CLOUD DAY 2020

29 OTTOBRE • #CLOUDDAY2020

CQRS/ES

ANDREA SALTARELLO
Founder UGIdotNET
CTO @ Managed Designs
Contract Professor @ Politecnico di Milano

Kudos

Microsoft

managed/designs

AWS User Group - Italy

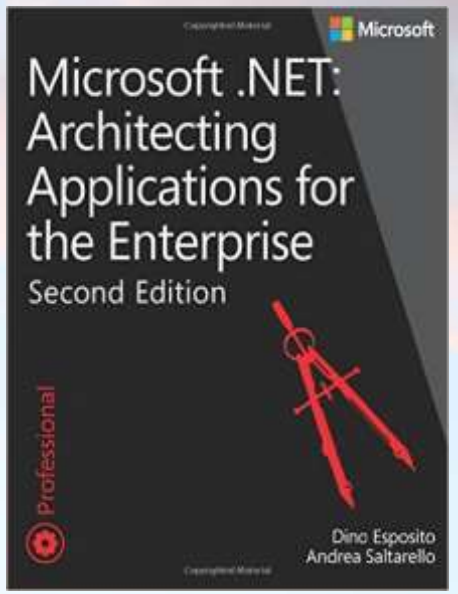
AZURE MEETUP MILANO

GDG Cloud Milano

There’s no silver bullet

Not the way to implement Event Sourcing,

still a working way to do it



Demo app available on Github (AGPL3): it’s a stripped down version of a product my company works on

The (Relational) Lord of the Rings

A bunch of fancy dressed blokes going for a jaunt

It really became clear to me in the last couple of years that we need a new building block and that is the Domain Event.

[Eric Evans]

An event is something that has happened in the past.

[Greg Young]

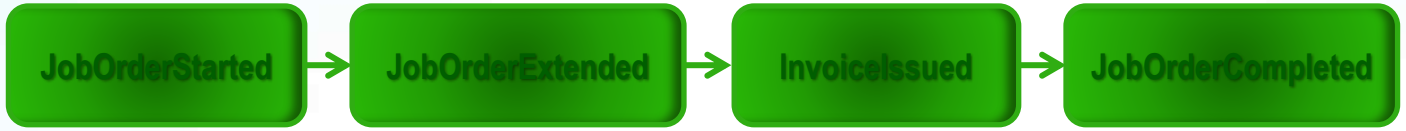
A domain event … captures the memory of something interesting which affects the domain

[Martin Fowler]

Event Sourcing in a nutshell

Instead of focusing on a system's last known state, we might note down every occurring event: this way, we would be able to (re)build the state the system was in at at at any point in time just replaying those events.

That way, we'd end up recording an ***event stream***



What's an event, anyway?

An (immutable) composition of:

- A (meaningful) name
- (Typed) Attributes

ProjectStarted
DateOfStart
ProjectId

ProjectCompleted
DateOfCompletion
ProjectId

InvoiceIssued
DateOfIssue
Customer
Price

ProjectRegistered
DateOfRegistration
DateOfEstimatedCompletion
ProjectId
CustomerId
Price

demo

Event stream

Event Stream vs. «My application»

Still, my users are more interested in knowing a job order's balance or whether an invoice has been paid. (cit.)

That is, we need a way to produce an entity state

Event Sourcing <3 DDD

DDD's Aggregates provide a convenient way to encapsulate event management

Aggregate: *A collection of objects that are bound together by a root entity, otherwise known as an aggregate root. The aggregate root guarantees the consistency of changes being made within the aggregate.* [Wikipedia]

An aggregate is responsible for:

- encapsulating business logic pertaining to an "entity"
- generating events to have them available for saving
- replaying events in order to rebuild a specific state

Aggregates vs. Events vs. Repos

```
var aggr = repository.GetById<TAggr>(id);          //Triggers [time travelling] event replay

aggr.DoSomething();                                //Biz logic + events

repository.Save(aggr);                             //Updates the event stream
```

Repository: *Mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects.* [DDD]

demo

Time travelling

Event Stream vs. «My application»

Still², my users are more interested in knowing a job order's balance or whether an invoice has been paid. Quickly.

