BUILDING MICROSERVICE ARCHITECTURE FOR SCALE WITH DJANGO

A BRIEF HISTORY OF US

Syed Ansab Waqar Gillani

Software Engineer

Syed Muhammad Dawoud Sheraz Ali

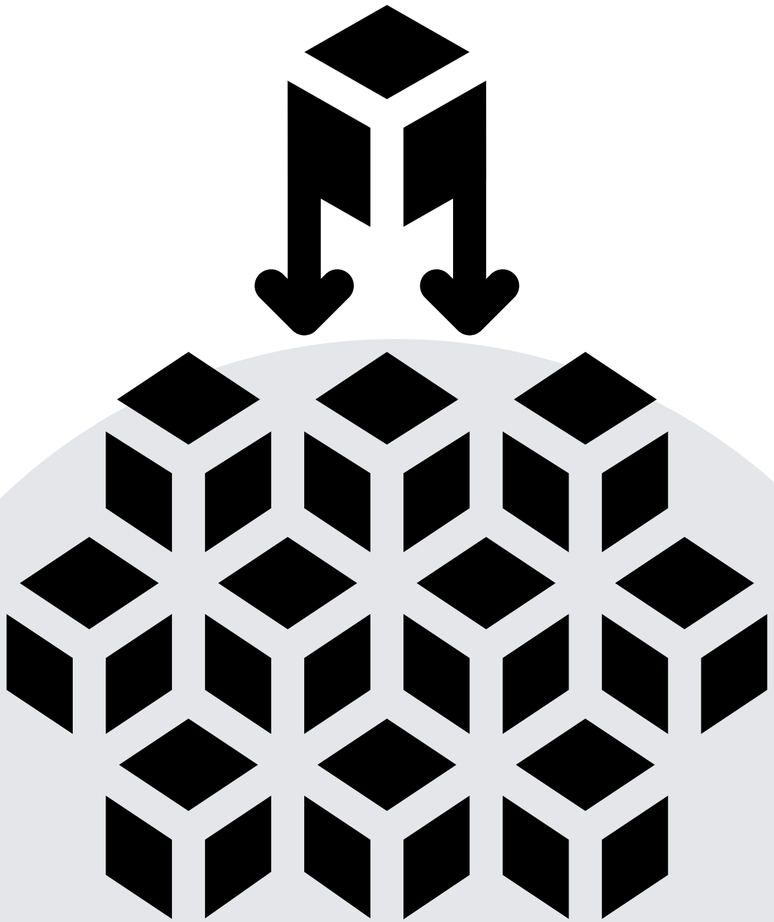
Software Engineer

ABOUT OUR COMPANY

"The amount of dedication, commitment, energy and loyalty, that a vast majority of, now nearly 900, people at this company bring to their work, leaves me in awe every single day. If you have a great team, no failure (or success) matters too much!"

