สยามชำนาญกิจ
SIAM CHAMNAN KIT

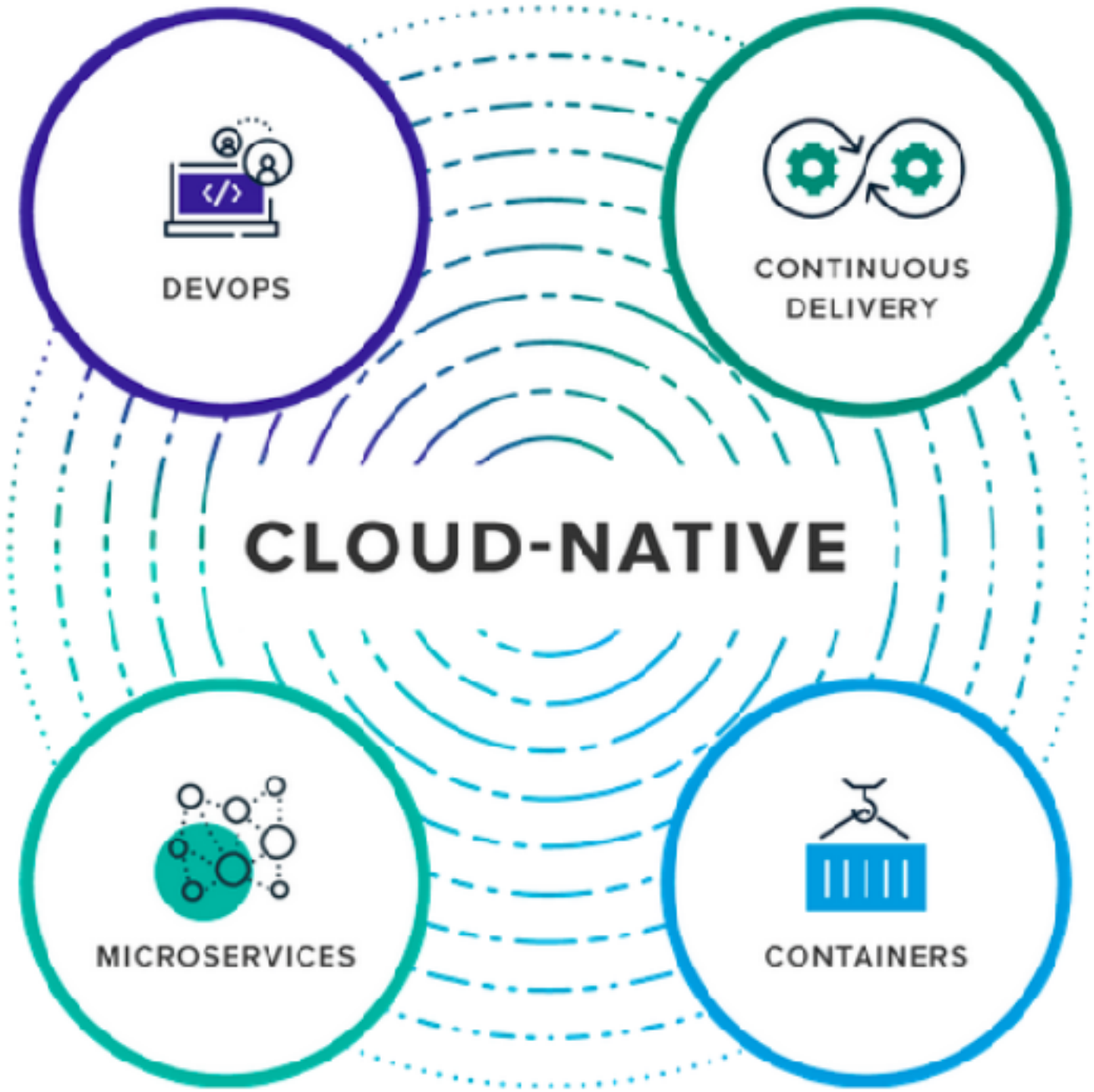
12-Factor !!

https://github.com/up1/course-12-factors-app

12 Factor App

https://12factor.net/

Foundation to build Cloud Native Application



Cloud Native Application

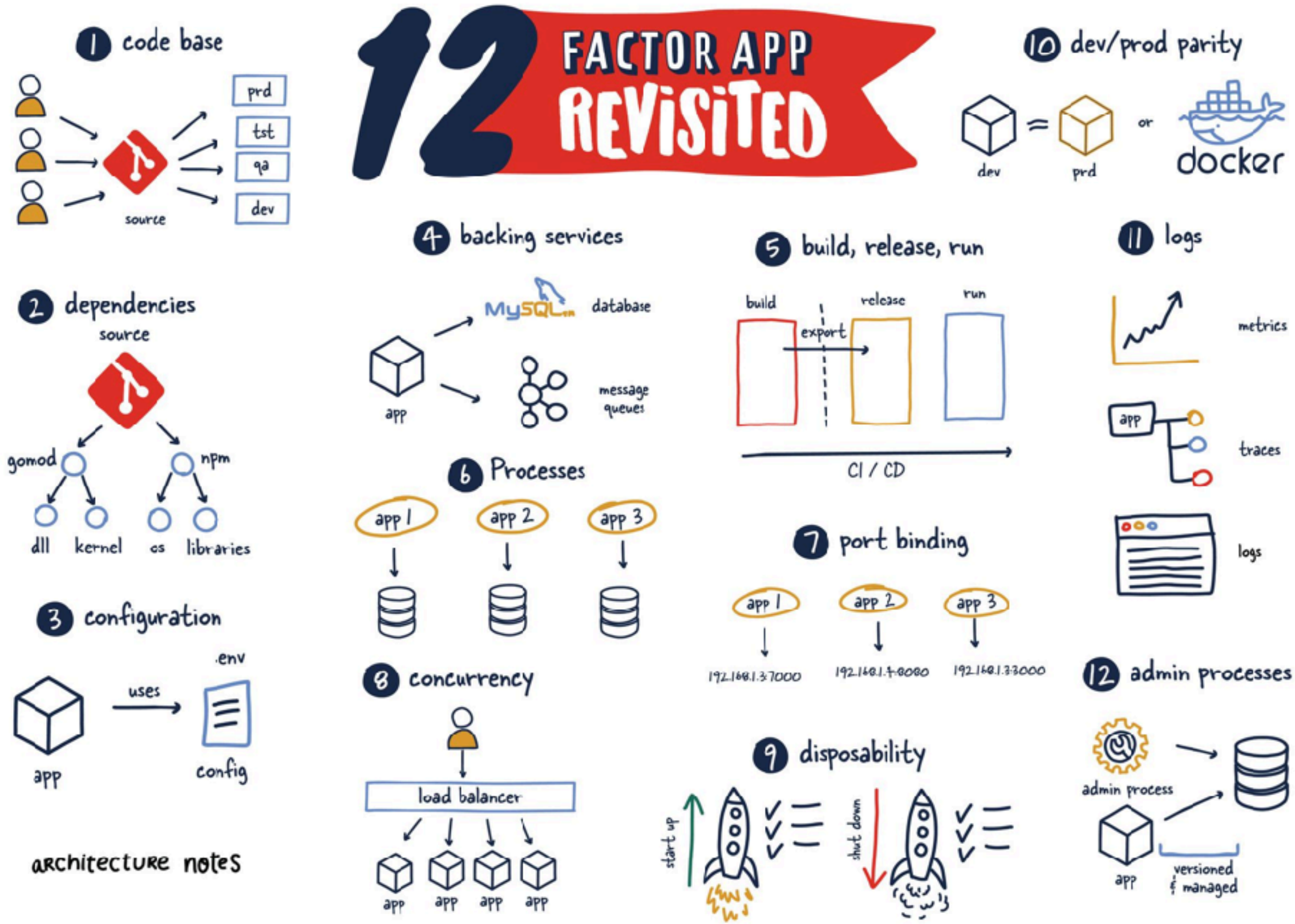
Increase efficiency
Reduce cost
Ensure availability

https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/definition

12 Factor App

Methodology for building software-as-a-service app
Declarative formats for setup automation
Portability
Deployment on modern platform
Minimize divergence (dev vs prod)
Easy to scale up/out

https://12factor.net/



https://architecturenotes.co/12-factor-app-revisited/

12 + 3 Factor

Codebase
One codebase tracked in version control, many deploys

Dependencies
Explicitly declare and isolate dependencies

Config
Store config in the environment

Backing services
Treat backing services as attached resources

API First
Define service contract

Build, Release and Run
Strictly separate build and run stages

Processes
Execute the app as one or more stateless processes

Port Binding
Expose services via port binding

Concurrency
Scale out via process model

Telemetry
Monitor process performance

Disposability
Maximize robustness with fast startup and graceful shutdown

Dev/Prod Parity
Keep development, staging and production as similar as possible

Logs
Treat Logs as event streams

Admin Processes
Run admin/management tasks as one-off processes

Authentication
Secure applications across hardware, network, and software layers

https://www.handsonarchitect.com/2022/08/cloud-native-fifteen-factor-apps.html

12 Factor

Build	Scalable	Maintainable
Portability	Security	Minimize divergence

Monitoring/Observability

12 Factor

Build	Scalable	Maintainable

1. Codebase
2. Dependencies
3. Config
4. Backing services
5. Build release run

API first

12 Factor

Build

Scalable

Maintainable

6. Processes
7. Port binding
8. Concurrency
9. Disposability

12 Factor

Build

Scalable

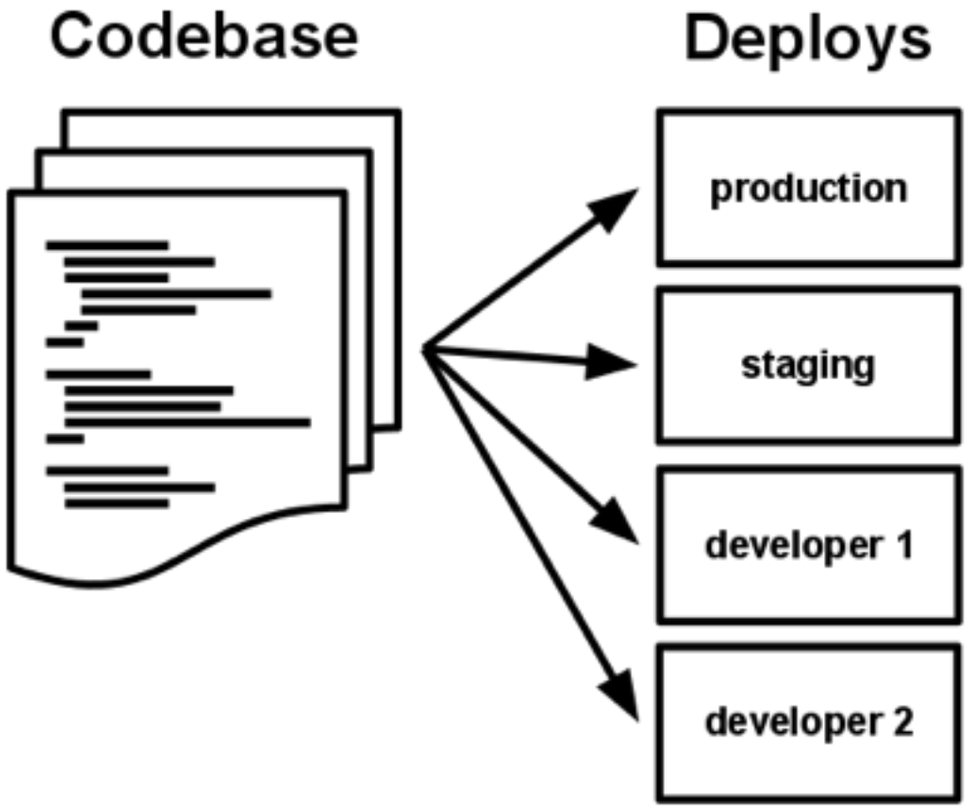
Maintainable

10. Dev/prod parity
11. Logs
12. Admin process

1. Codebase

1. Codebase

One codebase tracked in revision control, many deploys



Version Control System

https://git-scm.com/

Git Branching Strategies

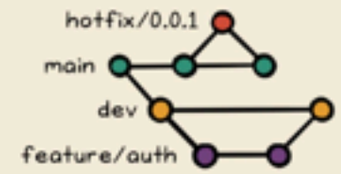
by levelupcoding.co

Feature branching

Each new feature has its own branch. Once the changes are complete and merged in, the branch can be deleted.

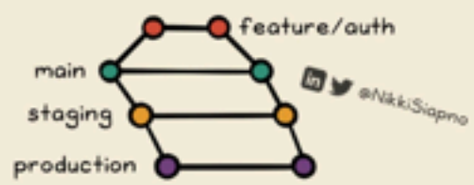
Gitflow

Has branches for features, releases, hotfixes, and a dedicated branch for production and development.



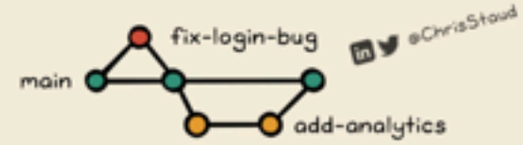
GitLab Flow

Branches are created for features, releases, and environments. The main branch is always production-ready.



GitHub Flow

Branches are created for new features, bug fixes, and experiments. The main branch is the only deployable branch, it is kept production-ready.



