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

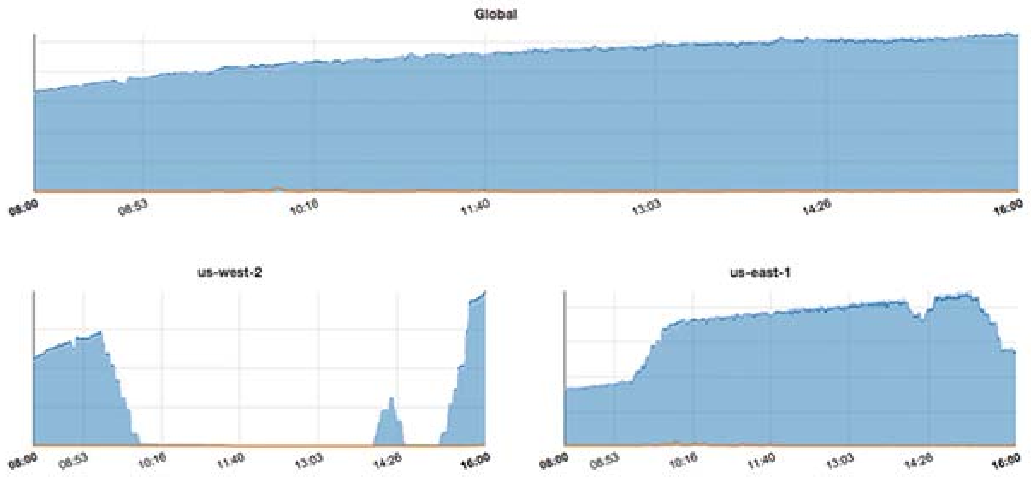
National Chengchi University

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



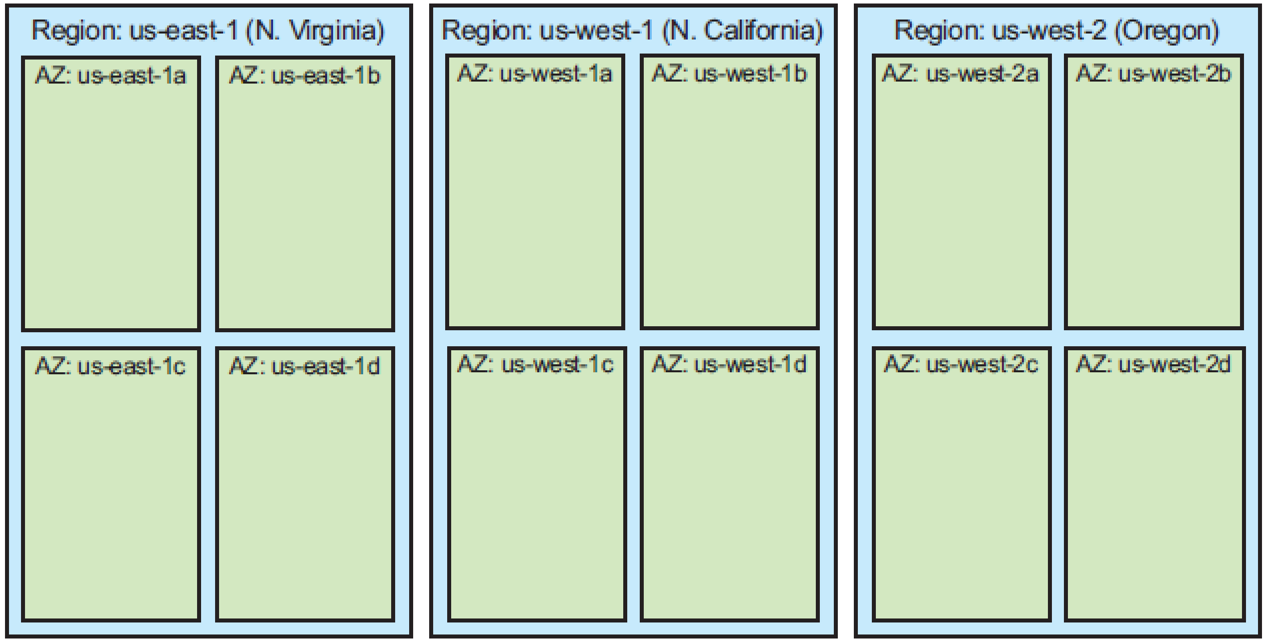
Chaos Kong exercise in progress

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

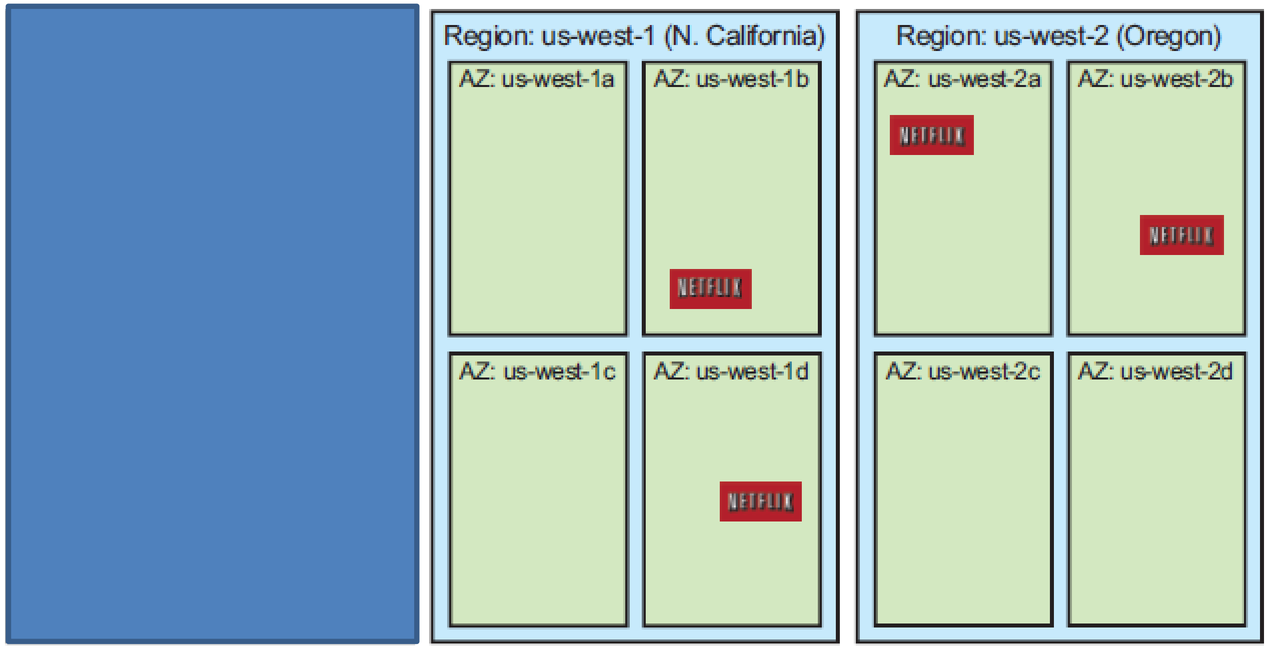
Chaos Monkey. The service pseudo-randomly plucks a server from our production deployment on AWS and kills it.

Case: AWS outage in 2015

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



Case: AWS outage in 2015

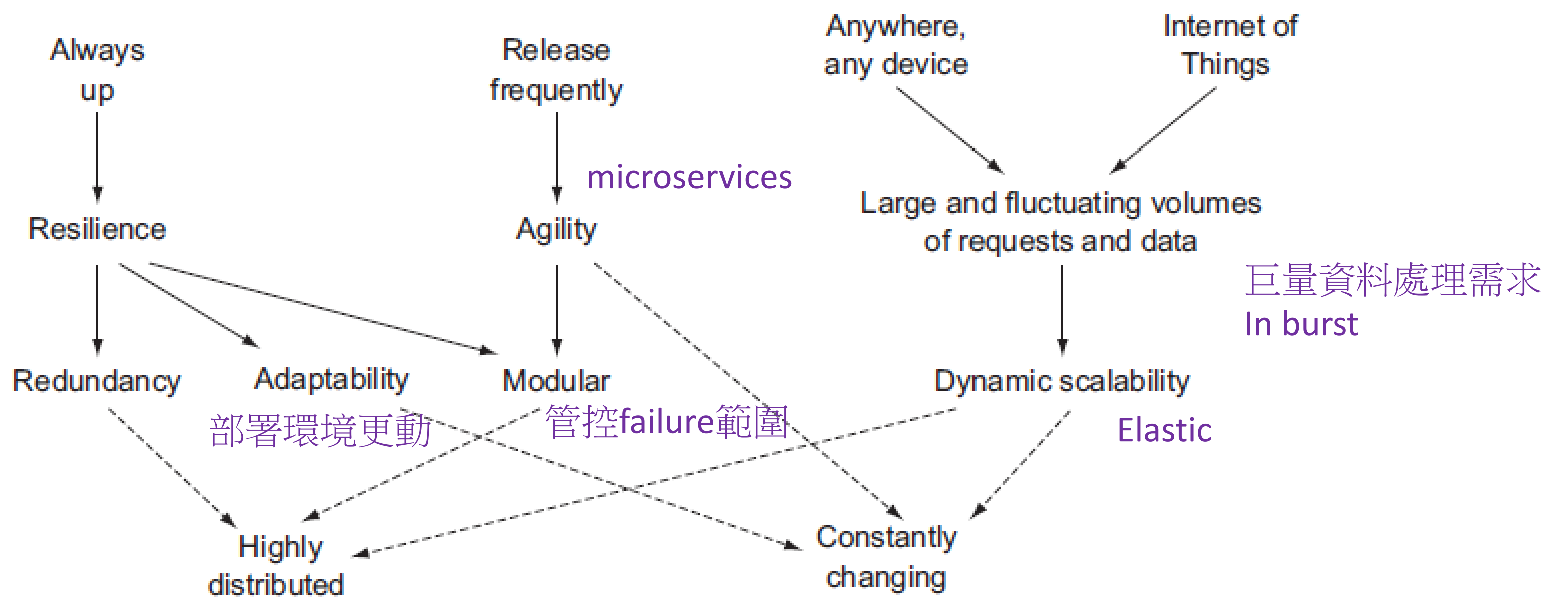


Case: AWS outage in 2015

- 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

Cloud Native Features

- Highly distributed and constantly changing



Cornelia Davis : Sr. Director of Technology at Pivotal Software

Definition

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

Cloud Native: 各種說法

- 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: 各種説法

- 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

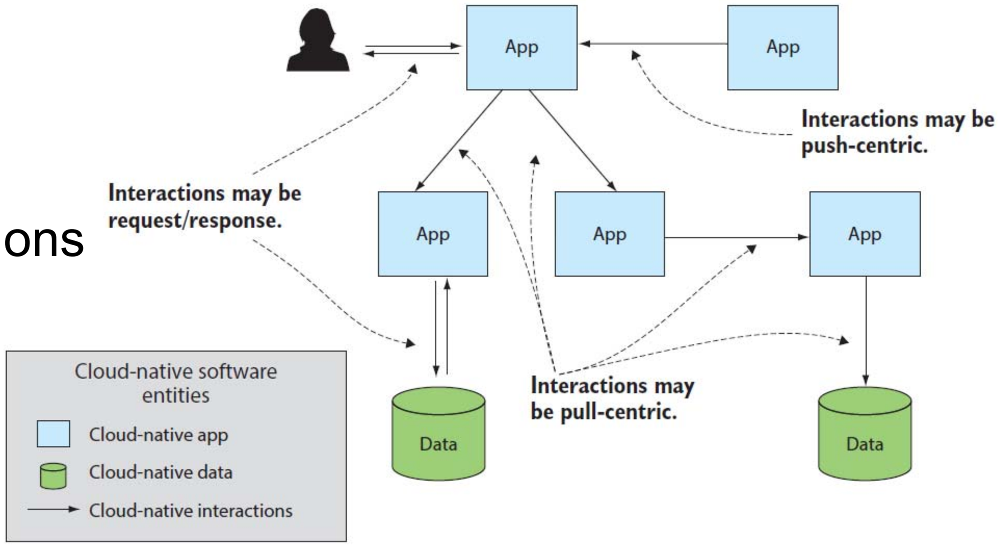
https://github.com/cncf/toc/blob/master/DEFINITION.md