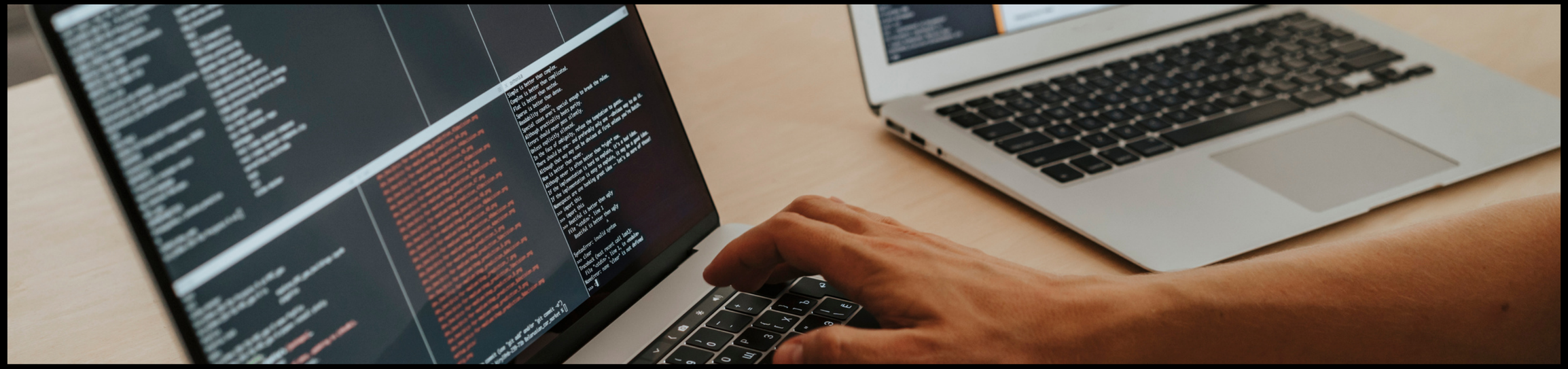
- Yasser Bashir

OPEN edX arbisoft



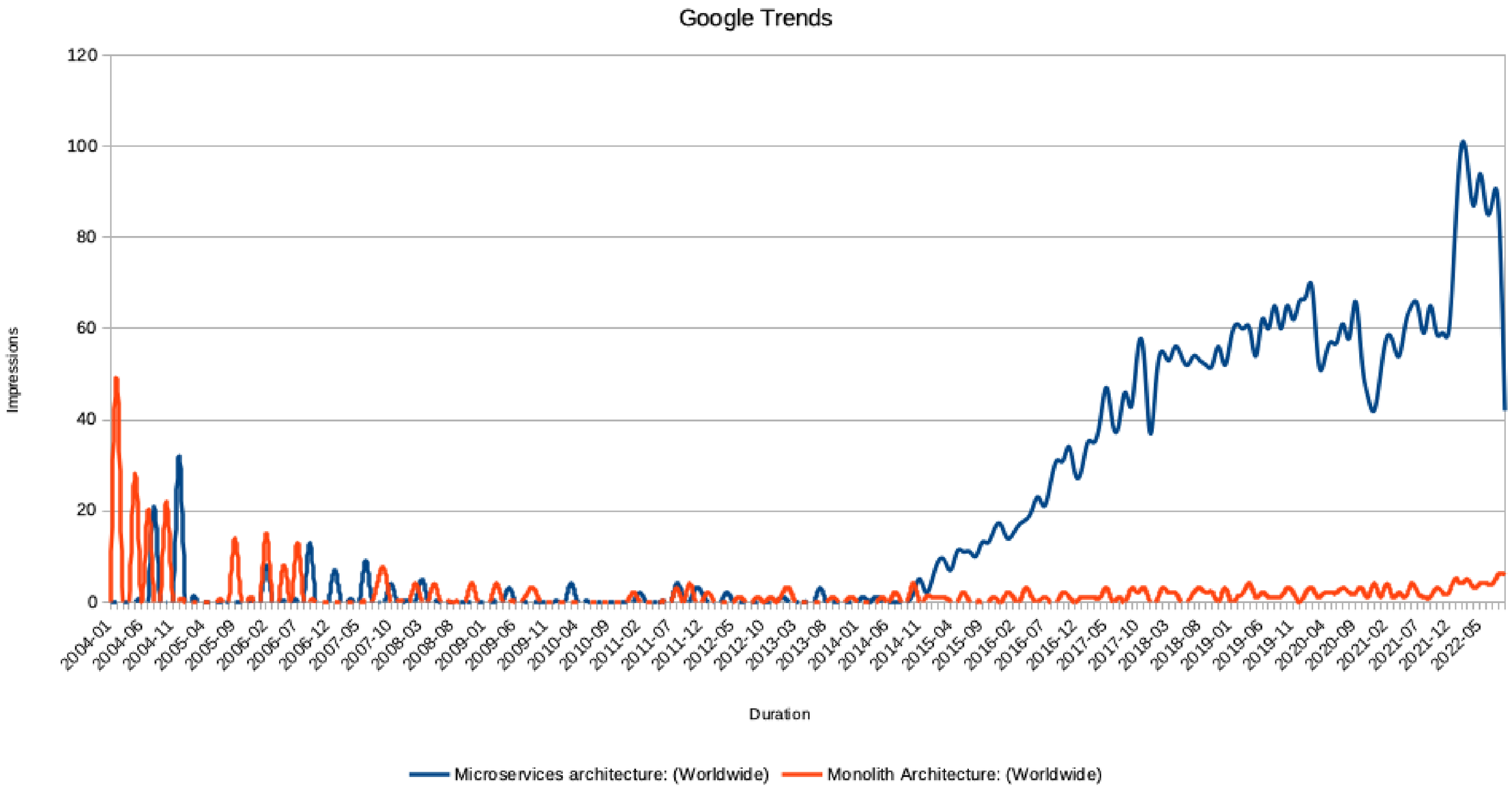
OUR AGENDA

- Monolith and Microservices Architectures
- Building Microservices
- Scaling Microservices
- Case Study: Open edX
- Q&A



MONOLITH & MICROSERVICE ARCHITECTURE

What makes each pattern unique?


Google Trends
Impressions
120
100
80
60
40
20
0
2004-01
2004-06
2004-11
2005-04
2005-09
2006-02
2006-07
2006-12
2007-05
2007-10
2008-03
2008-08
2009-01
2009-06
2009-11
2010-04
2010-09
2011-02
2011-07
2011-12
2012-05
2012-10
2013-03
2013-08
2014-01
2014-06
2014-11
2015-04
2015-09
2016-02
2016-07
2016-12
2017-05
2017-10
2018-03
2018-08
2019-01
2019-06
2019-11
2020-04
2020-09
2021-02
2021-07
2021-12
2022-05
Duration
Microservices architecture: (Worldwide)
Monolith Architecture: (Worldwide)

MONOLITH ARCHITECTURE

The monolithic architecture is considered to be a traditional way of building applications. A monolithic application is built as a single and indivisible unit. Usually, such a solution comprises a client-side user interface, a server side-application, and a database, unified, managed and served in one place.

What?

A monolithic architecture is a singular, large computing network with one code base that couples all of the business concerns together

How?

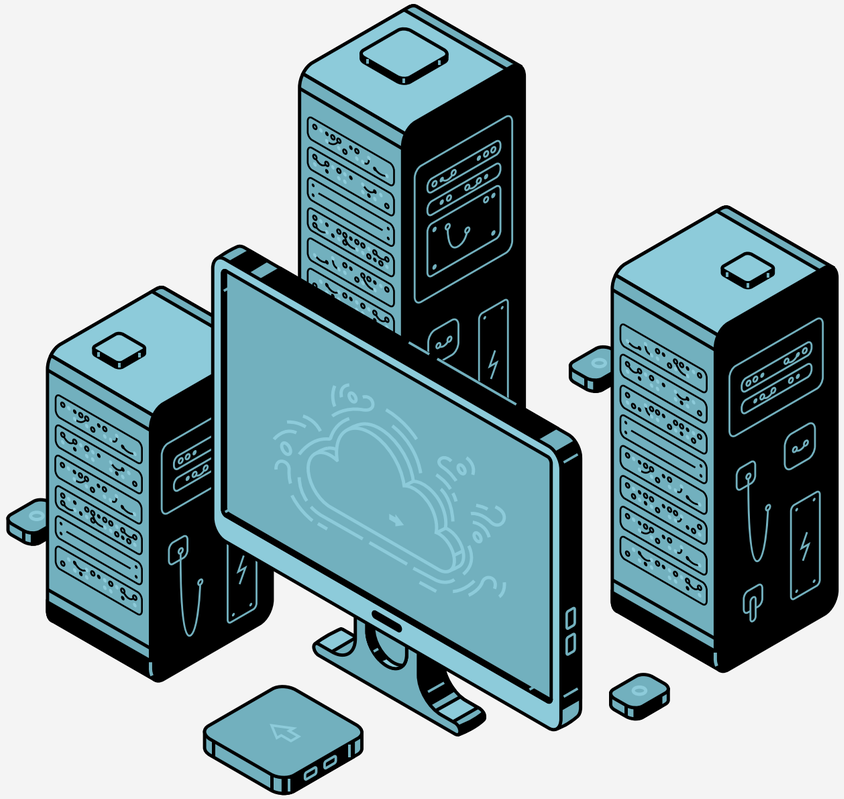
Normally, monolithic applications have one large code base and lack modularity. If developers want to update or change something, they access the same code base.

Why?

- Simple to develop
- Simple to deploy
- Simple to scale
- Simple to test
- Simple to debug

Why not?

- Slower development speed
- Reliability
- Barrier to technology adoption
- Lack of flexibility