Ways to achieve that:

- Snapshots can help
- CQRS to the rescue: let's have a database storing the usual «last known system state» and use it as a *read model*

demo

Read Model

Enter CQRS

Acronym for **C**ommand **Q**uery **R**esponsibility **S**egregation

Basically, an ad hoc application stack for either “writing” or reading:

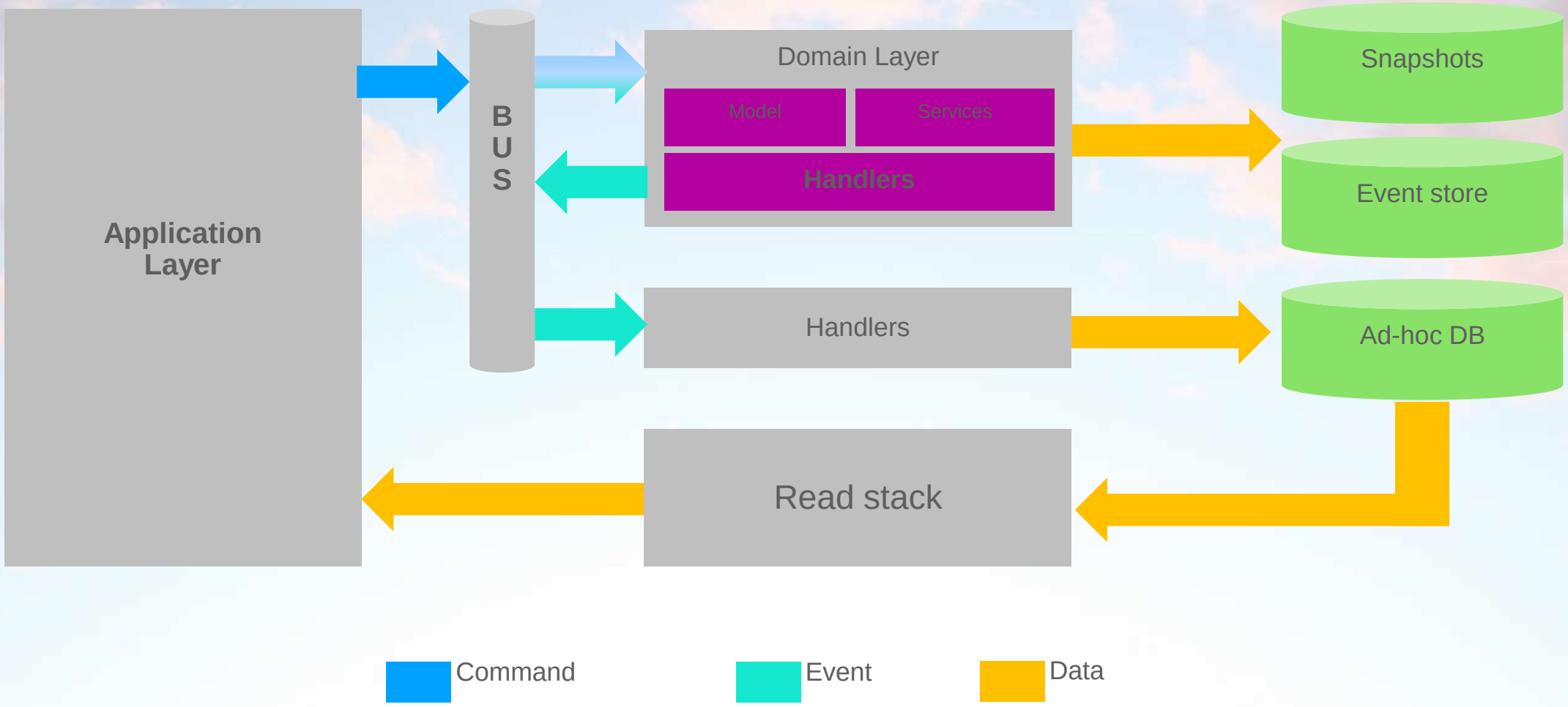
- “Command” stack produces events and snapshots
- “Query” stack reads from eventually consistent, reading purposes optimized database(s)

CQRS/ES' flow

1. Application sends a command to the system
2. Command execution might alter the system's state and then raise events to state success/failure
3. Events are notified to interested subscribers (a.k.a. handlers), such as:
    - Workflow managers (a.k.a. «Sagas») which could execute more commands
    - Denormalisers, which will update the read model database

Note: command/event dispatch/execution will usually be managed by a Mediator («bus»)

CQRS/ES at a glance



Assert.AreNotEqual(readModel, RDBMS);