@NikkiSiapno

Trunk-based

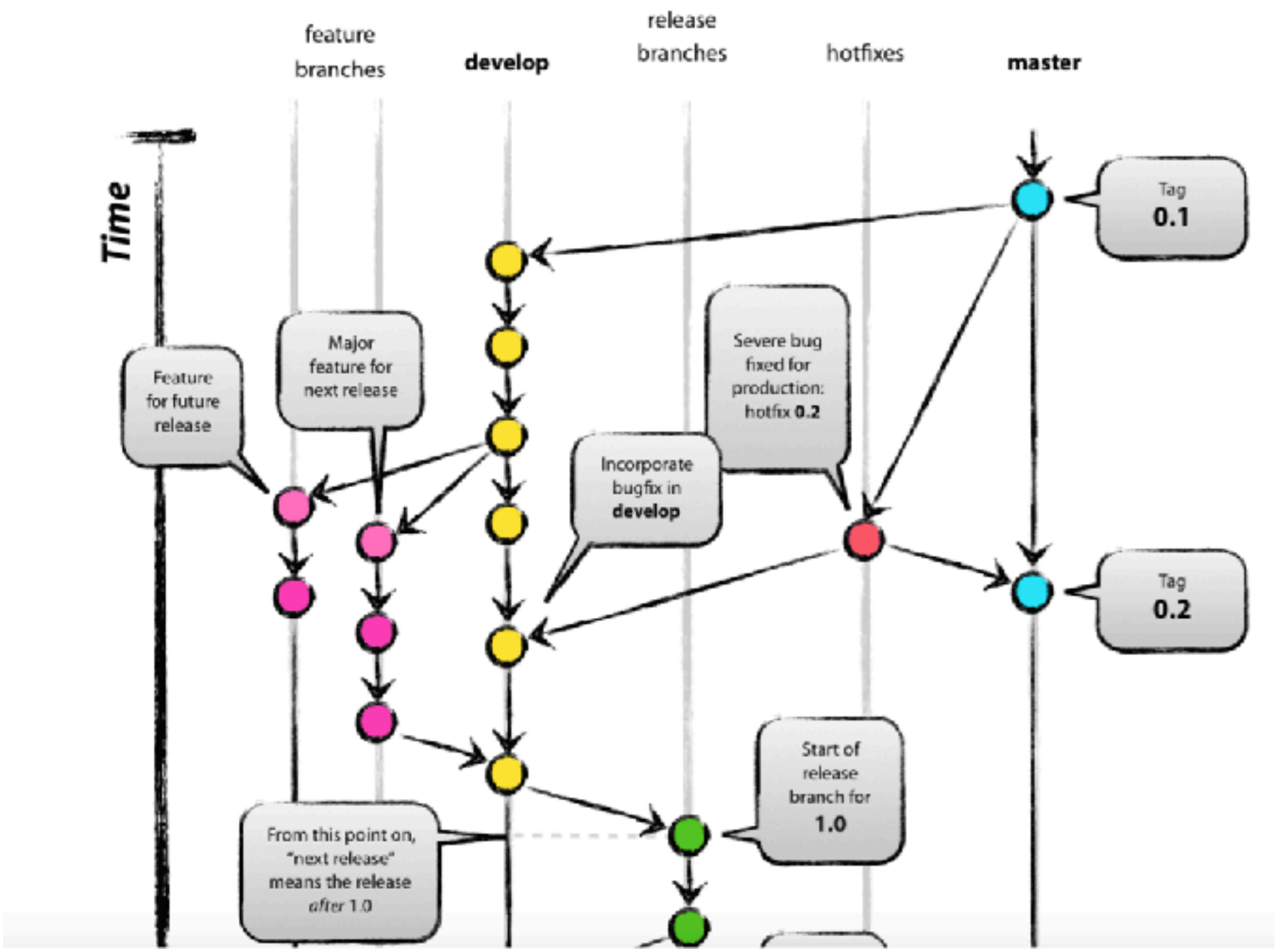
All branches other than the main branch are short-lived and merged in within a defined timeframe (usually a day). Large features are built incrementally and hidden behind feature flags.

@ChrisStaud

Brought to you by LEVEL UP CODING

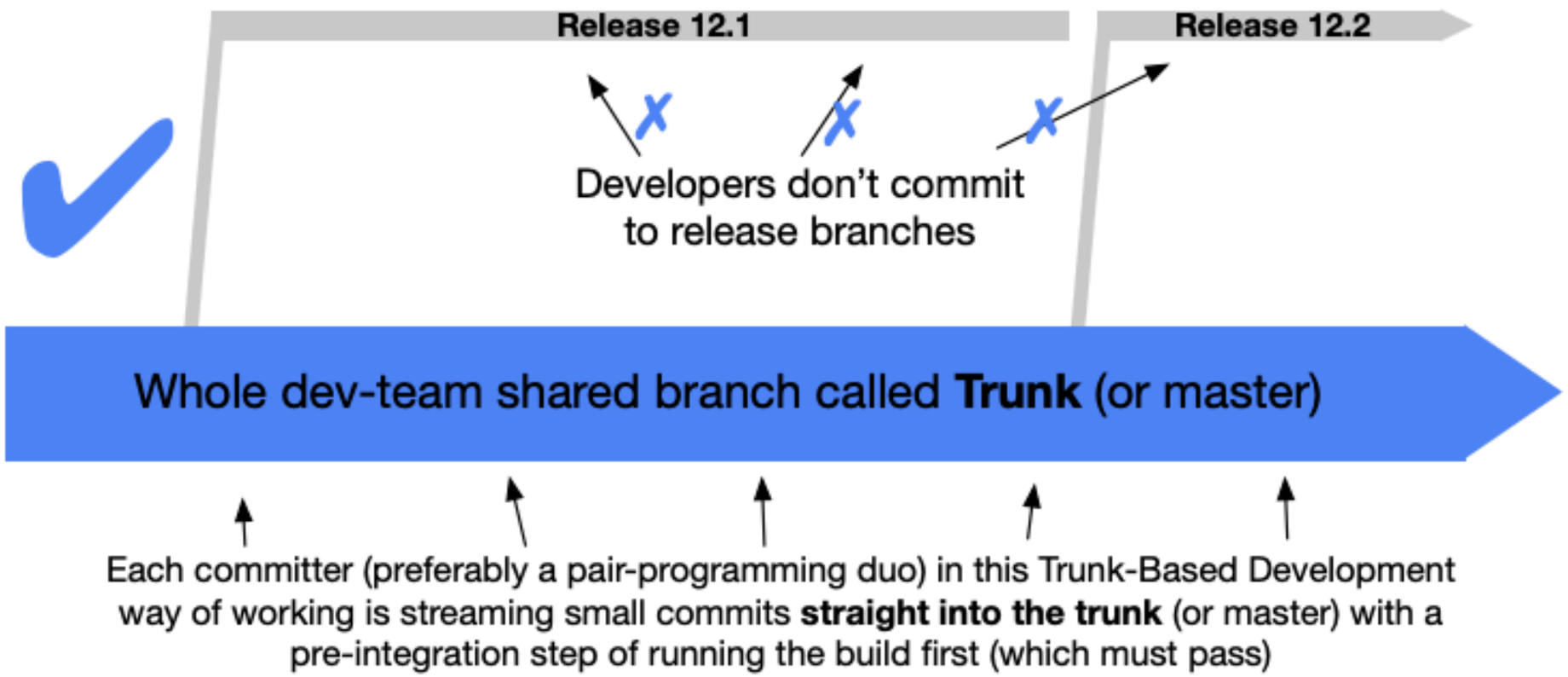
https://twitter.com/ChrisStaud/status/1752341883198853588

Git branching model



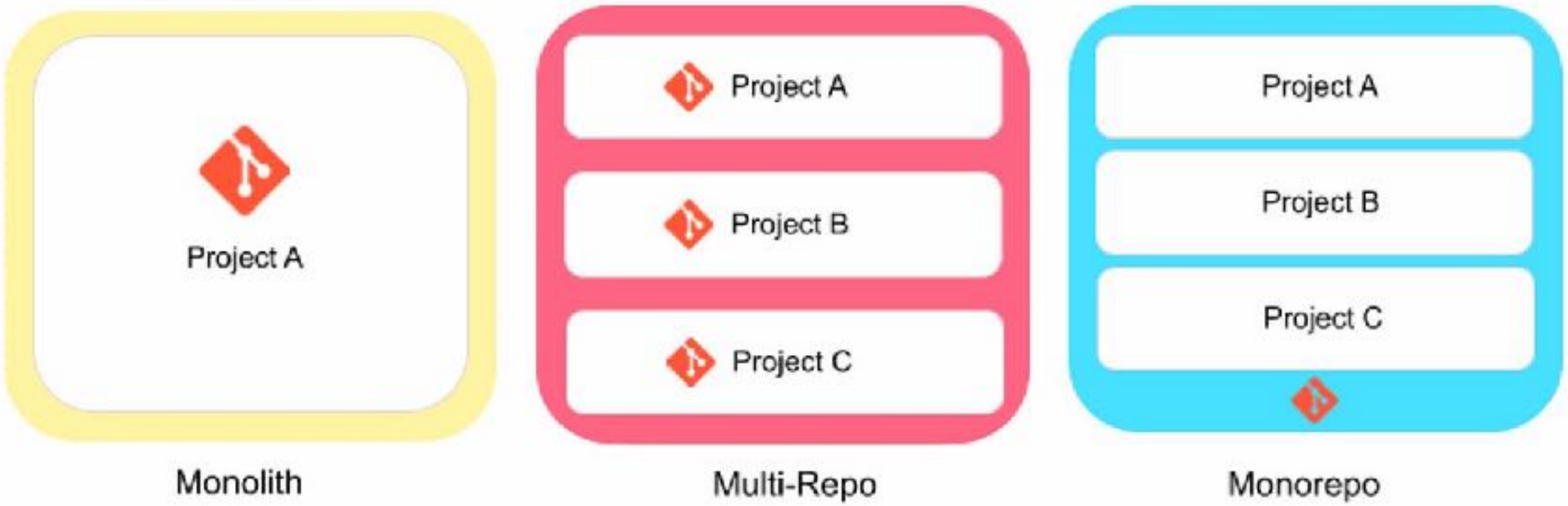
https://nvie.com/posts/a-successful-git-branching-model/