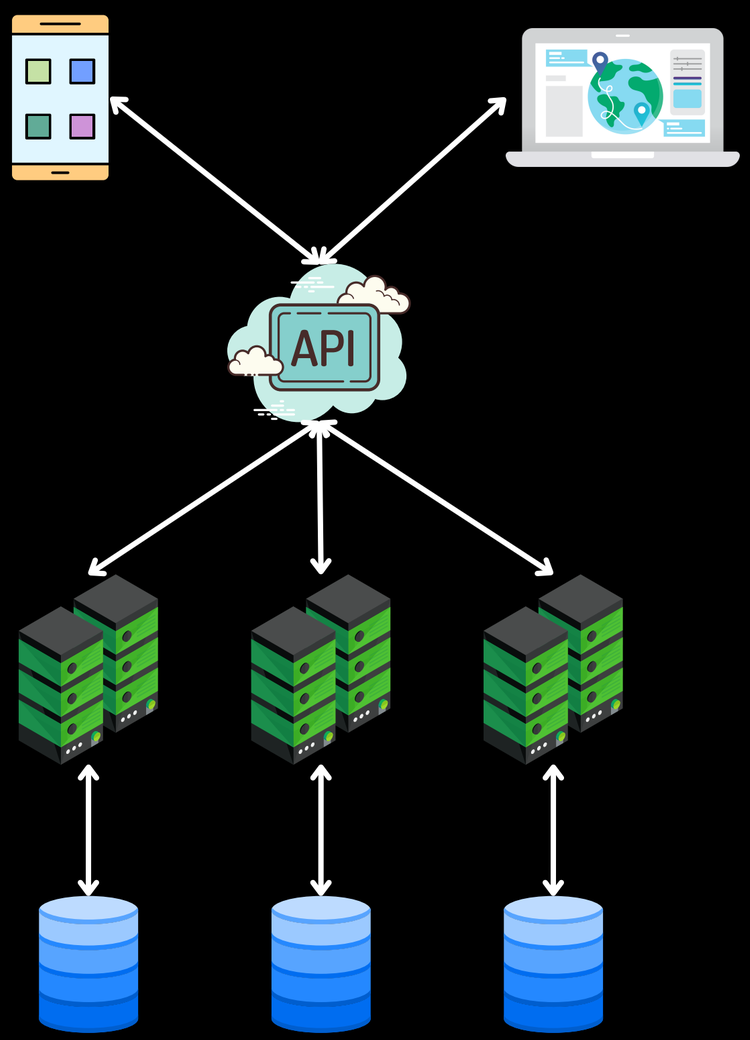



And this is what happens!!!

MICROSERVICES ARCHITECTURE

A microservices architecture consists of a collection of small, autonomous services. Each service is self-contained and should implement a single business capability within a bounded context. A bounded context is a natural division within a business and provides an explicit boundary within which a domain model exists.

Services communicate using either synchronous protocols such as HTTP/REST or asynchronous protocols such as AMQP. Services can be developed and deployed independently of one another. Each service has its own database in order to be decoupled from other services. Data consistency between services is maintained using the Saga pattern



SOME INDUSTRIAL EXAMPLES

Netflix
Pioneer of Monolith to Microservices migration
Migration period: 2009-2011
Over 1000 microservices

Spotify
The tipping point at 75M+ active users
Decoupled to about 800 services

Atlassian
Migrated in 2018
Motivated by scaling challenges of Confluence & Jira
Codenamed Project Vertigo

Amazon
Monolith arch 2001
Deployment and development challenges
Amazon’s re:Invent 2015 conference in Las Vegas

So, in summary...

01 Highly maintainable and testable

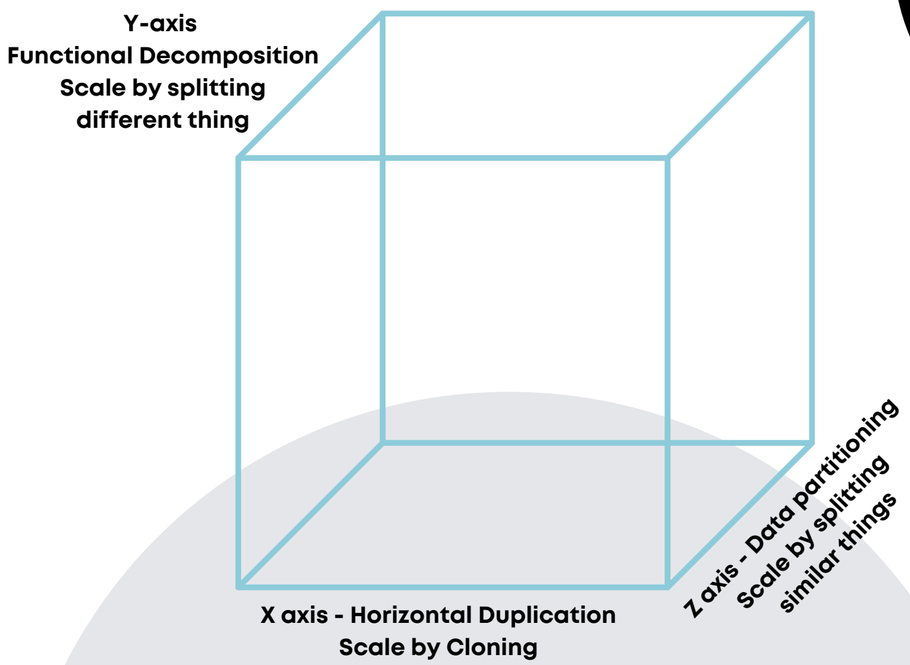
02 Loosely coupled

03 Independently deployable

04 Organized around business capabilities

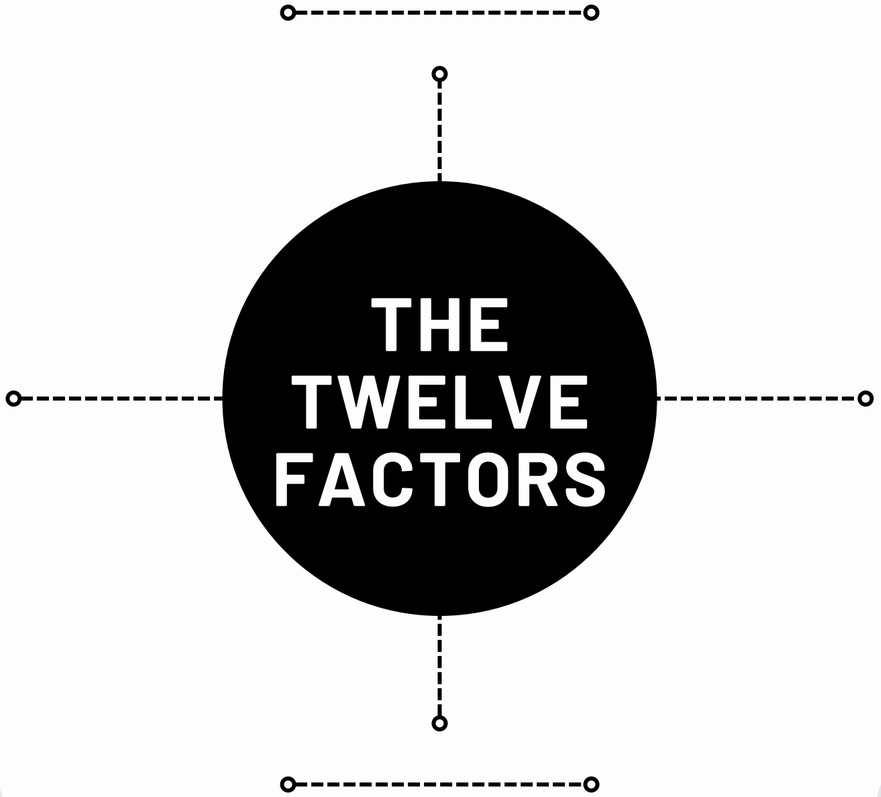
05 Owned by a small team

06 Rapid, frequent and reliable delivery of large, complex applications



MICROSERVICE PRINCIPLES

Defining architectural principles with microservices



01

Codebase

One codebase tracked in revision control, many deploys

02

Dependencies

Explicitly declare and isolate dependencies

03

Config

Store config in the environment

04

Backing services

Treat backing services as attached resources

05

Build, release, run

Strictly separate build and run stages

06

Processes

Execute the app as one or more stateless processes

07

Port binding

Export services via port binding

08

Concurrency

Scale out via the process model

09

Disposability

Maximize robustness with fast startup and graceful shutdown

10

Dev/prod parity

Keep development, staging, and production as similar as possible

11

Logs

Treat logs as event streams

12

Admin processes

Run admin/management tasks as one-off processes

SOLID Principles

These principles establish practices that lend to developing software with considerations for maintaining and extending as the project grows. Adopting these practices can also contribute to avoiding code smells, refactoring code, and Agile or Adaptive software development.

