context.Context, reply T) error

CommandHandler[*T Command*] func(ctx context.Context, cmd T) (Reply, error)

<<Interface>>
CommandMessageHandler

HandleMessage(ctx context.Context, msg CommandMessage) (ddd.Reply, error)

<<Interface>>
CommandSubscriber

Subscribe(topicName string, handler CommandMessagHandler, options ...SubscriberOption) error

<<Interface>>
CommandStream

MessagePublisher[*ddd.Command*]
CommandSubscriber

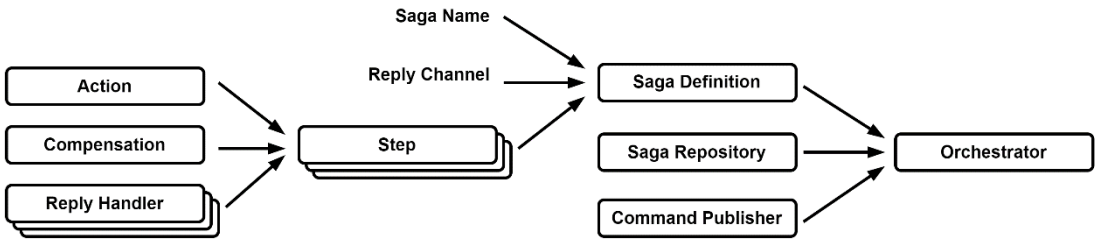
<<Interface>>
CommandMessage

Message
ddd.Command

commandStream

reg registry.Registry
stream MessageStream[RawMessage, RawMessage]

Publish(ctx context.Context, topicName string, event ddd.Event) error
Subscribe(topicName string, handler CommandMessageHandler, options ...SubscriberOptions) error



<<Interface>>
Orchestrator[*T any*]

Start(ctx context.Context, sagaID string, data T) error
ReplyTopic() string
HandleReply(ctx context.Context, reply ddd.Reply) error

orchestrator[*T any*]

saga Saga[T]
repo SagaRepository[T]
publisher am.CommandPublisher

Start(ctx context.Context, sagaID string, data T) error
ReplyTopic() string
HandleReply(ctx context.Context, reply ddd.Reply) error

<<Interface>>
Saga[*T any*]

AddStep() SagaStep[T]
Name() string
ReplyTopic() string

saga[*T any*]

name string
replyTopic string
steps []SagaStep[T]

AddStep() SagaStep[T]
Name() string
ReplyTopic() string

StepActionFunc[*T any*] func(ctx context.Context, data T) am.Command

StepRepyHandlerFunc[*T any*] func(ctx context.Context, data T, reply ddd.Reply) error

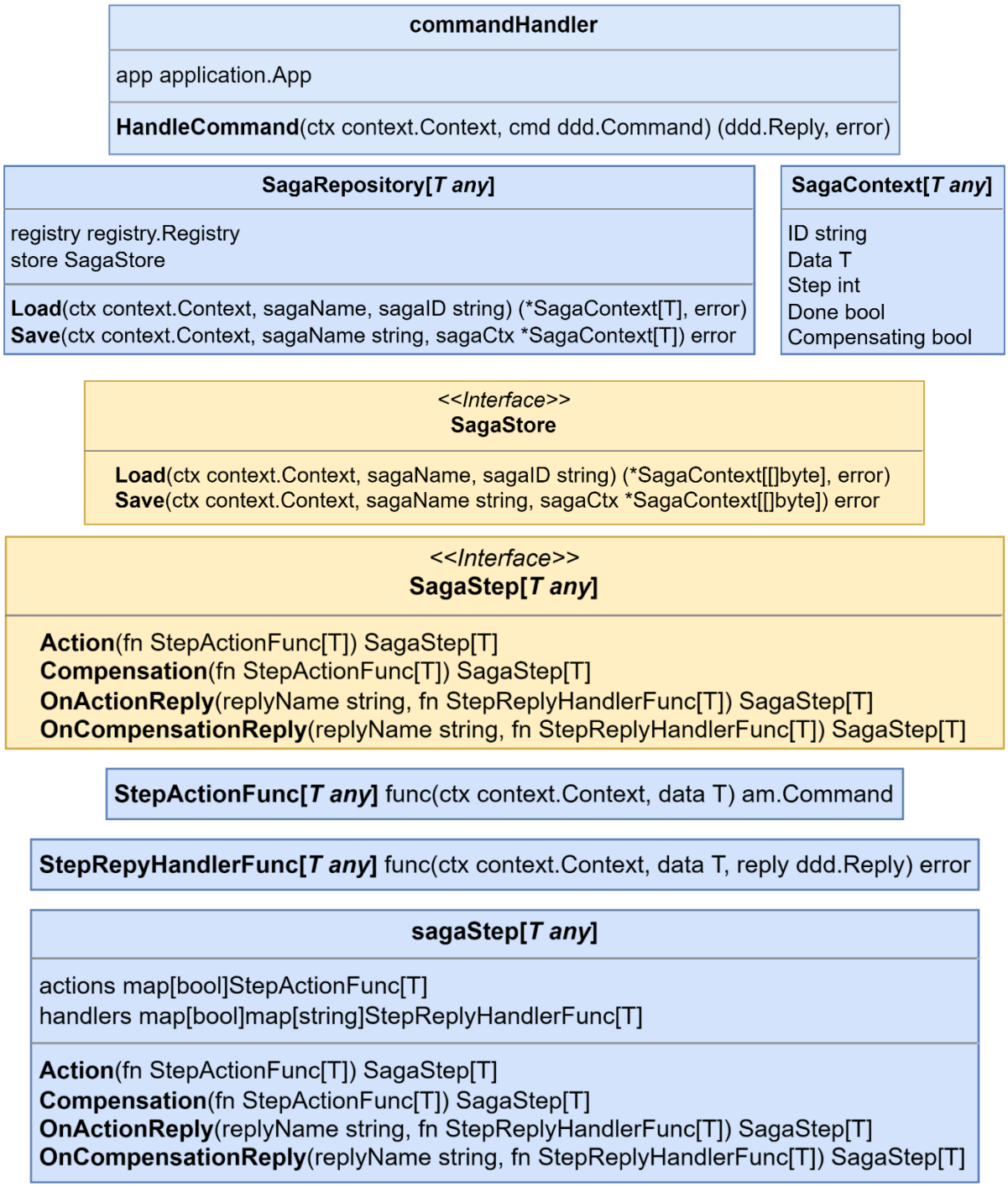
<<Interface>>
SagaStep[*T any*]

Action(fn StepActionFunc[T]) SagaStep[T]
Compensation(fn StepActionFunc[T]) SagaStep[T]
OnActionReply(replyName string, fn StepReplyHandlerFunc[T]) SagaStep[T]
OnCompensationReply(replyName string, fn StepReplyHandlerFunc[T]) SagaStep[T]

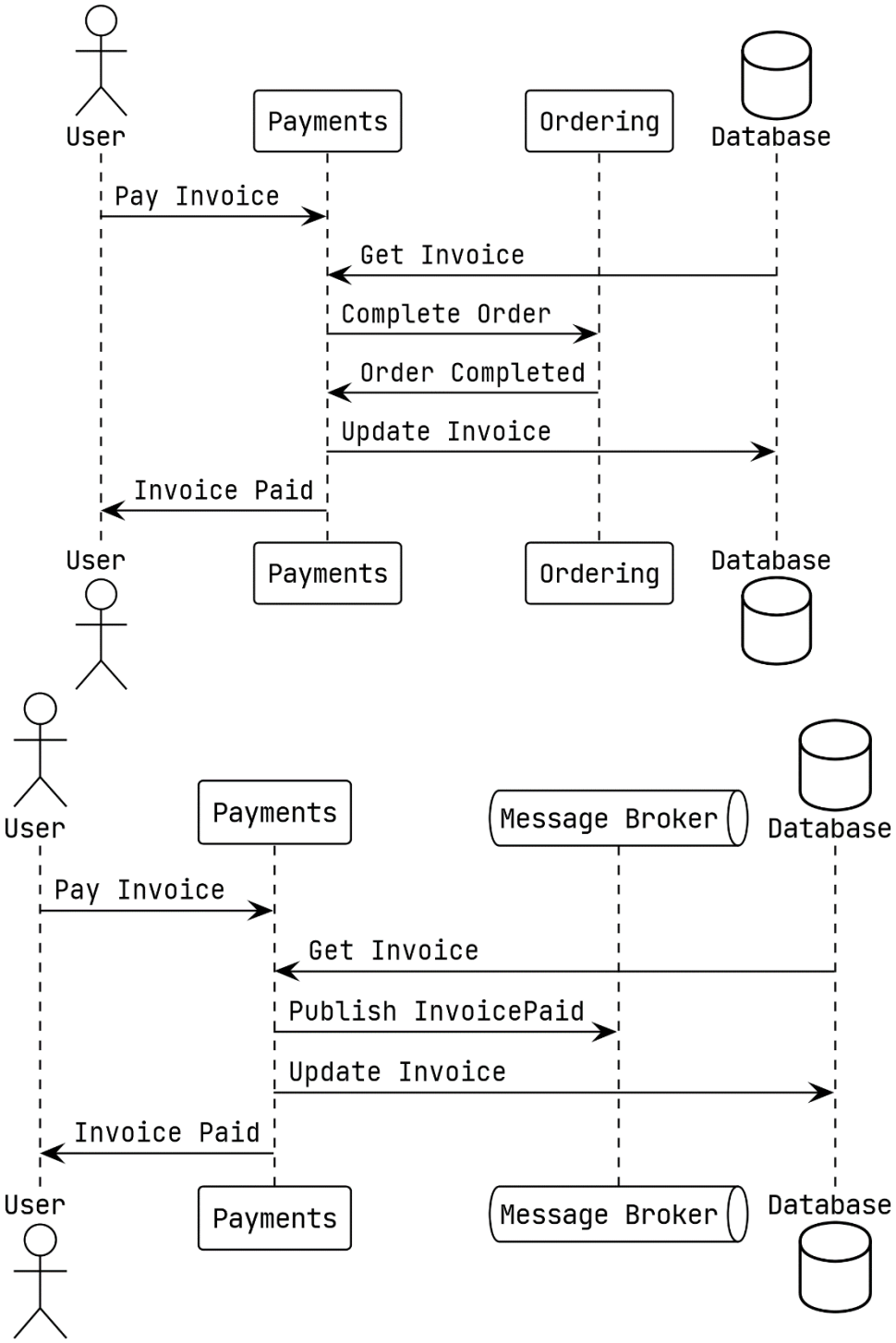
sagaStep[*T any*]

