



FLOW

Design Micro Service Architectures the Right Way

QCon
NEW YORK by **InfoQ**

Michael Bryzek

mike@flow.io / @mbryzek

Cofounder / CTO Flow

Cofounder / ex-CTO Gilt

A personal story

Could you change this URL from

<https://foo.com/latest/bar.js> to

[https://foo.com/1.5.3/ bar.js](https://foo.com/1.5.3/bar.js) ?

➔ **Sorry – that would take weeks; we don't have the resources to do that.**

It's just a friggin URL!!!



How does this happen?

- **URL in a library**
- **100s of services to update**
- **Some not touched in years, requiring dependency updates**



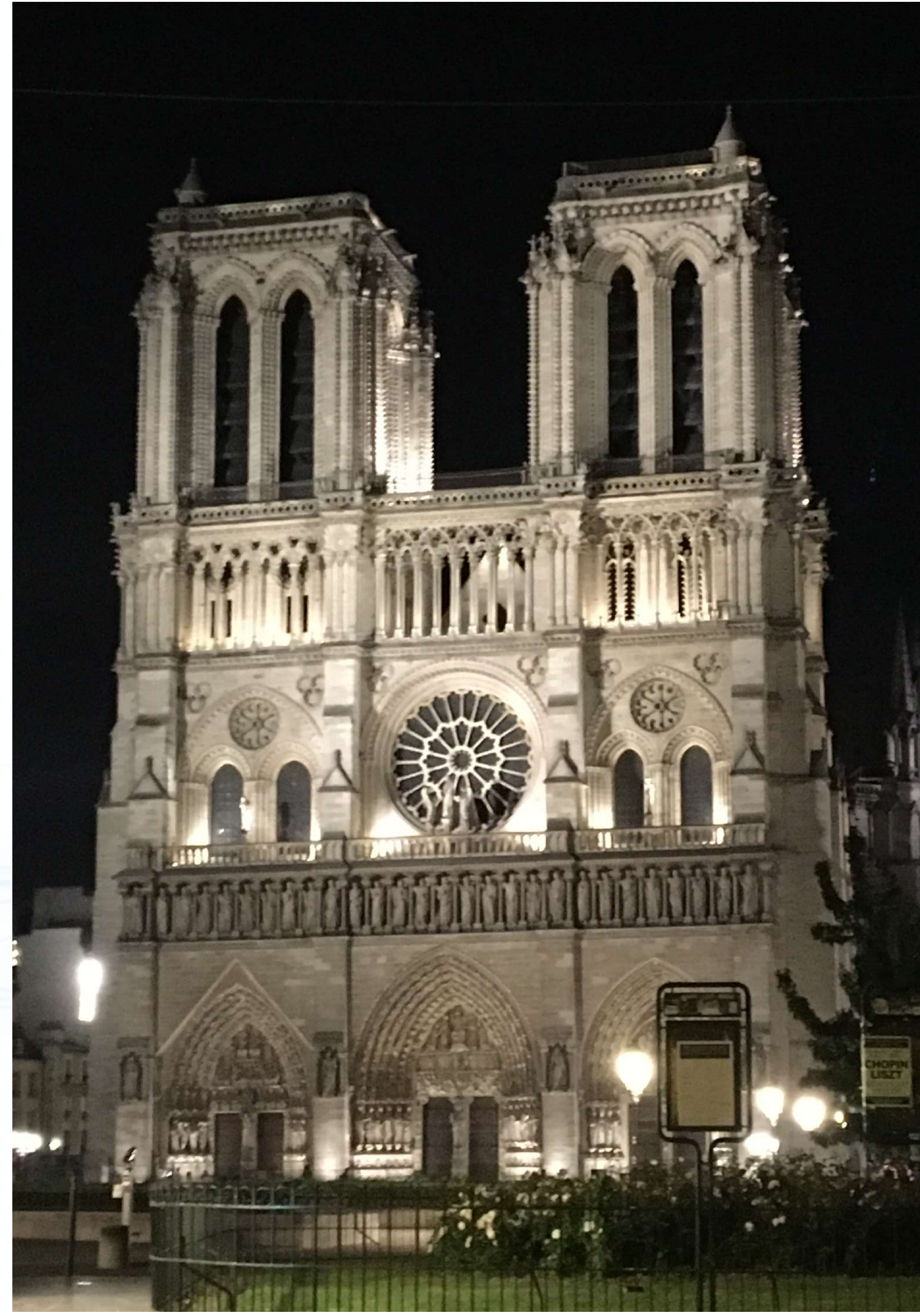
Great Architecture

Scales Development Teams

Delivers Quality

Enables High Performance / Low Cost

Supports Future Features Naturally



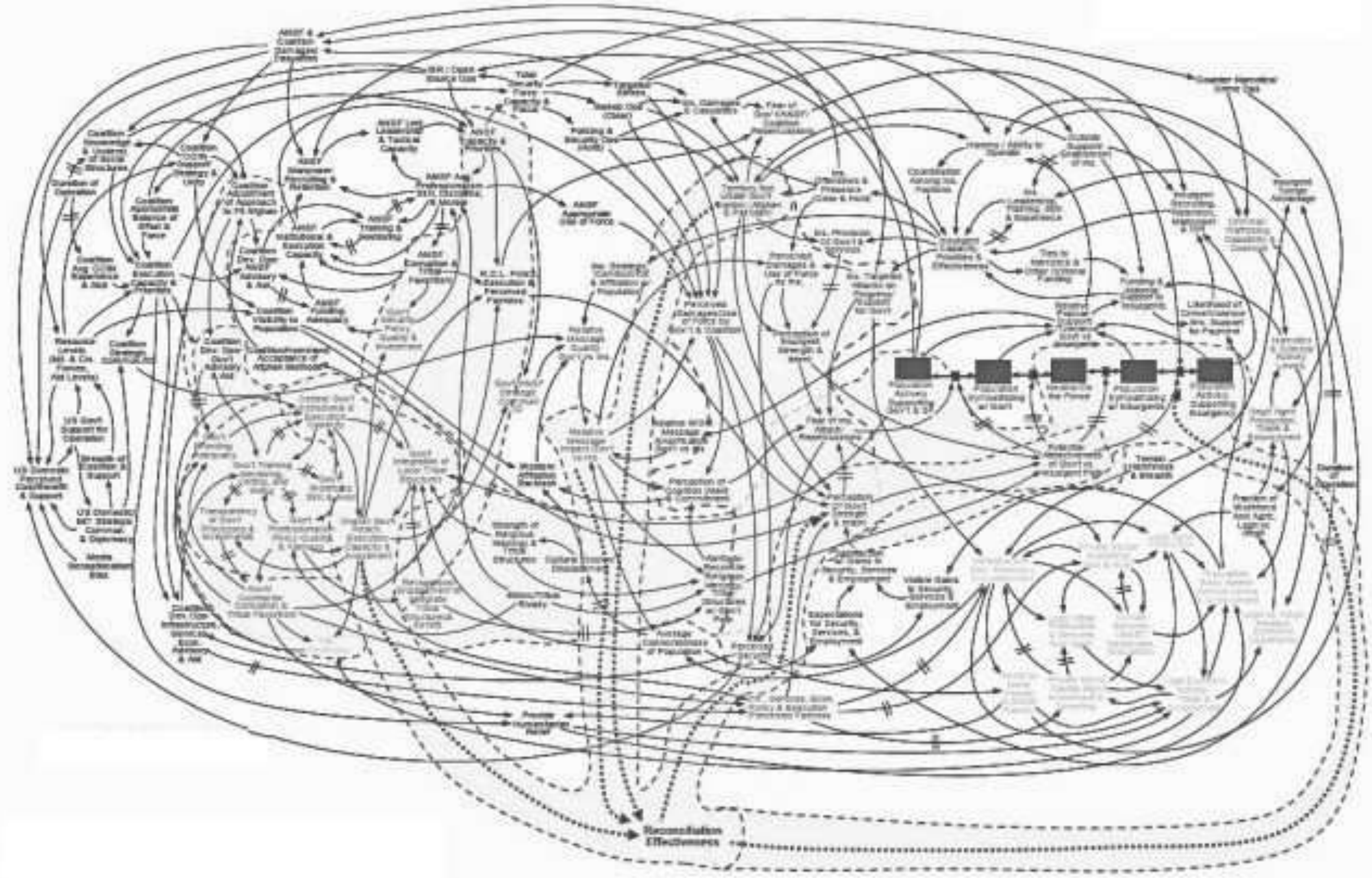
Not So Great Architecture

Tradeoffs:

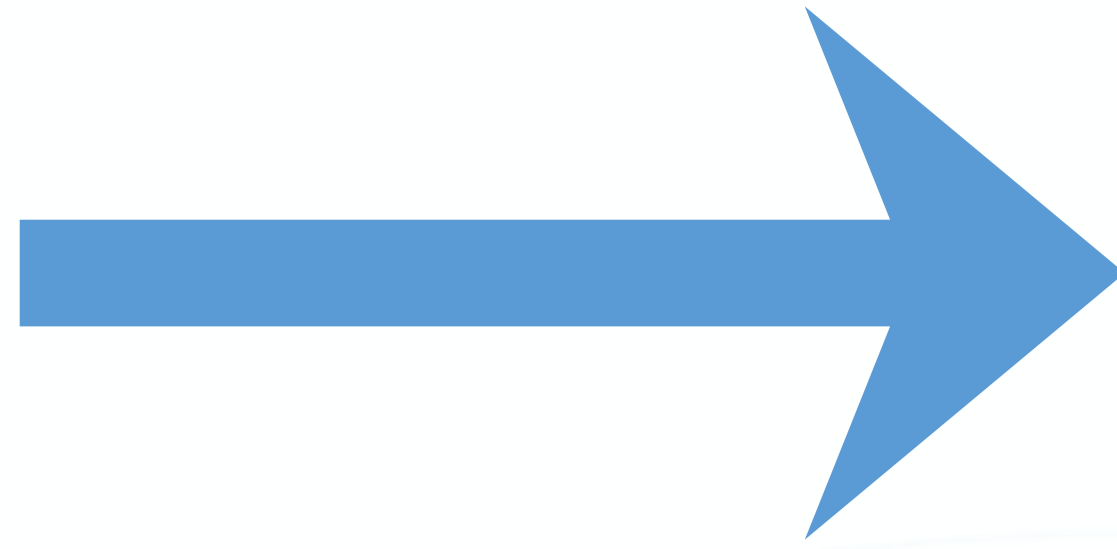
Near term velocity

Future Paralysis

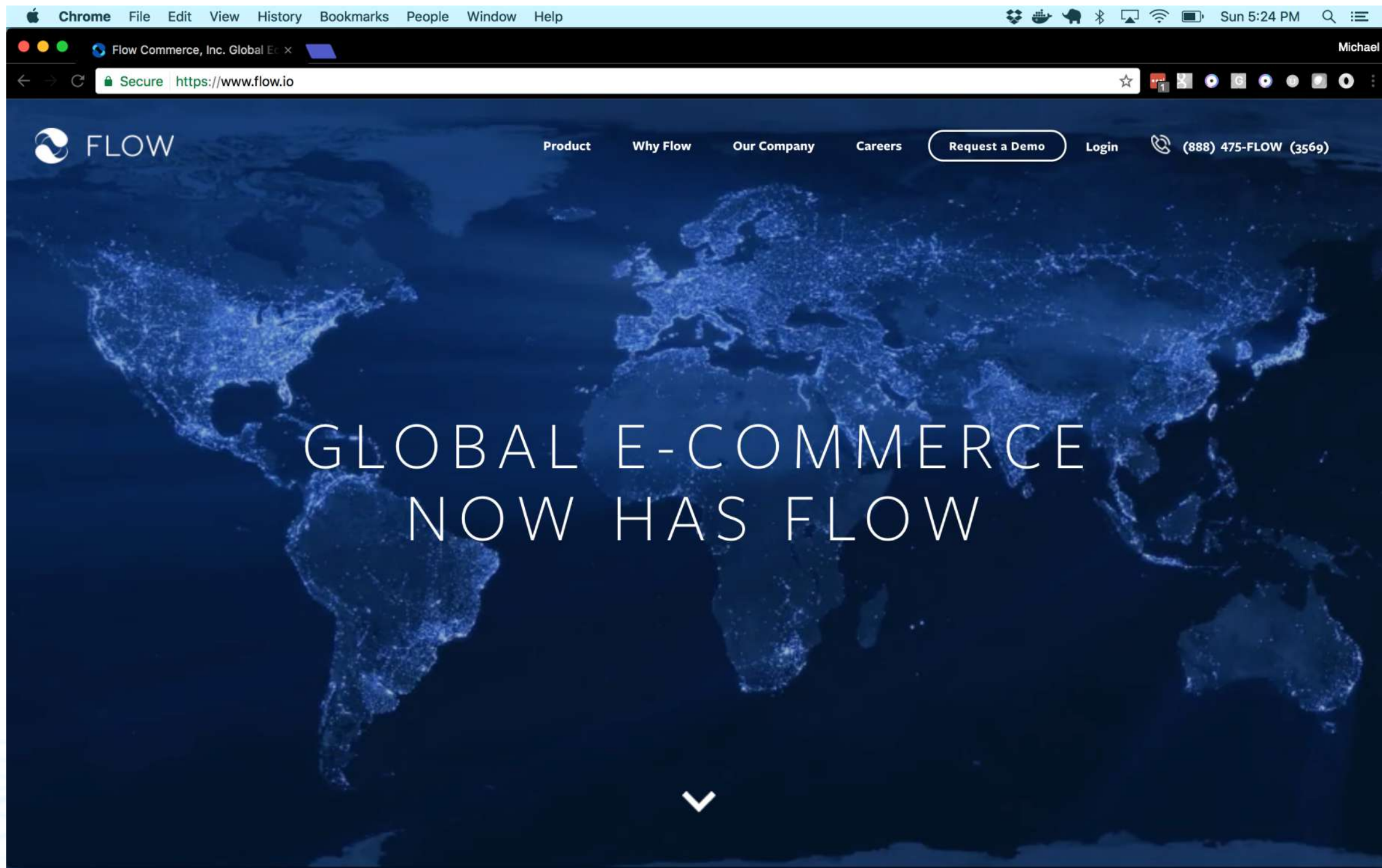
GOTO E SPAGHETTI CODE



Design Micro Service Architectures the Right Way

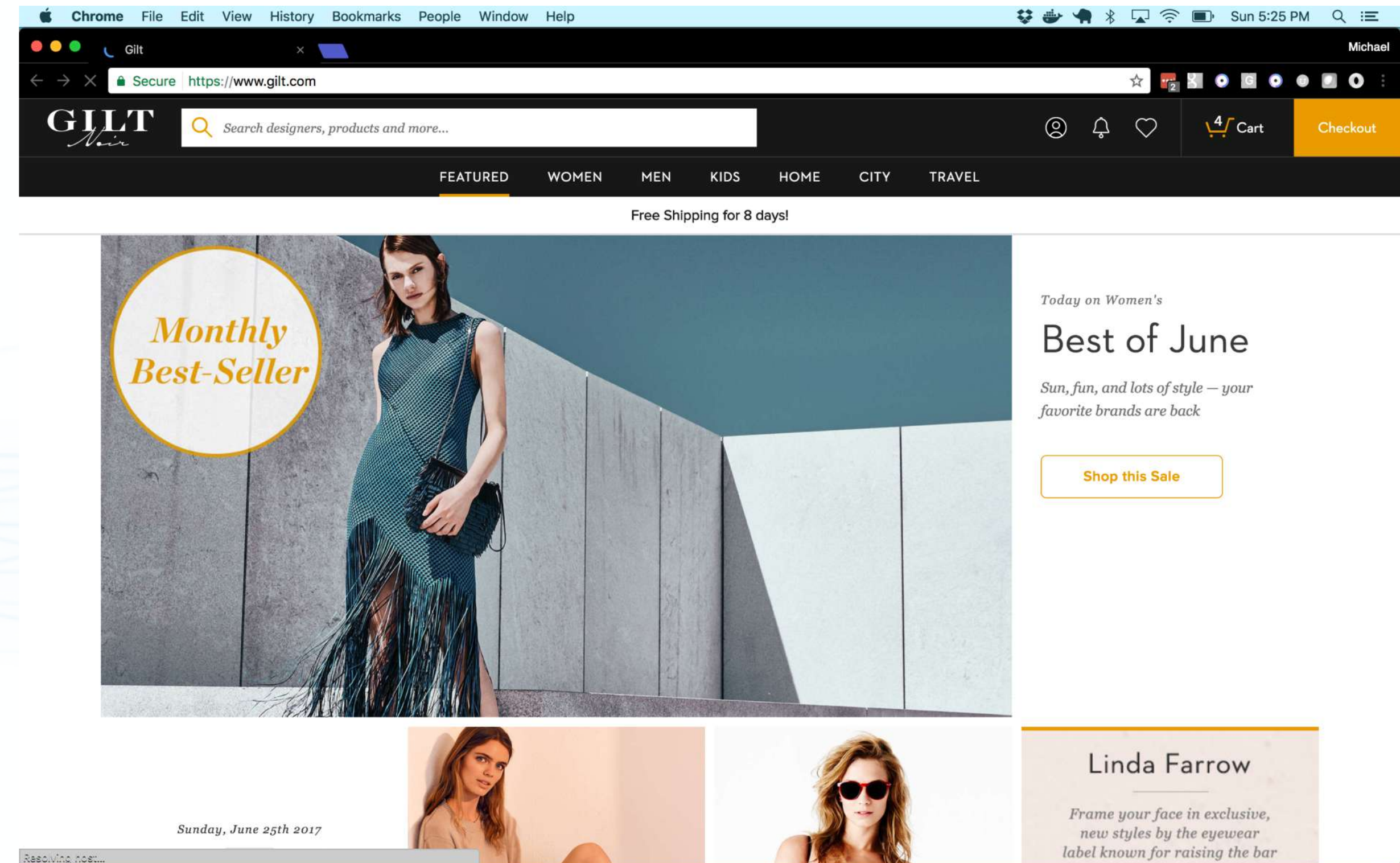


About Me



www.flow.io

www.gilt.com



**Let's Start
With a few
Misconceptions**

Misconception #1

Micro Services enable our teams to choose the best programming languages and frameworks for their tasks

Reality:

We'll demonstrate just how expensive this is.