Single-Responsibility Principle

A service should have one and only one reason to change, meaning that a service should have only one job.

Open-Closed Principle

Entities should be open for extension but closed for modification.

Liskov Substitution Principle

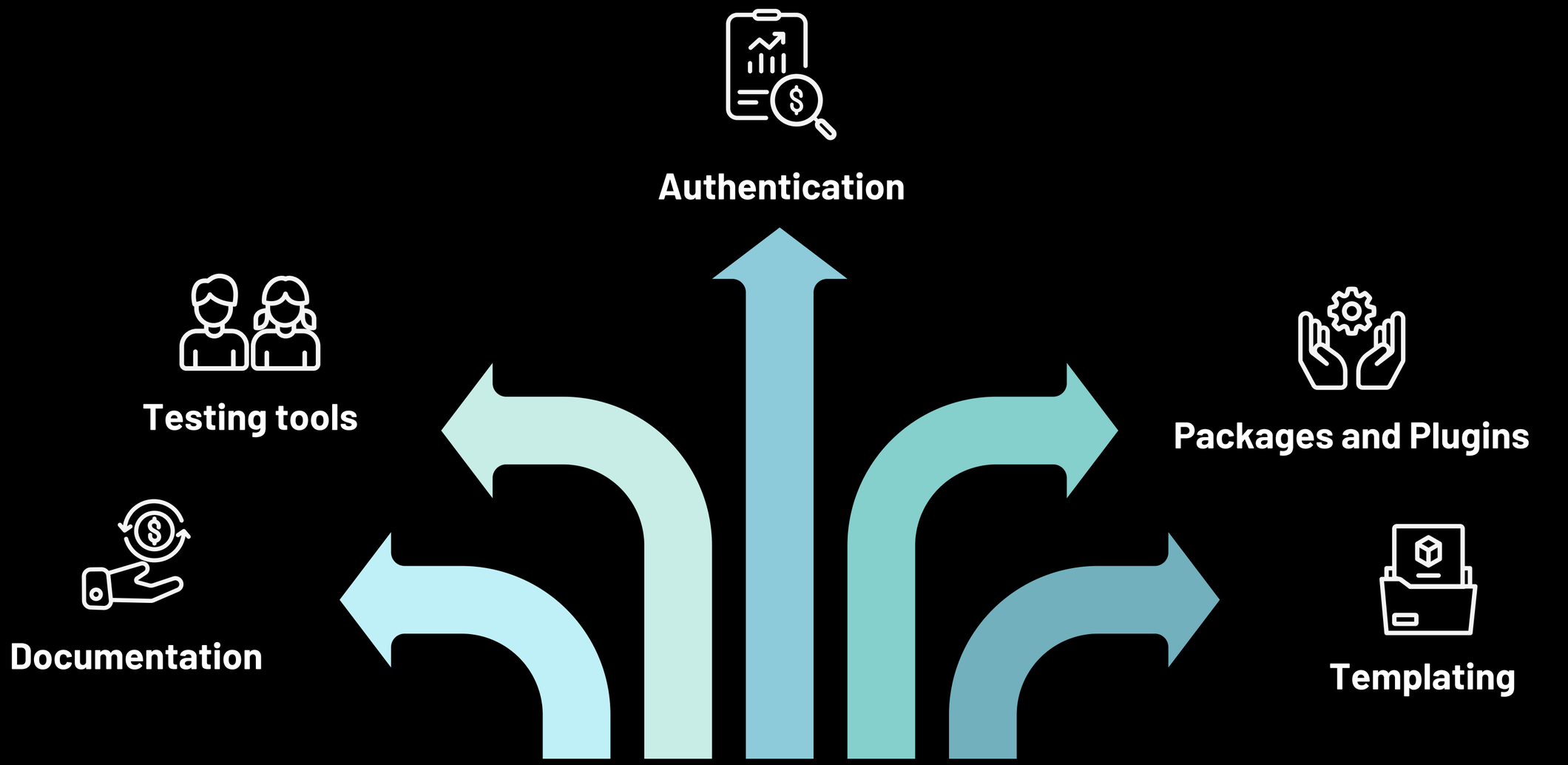
Every subclass or derived class should be substitutable for their base or parent class.

Interface Segregation Principle

A client should never be forced to implement an interface that it doesn't use, or clients shouldn't be forced to depend on methods they do not use.

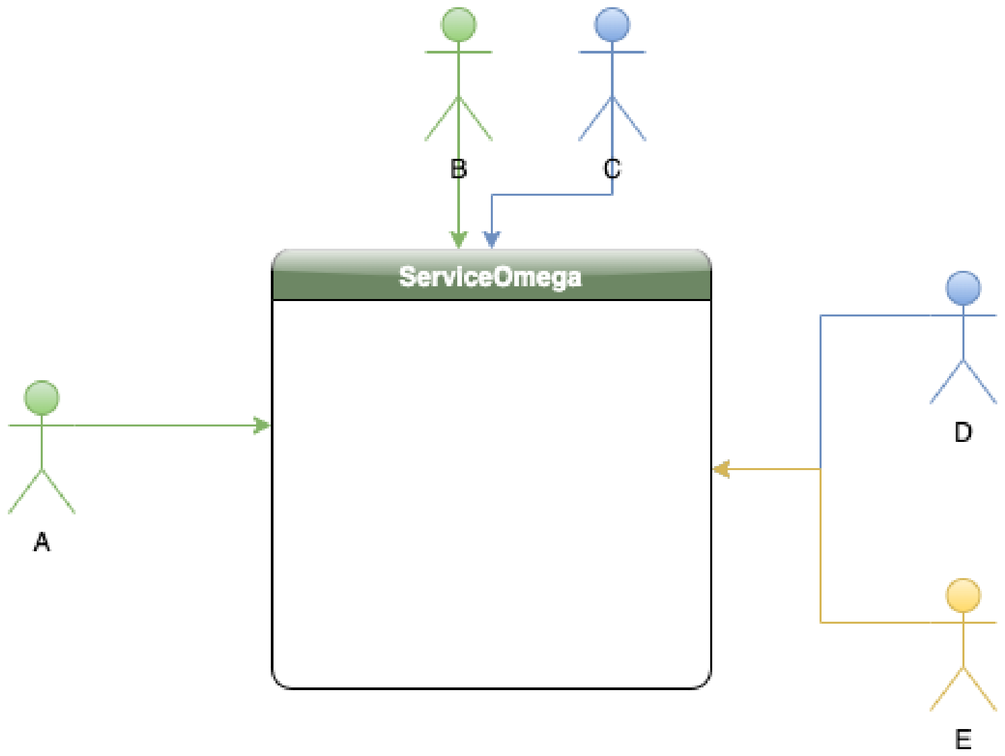
Dependency Inversion Principle

Entities must depend on abstractions, not on concretions. High-level modules must not depend on the low-level modules, but on abstractions.