Trunk Based Development

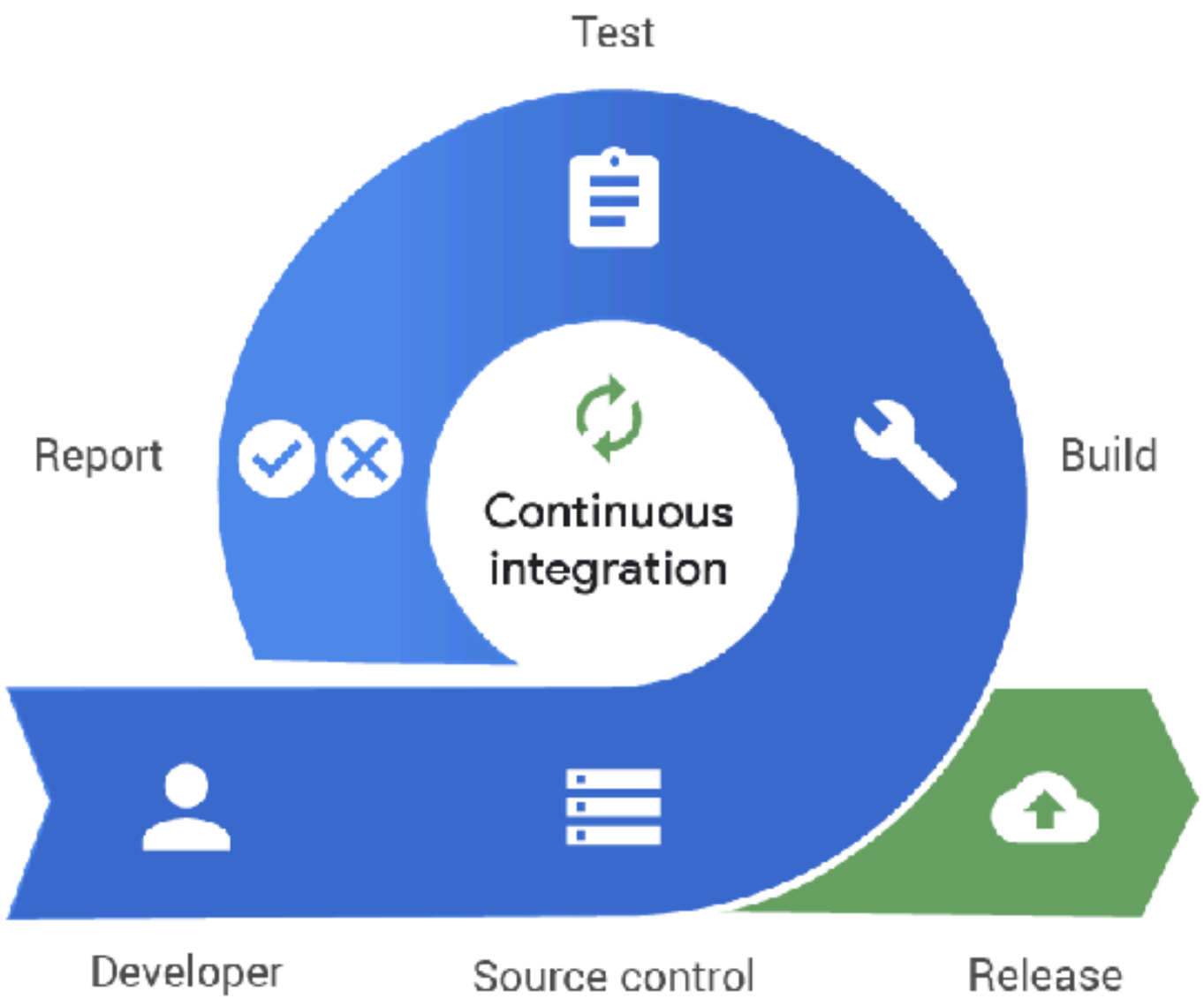


https://trunkbaseddevelopment.com/

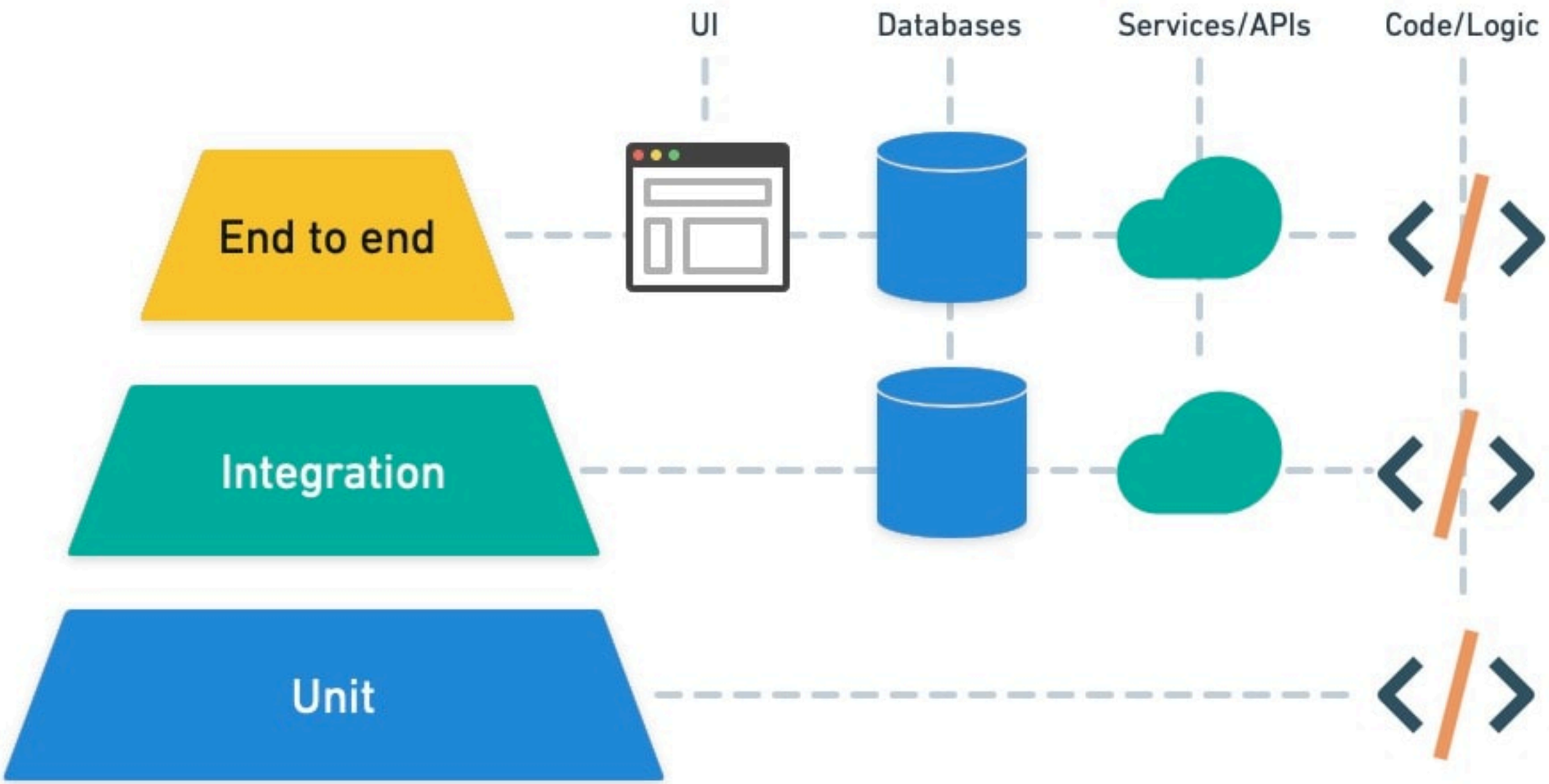
Repository ?



Continuous Integration



Automated Testing



2. Dependencies

2. Dependencies

Explicitly declare and isolate dependencies
Dependency management

3. Config

3. Config

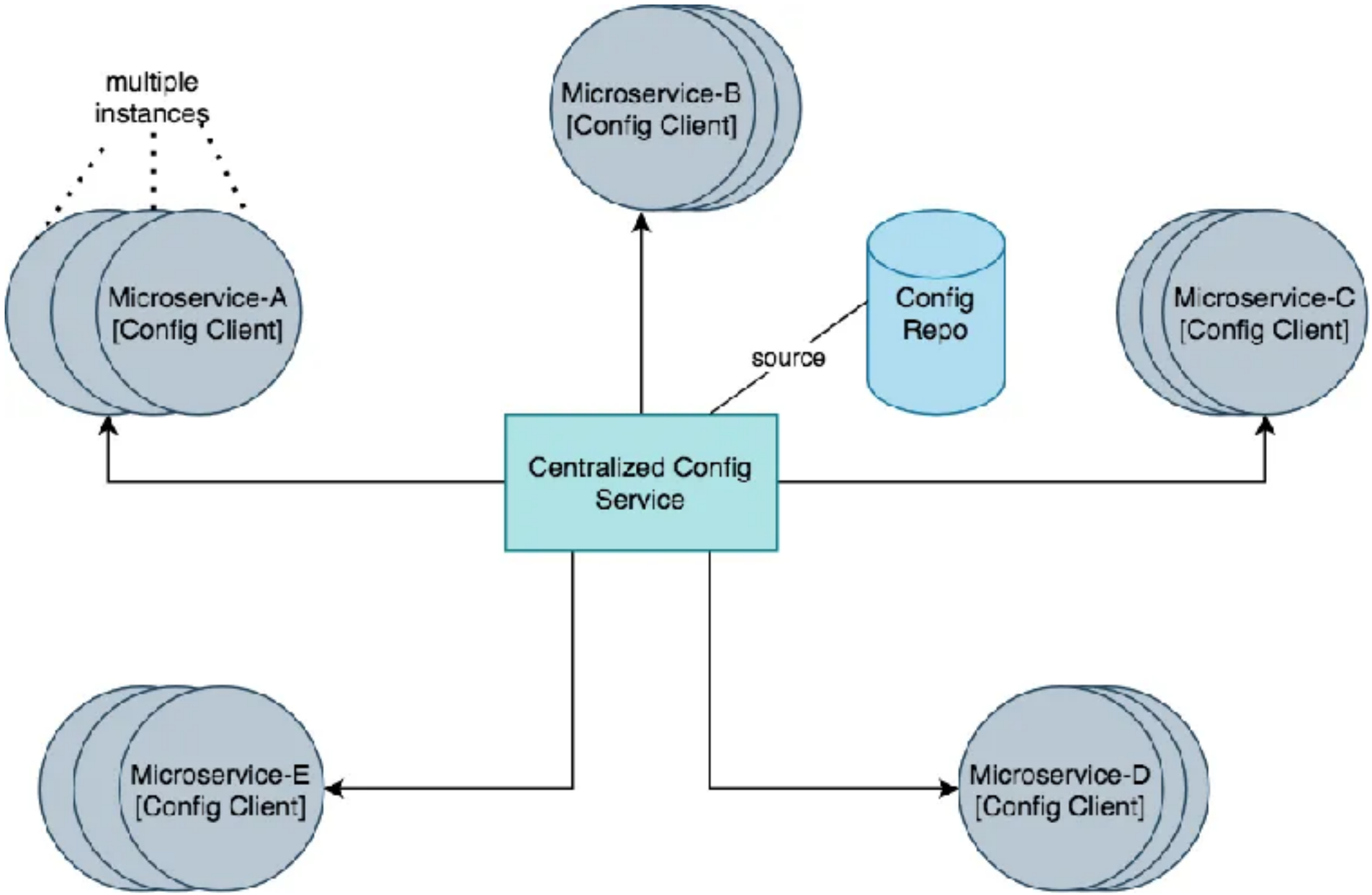
Store config in environment

Code	Config	Credential

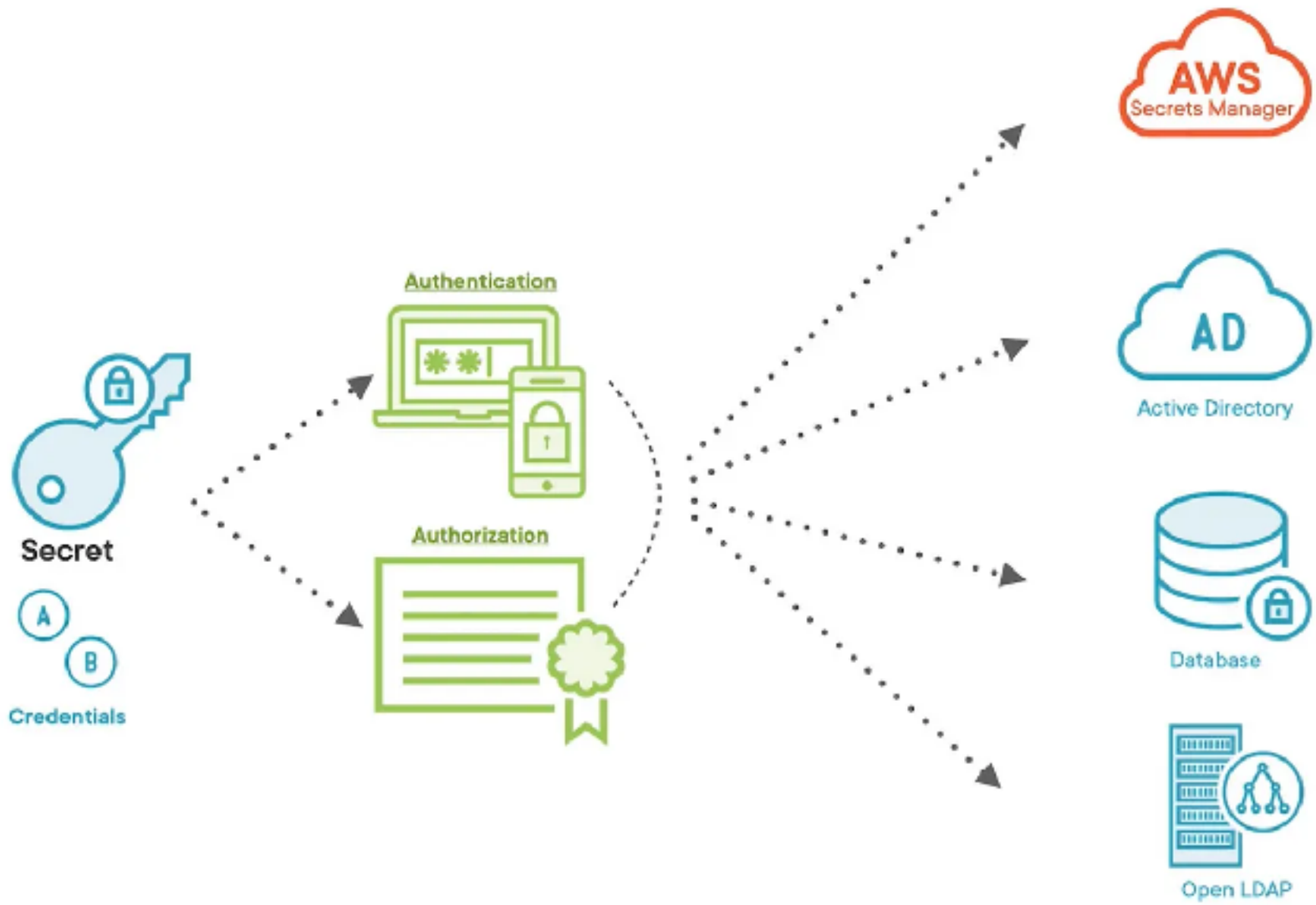
Configuration management

Environment variables (.env file)
Centralized management

Centralized ?



