RED HAT
SUMMIT

LOWERING THE RISK OF MONOLITH TO MICROSERVICES

One organization’s journey to microservices

@christianposta
Chief Architect, Cloud applications

8 May, 2018

Christian Posta

Chief Architect, cloud application development

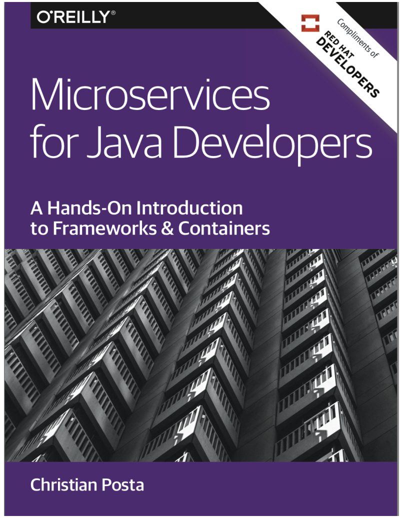
Twitter: @christianposta

Blog: http://blog.christianposta.com

Email: christian@redhat.com

Slides: http://slideshare.net/ceposta

- Author “Microservices for Java developers”, “Introducing Istio Service Mesh”, and other
- Committer/contributor to open-source projects
- Blogger, speaker, writer



O'REILLY®

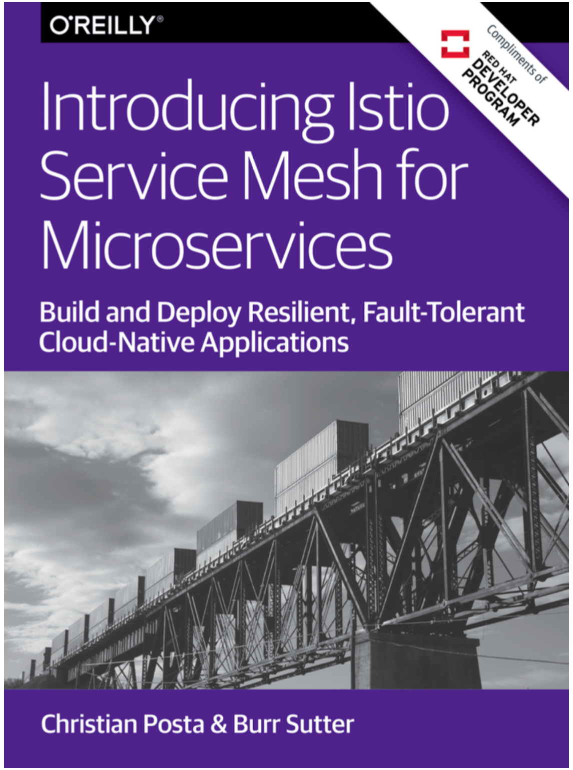
Compliments of RED HAT DEVELOPERS

Microservices for Java Developers

A Hands-On Introduction to Frameworks & Containers

Christian Posta

http://bit.ly/istio-book



O'REILLY®

Compliments of RED HAT DEVELOPER PROGRAM

Introducing Istio Service Mesh for Microservices

Build and Deploy Resilient, Fault-Tolerant Cloud-Native Applications

Christian Posta & Burr Sutter

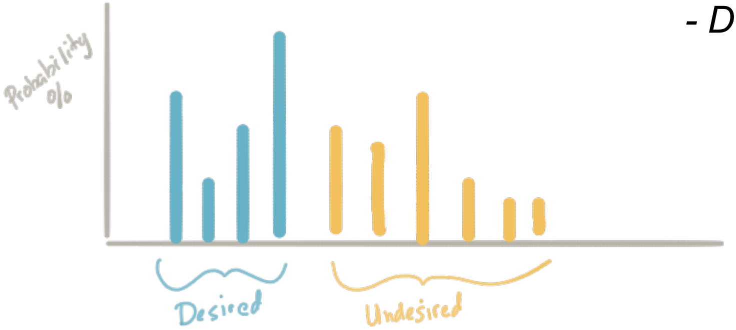
LOW RISK
MONOLITH
MICROSERVICES

Low Risk

“The existence of more than one possibility. The “true” outcome/state/result/value is not know”

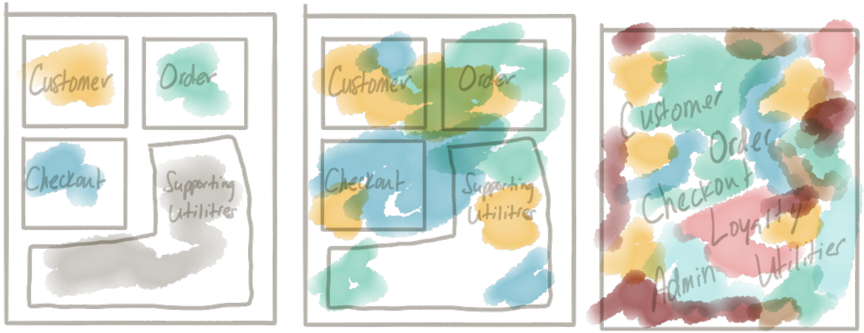
“A state of uncertainty where some of the possibilities involve a loss, catastrophe, or other undesirable outcome”

- Douglas Hubbard



Monolith

An existing large application developed over the course of many years by different teams that provides proven business value. Its structure has eroded insofar it has become very difficult to update and maintain.



Microservice

A highly distracting word that serves to confuse developers, architects, and IT leaders into believing that we can actually have a utopian application architecture.

Microservices

~~A highly distracting word that serves to confuse developers, architects, and IT leaders into believing that we can actually have a utopian application architecture.~~

An architecture *optimization* that treats the modules of an application as independently owned and deployed services for the purposes of increasing an organization’s velocity

> “We can now assert with confidence that high IT performance correlates with strong business performance, helping to boost productivity, profitability and market share.”

https://puppet.com/resources/whitepaper/2014-state-devops-report

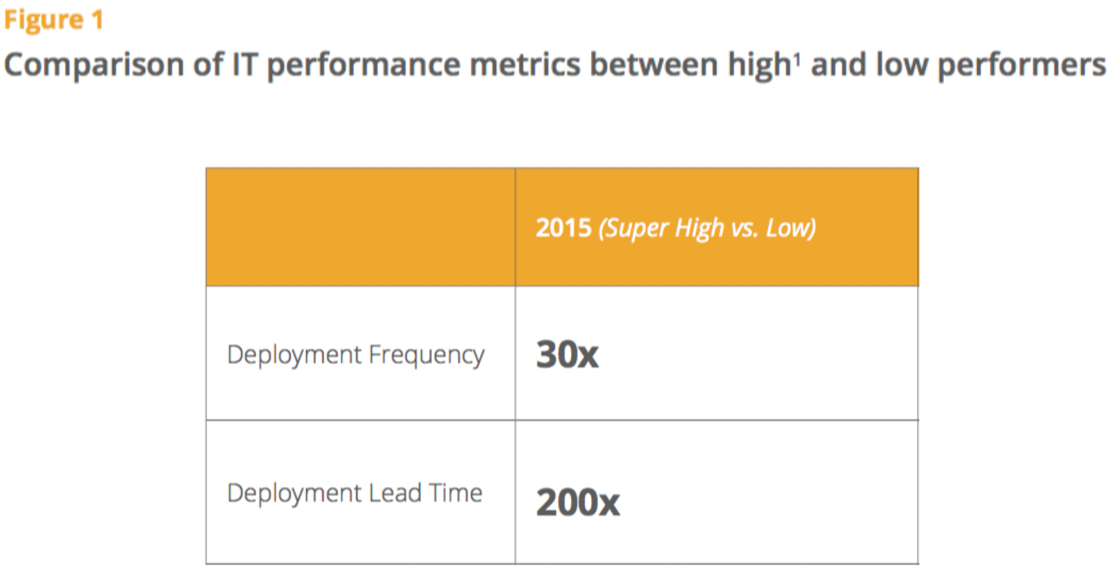


Figure 1

Comparison of IT performance metrics between high[1] and low performers

	2015 *(Super High vs. Low)*
Deployment Frequency	**30x**
Deployment Lead Time	**200x**

https://puppet.com/resources/whitepaper/2015-state-devops-report

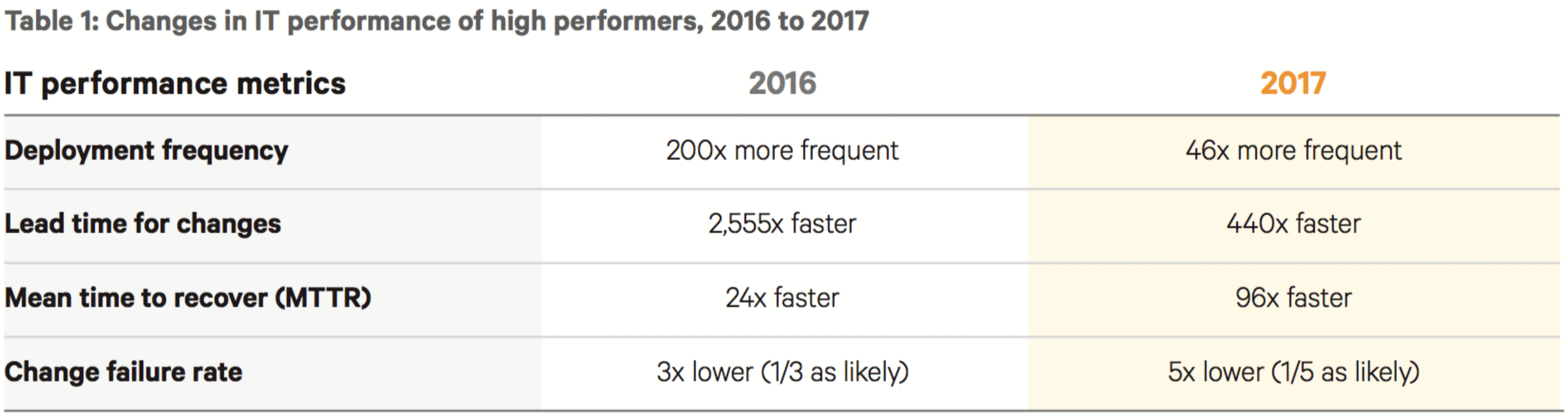


Table 1: Changes in IT performance of high performers, 2016 to 2017

IT performance metrics	2016	2017
Deployment frequency	200x more frequent	46x more frequent
Lead time for changes	2,555x faster	440x faster
Mean time to recover (MTTR)	24x faster	96x faster
Change failure rate	3x lower (1/3 as likely)	5x lower (1/5 as likely)

https://puppet.com/resources/whitepaper/state-of-devops-report

Goal

We want to use microservices architecture, where it makes sense, to help speed up an organization’s development velocity while lowering the chances of bad things happening or being able to understand and recover quickly if it does.