Team size and investment are critical inputs.

Misconception #2

Code Generation is Evil

Reality:

What's important is creating a defined schema that is 100% trusted.

We'll demonstrate one technique leveraging code generation

Misconception #3

The Event Log Must be the Source of Truth

Reality:

Events are critical parts of an interface.

***But it's okay for services to be the
system of record for their resources.***

Misconception #4

Developers can maintain no more than 3 services each

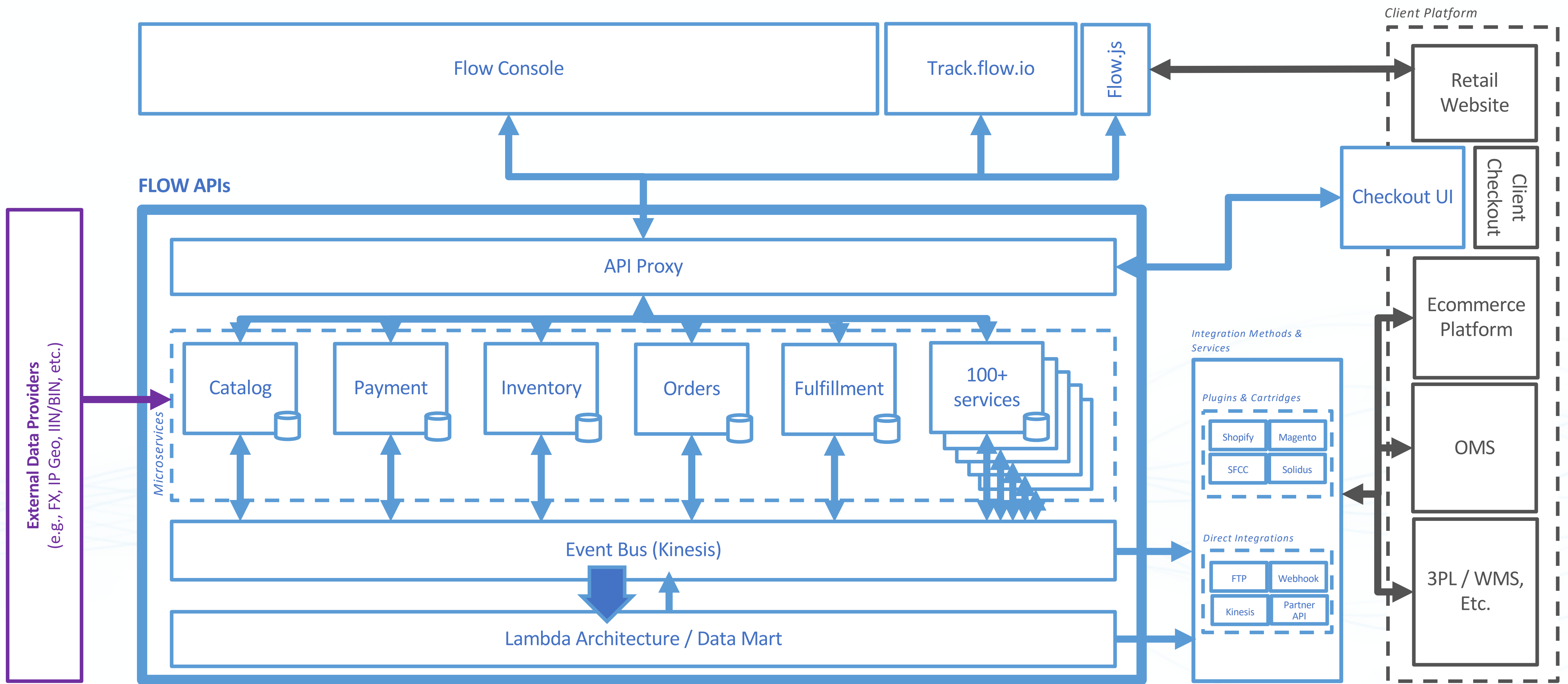
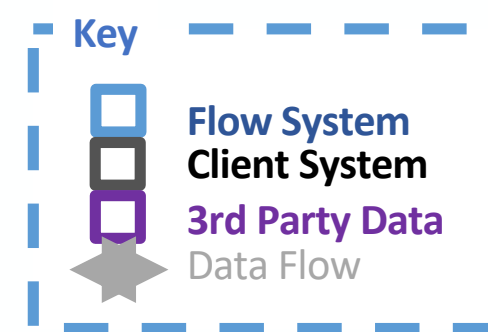
Reality:

Wrong metric; we'll demonstrate where automation shines.