Credential/Secret management



4. Backing services

4. Backing services

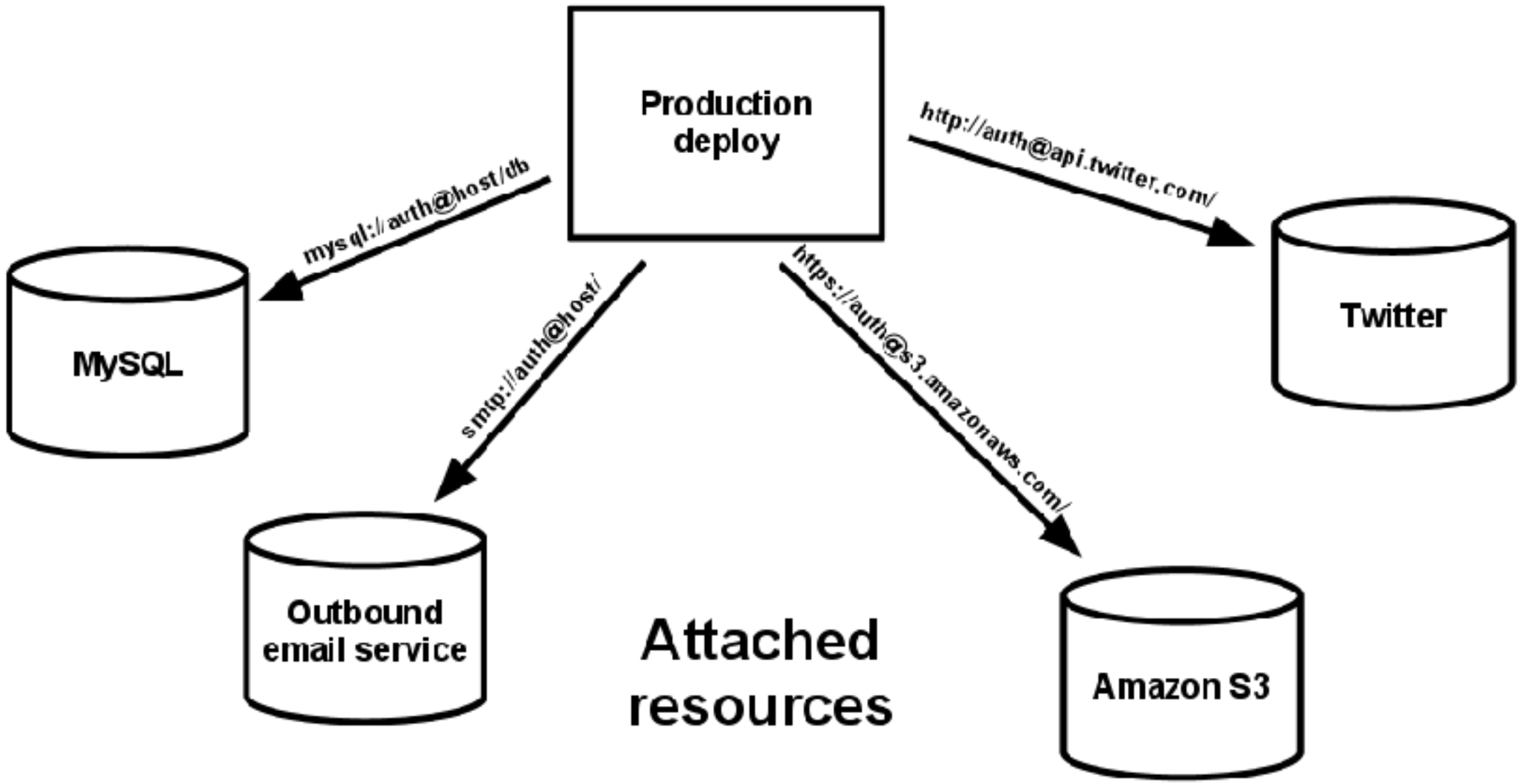
Treat backing services as attached resources
Expose via addressable URL
Decouple resources from app

Database

Caching

Messaging

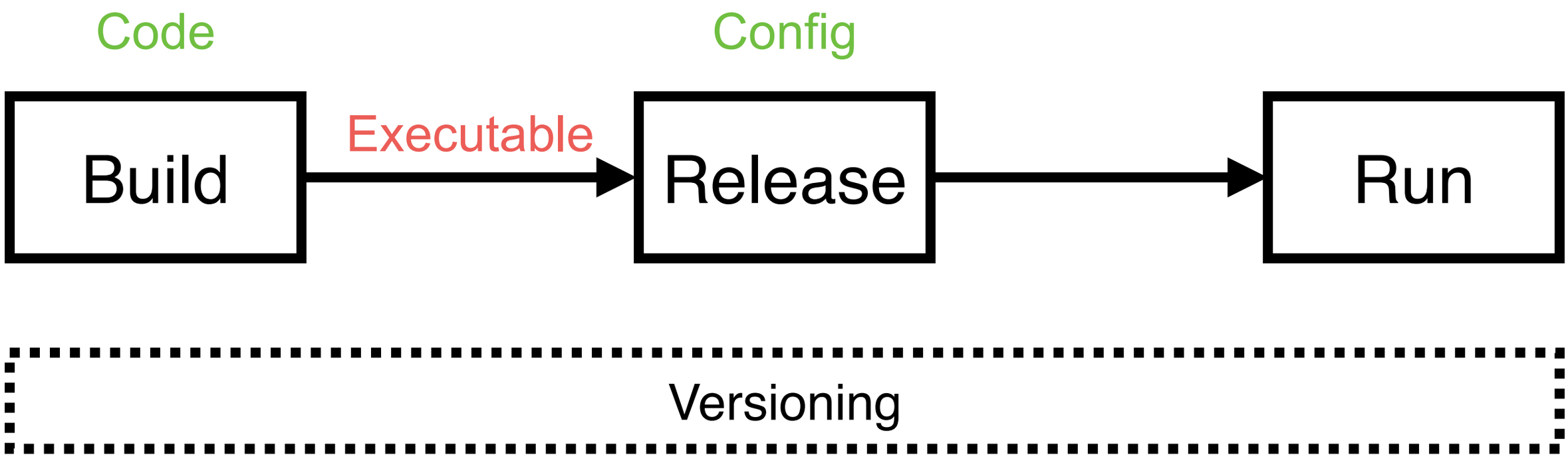
Backing services



5. Build, release, run

5. Build, release, run(time)

Strictly separate build and run stages

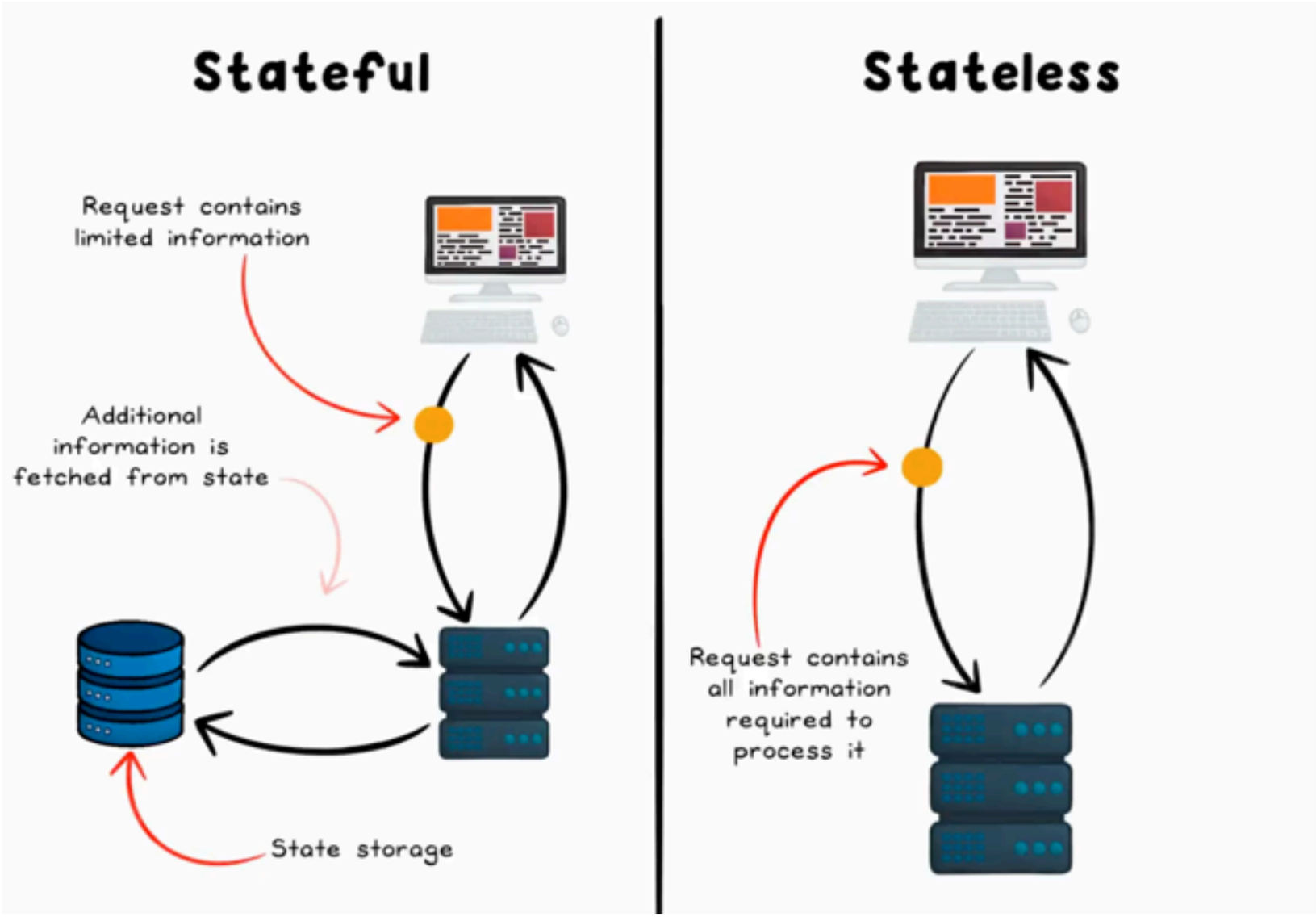


6. Processes

6. Processes

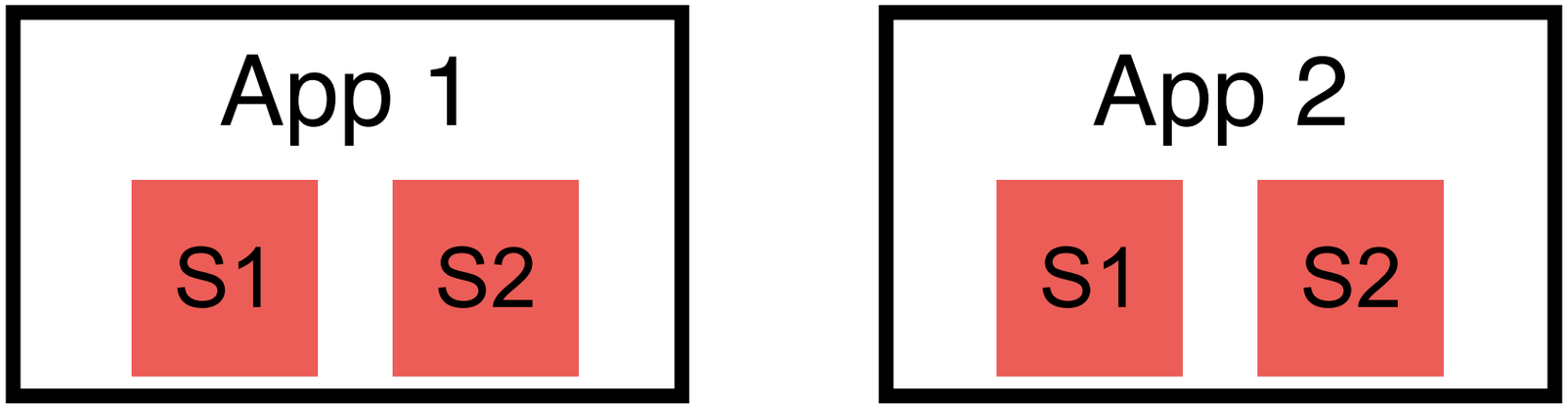
Execute the app as one or more **stateless** processes
Shared nothing
Isolated from other running services

Stateful vs Stateless

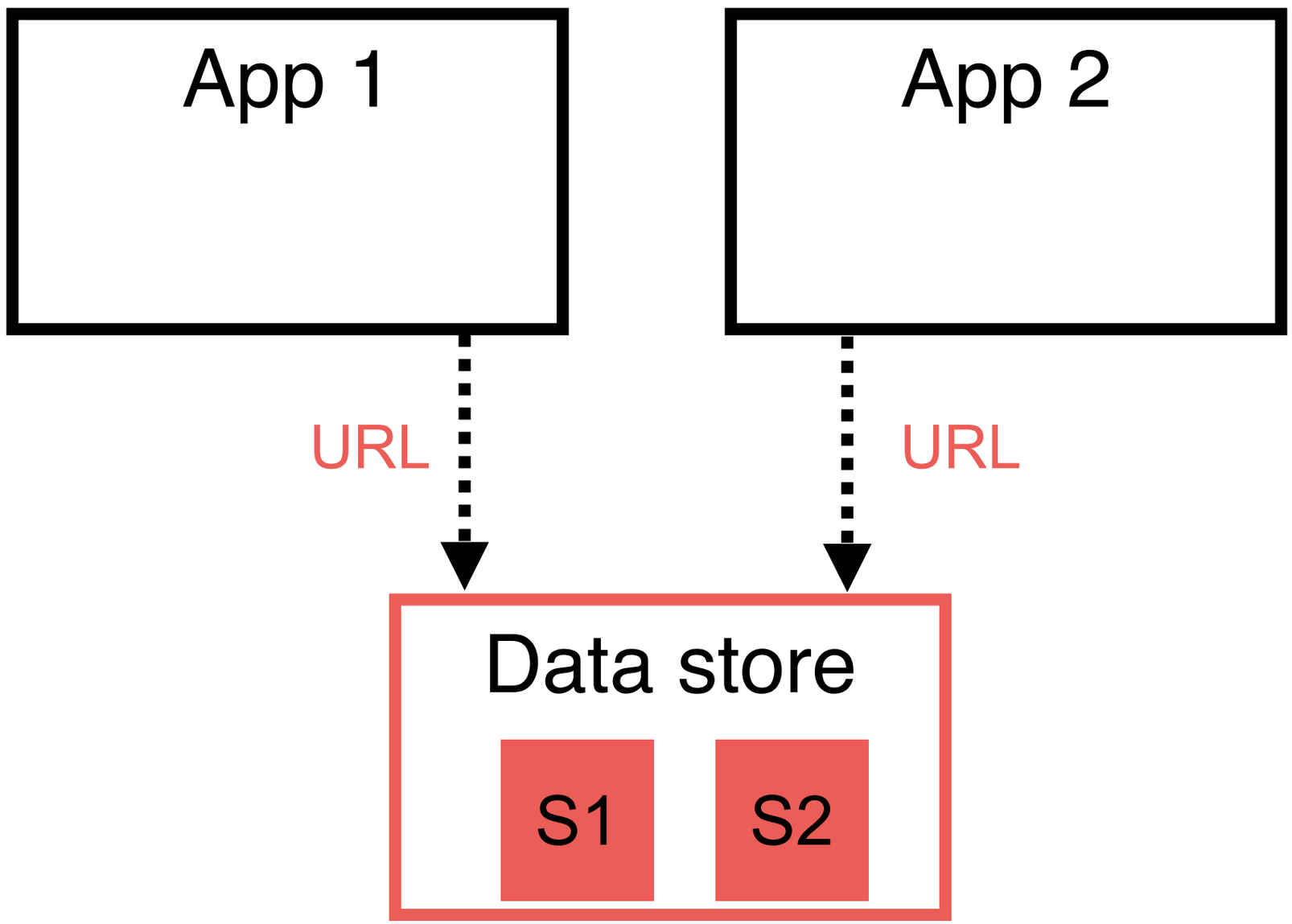


https://x.com/ChrisStaud/status/1737481416307323316

Isolated ?



