# Application Modeling

With Morphir

# Introduction

**Open-sourced by Morgan Stanley:** Morphir

**Goals:**
Productivity
Efficiency
Risk

**Stakes:**
Costs
Reputation
Competitivity

**Desired Outcome:**
New ways to develop
Community participation

# Why Morphir?

**Value:**

- Development automation
- Effortless compliance
- Bug free guarantees

**Who:**

- App developers
- Infrastructure engineers
- Business users & analysts

**Learn More:**

- Morphir open-source

# Common Impediments

- Developers
  - Most development effort on non-business code.
  - Keeping up-to-date is a costly burden.
- Infrastructure engineers
  - Lack of consistency prevents holistic optimization & automation
- Business users
  - Disconnect between business and technology
  - Lack of transparency

# Example: Developer Effort

## The most value

- Turn business concepts into computer concepts
- Make it run

## The most effort

- Code to frameworks & libraries
- Conform to regulations (i.e., Policy 2.0)
- Ensure the code does what it’s supposed to
- Keep documents up-to-date
- Hygiene & keeping tech up-to-date
- Follow blueprints and best practices
- Monitoring
- Security & vulnerability remediation
- Supportability
- Testing
- Audit
- Plug into firm infrastructure (i.e., API Gateway)
- Evolution & future readiness (i.e., cloud)
- Telemetry, metrics, and observability (i.e., Backtrace)

# What if…

## Let developers focus on this:

- **Turn business concepts into computer concepts**
- **Make it run**

## Automate this:

- Code to frameworks & libraries
- Conform to regulations (i.e., Policy 2.0)
- Ensure the code does what it’s supposed to
- Keep documents up-to-date
- Hygiene & keeping tech up-to-date
- Follow blueprints and best practices
- Monitoring
- Security & vulnerability remediation
- Supportability
- Testing
- Audit
- Plug into firm infrastructure (i.e., API Gateway)
- Evolution & future readiness (i.e., cloud)
- Telemetry, metrics, and observability (i.e., Backtrace)

# Demo

Morphir + Dapr

# Optimize the Development Pipeline

Manual

Semi Automatic

Automatic

Development now


+ developer automation

Development with Morphir + Dapr

# The Tools

**Dapr**

- Framework:
  - Infrastructure abstraction
  - Cloud and on-prem
  - Out-of-the box features
- Open-sourced by Microsoft

**Morphir**

- Application modeling
- Development automation
- Knowledge tools
- Correctness
- Open-sourced by Morgan Stanley

# Modeling services

- Patterns enable automation
- Modeling defines those patterns
- Demo pattern = service
  - Take requests to perform some action
  - Produce a result
  - Manage some state
- Simple Books & Records service for managing deals

# Books & Records Service: **Request**

**Specifications:**

1. As a client, I want to instruct to record a new deal for a quantity of a product at a specified price…
2. As a client, I want to instruct to close an existing deal…

```
{- These define the requests that can be made of this service -}
type DealCmd
    = OpenDeal ID ProductID Price Quantity
    | CloseDeal ID
```

```
1  module Morphir.Dapr.Input.Example exposing (..)
2
3  import Morphir.SDK.StatefulApp exposing (StatefulApp)
4
5  {- Type aliases for modeling in the language of the business -}
6  type alias ID = String
7  type alias ProductID = String
8  type alias Price = Decimal
9  type alias Quantity = Int
10
11 {- These define the requests that can be made of this service -}
12 type DealCmd
13     = OpenDeal ID ProductID Price Quantity
14     | CloseDeal ID
15
```

# Books & Records Service: **Result**

**Specifications:**

1. As a client, I want to know how my instructions were processed…
2. As an interested party, I want to be notified of any deal activity…
3. As a regulator, I want to ensure there's a temporal system of record for all deal activity…

```
{- These define the responses that would result from requests -}
type DealEvent
    = DealOpened ID ProductID Price Quantity
    | DealClosed ID
    | InvalidQuantity ID Quantity
    | InvalidPrice ID Price
    | DuplicateDeal ID
    | DealNotFound ID
```

```
module Morphir.Dapr.Input.Example exposing (..)

import Morphir.SDK.StatefulApp exposing (StatefulApp)

{- Type aliases for modeling in the language of the business -}
type alias ID = String
type alias ProductID = String
type alias Price = Decimal
type alias Quantity = Int

{- These define the requests that can be made of this service -}
type DealCmd
    = OpenDeal ID ProductID Price Quantity
    | CloseDeal ID

{- These define the responses that would result from requests -}
type DealEvent
    = DealOpened ID ProductID Price Quantity
    | DealClosed ID
    | InvalidQuantity ID Quantity
    | InvalidPrice ID Price
    | DuplicateDeal ID
    | DealNotFound ID
```

