Distributed Systems

Chun-Feng Liao

廖峻鋒

Department of Computer Science

National Chengchi University

Distributed Systems

Cloud Native

Chun-Feng Liao

廖峻鋒

Dept. of Computer Science

National Chengchi University

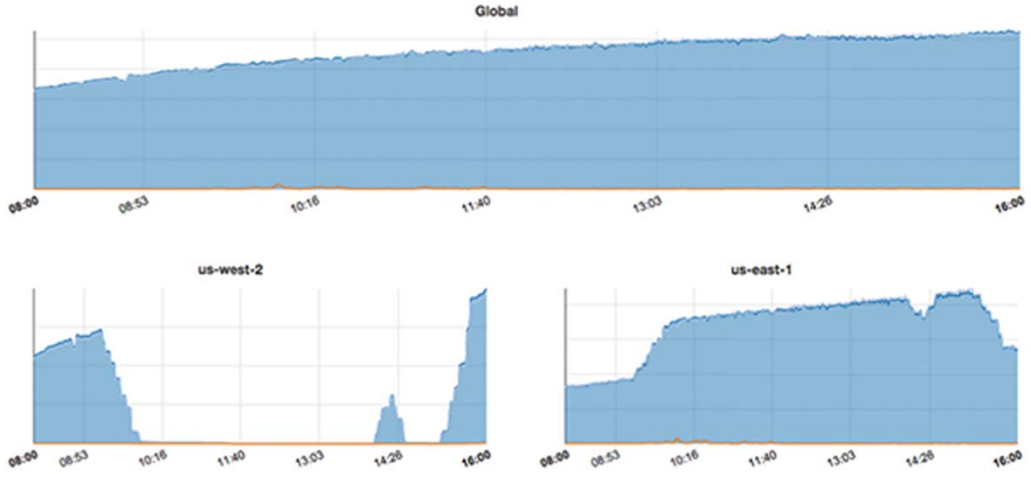
- From cloud to cloud native
- Cloud and cloud native
  - Cloud is where
  - Cloud native is how

Today's application requirements

- Zero downtime
- Shortened feedback cycles
  - release code frequently
- Mobile and multi-device / IoT support
- Data-driven
  - Volumes of data is increasing
  - Data tend to distributed and localized

Case: AWS outage in 2015

- On Sep. 20, 2015, AWS experienced a significant outage
  - Netflix, Airbnb, Nest, IMDb, and more all experienced downtime
- Netflix quickly recovers from outage
  - 平時已充分進行outage的演習，迅速於未受影響的區域重建服務



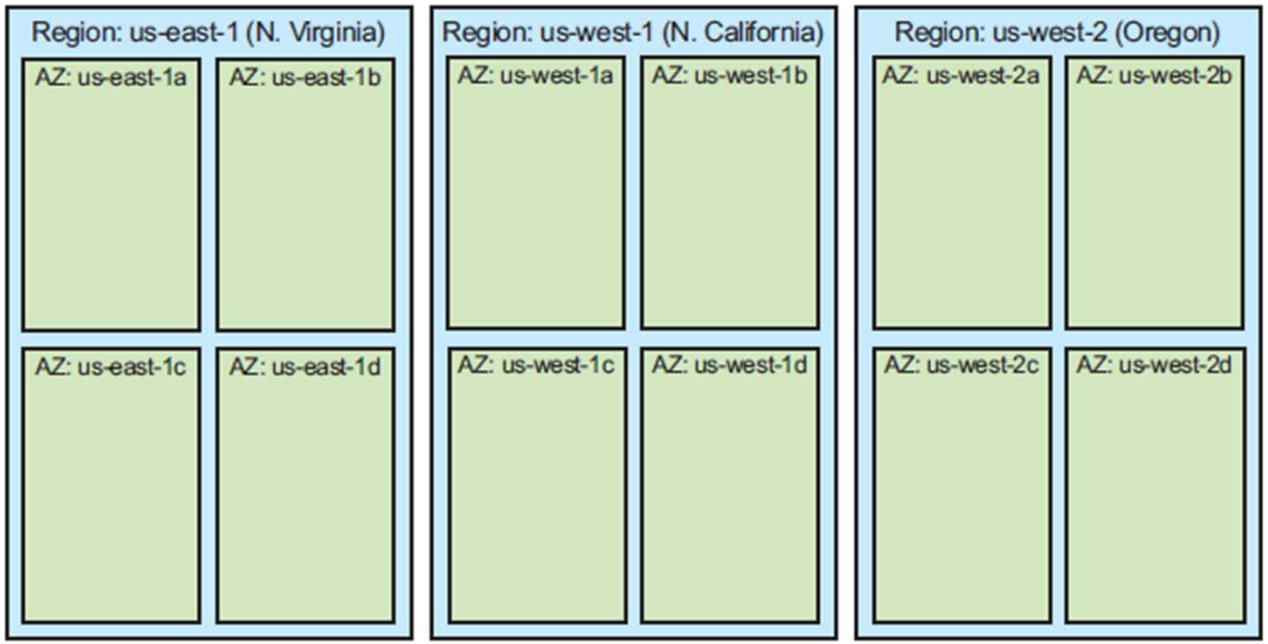
Chaos Kong exercise in progress

https://netflixtechblog.com/chaos-engineering-upgraded-878d341f15fa

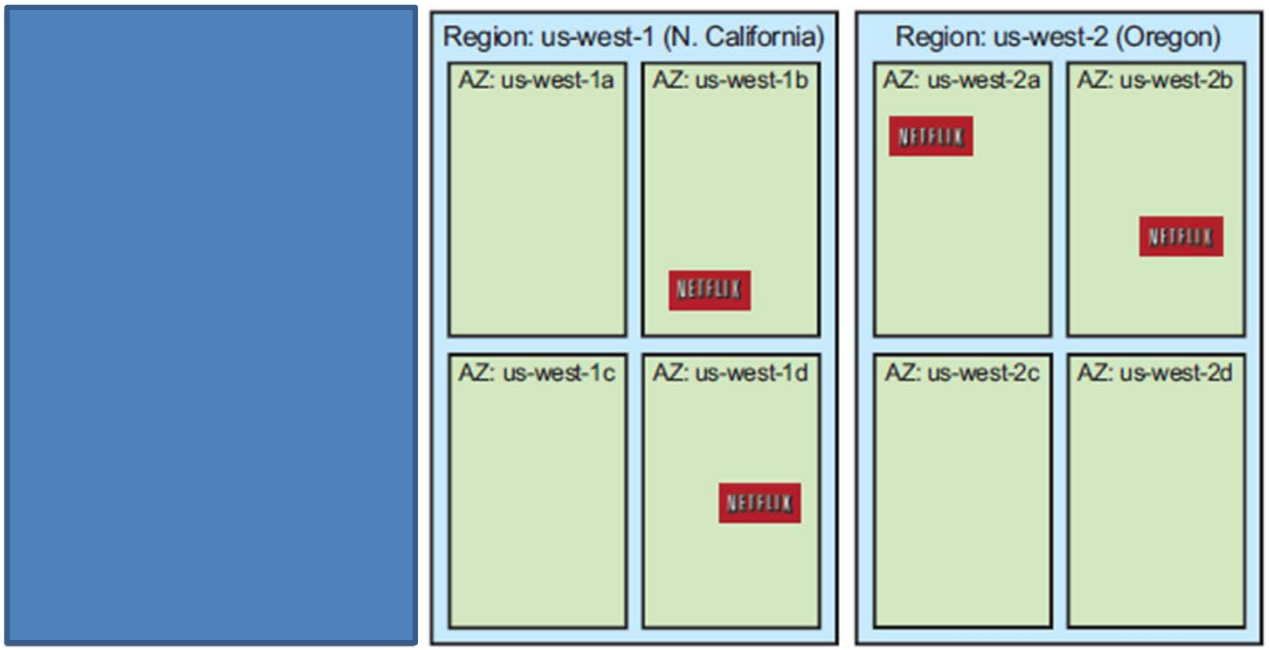
Chaos Monkey. The service pseudo-randomly plucks a server from our production deployment on AWS and kills it.

Case: AWS outage in 2015

- AWS結構
  - 區分多個region與AZ (availability zones)



Case: AWS outage in 2015



Case: AWS outage in 2015

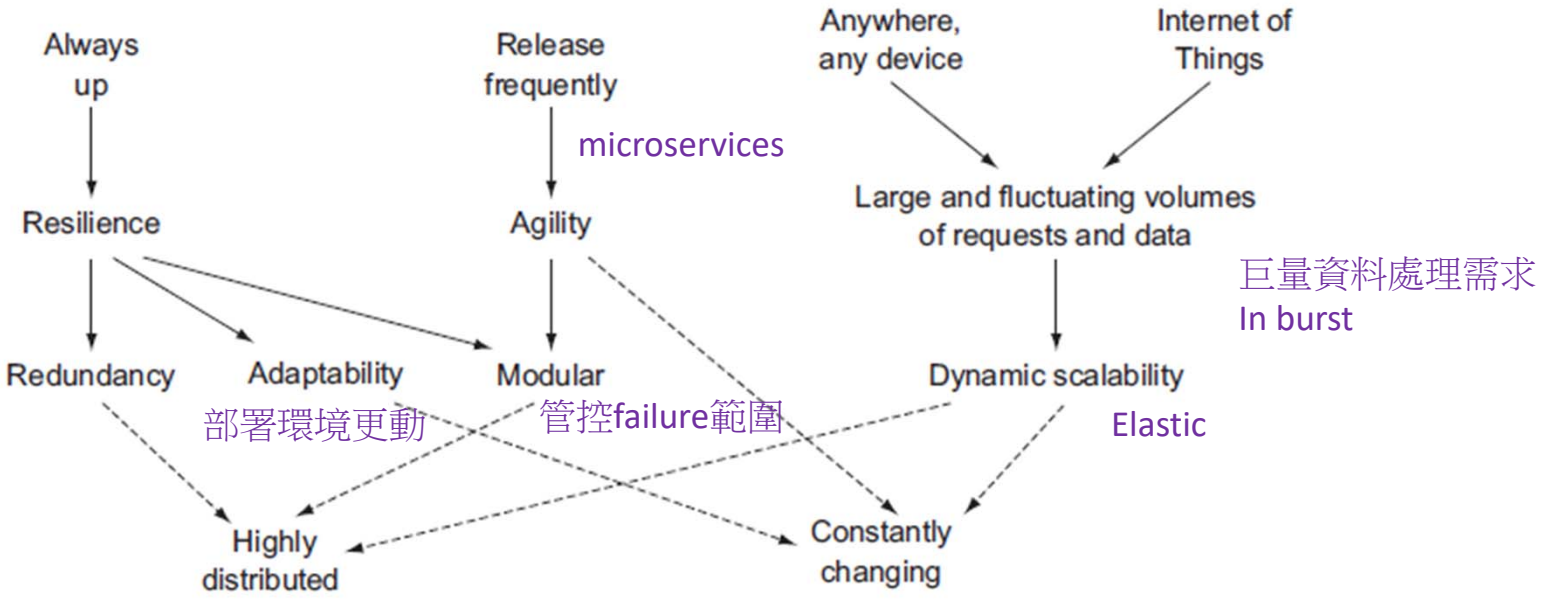
- Cloud-native software
  - Designed to anticipate failure（視失效為常態)
  - Remain stable when the infrastructure outages or changing
- 觀察點
  - Reliable services over a unreliable infrastructure

Definition

- Cloud-native software is
  - highly distributed,
  - must operate in a constantly changing environment, and
  - is itself constantly changing

Cloud Native Features

- Highly distributed and constantly changing



Cornelia Davis : Sr. Director of Technology at Pivotal Software

Cloud Native: 各種説法

- 緣起
  - Cloud Native概念最早由2013 Matt Stine (Pivotal)提出
  - Matt Stine在2015更新
    - The Twelve-Factor App
    - Microservices
    - Self-Service Agile Infrastructure
    - API-based Collaboration
    - Anti-Fragility (robust)

Cloud Native: 各種說法

- 2017 年 Matt Stine 接受 InfoQ 訪問時之修正
  - Modularity
  - Observability
  - Deployability
  - Testability
  - Replaceability
  - Disposability

Cloud Native: 各種説法

- Pivotal 2019
  - DevOps
  - Continuous Delivery
  - Microservices
  - Containers

Cloud Native: 各種説法

- CNCF (Cloud Native Computing Foundation) 定義原文
  - Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds.
    - Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.
  - These techniques enable loosely coupled systems that are resilient, manageable, and observable.
  - Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil

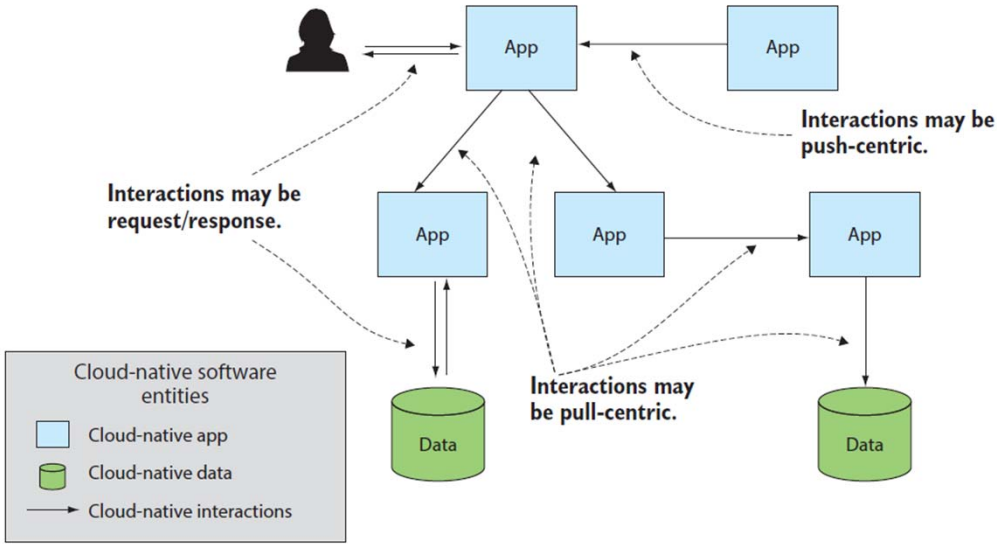
https://github.com/cncf/toc/blob/master/DEFINITION.md