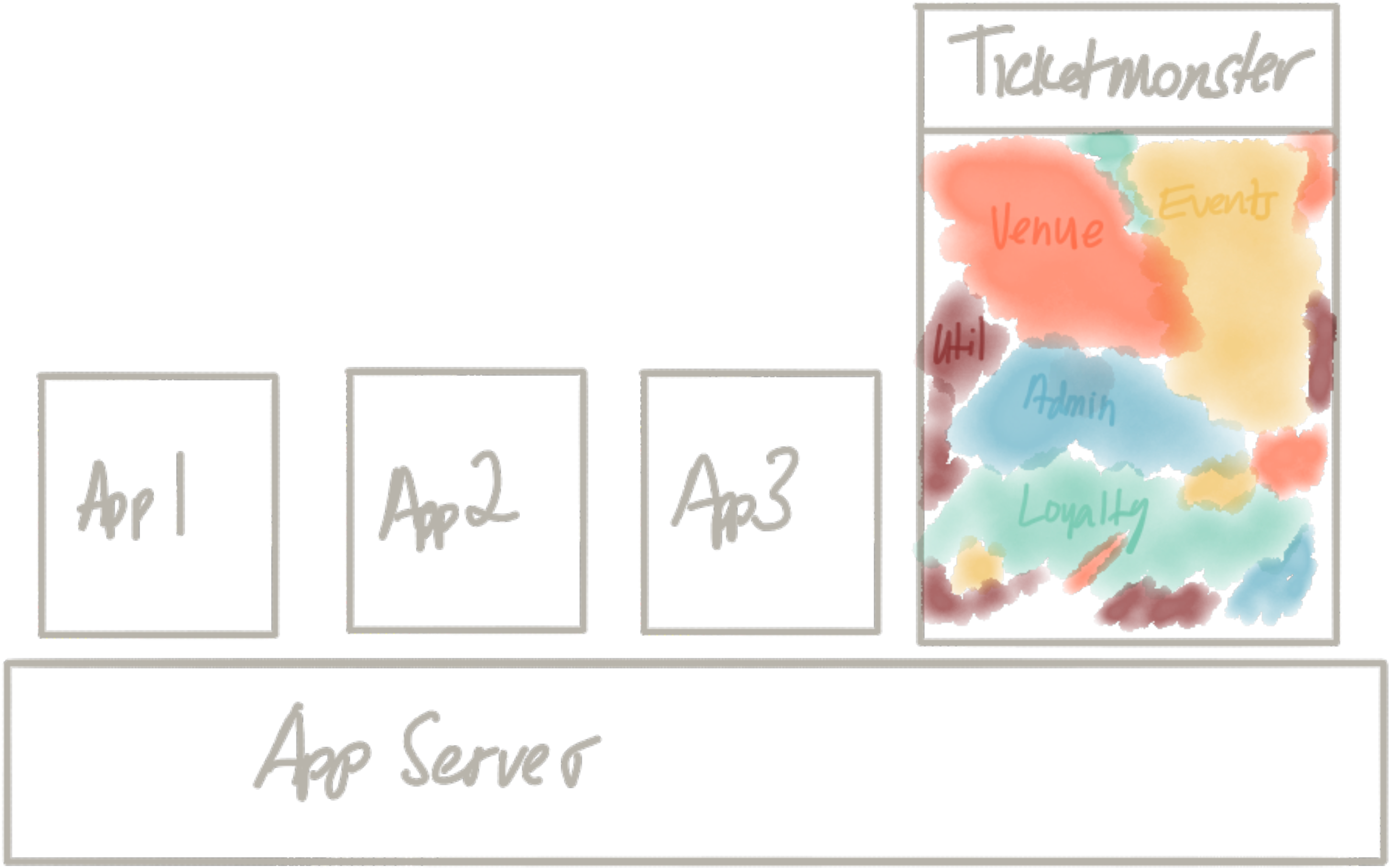
MEET OUR CASE STUDY

TicketMonster.

A JBoss Example.

TicketMonster is an online ticketing demo application that gets you started with JBoss

https://developers.redhat.com/ticket-monster/


Ticketmonster
Venue
Events
Util
Admin
Loyalty
App1
App2
App3
App Server

Some pain maintaining a monolith:

- Making changes in one place negatively affects unrelated areas
- Low confidence making changes that don’t break things
- Spend lots of time trying to coordinate work between team members
- Structure in the application has eroded or is non-existant
- We have no way to quantify how long code merges will take

Some pain maintaining a monolith:

- Development time is slow simply because the project is so big (IDE bogs down, running tests is slow, slow bootstrap time, etc)
- Changes to one module force changes across other modules
- Difficult to sunset outdated technology
- We’ve built our new applications around old premises like batch processing
- Application steps on itself at runtime managing resources, allocations, computations

QUICK INTERLUDE:
WHEN TO DO MICROSERVICES

Microservices is about optimizing for speed

So, do we microservices all the way down?

Ask a very honest, and critical, question:

Is our *application architecture* the bottleneck for being able to go faster?

“No”, “Not really”, “Not yet”... then stop

Go find out what is. Improve that. Then come back.

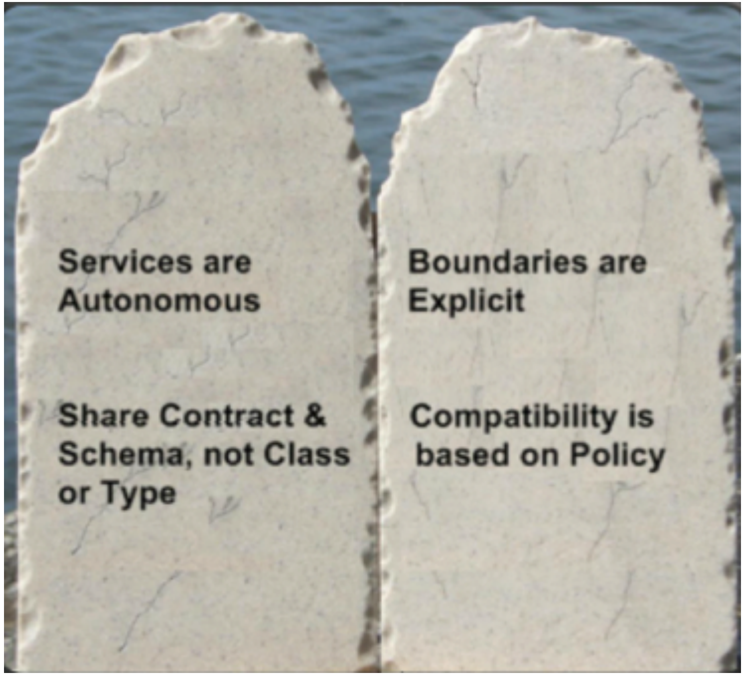
MEANWHILE…

How do you break this thing up?

Some ramblings...

- Do one thing and do it well
- Single responsibility principle
- Organize around nouns
- Organize around verbs
- Bounded context
- Products not projects
- Unix philosophy

Reminds me of yesteryear

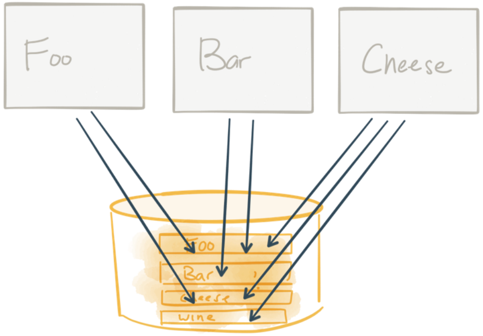

Services are Autonomous
Share Contract & Schema, not Class or Type
Boundaries are Explicit
Compatibility is based on Policy

https://www.infoq.com/presentations/SOA-Business-Autonomous-Components

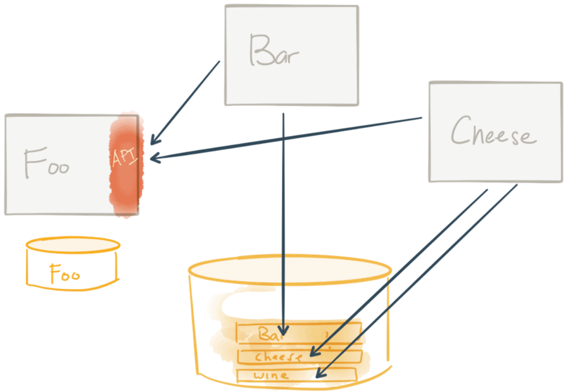
Try one more time...

- Identify modules, boundaries
- Align to business capabilities
- Identify data entities responsible for features/modules
- Break out these entities and wrap with an API/service
- Update old code to call this new API service

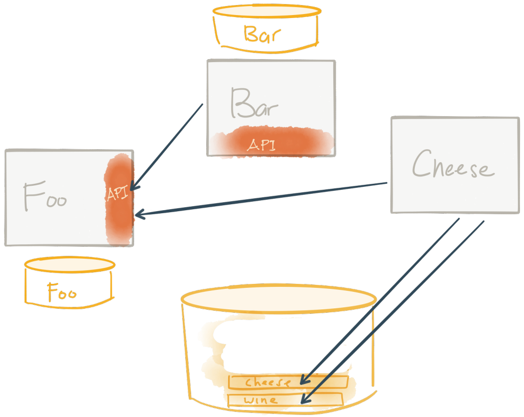
Identify modules



Break out API



Rinse, repeat



Generally good; misses a lot of detail!

Try one more time...

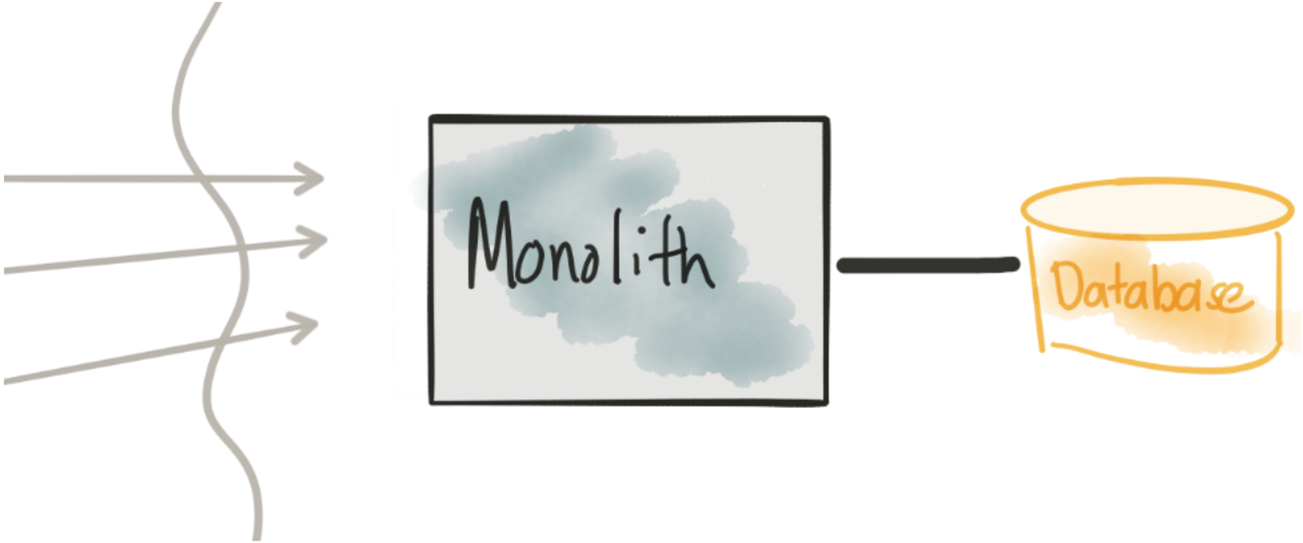
- Not easy to “re-modularize” a monolith
- Tight coupling/integrity constraints between normalized tables
- Difficult to understand which modules use which tables
- We cannot stop the world to perform migrations
- there will be some ugly migration steps that cannot just be wished away
- there is probably a point of diminishing returns where it doesn’t make sense to break things out of the monolith