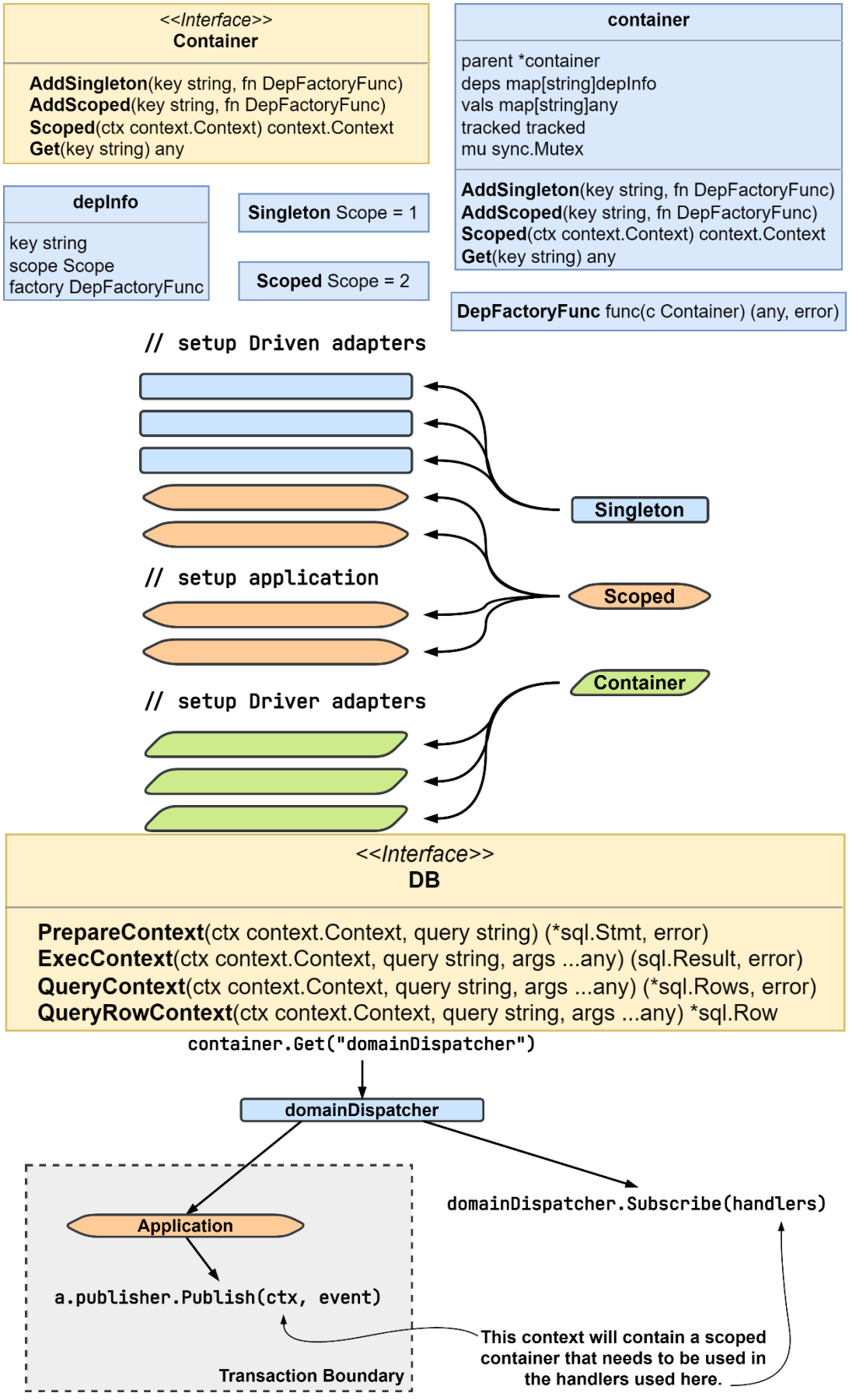
actions map[bool]StepActionFunc[T]
handlers map[bool]map[string]StepReplyHandlerFunc[T]

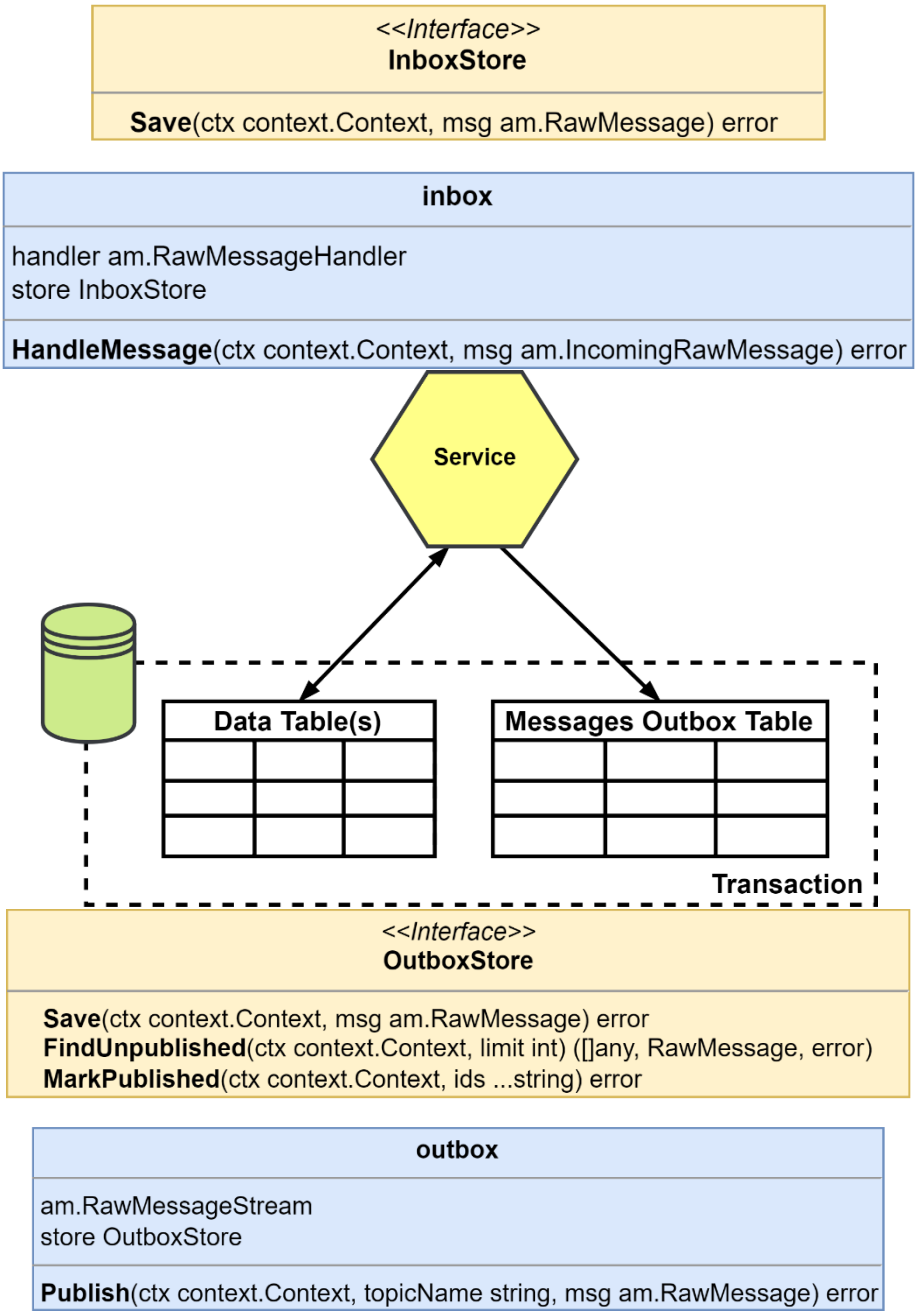
Action(fn StepActionFunc[T]) SagaStep[T]
Compensation(fn StepActionFunc[T]) SagaStep[T]
OnActionReply(replyName string, fn StepReplyHandlerFunc[T]) SagaStep[T]
OnCompensationReply(replyName string, fn StepReplyHandlerFunc[T]) SagaStep[T]

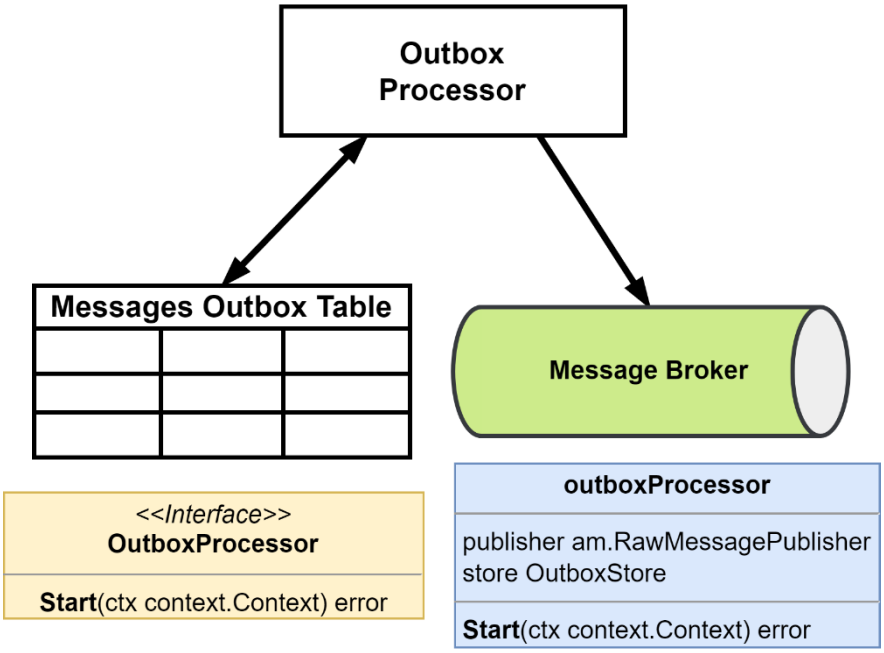

commandHandler
app application.App
HandleCommand(ctx context.Context, cmd ddd.Command) (ddd.Reply, error)
SagaRepository[T any]
registry registry.Registry
store SagaStore
Load(ctx context.Context, sagaName, sagaID string) (*SagaContext[T], error)
Save(ctx context.Context, sagaName string, sagaCtx *SagaContext[T]) error
SagaContext[T any]
ID string
Data T
Step int
Done bool
Compensating bool
<<Interface>>
SagaStore
Load(ctx context.Context, sagaName, sagaID string) (*SagaContext[[]byte], error)
Save(ctx context.Context, sagaName string, sagaCtx *SagaContext[[]byte]) error
<<Interface>>
SagaStep[T any]
Action(fn StepActionFunc[T]) SagaStep[T]
Compensation(fn StepActionFunc[T]) SagaStep[T]
OnActionReply(replyName string, fn StepReplyHandlerFunc[T]) SagaStep[T]
OnCompensationReply(replyName string, fn StepReplyHandlerFunc[T]) SagaStep[T]
StepActionFunc[T any] func(ctx context.Context, data T) am.Command
StepRepyHandlerFunc[T any] func(ctx context.Context, data T, reply ddd.Reply) error
sagaStep[T any]
actions map[bool]StepActionFunc[T]
handlers map[bool]map[string]StepReplyHandlerFunc[T]
Action(fn StepActionFunc[T]) SagaStep[T]
Compensation(fn StepActionFunc[T]) SagaStep[T]
OnActionReply(replyName string, fn StepReplyHandlerFunc[T]) SagaStep[T]
OnCompensationReply(replyName string, fn StepReplyHandlerFunc[T]) SagaStep[T]