五大原則

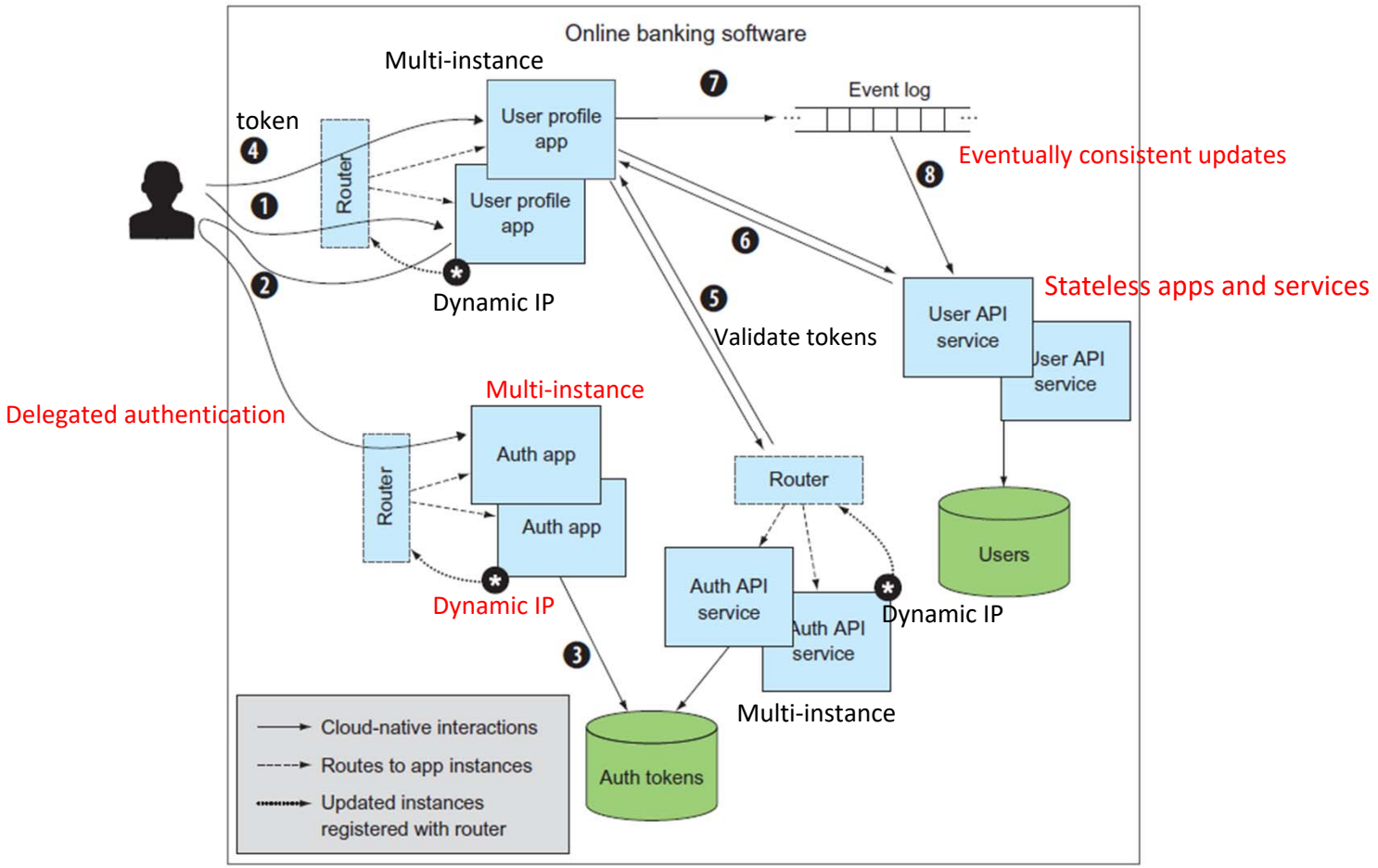
- Containerization
- Dynamic management
- Microservices
- Automation
- Orchestration

Characteristics of Cloud Native Systems

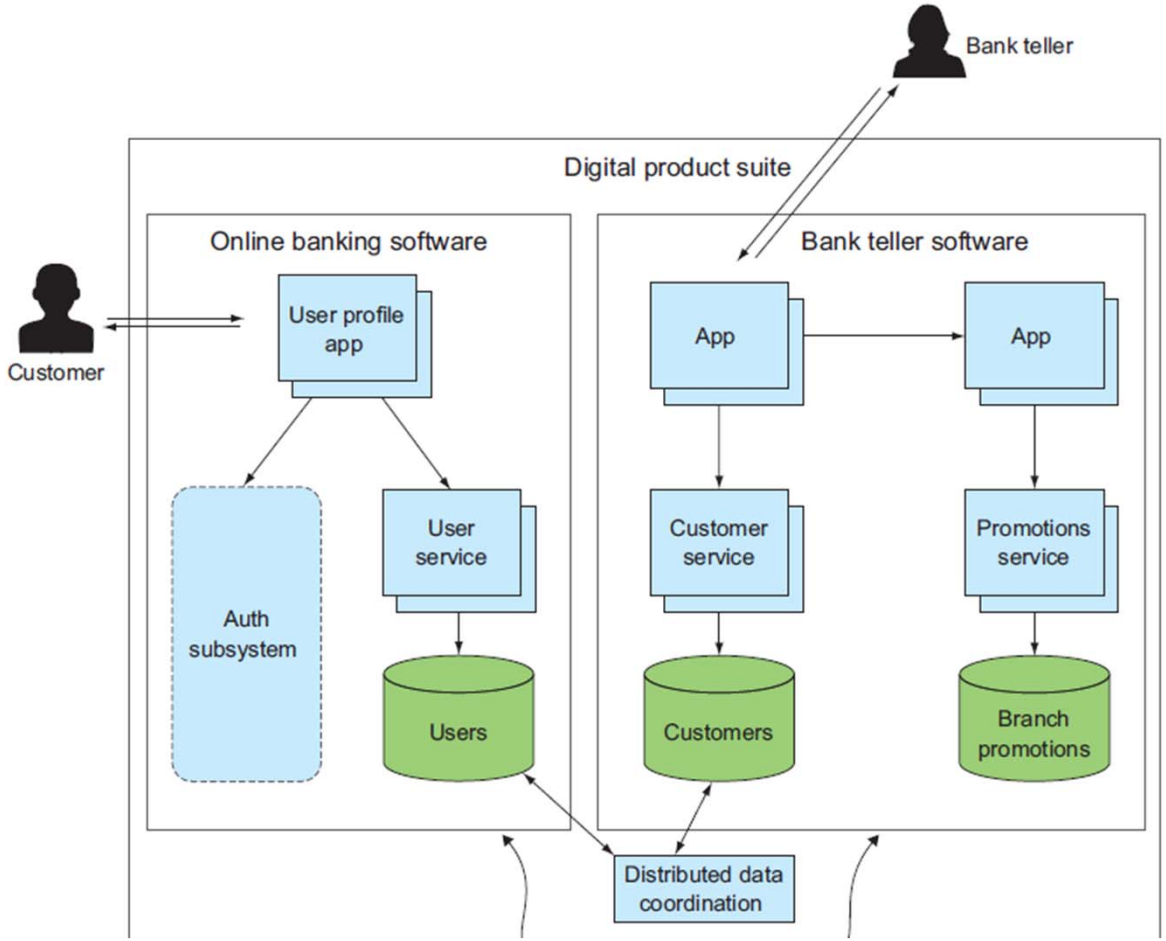
- Cloud native app
  - Stateless, redundant, scalable, dynamic deployable
- Cloud native data
  - No centralized database
  - Event sourcing
    - Treating state as an outcome of a series of modifications
  - Query/Write operations are typically separated
    - Query: catchable
    - Write: eventually consistent
- Cloud native interactions
  - Late binding → Indirect interactions
    - Circuit breakers
    - Proxy (dynamic routing)



A Cloud Native System Example



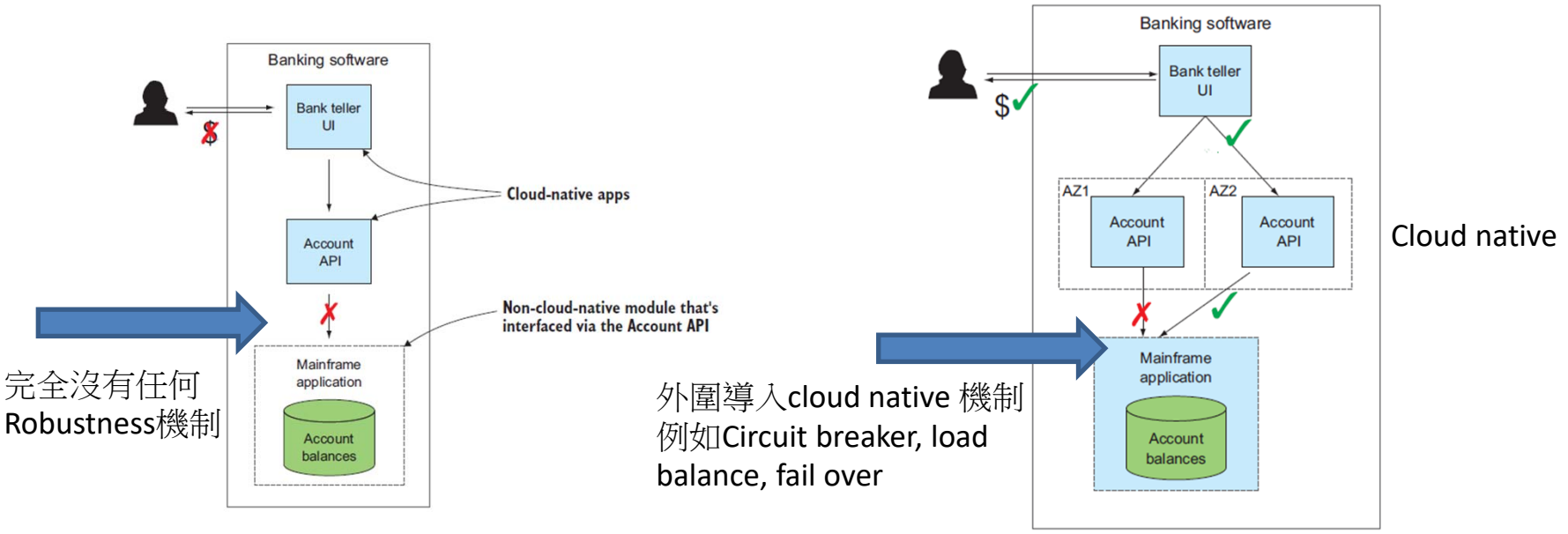
A Cloud Native System Example (2)



Cloud native 系統本身必須設計處理跨服務資料同步機制
(Eventually Consistent)

Partial Cloud Native

- Cloud native 系統與其它系統容易結合混用
  - Stateless and loosely coupled services
  - 既有系統改用Cloud Native架構時，可漸進式修改，新舊並存
    - 即使只有部份改用Cloud Native也可享受到好處
    - Netflix花了7年的時間migrate到cloud native



https://about.netflix.com/en/news/completing-the-netflix-cloud-migration

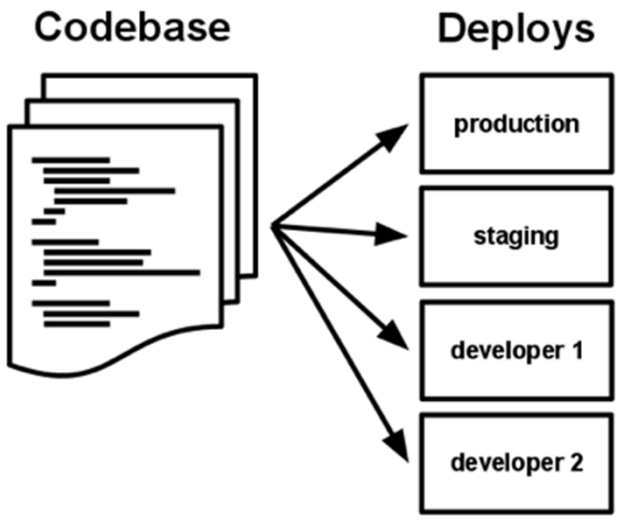
Twelve Factor App

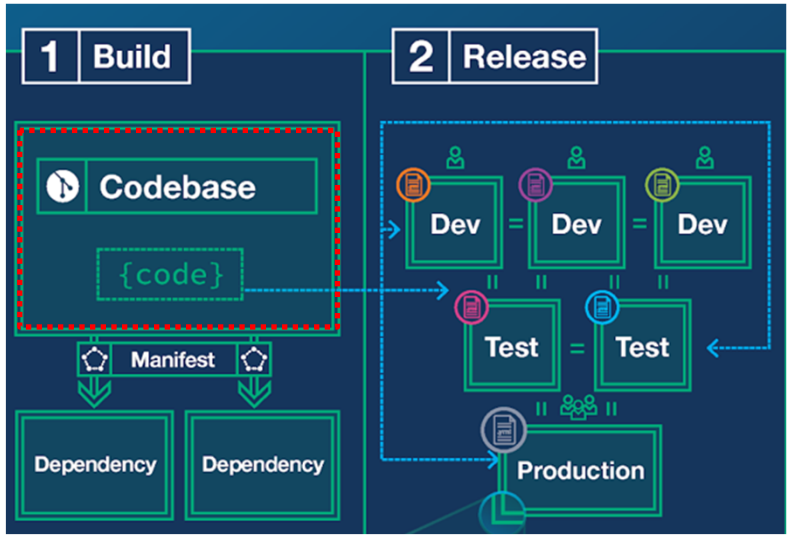
https://12factor.net/

- Codebase

A *deploy* is a running instance of the app.

  - One codebase tracked in revision control, many deploys
  - 使用版本控制系統
  - 在不同的環境會對應到同一份codebase (的不同版本)

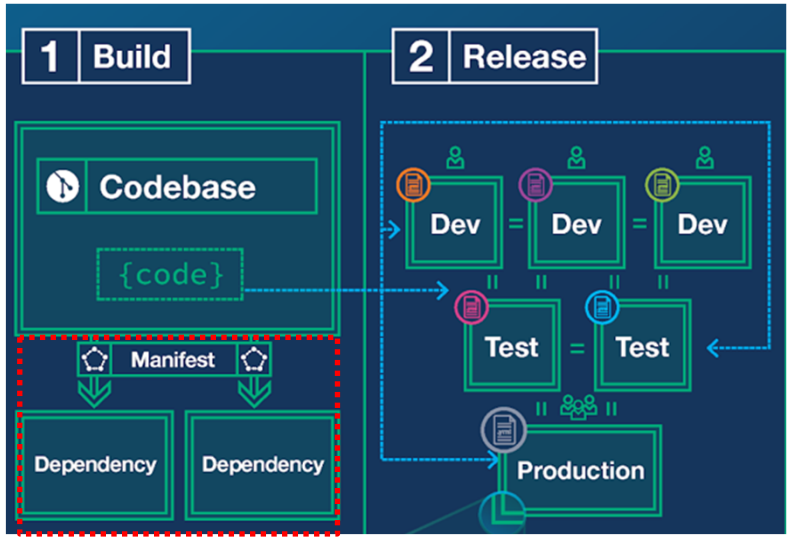




https://marcus116.blogspot.com/2020/09/architecture-12-factor-app.html

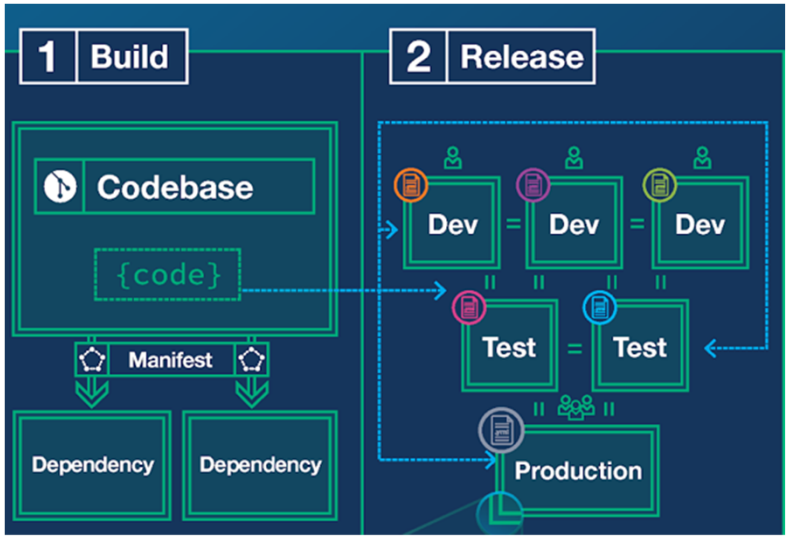
Twelve Factor App

- Dependencies
  - Explicitly declare and isolate dependencies
  - 所依賴的函式庫要明確宣告在Manifest中
    - JS: package.json
    - Java: build.gradle; pom.xml



