Design .NET Core Cloud Native Apps

Thang Chung

Ho Chi Minh - Sep 2018

NASH TECH

The Power to Innovate

- Key Takeaways
  - Cloud Native Fundamentals
  - Design .NET Core Cloud-native Applications
  - Demo
- Q&A

NASH TECH
The Power to Innovate

Cloud Native Fundamentals

Everywhere is cloudy, especially in .NET Conf. and others

Cloud-native Apps Maturity

L3: Cloud Native

- Microservices architecture and principles
- API first design
- Scale dynamically
- Dynamic infrastructure migration without down-time

- Microservices & APIs

L2: Cloud Resilient

- Fault tolerant and resilient design
- Metrics and monitoring build-in
- Run anywhere, and cloud agnostic

- Resilience: monitoring, logging and exception handling
- DevsOps
- Cloud agnostic

L1: Cloud Friendly

- Loosely coupled systems
- Horizontally scalable (services by name)
- Follow 12 factors Apps
- Leverage platform for high availability
- Design for failure (include proactive testing for failure)

- 12 factors apps
- Stateless & Scaling

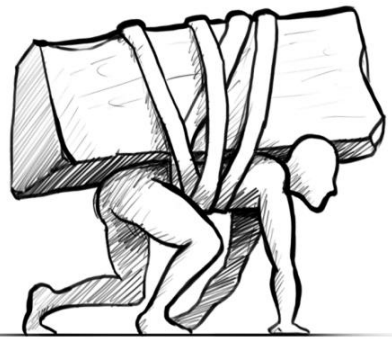
L0: Cloud Ready

- No file system
- Self-contained application
- Run on VM with managed Ports and Addressing
- Consume platform services

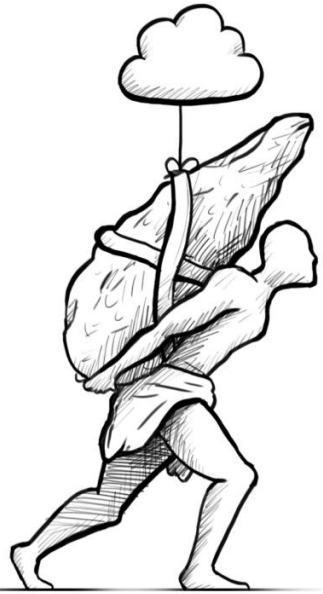
- Containers & Compute Units
- Platforms & Services

Allan Beck & John McTeague (JPMorgan Chase & Co.)

Cloud Native Application Maturity (cont.)



- Monolithic Deployment
- Traditional Infrastructure



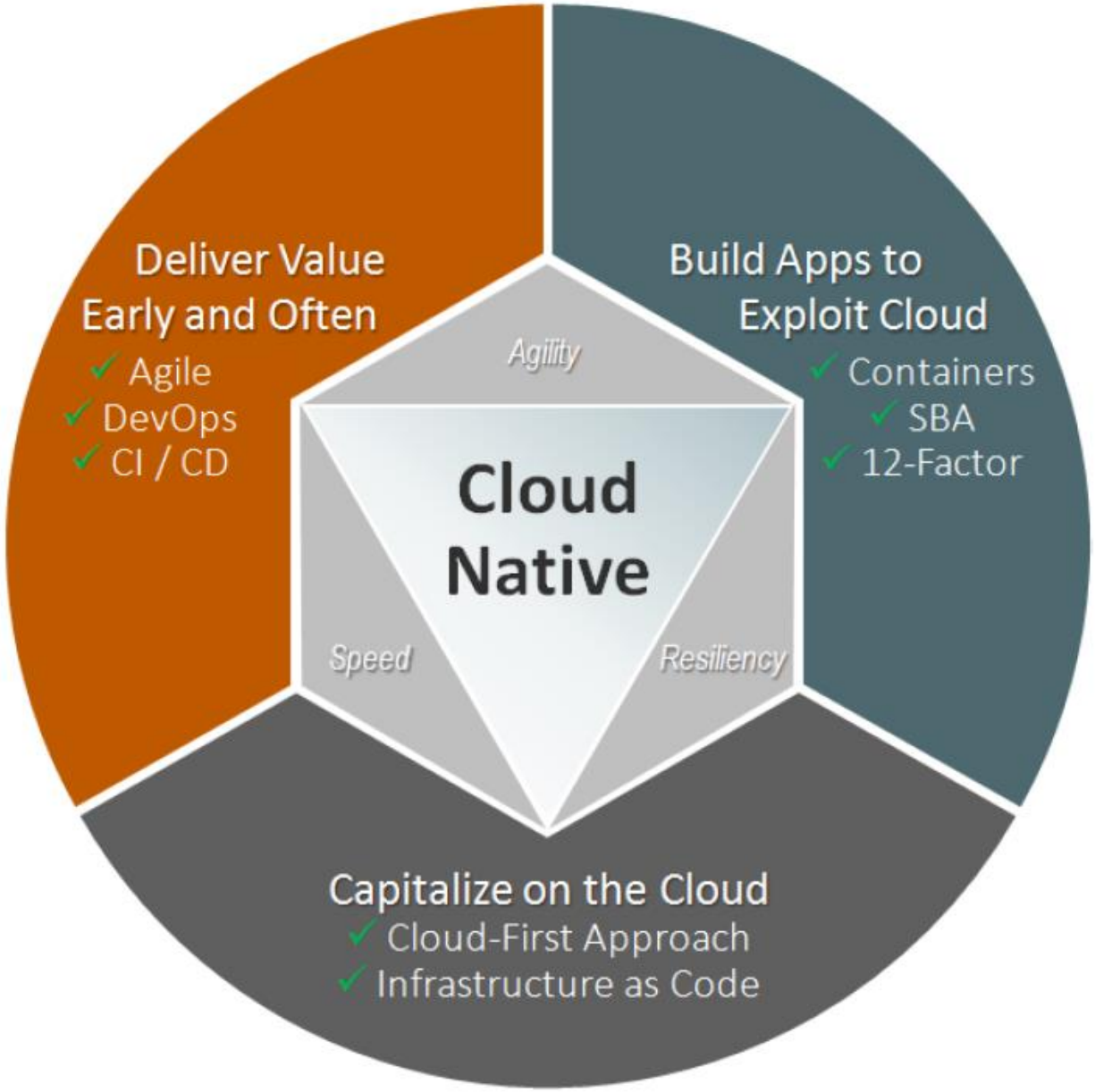
- Containerization
- 12-Factor App Principles