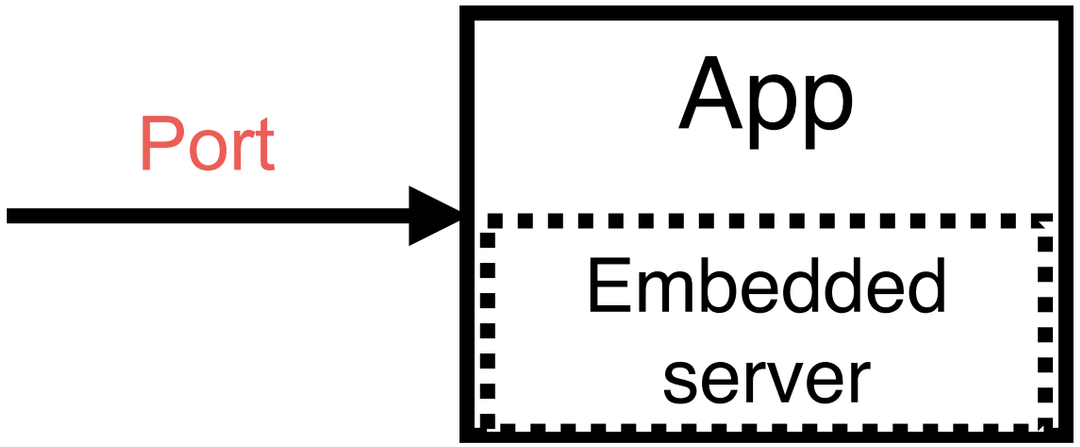
Sharing with backing service



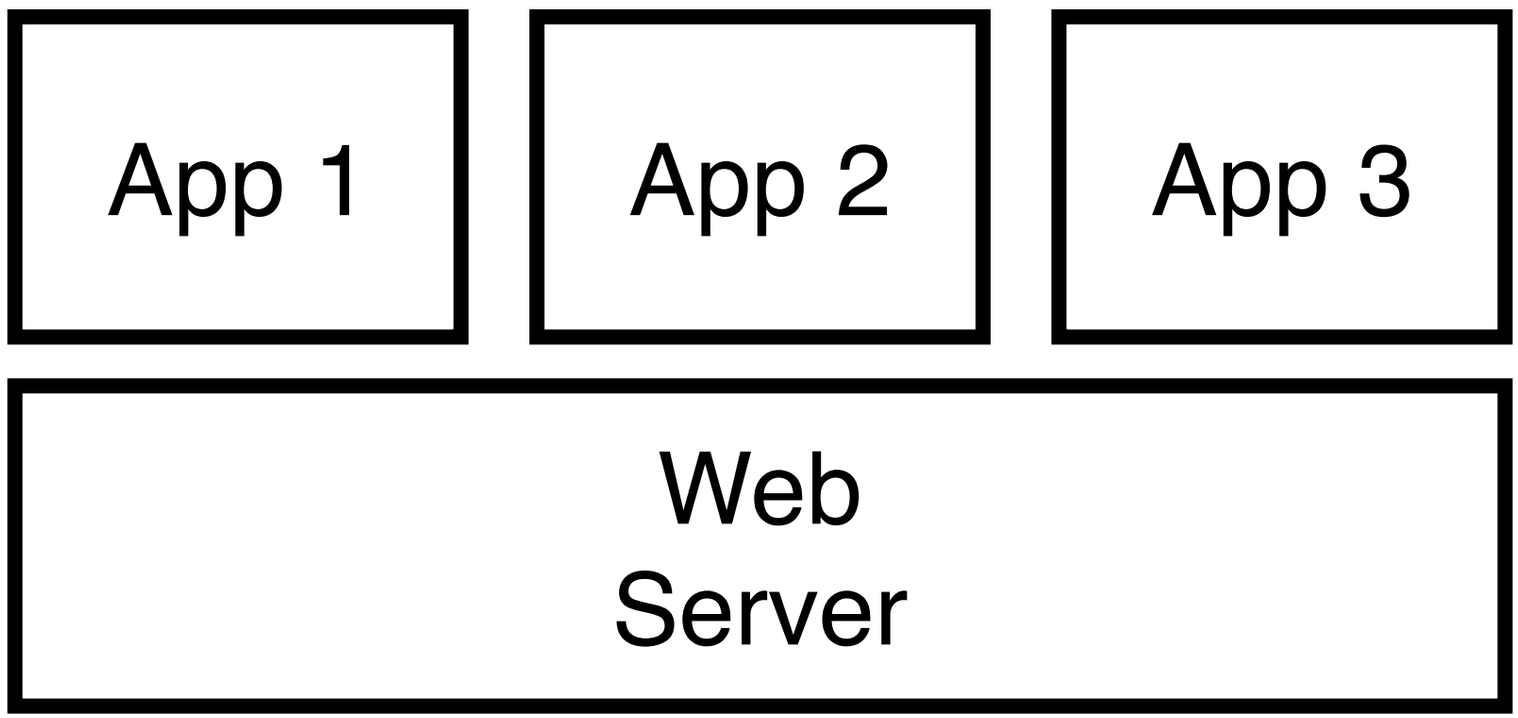
7. Port binding

7. Port binding

Export services via port binding
Self-contained



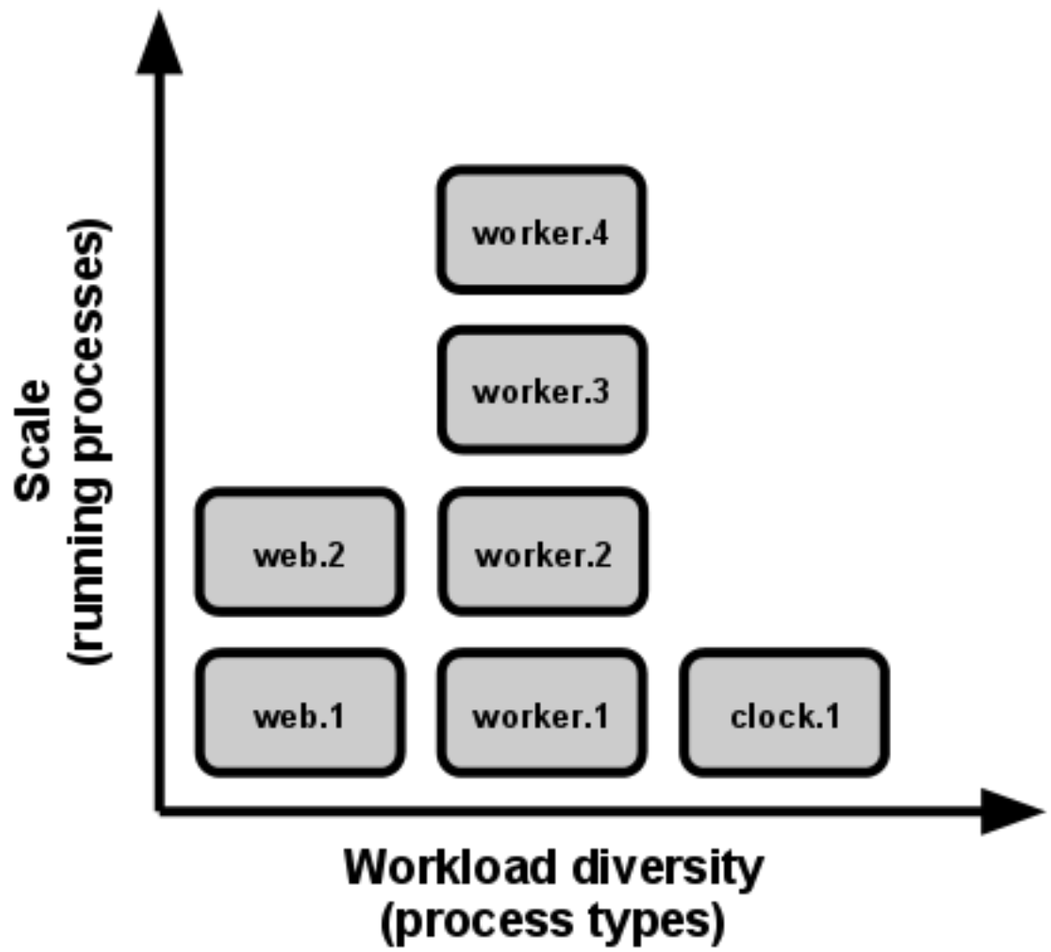
Don’t



8. Concurrency

8. Concurrency

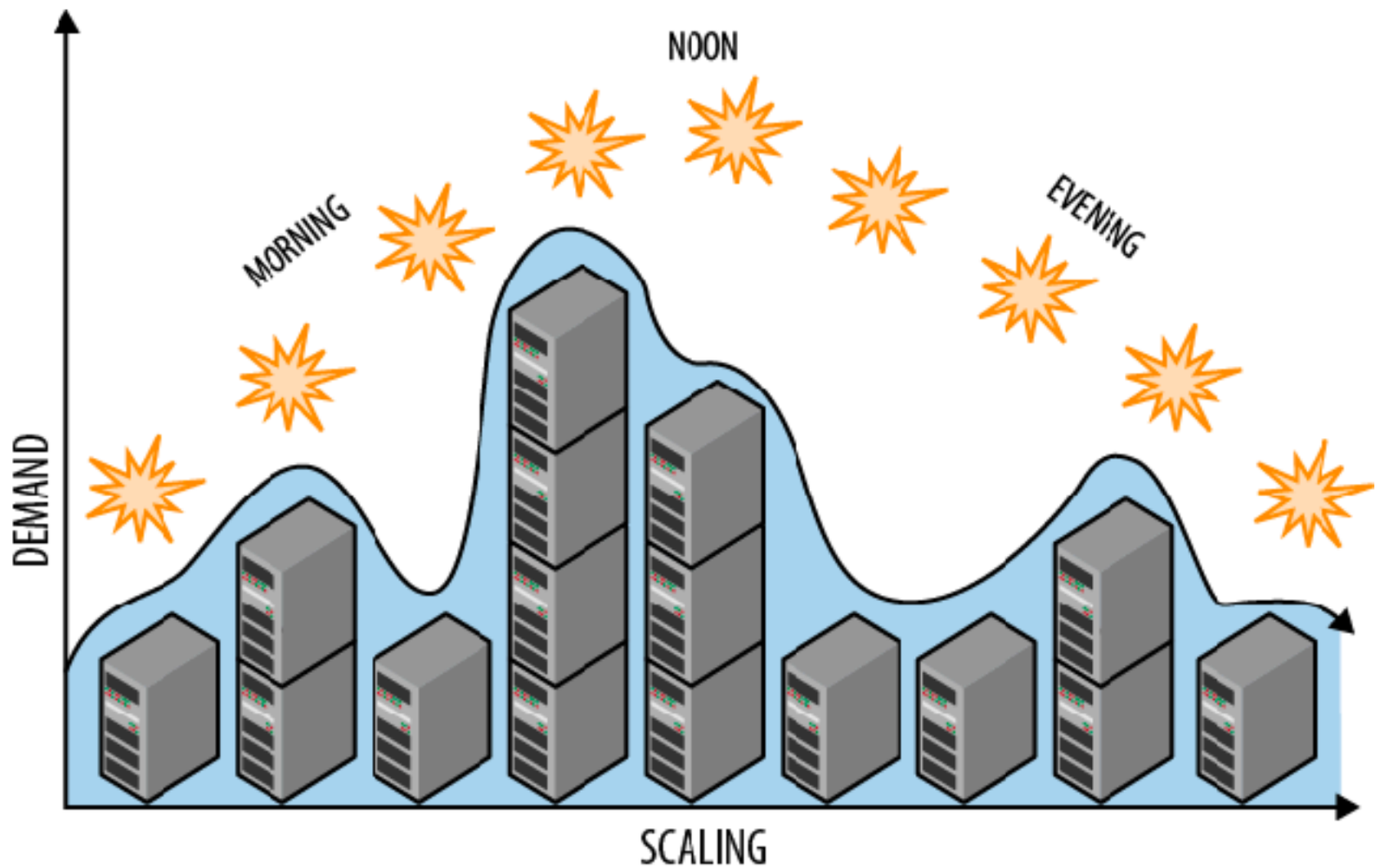
Scale out via the **process** model



Scaling ?

Elasticity Scaling

Reduce cost and improve performance by scaling automatically !!