Twelve Factor App

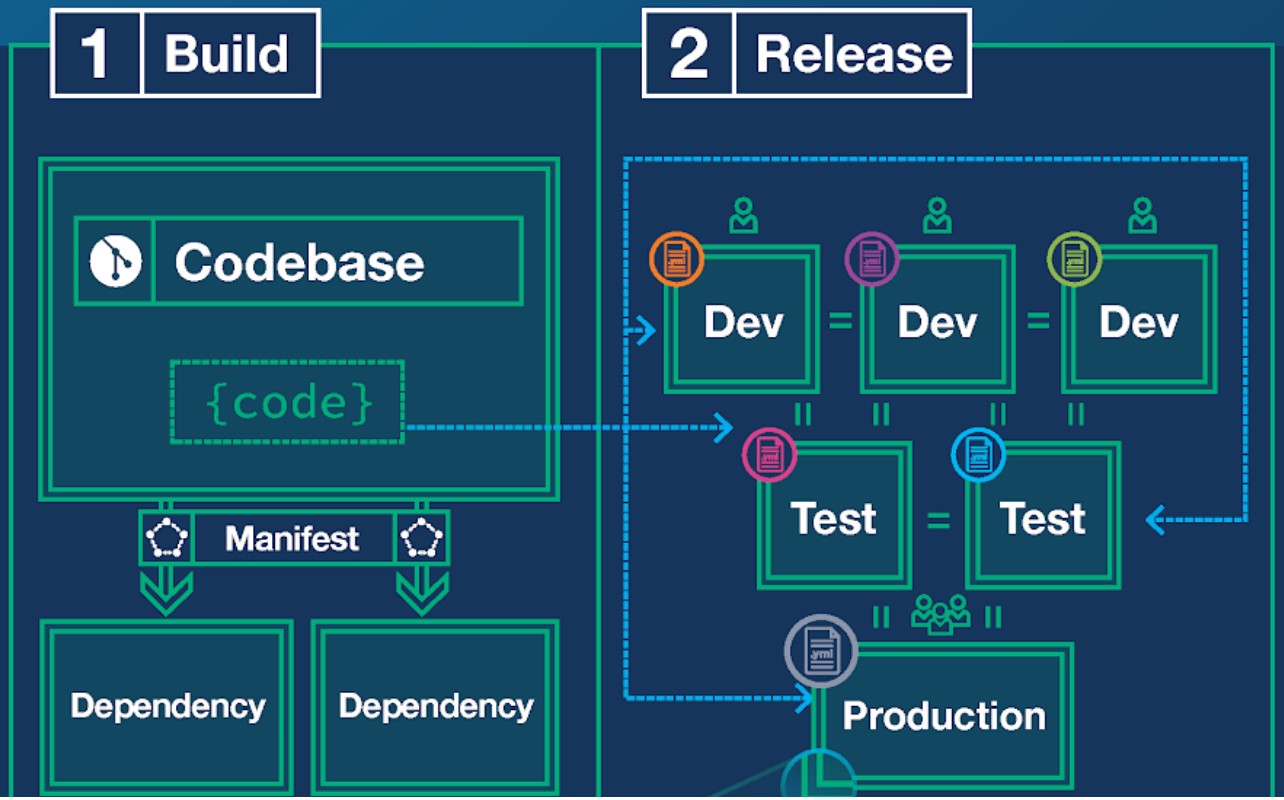
- Config
  - Store config in the environment (環境變數)
    - 應用程式的設定拉出來透過環境變數修改
    - 例: docker run --rm -it --name db -e MYSQL_ROOT_PASSWORD=passpercona



An app's *config* is everything that is likely to vary between deploys

The 12-Factor App

Modern web applications run in heterogeneous environments, scale elastically, update frequently, and depend on independently deployed backing services. Modern application architectures and development practices must be designed accordingly. The PaaS-masters at Heroku summarized lessons learned from building hundreds of cloud-native applications into the twelve factors visualized below.



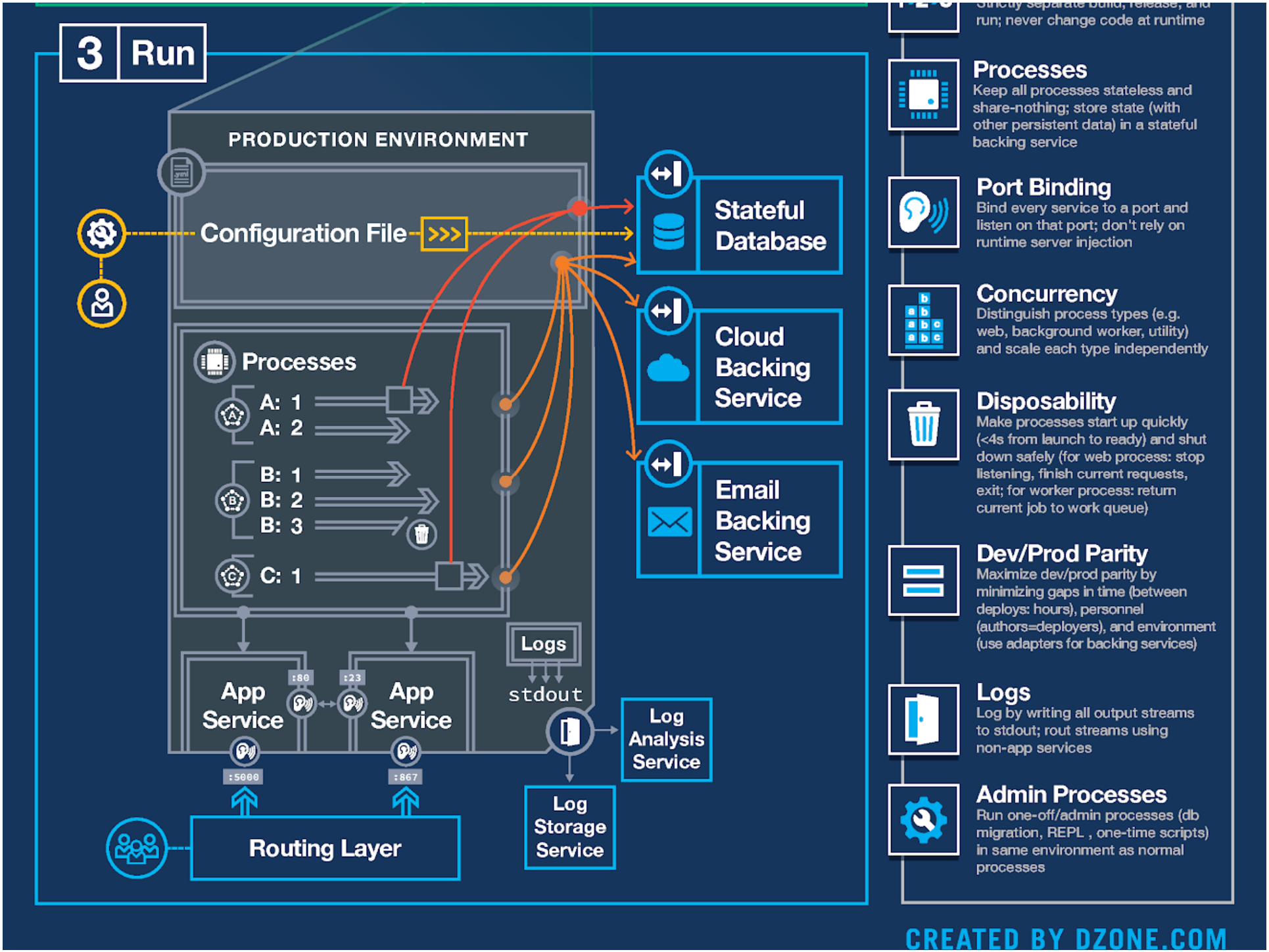
the 12 factors

Codebase
One codebase, many deploys, strict version control

Dependencies
Explicitly declare and isolate dependencies

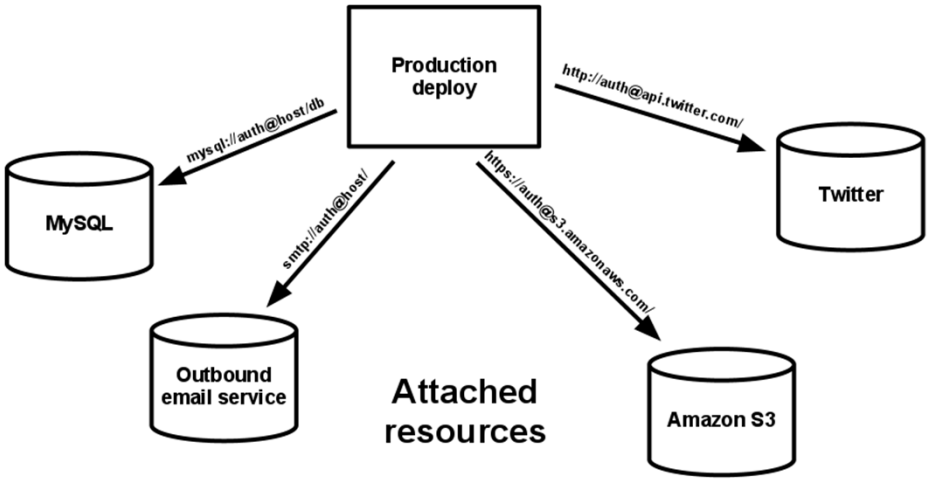
Configuration
Store config in each deploy environment, preferably using environmental variables

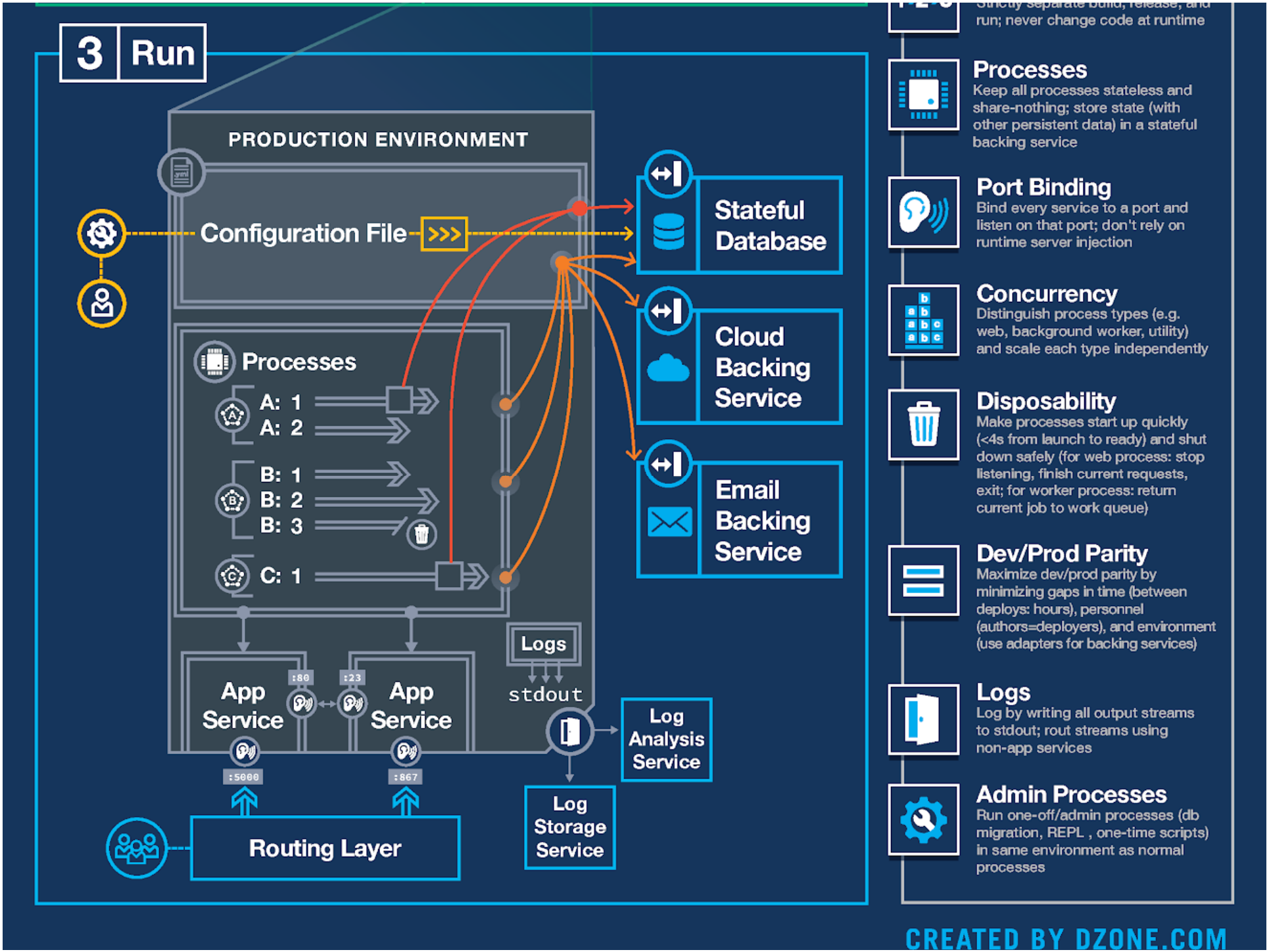
Backing Services
Treat backing services as resources (neutral as to local vs. third-party) located via config


3 Run
PRODUCTION ENVIRONMENT
Configuration File
Processes
A: 1
A: 2
B: 1
B: 2
B: 3
C: 1
App Service
App Service
:80
:23
:5000
:867
Logs
stdout
Routing Layer
Log Analysis Service
Log Storage Service
Stateful Database
Cloud Backing Service
Email Backing Service
run; never change code at runtime
Processes
Keep all processes stateless and share-nothing; store state (with other persistent data) in a stateful backing service
Port Binding
Bind every service to a port and listen on that port; don't rely on runtime server injection
Concurrency
Distinguish process types (e.g. web, background worker, utility) and scale each type independently
Disposability
Make processes start up quickly (<4s from launch to ready) and shut down safely (for web process: stop listening, finish current requests, exit; for worker process: return current job to work queue)
Dev/Prod Parity
Maximize dev/prod parity by minimizing gaps in time (between deploys: hours), personnel (authors=deployers), and environment (use adapters for backing services)
Logs
Log by writing all output streams to stdout; rout streams using non-app services
Admin Processes
Run one-off/admin processes (db migration, REPL , one-time scripts) in same environment as normal processes
CREATED BY DZONE.COM

Twelve Factor App

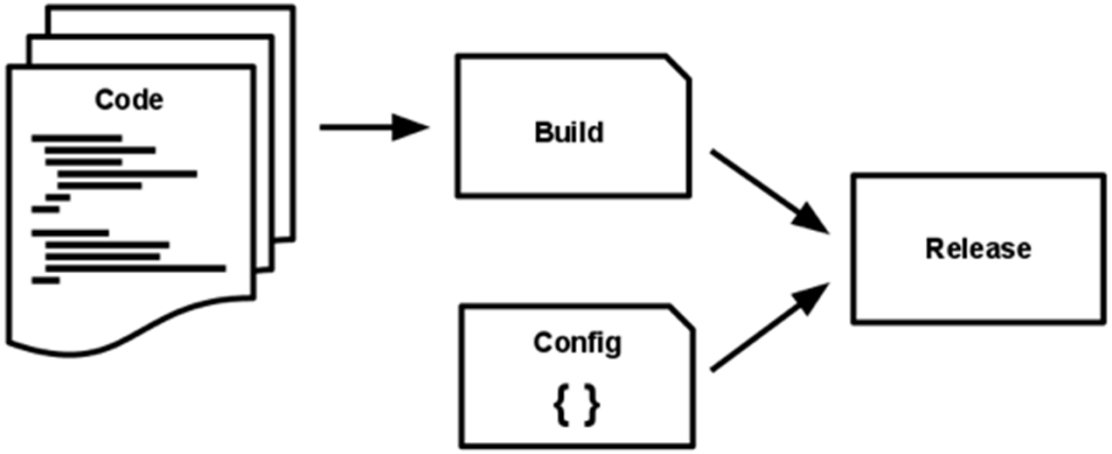
- Backing services
  - Treat backing services as attached resources
  - 保持和後端服務的loose coupling: 隨時可調整抽換
    - 透過 url 設定
    - 例: 在不異動程式情況下，將 MySQL 資料庫換成Amazon RDS