Flow developers today each maintain ~5 services

Weekly maintenance <5% of time

Flow Platform Architecture



API Definition

Done correctly, even things like
GDPR compliance can be modeled

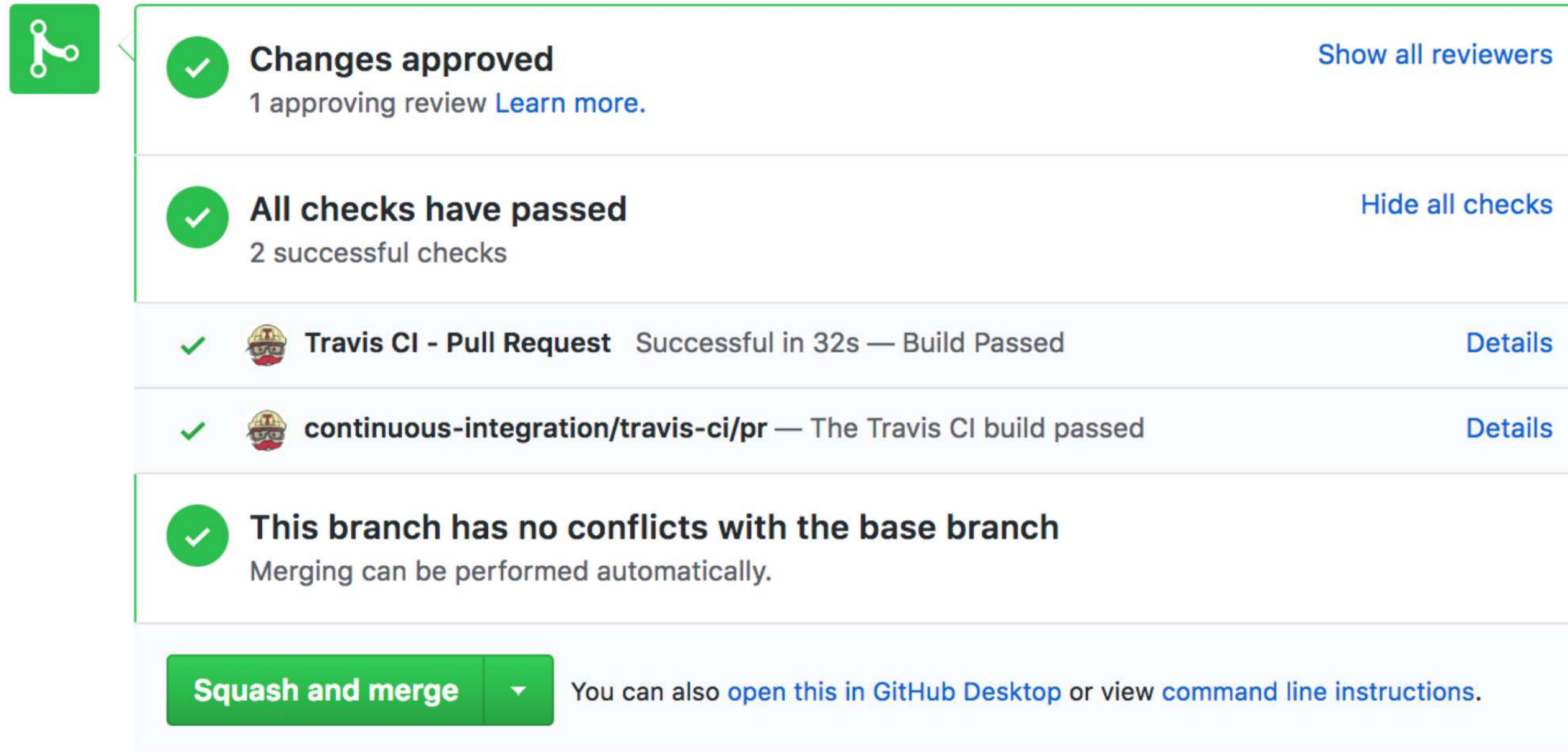
```
"user": {
  "description": "Represents a single user in the system",
  "fields": [
    { "name": "id", "type": "string" },
    { "name": "email", "type": "string", "required": false, "annotations": ["personal_data"] },
    { "name": "name", "type": "name", "annotations": ["personal_data"] },
    { "name": "status", "type": "user_status", "default": "active" }
  ]
},
```

```
"user_form": {
  "fields": [
    { "name": "email", "type": "string", "required": false, "annotations": ["personal_data"] },
    { "name": "password", "type": "string", "required": false, "annotations": ["personal_data"] },
    { "name": "name", "type": "name_form", "required": false, "annotations": ["personal_data"] }
  ]
},
```

Resource Oriented

```
"resources": {
  "io.flow.common.v0.models.user": {
    "operations": [
      {
        "method": "GET",
        "description": "Returns information about a specific user.",
        "path": "/:id",
        "responses": {
          "200": { "type": "io.flow.common.v0.models.user" },
          "401": { "type": "unit" },
          "404": { "type": "unit" }
        }
      },
      {
        "method": "POST",
        "description": "Create a new user. Note that new users will be created with a status of pending and will not be able
Flow team.",
        "body": { "type": "user_form" },
        "responses": {
          "201": { "type": "io.flow.common.v0.models.user" },
          "401": { "type": "unit" },
          "422": { "type": "io.flow.error.v0.models.generic_error" }
        }
      }
    ]
  }
}
```


Definitions in Git, with Continuous Integration



A screenshot of a GitHub pull request status summary. It features a green checkmark icon on the left. The summary is divided into several sections, each with a green checkmark icon and a title. The first section is 'Changes approved' with a link to 'Show all reviewers' and '1 approving review Learn more.'. The second section is 'All checks have passed' with a link to 'Hide all checks' and '2 successful checks'. The third section is 'Travis CI - Pull Request' with a Travis CI logo, 'Successful in 32s — Build Passed', and a link to 'Details'. The fourth section is 'continuous-integration/travis-ci/pr' with a Travis CI logo and 'The Travis CI build passed' and a link to 'Details'. The fifth section is 'This branch has no conflicts with the base branch' with 'Merging can be performed automatically.'. At the bottom, there is a green 'Squash and merge' button with a dropdown arrow and the text 'You can also open this in GitHub Desktop or view command line instructions.'

Changes approved [Show all reviewers](#)
1 approving review [Learn more.](#)

All checks have passed [Hide all checks](#)
2 successful checks

Travis CI - Pull Request Successful in 32s — Build Passed [Details](#)

continuous-integration/travis-ci/pr — The Travis CI build passed [Details](#)

This branch has no conflicts with the base branch
Merging can be performed automatically.

Squash and merge You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

```
547 =====  
548 Linter Starting  
549 =====  
550 catalog... Valid!  
551 currency... Valid!  
552 error... Valid!  
553 experience... Valid!  
554 export... Valid!  
555 feed... Valid!  
556 fraud... Valid!  
557 fulfillment... Valid!  
558 harmonization... Valid!  
559 healthcheck... Valid!  
560 import... Valid!  
561 inventory... Valid!  
562 item... Valid!  
563 jsonp... Valid!  
564 label... Valid!
```

Tests Enforce Consistency

```
/**
 * We keep all paths in lower case to avoid any issues with case
 * sensitivity.
 */
case object LowerCasePaths extends Linter with Helpers {

  override def validate(service: Service): Seq[String] = {
    service.resources.flatMap { resource =>
      resource.operations.
        filter( op => op.path != op.path.toLowerCase ).map { op =>
          error(resource, op, "Path must be all lower case")
        }
      }
    }
  }
}
```