Chapter 9: Transactional Messaging

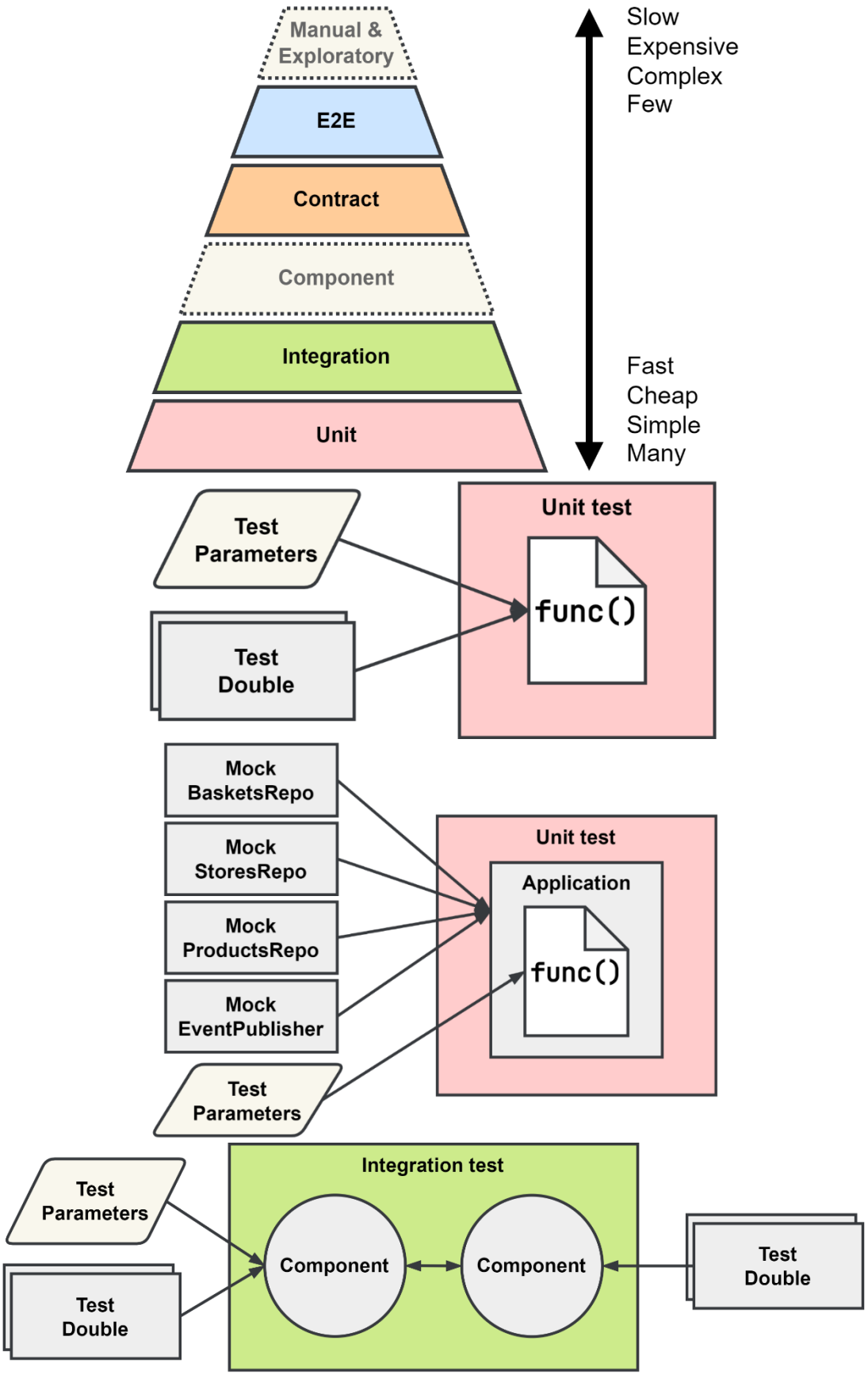


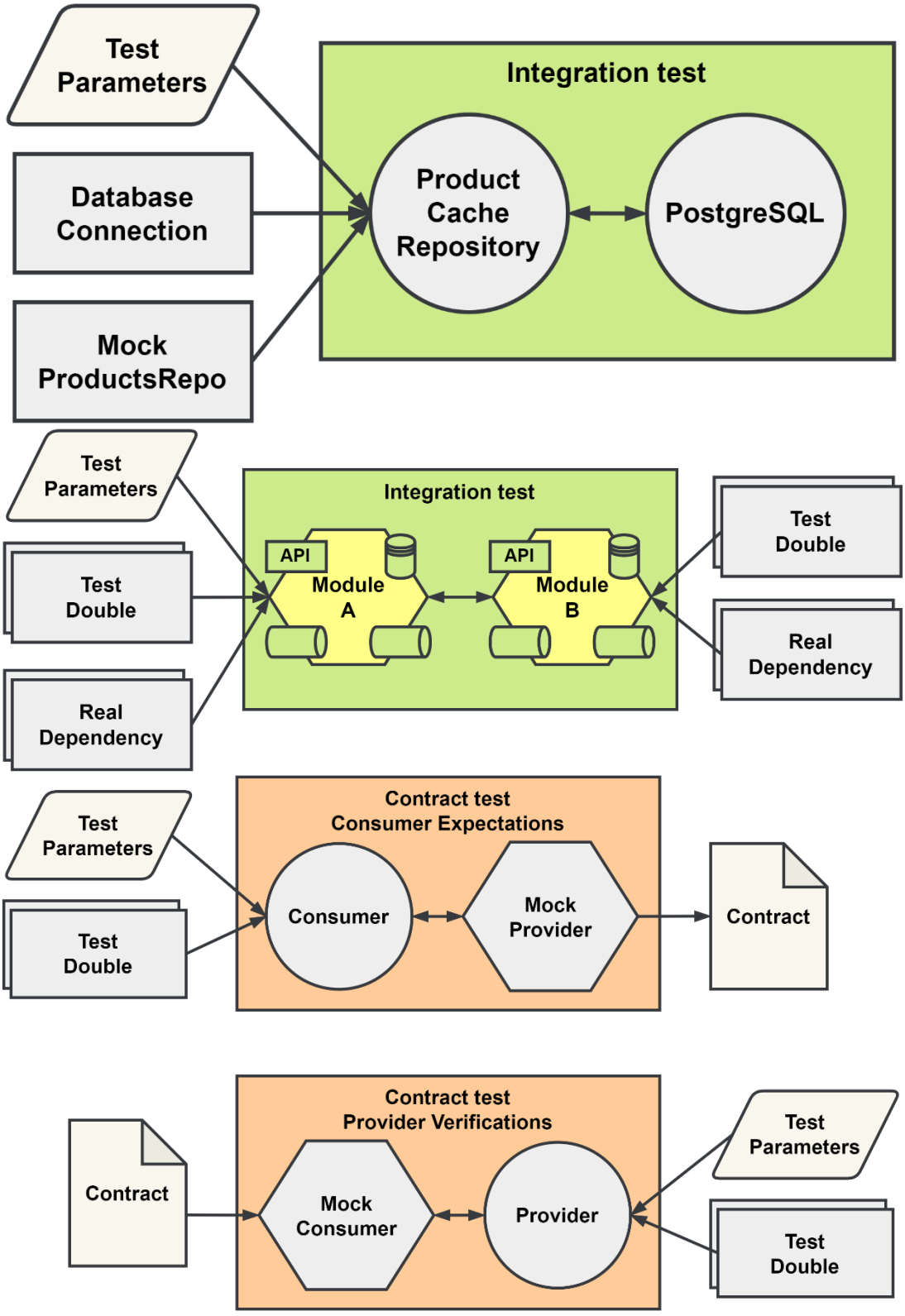

<<Interface>>
Container
AddSingleton(key string, fn DepFactoryFunc)
AddScoped(key string, fn DepFactoryFunc)
Scoped(ctx context.Context) context.Context
Get(key string) any
container
parent *container
deps map[string]depInfo
vals map[string]any
tracked tracked
mu sync.Mutex
AddSingleton(key string, fn DepFactoryFunc)
AddScoped(key string, fn DepFactoryFunc)
Scoped(ctx context.Context) context.Context
Get(key string) any
depInfo
key string
scope Scope
factory DepFactoryFunc
Singleton Scope = 1
Scoped Scope = 2
DepFactoryFunc func(c Container) (any, error)
// setup Driven adapters
Singleton
// setup application
Scoped
Container
// setup Driver adapters
<<Interface>>
DB
PrepareContext(ctx context.Context, query string) (*sql.Stmt, error)
ExecContext(ctx context.Context, query string, args ...any) (sql.Result, error)
QueryContext(ctx context.Context, query string, args ...any) (*sql.Rows, error)
QueryRowContext(ctx context.Context, query string, args ...any) *sql.Row
container.Get("domainDispatcher")
domainDispatcher
Application
a.publisher.Publish(ctx, event)
Transaction Boundary
domainDispatcher.Subscribe(handlers)
This context will contain a scoped container that needs to be used in the handlers used here.