Make sure…

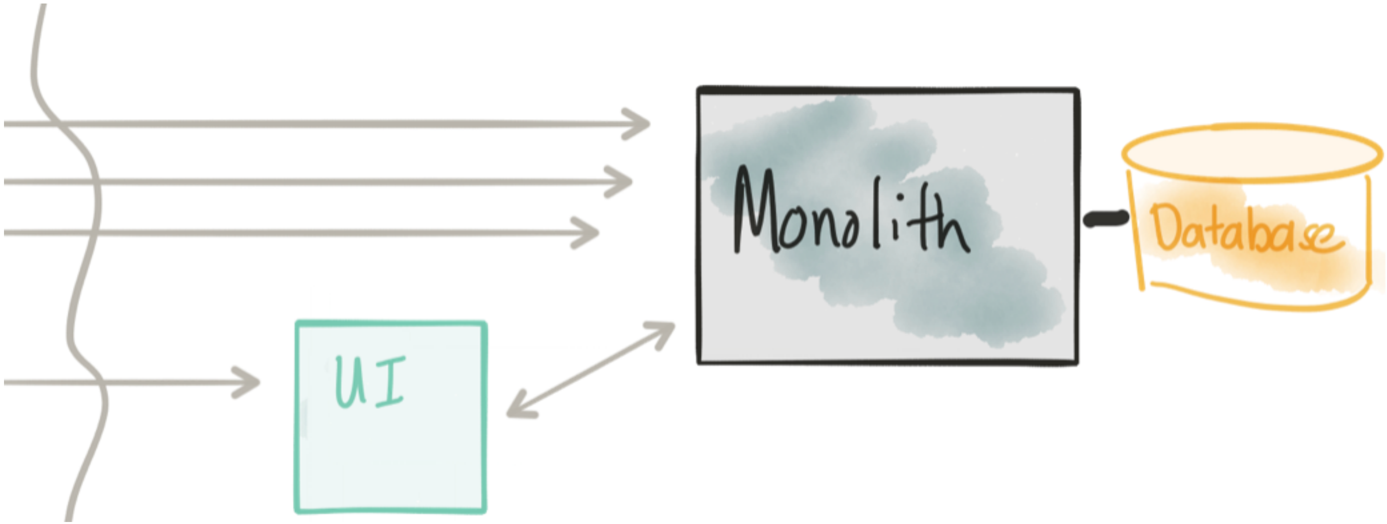
- You have test coverage for existing project (ie, passing tests, CI processes, etc)
- Consider Arquillian for integration testing
- Make sure you have some level of monitoring to detect issues / exceptions / etc
- Have some level of black-box system tests in place / load testing (JMeter, Gattling)
- Can deploy reliable to an environment (ideally OpenShift/ Kubernetes!)
- Have some kind of CI/CD to be able to make changes economical

OUR MONOLITH

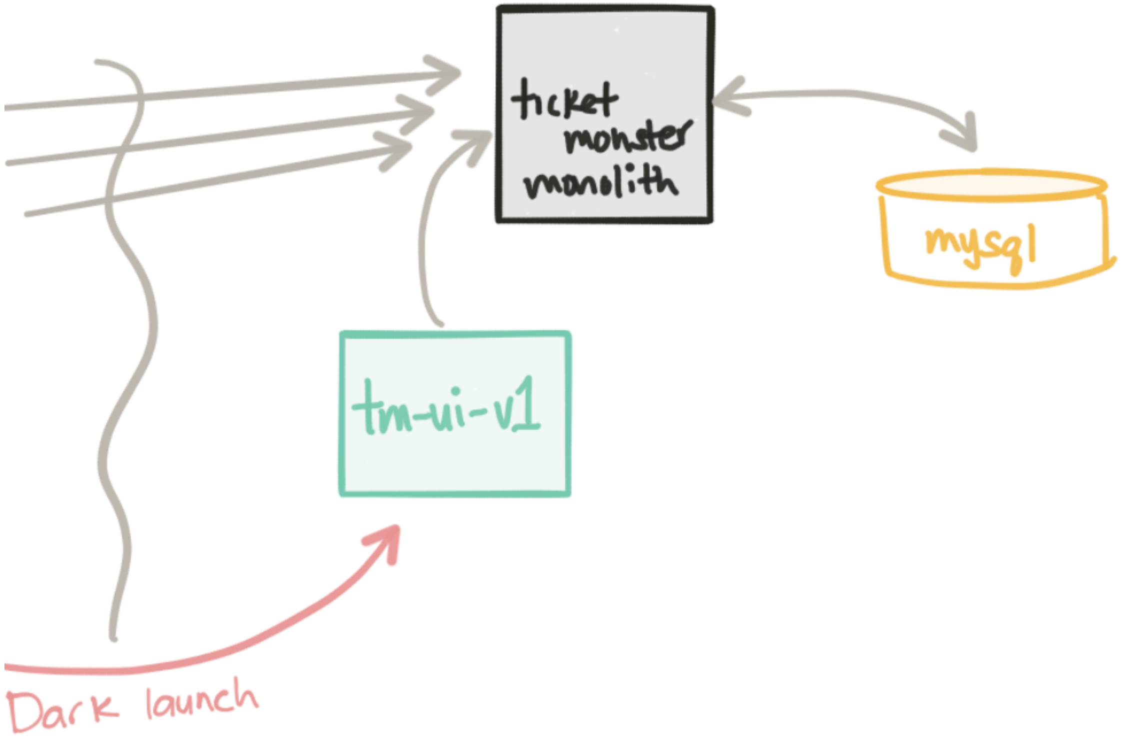
Our monolith



Break out UI (if applicable)

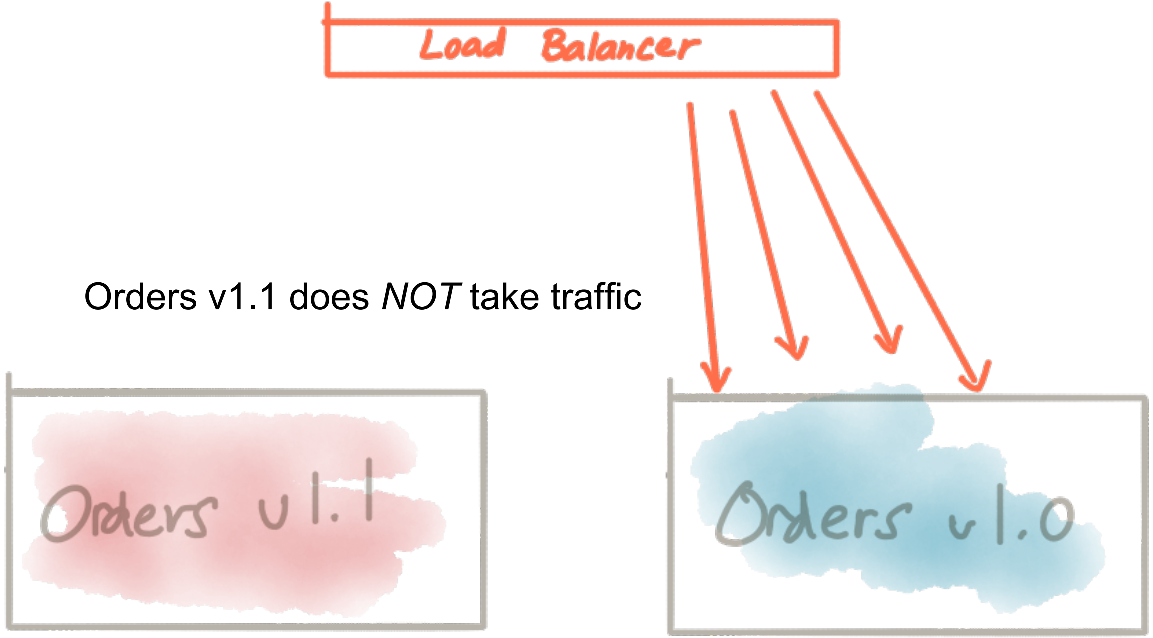


Deployment v release gives us flexibility



QUICK INTERLUDE:
DEPLOYMENT VS RELEASE

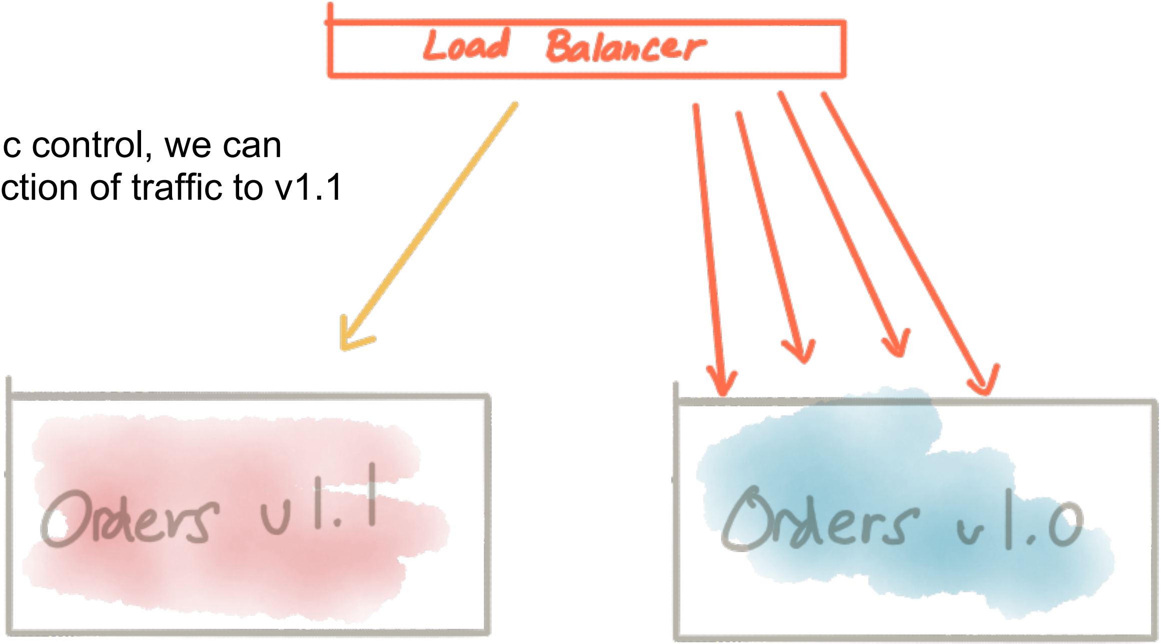
Decoupling deployment from release



Orders v1.1 does *NOT* take traffic

Here, we've *deployed* Orders v1.1

Decoupling deployment from release



Using traffic control, we can direct a fraction of traffic to v1.1

Here, we've begun a release of Orders v1.1

Meet Istio.io

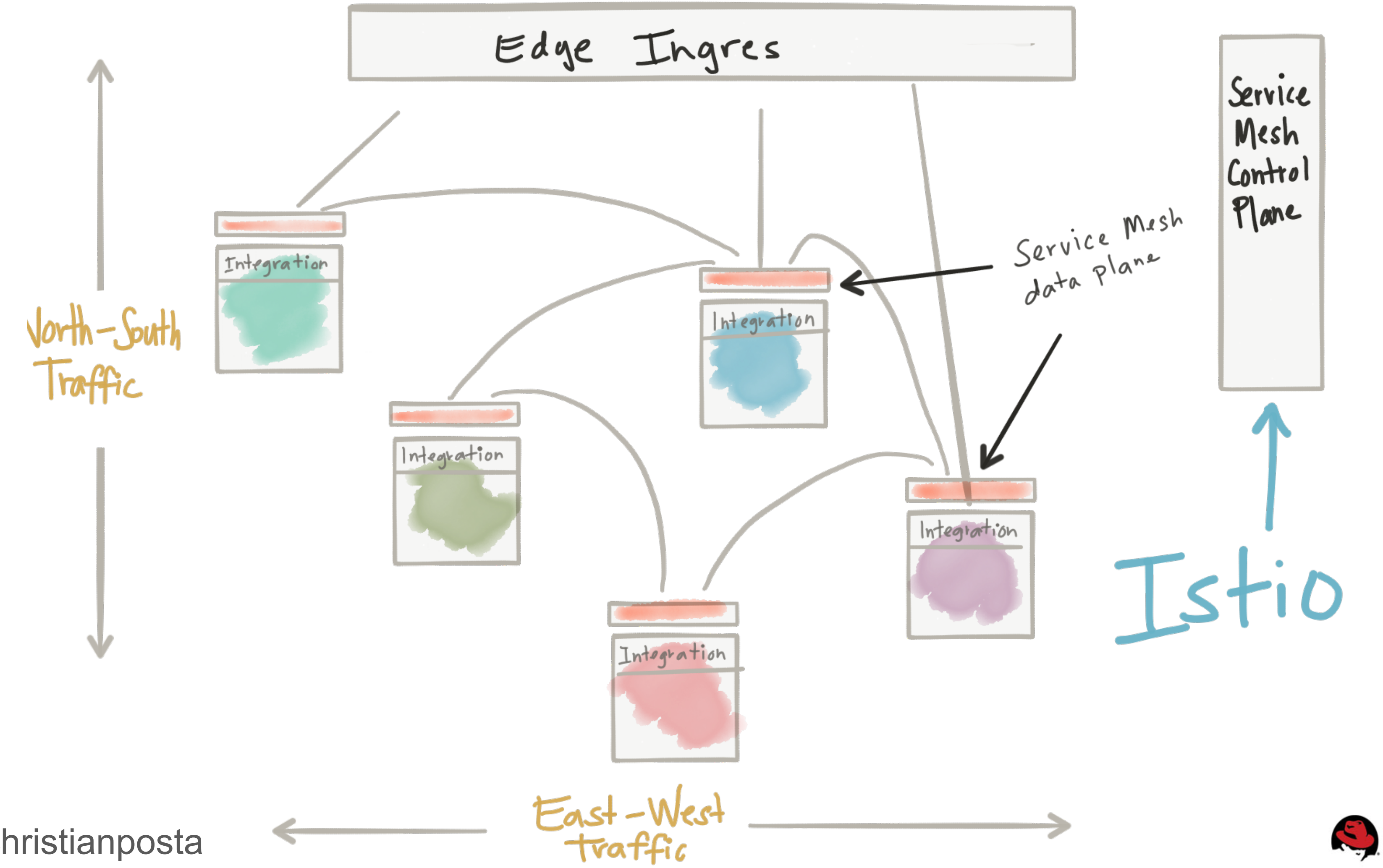
http://istio.io

An open-source service mesh

Time for definitions:

A **service mesh** is *decentralized* application-networking infrastructure *between your services* that provides *resiliency*, *security*, *observability*, and *routing control*.

A service mesh is comprised of a *data plane* and *control plane*.

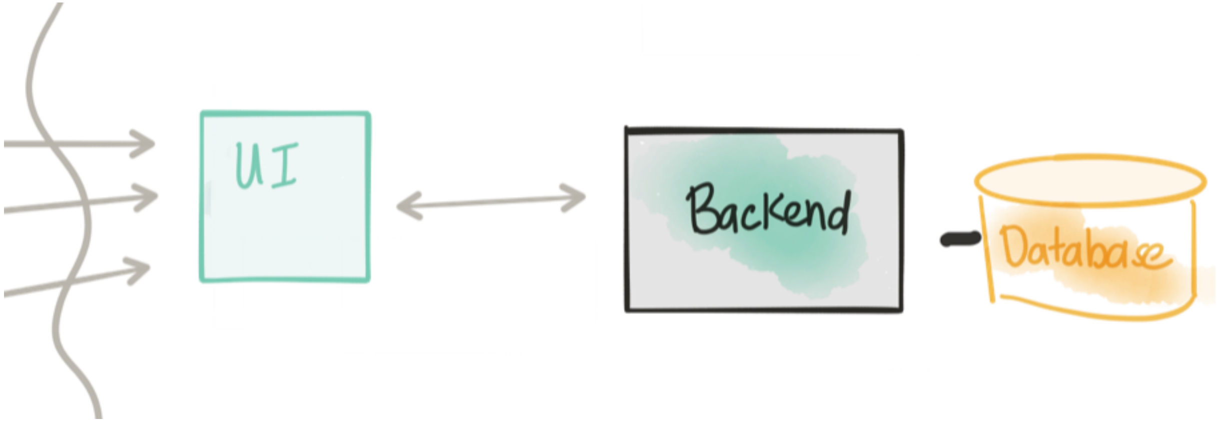


@christianposta

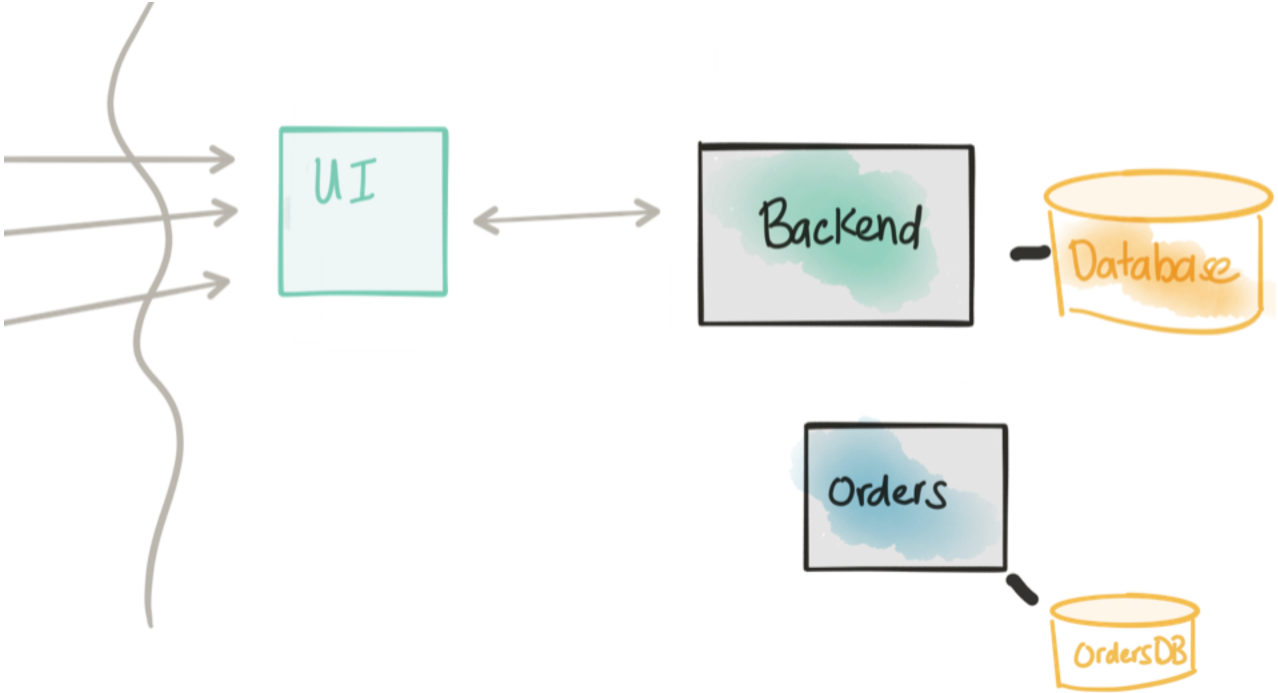
redhat.

MEANWHILE…

Let’s call it backend now...



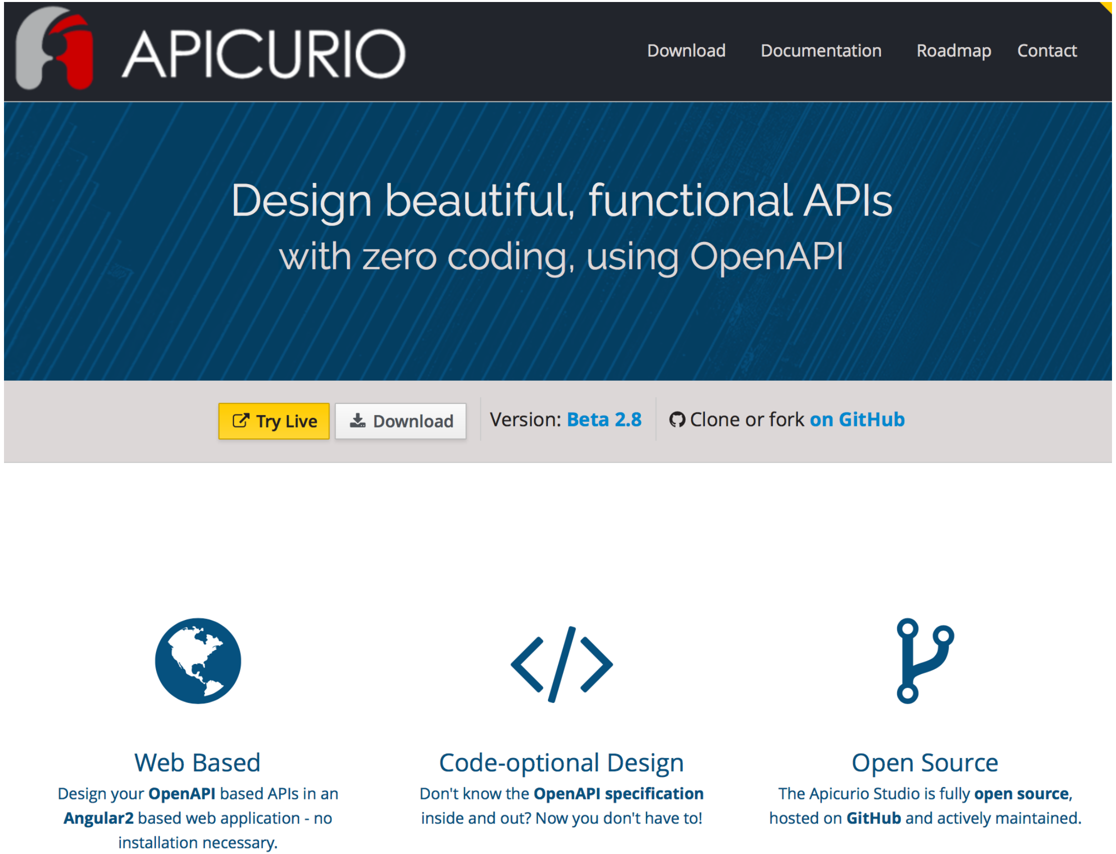
Introduce a new *Orders* service



Introducing new service API

- We want to focus on the API design / boundary of our extracted service
- This may be a re-write from what exists in the monolith
- We should iterate on the API and share with our collaborators
- We can stub out the service with Microcks/Hoverfly
- This service will have its own data storage
- This service will not receive any traffic at this point
- Put in place “walking skeleton” to exercise CI/CD pipeline

apicur.io for designing the API



Design beautiful, functional APIs
with zero coding, using OpenAPI

Version: **Beta 2.8**

Clone or fork **on GitHub**

Web Based

Design your **OpenAPI** based APIs in an **Angular2** based web application - no installation necessary.

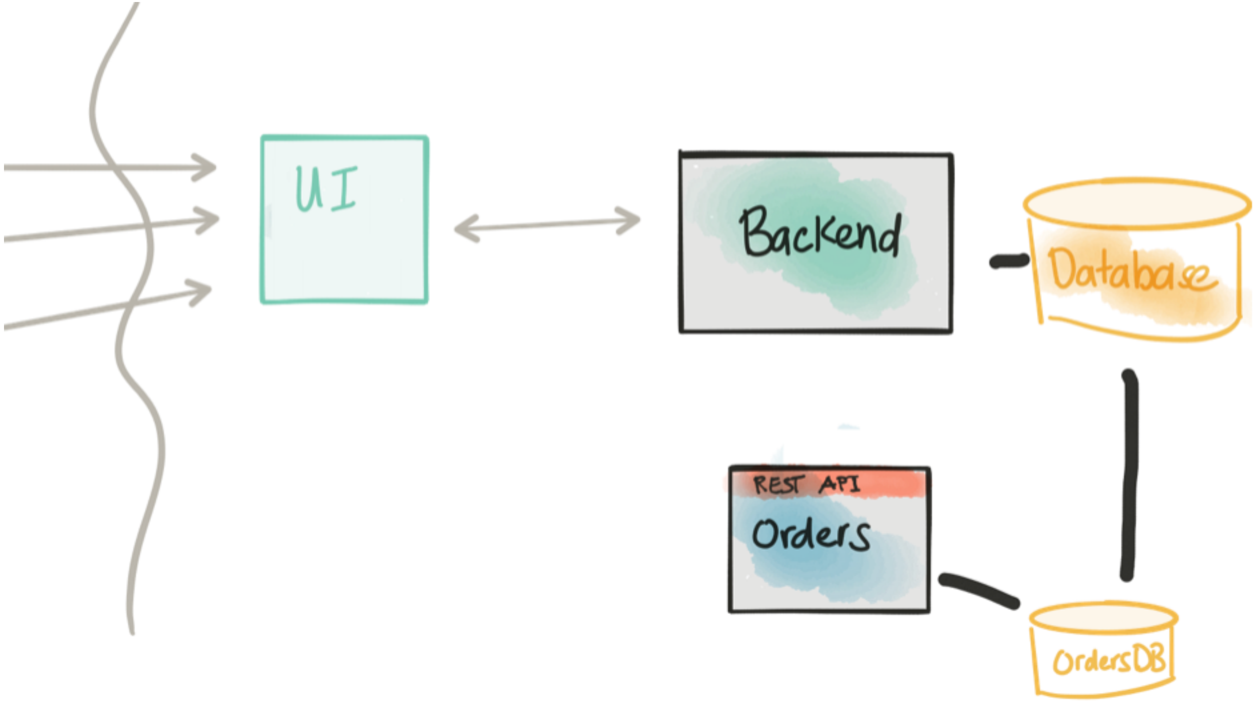
Code-optional Design

Don't know the **OpenAPI specification** inside and out? Now you don't have to!