Scaling Microservices

Authentication with Monoliths



Easier Implementation

Easy to handle user specific authentication cases

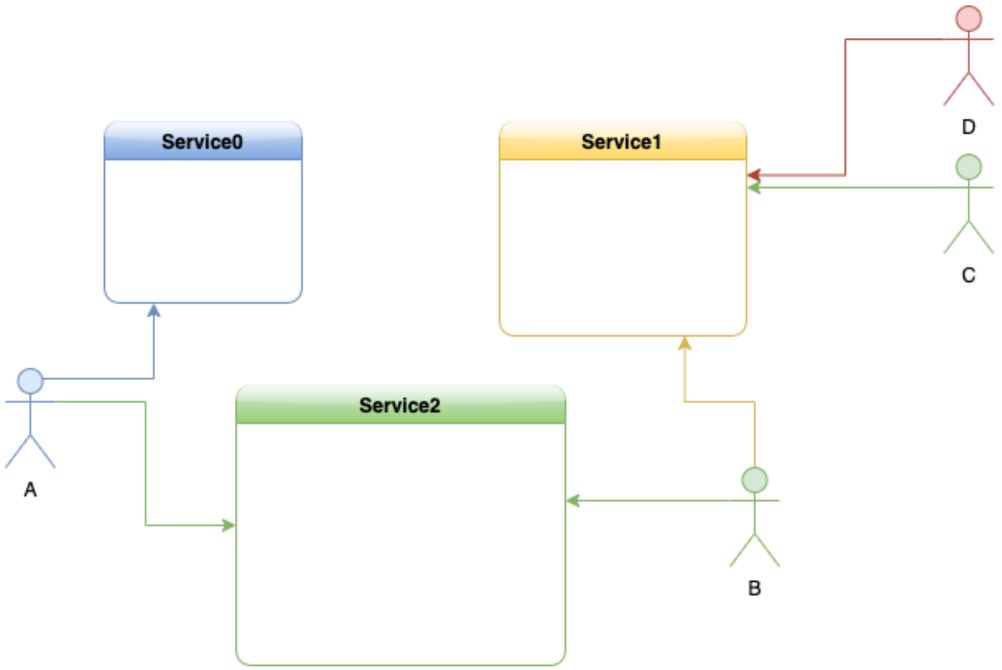
Single gateway

All type of users access the login from same source

Lesser Complications

No need to manage various authentication tokens

Authentication with Microservices



Complicated Implementation

Who is source of truth?

Seperate gateway

Users accessing the services that they need to access

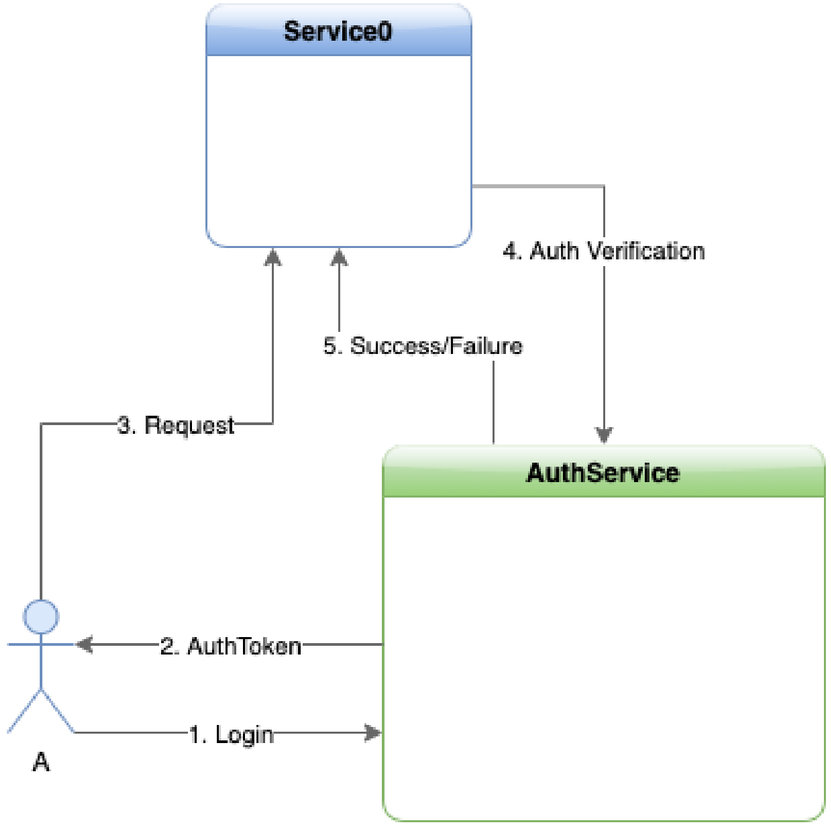
Authenticating a single user across multiple services

How to verify an authentication user from one service in another service

Question to Ponder?

- Single or Multiple sources of auth
- Account Duplication across services
- Decorating authentication with system-specific roles
- Uniform authentication vs Service type specific authentication

Authentication with Django / DRF



Authentication Types

Django and DRF typically allow a variety of authentication methods:

- Session
- Bearer
- JWT

User-level permissions

Ability to add user-specific roles to auth type during authentication

Packages and Plugins

Importing utilities across microservices

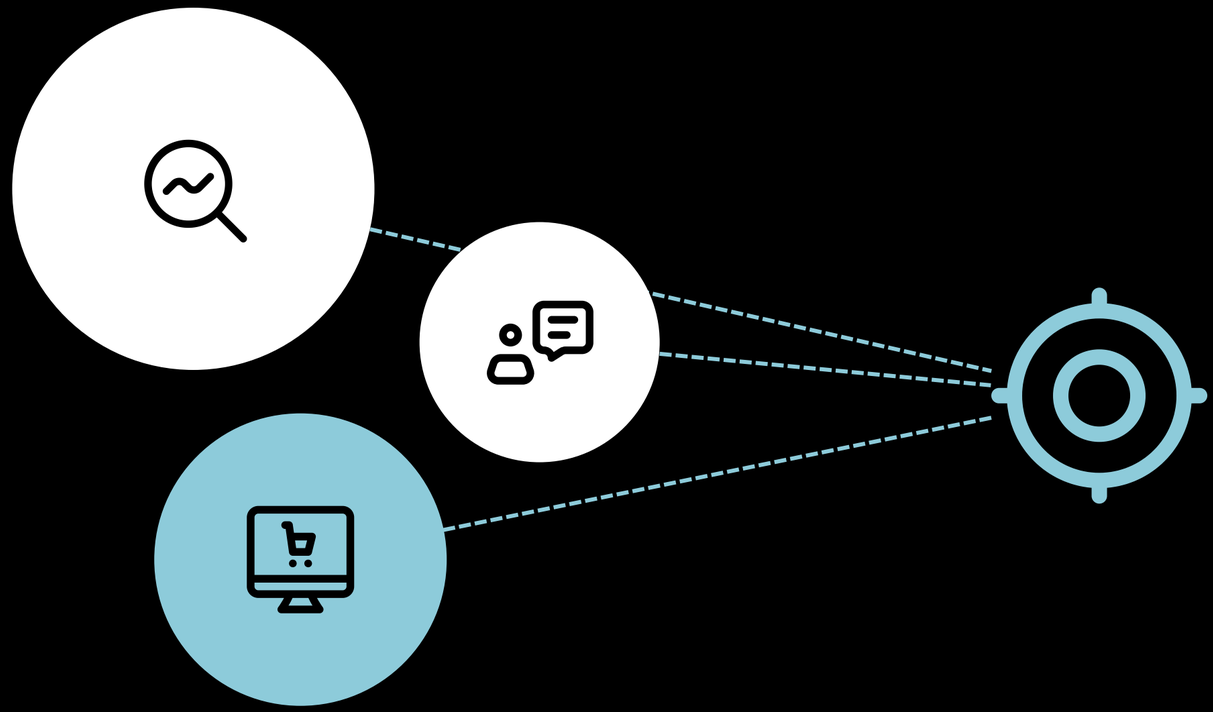
How to use same util across different services?

Code Duplication

Duplication of the util in each service?

Violation of the DRY Principle

How to manage the changes in util across services?



Packages and Plugins

Solution

Creating Packages

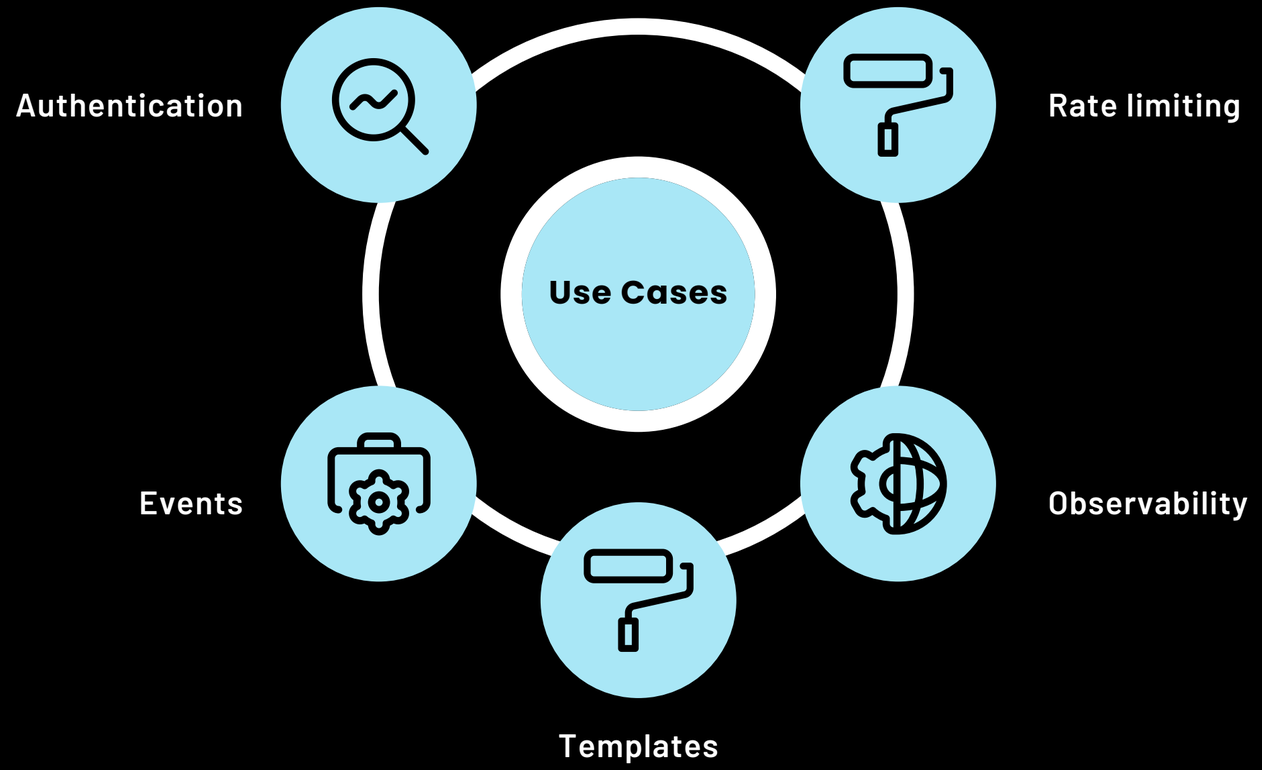
PyPi/NPM packages

Install the package(s)

Use the package(s) where it is needed

Atomic packages for Jobs

Separate Deployment pipelines for utils



Testing Tools

Communication

Key aspect of Microservices

Integration

Ensure the services communicate as expected

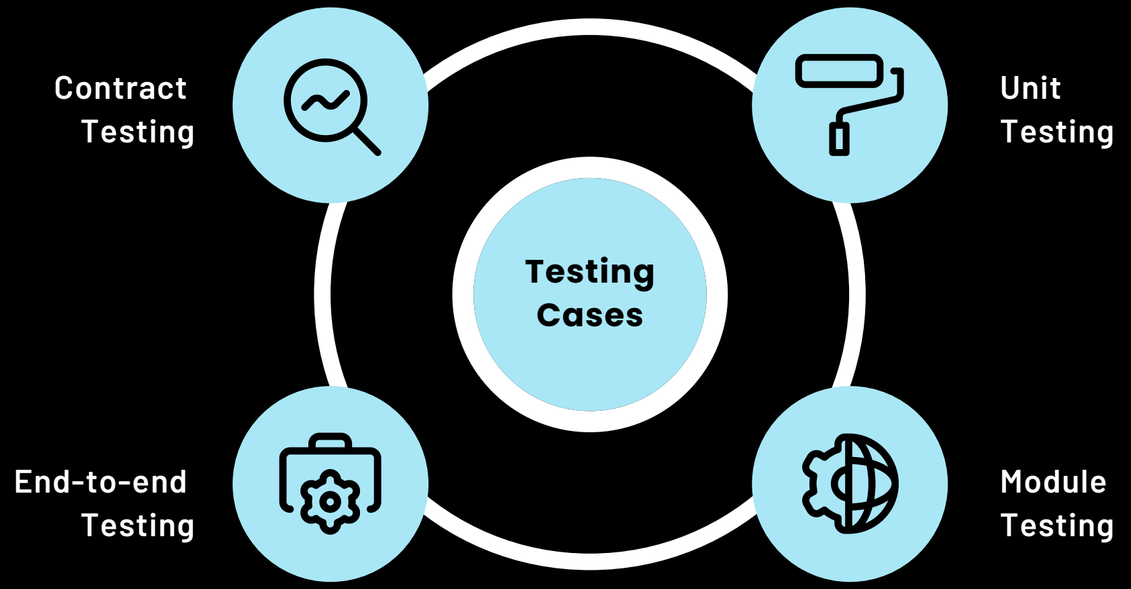
Data Integrity

Ensure the services get the data and behavior they expect and need

Validity

Flagging when an integration would break

Testing tools



Solution

Separate Module and Integration Testing

Distinguish between functional and Integration testing

Divide tests based on approaches

Dividing the tests into the appropriate testing approach

- Integration or E2E Testing
  - Cypress
  - Selenium
- Contract Testing
  - Pact.io
  - Spring Cloud Contract

