

# API-driven Infrastructure as Code Kubernetes APIs as the Contract Bridge between Teams

**Allianz Technology**

Florian Hopfensperger, Allianz Technology  
Yury Tsarev, Upbound

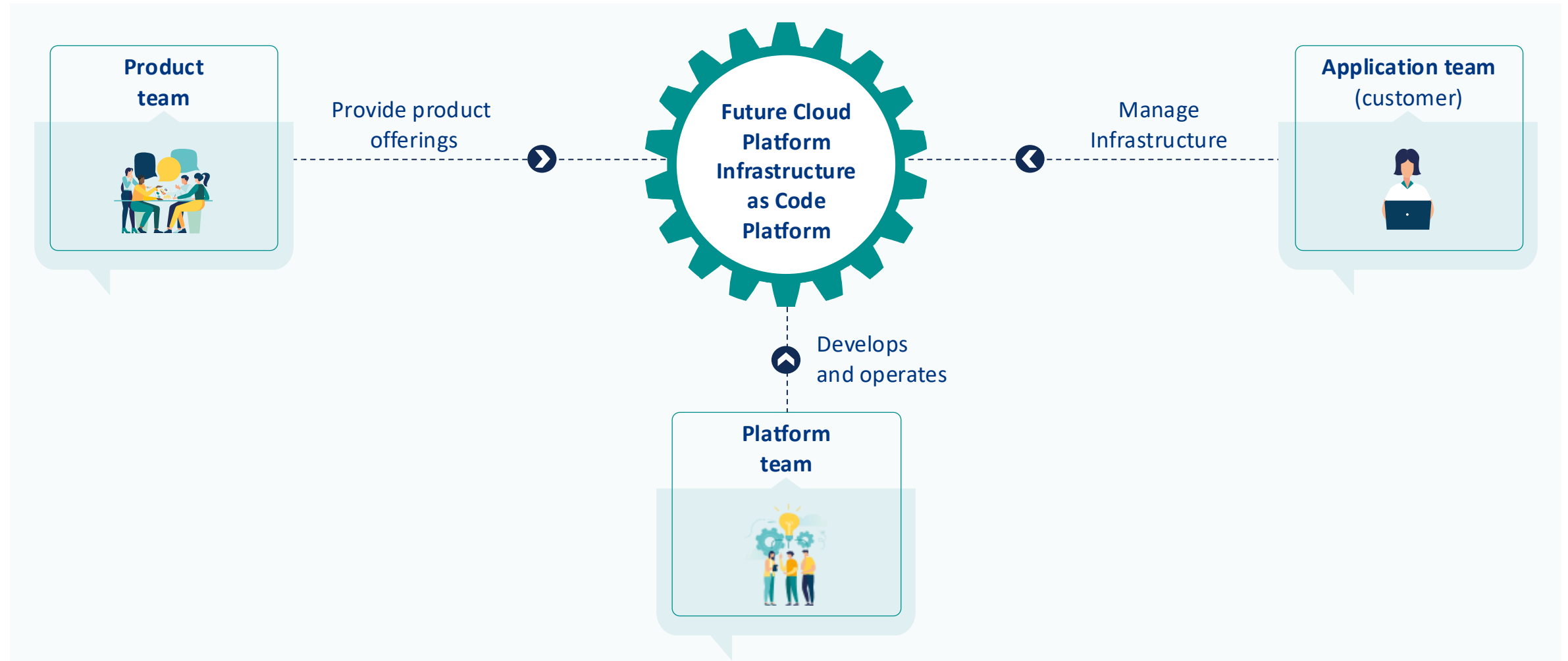
23 March 2026

# 01

## Quick Recap – The Platform We Built



# Describing the context



# Goals, Results and Technologies



01

Self-Service sign-up

1000+ control planes



02