五大原則

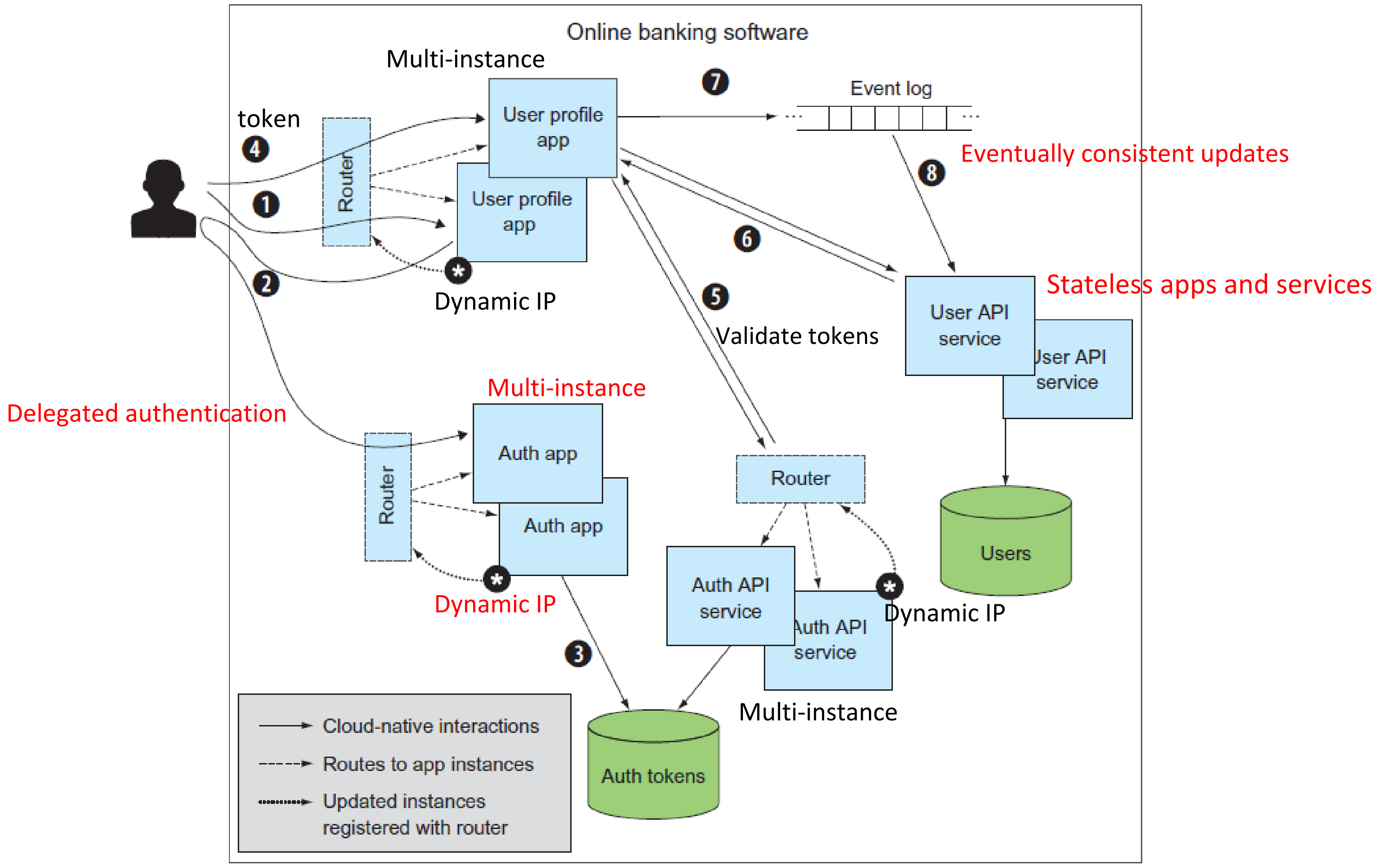
- Containerization
- Dynamic management
- Microservices
- Automation
- Orchestration

Characteristics of Cloud Native Systems

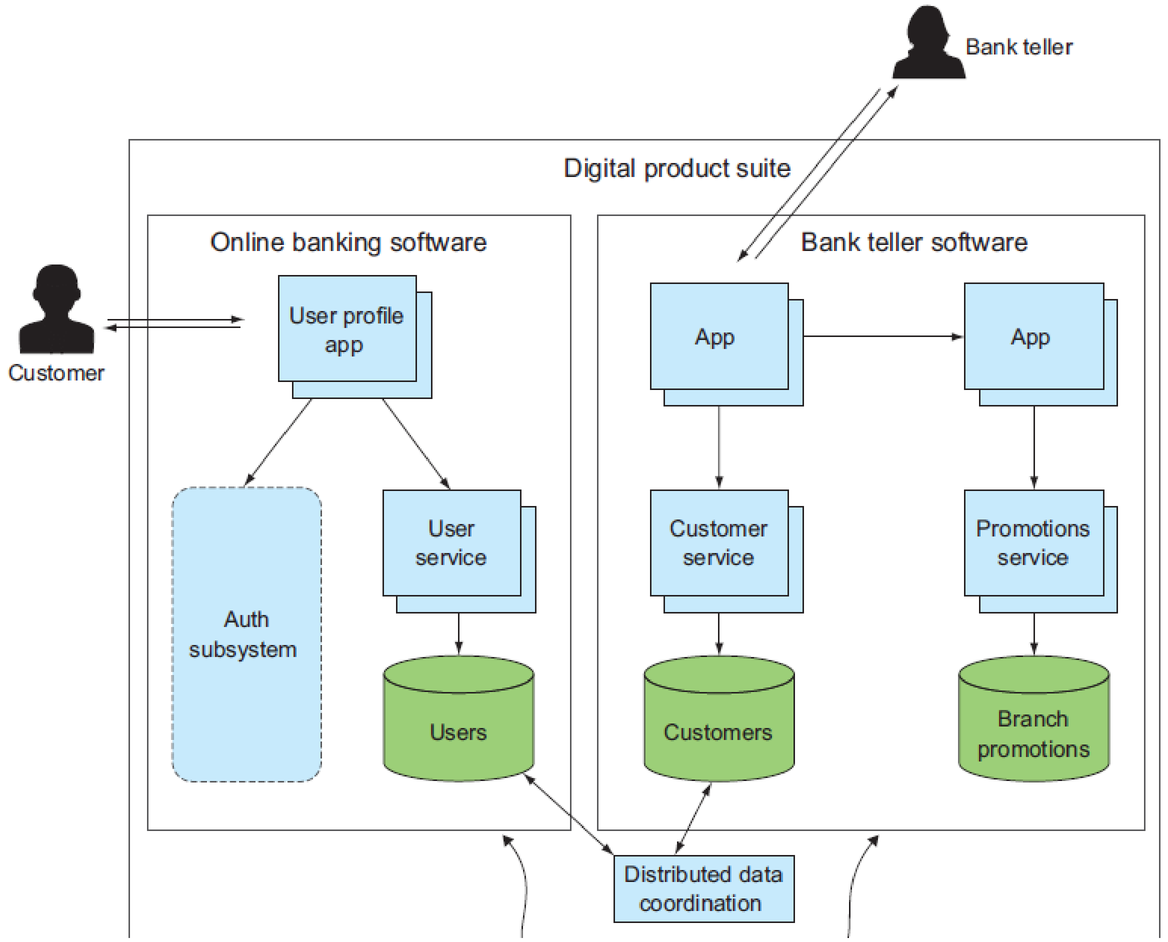
- Cloud native app
  - Stateless, redundant, scalable, dynamic deployable
- Cloud native data
  - Event sourcing
    - Treating state as an outcome of a series of modifications forms the core of data fabric
  - Query/Write operations are typically separated
    - Query: catchable
    - Write: eventually consistency
- Cloud native interactions
  - Late binding → Indirect interactions
    - Circuit breakers
    - Proxy (dynamic routing)


App
App
Interactions may be push-centric.
Interactions may be request/response.
App
App
App
Data
Interactions may be pull-centric.
Data
Cloud-native software entities
Cloud-native app
Cloud-native data
Cloud-native interactions

A Cloud Native System Example



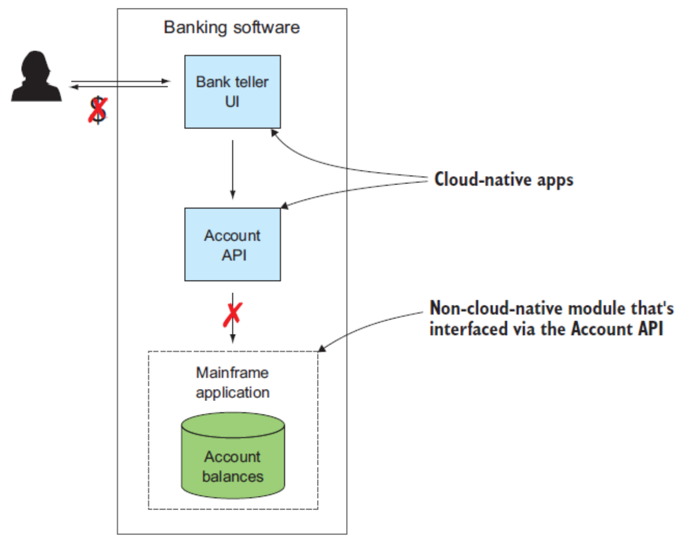
A Cloud Native System Example (2)

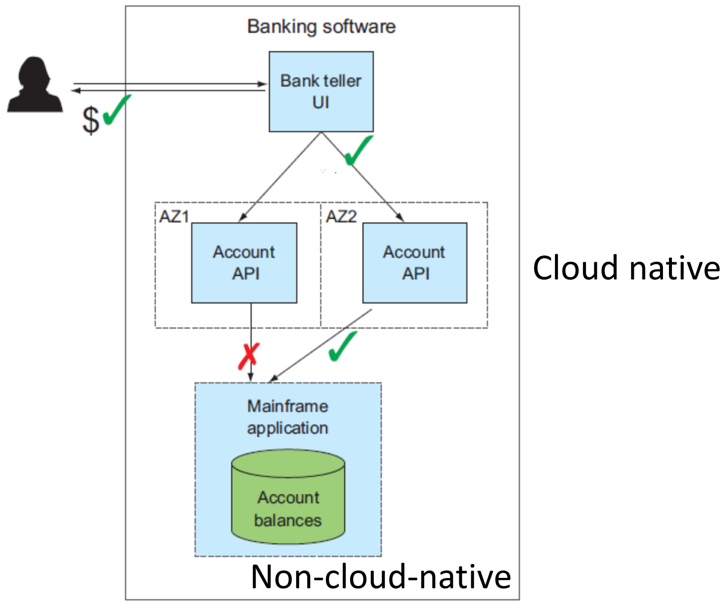


Cloud native 系統本身必須設計處理跨服務資料同步機制
(Eventually Consistent)