- Microservices
- Cloud-native Apps

Reference at **Steinzeit war gestern! Die vielfältigen Wege der Cloud-nativen Evolution**

Cloud Native Application Development

2017



Reference from Cloud Native Application development - A New Computing Paradigm by Oracle

NASH TECH
The Power to Innovate

Design .NET Core Cloud-native Applications

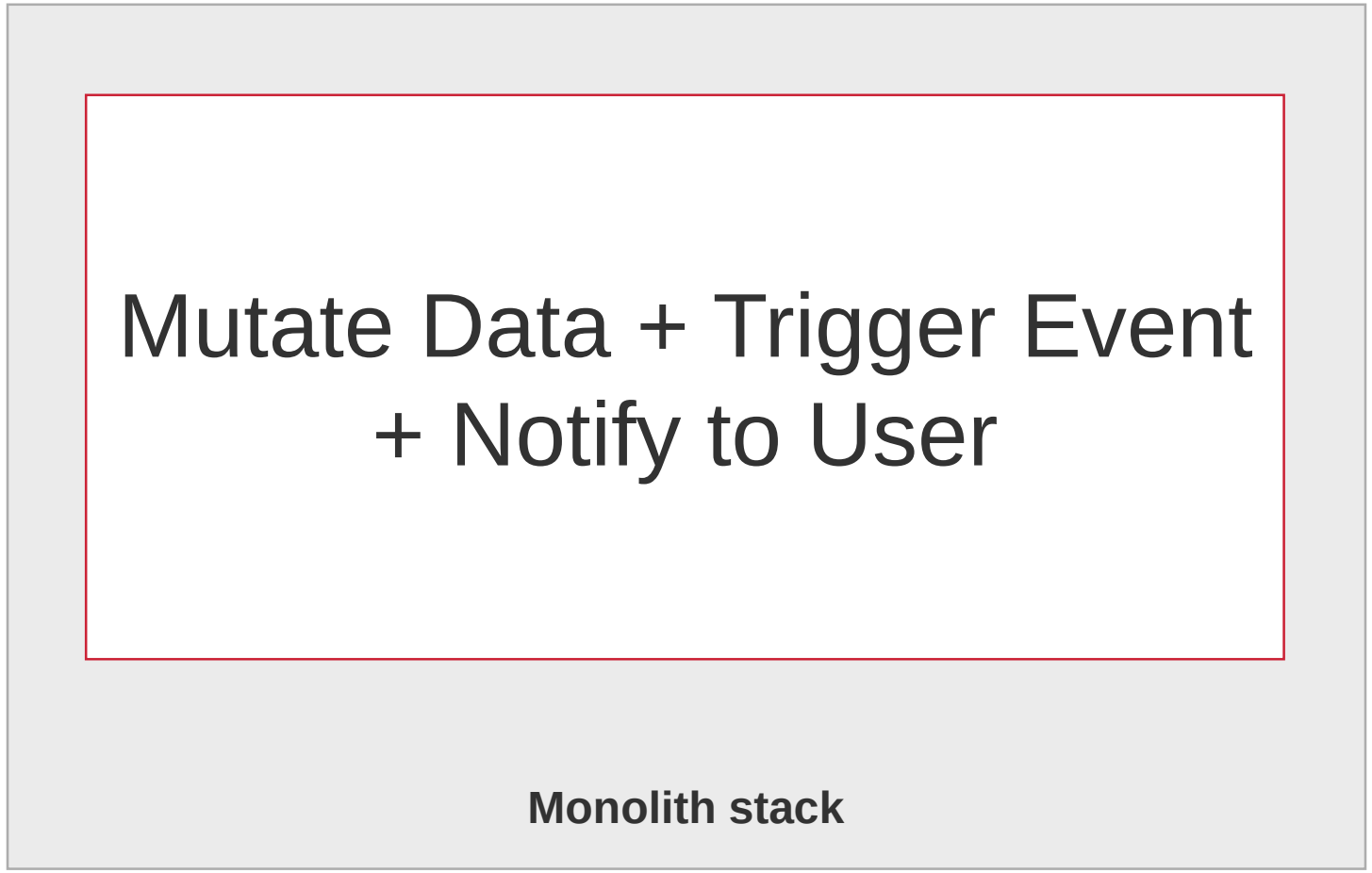
Make .NET Core Apps is able to run on the cloud native environment.

Design .NET Core Cloud-native Applications

- **Factor 1**: One Code-base, One Application
- **Factor 2**: API first
- **Factor 3**: Dependency Management
- **Factor 4**: Design, Build, Release and Run
- **Factor 5**: Configuration, Credentials and Code
- **Factor 6**: Logs
- **Factor 7**: Disposability
- **Factor 8**: Backing Services
- **Factor 9**: Environment Parity
- **Factor 10**: Administrative Processes
- **Factor 11**: Port Binding
- **Factor 12**: Stateless Processes
- **Factor 13**: Concurrency
- **Factor 14**: Telemetry
- **Factor 15**: Authentication & Authorization