# Books & Records Service:
# State

**Specifications:**

1. As a client, I want the state of all of my deals to be accessible at any time…
2. As a system owner, I want to ensure that deal state is resilient to process restarts…

```
{- Identifies a structure that can be associated to a persistance entity -}
type alias Deal =
    { id : ID
    , product : ProductID
    , price : Price
    , quantity : Quantity
    }
```

```
module Morphir.Dapr.Input.Example exposing (..)

import Morphir.SDK.StatefulApp exposing (StatefulApp)

{- Type aliases for modeling in the language of the business -}
type alias ID = String
type alias ProductID = String
type alias Price = Decimal
type alias Quantity = Int

{- These define the requests that can be made of this service -}
type DealCmd
    = OpenDeal ID ProductID Price Quantity
    | CloseDeal ID

{- These define the responses that would result from requests -}
type DealEvent
    = DealOpened ID ProductID Price Quantity
    | DealClosed ID
    | InvalidQuantity ID Quantity
    | InvalidPrice ID Price
    | DuplicateDeal ID
    | DealNotFound ID

{- Identifies a structure that can be associated to a persistance entity -}
type alias Deal =
    { id : ID
    , product : ProductID
    , price : Price
    , quantity : Quantity
    }
```

# Books & Records Service:
# Processing logic

**Specifications:**

1. As a business owner, I want an open deal request to be accepted only if that deal doesn't already exist, the price is free or more, and the quantity is 1 or more.
2. As a business owner, I want a close deal request to be accepted only if the deal is currently open.

```elm
{- Defines the business logic of this app. That is whether or not to accept a request to open or close a deal. -}
logic : ID -> Maybe Deal -> DealCmd -> ( ID, Maybe Deal, DealEvent )
logic dealId deal dealCmd =
    -- Act accordingly based on whether the deal already exists.
    case deal of
        Just d ->
            case dealCmd of
                CloseDeal _ ->
                    ( dealId, Nothing, DealClosed dealId )

                OpenDeal _ _ _ _ ->
                    ( dealId, deal, DuplicateDeal dealId )

        Nothing ->
            case dealCmd of
                OpenDeal id productId price qty ->
                    if price < 0 then
                        ( dealId, deal, InvalidPrice id price )

                    else if qty < 0 then
                        ( dealId, deal, InvalidQuantity id qty )

                    else
                        ( dealId
                        , Deal id productId price qty |> Just
                        , DealOpened id productId price qty
                        )

                CloseDeal _ ->
                    ( dealId, deal, DealNotFound dealId )
```

```elm
module Morphir.Dapr.Input.Example exposing (..)

import Morphir.SDK.StatefulApp exposing (StatefulApp)

{- Type aliases for modeling in the language of the business -}
type alias ID = String
type alias ProductID = String
type alias Price = Decimal
type alias Quantity = Int

{- These define the requests that can be made of this service -}
type DealCmd
    = OpenDeal ID ProductID Price Quantity
    | CloseDeal ID

{- These define the responses that would result from requests -}
type DealEvent
    = DealOpened ID ProductID Price Quantity
    | DealClosed ID
    | InvalidQuantity ID Quantity
    | InvalidPrice ID Price
    | DuplicateDeal ID
    | DealNotFound ID

{- Identifies a structure that can be associated to a persistance entity -}
type alias Deal =
    { id : ID
    , product : ProductID
    , price : Price
    , quantity : Quantity
    }

{- Defines the business logic of this app. That is whether or not to accept a request to open or close a deal. -}
logic : ID -> Maybe Deal -> DealCmd -> ( ID, Maybe Deal, DealEvent )
logic dealId deal dealCmd =
    -- Act accordingly based on whether the deal already exists.
    case deal of
        Just d ->
            case dealCmd of
                CloseDeal _ ->
                    ( dealId, Nothing, DealClosed dealId )

                OpenDeal _ _ _ _ ->
                    ( dealId, deal, DuplicateDeal dealId )

        Nothing ->
            case dealCmd of
                OpenDeal id productId price qty ->
                    if price < 0 then
                        ( dealId, deal, InvalidPrice id price )

                    else if qty < 0 then
                        ( dealId, deal, InvalidQuantity id qty )

                    else
                        ( dealId
                        , Deal id productId price qty |> Just
                        , DealOpened id productId price qty
                        )

                CloseDeal _ ->
                    ( dealId, deal, DealNotFound dealId )

{- Defines that this is a stateful application that uses ID as the entity key (for possible partioning), -}
type alias App = StatefulApp ID DealCmd Deal DealEvent

app : App
app = StatefulApp logic
```