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

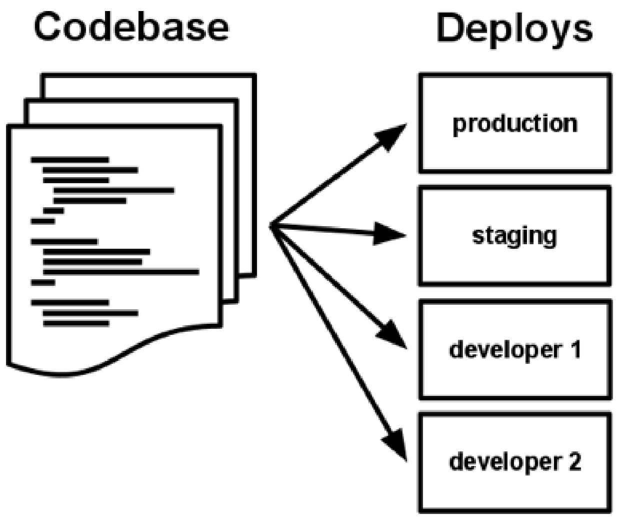
Twelve Factor App

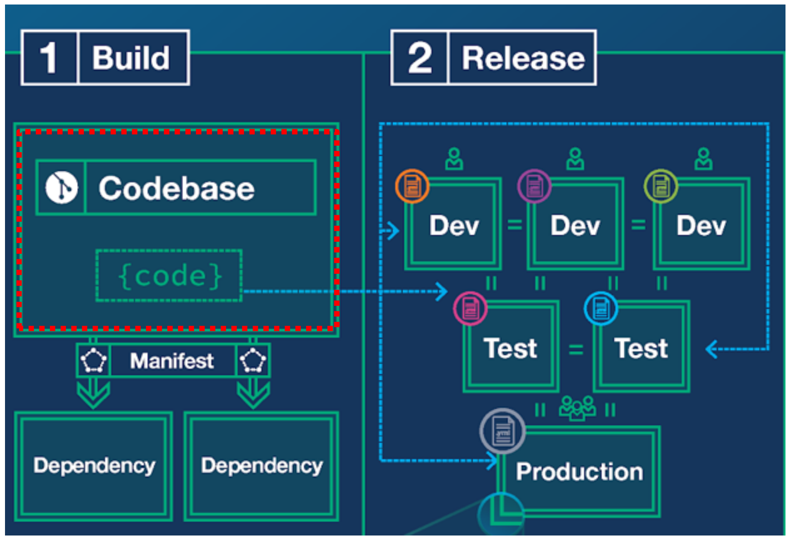
https://12factor.net/

- 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)

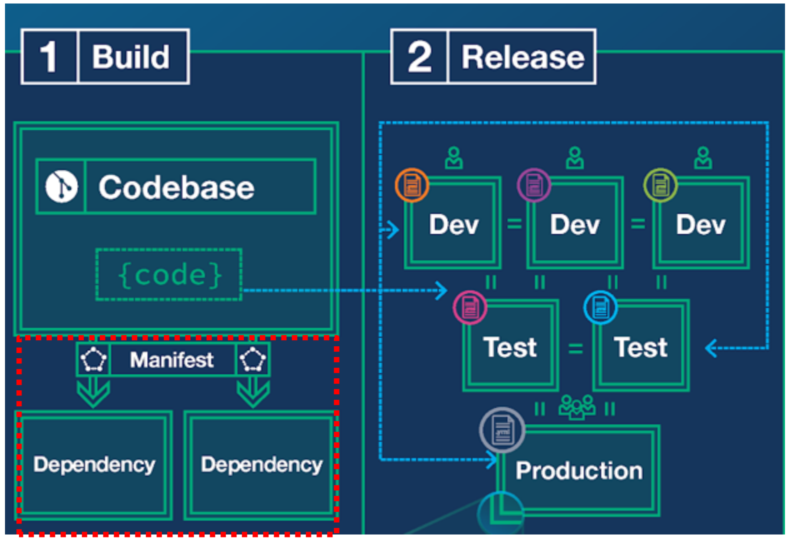




https://marcus116.blogspot.com/2020/09/architecture-12-factor-app.html

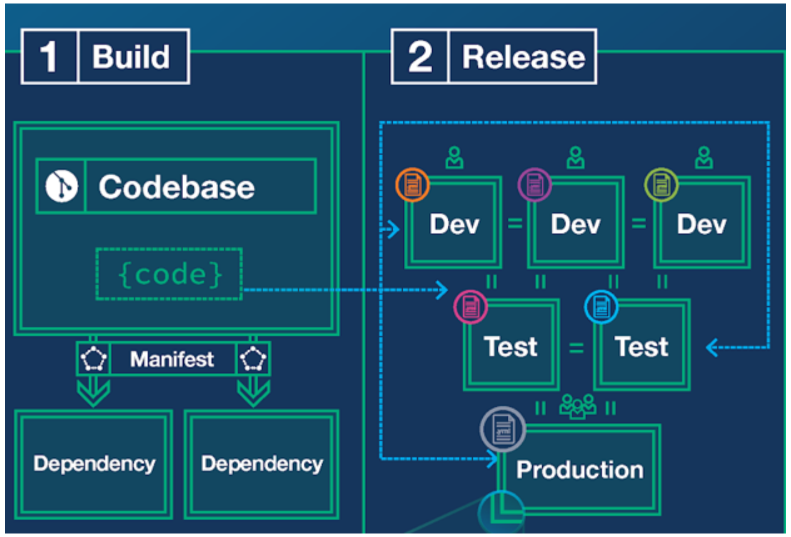
Twelve Factor App

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



Twelve Factor App

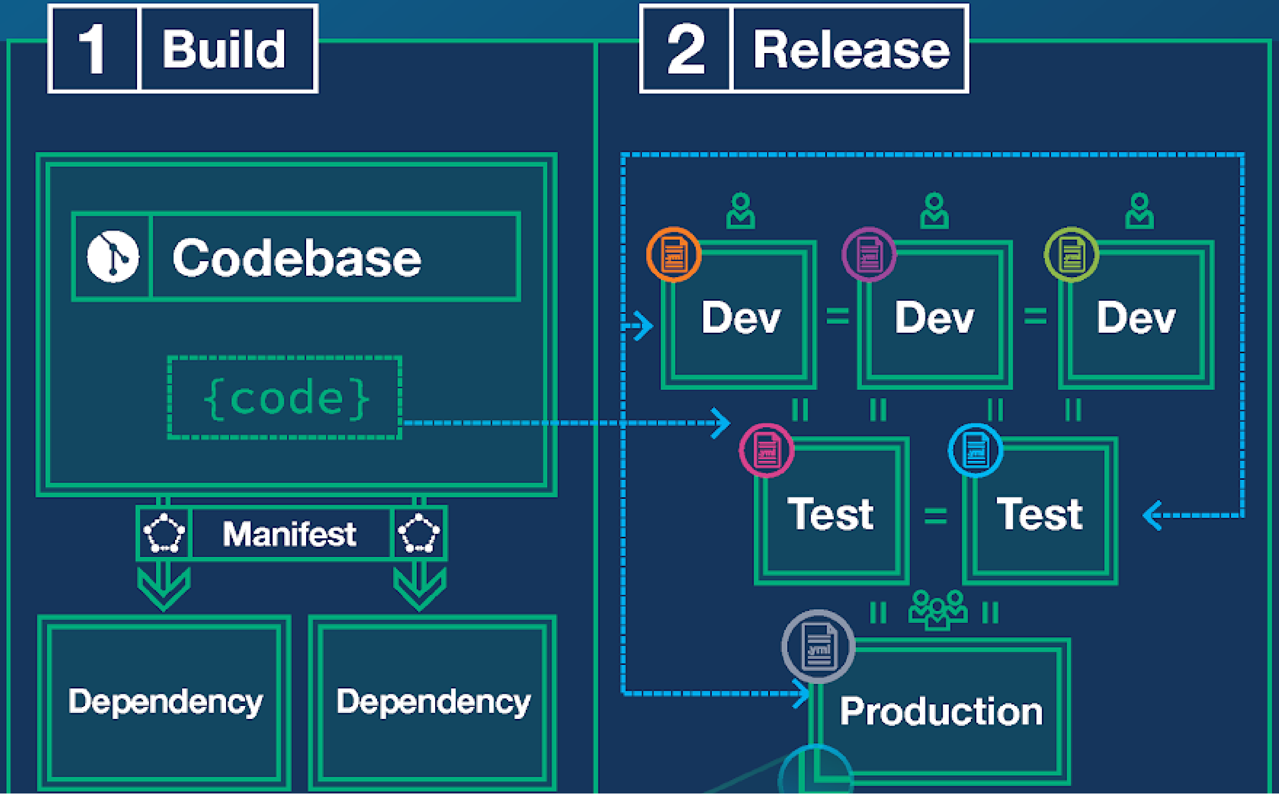
- Config
  - Store config in the environment (環境變數)
    - 應用程式的設定拉出來透過環境變數修改
    - 例: docker run --rm -it --name db -e MYSQL_ROOT_PASSWORD=passpercona



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.