Design .NET Core Cloud-native Applications

Design .NET Core Cloud-native Applications

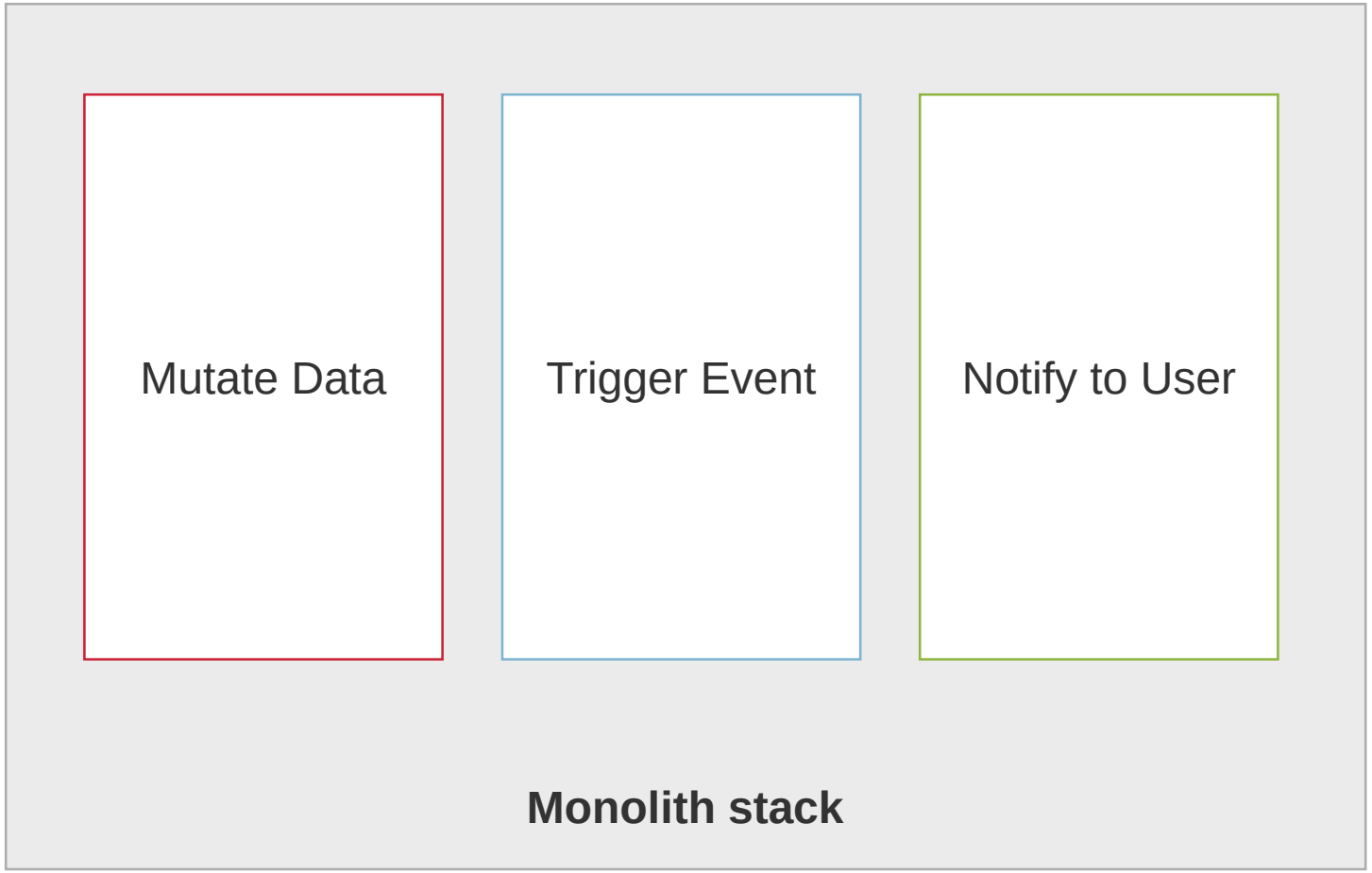


https://martinfowler.com/bliki/MicroservicePremium.html
https://m.signalvnoise.com/the-majestic-monolith-29166d022228

Design .NET Core Cloud-native Applications

Factor 1: One Code-base, One Application
Factor 7: Disposability

- Software Architecture (3 layers, Domain-driven Design, Clean Architecture…)
- Patterns (Repository, Service, Domain Event, Pub/Sub…)