# What we automated

```elm
module Morphir.Dapr.Input.Example exposing (..)

import Morphir.SDK.StatefulApp exposing (StatefulApp)

{- Type aliases for modeling in the language of the business -}
type alias ID = String
type alias ProductID = String
type alias Price = Decimal
type alias Quantity = Int

{- These define the requests that can be made of this service -}
type DealCmd
    = OpenDeal ID ProductID Price Quantity
    | CloseDeal ID

{- These define the responses that would result from requests -}
type DealEvent
    = DealOpened ID ProductID Price Quantity
    | DealClosed ID
    | InvalidQuantity ID Quantity
    | InvalidPrice ID Price
    | DuplicateDeal ID
    | DealNotFound ID

{- Identifies a structure that can be associated to a persistance entity -}
type alias Deal =
    { id : ID
    , product : ProductID
    , price : Price
    , quantity : Quantity
    }

{- Defines the business logic of this app. That is whether or not to accept a request to open or close a deal
logic : ID -> Maybe Deal -> DealCmd -> ( ID, Maybe Deal, DealEvent )
logic dealId deal dealCmd =
    -- Act accordingly based on whether the deal already exists.
    case deal of
        Just d ->
            case dealCmd of
                CloseDeal _ ->
                    ( dealId, Nothing, DealClosed dealId )

                OpenDeal _ _ _ _ ->
                    ( dealId, deal, DuplicateDeal dealId )

        Nothing ->
            case dealCmd of
                OpenDeal id productId price qty ->
                    if price < 0 then
                        ( dealId, deal, InvalidPrice id price )

                    else if qty < 0 then
                        ( dealId, deal, InvalidQuantity id qty )

                    else
                        ( dealId
                        , Deal id productId price qty |> Just
                        , DealOpened id productId price qty
                        )

                CloseDeal _ ->
                    ( dealId, deal, DealNotFound dealId )

{- Defines that this is a stateful application that uses ID as the entity key (for possible partioning), -}
type alias App = StatefulApp ID DealCmd Deal DealEvent

app : App
app = StatefulApp logic
```