9. Disposability

Disposable

อังกฤษ ⇄ ไทย

disposable

dəˈspōzəb(ə)l

แบบใช้แล้วทิ้ง

Bæb chî læ̂w thîng

คำแปลของ disposable

คุณศัพท์

ขายถ่ายเทได้

disposable

จัดการได้

disposable, manageable

9. Disposability

Maximize robustness
Fast startup and graceful shutdown

Fast startup to increase scalability opportunity

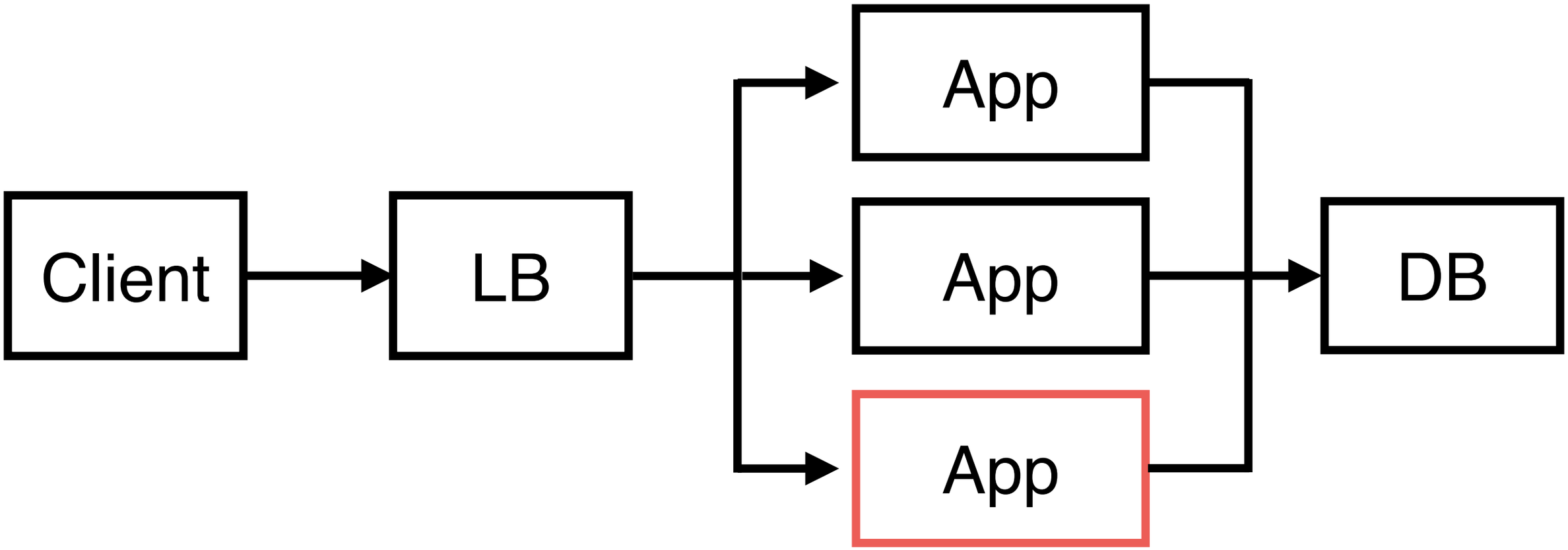
Graceful shutdown to leave the system in a correct state

Why graceful shutdown ?

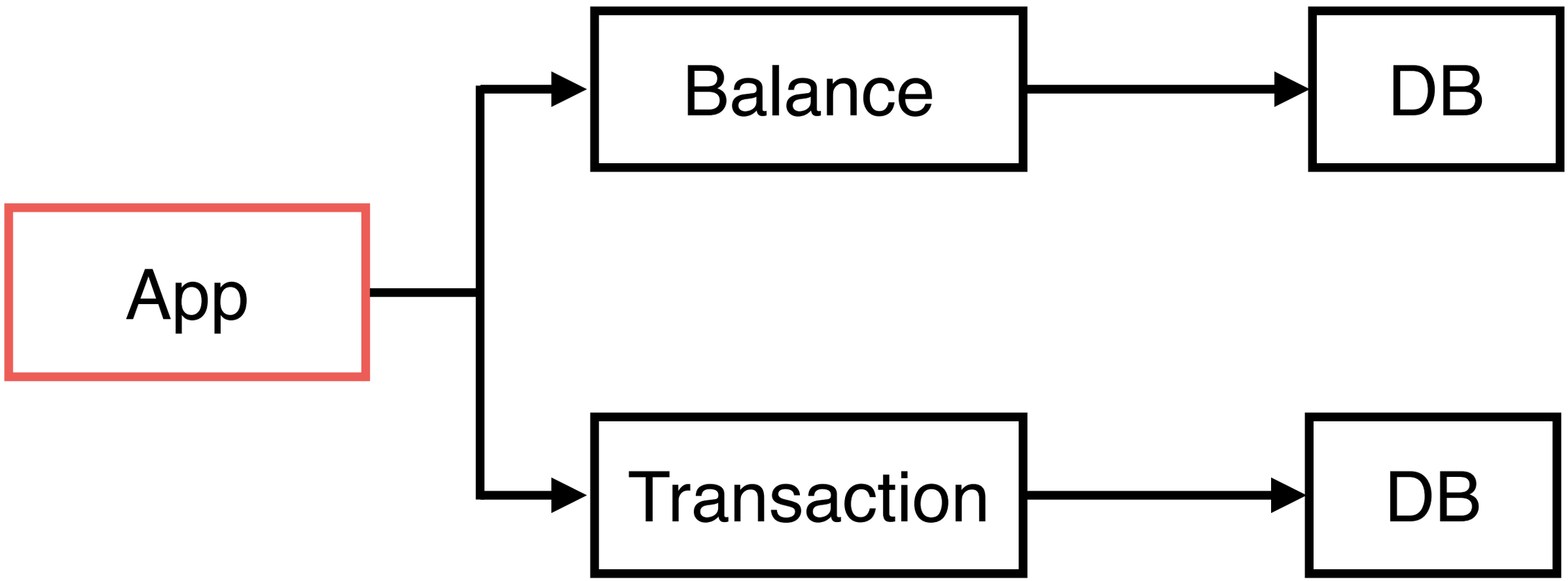
Resource cleanup
Transaction integrity

https://levelup.gitconnected.com/maximizing-resilience-with-graceful-shutdown-in-cloud-native-golang-applications-7f0b2edef4a8

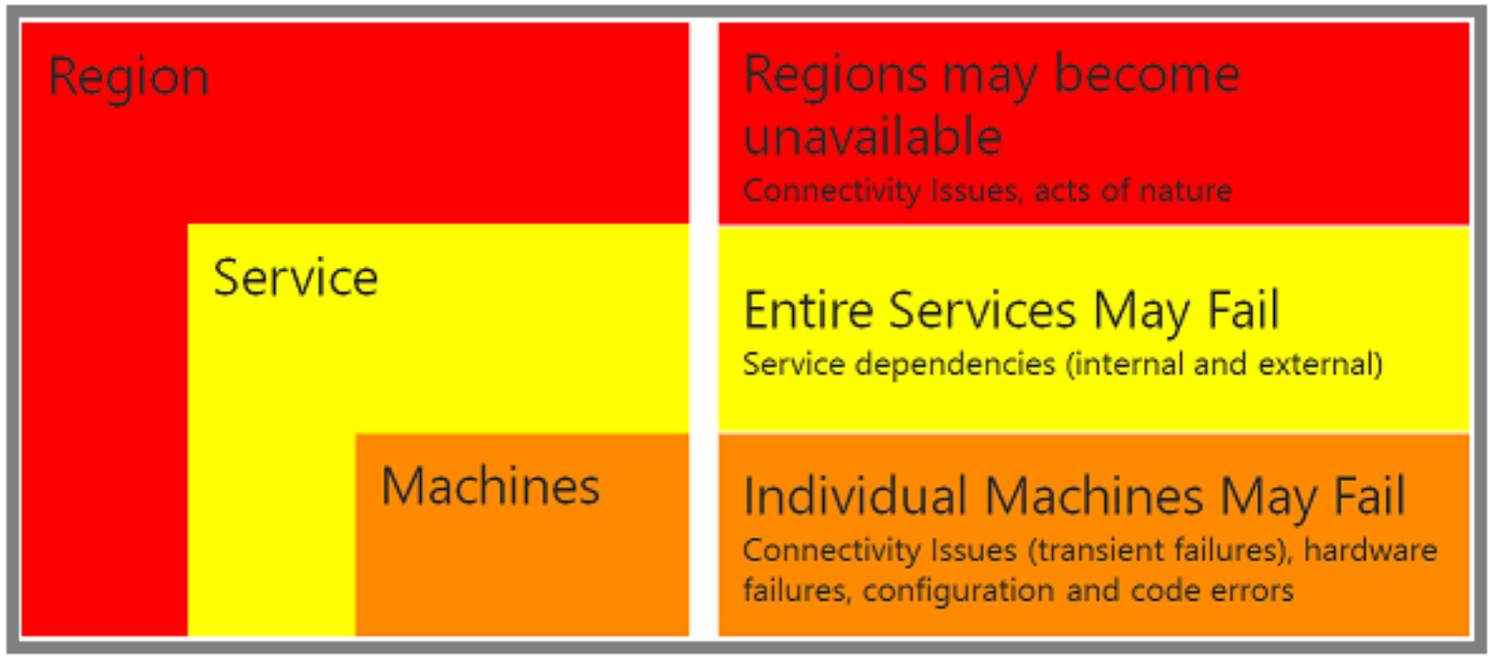
Resource cleanup ?



Transaction integrity ?



Design for Failure



https://learn.microsoft.com/en-us/aspnet/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/design-to-survive-failures

Failures ?

Network latency

Long running sync operation

Connection error

Server restart

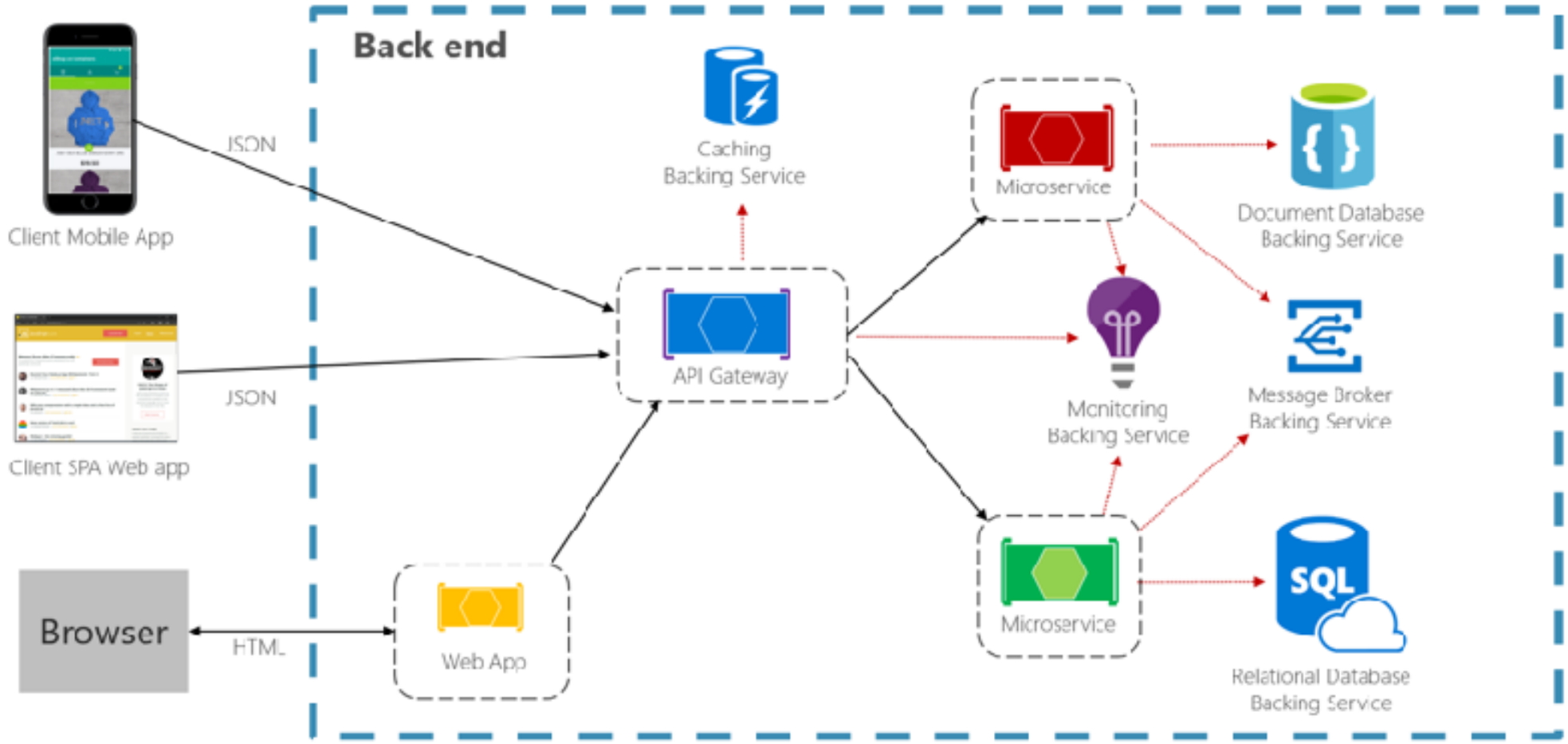
Overload traffic

Rolling update

Hardware failure

Cloud Native Resilience

Ability of your system to react to failure
System still remain functional



https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/resiliency

How to plan/design to handle failures ?