Offer unified interface to manage infrastructure

20K+ infrastructure claim resources



03

Marketplace to offer products

90+ Products managed by ~15 teams



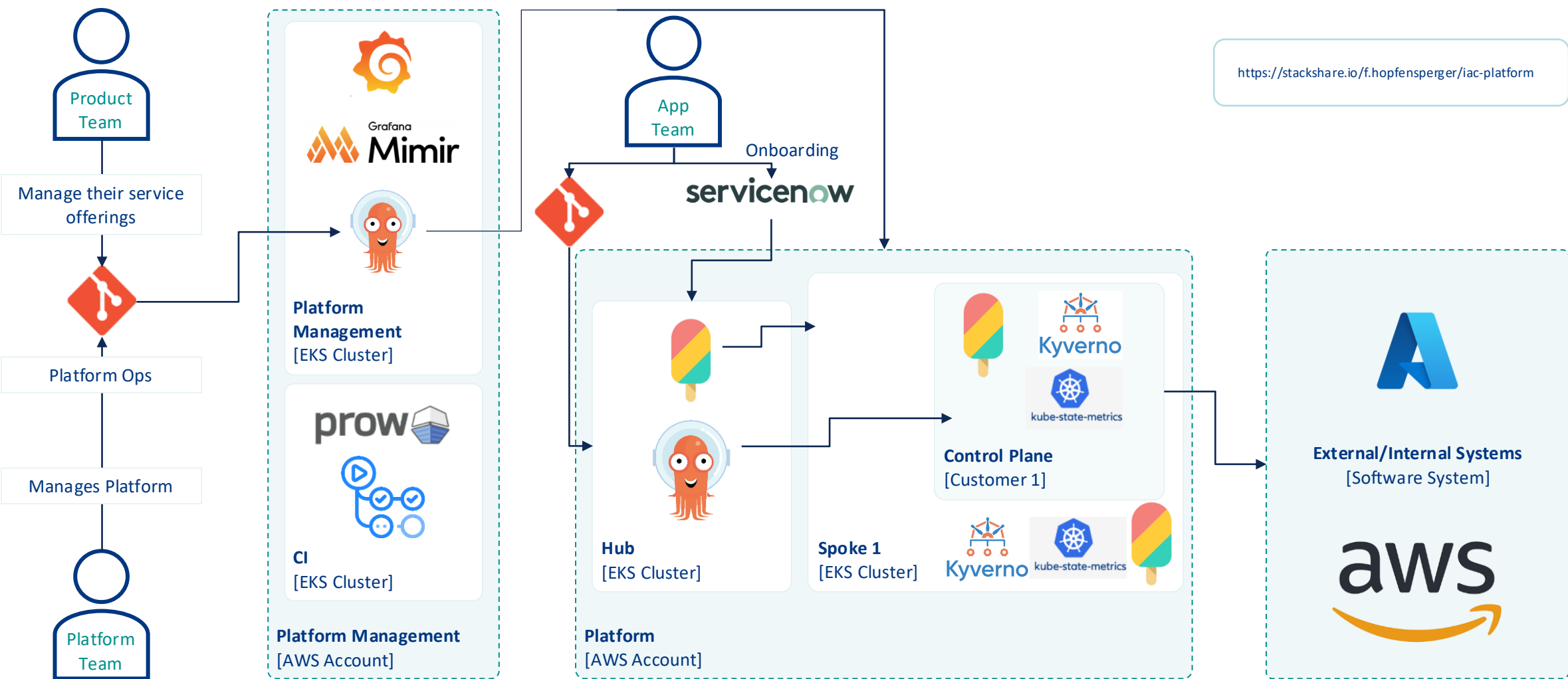
04

Integrate enterprise systems through APIs

Custom/Community Crossplane provider



# High-Level Architecture

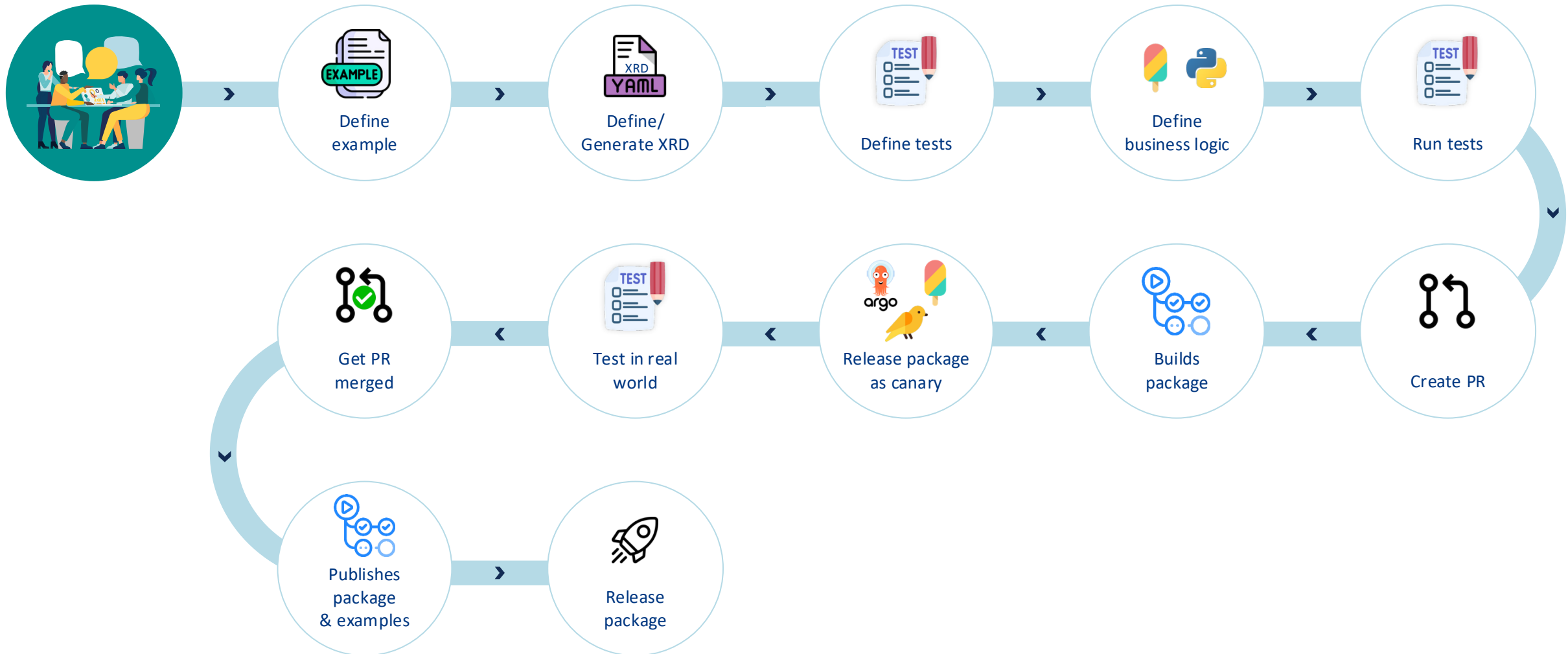


02

## How product teams develop offerings



# Product team development flow



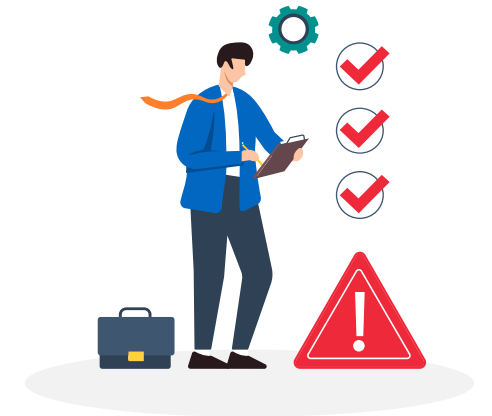
# The Hidden Challenge – Team Collaboration at Scale



Building the platform was the easy part.  
Making teams collaborate effectively was the **hard part**.

## Problems we encountered

- 1 Unclear ownership boundaries
- 2 “Super”-Compositions
- 3 Breaking changes impacting consumers
- 4 Undocumented assumptions



Quote: Who owns this when it breaks?