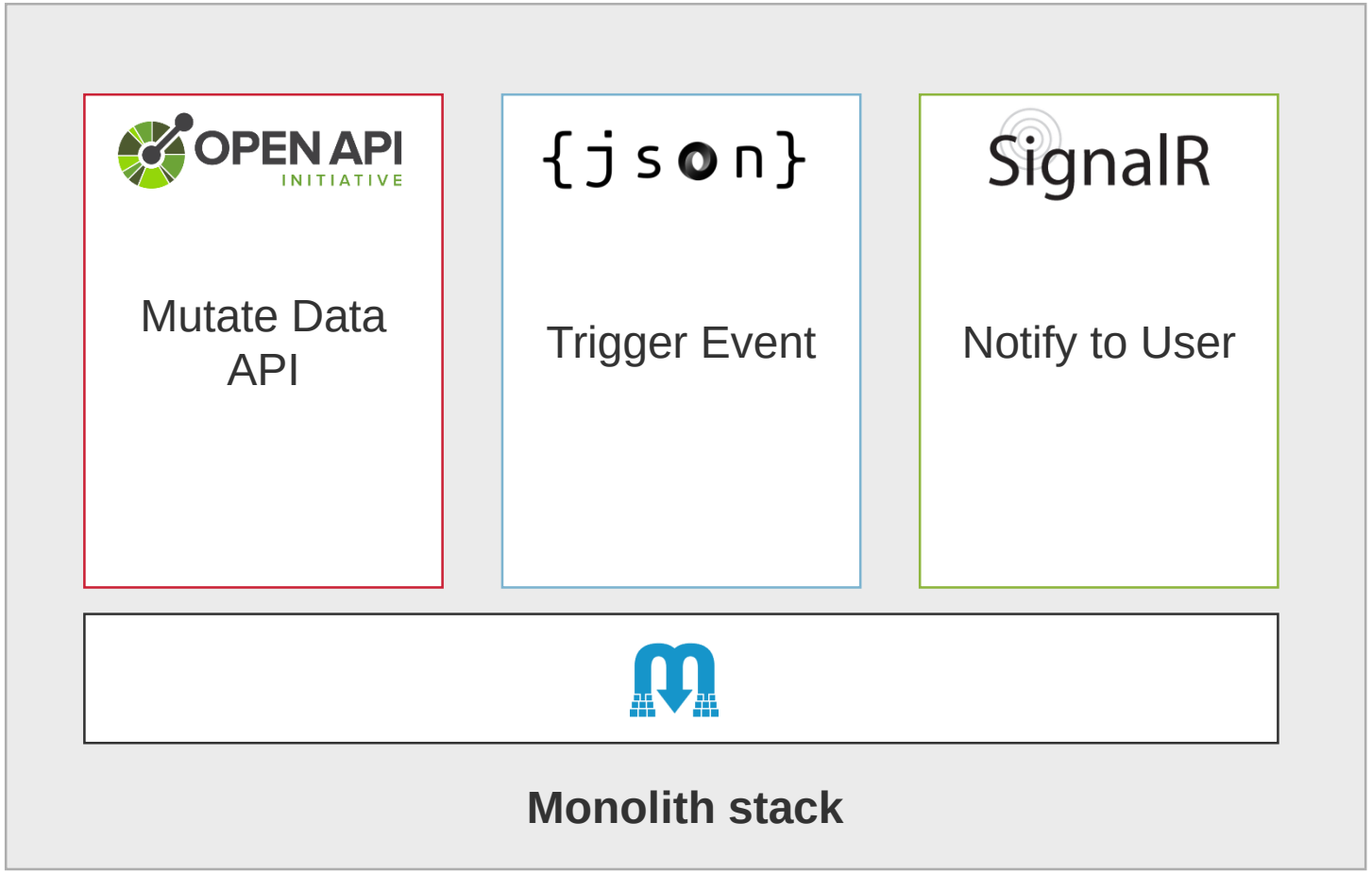


https://hackernoon.com/clean-domain-driven-design-in-10-minutes-6037a59c8b7b

Design .NET Core Cloud-native Applications

Factor 2: API-first

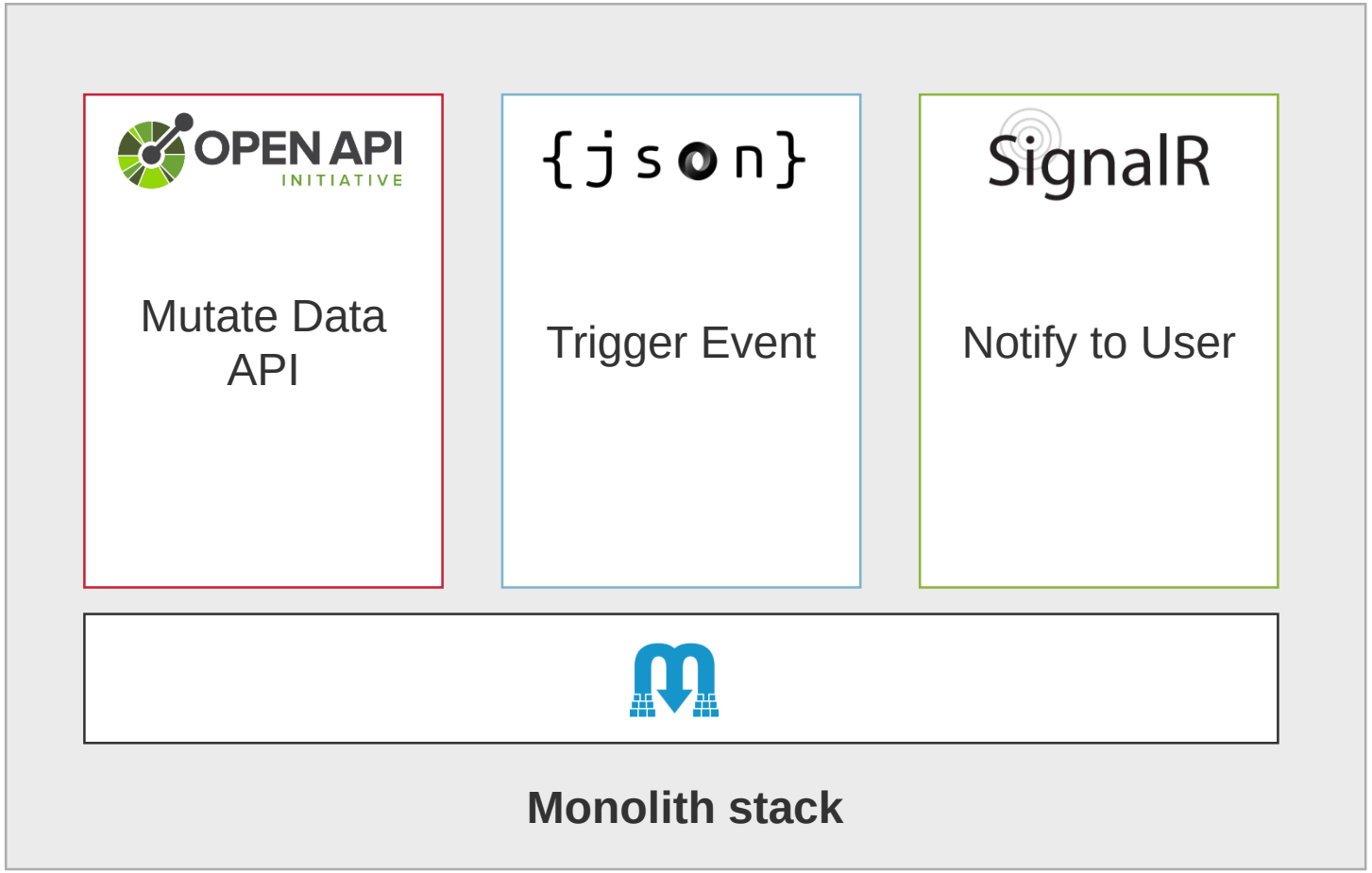
- How components communicate with others?