3 Run
PRODUCTION ENVIRONMENT
Configuration File
Processes
A: 1
A: 2
B: 1
B: 2
B: 3
C: 1
App Service
App Service
:80
:23
:5000
:867
Logs
stdout
Routing Layer
Log Analysis Service
Log Storage Service
Stateful Database
Cloud Backing Service
Email Backing Service
run; never change code at runtime
Processes
Keep all processes stateless and share-nothing; store state (with other persistent data) in a stateful backing service
Port Binding
Bind every service to a port and listen on that port; don't rely on runtime server injection
Concurrency
Distinguish process types (e.g. web, background worker, utility) and scale each type independently
Disposability
Make processes start up quickly (<4s from launch to ready) and shut down safely (for web process: stop listening, finish current requests, exit; for worker process: return current job to work queue)
Dev/Prod Parity
Maximize dev/prod parity by minimizing gaps in time (between deploys: hours), personnel (authors=deployers), and environment (use adapters for backing services)
Logs
Log by writing all output streams to stdout; rout streams using non-app services
Admin Processes
Run one-off/admin processes (db migration, REPL , one-time scripts) in same environment as normal processes
CREATED BY DZONE.COM

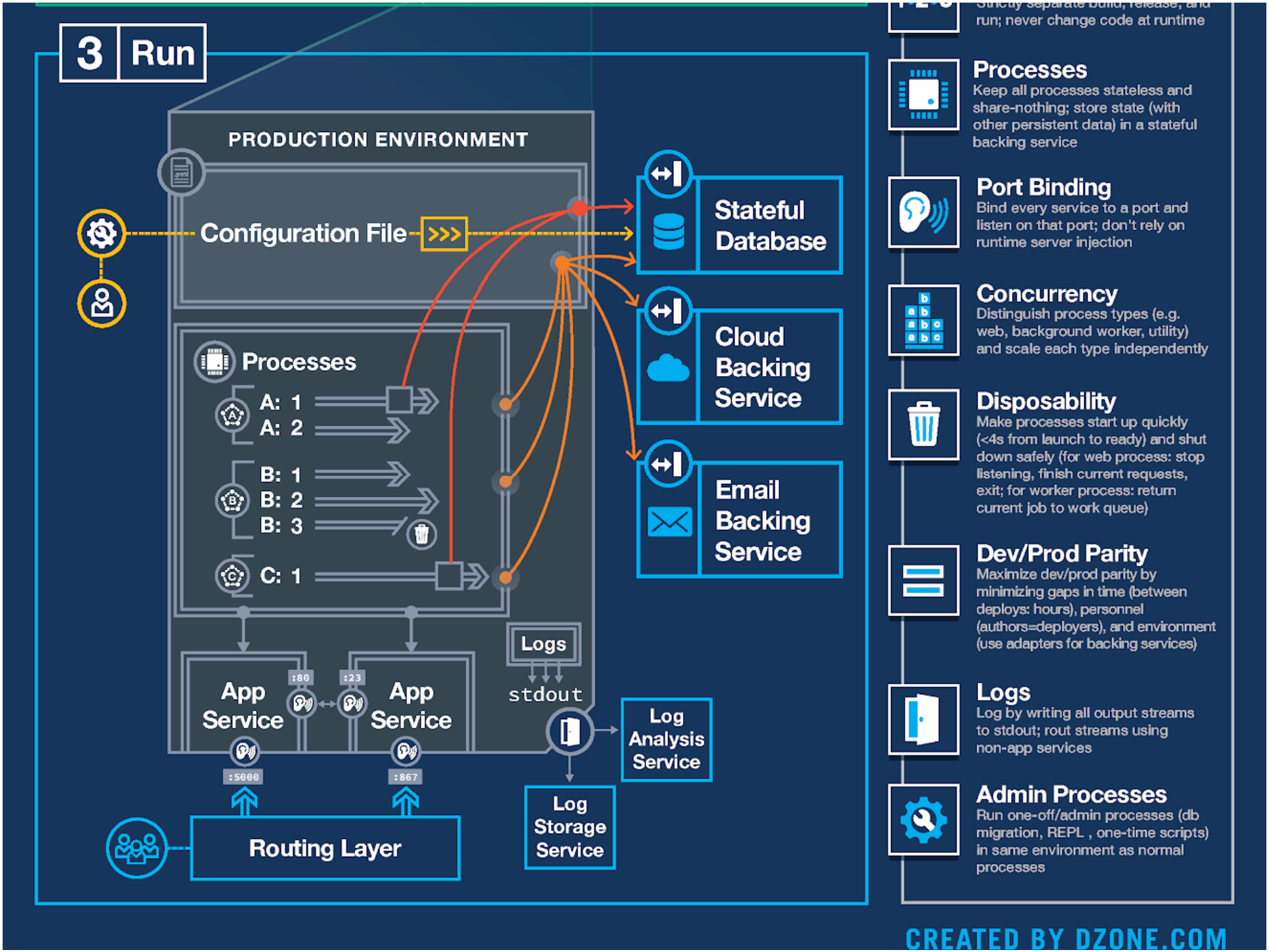
Twelve Factor App

- Build, release, run
  - Strictly separate build and run stages
  - 不要直接修改release、運行中的程式
    - 要透過build→release→run才行!
  - 每次release都要對應到唯一ID，Git hash



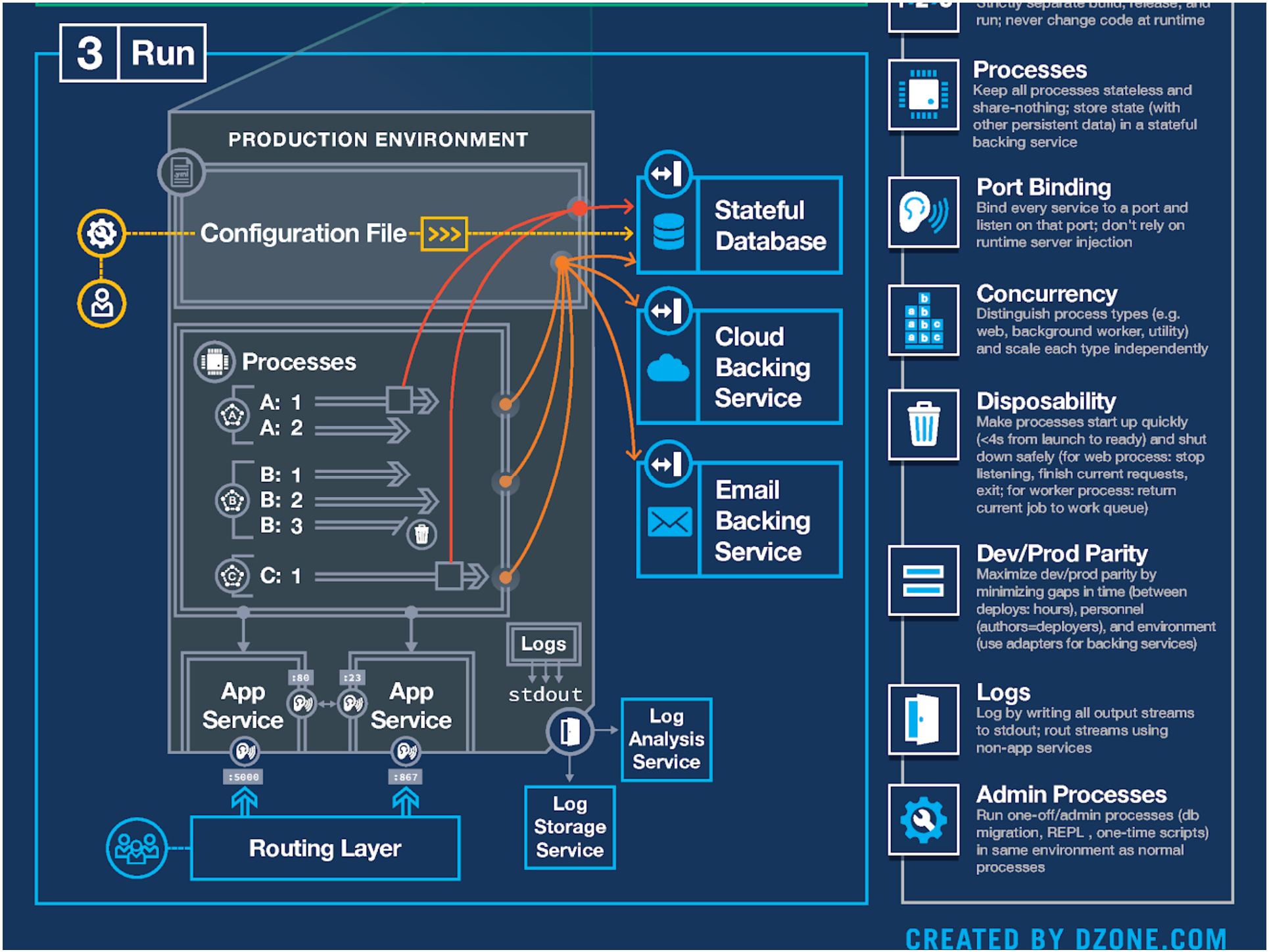
Twelve Factor App

- Processes
  - Execute the app as one or more stateless processes
  - 需要長期保存的資料可以放在 Backing Services


3 Run
PRODUCTION ENVIRONMENT
Configuration File
Processes
A: 1
A: 2
B: 1
B: 2
B: 3
C: 1
App Service
App Service
:80
:23
:5000
:867
Logs
stdout
Routing Layer
Log Analysis Service
Log Storage Service
Stateful Database
Cloud Backing Service
Email Backing Service
run; never change code at runtime
Processes
Keep all processes stateless and share-nothing; store state (with other persistent data) in a stateful backing service
Port Binding
Bind every service to a port and listen on that port; don't rely on runtime server injection
Concurrency
Distinguish process types (e.g. web, background worker, utility) and scale each type independently
Disposability
Make processes start up quickly (<4s from launch to ready) and shut down safely (for web process: stop listening, finish current requests, exit; for worker process: return current job to work queue)
Dev/Prod Parity
Maximize dev/prod parity by minimizing gaps in time (between deploys: hours), personnel (authors=deployers), and environment (use adapters for backing services)
Logs
Log by writing all output streams to stdout; rout streams using non-app services
Admin Processes
Run one-off/admin processes (db migration, REPL , one-time scripts) in same environment as normal processes
CREATED BY DZONE.COM

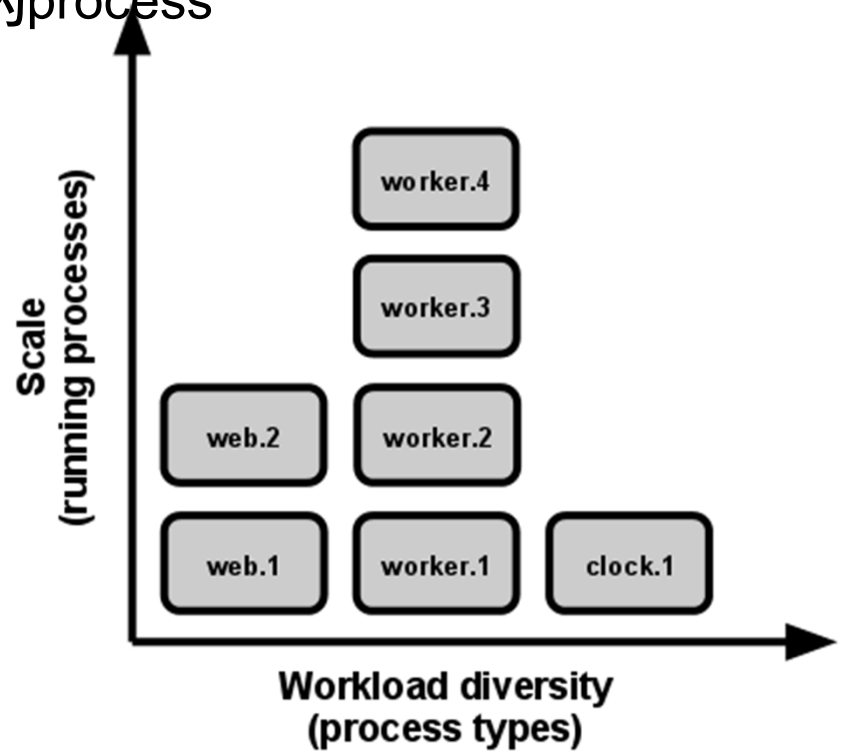
Twelve Factor App

- Port binding
  - Export services via port binding
  - 不要讓網路服務佔據、固定的port，保持彈性
    - docker run -d -p 8080:80 httpd


3 Run
PRODUCTION ENVIRONMENT
Configuration File
Processes
A: 1
A: 2
B: 1
B: 2
B: 3
C: 1
App Service
App Service
:80
:23
:5000
:867
Logs
stdout
Routing Layer
Log Analysis Service
Log Storage Service
Stateful Database
Cloud Backing Service
Email Backing Service
run; never change code at runtime
Processes
Keep all processes stateless and share-nothing; store state (with other persistent data) in a stateful backing service
Port Binding
Bind every service to a port and listen on that port; don't rely on runtime server injection
Concurrency
Distinguish process types (e.g. web, background worker, utility) and scale each type independently
Disposability
Make processes start up quickly (<4s from launch to ready) and shut down safely (for web process: stop listening, finish current requests, exit; for worker process: return current job to work queue)
Dev/Prod Parity
Maximize dev/prod parity by minimizing gaps in time (between deploys: hours), personnel (authors=deployers), and environment (use adapters for backing services)
Logs
Log by writing all output streams to stdout; rout streams using non-app services
Admin Processes
Run one-off/admin processes (db migration, REPL , one-time scripts) in same environment as normal processes
CREATED BY DZONE.COM