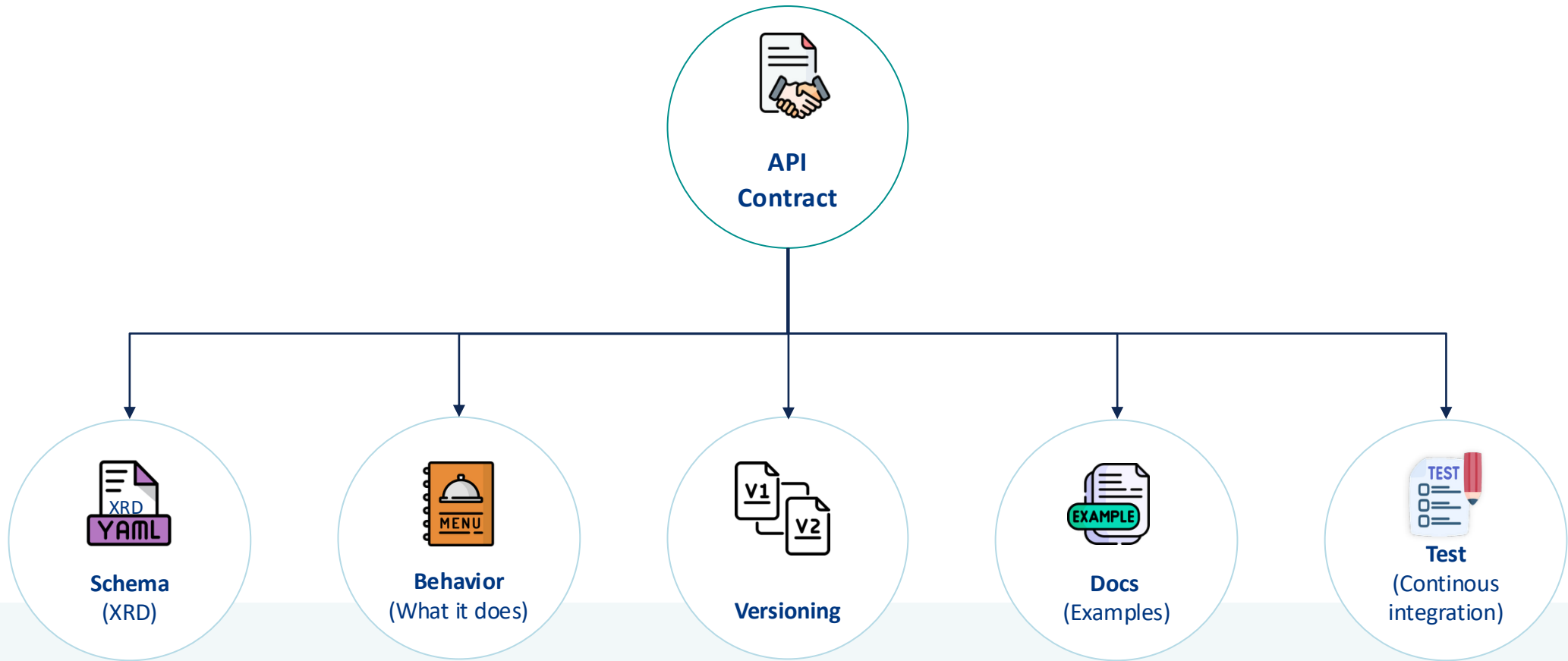


03

# What are API Contracts? The Foundation of Team Collaboration



# API Contracts – In Our Platform Context



# API Contracts – Provider Side



## Schema

- Required vs optional fields
- Valid values (enums)
- Type validation



## Testing

- Schema validation tests
- Behavioral tests
- Backward compatibility
- Runs in CI, blocks merges

```
apiVersion: az.allianz.io/v1alpha1 # Version
kind: DedicatedSubscription
metadata:
  name: my-sub
  namespace: customer
  annotations:
    docs.allianz.io/description: "Isolated Azure subscription"
    docs.allianz.io/dependencies: "None"
spec:
  accountData:
    customerEnvironment: dev # Required: dev, test, prod
    costCenter: "12345" # Required: valid cost center
    region: westeurope # Optional: defaults to westeurope
  status:
    conditions:
      - type: Ready
        status: "True"
  subscriptionId: "sub-xxxxx" # Contract: Always provided
```



## Versioning

- Follow Kubernetes conventions
- Avoid breaking changes



## Documentation

- Generated from annotations
- Always in sync with code
- Examples are "executable"

# API Contracts – Provider consumes Provider Side

```
apiVersion: db.allianz.io/v1alpha1
kind: Database
metadata:
  name: mydatabase
spec:
  parameters:
    engine: mysql # enum "postgres"
    dbName: mydb
    subscriptionId: sub-123456 # optional either this
    subscriptionRef:
      name: my-sub # optional or that
```

## The Challenge

- Database Composition needs subscription ID from another team's XR
- Subscription managed by different provider team
- How to cross provider boundaries?



## EASY: Same Crossplane Provider

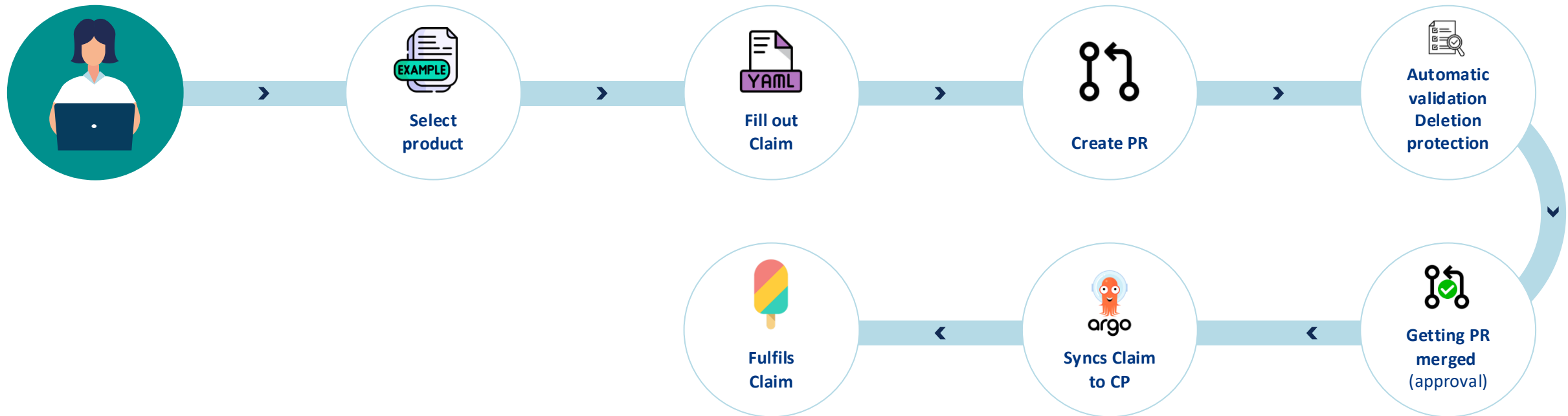
- **Scenario:** Both use provider-azure
- Database Composition  
↓ selector works!  
Subscription Resource
  - ✓ Can use selectors/references
  - ✓ Same provider knows both resources