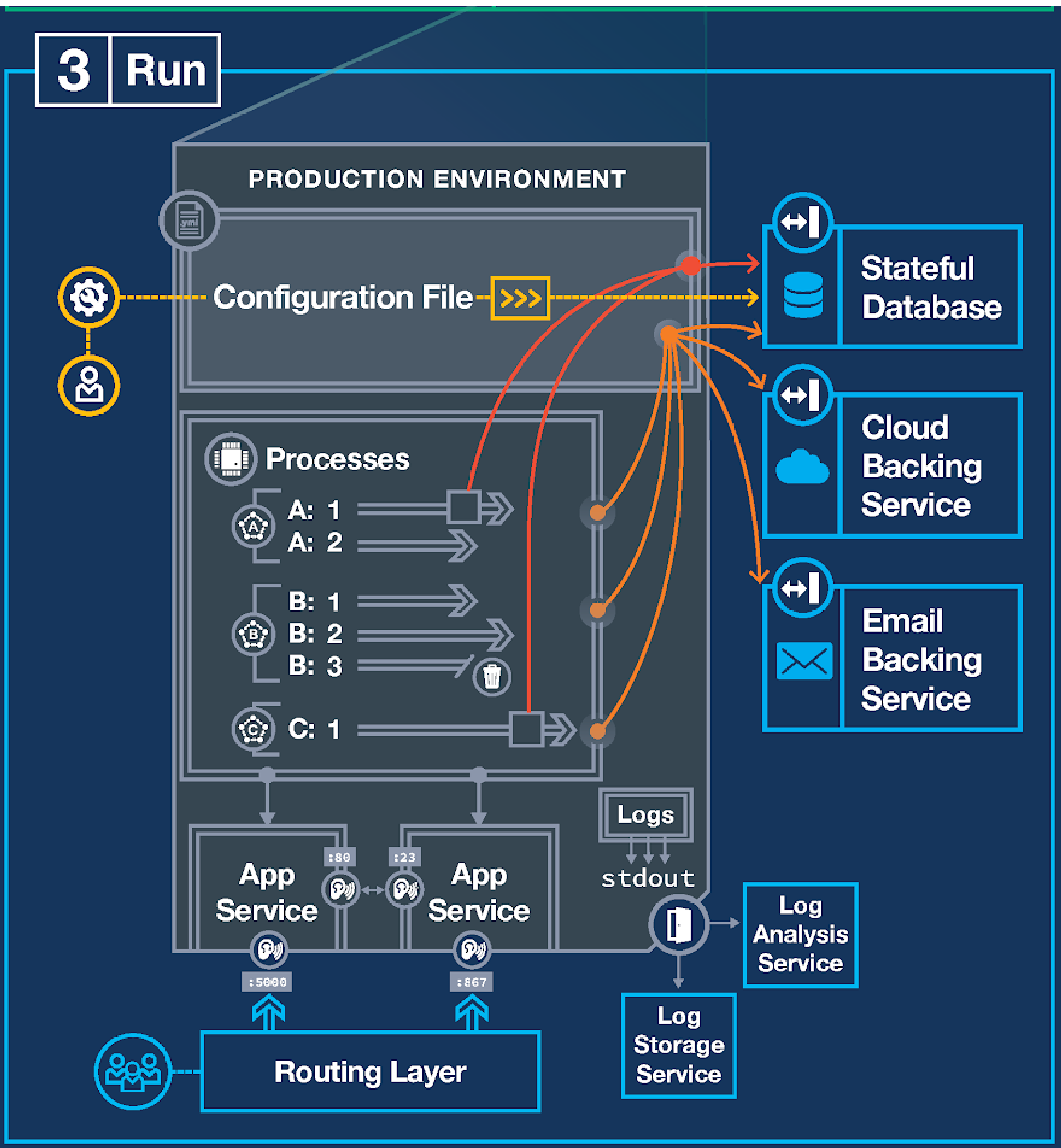
the 12 factors

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



Strictly separate build, release, and 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)

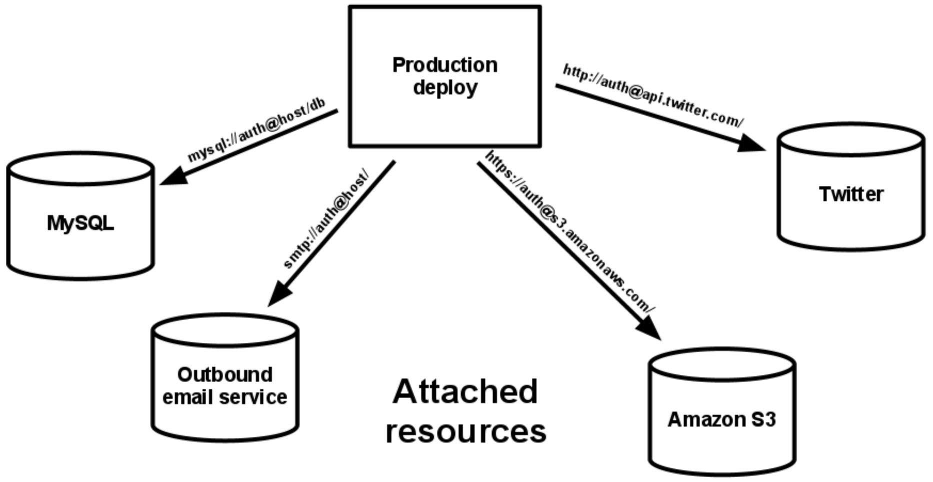
Logs
Log by writing all output streams to stdout; rout streams using non-app services

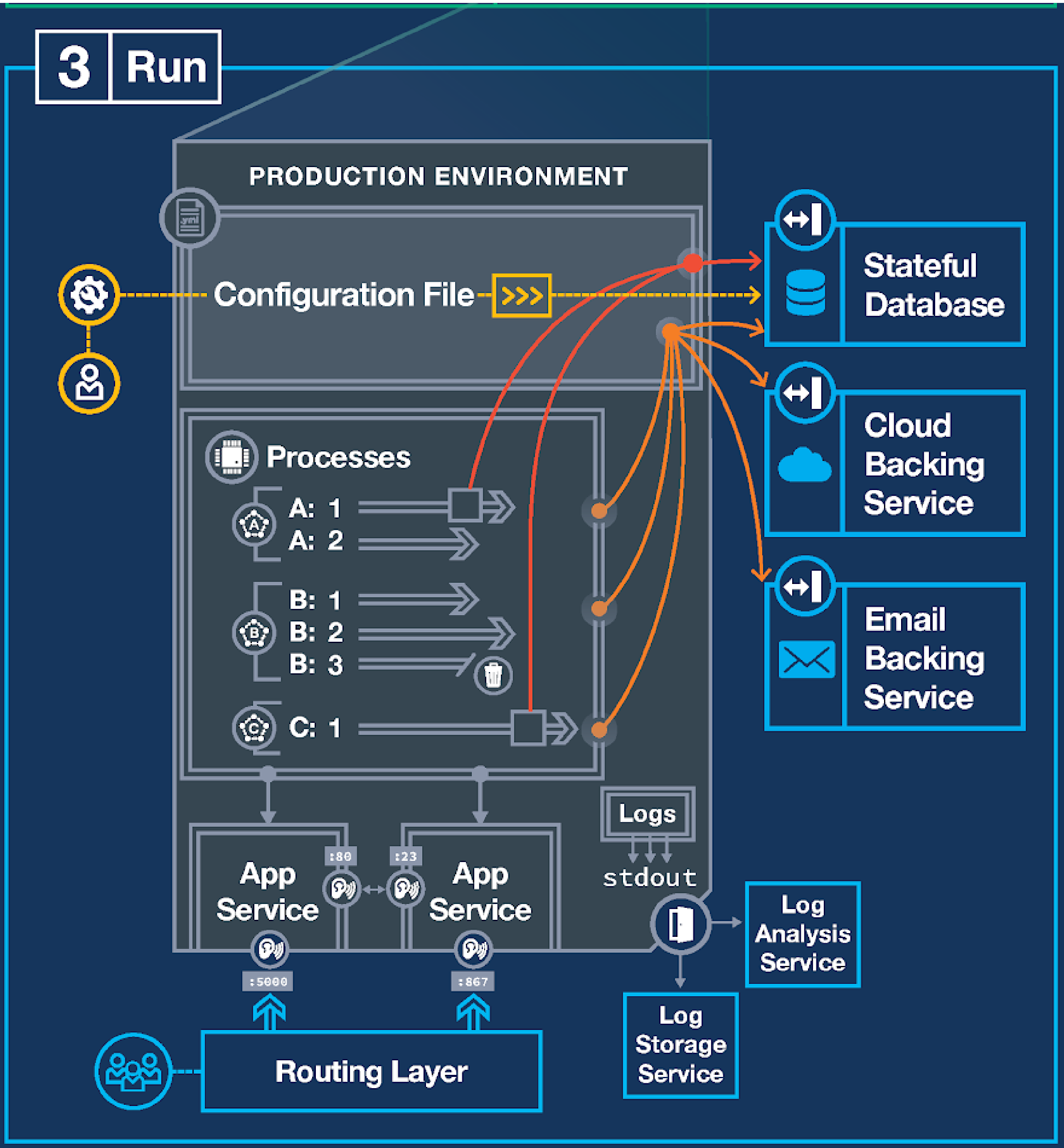
Admin Processes
Run one-off/admin processes (db migration, REPL , one-time scripts) in same environment as normal processes

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





Strictly separate build, release, and 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)

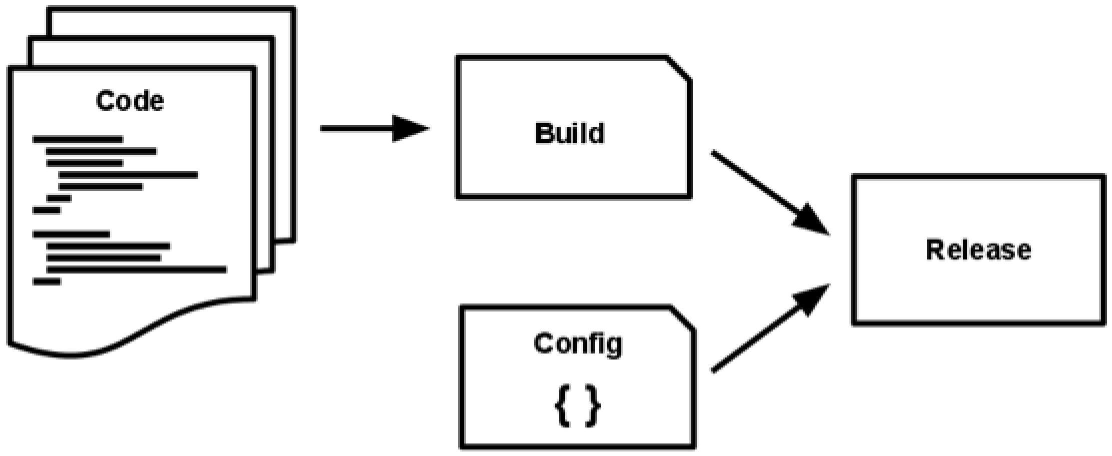
Logs
Log by writing all output streams to stdout; rout streams using non-app services

Admin Processes
Run one-off/admin processes (db migration, REPL , one-time scripts) in same environment as normal processes