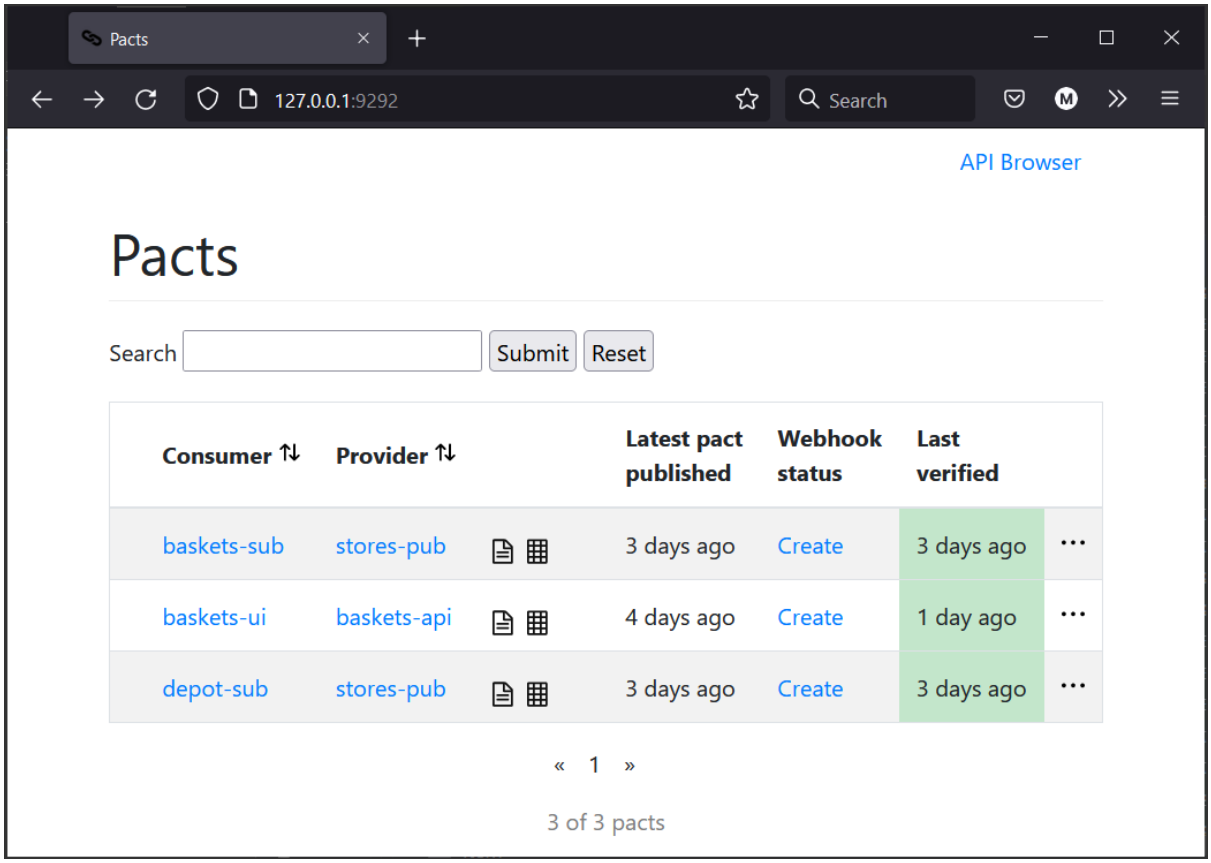

<<Interface>>
InboxStore
Save(ctx context.Context, msg am.RawMessage) error
inbox
handler am.RawMessageHandler
store InboxStore
HandleMessage(ctx context.Context, msg am.IncomingRawMessage) error
Service
Data Table(s)
Messages Outbox Table
Transaction
<<Interface>>
OutboxStore
Save(ctx context.Context, msg am.RawMessage) error
FindUnpublished(ctx context.Context, limit int) ([]any, RawMessage, error)
MarkPublished(ctx context.Context, ids ...string) error
outbox
am.RawMessageStream
store OutboxStore
Publish(ctx context.Context, topicName string, msg am.RawMessage) error

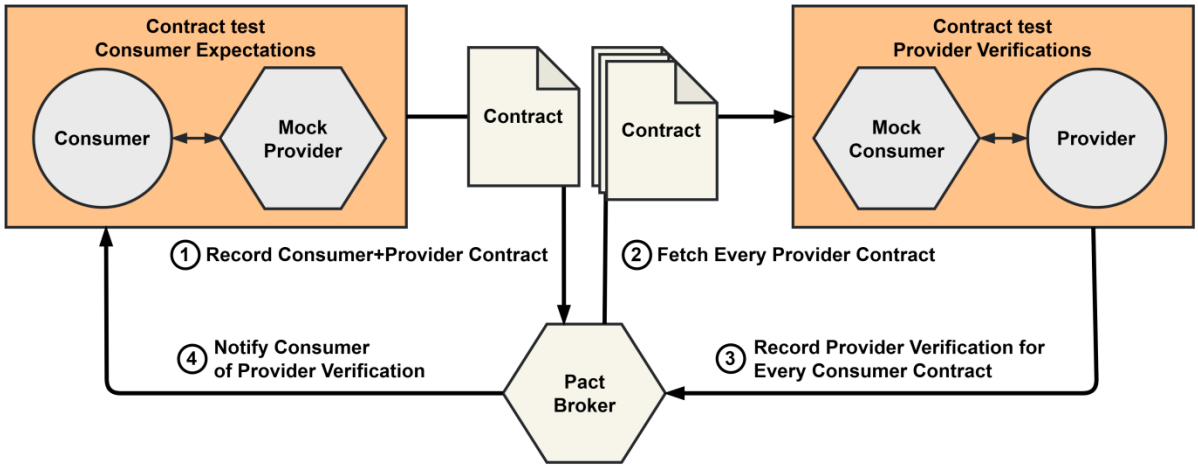

Outbox
Processor
Messages Outbox Table
Message Broker
<<Interface>>
OutboxProcessor
Start(ctx context.Context) error
outboxProcessor
publisher am.RawMessagePublisher
store OutboxStore
Start(ctx context.Context) error

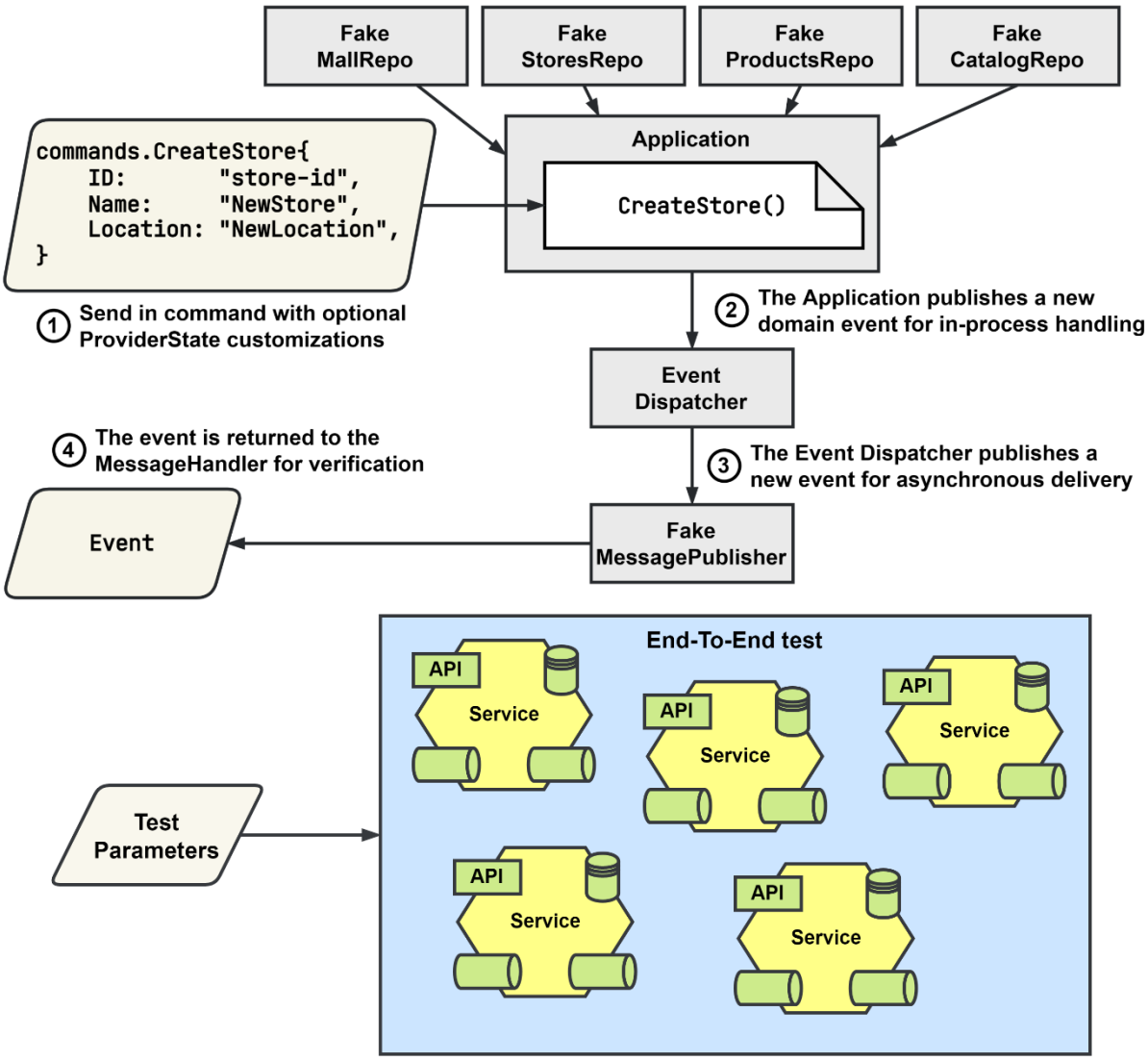
Chapter 10: Testing

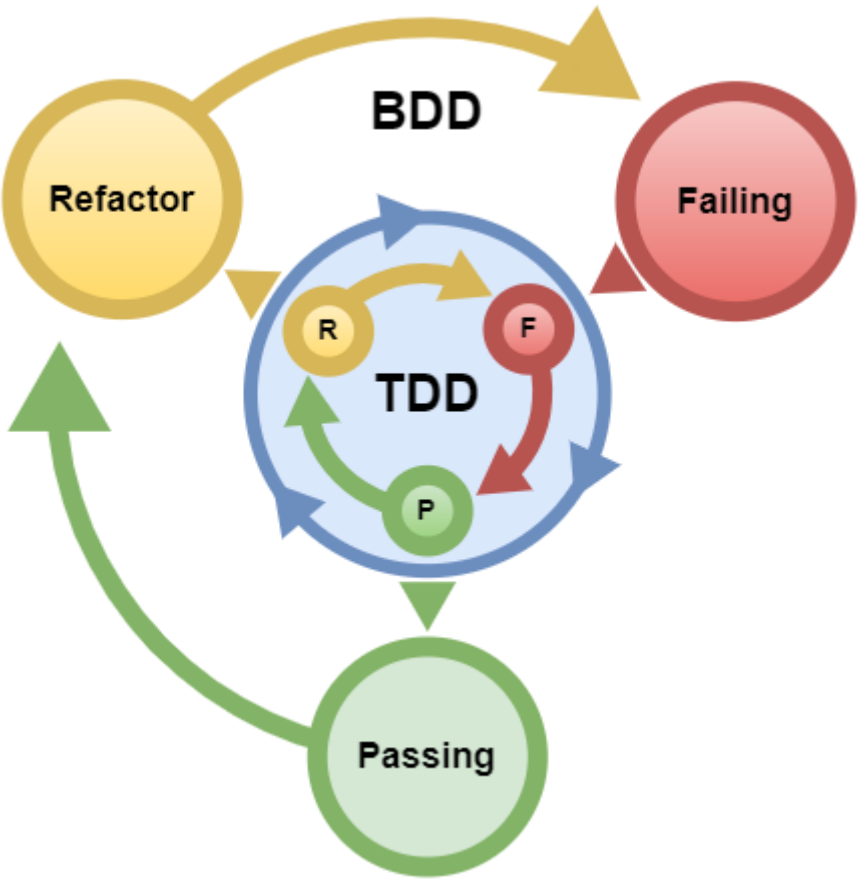



Test Parameters
Database Connection
Mock ProductsRepo
Integration test
Product Cache Repository
PostgreSQL
Test Parameters
Test Double
Real Dependency
Integration test
API
Module A
API
Module B
Test Double
Real Dependency
Test Parameters
Test Double
Contract test
Consumer Expectations
Consumer
Mock Provider
Contract
Contract
Contract test
Provider Verifications
Mock Consumer
Provider
Test Parameters
Test Double