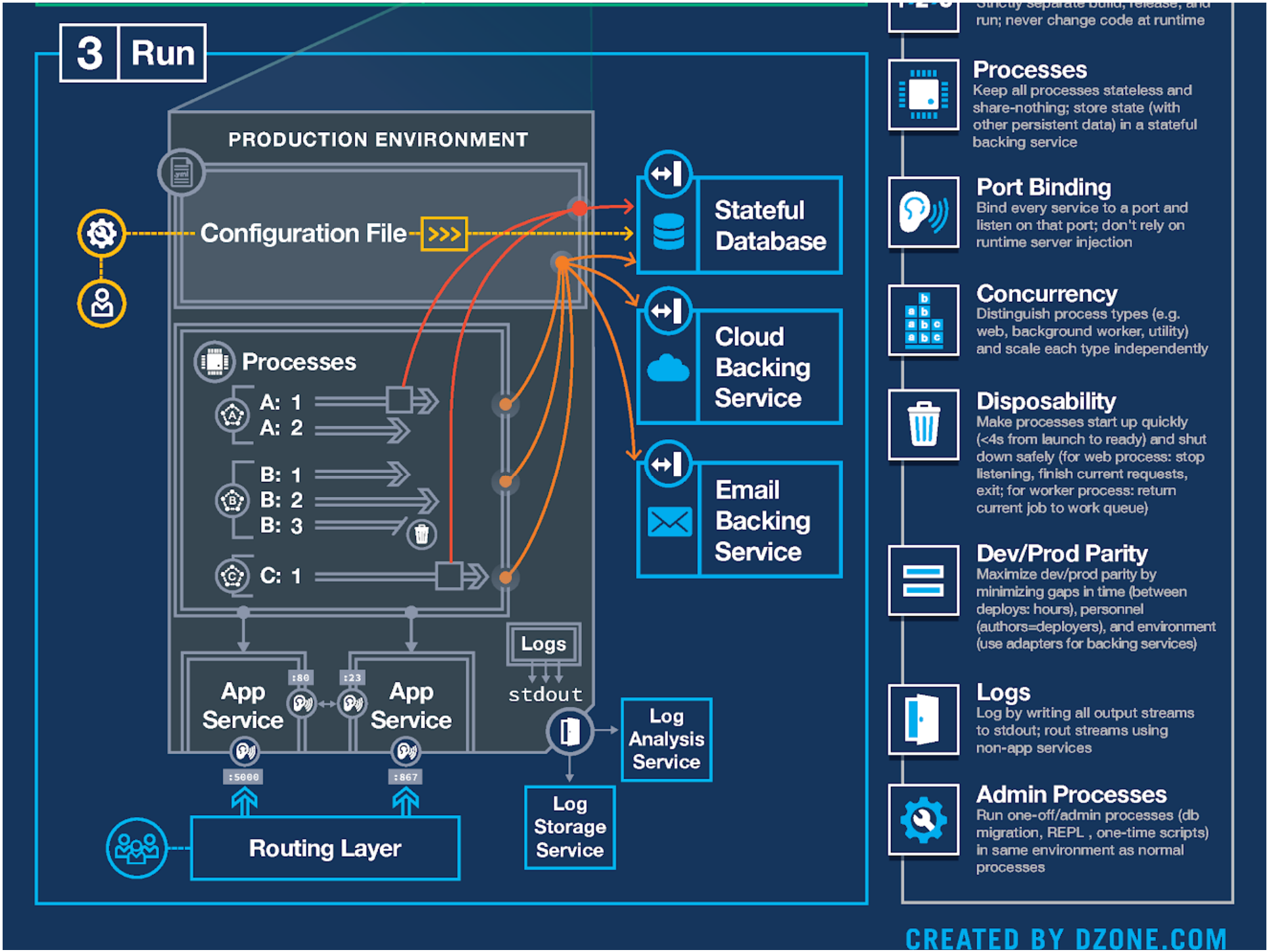
Twelve Factor App

- Concurrency
  - Scale out via the process
  - 要做到並行時，使用多個並行的process
    - 而不是用thread



Twelve Factor App

- Disposability
  - Maximize robustness with fast startup and graceful shutdown
  - Process應該要可以隨時開/停
  - 如果dependency少，那麼重開就比較簡單

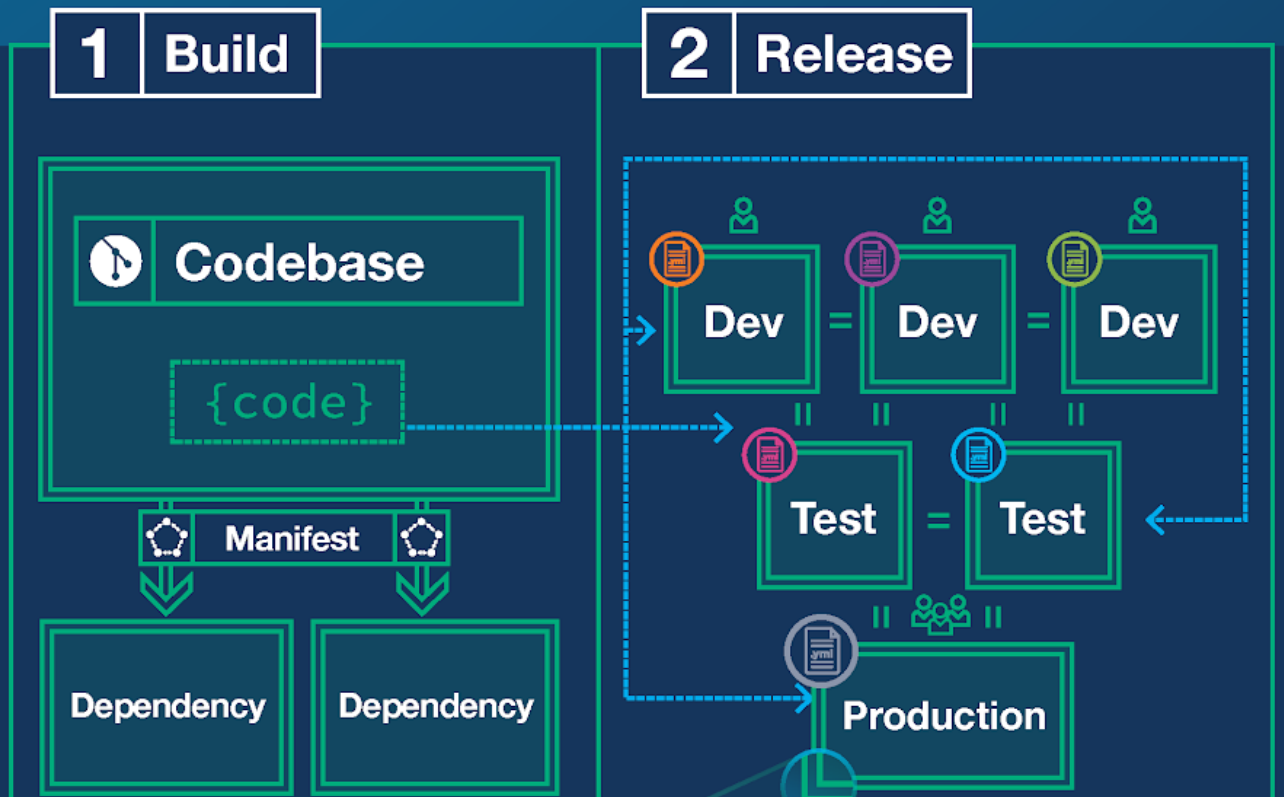

3 Run
PRODUCTION ENVIRONMENT
Configuration File
Processes
A: 1
A: 2
B: 1
B: 2
B: 3
C: 1
App Service
App Service
:80
:23
:5000
:867
Logs
stdout
Routing Layer
Log Analysis Service
Log Storage Service
Stateful Database
Cloud Backing Service
Email Backing Service
run; never change code at runtime
Processes
Keep all processes stateless and share-nothing; store state (with other persistent data) in a stateful backing service
Port Binding
Bind every service to a port and listen on that port; don't rely on runtime server injection
Concurrency
Distinguish process types (e.g. web, background worker, utility) and scale each type independently
Disposability
Make processes start up quickly (<4s from launch to ready) and shut down safely (for web process: stop listening, finish current requests, exit; for worker process: return current job to work queue)
Dev/Prod Parity
Maximize dev/prod parity by minimizing gaps in time (between deploys: hours), personnel (authors=deployers), and environment (use adapters for backing services)
Logs
Log by writing all output streams to stdout; rout streams using non-app services
Admin Processes
Run one-off/admin processes (db migration, REPL , one-time scripts) in same environment as normal processes
CREATED BY DZONE.COM

Twelve Factor App

- Dev/prod parity
  - Keep development, staging, and production as similar as possible
  - 開發、staging和正式上線的環境保持相同

The 12-Factor App

Modern web applications run in heterogeneous environments, scale elastically, update frequently, and depend on independently deployed backing services. Modern application architectures and development practices must be designed accordingly. The PaaS-masters at Heroku summarized lessons learned from building hundreds of cloud-native applications into the twelve factors visualized below.



the 12 factors

Codebase
One codebase, many deploys, strict version control

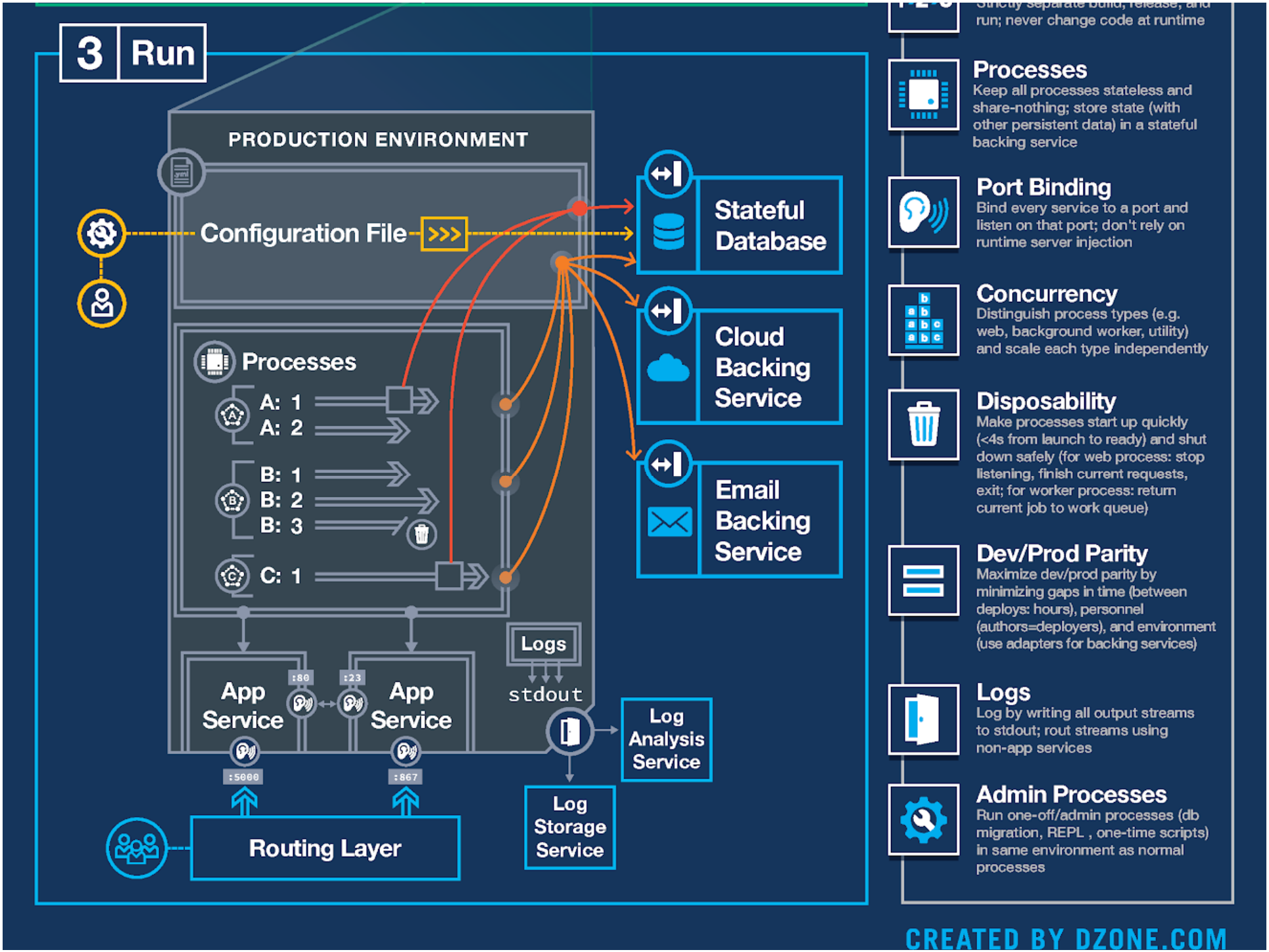
Dependencies
Explicitly declare and isolate dependencies

Configuration
Store config in each deploy environment, preferably using environmental variables

Backing Services
Treat backing services as resources (neutral as to local vs. third-party) located via config

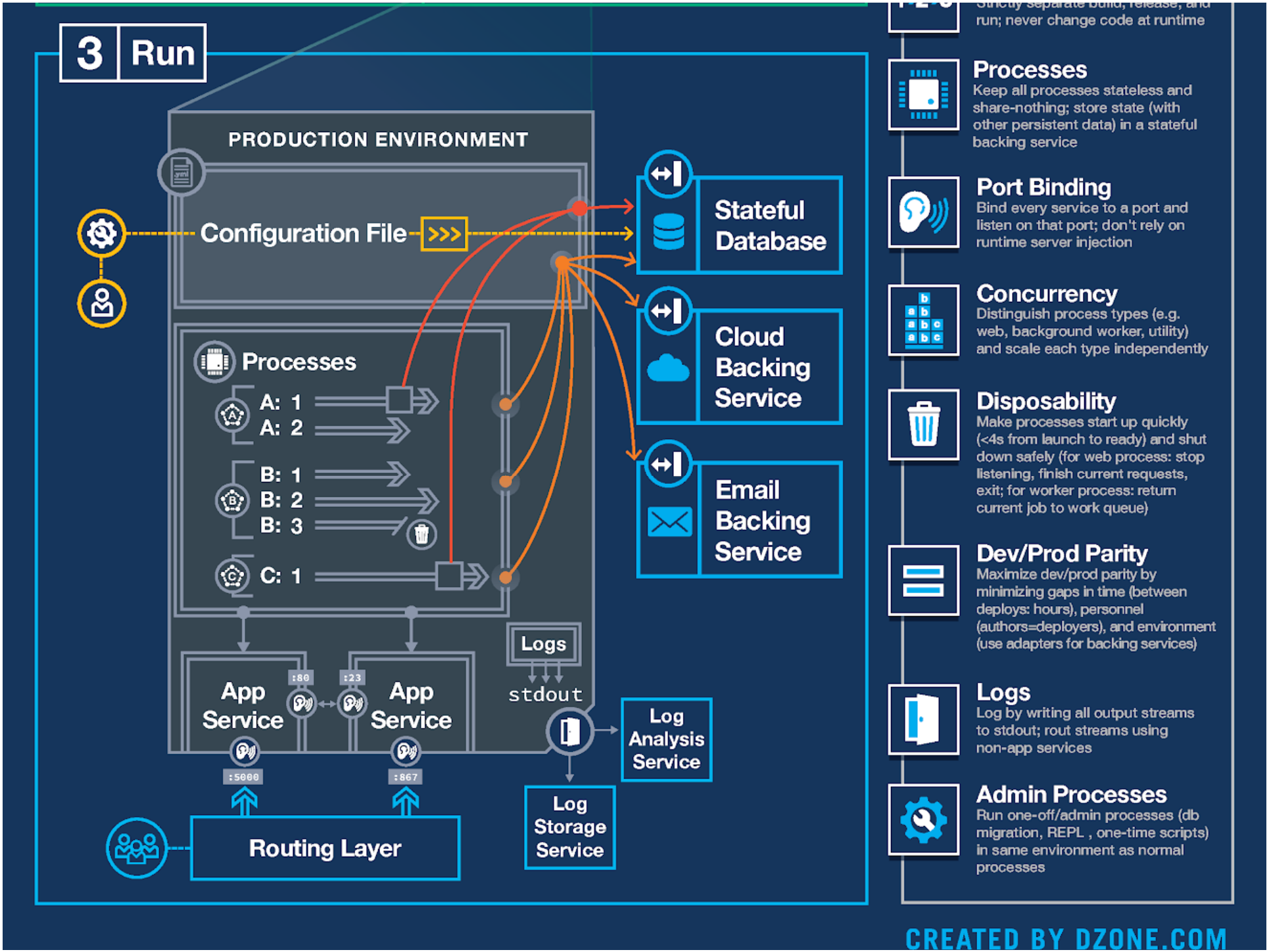
Twelve Factor App

- Logs
  - Treat logs as event streams
  - 應用程式記錄檔寫到STDOUT，方便由工具統一收集處理
    - 不要在container image中保留log，而是要把 log 作為 event stream 輸出
    - 例:利用 Fluentd 等工具擷取後，透過 Elasticsearch 分析及 Kinbana 呈現結果