CREATED BY DZONE.COM

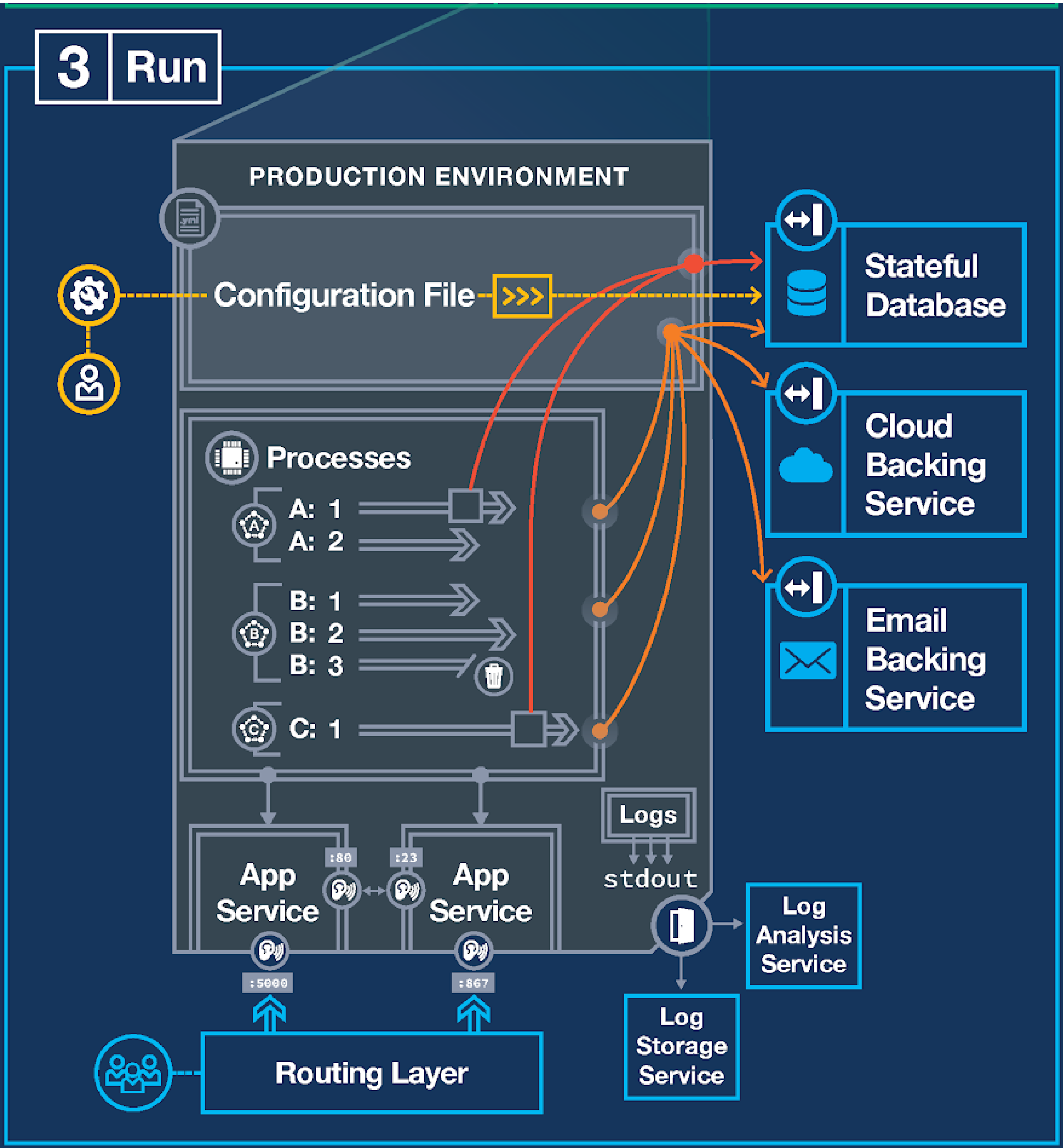
Twelve Factor App

- 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



Strictly separate build, release, and 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)

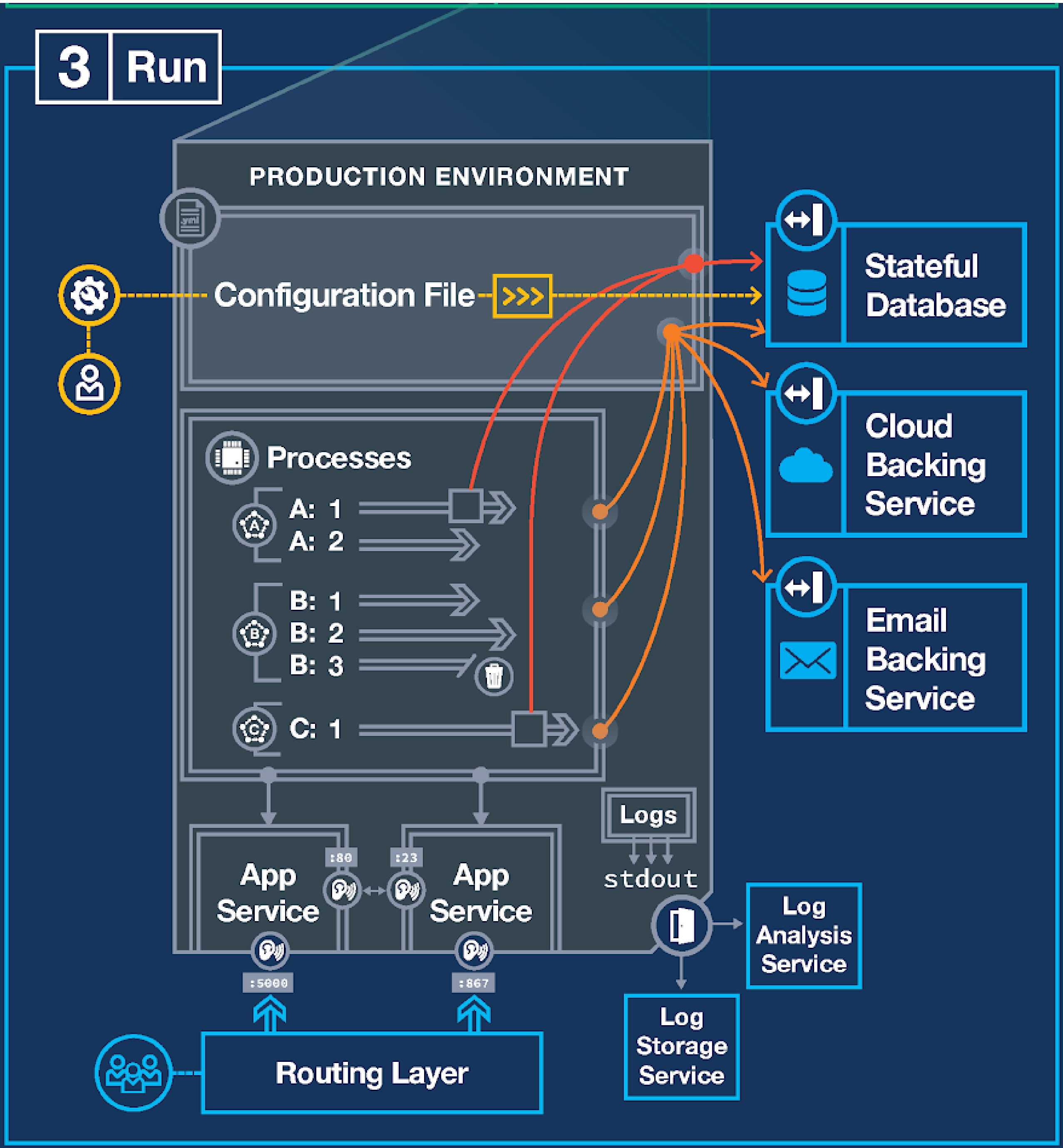
Logs
Log by writing all output streams to stdout; rout streams using non-app services

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



Strictly separate build, release, and 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

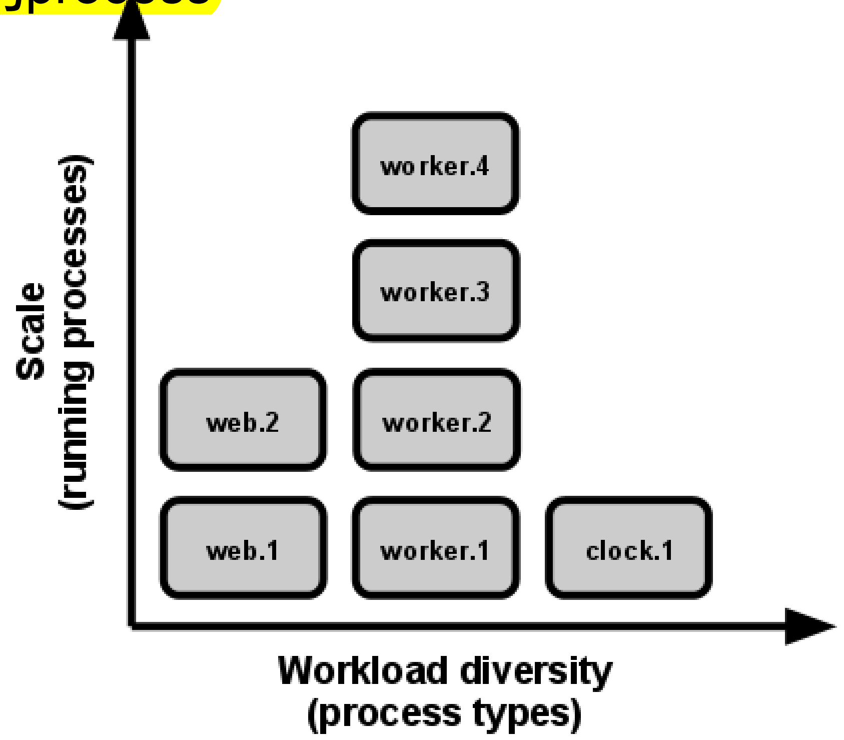
Admin Processes
Run one-off/admin processes (db migration, REPL , one-time scripts) in same environment as normal processes

CREATED BY DZONE.COM

Twelve Factor App

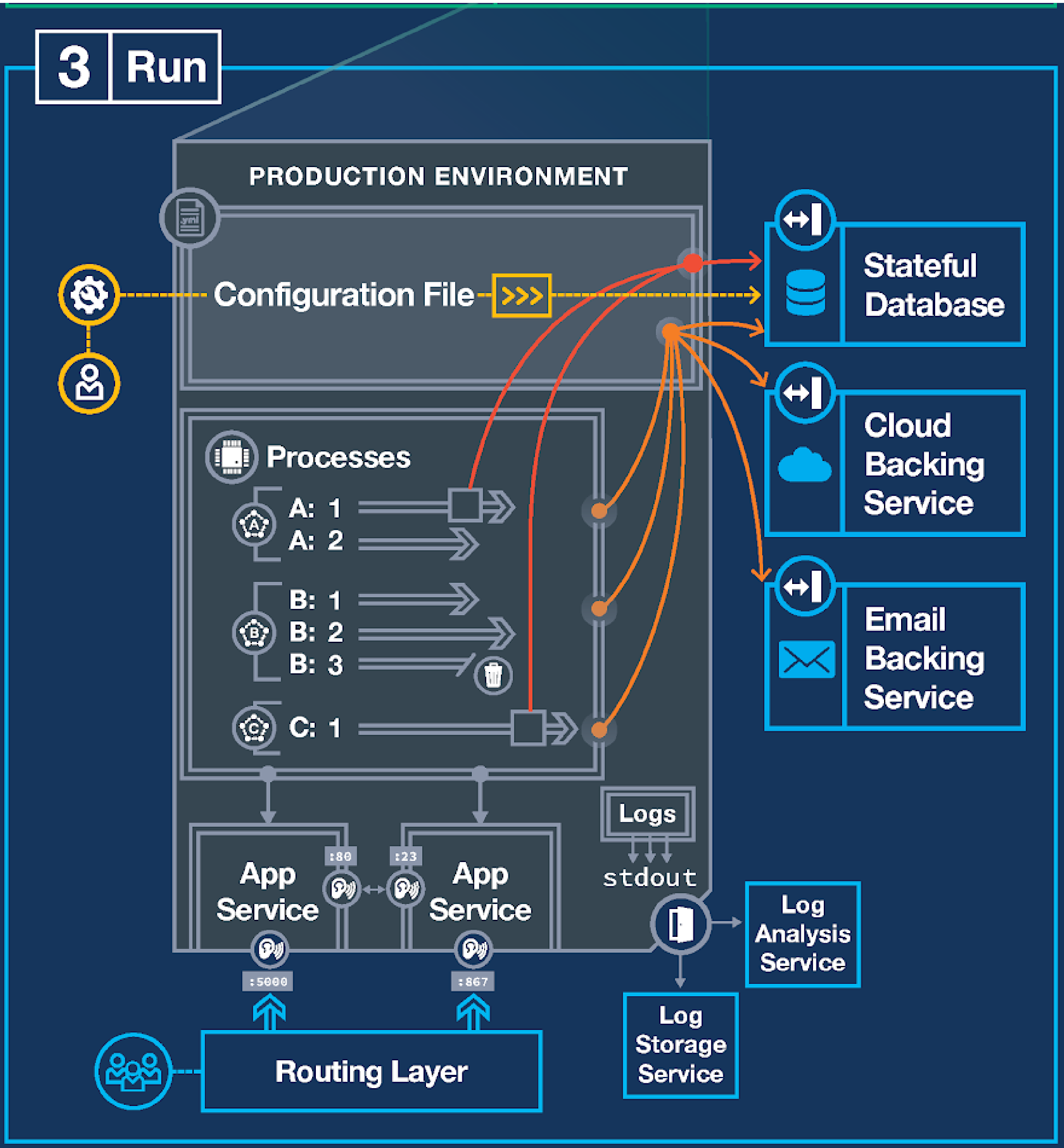
Application 層面使用 process
因為 process 可以無狀態
但 thread 會共享全域變數，有狀態