Open Source

The Apicurio Studio is fully **open source**, hosted on **GitHub** and actively maintained.

Create an implementation



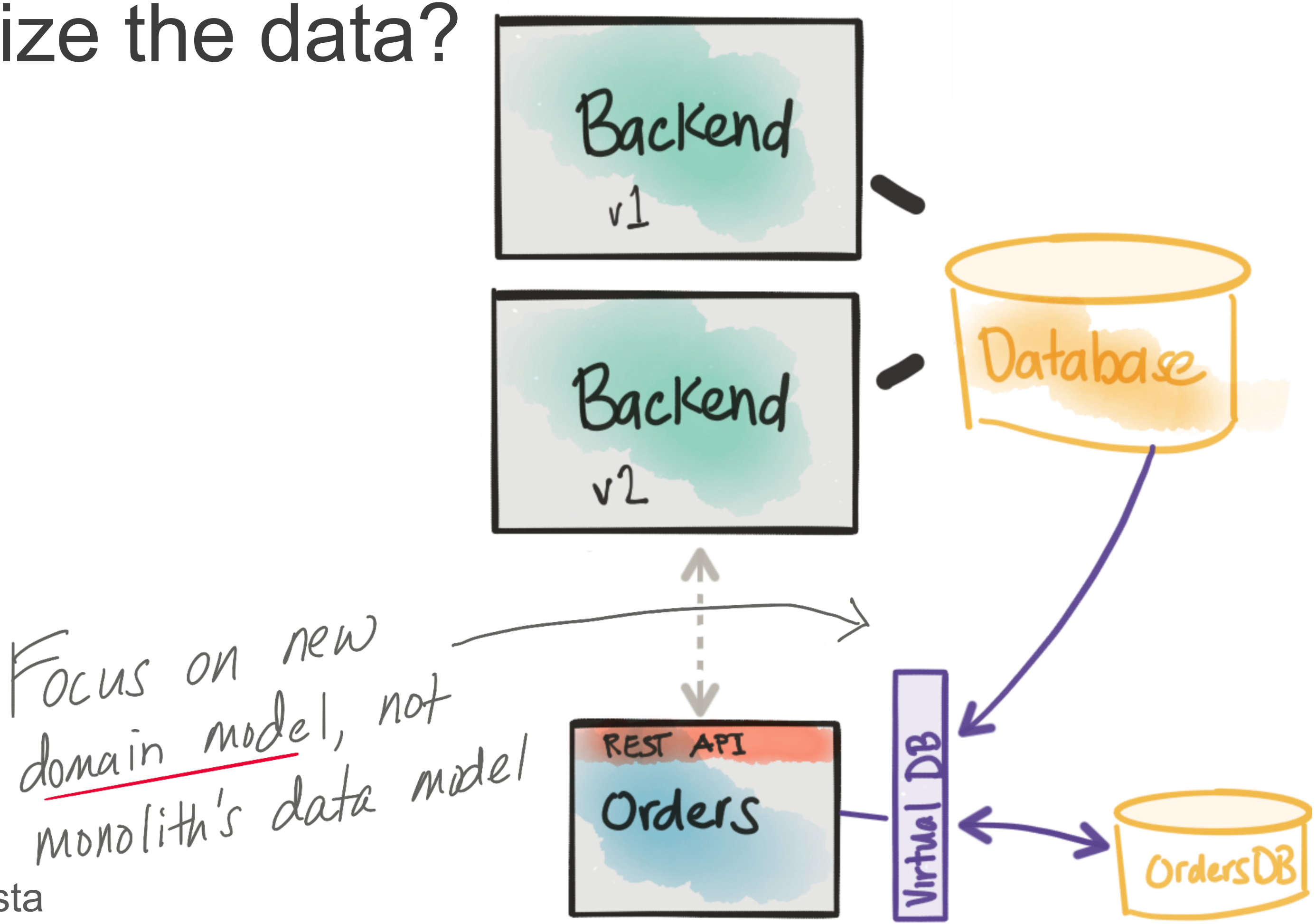
Shared data

- New service will share concepts with monolith
- We will need a way to reify that data within the microservice
- The monolith probably doesn’t provide an API at the right level
- Shaping the data from the monolith’s API requires boiler plate code
- Could create a new API for the monolith
- Could copy the data
- Could connect right up (yuck!)

Virtualize the data?

- Focus on the new service's *domain model*
- Eliminate any boiler plate code
- Read only virtual view of the monolith's data
- Read/write our own database, without changing data model
- Part of a series of steps that ends with eliminating the virtual view

Virtualize the data?



Focus on new domain model, not monolith's data model

@christianposta

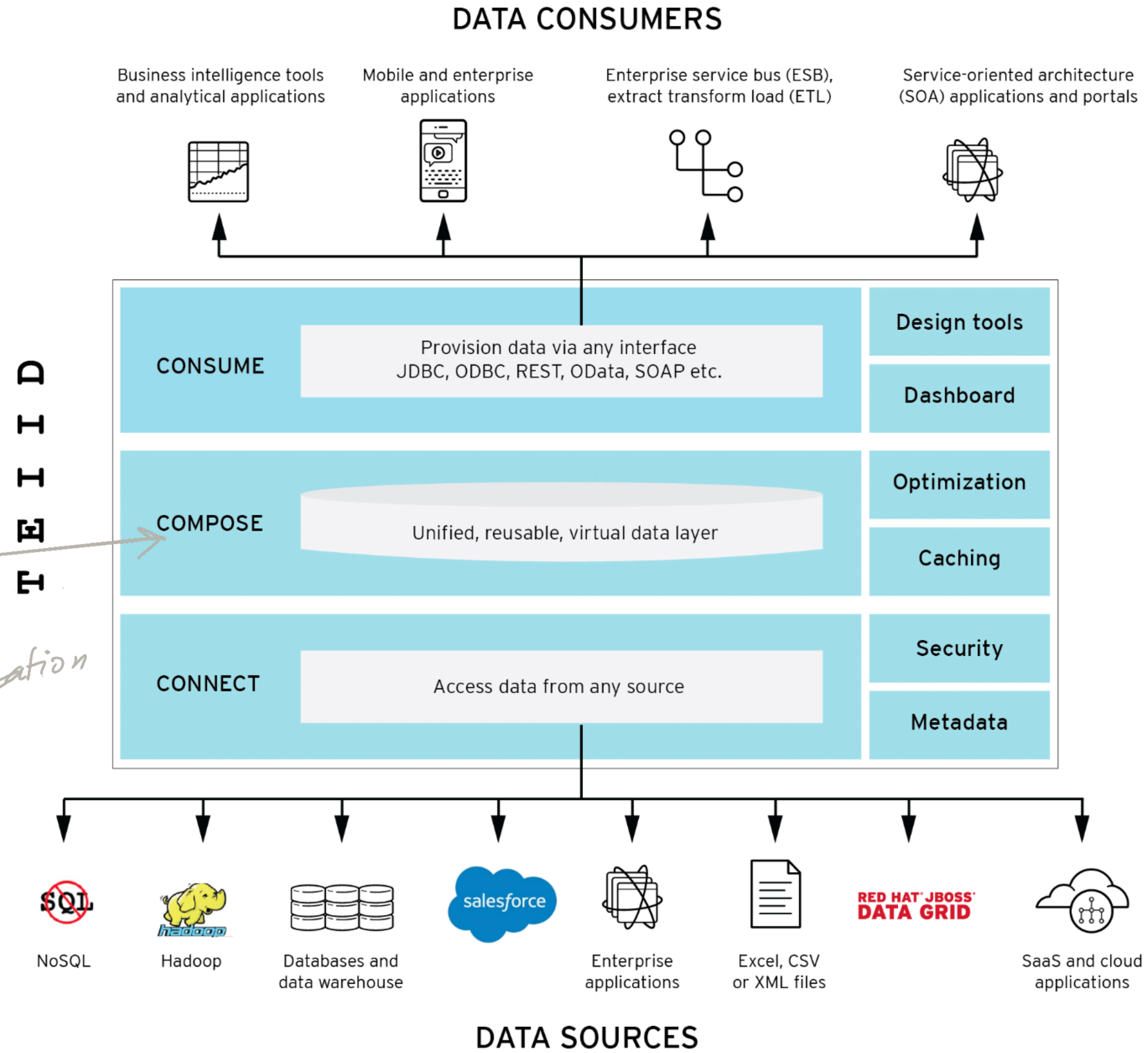
redhat.

QUICK INTERLUDE:
BOILERPLATE DATA INTEGRATION

Teiid

The data you want from the data you have.

http://teiid.jboss.org



DATA CONSUMERS
Business intelligence tools and analytical applications
Mobile and enterprise applications
Enterprise service bus (ESB), extract transform load (ETL)
Service-oriented architecture (SOA) applications and portals
TEIID
CONSUME
Provision data via any interface JDBC, ODBC, REST, OData, SOAP etc.
COMPOSE
Unified, reusable, virtual data layer
CONNECT
Access data from any source
Design tools
Dashboard
Optimization
Caching
Security
Metadata
NoSQL
Hadoop
Databases and data warehouse
salesforce
Enterprise applications
Excel, CSV or XML files
RED HAT JBOSS DATA GRID
SaaS and cloud applications
DATA SOURCES

Pretty powerful, but we just need the embedded virtualization engine

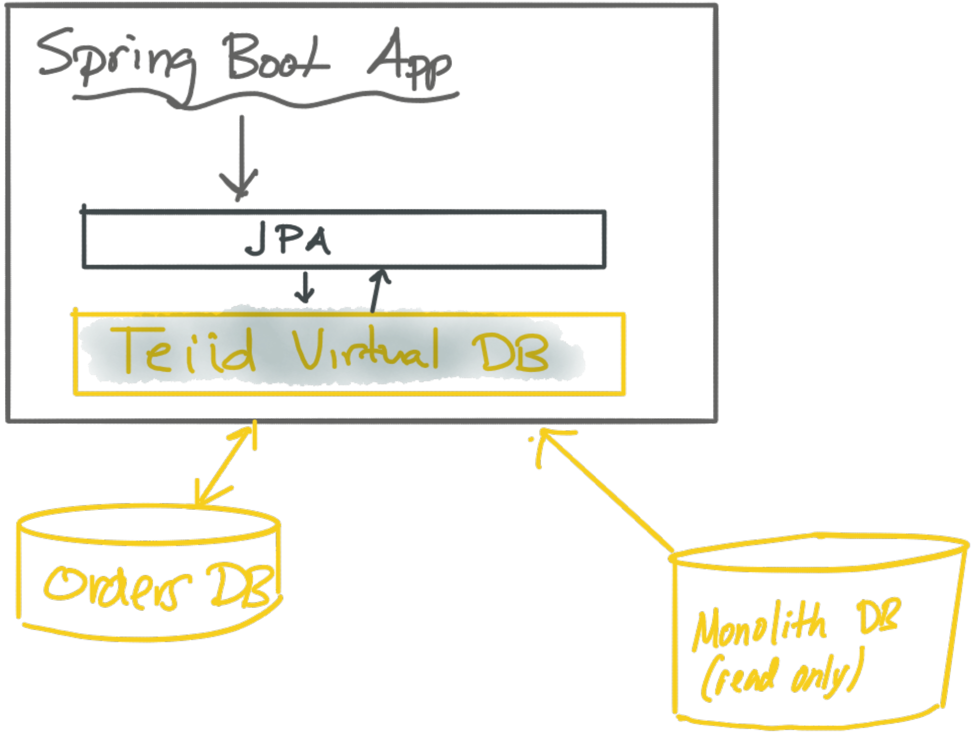
@christianposta

redhat

Teiid

The data you want from the data you have.