Service Documentation

Difficulties in Tracking

1

Ownerships
Dependencies
Endpoints/Paths in a service

Ever expanding list of Microservces

3

With the atomicity in play, microservices keep on dividing and expanding quantitatively.

Data Discovery

2

It becomes almost impossible to track which data is redirected where?

Accessibility

4

With micro-ownerships, many documents become inaccessible for other teams based on access.

The Solution?

1

Service Repository

Containing metadata for all repository

2

Data contracts

Generated as part of contract testing

3

API Documentations

Create API documentations using:
Swagger
Stoplight

Microservice Templating

Adding new microservices

For every business use case, create a new microservice

Rapid Prototyping

Create microservice that are easy to create and easy to develop - in a matter of minutes.

Reusable Code Templates

Quicker development based on reusable and consistent codebase structure, requirements, testing and containerization methods.

Reusable Infrastructure Templates

Create infrastructure codebase templates for quicker development.

Helpful Packages

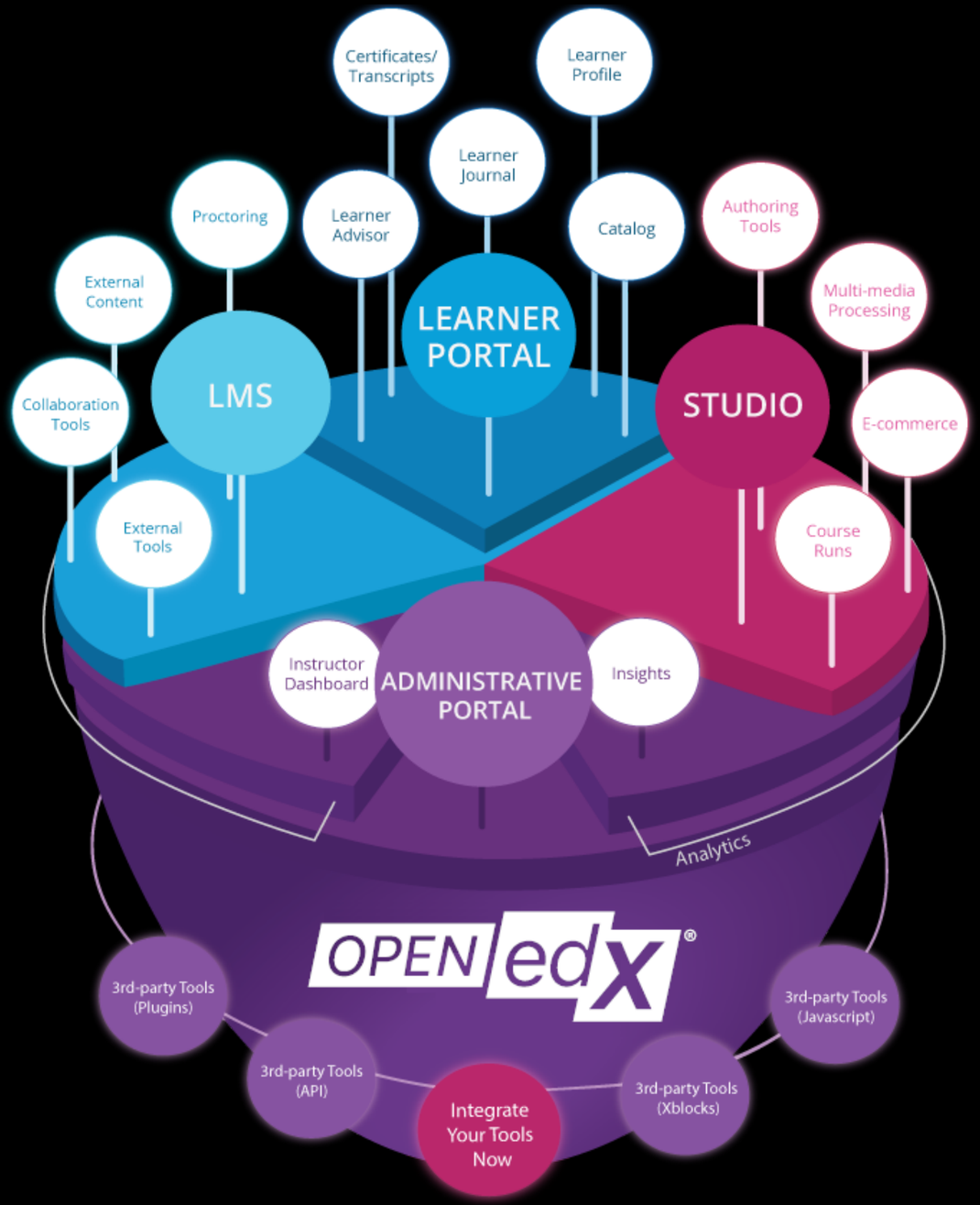
Engineer and use helpful packages such as Terraform and Cookiecutter.

CASE STUDY: OPEN EDX

How OpenEdX engineered its microservices.

OPEN EDX MICROSERVICES

- Independently Deployed Applications (IDA)
  - Backend Django applications
- Micro Frontends (MFE)
  - Frontend React + Redux applications
- ~46 microservices, nearly equal split between IDAs and MFEs



THE JOURNEY

A brief history of open-edX services

edX-platform

The core and the monolith of open edX.