- 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



Strictly separate build, release, and 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

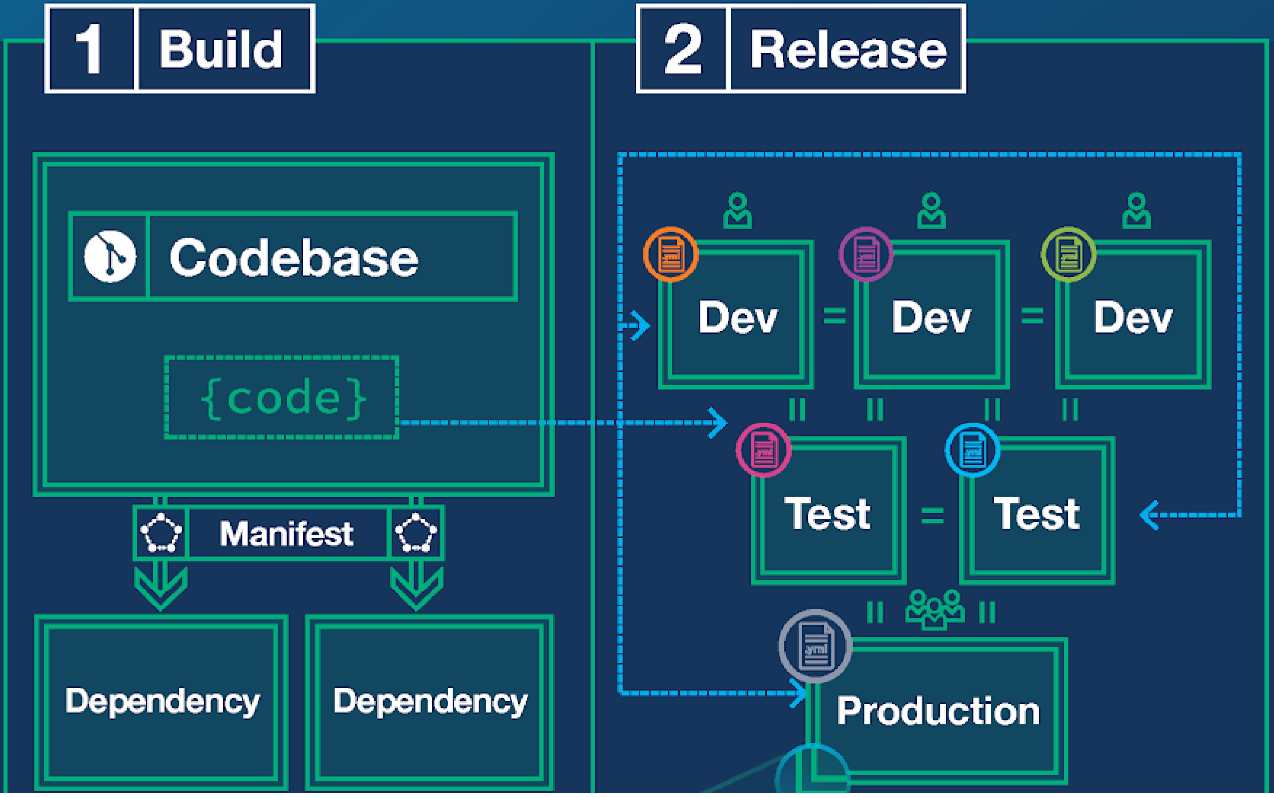
CREATED BY DZONE.COM

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

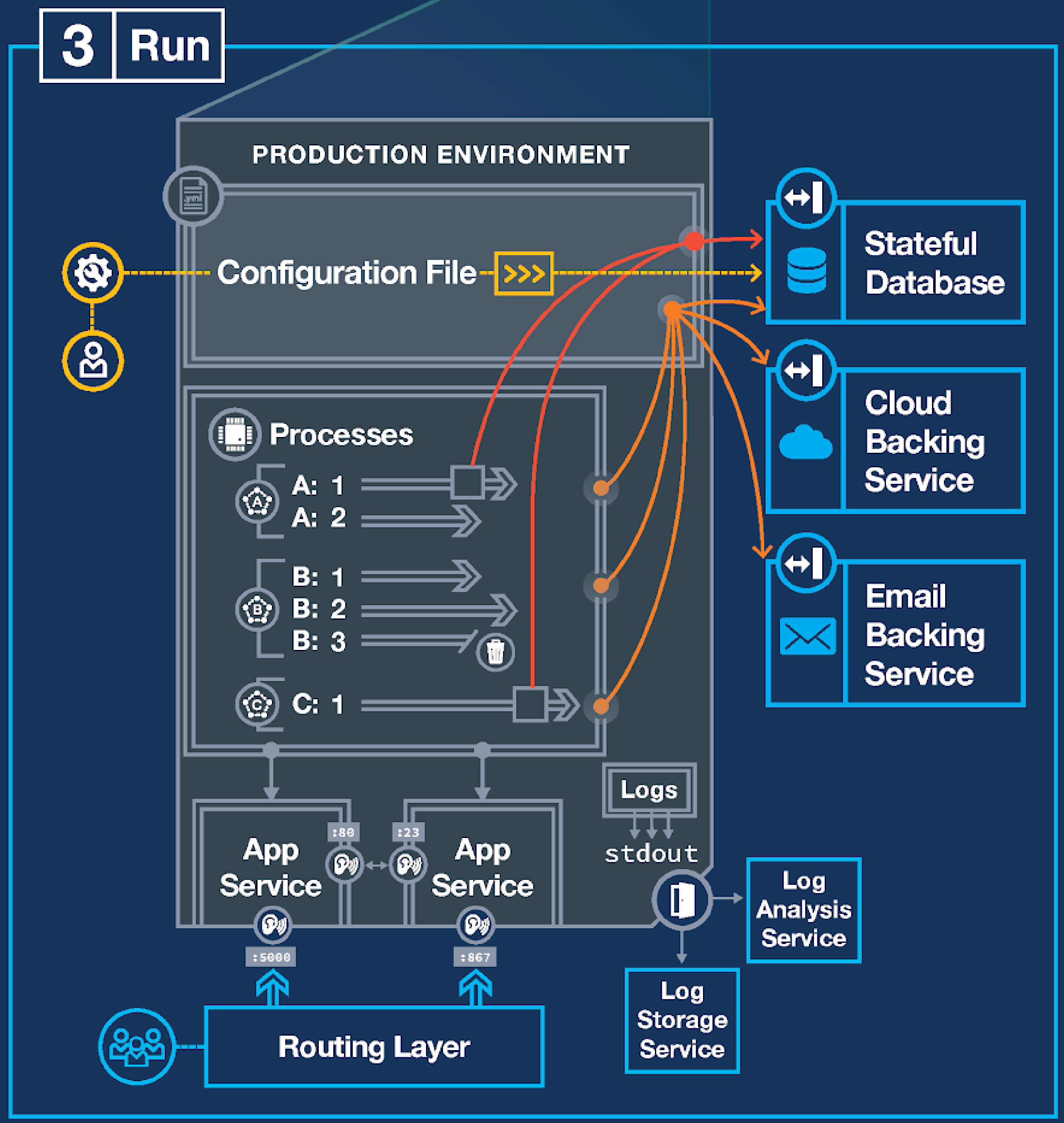
Dependencies
Explicitly declare and isolate dependencies

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 Fluented 等工具去擷取後，透過 Elasticsearch 分析及 Kinbana 呈現結果



Strictly separate build, release, and 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)

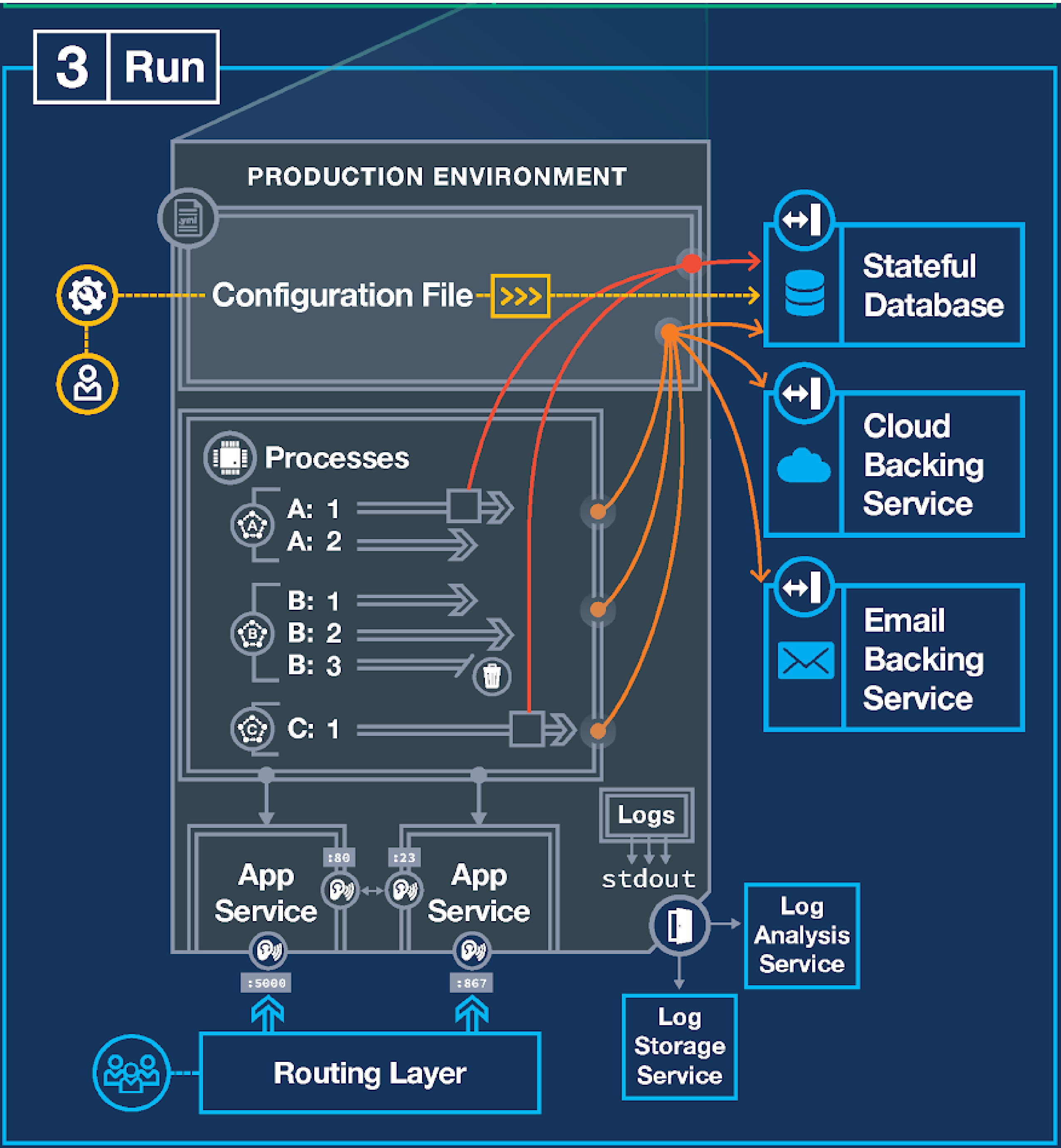
Logs
Log by writing all output streams to stdout; rout streams using non-app services