Goal: It should feel like one person wrote the entire API

Tests Prevent Errors

Verify potentially breaking changes during API Design Phase

6/20/18	mbryzek	flow/api:0.5.60	Breaking	model consumer_invoice required field added: attributes
6/20/18	mbryzek	flow/api:0.5.60		model consumer_invoice_form optional field added: attributes
6/20/18	mbryzek	flow/api:0.5.60		resource consumer_invoice_url operation GET /consumer/invoice/tokens/:token/type/:type attribute added: linter
6/19/18	mbryzek	flow/api:0.5.59		resource consumer_invoice operation added: DELETE /:organization/consumer/invoices/:key
6/19/18	mbryzek	flow/api:0.5.59		resource consumer_invoice operation PUT /:organization/consumer/invoices/:key response added: 200
6/19/18	mbryzek	flow/api:0.5.58		enum added: consumer_invoice_document_type

API Implementation Supported by Code Generation

```
$ apibuilder update --app user
```

```
Fetching code from https://api.apibuilder.io
```

```
flow/user/latest/play_2_x_routes...
```

```
api/conf/user.routes: changed
```

```
flow/user/latest/play_2_6_client...
```

```
api/app/generated/FlowUserV0Client.scala: changed
```

```
flow/user/latest/play_2_6_mock_client...
```

```
api/test/generated/FlowUserV0MockClient.scala: changed
```

```
Copying updated code
```

```
- play_2_x_routes => /web/flowcommerce/user/api/conf/user.routes
```

```
- play_2_6_client => /web/flowcommerce/user/api/app/generated/FlowUserV0Client.scala
```

```
- play_2_6_mock_client => /web/flowcommerce/user/api/test/generated/FlowUserV0MockClient.scala
```

Code Generation: Routes

```
GET      /users/:id      controllers.Users.getById(id: String)
POST     /users          controllers.Users.post()
```

Guarantee that API operations are actually defined

User friendly paths

Consistent naming for methods

Code Generation: Client

```
467  override def post(  
468     userForm: io.flow.user.v0.models.UserForm,  
469     requestHeaders: Seq[(String, String)] = Nil  
470 )(implicit ec: scala.concurrent.ExecutionContext): scala.concurrent.Future[io.flow.common.v0.models.User] = {  
471     val payload = play.api.libs.json.Json.toJson(userForm)  
472  
473     _executeRequest("POST", s"/users", body = Some(payload), requestHeaders = requestHeaders).map {  
474         case r if r.status == 201 => {  
475             _root_.io.flow.user.v0.Client.parseJson(  
476                 "io.flow.common.v0.models.User", r, _.validate[io.flow.common.v0.models.User]  
477             )  
478         }  
479         case r if r.status == 401 => throw io.flow.user.v0.errors.UnitResponse(r.status)  
480         case r if r.status == 422 => throw io.flow.user.v0.errors.GenericErrorResponse(r)  
481         case r => throw io.flow.user.v0.errors.FailedRequest(  
482             r.status, s"Unsupported response code[${r.status}]. Expected: 201, 401, 422"  
483         )  
484     }  
485 }
```

Code Generation: Mock Client

```
98 def post(  
99     userForm: io.flow.user.v0.models.UserForm,  
100     requestHeaders: Seq[(String, String)] = Nil  
101 )(implicit ec: scala.concurrent.ExecutionContext): scala.concurrent.Future[io.flow.common.v0.models.User] = {  
102     scala.concurrent.Future.successful {  
103         io.flow.common.v0.mock.Factories.makeUser()  
104     }  
105 }
```

Mock and Client from Same Source

Enables High Fidelity, Fast Testing

Code Generation: Clients

The system of record IS the API specification

Code generation ensures that we actually adhere to the spec

Provide many common languages for our users

[<https://app.apibuilder.io/generators/>]

Now Let's Implement

```
def post() : Action[JsValue] = Anonymous.async(parse.json) { request =>
  Future {
    usersDao.create(request.user, request.body.as[UserForm]) match {
      case Left(errors) => {
        UnprocessableEntity(Json.toJson(Validation.errors(errors)))
      }
      case Right(newUser) => {
        Created(Json.toJson(newUser))
      }
    }
  }
}
```

Goal: Code we actually write is simple, consistent

Database Architecture

Each micro service application owns its database

No other service is allowed to connect to the database

Other services use only the service interface (API + Events)

Create a Database

CLI as single interface for infra and common development tasks

```
$ dev rds --app test
```

Confirm settings:

- db_name: testdb
- storage: 100
- db_instance_class: db.t2.medium
- db_instance_id: testdb20180623

Proceed? (y/n): █

Define storage requirements in metadata

Describe "scala" requirements

Describe "psql" requirements

Code generate the table definition

Code generate the Data Access Object

Note data tier independent from API,

Just uses same toolchain

```
"attributes": [  
  {  
    "name": "scala",  
    "value": {  
      "package": "db.generated",  
      "id_generator": {  
        "prefix": "usr"  
      }  
    }  
  },  
  {  
    "name": "psql",  
    "value": {  
      "pkey": "id",  
      "indexes": [  
        { "fields": ["email"] }  
      ]  
    }  
  }  
]
```

Code generation: Create a Table

```
create table users (  
  id          text primary key check(util.non_empty_trimmed_string(id)),  
  email       text check(util.null_or_non_empty_trimmed_string(email)),  
  first_name  text check(util.null_or_non_empty_trimmed_string(first_name)),  
  last_name   text check(util.null_or_non_empty_trimmed_string(last_name)),  
  status      text not null check(util.non_empty_trimmed_string(status)),  
  created_at  timestampz not null default now(),  
  updated_at  timestampz not null default now(),  
  updated_by_user_id text not null check(util.non_empty_trimmed_string(updated_by_user_id)),  
  hash_code   bigint not null  
);  
  
create index users_email_idx on users(email);  
  
select schema_evolution_manager.create_updated_at_trigger('public', 'users');
```

Consistent checks, metadata

Enable global features like 'hash_code' to minimize writes

Code generation: Scala Class

```
def insert(
  c: Connection,
  updatedBy: UserReference,
  form: UserForm
): String = {
  val id = randomId()
  bindQuery(InsertQuery, form).
    bind("id", id).
    bind("updated_by_user_id", updatedBy.id).
    anormSql.execute()
  id
}
```

```
def findAll(
  ids: Option[Seq[String]] = None,
  email: Option[String] = None,
  hasEmail: Option[Boolean] = None,
  limit: Long,
  offset: Long = 0,
  orderBy: OrderBy = OrderBy("users.id")
) (
  implicit customQueryModifier: Query => Query = { q => q }
): Seq[User] = {
```