Design .NET Core Cloud-native Applications

Factor 3: Dependency Management

- Consider to use NuGet, MyGet, NPM, Docker Hub… as much as possible



https://www.nuget.org/packages?q=NetCoreKit

Design .NET Core Cloud-native Applications

Factor 4: Design, Build, Release and Run

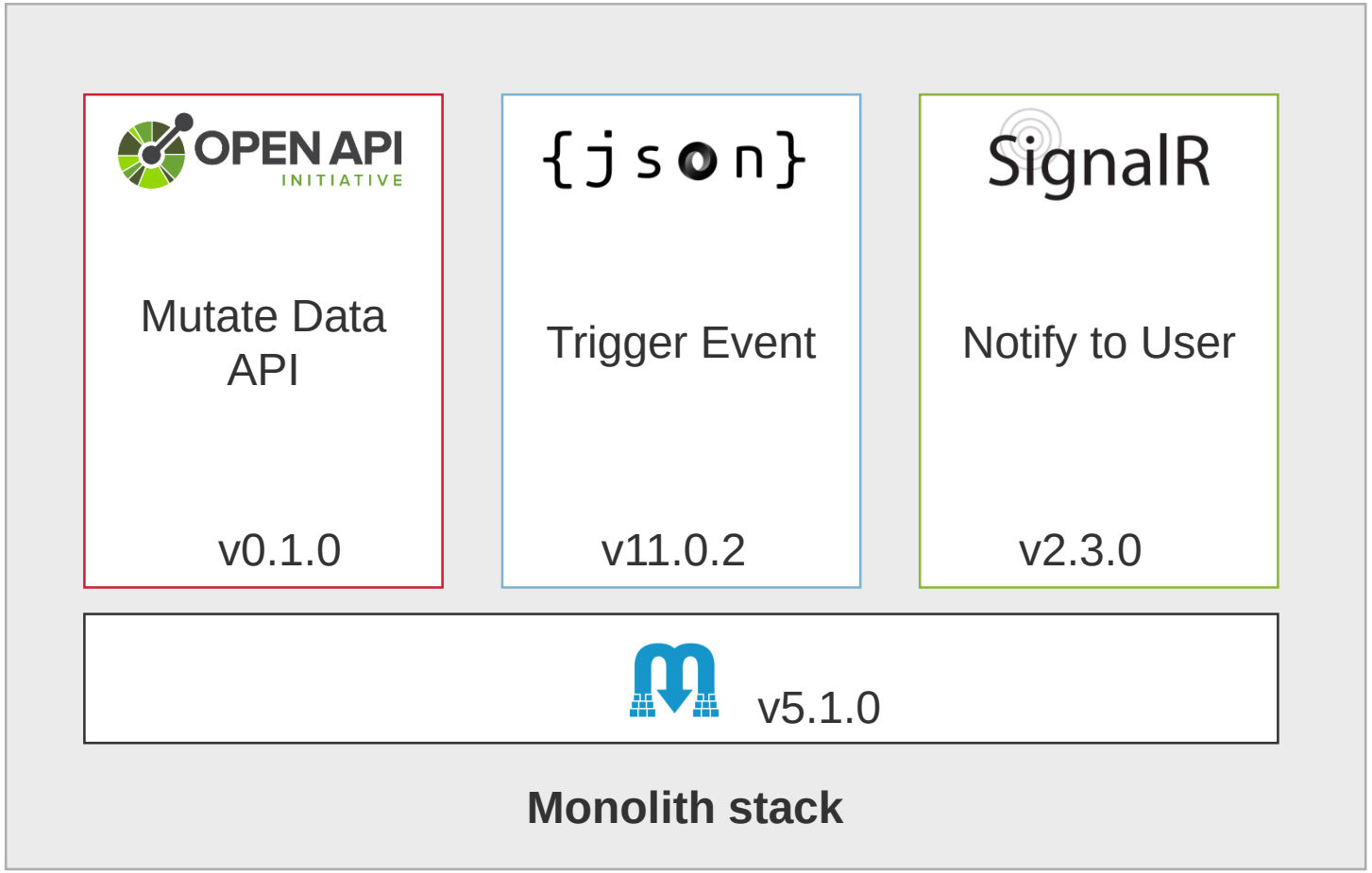
- Versioning for code, build and Dockerfile

```
ARG service_version
ENV SERVICE_VERSION ${service_version:-0.0.1}

ARG api_version
ENV API_VERSION ${api_version:-1.0}
```