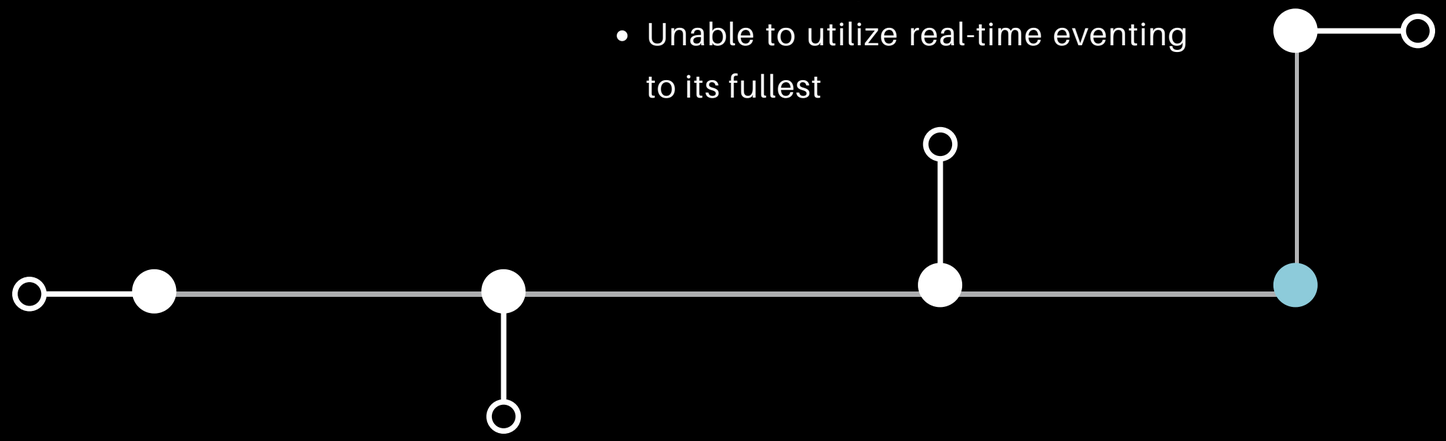
- LMS
- Studio
- Authentications
- Ecommerce
- Video module
- and much more

Development Challenges

Approximate 2.5M lines of code
Long deployment times (2-4 hours)
No clear boundaries of ownership

Scalability Challenges

- Tightly coupled applications
- Extensibility and Performance
- Unwanted Dependencies
- Unable to utilize real-time eventing to its fullest



Present

OpenEdX migrations to microservices

Transition

Open-edX migrations from monolith took approximately 3 years in development before final divisions into the system.

Several criteria were taken into account but the main division category was the division between backend and frontend services.

Dedicated repositories and deployment pipelines

When you can easily connect with your customers, this helps create a solid brand image and customer loyalty.

100K lines of code, 10-30 minutes of deployment.

Your business can reach audiences outside of your geographical area. It means more website visitors and an overall more extensive market reach.

Domain-driven, clearly bounded ownerships

Better customer service means finding effective ways to build trust. For instance, software gives businesses the opportunity to know what customers think of their products and industry in general through reviews.

Enroute Scaling

1

Role of LMS

LMS acts as the central auth and identity provider

2

Utility Packages

Various utility packages (Django, DRF, eventing)

3

Testing Frameworks

Cypress established for E2E testing, Pact under Pilot

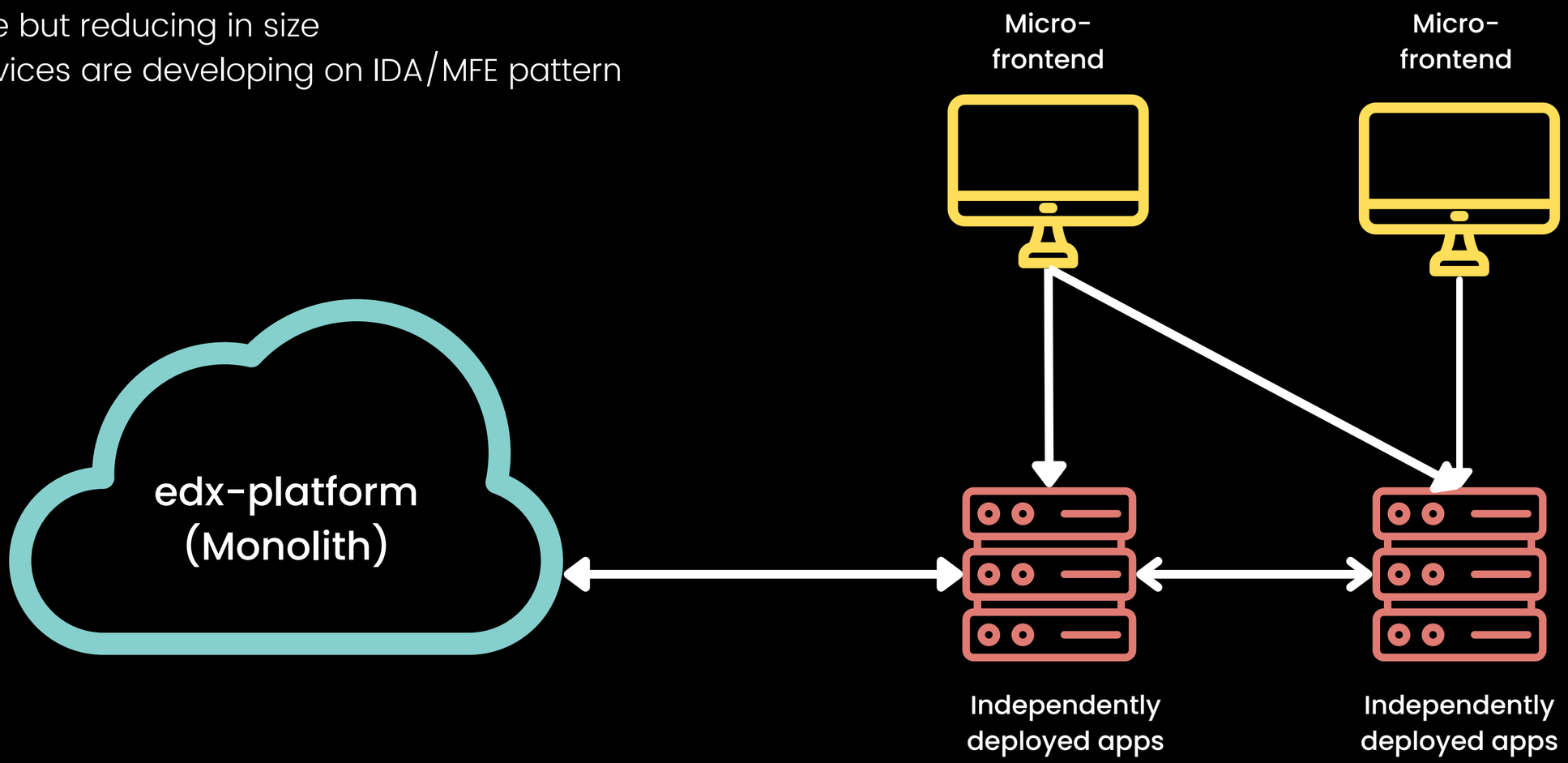
4

Cookiecutter

Cookiecutter established for backend, frontend, and plugin applications

What about platform Monolith?

Still there but reducing in size
New services are developing on IDA/MFE pattern



References

- Microservices vs Monolith
- Netflix Journey from Monolith to Microservices
- Monolith to Microservices: Industry Examples
- Amazon's Microservices journey
- https://open-edx-proposals.readthedocs.io/en/latest/best-practices/oep-0042-bp-authentication.html
- https://github.com/openedx/cypress-e2e-tests
- https://github.com/openedx/edx-cookiecutters
- https://github.com/openedx/edx-django-utils

THANK YOU