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
    - docker exec -it web php artisan migrate
  - 一次性管理程式應該和正常的程式使用同樣的環境
  - 一次性管理程式也要像原始碼一般進行版本控制



Strictly separate build, release, and 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)

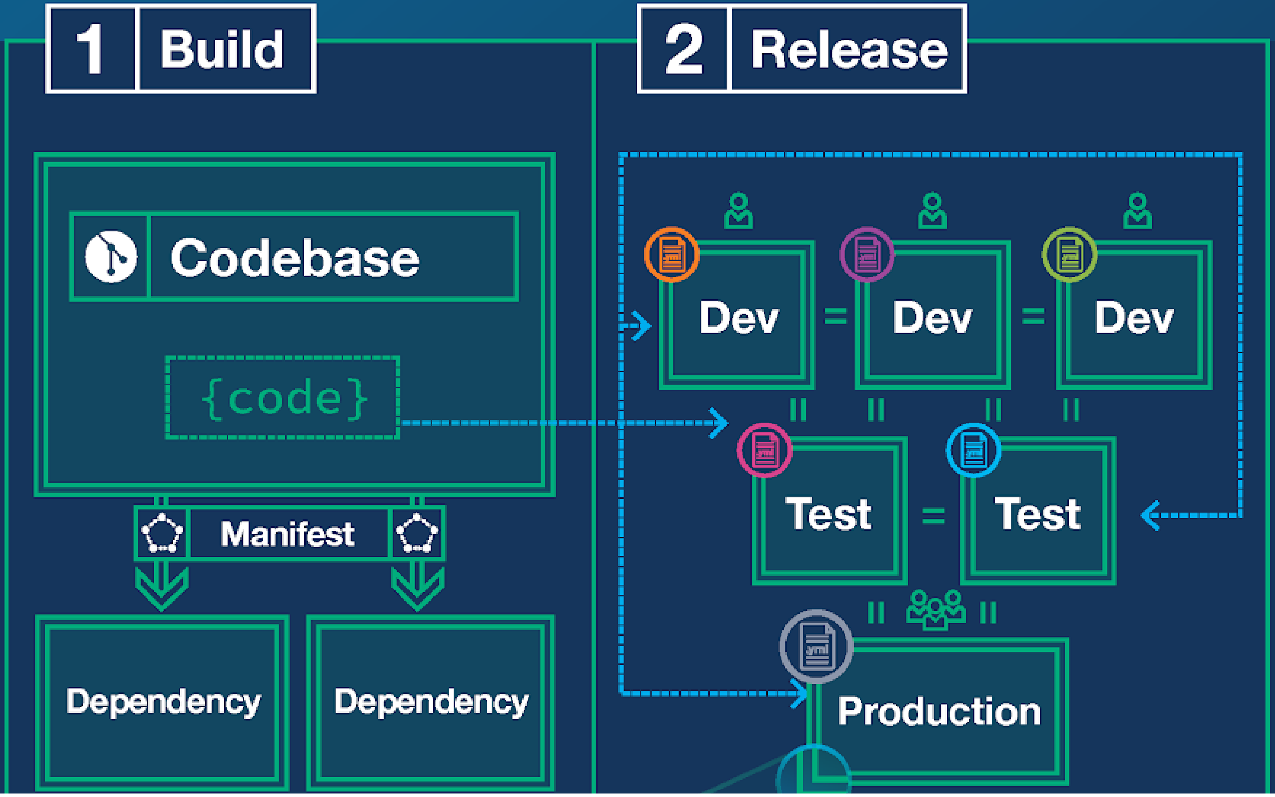
Logs
Log by writing all output streams to stdout; rout streams using non-app services

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

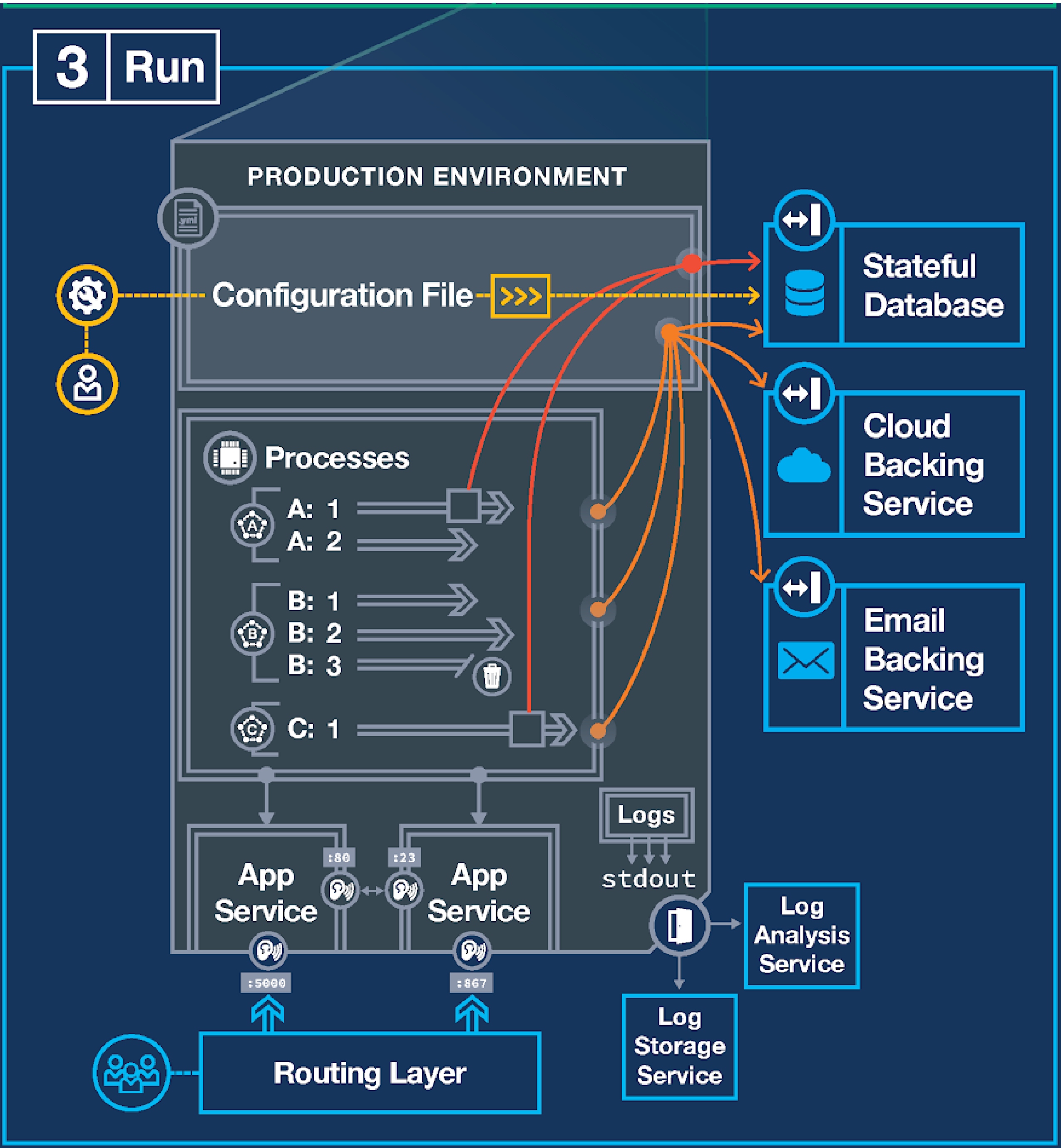
Codebase
One codebase, many deploys, strict version control

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



Strictly separate build, release, and 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)

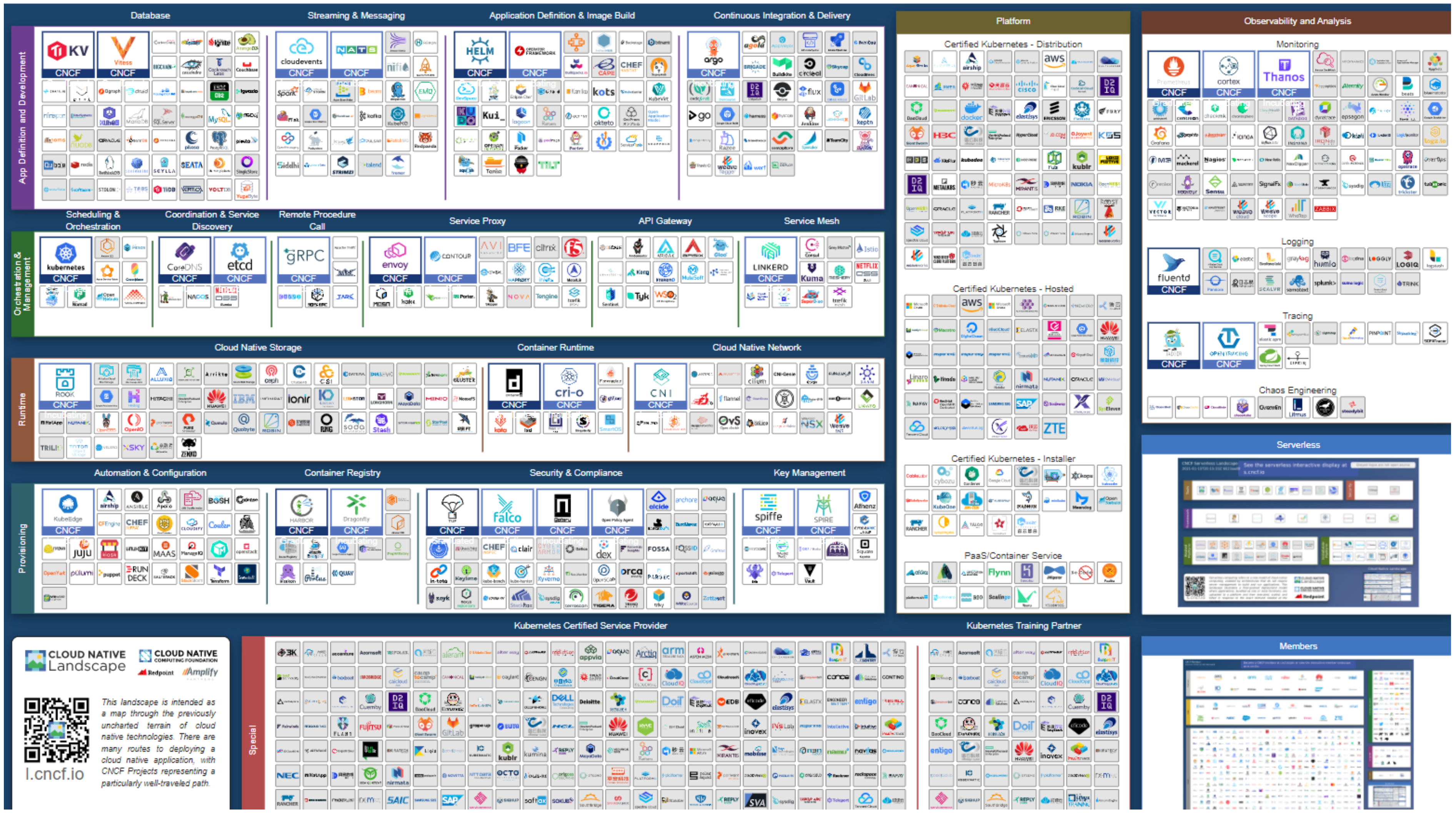
Dev/Prod Parity
Maximize dev/prod parity by minimizing gaps in time (between deploys: hours), personnel (authors=deployers), and environment (use adapters for backing services)