https://github.com/teiid/teiid-spring-boot



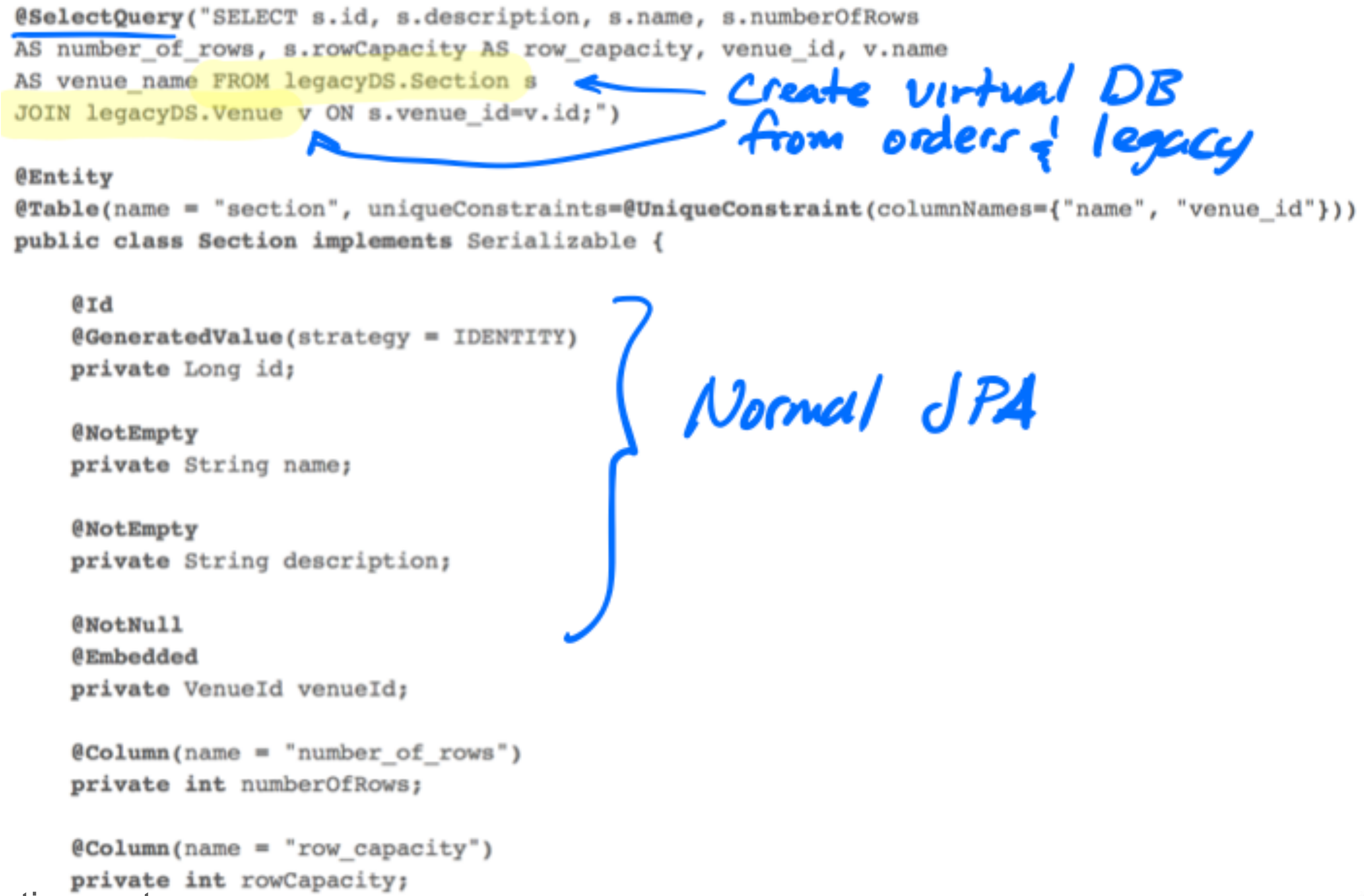
Set up Spring Boot

```
<dependency>
  <groupId>org.teiid.spring</groupId>
  <artifactId>teiid-spring-boot-starter</artifactId>
  <version>1.0.0-SNAPSHOT</version>
</dependency>
```

← Pom.xml

application.properties

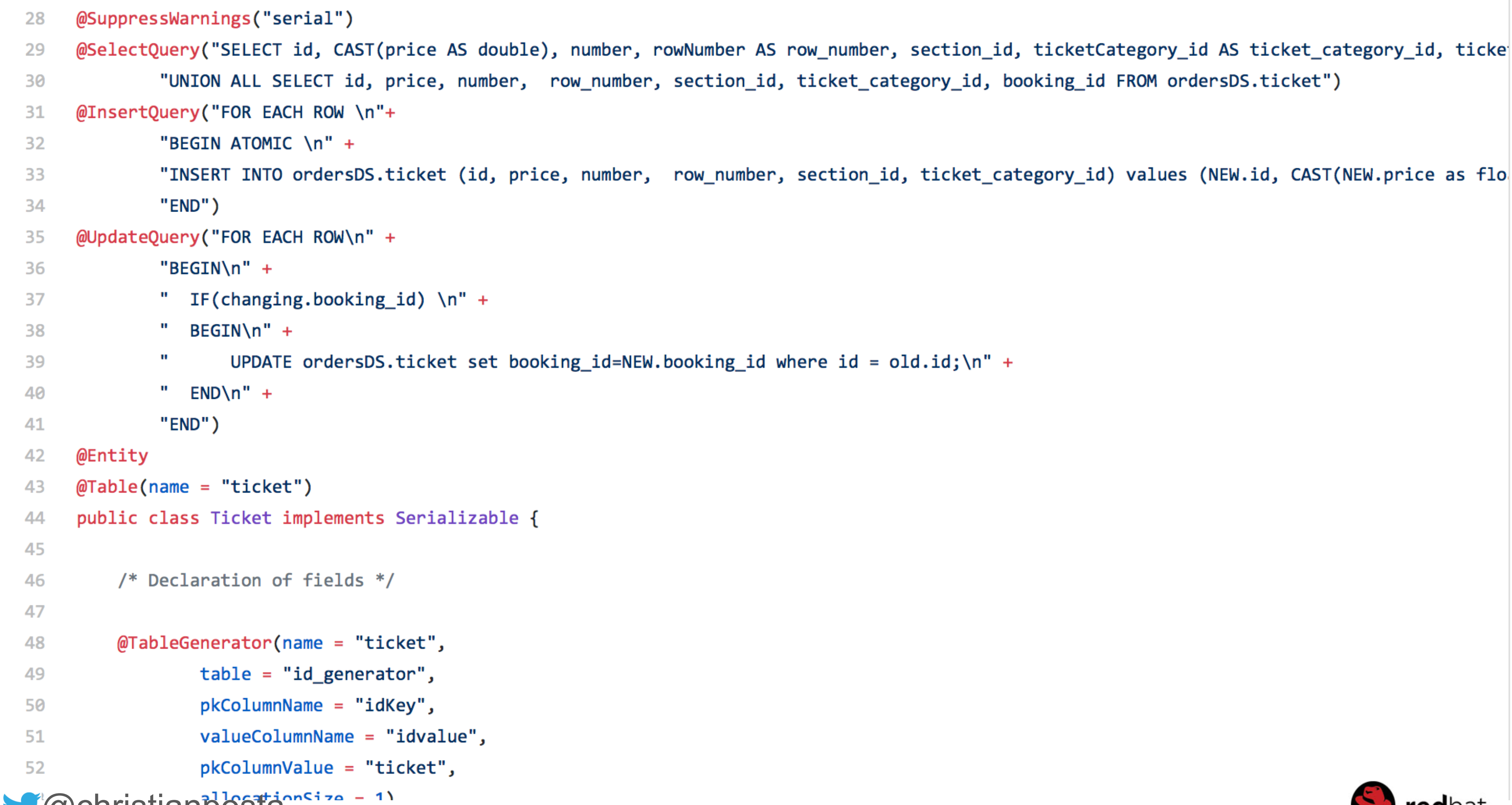
```
spring.datasource.legacyDS.url=jdbc:mysql://localhost:3306/ticketmonster?useSSL=false
spring.datasource.legacyDS.username=ticket
spring.datasource.legacyDS.password=monster
spring.datasource.legacyDS.driverClassName=com.mysql.jdbc.Driver

spring.datasource.ordersDS.url=jdbc:mysql://localhost:3306/orders?useSSL=false
spring.datasource.ordersDS.username=ticket
spring.datasource.ordersDS.password=monster
spring.datasource.ordersDS.driverClassName=com.mysql.jdbc.Driver
```



```
@SelectQuery("SELECT s.id, s.description, s.name, s.numberOfRows
AS number_of_rows, s.rowCapacity AS row_capacity, venue_id, v.name
AS venue_name FROM legacyDS.Section s
JOIN legacyDS.Venue v ON s.venue_id=v.id;")

@Entity
@Table(name = "section", uniqueConstraints=@UniqueConstraint(columnNames={"name", "venue_id"}))
public class Section implements Serializable {

    @Id
    @GeneratedValue(strategy = IDENTITY)
    private Long id;

    @NotEmpty
    private String name;

    @NotEmpty
    private String description;

    @NotNull
    @Embedded
    private VenueId venueId;

    @Column(name = "number_of_rows")
    private int numberOfRows;

    @Column(name = "row_capacity")
    private int rowCapacity;
```