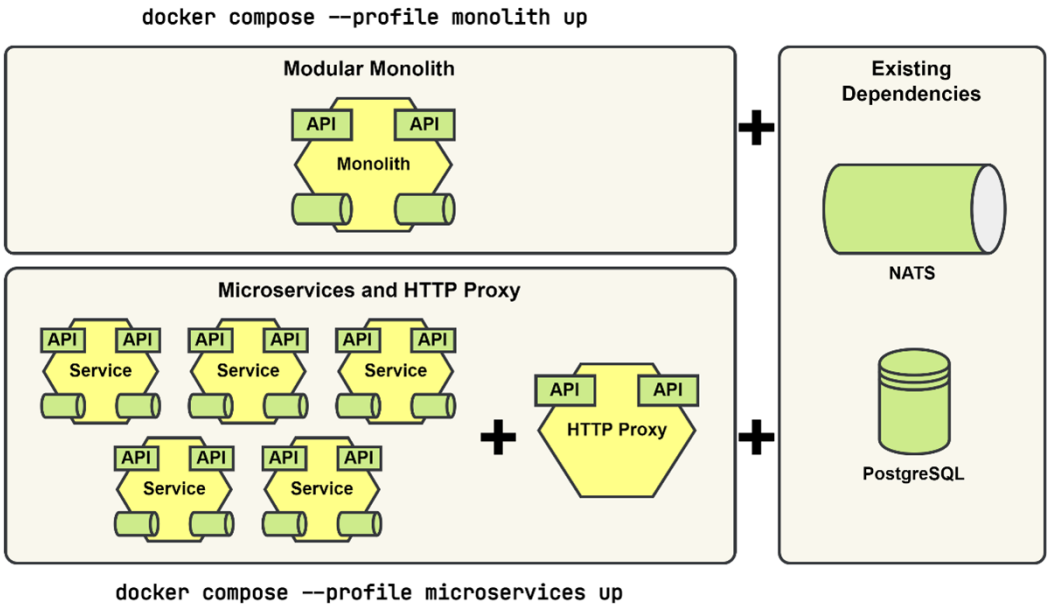

Pacts
127.0.0.1:9292
Search
API Browser
Pacts
Search
Submit
Reset
Consumer
Provider
Latest pact published
Webhook status
Last verified
baskets-sub
stores-pub
3 days ago
Create
3 days ago
baskets-ui
baskets-api
4 days ago
Create
1 day ago
depot-sub
stores-pub
3 days ago
Create
3 days ago
« 1 »
3 of 3 pacts

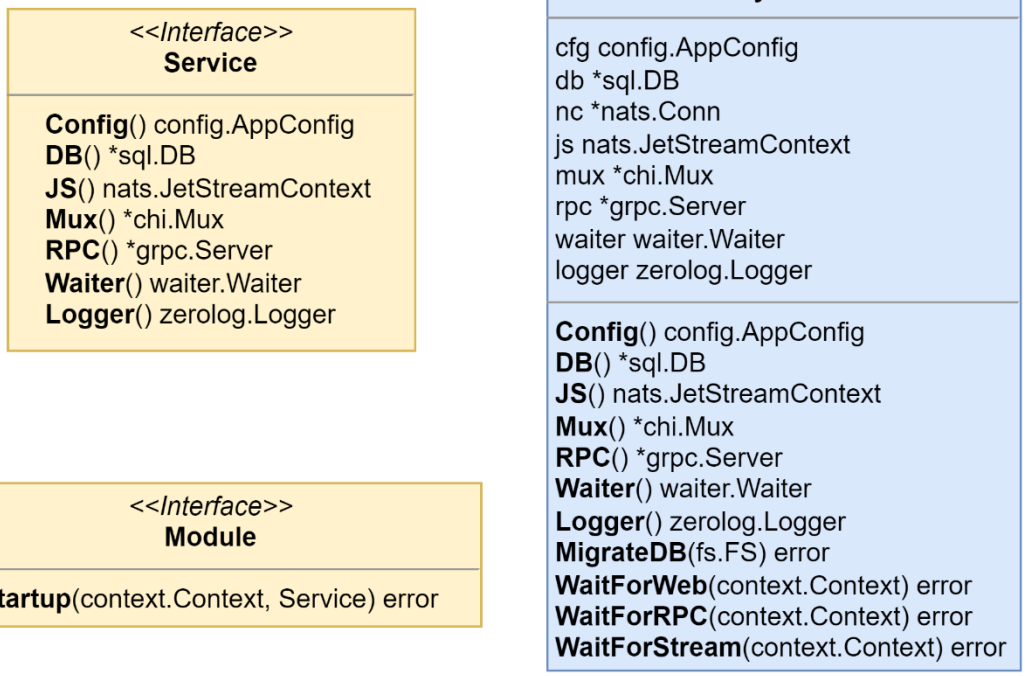

Contract test
Consumer Expectations
Consumer
Mock Provider
Contract
Contract
Contract test
Provider Verifications
Mock Consumer
Provider
1 Record Consumer+Provider Contract
2 Fetch Every Provider Contract
4 Notify Consumer of Provider Verification
Pact Broker
3 Record Provider Verification for Every Consumer Contract

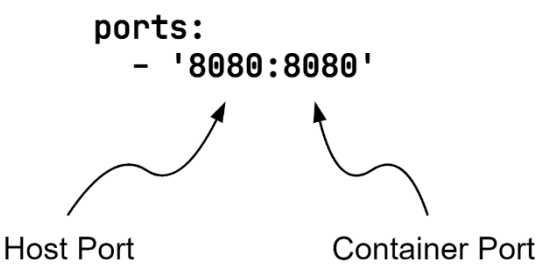

Fake MallRepo
Fake StoresRepo
Fake ProductsRepo
Fake CatalogRepo
commands.CreateStore{
ID: "store-id",
Name: "NewStore",
Location: "NewLocation",
}
Application
CreateStore()
1 Send in command with optional ProviderState customizations
2 The Application publishes a new domain event for in-process handling
Event Dispatcher
3 The Event Dispatcher publishes a new event for asynchronous delivery
4 The event is returned to the MessageHandler for verification
Event
Fake MessagePublisher
End-To-End test
API
Service
Test Parameters

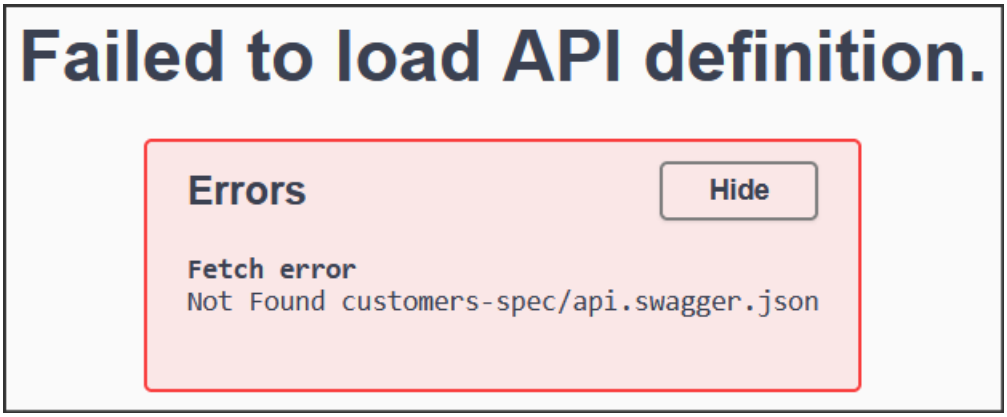

BDD
Refactor
Failing
R
F
TDD
P
Passing

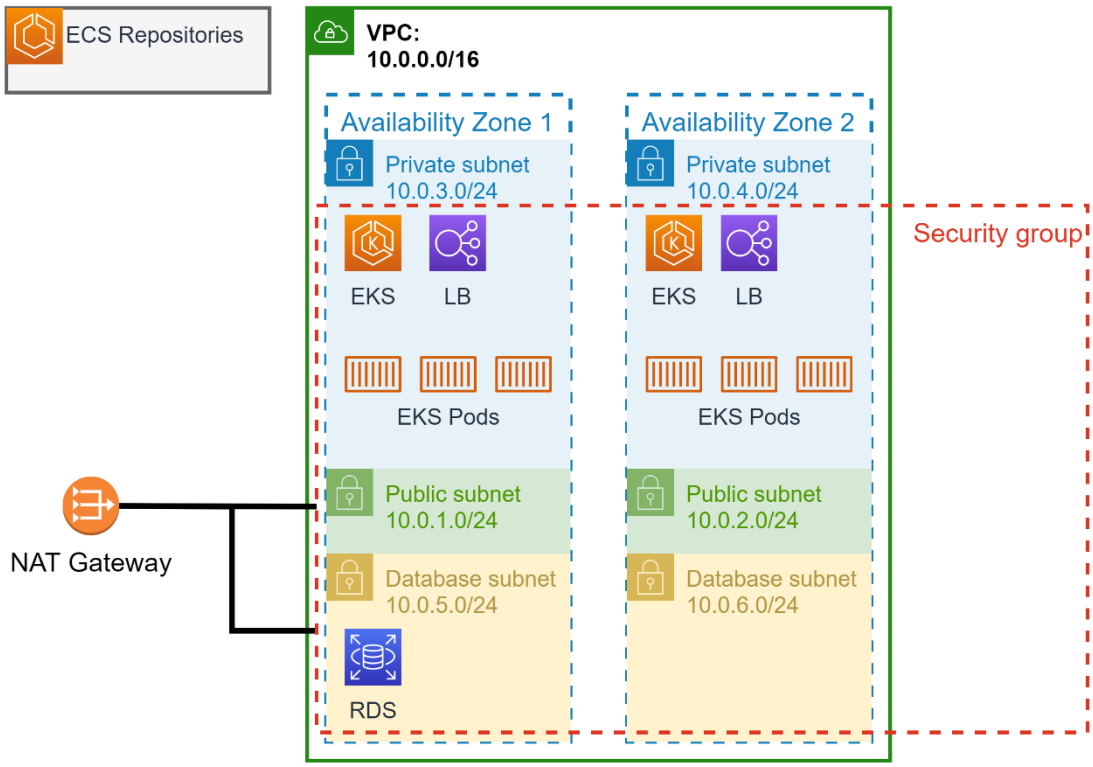
Chapter 11: Deploying applications to the cloud