Design for Failure patterns

Timeout
Retry
Caching
Buikhead
Circuit breaker
Fallback
Messaging

Infrastructure design

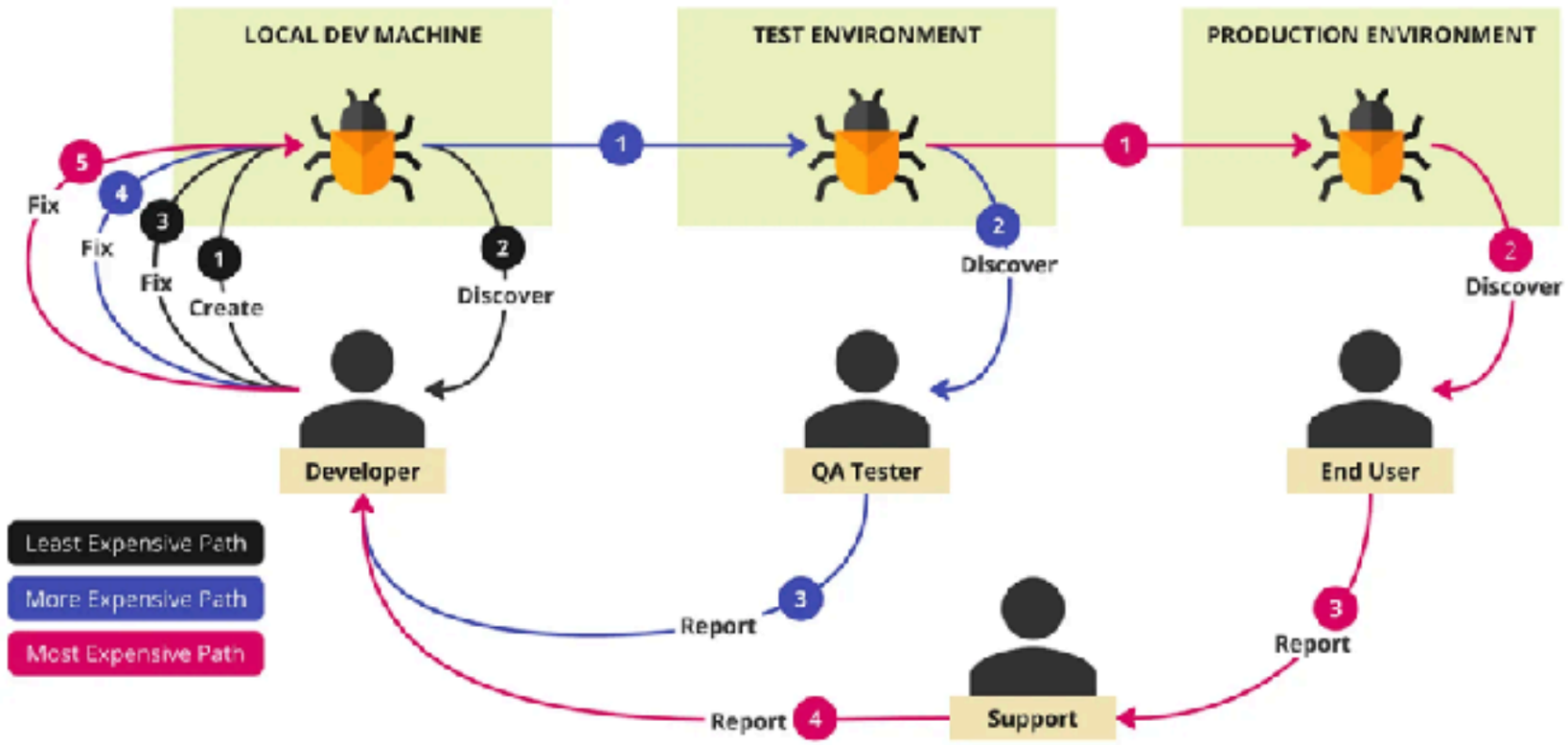
Active-standby
Auto healing
Replication
Clustering
Multi-region

10. Dev/prod parity

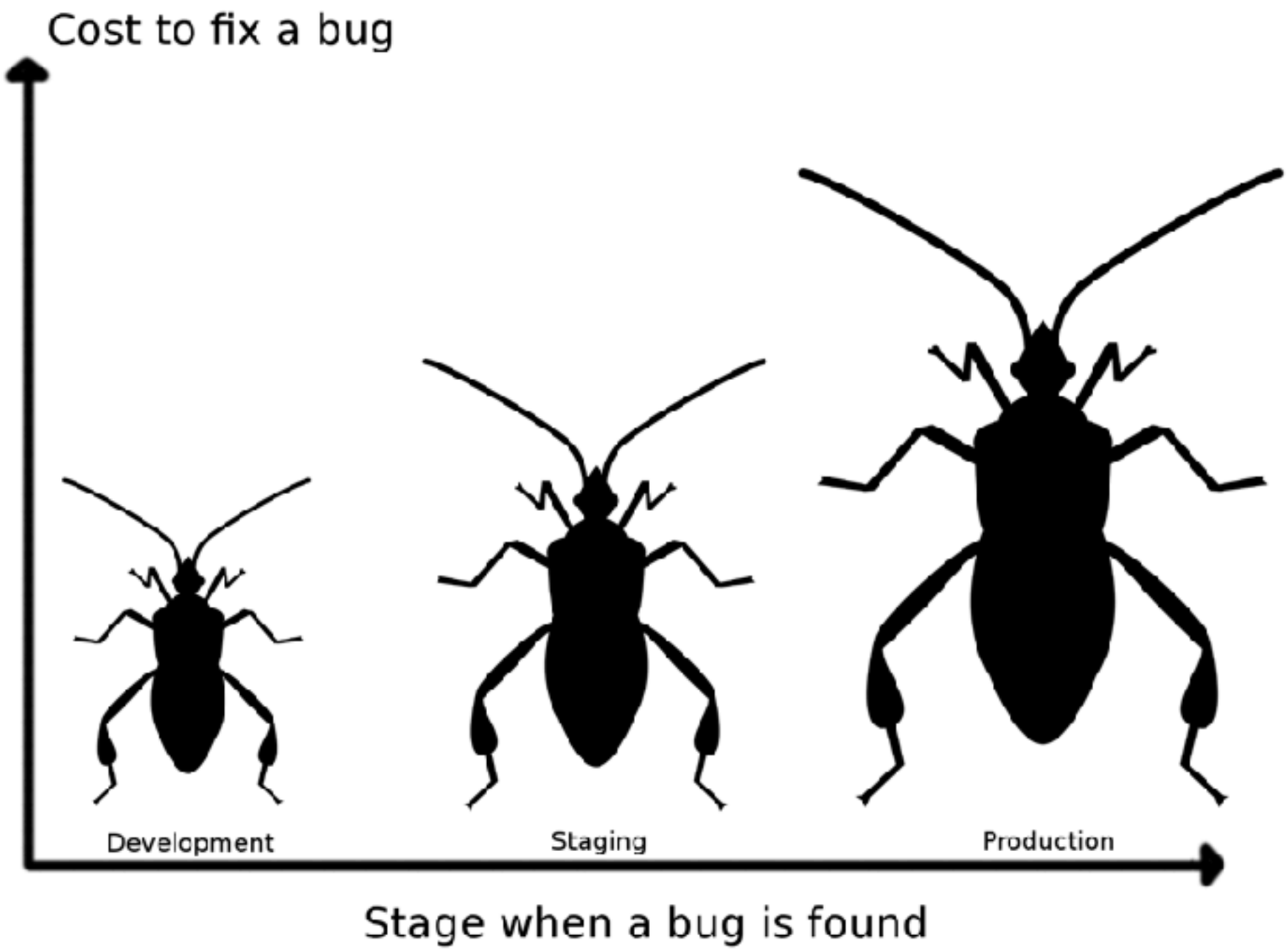
10. Dev/prod parity

Keep development, staging and production as similar as possible

Detect problems early



Cost of Bug



Gap between Dev and Prod

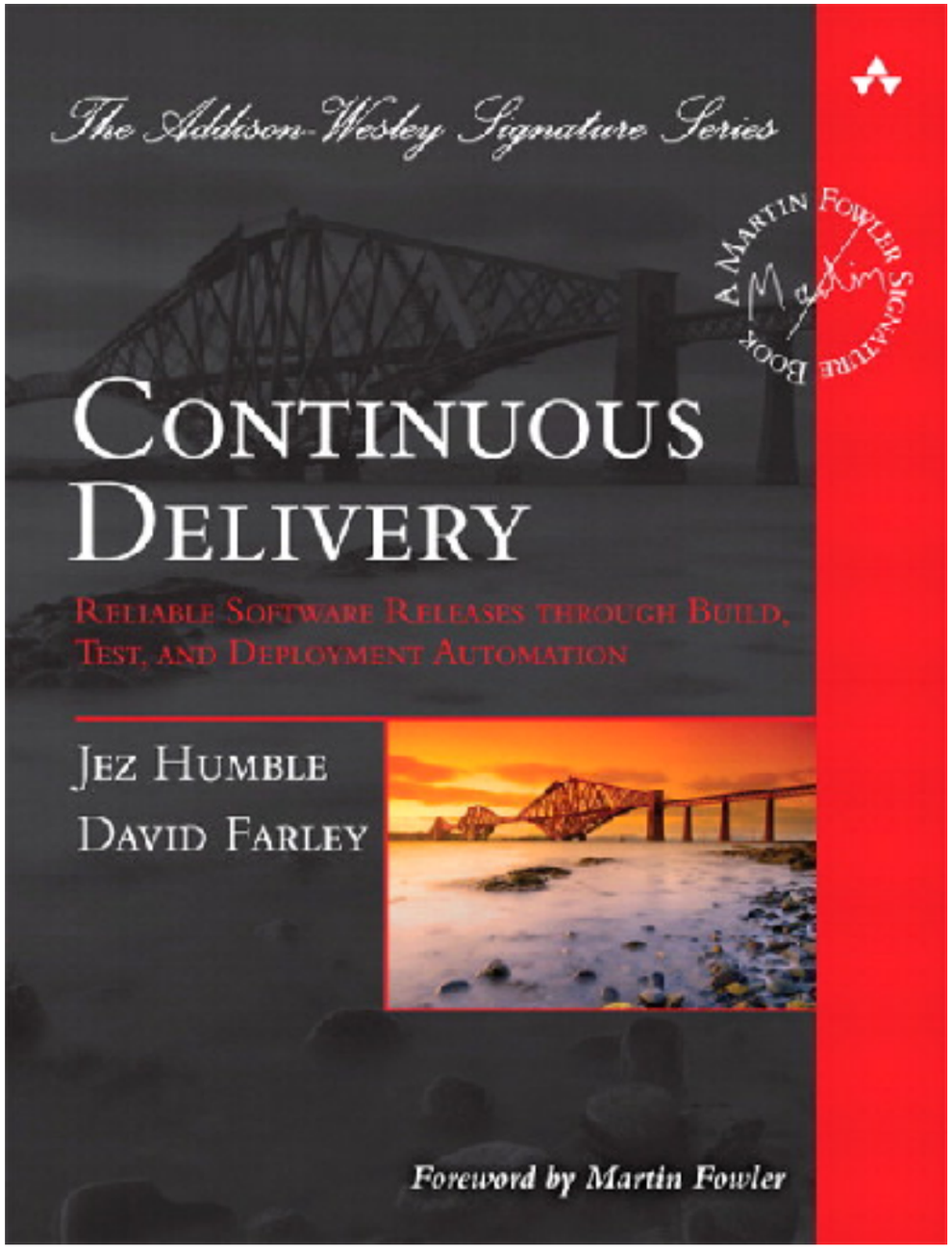
Time-gap
Personal-gap
Tool-gab

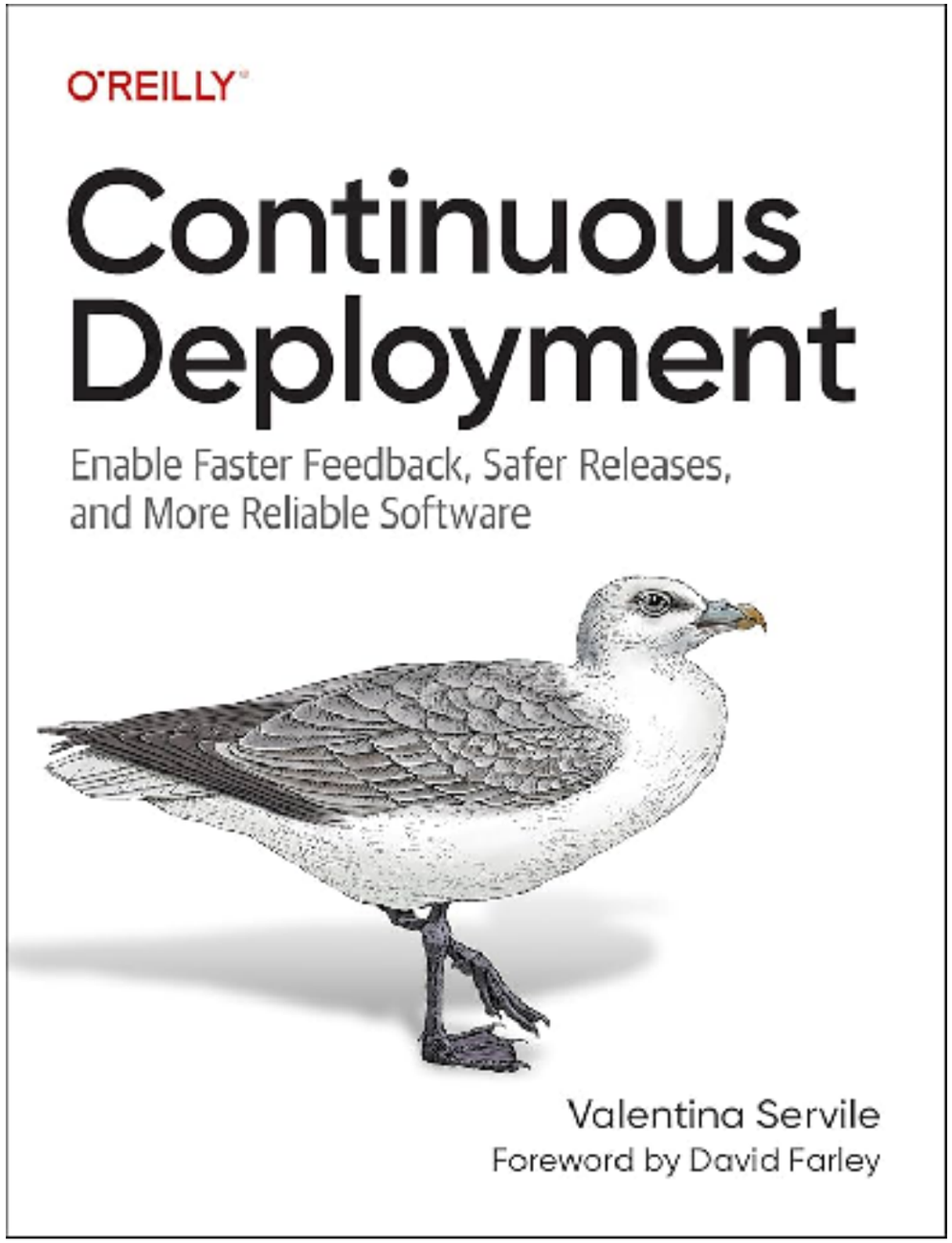
12-factor designed for **Continuous Deployment** by keeping the gab between dev and prod small