Create virtual DB from orders & legacy

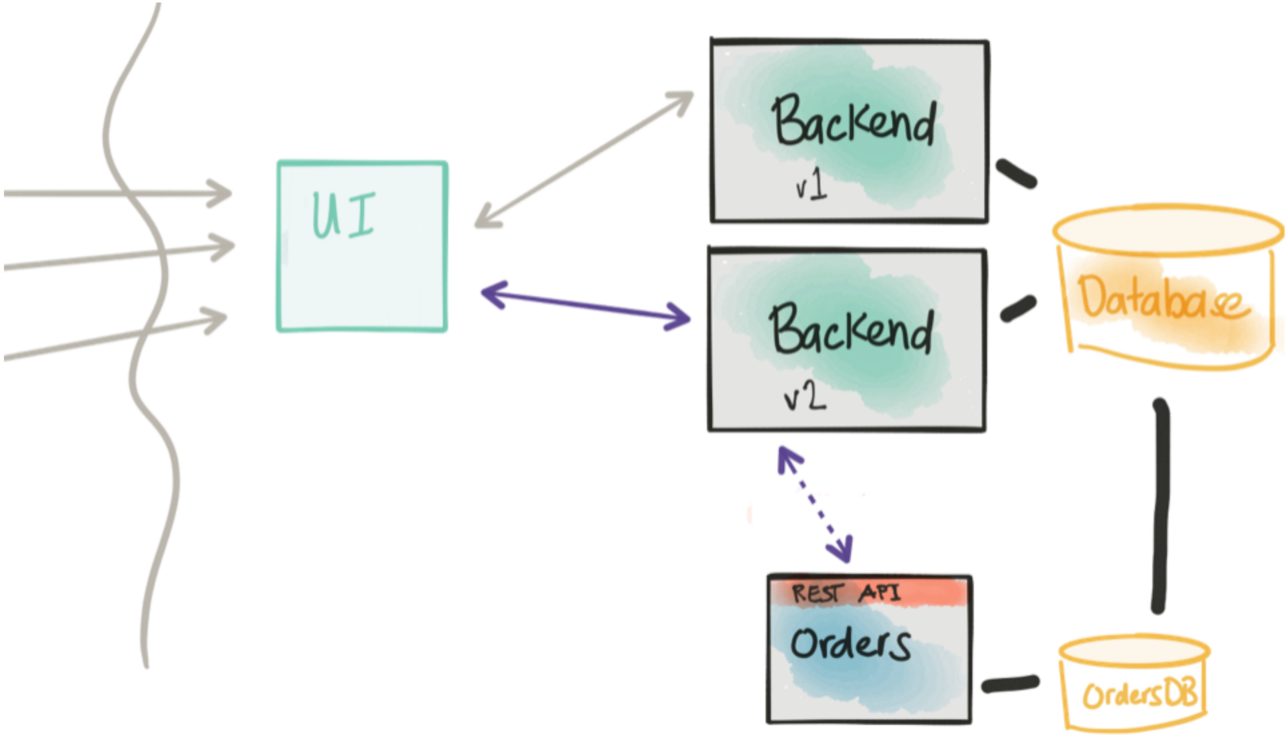
Normal JPA



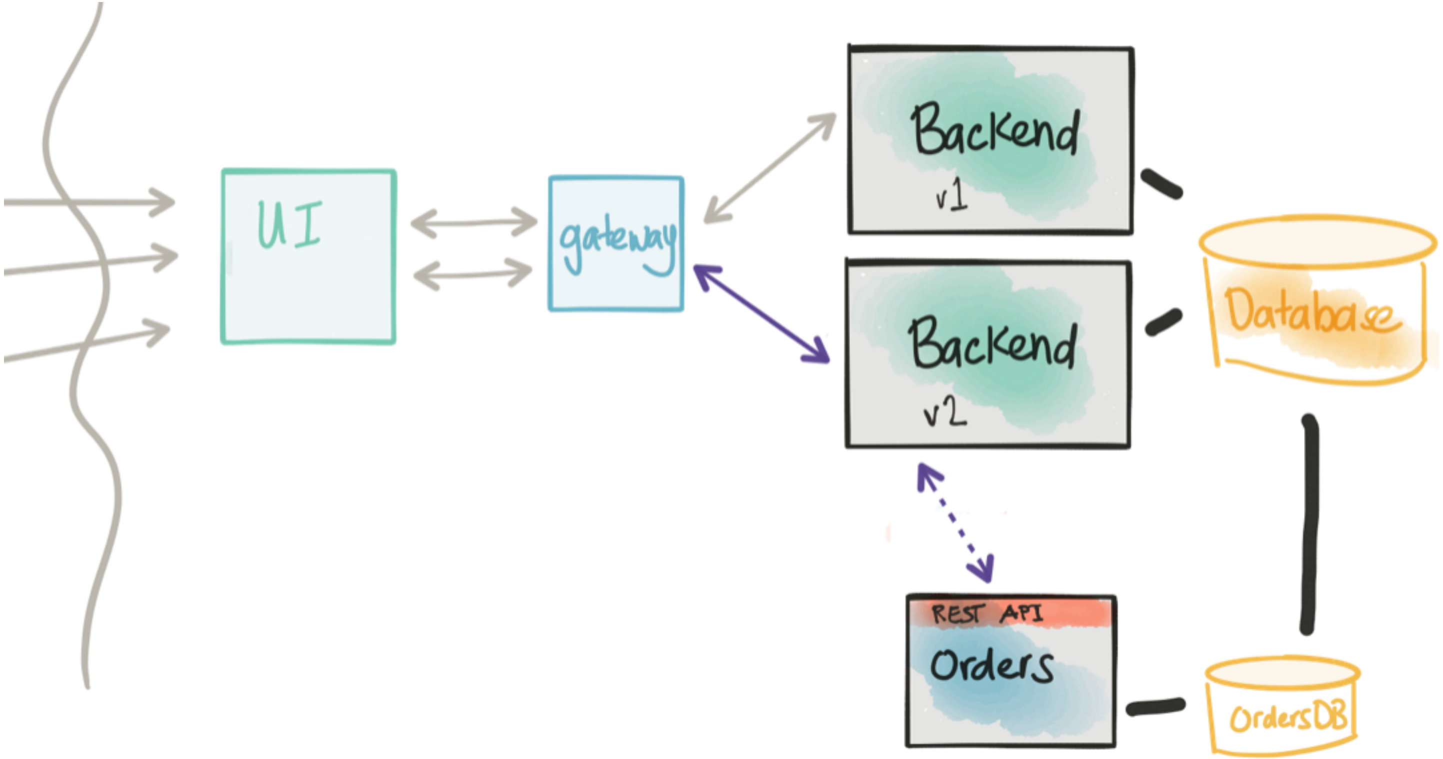
```
@SuppressWarnings("serial")
@SelectQuery("SELECT id, CAST(price AS double), number, rowNumber AS row_number, section_id, ticketCategory_id AS ticket_category_id, ticke
        "UNION ALL SELECT id, price, number,  row_number, section_id, ticket_category_id, booking_id FROM ordersDS.ticket")
@InsertQuery("FOR EACH ROW \n"+
        "BEGIN ATOMIC \n" +
        "INSERT INTO ordersDS.ticket (id, price, number,  row_number, section_id, ticket_category_id) values (NEW.id, CAST(NEW.price as flo
        "END")
@UpdateQuery("FOR EACH ROW\n" +
        "BEGIN\n" +
        "  IF(changing.booking_id) \n" +
        "  BEGIN\n" +
        "      UPDATE ordersDS.ticket set booking_id=NEW.booking_id where id = old.id;\n" +
        "  END\n" +
        "END")
@Entity
@Table(name = "ticket")
public class Ticket implements Serializable {

    /* Declaration of fields */

    @TableGenerator(name = "ticket",
            table = "id_generator",
            pkColumnName = "idKey",
            valueColumnName = "idvalue",
            pkColumnValue = "ticket",
            allocationSize = 1)
```