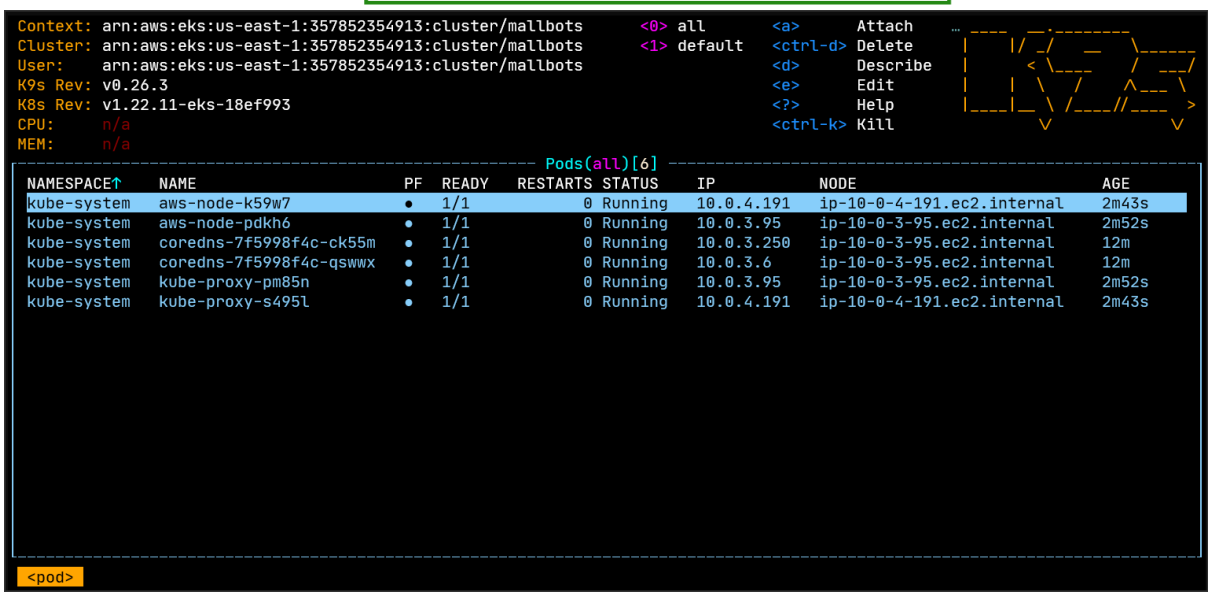




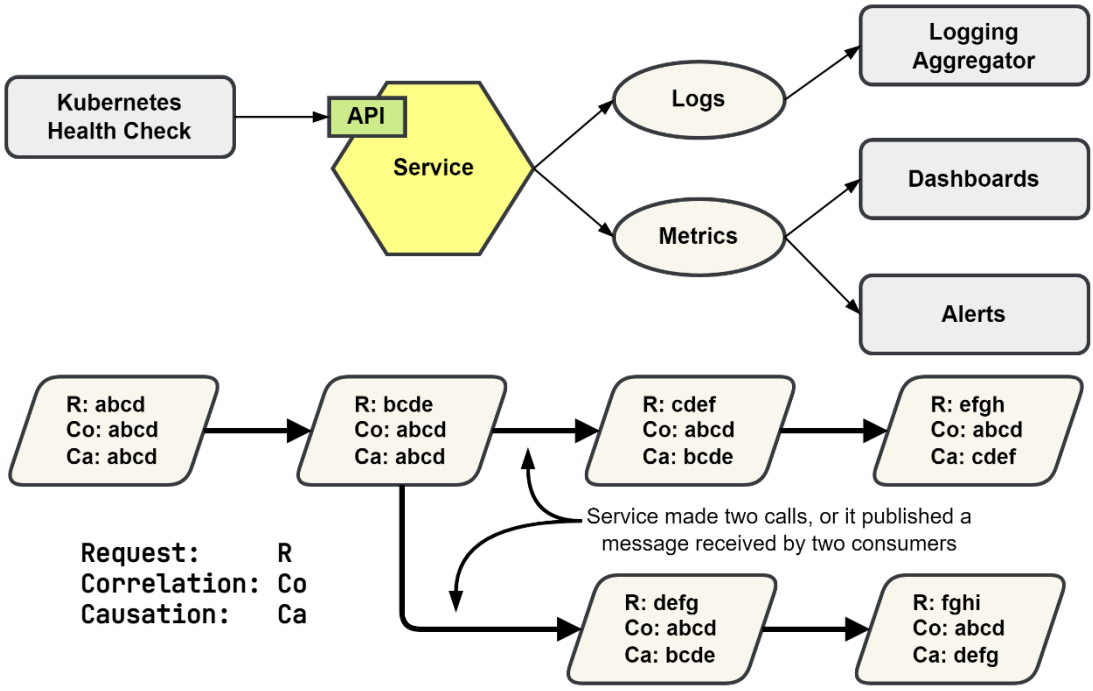


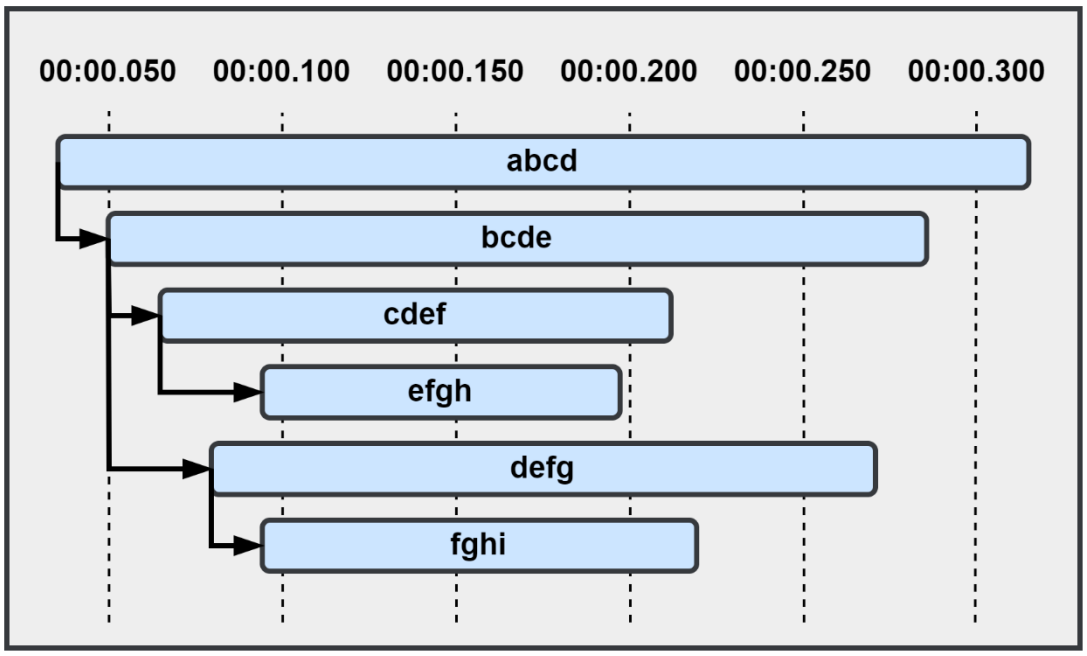

Failed to load API definition.
Errors
Hide
Fetch error
Not Found customers-spec/api.swagger.json

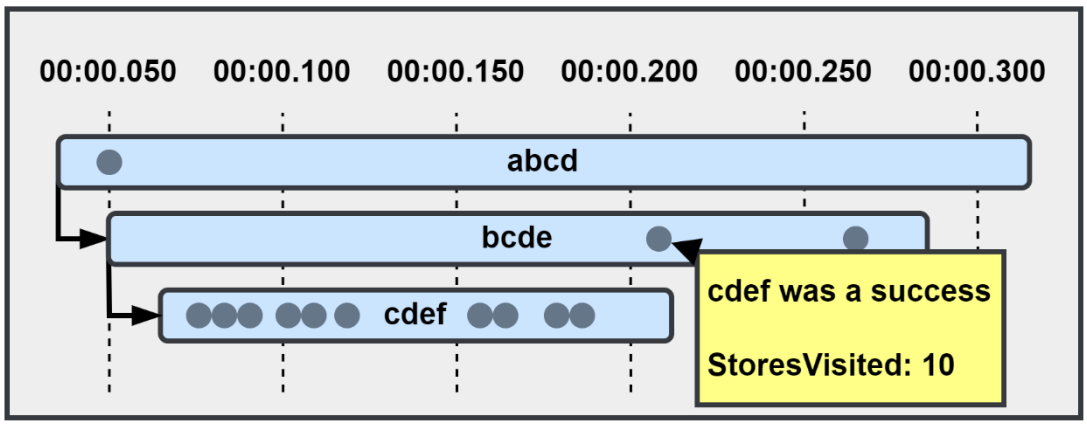

ECS Repositories
VPC:
10.0.0.0/16
Availability Zone 1
Private subnet
10.0.3.0/24
EKS
LB
EKS Pods
Public subnet
10.0.1.0/24
Database subnet
10.0.5.0/24
RDS
Availability Zone 2
Private subnet
10.0.4.0/24
EKS
LB
EKS Pods
Public subnet
10.0.2.0/24
Database subnet
10.0.6.0/24
Security group
NAT Gateway

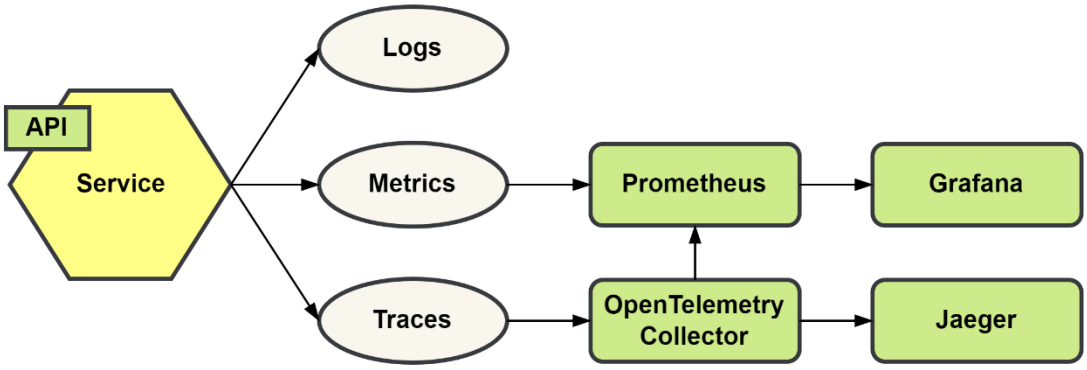

Context: arn:aws:eks:us-east-1:357852354913:cluster/mallbots
Cluster: arn:aws:eks:us-east-1:357852354913:cluster/mallbots
User: arn:aws:eks:us-east-1:357852354913:cluster/mallbots
K9s Rev: v0.26.3
K8s Rev: v1.22.11-eks-18ef993
CPU: n/a
MEM: n/a
<0> all
<1> default
<a> Attach
<ctrl-d> Delete
<d> Describe
<e> Edit
<?> Help
<ctrl-k> Kill
Pods(all)[6]
NAMESPACE↑ NAME PF READY RESTARTS STATUS IP NODE AGE
kube-system aws-node-k59w7 • 1/1 0 Running 10.0.4.191 ip-10-0-4-191.ec2.internal 2m43s
kube-system aws-node-pdkh6 • 1/1 0 Running 10.0.3.95 ip-10-0-3-95.ec2.internal 2m52s
kube-system coredns-7f5998f4c-ck55m • 1/1 0 Running 10.0.3.250 ip-10-0-3-95.ec2.internal 12m
kube-system coredns-7f5998f4c-qswwx • 1/1 0 Running 10.0.3.6 ip-10-0-3-95.ec2.internal 12m
kube-system kube-proxy-pm85n • 1/1 0 Running 10.0.3.95 ip-10-0-3-95.ec2.internal 2m52s
kube-system kube-proxy-s495l • 1/1 0 Running 10.0.4.191 ip-10-0-4-191.ec2.internal 2m43s
<pod>