CLOUD DAY 2020

29 OTTOBRE • #CLOUDDAY2020

demo

Return of the Read Model

CLOUD DAY 2020

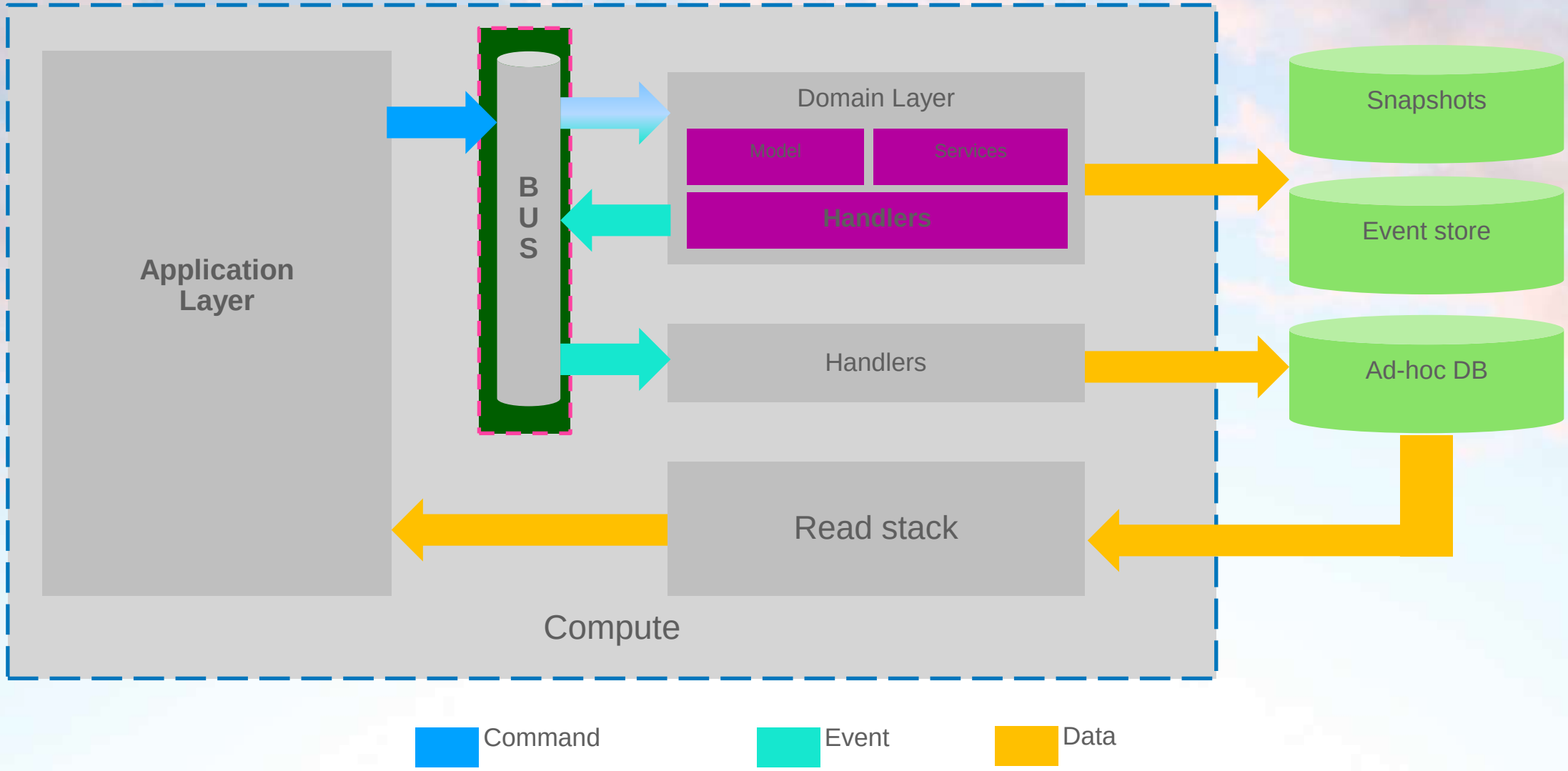
29 OTTOBRE • #CLOUDDAY2020

Deploy

Event Store: a few options

CosmosDB/DocumentDB	MongoDB Atlas	VM
PROS: • Multiple APIs (e.g.: MongoDB, Cassandra, Graph, ...) • PaaS • Predictable performance • Scalability/Georedundancy	PROS: • Vendor's official offering • PaaS • Large ecosystem	PROS: • Unparalleled configuration options
CONS: • Perf-wise, lots of data needed to harness it	CONS: • Pricey	CONS: • Management