## HARD: Cross-Provider

- **Scenario:** Different providers
- Database Composition (provider-sql)  
↓ ??? broken  
Subscription Resource (provider-azure)
- **Solution: Extra Resources (required Resources)**
  - ✓ Import ANY resource from ANY MR/XR
  - ✓ Makes cross-team dependencies explicit
  - ✓ Enables composition of compositions
  - ✓ Testable with `crossplane render -e <required-resources>`

# API Contracts – Consumer Side



## Provider Tests

"We're fulfilling our promises"



### What Providers Test

- ✓ **Schema compliance**  
XRD changes don't break existing manifests
- ✓ **Behavioural contracts**  
Compositions create expected resources
- ✓ **Backward compatibility**  
Old manifests (from 6 months ago) still work
- ✓ **Documentation accuracy**  
Examples in docs actually work
- ✓ **Performance/SLA compliance**  
Resources ready within promised time

**Tools:** crossplane render + schema validation

## Consumer Tests

"The API works for our use case"



### What Consumers Test

- ✓ **Can create resources**  
Basic operations work
- ✓ **Can update resources**  
Changes are applied
- ✓ **Can delete resources**  
Cleanup works properly
- ✓ **Error handling**  
Invalid inputs fail gracefully
- ✓ **Performance expectations**  
Resources ready in acceptable time

**Tool:** Claims in test namespaces/control planes & crossplane-diff

### Other related tools:

[github.com/crossplane-contrib/xprin](https://github.com/crossplane-contrib/xprin)  
[github.com/swisscom/crossplane-composition-tester](https://github.com/swisscom/crossplane-composition-tester)  
[github.com/kyvemo/chainsaw](https://github.com/kyvemo/chainsaw)  
Upbound CLI



04

## Lessons Learned



# Three Challenges & Solutions

|                                                        | Problem                                                                | Solution                                                                                   | Result                                                         |
|--------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>01</b><br><b>Service Boundaries &amp; Ownership</b> | <ul style="list-style-type: none"><li>• Who owns what when it fails?</li><li>• Finger-pointing between teams</li></ul>                                    | <ul style="list-style-type: none"><li>• Clear contract boundaries</li><li>• Provider owns: API stability</li><li>• Consumer owns: Valid inputs</li></ul>                      | <ul style="list-style-type: none"><li>• Error messages show ownership</li><li>• "This is a provider issue"</li><li>• "Check your input"</li></ul> |
| <b>02</b><br><b>Breaking Changes</b>                   | <ul style="list-style-type: none"><li>• Provider change broke 50+ teams</li><li>• Discovered only in production</li><li>• Rollback caused chaos</li></ul> | <ul style="list-style-type: none"><li>• Contract testing + versioning</li><li>• Tests catch breaks in CI</li><li>• "Dual" versioning + webhooks</li><li>• "New" XRD</li></ul> | <ul style="list-style-type: none"><li>• Zero production incidents</li><li>• Safe API evolution</li><li>• Teams migrate at own pace</li></ul>      |
| <b>03</b><br><b>Documentation Drift</b>                | <ul style="list-style-type: none"><li>• Docs written once, never updated</li><li>• Examples stopped working</li><li>• New fields undocumented</li></ul>   | <ul style="list-style-type: none"><li>• Documentation as code</li><li>• Generated from XRD annotations</li><li>• CI verifies examples work</li></ul>                          | <ul style="list-style-type: none"><li>• Always accurate</li><li>• Examples are tested</li><li>• High adoption rate</li></ul>                      |

# Thank you!



## Demo Code



## Rate the talk