3 Run
PRODUCTION ENVIRONMENT
Configuration File
Processes
A: 1
A: 2
B: 1
B: 2
B: 3
C: 1
App Service
App Service
:80
:23
:5000
:867
Logs
stdout
Routing Layer
Log Analysis Service
Log Storage Service
Stateful Database
Cloud Backing Service
Email Backing Service
run; never change code at runtime
Processes
Keep all processes stateless and share-nothing; store state (with other persistent data) in a stateful backing service
Port Binding
Bind every service to a port and listen on that port; don't rely on runtime server injection
Concurrency
Distinguish process types (e.g. web, background worker, utility) and scale each type independently
Disposability
Make processes start up quickly (<4s from launch to ready) and shut down safely (for web process: stop listening, finish current requests, exit; for worker process: return current job to work queue)
Dev/Prod Parity
Maximize dev/prod parity by minimizing gaps in time (between deploys: hours), personnel (authors=deployers), and environment (use adapters for backing services)
Logs
Log by writing all output streams to stdout; rout streams using non-app services
Admin Processes
Run one-off/admin processes (db migration, REPL , one-time scripts) in same environment as normal processes
CREATED BY DZONE.COM

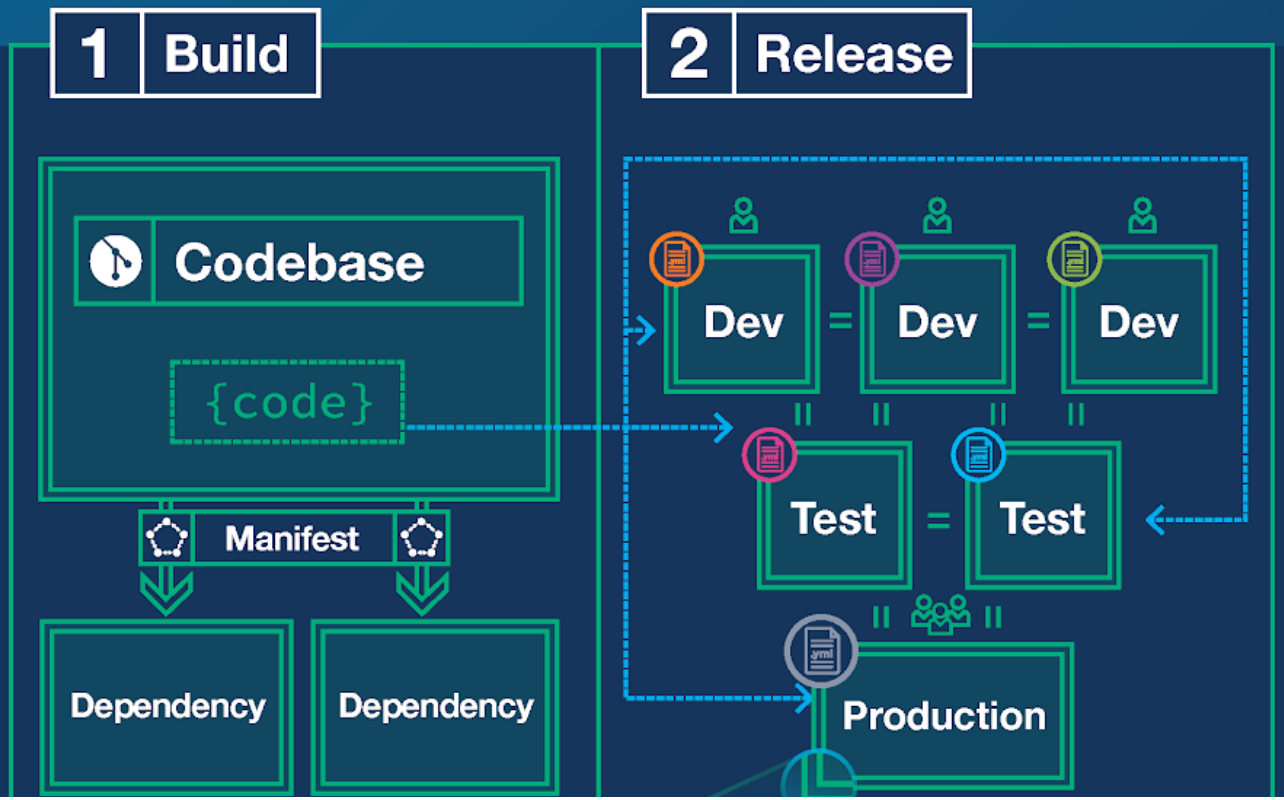
Twelve Factor App

- Admin processes
  - Run admin/management tasks as one-off processes
  - 後台管理工作、維護任務作為一次性的 process 執行
    - 例: 轉移資料、清理環境
  - 一次性管理程式應該和正常的程式使用同樣的環境
    - 例如: 全部使用docker來運行這類一次性scripts
    - docker exec -it web php artisan migrate
  - 一次性管理程式也要像原始碼一般進行版本控制
    - Admin code must ship with application code to avoid synchronization issues


3 Run
PRODUCTION ENVIRONMENT
Configuration File
Processes
A: 1
A: 2
B: 1
B: 2
B: 3
C: 1
App Service
App Service
:80
:23
:5000
:867
Logs
stdout
Routing Layer
Log Analysis Service
Log Storage Service
Stateful Database
Cloud Backing Service
Email Backing Service
run; never change code at runtime
Processes
Keep all processes stateless and share-nothing; store state (with other persistent data) in a stateful backing service
Port Binding
Bind every service to a port and listen on that port; don't rely on runtime server injection
Concurrency
Distinguish process types (e.g. web, background worker, utility) and scale each type independently
Disposability
Make processes start up quickly (<4s from launch to ready) and shut down safely (for web process: stop listening, finish current requests, exit; for worker process: return current job to work queue)
Dev/Prod Parity
Maximize dev/prod parity by minimizing gaps in time (between deploys: hours), personnel (authors=deployers), and environment (use adapters for backing services)
Logs
Log by writing all output streams to stdout; rout streams using non-app services
Admin Processes
Run one-off/admin processes (db migration, REPL , one-time scripts) in same environment as normal processes
CREATED BY DZONE.COM

The 12-Factor App

Modern web applications run in heterogeneous environments, scale elastically, update frequently, and depend on independently deployed backing services. Modern application architectures and development practices must be designed accordingly. The PaaS-masters at Heroku summarized lessons learned from building hundreds of cloud-native applications into the twelve factors visualized below.



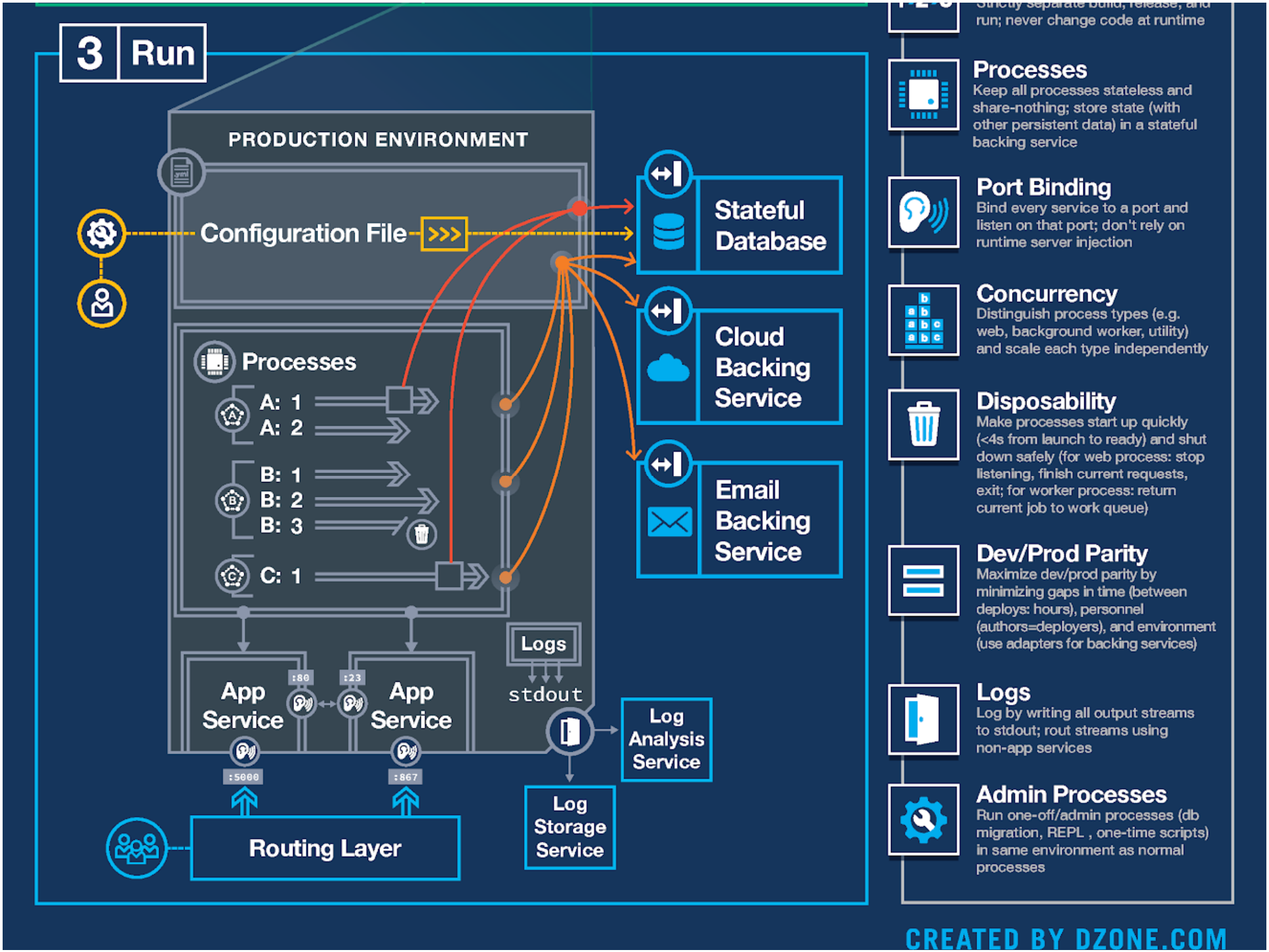
the 12 factors

Codebase
One codebase, many deploys, strict version control

Dependencies
Explicitly declare and isolate dependencies

Configuration
Store config in each deploy environment, preferably using environmental variables

Backing Services
Treat backing services as resources (neutral as to local vs. third-party) located via config


3 Run
PRODUCTION ENVIRONMENT
Configuration File
Processes
A: 1
A: 2
B: 1
B: 2
B: 3
C: 1
App Service
App Service
:80
:23
:5000
:867
Logs
stdout
Routing Layer
Log Analysis Service
Log Storage Service
Stateful Database
Cloud Backing Service
Email Backing Service
run; never change code at runtime
Processes
Keep all processes stateless and share-nothing; store state (with other persistent data) in a stateful backing service
Port Binding
Bind every service to a port and listen on that port; don't rely on runtime server injection
Concurrency
Distinguish process types (e.g. web, background worker, utility) and scale each type independently
Disposability
Make processes start up quickly (<4s from launch to ready) and shut down safely (for web process: stop listening, finish current requests, exit; for worker process: return current job to work queue)
Dev/Prod Parity
Maximize dev/prod parity by minimizing gaps in time (between deploys: hours), personnel (authors=deployers), and environment (use adapters for backing services)
Logs
Log by writing all output streams to stdout; rout streams using non-app services
Admin Processes
Run one-off/admin processes (db migration, REPL , one-time scripts) in same environment as normal processes
CREATED BY DZONE.COM

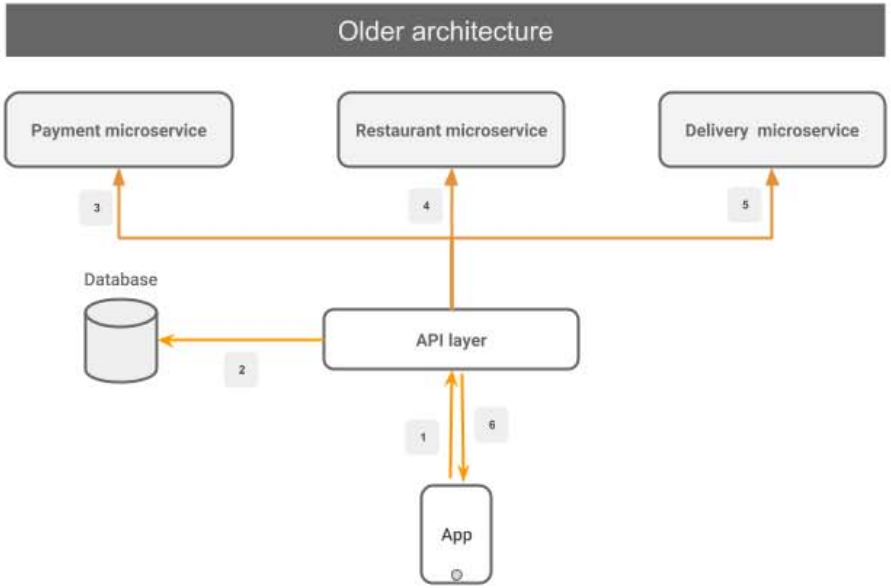
https://github.com/hasura/3factor-example/