MEANWHILE…

Mirror traffic to new service

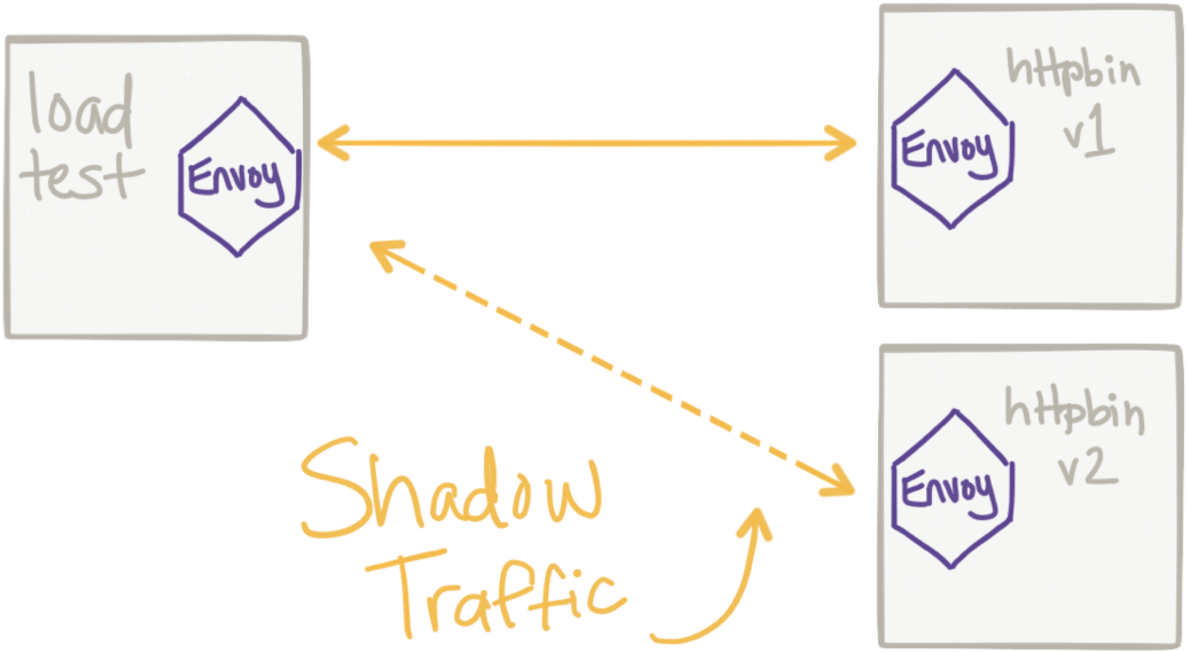


Mirror traffic to new service

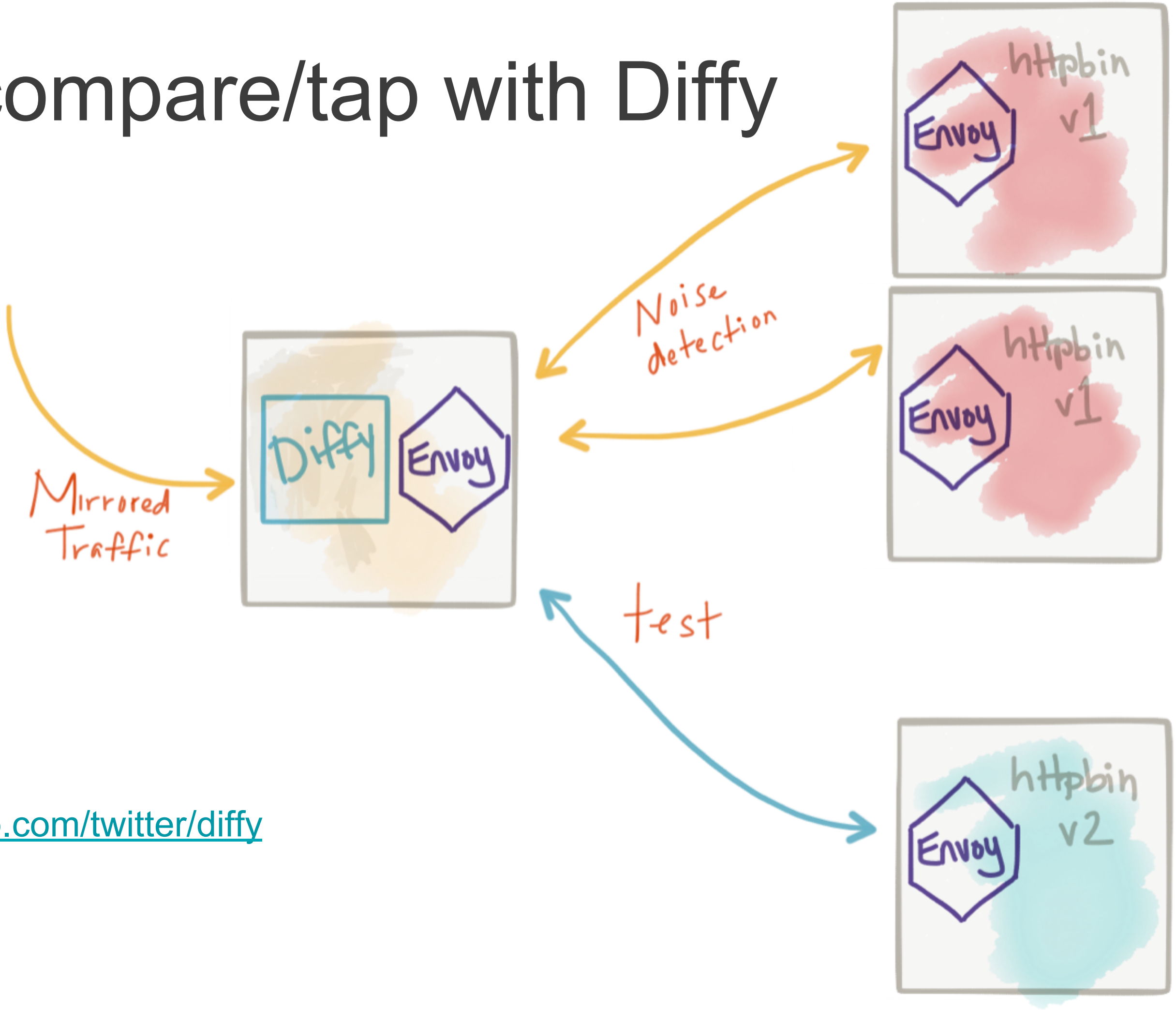


QUICK INTERLUDE: TRAFFIC MIRRORING

Mirror traffic with Istio



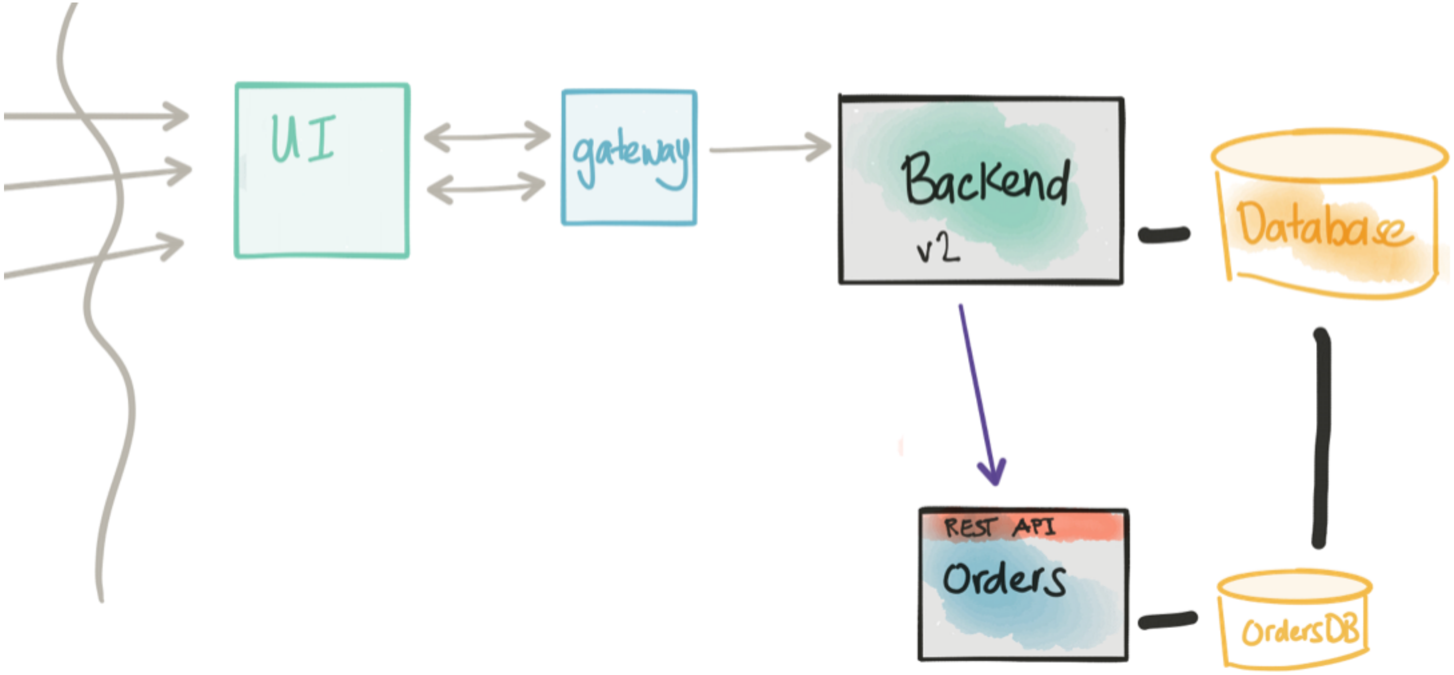
Traffic compare/tap with Diffy



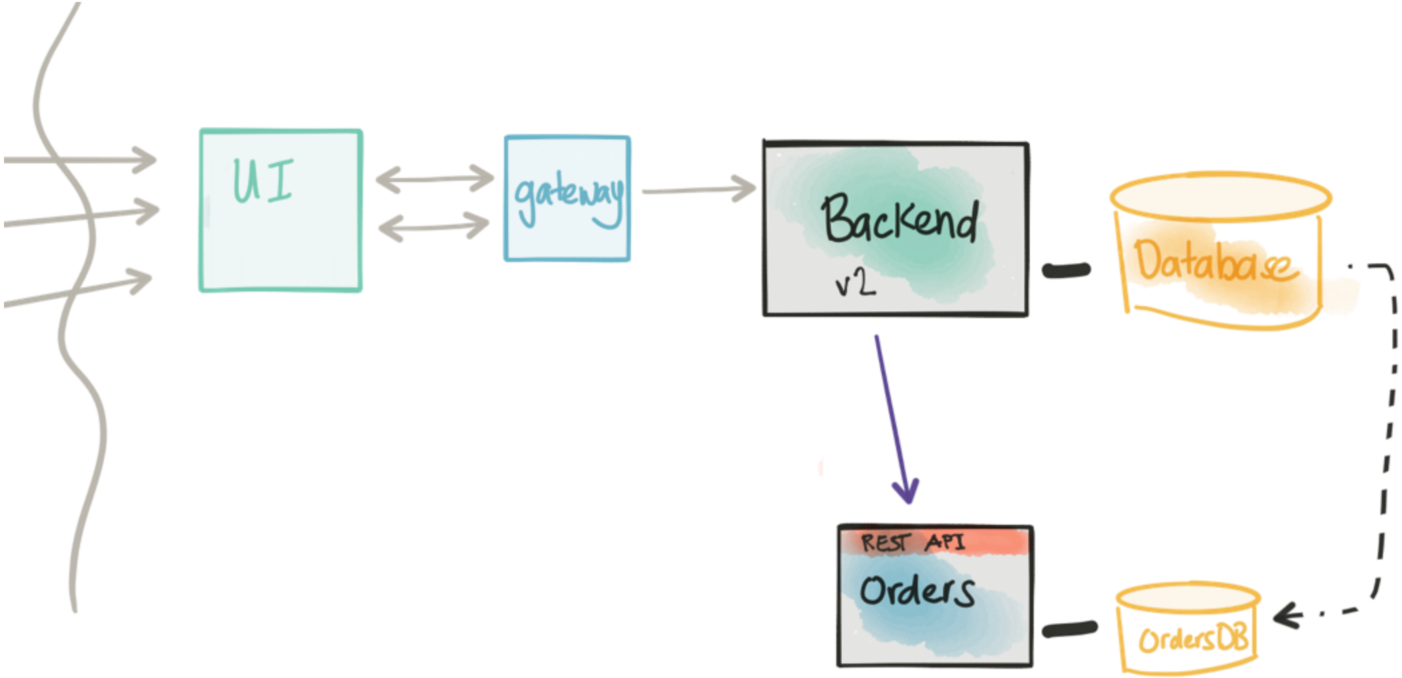
https://github.com/twitter/diffy

MEANWHILE…

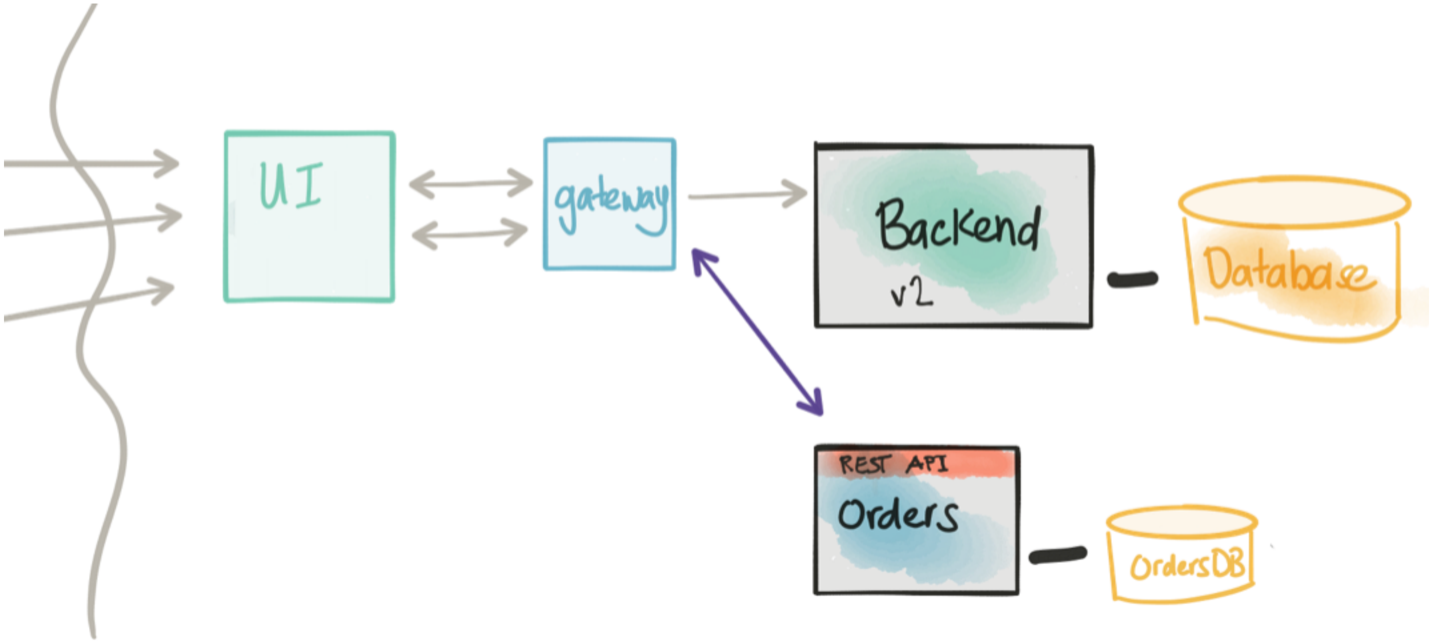
Feature flags for runtime kill switch



Async Change Data Capture with Debezium.io?



Eliminate dependency on monolith DB



Recap

- Write lots of tests (for monolith if you can; especially new service)
- Use advanced deployment techniques (canarying, tap compare, mirroring)
- Use fine-grain traffic control to separate deployment from release
- Reduce boiler plate code for data integration in initial service implementation
- Use technical debt to your advantage
- Have lots of monitoring in place
- Leverage your deployment and release infrastructure to experiment and learn!

Quick demo?

Twitter: @christianposta

Blog: http://blog.christianposta.com

Email: christian@redhat.com

Slides: http://slideshare.net/ceposta

Follow up links:

- http://openshift.io
- http://launch.openshift.io
- http://blog.openshift.com
- http://developers.redhat.com/blog
- https://www.redhat.com/en/open-innovation-labs
- https://www.redhat.com/en/technologies/jboss-middleware/3scale
- https://www.redhat.com/en/technologies/jboss-middleware/fuse

BTW: Hand drawn diagrams made with Paper by FiftyThree.com ☺

RED HAT
SUMMIT

THANK YOU

- plus.google.com/+RedHat
- facebook.com/redhatinc
- linkedin.com/company/red-hat
- twitter.com/RedHat
- youtube.com/user/RedHatVideos