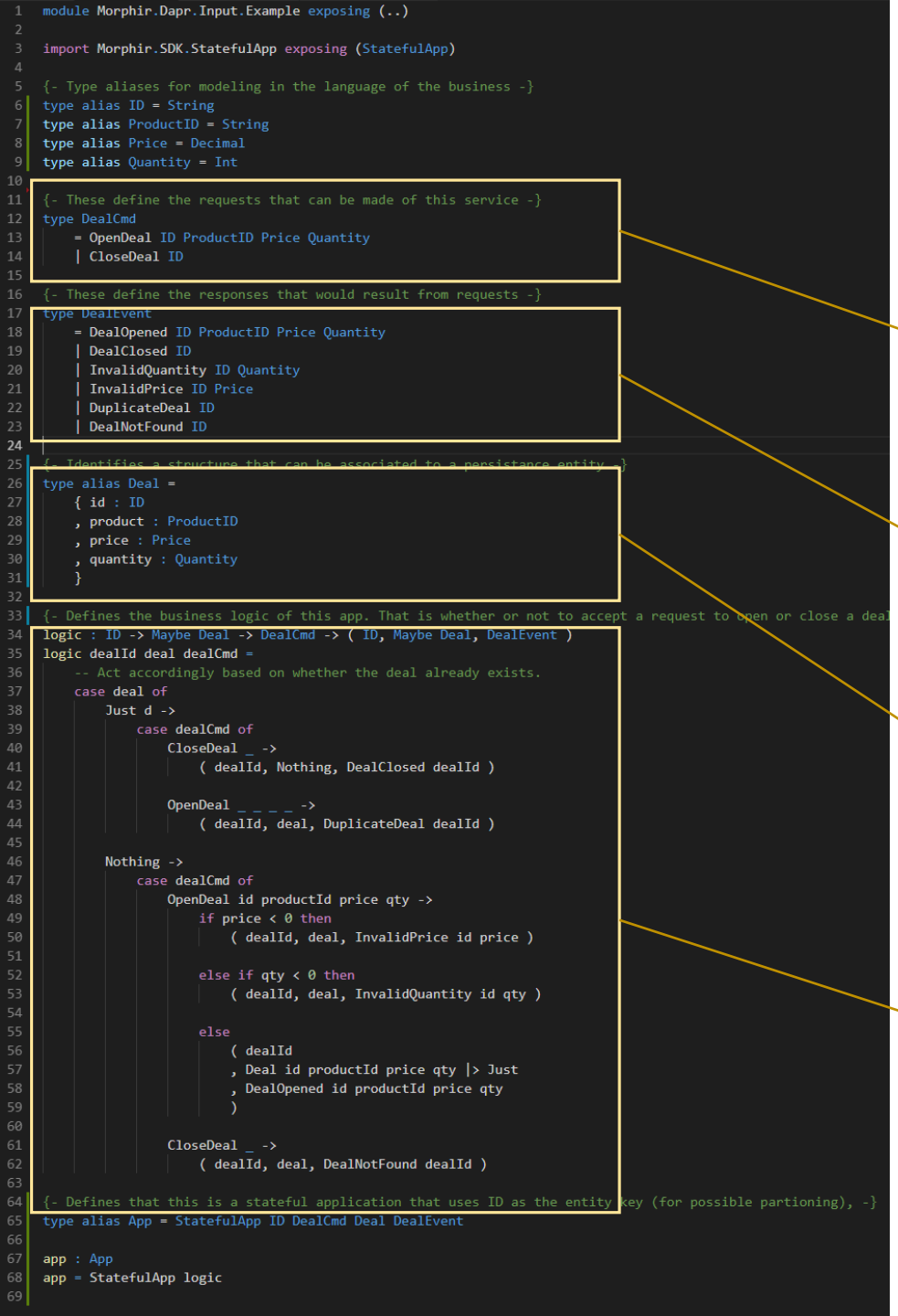
- Code
- JSON serialization

- Code
- JSON serialization
- Kafka Publish

- Code
- Persistence

- Code
- REST binding

# What else could we automate?



```
module Morphir.Dapr.Input.Example exposing (..)

import Morphir.SDK.StatefulApp exposing (StatefulApp)

{- Type aliases for modeling in the language of the business -}
type alias ID = String
type alias ProductID = String
type alias Price = Decimal
type alias Quantity = Int

{- These define the requests that can be made of this service -}
type DealCmd
    = OpenDeal ID ProductID Price Quantity
    | CloseDeal ID

{- These define the responses that would result from requests -}
type DealEvent
    = DealOpened ID ProductID Price Quantity
    | DealClosed ID
    | InvalidQuantity ID Quantity
    | InvalidPrice ID Price
    | DuplicateDeal ID
    | DealNotFound ID

{- Identifies a structure that can be associated to a persistance entity -}
type alias Deal =
    { id : ID
    , product : ProductID
    , price : Price
    , quantity : Quantity
    }

{- Defines the business logic of this app. That is whether or not to accept a request to open or close a deal
logic : ID -> Maybe Deal -> DealCmd -> ( ID, Maybe Deal, DealEvent )
logic dealId deal dealCmd =
    -- Act accordingly based on whether the deal already exists.
    case deal of
        Just d ->
            case dealCmd of
                CloseDeal _ ->
                    ( dealId, Nothing, DealClosed dealId )

                OpenDeal _ _ _ _ ->
                    ( dealId, deal, DuplicateDeal dealId )

        Nothing ->
            case dealCmd of
                OpenDeal id productId price qty ->
                    if price < 0 then
                        ( dealId, deal, InvalidPrice id price )

                    else if qty < 0 then
                        ( dealId, deal, InvalidQuantity id qty )

                    else
                        ( dealId
                        , Deal id productId price qty |> Just
                        , DealOpened id productId price qty
                        )

                CloseDeal _ ->
                    ( dealId, deal, DealNotFound dealId )

{- Defines that this is a stateful application that uses ID as the entity key (for possible partioning), -}
type alias App = StatefulApp ID DealCmd Deal DealEvent

app : App
app = StatefulApp logic
```

- Code
- JSON serialization

- Code
- JSON serialization
- Kafka Publish

- Code
- Persistence

- Code
- REST binding

- **Observability & telemetry**
- **Policy 2.0 Data dictionary & lineage**
- **Documentation & audit**
- **Service Discovery**
- **OpenAPI / API Gateway**
- **GraphQL**
- **Schema registry**
- **Cloud & security Blueprints**
- **Contract-Driven Development full verification**
- **Tests**
- **Hygiene & Vulnerability remediation**
- **Firm-wide Performance optimizations**
- **Cloud readiness**
- **Future platforms…**

# The Code(s)

**Morphir**

- Application Model
- Query

**Generated**

- Dapr
- Spring Boot

# Other patterns…

## Query & Aggregation

Morphir Model

```
module Company.Operations.BooksAndRecords exposing (..)

import Company.Operations.BooksAndRecords exposing (..)

positionByProduct deals =
    deals
        |> List.groupWhile (\a,b -> a.productId == b.productId)
        |> List.map
            (\key,items -> (key, items
                |> List.map (.quantity)
                |> List.sum
            )
```

SQL

```
SELECT product_id, sum (quantity)
FROM DEALS
GROUP BY product_id
```

Spark, Kafka Streams, Jet, Flink…

```
deals
    .groupBy (x => x.productId)
    .map ( (key,items) => (key, items.map(_.amount).sum))
```

```
deals
    .groupBy (x => x.productId)
    .map ( (key,items) => (key, items.map(_.amount).sum))
```

```
deals
    .groupBy (x => x.productId)
    .map ( (key,items) => (key, items.map(_.amount).sum))
```

# What’s Next?

**Community**

- By developers for developers

**Open-source**

- Morphir