Normalize access to DB

Ensure proper indexes exist from start

Automated Tests

```
def identifiedClient(
  user: UserReference = testUser
): Client = {
  new Client(
    wsClient,
    s"http://localhost:$port",
    defaultHeaders = authHeaders.headers(AuthHeaders.user(user))
  )
}
```

Use the generated mock client

Test Resource Operations

```
"GET /users/:id" in {  
  val user = createUser()  
  await(  
    identifiedClient().users.getById(user.id)  
  ) must equal(user)  
}
```

```
"GET /users/:id w/ invalid id returns 404" in {  
  expectNotFound(  
    identifiedClient().users.getById(UUID.randomUUID.toString)  
  )  
}
```

Use the generated mock clients to write simple tests

Time to Deploy

The bottom half of the slide features a decorative graphic consisting of several overlapping, wavy lines in a light blue color, creating a sense of motion or a modern, fluid design.

Continuous Delivery

“Continuous Delivery is a prerequisite to managing micro service architectures”

--@mbryzek



Continuous Delivery

Deploy triggered by a git tag

Git tags created automatically by a change on master (e.g. merge PR)

100% automated, 100% reliable



Continuous Delivery is Critical

delta	mbryzek@alum.mit.edu ▾	Search	Q
Dashboard	flow/demandware	13 hours ago	Running 0.3.97 (1)
Projects	flow/docs	1 day ago	Running 0.6.42 (2)
Event Log	flow/order-messenger	1 day ago	Running 0.1.30 (1)
Subscriptions	flow/return	1 day ago	Running 0.2.29 (2)
	flow/catalog	1 day ago	Running 0.10.17 (2)
	flow/dependency-www	1 day ago	Running 0.6.32 (2)
	flow/dependency-api	1 day ago	Running 0.6.32 (1)
	flow/inventory	1 day ago	Running 0.3.38 (2)

Auto Deploy on New Commit on Master

Build	Desired state last set	State
flow/user	seconds ago	Transitioning from 0.4.54 (2) to 0.4.55 (2)

Desired State

0.4.55

2 instances
Updated seconds ago

Last State

0.4.54

2 instances
Updated seconds ago

Microservice Infrastructure – keep it simple

```
1  builds:
2    - root:
3      instance.type: t2.small
4      port.container: 9000
5      port.host: 6021
6      version: 1.3
```

Standard Health Checks

```
$ curl --silent https://user.api.flow.io/_internal_/healthcheck | jq .  
{  
  "status": "healthy"  
}
```

```
"models": {  
  "healthcheck": {  
    "fields": [  
      { "name": "status", "type": "string", "example": "healthy" }  
    ]  
  }  
},  
  
"resources": {  
  "healthcheck": {  
    "path": "/_internal_",  
    "operations": [  
      {  
        "method": "GET",  
        "path": "/healthcheck",  
        "responses": {  
          "200": { "type": "healthcheck" },  
          "422": { "type": "io.flow.error.v0.models.generic_error" }  
        }  
      }  
    ]  
  }  
}
```

Events



**“We have an amazing API, but
please subscribe to our event
streams instead.”**

Principles of an Event Interface

First class schema for all events

Producers guarantee at least once delivery

Consumers implement idempotency

Flow:

- **End to end single event latency ~ 500 ms**

- **Based on postgresql – scaled to ~1B events / day / service**

Events: Approach

Producers:

- **Create a journal of ALL operations on table**
- **Record operation (insert, update, delete)**
- **On creation, queue the journal record to be published**
- **Real time, async, we publish 1 event per journal record**
- **Enable replay by simply requeuing journal record**

Events: Approach

Consumers:

- **Store new events in local database, partitioned for fast removal**
- **On event arrival, queue record to be consumed**
- **Process incoming events in micro batches (by default every 250ms)**
- **Record failures locally**

Events: Schema First

1 model / event

N events in one union type

1 union type / stream

Stream owned by 1 service

Most services define exactly 1 stream

```
"unions": {
  "user_event": {
    "discriminator": "discriminator",
    "types": [
      { "type": "user_upserted" },
      { "type": "user_deleted" }
    ]
  }
},

"models": {
  "user_upserted": {
    "fields": [
      { "name": "event_id", "type": "string" },
      { "name": "timestamp", "type": "date-time-iso8601" },
      { "name": "user", "type": "io.flow.common.v0.models.user" }
    ]
  },

  "user_deleted": {
    "fields": [
      { "name": "event_id", "type": "string" },
      { "name": "timestamp", "type": "date-time-iso8601" },
      { "name": "user", "type": "io.flow.common.v0.models.user" }
    ]
  }
}
```

Events: Schema Linter

```
/**
 * For event models (models ending with 'upserted', 'deleted'), validate:
 *
 * a. second field is timestamp
 * b. if 'organization', next
 * c. if 'number', next
 */
case object EventModels extends Linter with Helpers {

  override def validate(service: Service): Seq[String] = {
    service.models.
      filter(m => !ignored(m.attributes, "event_model")).
      filter(isEvent).
      flatMap(validateModel)
  }
}
```

Producers: Database Journal

Document retention period

Code generate journal

Use partitions to manage storage

```
"journal": {  
  "interval": "daily",  
  "retention": 3  
}
```

```
select journal.refresh_journaling('public', 'users', 'journal', 'users');  
select partman.create_parent('journal.users', 'journal_timestamp', 'time', 'daily');  
  
update partman.part_config  
  set retention = '3 day',  
      retention_keep_table = false,  
      retention_keep_index = false  
where parent_table = 'journal.users';
```

Producers: Streams

```
private[this] val stream = queue.producer[UserEvent]()
```

Reflection used to generate stream name

`io.flow.v0.user_event.json`

Producing an Event

```
override def process(record: UserVersion)(
  implicit ec: ExecutionContext
): Unit = {
  record.journalOperation match {
    case ChangeType.Insert | ChangeType.Update => {
      stream.publish(
        UserUpserted(
          eventId = eventIdGenerator.randomId(),
          timestamp = DateTime.now,
          user = record.userVersion.user
        )
      )
    }
    case ChangeType.Delete => {
      stream.publish(
        UserDeleted(
          eventId = eventIdGenerator.randomId(),
          timestamp = DateTime.now,
          id = record.userVersion.user.id
        )
      )
    }
  }
}
```

Note the UserVersion class which is also code generated.