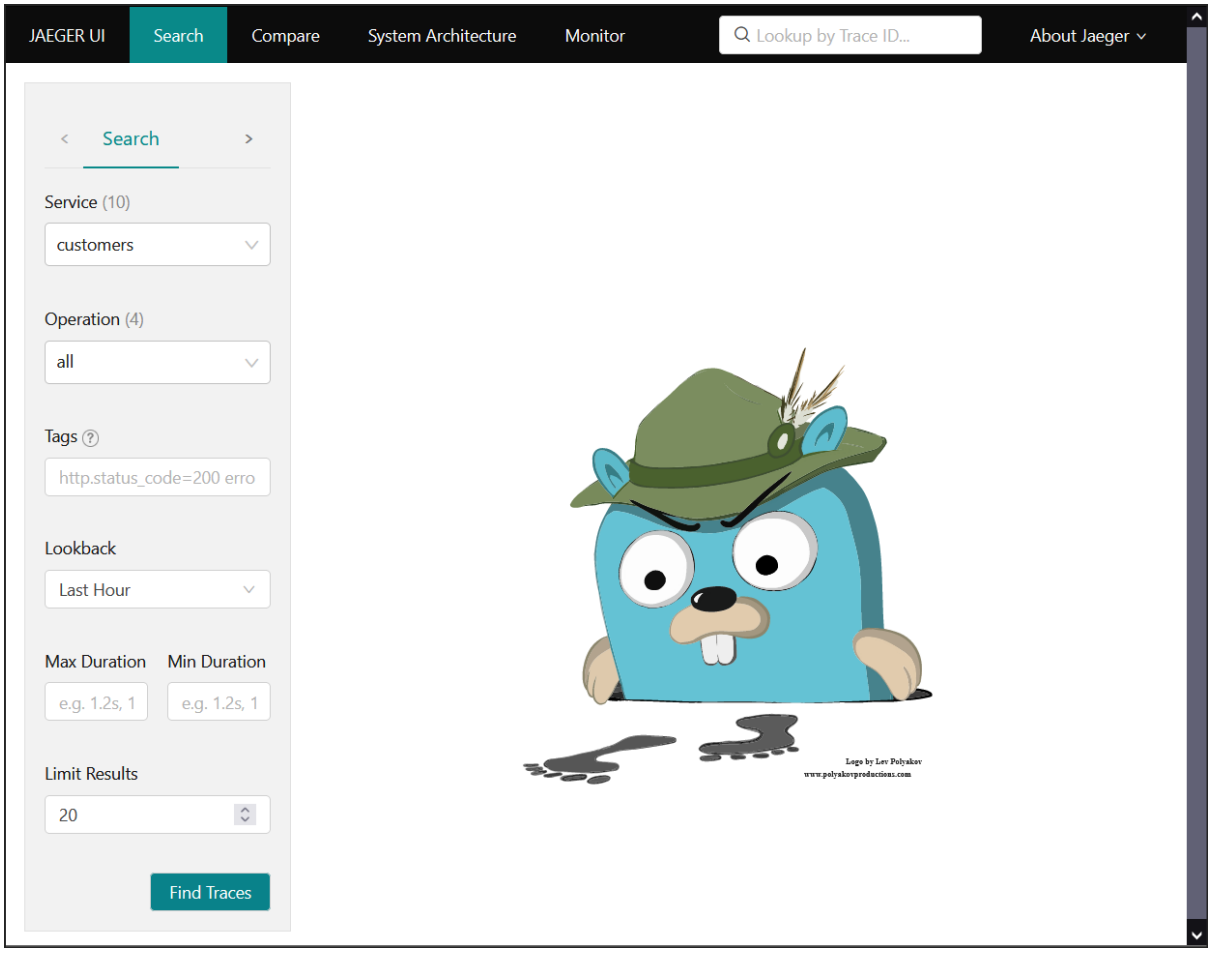
Chapter 12: Monitoring and Observability