API 1.0 1.0

/swagger/v1/swagger.json

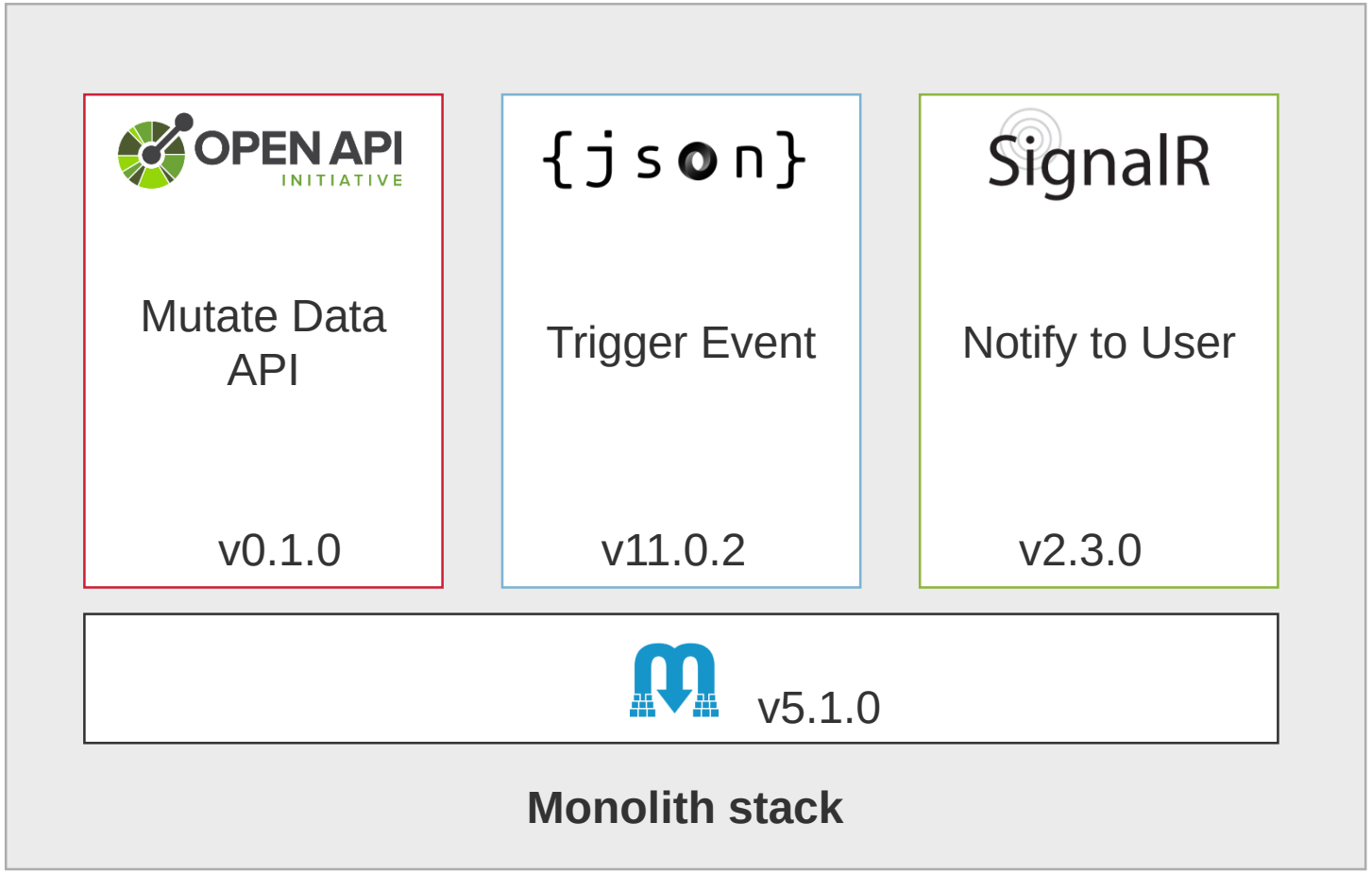


Design .NET Core Cloud-native Applications

Factor 5: Configuration, Credentials and Code
Factor 9: Environment Parity

- Think about **various environments** when cloud-native apps run
- Applications should change behaviors in each **environment** depends on **configuration** + **credentials**, not on **code**

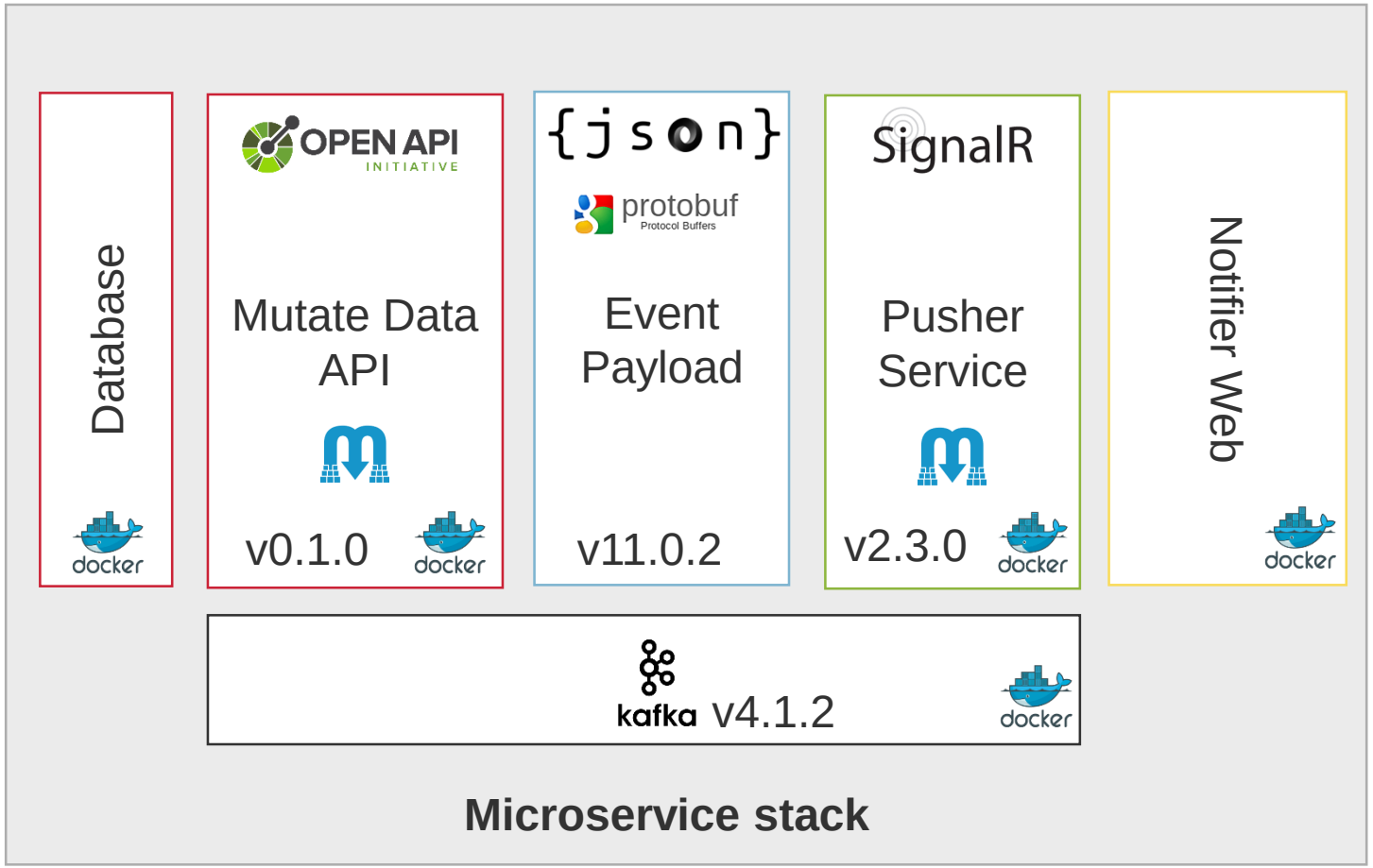
appsettings.json
appsettings.Development.json



Design .NET Core Cloud-native Applications

Factor 8: Backing Services
Factor 10: Administrative Processes
Factor 12: Stateless Processes
Factor 13: Concurrency

- Separated into smaller services, the application state only depends on backing services
- Migration and schedule job… depends on backing services
- Each microservice should be stateless as much as possible
- Then it's able to concurrency scale it out