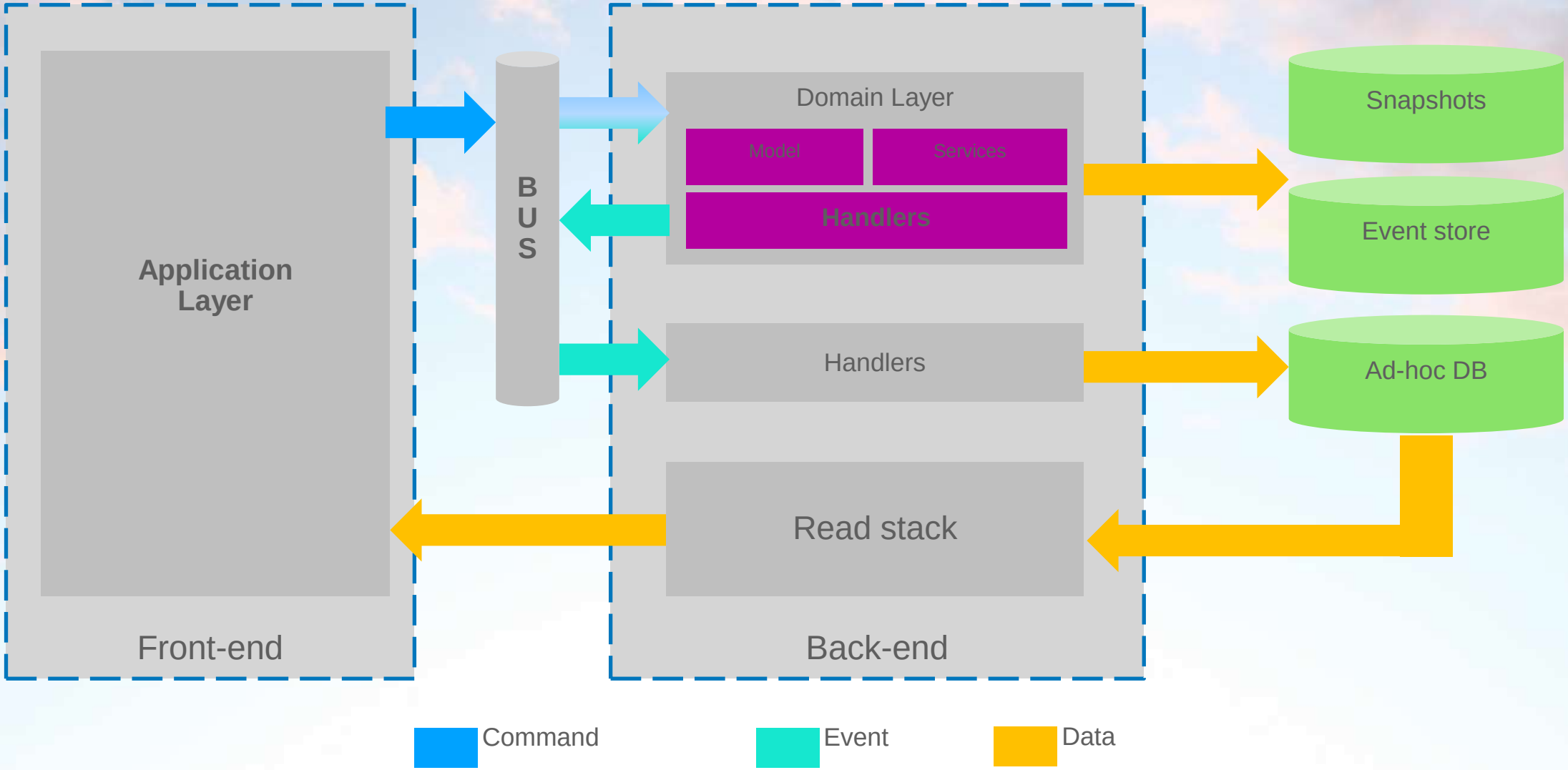
Cloud <3 CQRS: full stack



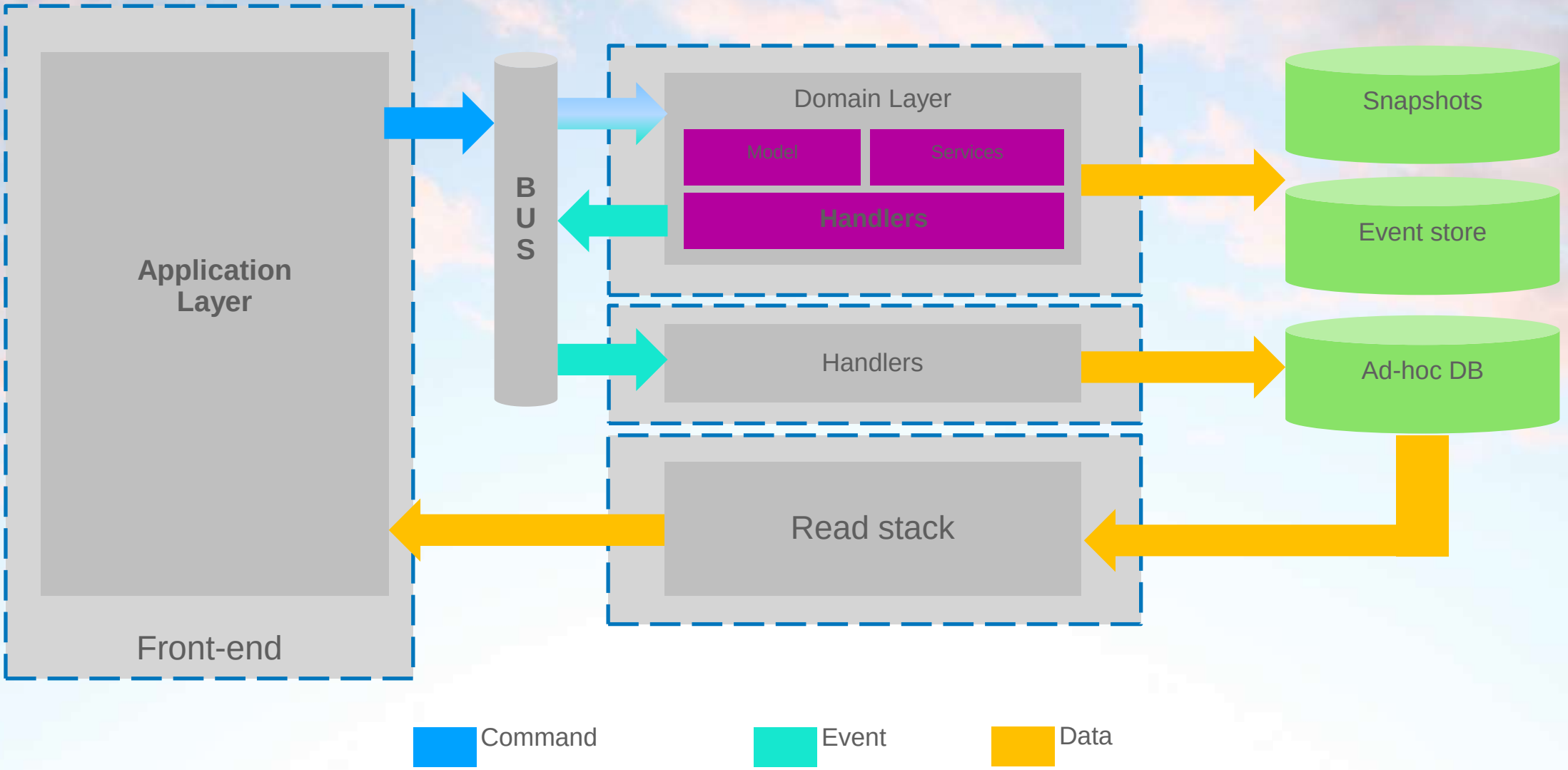
Full stack deploy

- Bus: ASB/SQS
- App Service/Beanstalk
- Docker
- VM

Cloud <3 CQRS: n-tiered



Cloud <3 CQRS: n-tiered



N-tiered deploy

Front-end	Back-end	Bus
• App Service/Beanstalk • Azure Functions/Lambda • Docker • VM	• App Service/Beanstalk • Azure Functions/Lambda • Docker • Service Fabric • VM	• ASB/SQS

N-tiered vs. Full stack

A fine-grained deploy allows each “component” to be:

- Evolved
- Deployed
- Configured (e.g.: scaled)

independently, thus having each one only consuming the resources it really needs but having more moving parts to manage as well

CLOUD DAY 2020

29 OTTOBRE • #CLOUDDAY2020

Costs

Merp facts

Storing 14 years of real-world accounting information sums up to:

- ~32,5K events
- An average event size of 560 *bytes*
- Using ~14Mb of disk storage for the event stream (data + indexes)
- Using ~350Mb of blob storage for documents (e.g.: PDF files)

Merp a la carte

To have Merp running we need:

- Some Compute to host web apps
- A RDMBS for read models
- A NoSQL DB for event stores
- A bus

monthly cost spans from ~300 to ~500 EUR/m depending on perf and HA

Thank You!

Demo source code available at https://github.com/mastreeno/merp

Contact me at

https://www.facebook.com/saltarelloandrea

https://github.com/andysal

https://linkedin.com/in/andysal

https://twitter.com/andysal74