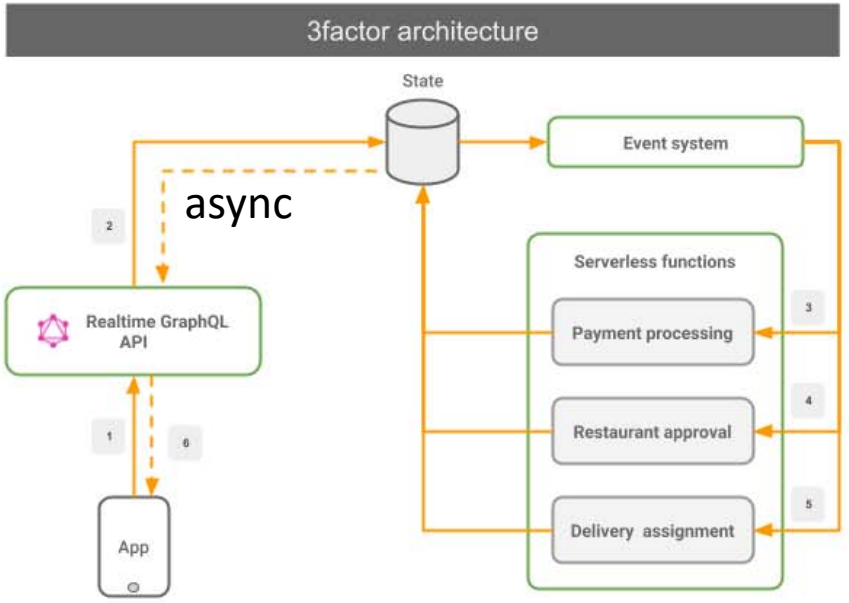
https://3factor.app/

3-factor App

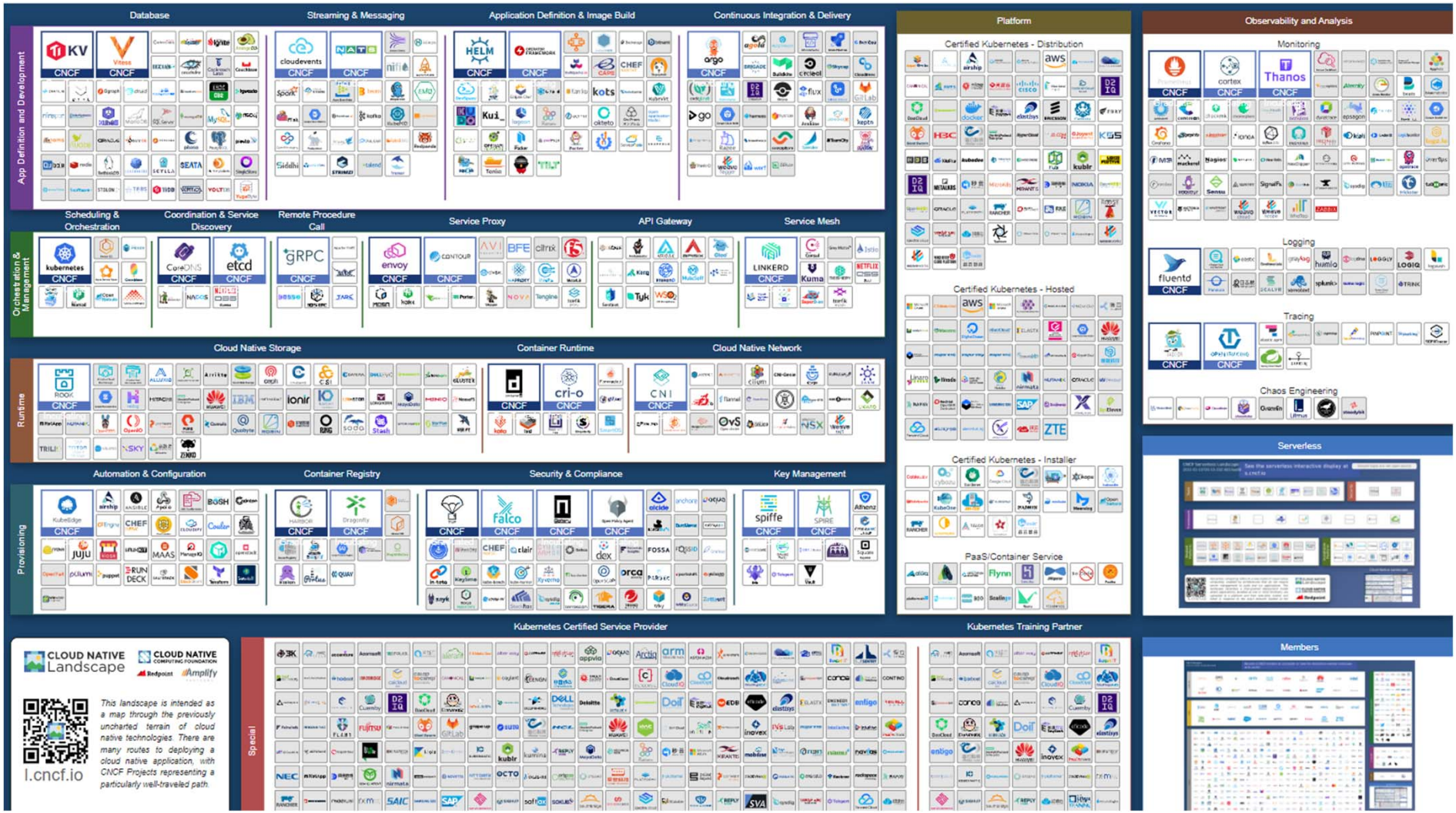
- Realtime GraphQL
- Reliable eventing
- Async serverless

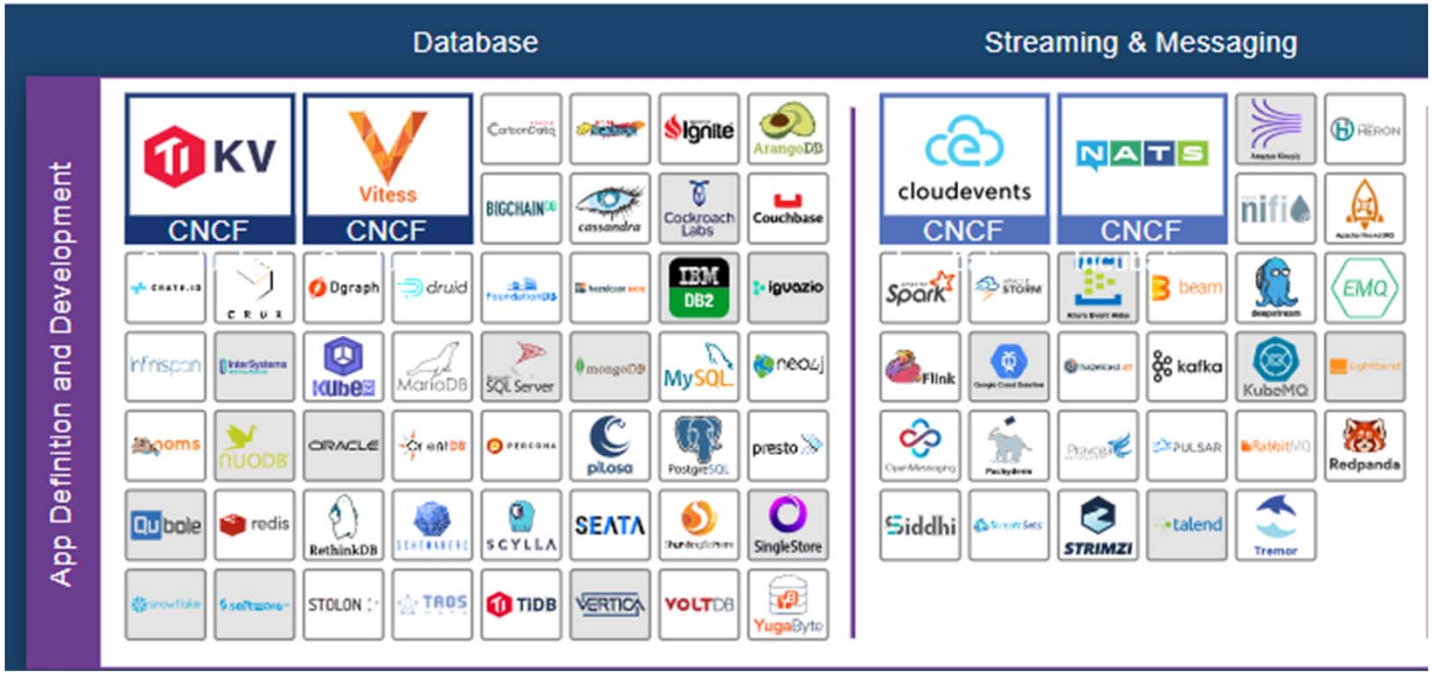
The 3factor app architecture is an implementation of the CQRS pattern in many ways.