Gap between Dev and Prod

	Traditional	12-factor
Time between deploys	Weeks	Hours
Code author and code deployer	Different people	Same people
Dev vs Prod environments	Divergent	As similar as posible

Continuous Deployment





Continuous Deployment

Fast feedback
Reduce risk
Safety release
Increase reliable of software

https://martinfowler.com/bliki/FrequencyReducesDifficulty.html

Tools

Docker

Chef/puppet

Kubernetes

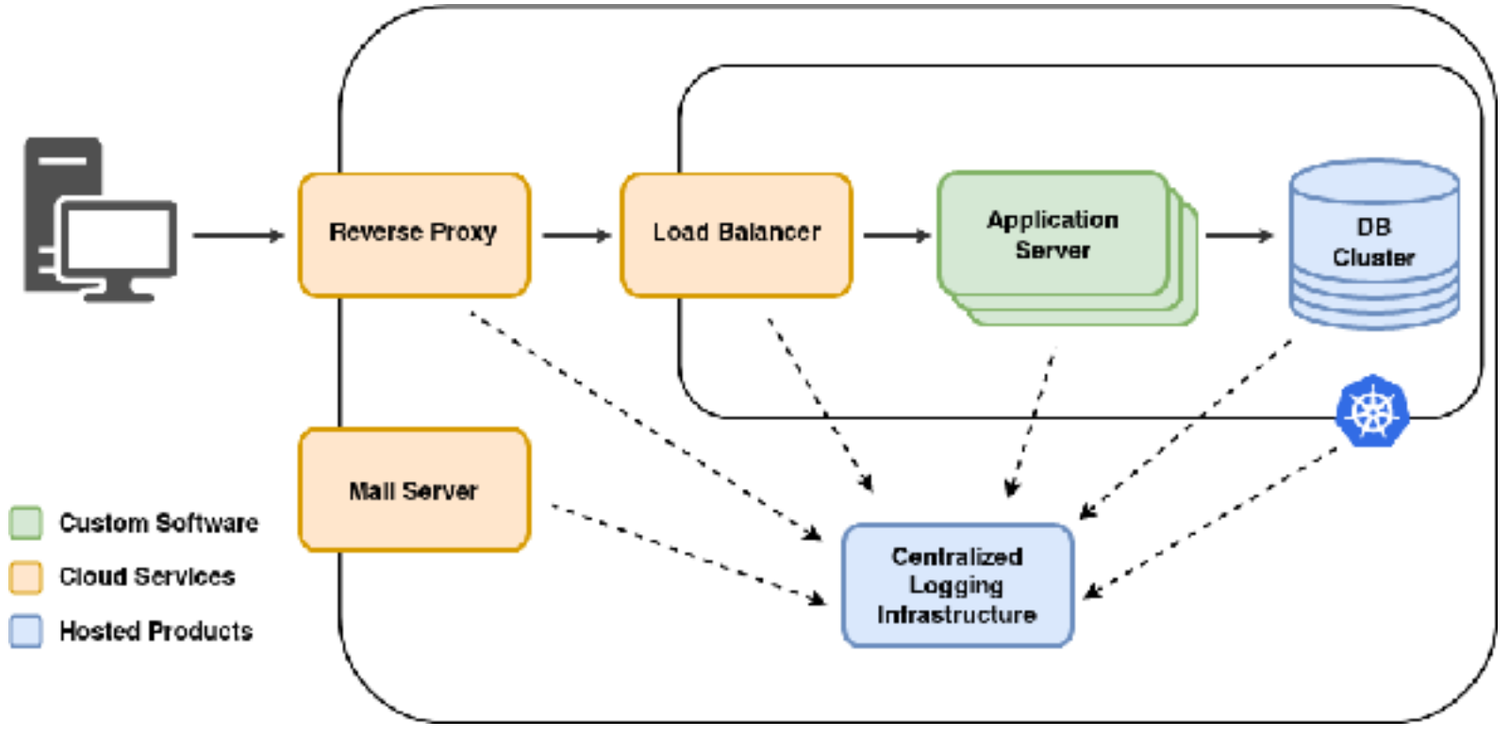
Terraform

11. Logs

11. Logs

Treat log as event streams

Logs provide visibility into the **behavior** of a running app



Structured logging !!

Structured logging

Log Formats Explained

Structured

- Usually JSON or LogFmt.
- Least human-readable.
- Easy to parse, search and filter.
- Consistent formatting.
- Faster troubleshooting due to added context.
- Automated reporting and analysis.

RECOMMENDED

Semi-structured

- Mix of structured and unstructured data.
- Human-readable.
- Formatting variability leads to inconsistency.
- Limited automation possibilities.
- Requires a more complex parsing logic.

Unstructured

- Free-form plain text messages.
- Human-readable.
- Challenging to parse.
- Requires manual interpretation.
- Suitable only for quick ad-hoc logging in development environments.

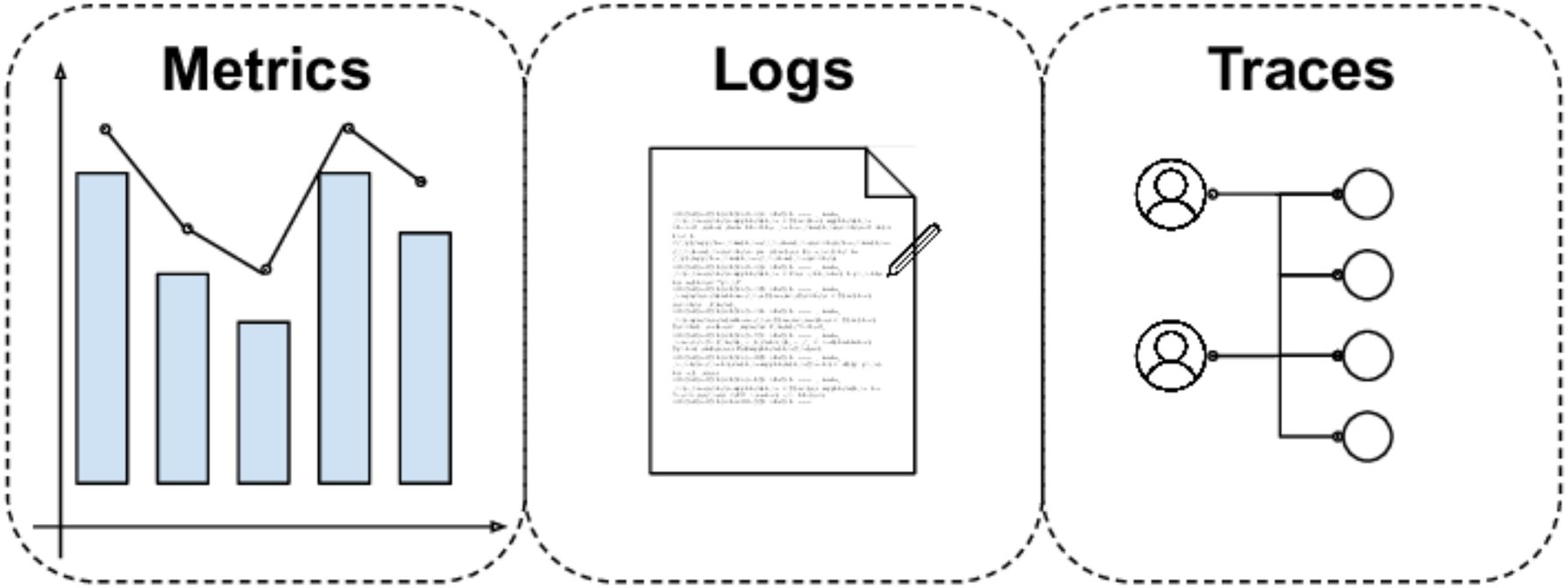
NOT RECOMMENDED

Better Stack

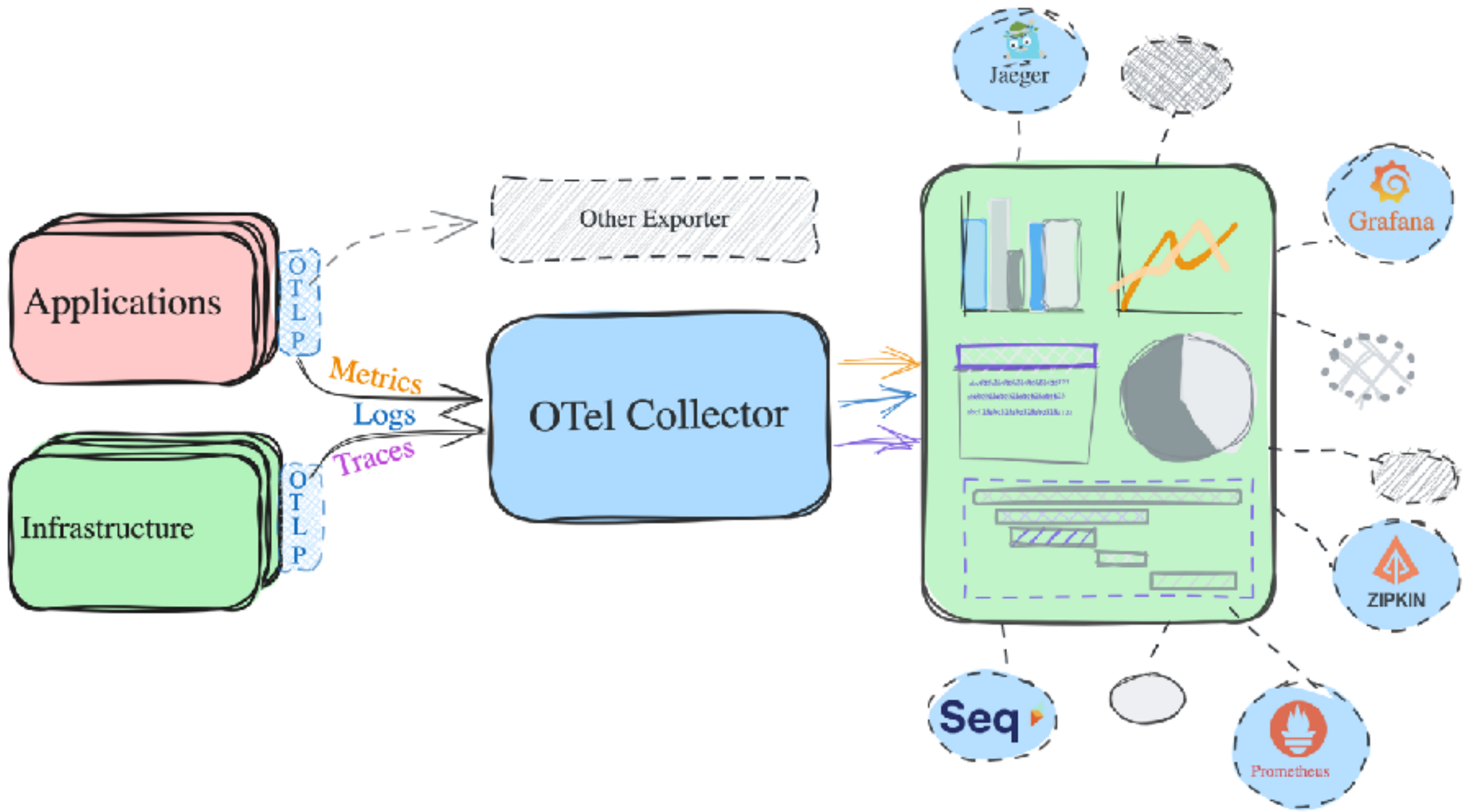
https://betterstack.com/community

https://betterstack.com/community/guides/logging/structured-logging/

Not Only Log !!



OpenTelemetry



12. Admin processes

12. Admin processes

Run admin/management tasks as one-off processes

Reduce ad-hoc tasks
Separated from app

Data migration

Data cleanup

Computing analytics

12 + 3 Factor

https://raw.githubusercontent.com/ffisk/books/master/beyond-the-twelve-factor-app.pdf

13. API First

14. Telemetry

15. Authentication and Authorization

Workshop

C#

Go

Docker

Q/A