Guarantees that all code in all services looks the same.

Producers: Testing

```
"publishes event on user creation" in {  
  val user = createUser()  
  
  eventuallyInNSeconds() {  
    stream.all.  
      flatMap(_.js.asOpt[UserUpserted]).  
      find { e =>  
        e.id == user.id && e.email == user.email && e.name == Name()  
      }.  
      get  
    }  
  }  
}
```

Consumers: Processing Incoming Events

```
override def process(json: JsValue) = {  
  json.as[UserEvent] match {  
    case UserUpserted(_, _, user) => {  
      usersDao.upsertById(  
        Constants.SystemUser,  
        UserForm(  
          id = user.id,  
          email = user.email,  
          firstName = user.name.first,  
          lastName = user.name.last  
        )  
      )  
    }  
  
    case UserDeleted(_, _, user) => {  
      usersDao.deleteById(Constants.SystemUser, user.id)  
    }  
  }  
}
```

Consumers: Testing

```
val event = Factories.makeUserUpserted()
producer.publish(event)

val user = eventuallyInThreeSeconds {
    userDao.findById(event.user.id).get
}
```

Factories classes generated from the API Spec

Dependencies

Keeping things up to date

The bottom of the slide features a decorative graphic consisting of several overlapping, wavy, light blue lines that create a sense of motion and depth.

Dependencies

Goal: Automatically update all services to latest dependencies

- **Critical for security patches / bug fixes in core libraries**
- **Takes hours (not weeks or months)**
- **Same process for internally developed libraries and open source**
- **Flow: we upgrade all services every week to latest dependencies**

Dependencies: Tracking

dependency mbryzek@alum.mit.edu

Dashboard

- Projects
- Libraries
- Binaries
- Resolvers
- Subscriptions

Recommended Upgrades

Date	Project	Name	Current	Recommended
6/24/18	user	io.flow.lib-validation	0.0.17	0.0.18
6/24/18	beacon	io.flow.lib-validation	0.0.17	0.0.18
6/24/18	billing	io.flow.lib-validation	0.0.17	0.0.18
6/24/18	feature	io.flow.lib-validation	0.0.17	0.0.18
6/23/18	link	io.flow.lib-play-play26	0.4.73	0.4.74
6/23/18	location	io.flow.lib-play-play26	0.4.73	0.4.74
6/23/18	secret	io.flow.lib-play-play26	0.4.73	0.4.74

Recommendations

Name	Current	Recommended
io.flow.lib-validation	0.0.17	0.0.18

Dependencies

Name	Version	File
sbt	1.1.5	project/build.properties
scala	2.12.6	build.sbt

Library	Version	Cross built for	File
com.gilt.sbt.sbt-newrelic	0.2.4	-	project/plugins.sbt
com.typesafe.play.play-json	2.6.9	2.12	build.sbt
com.typesafe.play.play-json-joda	2.6.9	2.12	build.sbt
com.typesafe.play.sbt-plugin	2.6.15	-	project/plugins.sbt
io.flow.lib-event-sync-play26	0.2.92	2.12	build.sbt


<https://dependency.flow.io/>

<https://github.com/flowcommerce/dependency>

Dependencies: Updating

```
amm scripts/upgrade/upgrade.sc
```

🔗 112 Open	✓ 28,098 Closed	Visibility ▾	Organization ▾	Sort ▾
🔗	flowcommerce/session	Upgrade sbt to 1.1.5, scala to 2.12.6, play to 2.6.13 ●		
	#276	opened 5 minutes ago by mbryzek		
🔗	flowcommerce/lib-postgresql	Upgrade sbt to 1.1.5, scala to 2.12.6, play to 2.6.13 ✓		
	#88	opened 5 minutes ago by mbryzek		
🔗	flowcommerce/lib-event-sync	Upgrade sbt to 1.1.5, scala to 2.12.6, play to 2.6.13 ●		
	#296	opened 5 minutes ago by mbryzek		
🔗	flowcommerce/installment	Upgrade sbt to 1.1.5, scala to 2.12.6, play to 2.6.13 ●		
	#120	opened 6 minutes ago by mbryzek		
🔗	flowcommerce/lib-event	Upgrade sbt to 1.1.5, scala to 2.12.6, play to 2.6.13 ●		
	#339	opened 6 minutes ago by mbryzek		
🔗	flowcommerce/lib-csv	Upgrade sbt to 1.1.5, scala to 2.12.6, play to 2.6.13 ●		
	#17	opened 6 minutes ago by mbryzek		
🔗	flowcommerce/lib-logistics	Upgrade sbt to 1.1.5, scala to 2.12.6, play to 2.6.13 ●		
	#49	opened 6 minutes ago by mbryzek		
🔗	flowcommerce/lib-experience-picker	Upgrade sbt to 1.1.5, scala to 2.12.6, play to 2.6.13 ●		
	#66	opened 6 minutes ago by mbryzek		
🔗	flowcommerce/lib-query	Upgrade sbt to 1.1.5, scala to 2.12.6, play to 2.6.13 ●		
	#140	opened 6 minutes ago by mbryzek		
🔗	flowcommerce/lib-play	Upgrade sbt to 1.1.5, scala to 2.12.6, play to 2.6.13 ✓		
	#217	opened 6 minutes ago by mbryzek		



Dependencies & Continuous Delivery

Deploy Once the Build Passes



Summary: Critical Decisions

- 💡 **Design schema first for all APIs and Events**
 - consume events (not API) by default
- 💡 **Invest in automation**
 - deployment, code generation, dependency management
- 💡 **Enable teams to write amazing and simple tests**
 - drives quality, streamlines maintenance, enables continuous delivery



FLOW

Thank You!

Go forth and Design Micro Service Architectures the Right Way

Michael Bryzek

mike@flow.io / @mbryzek

Cofounder / CTO Flow

Cofounder / ex-CTO Gilt



We're hiring: <https://www.flow.io/careers>