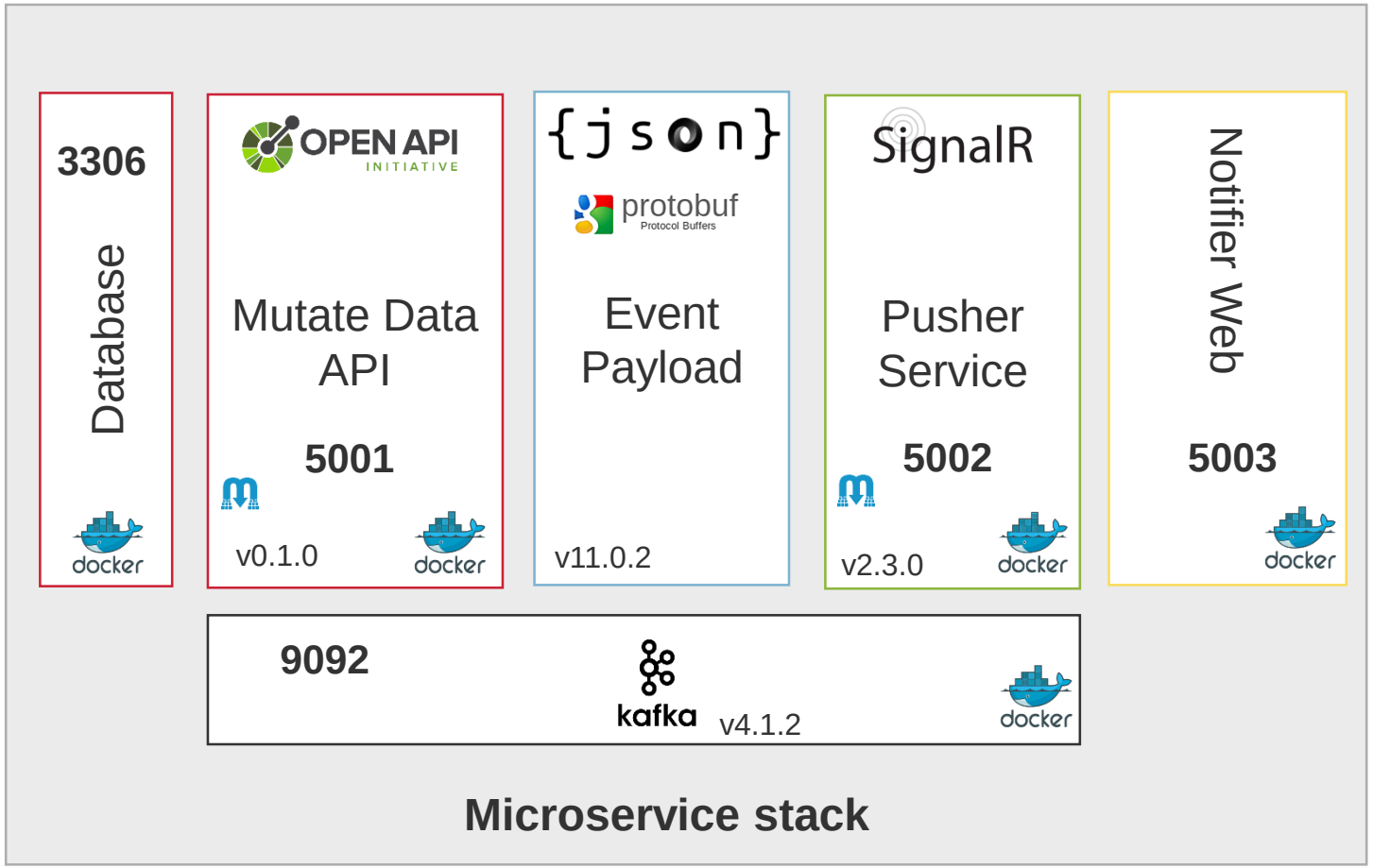
Design .NET Core Cloud-native Applications

Factor 11: Port Binding

```
"samples": {
  "commandName": "Project",
  "launchBrowser": true,
  "environmentVariables": {
    "ASPNETCORE_ENVIRONMENT": "Development"
  },
  "applicationUrl": "http://localhost:5001"
}
```

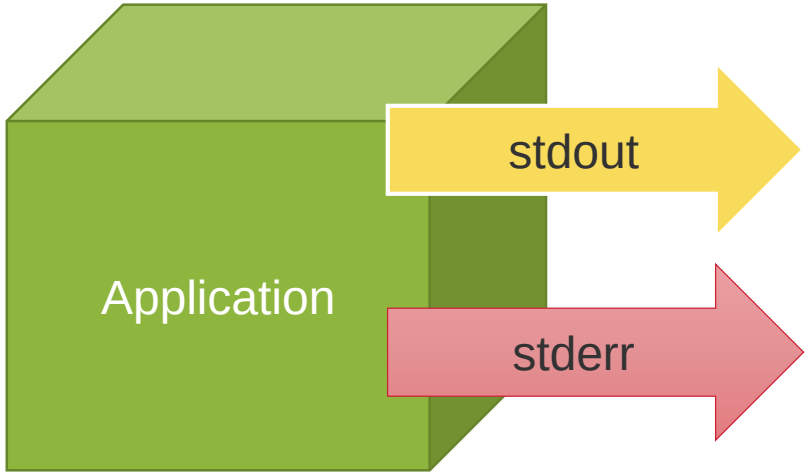
```
ENV ASPNETCORE_URLS http://+:5001
EXPOSE 5001
```

```
ports:
- port: 80
  targetPort: 5001
  nodePort: 32501
  protocol: TCP
  name: http
```



What is not included in this design so far?