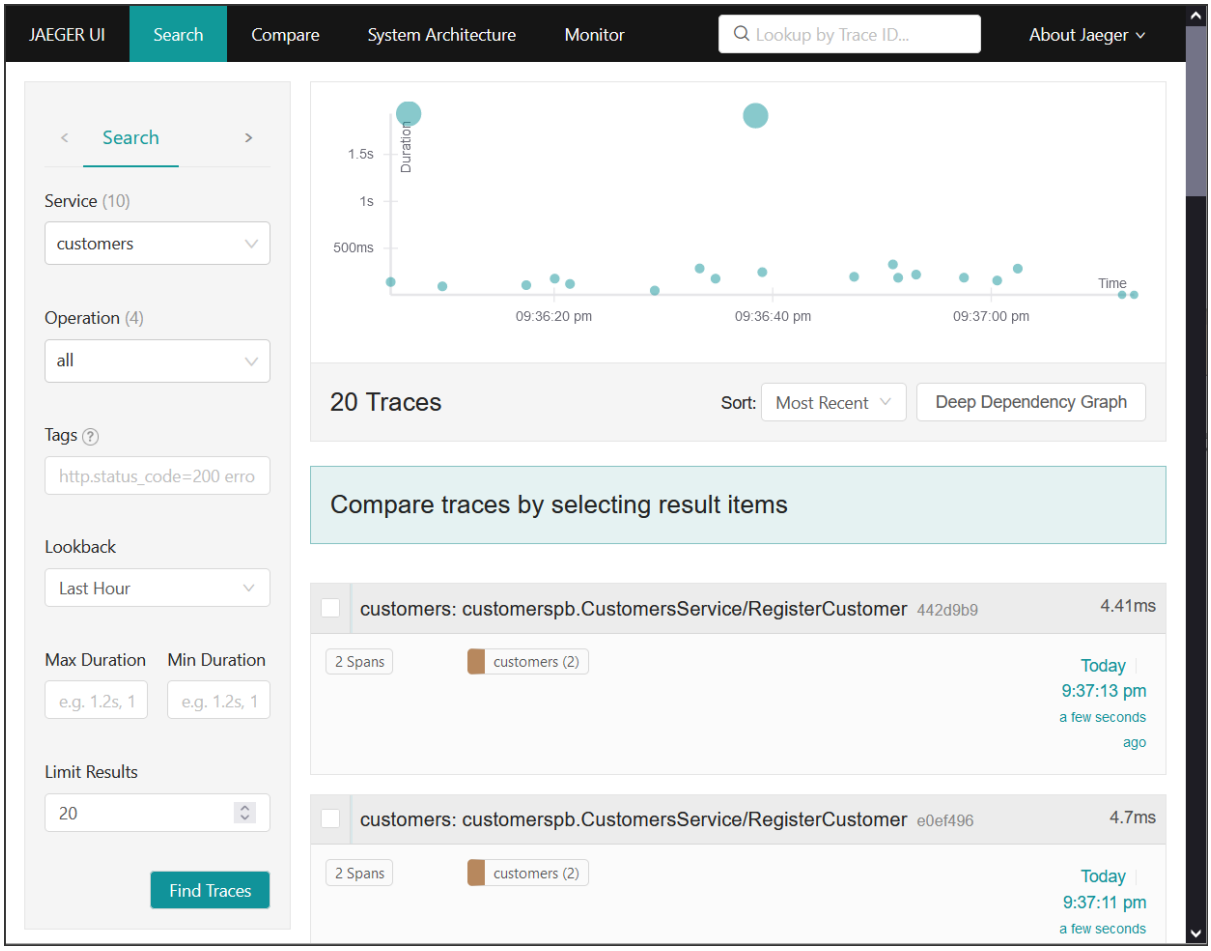


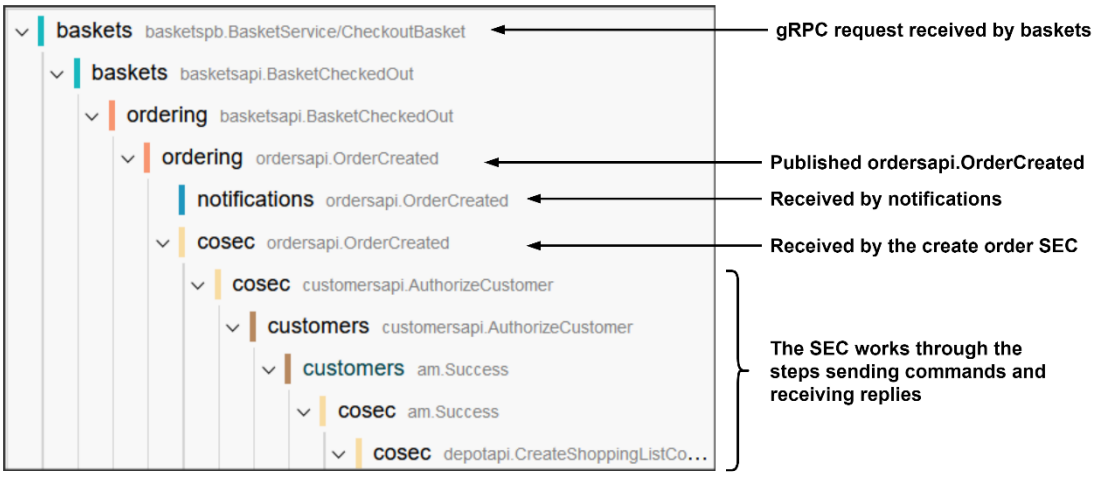


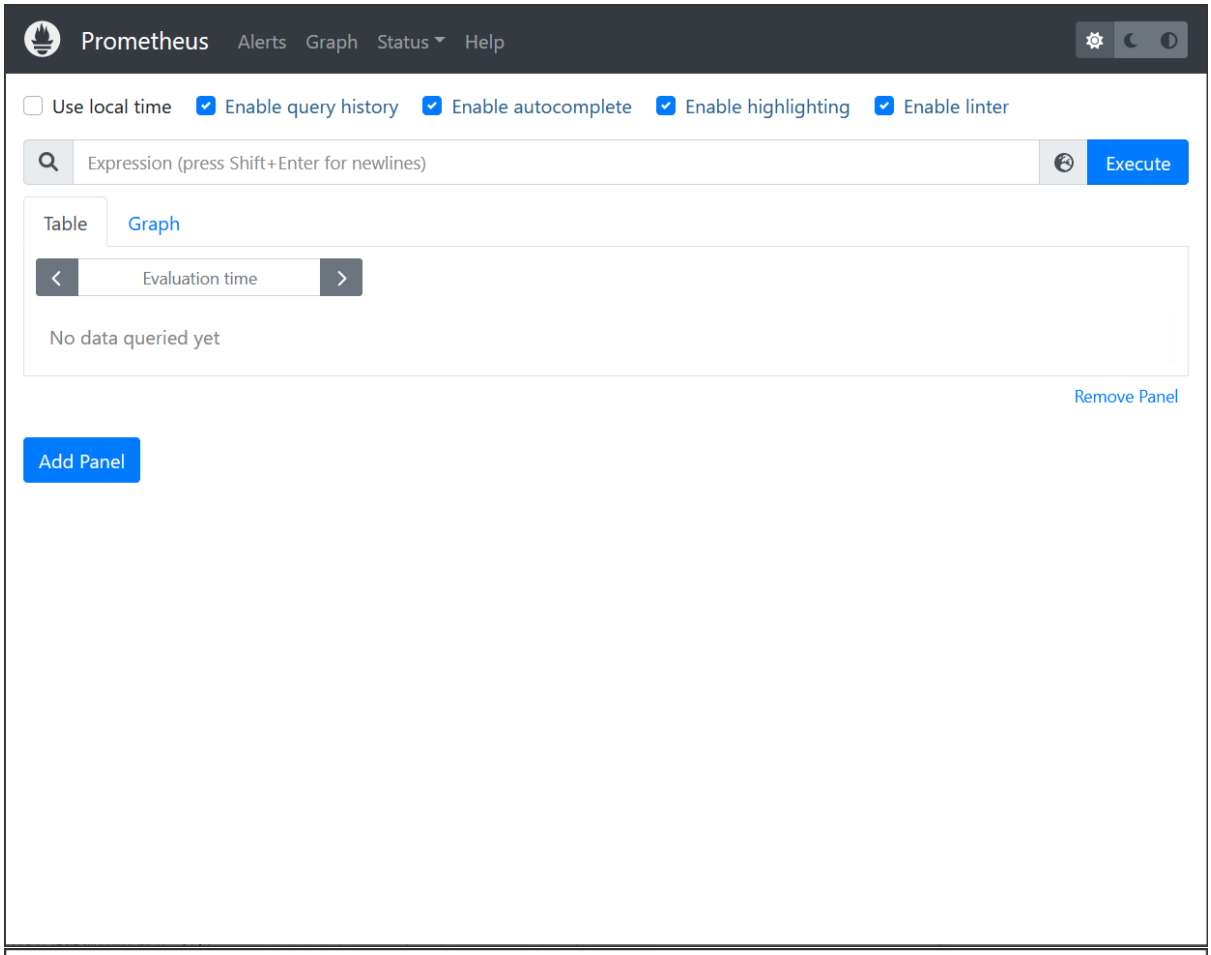

00:00.050
00:00.100
00:00.150
00:00.200
00:00.250
00:00.300
abcd
bcde
cdef
cdef was a success
StoresVisited: 10

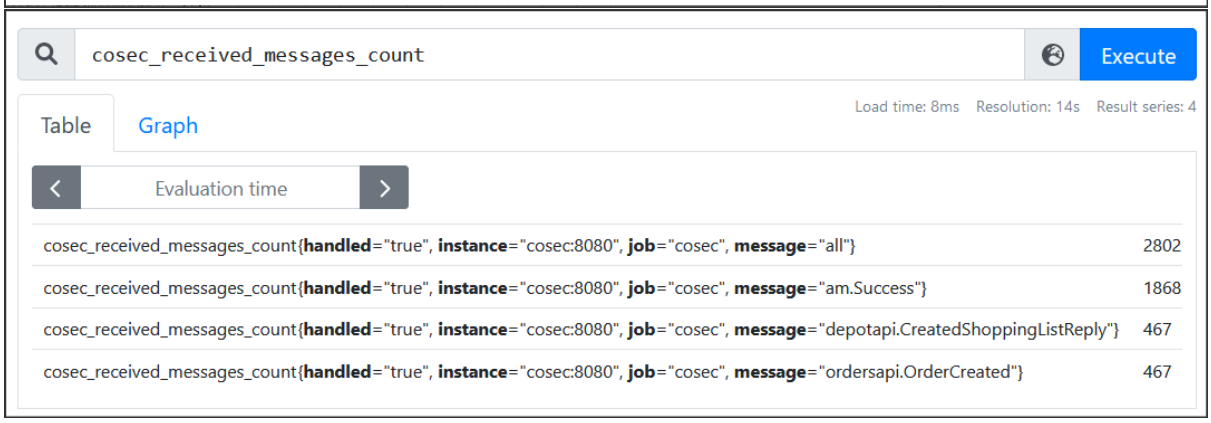

API
Service
Logs
Metrics
Traces
Prometheus
Grafana
OpenTelemetry Collector
Jaeger

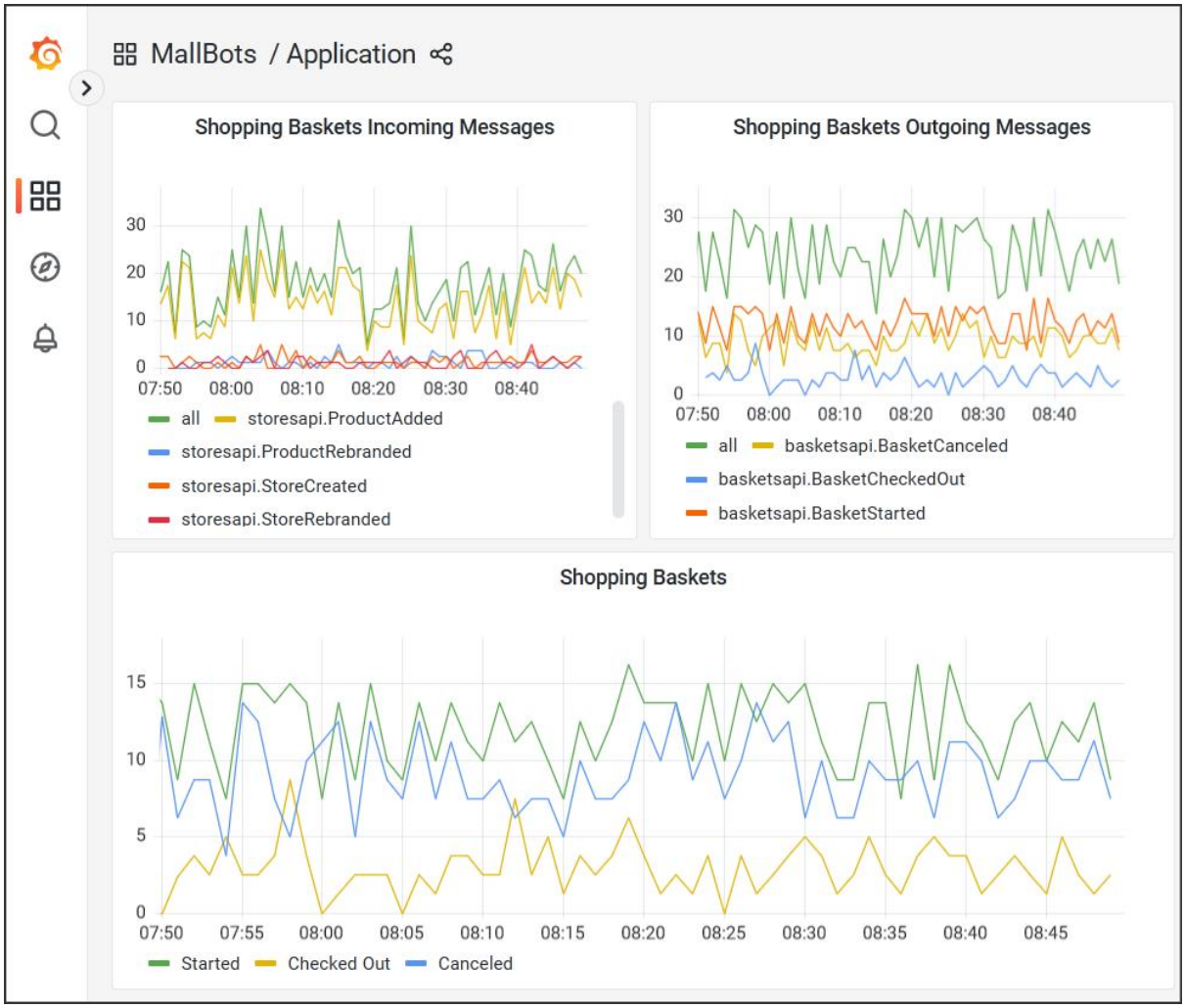

JAEGER UI
Search
Compare
System Architecture
Monitor
Lookup by Trace ID...
About Jaeger
Search
Service (10)
customers
Operation (4)
all
Tags
http.status_code=200 erro
Lookback
Last Hour
Max Duration
Min Duration
e.g. 1.2s, 1
e.g. 1.2s, 1
Limit Results
20
Find Traces
Logo by Lev Polyakov
www.polyakovproductions.com


JAEGER UI
Search
Compare
System Architecture
Monitor
Lookup by Trace ID...
About Jaeger
Search
Service (10)
customers
Operation (4)
all
Tags
http.status_code=200 erro
Lookback
Last Hour
Max Duration
Min Duration
e.g. 1.2s, 1
e.g. 1.2s, 1
Limit Results
20
Find Traces
Duration
1.5s
1s
500ms
Time
09:36:20 pm
09:36:40 pm
09:37:00 pm
20 Traces
Sort:
Most Recent
Deep Dependency Graph
Compare traces by selecting result items
customers: customerspb.CustomersService/RegisterCustomer 442d9b9
4.41ms
2 Spans
customers (2)
Today
9:37:13 pm
a few seconds ago
customers: customerspb.CustomersService/RegisterCustomer e0ef496
4.7ms
2 Spans
customers (2)
Today
9:37:11 pm
a few seconds


baskets basketspb.BasketService/CheckoutBasket
baskets basketsapi.BasketCheckedOut
ordering basketsapi.BasketCheckedOut
ordering ordersapi.OrderCreated
notifications ordersapi.OrderCreated
cosec ordersapi.OrderCreated
cosec customersapi.AuthorizeCustomer
customers customersapi.AuthorizeCustomer
customers am.Success
cosec am.Success
cosec depotapi.CreateShoppingListCo...
gRPC request received by baskets
Published ordersapi.OrderCreated
Received by notifications
Received by the create order SEC
The SEC works through the steps sending commands and receiving replies


Prometheus
Alerts
Graph
Status
Help
Use local time
Enable query history
Enable autocomplete
Enable highlighting
Enable linter
Expression (press Shift+Enter for newlines)
Execute
Table
Graph
Evaluation time
No data queried yet
Remove Panel
Add Panel


cosec_received_messages_count
Execute
Load time: 8ms
Resolution: 14s
Result series: 4
Table
Graph
Evaluation time
cosec_received_messages_count{handled="true", instance="cosec:8080", job="cosec", message="all"} 2802
cosec_received_messages_count{handled="true", instance="cosec:8080", job="cosec", message="am.Success"} 1868
cosec_received_messages_count{handled="true", instance="cosec:8080", job="cosec", message="depotapi.CreatedShoppingListReply"} 467
cosec_received_messages_count{handled="true", instance="cosec:8080", job="cosec", message="ordersapi.OrderCreated"} 467


MallBots / Application
Shopping Baskets Incoming Messages
30
20
10
0
07:50
08:00
08:10
08:20
08:30
08:40
all
storesapi.ProductAdded
storesapi.ProductRebranded
storesapi.StoreCreated
storesapi.StoreRebranded
Shopping Baskets Outgoing Messages
30
20
10
0
07:50
08:00
08:10
08:20
08:30
08:40
all
basketsapi.BasketCanceled
basketsapi.BasketCheckedOut
basketsapi.BasketStarted
Shopping Baskets
15
10
5
0
07:50
07:55
08:00
08:05
08:10
08:15
08:20
08:25
08:30
08:35
08:40
08:45
Started
Checked Out
Canceled