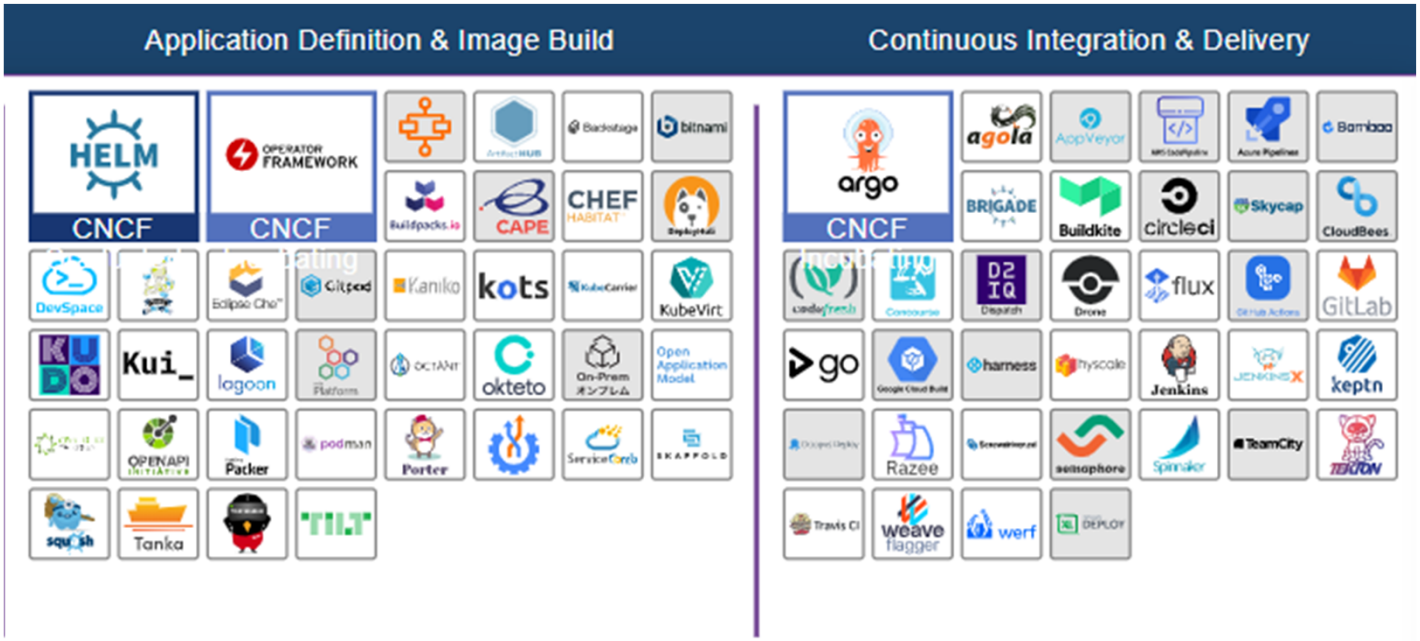


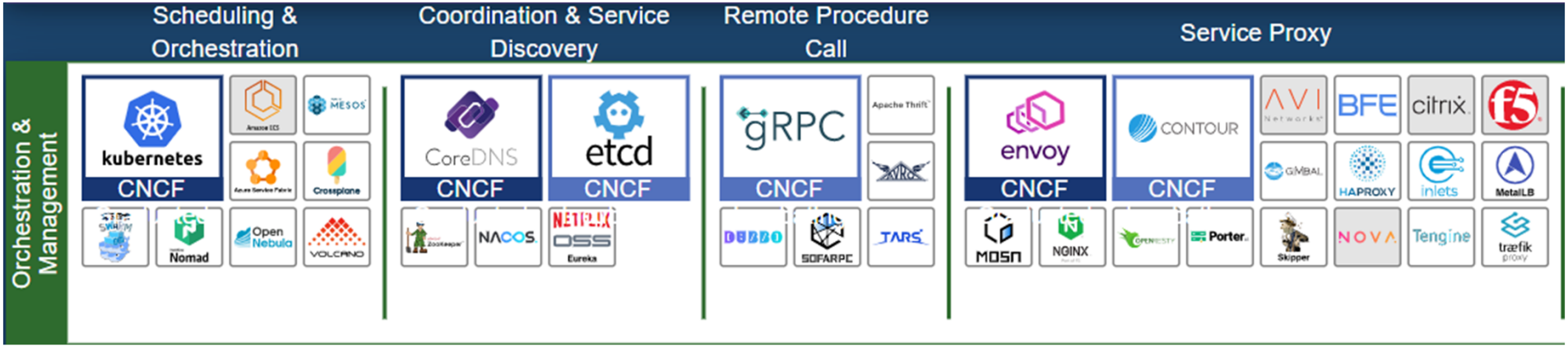


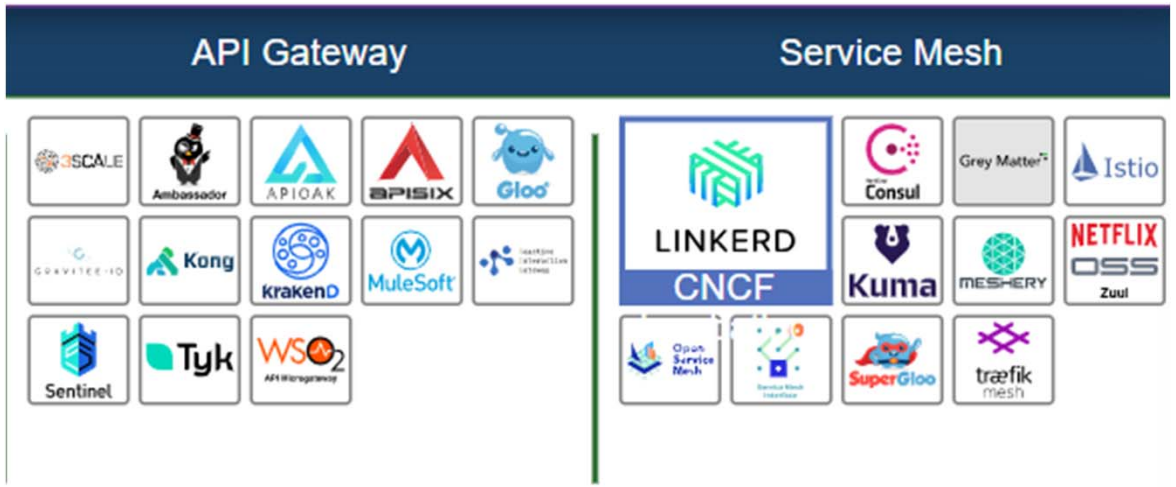
CNCF Cloud Native Landscape

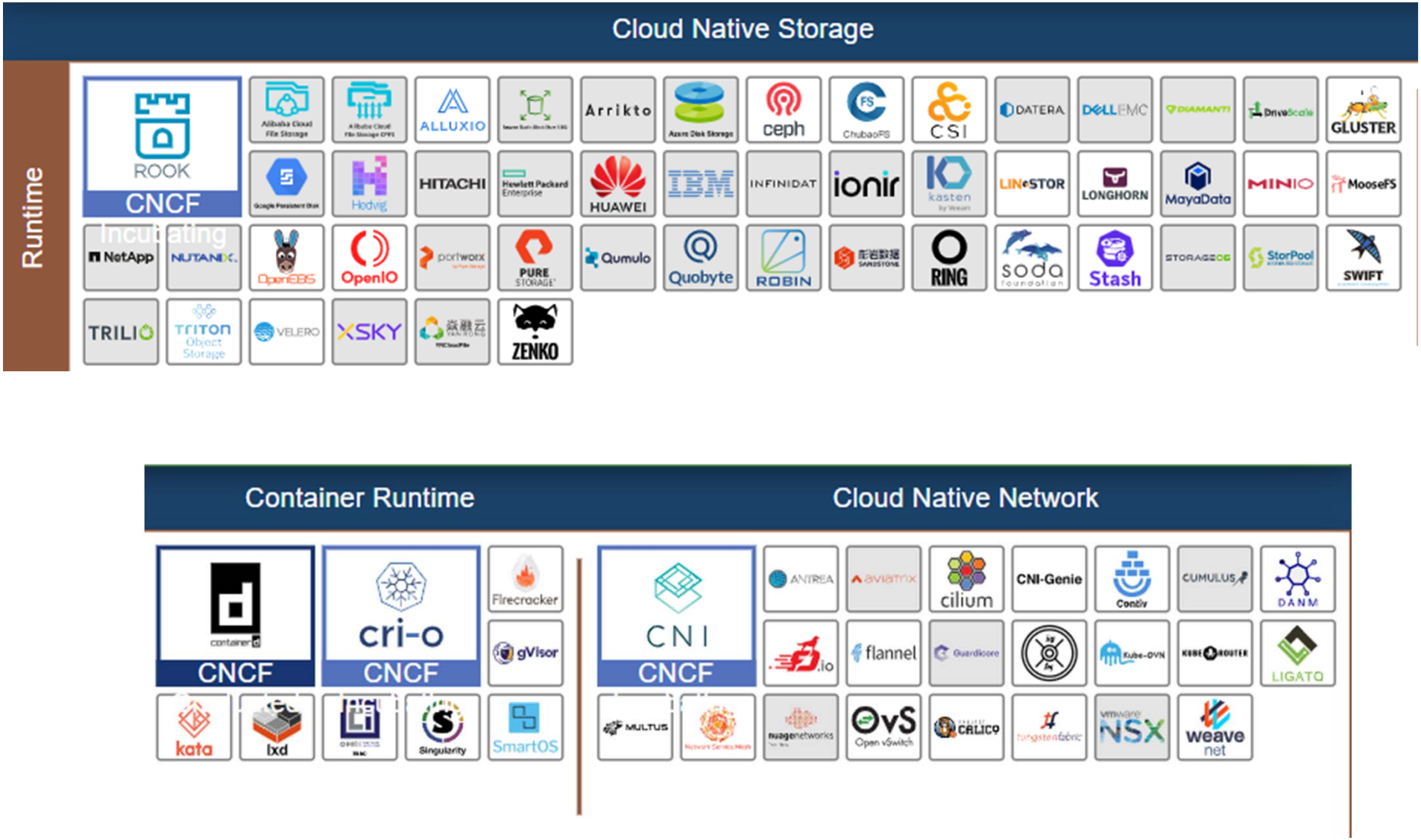


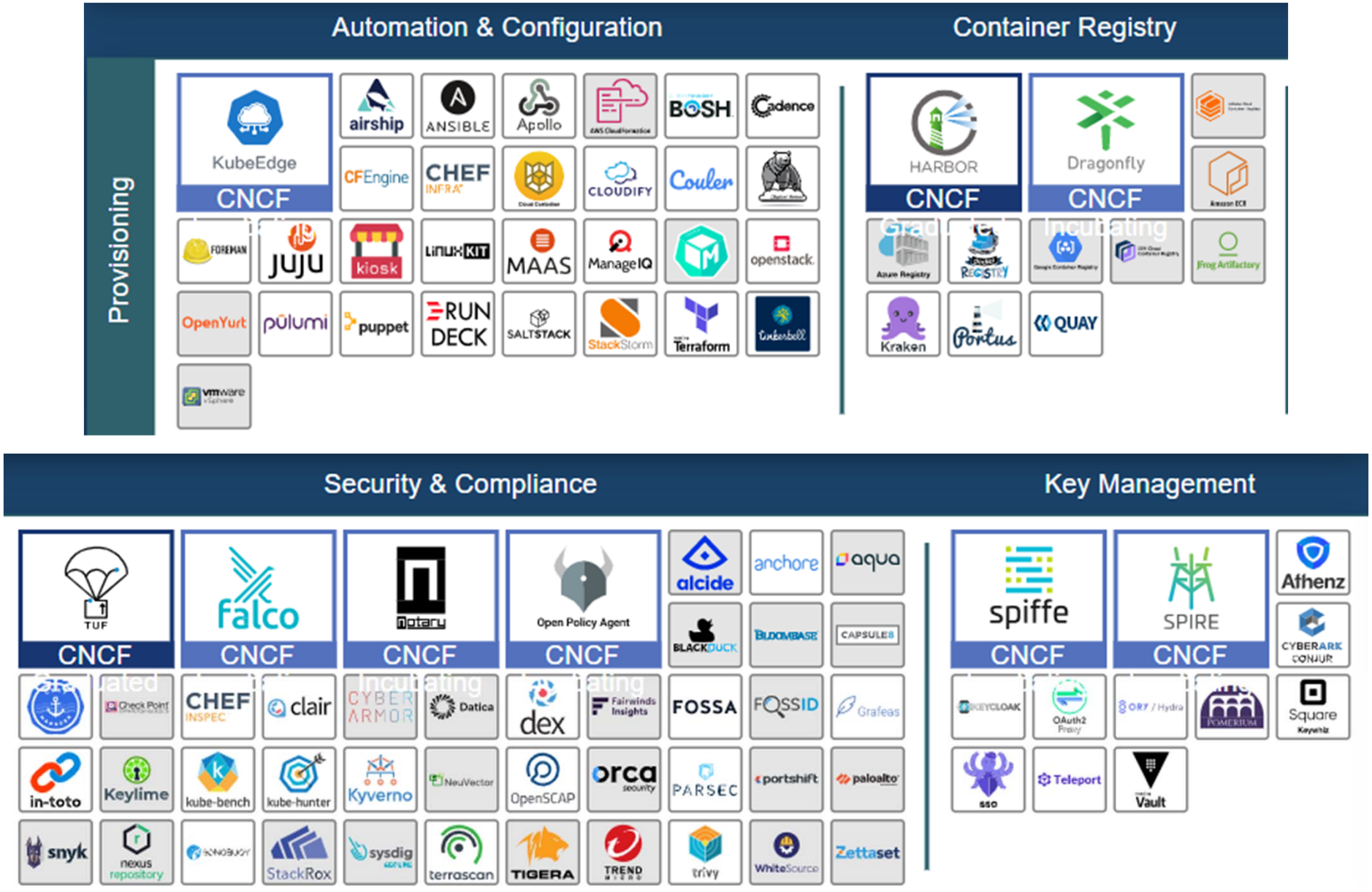

Database
Streaming & Messaging
App Definition and Development
KV
CNCF
Vitess
CNCF
Ignite
ArangoDB
BIGCHAINDB
cassandra
Cockroach Labs
Couchbase
Dgraph
druid
IBM DB2
iguazio
Infinispan
KubeDB
MariaDB
SQL Server
MySQL
neo4j
NUODB
ORACLE
PERCONA
pitosa
PostgreSQL
presto
Qubole
redis
RethinkDB
SCYLLA
SEATA
SingleStore
snowflake
STOLON
TIDB
VERTICA
VOLTDB
YugaByte
cloudevents
CNCF
NATS
CNCF
HERON
nifi
Spark
STORM
beam
EMQ
Flink
kafka
KubeMQ
PULSAR
RabbitMQ
Redpanda
Siddhi
STRIMZI
talend
Tremor

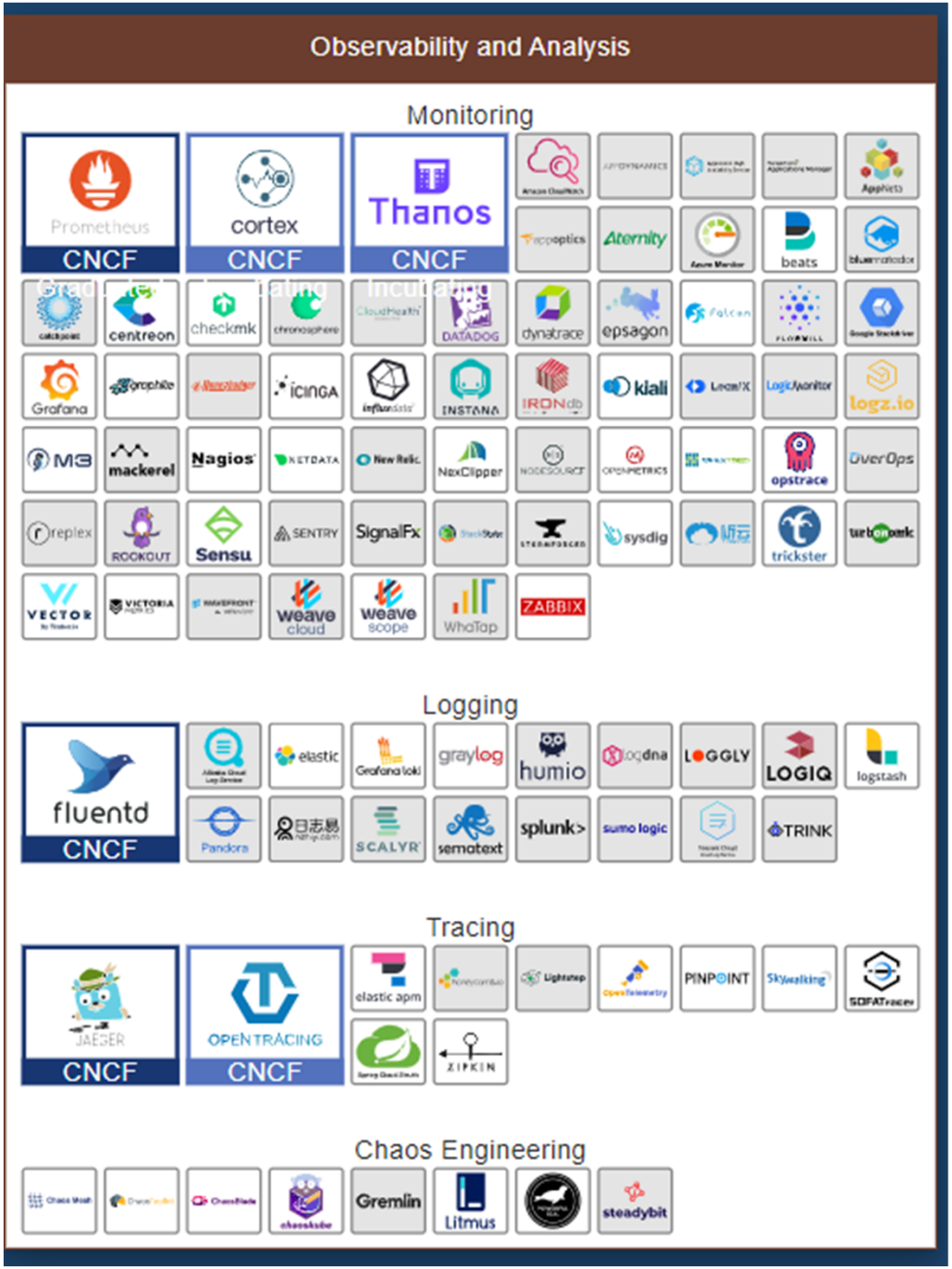

Application Definition & Image Build
Continuous Integration & Delivery
HELM
CNCF
OPERATOR FRAMEWORK
CNCF
bitnami
Buildpacks.io
CAPE
CHEF HABITAT
DevSpace
Eclipse Che
Gitkube
Kaniko
kots
KubeVirt
KUDO
Kui
logoon
OCTANT
okteto
Open Application Model
OPENAPI INITIATIVE
Packer
podman
Porter
ServiceCatalog
SKAFFOLD
squash
Tanka
TILT
argo
CNCF
agola
AppVeyor
Bamboo
BRIGADE
Buildkite
circleci
Skycap
CloudBees
D2IQ Dispatch
Drone
flux
GitLab
go
harness
Jenkins
JENKINS X
keptn
Razee
semaphore
Spinnaker
TeamCity
Travis CI
weave flagger
werf


Orchestration & Management
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
kubernetes
CNCF
Amazon ECS
MESOS
Azure Service Fabric
Crossplane
Nomad
Open Nebula
VOLCANO
CoreDNS
CNCF
etcd
CNCF
NACOS
NETFLIX OSS
Eureka
gRPC
CNCF
Apache Thrift
DUBBO
SOFARPC
JARS
envoy
CNCF
CONTOUR
CNCF
AVI Networks
BFE
citrix
f5
GIMBAL
HAPROXY
inlets
MetalLB
MOSN
NGINX
OpenResty
Porter
Skipper
NOVA
Tengine
træfik proxy


API Gateway
Service Mesh
3SCALE
Ambassador
APIOAK
APISIX
Gloo
GRAVITEE.IO
Kong
KrakenD
MuleSoft
Sentinel
Tyk
WSO2
LINKERD
CNCF
Consul
Grey Matter
Istio
Kuma
MESHERY
NETFLIX OSS
Zuul
Open Service Mesh
SuperGloo
træfik mesh


Cloud Native Storage
Runtime
ROOK
CNCF
Incubating
NetApp
NUTANIX
OpenEBS
OpenIO
portworx
PURE STORAGE
Qumulo
Quobyte
ROBIN
SANDSTONE
RING
soda foundation
Stash
StorPool
SWIFT
TRILIO
Triton Object Storage
VELERO
XSKY
ZENKO
ALLUXIO
Arrikto
ceph
ChubaoFS
CSI
DATERA
DELL EMC
DIAMANTI
DriveScale
GLUSTER
Hedvig
HITACHI
Hewlett Packard Enterprise
HUAWEI
IBM
INFINIDAT
ionir
kasten
LINSTOR
LONGHORN
MayaData
MINIO
MooseFS
Container Runtime
Cloud Native Network
CNCF
cri-o
CNCF
Firecracker
gVisor
kata
lxd
Singularity
SmartOS
CNI
CNCF
ANTREA
aviatrix
cilium
CNI-Genie
Contiv
CUMULUS
DANM
flannel
Guardicore
Kube-OVN
KUBE ROUTER
LIGATO
MULTUS
nuagenetworks
OvS
Open vSwitch
CALICO
tungstenfabric
vmware NSX
weave net


Automation & Configuration
Container Registry
Provisioning
KubeEdge
CNCF
airship
ANSIBLE
Apollo
BOSH
Cadence
CFEngine
CHEF
INFRA
CLOUDIFY
Couler
FOREMAN
JUJU
kiosk
LINUXKIT
MAAS
ManageIQ
openstack
OpenYurt
pulumi
puppet
RUNDECK
SALTSTACK
StackStorm
Terraform
tinkerbell
vmware
HARBOR
CNCF
Dragonfly
CNCF
Azure Registry
REGISTRY
JFrog Artifactory
Kraken
Portus
QUAY
Security & Compliance
Key Management
TUF
CNCF
falco
CNCF
notary
CNCF
Open Policy Agent
CNCF
alcide
anchore
aqua
BLACKDUCK
BLOOMBASE
CAPSULE8
Check Point
CHEF
INSPEC
clair
CYBER ARMOR
Datica
dex
Fairwinds Insights
FOSSA
FOSSID
Grafeas
in-toto
Keylime
kube-bench
kube-hunter
Kyverno
NeuVector
OpenSCAP
orca security
PARSEC
portshift
paloalto
snyk
nexus repository
SONOBUOY
StackRox
sysdig
terrascan
TIGERA
TREND MICRO
trivy
WhiteSource
Zettaset
spiffe
CNCF
SPIRE
CNCF
Athenz
CYBERARK CONJUR
KEYCLOAK
OAuth2 Proxy
ORY / Hydra
POMERIUM
Square
sso
Teleport
Vault


Observability and Analysis
Monitoring
Prometheus
CNCF
cortex
CNCF
Thanos
CNCF
Graduated
Incubating
Incubating
Grafana
Nagios
Sensu
SENTRY
SignalFx
ZABBIX
Logging
fluentd
CNCF
elastic
Grafana loki
graylog
humio
LOGGLY
LOGIQ
logstash
Pandora
splunk>
sumo logic
sematext
TRINK
Tracing
JAEGER
CNCF
OPENTRACING
CNCF
elastic apm
ZIPKIN
PINPOINT
SkyWalking
Chaos Engineering
Gremlin
Litmus
steadybit

Q & A