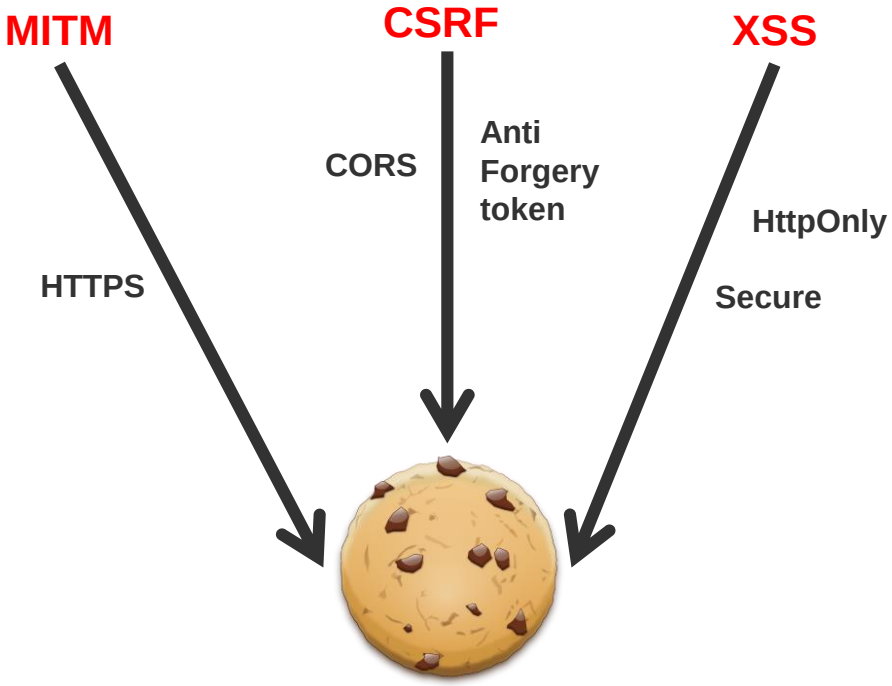
Factor 6: Logs



Factor 14: Telemetry

- Application performance monitoring (APM)
- Domain-specific telemetry
- Health and system logs

Factor 15: Authentication & Authorization

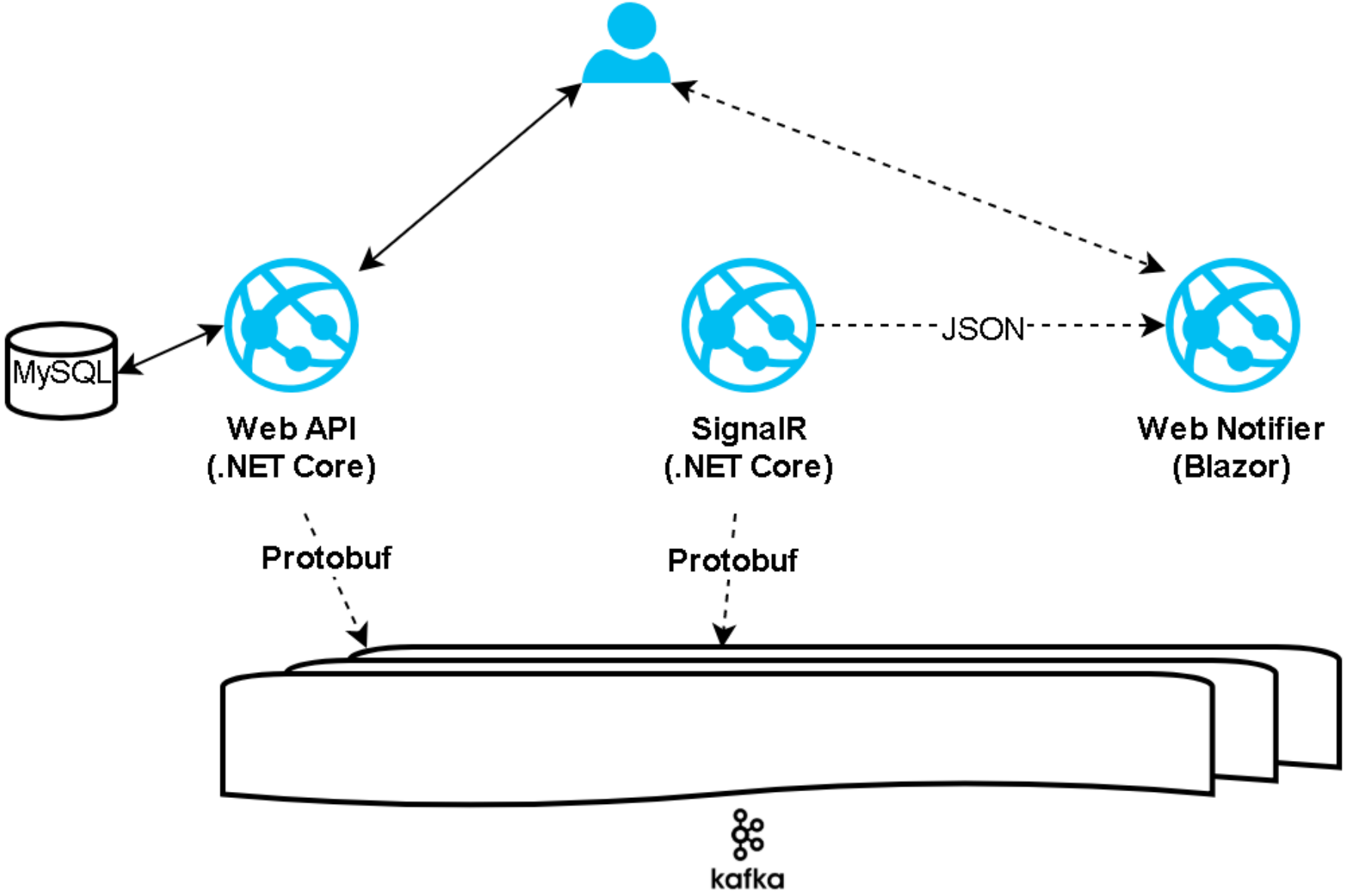


NASH TECH
The Power to Innovate

Demo

Show us how the code run so I can believe in you…

DEMO: .NET Cloud-native Applications



https://github.com/cloudnative-netcore/netcorekit

THANK YOU

www.nashtechglobal.com

NASH TECH
The Power to Innovate

Q&A

References

- Beyond the Twelve-Factor App - Kevin Hoffman
- Cloud Native Infrastructure - Kris Nova and Justin Garrison
- https://martinfowler.com/bliki/MicroservicePremium.html
- https://m.signalvnoise.com/the-majestic-monolith-29166d022228
- https://github.com/cloudnative-netcore/netcorekit
- https://github.com/vietnam-devs/coolstore-microservices