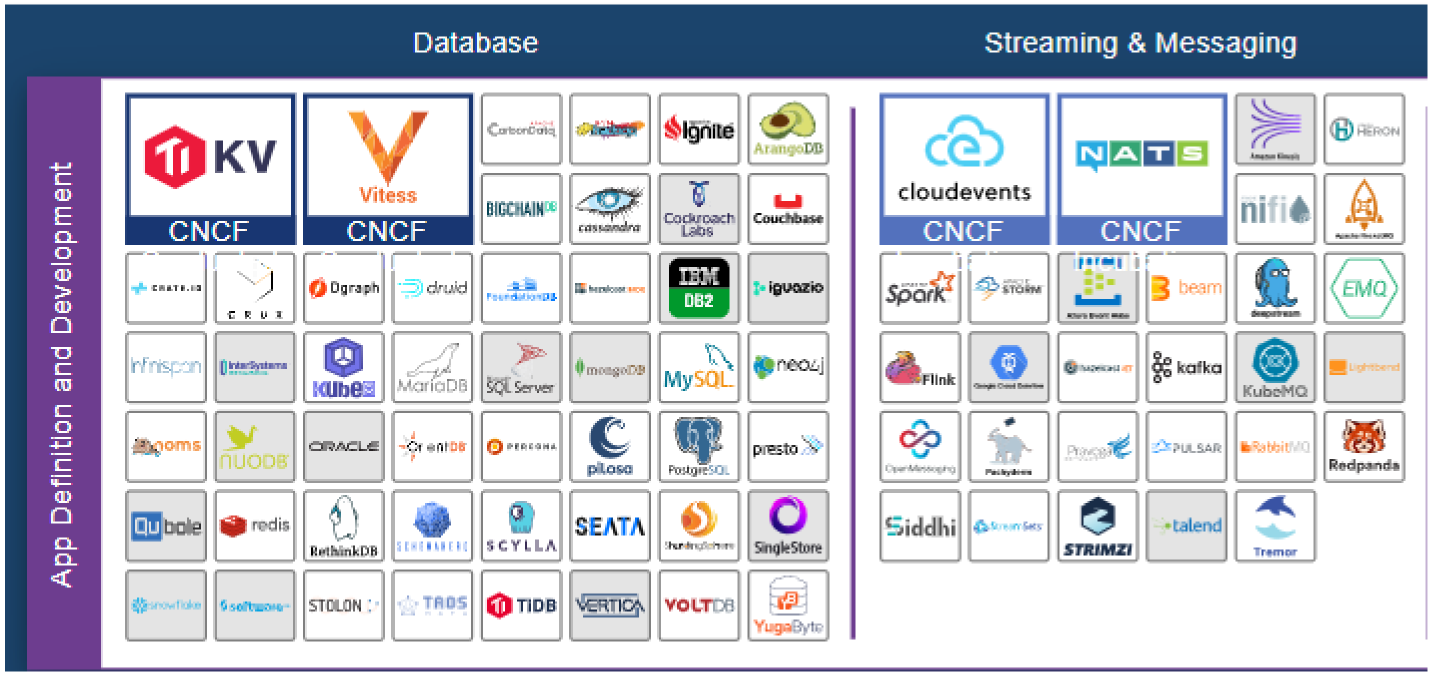
Logs
Log by writing all output streams to stdout; rout streams using non-app services

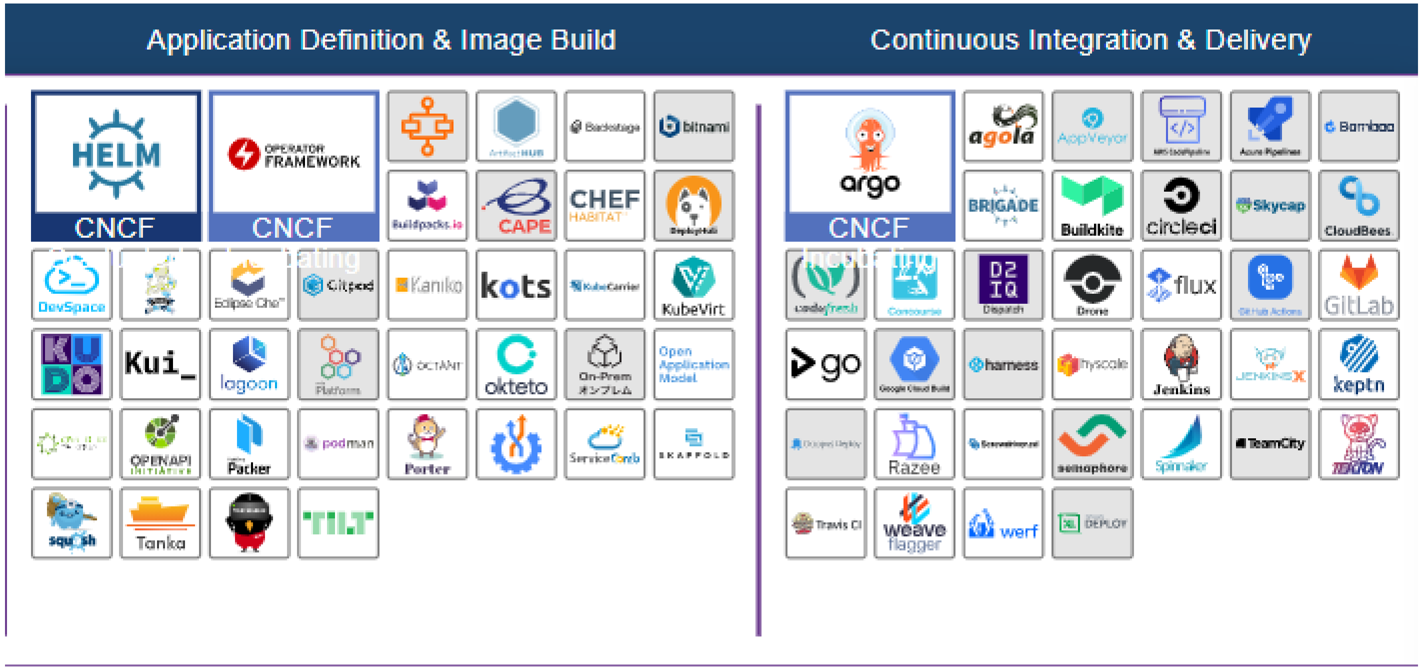
Admin Processes
Run one-off/admin processes (db migration, REPL , one-time scripts) in same environment as normal processes

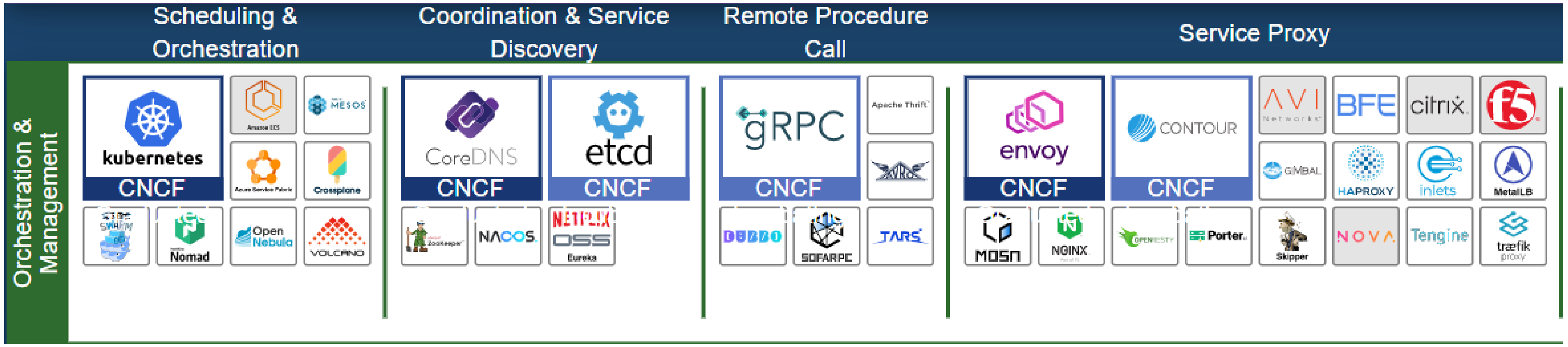
CREATED BY DZONE.COM

CNCF Cloud Native Landscape

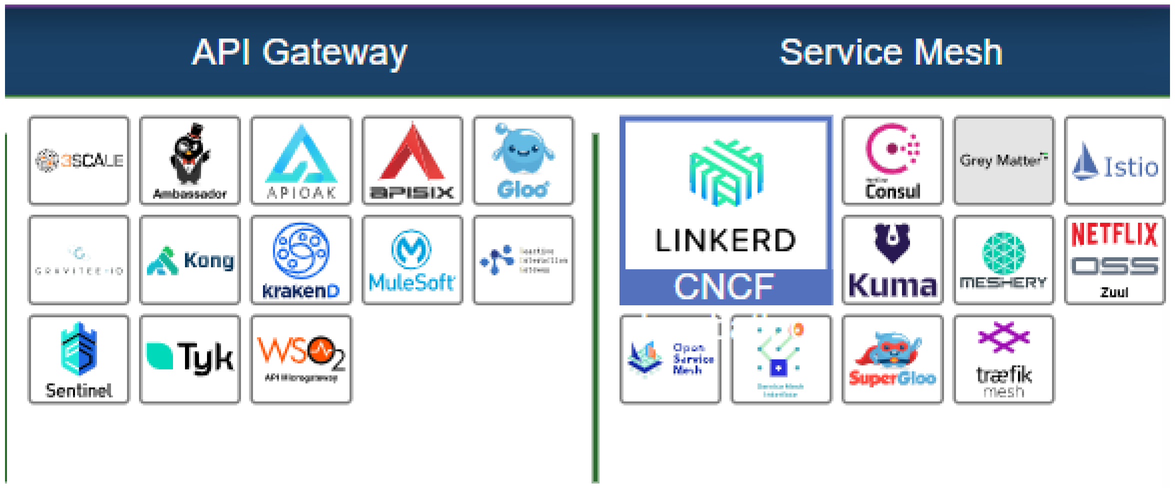

Database
Streaming & Messaging
Application Definition & Image Build
Continuous Integration & Delivery
Platform
Observability and Analysis
App Definition and Development
Certified Kubernetes - Distribution
Monitoring
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
API Gateway
Service Mesh
Orchestration & Management
Logging
Certified Kubernetes - Hosted
Tracing
Cloud Native Storage
Container Runtime
Cloud Native Network
Runtime
Chaos Engineering
Certified Kubernetes - Installer
Serverless
Automation & Configuration
Container Registry
Security & Compliance
Key Management
Provisioning
PaaS/Container Service
Kubernetes Certified Service Provider
Kubernetes Training Partner
Members
Special
CLOUD NATIVE Landscape
CLOUD NATIVE COMPUTING FOUNDATION
l.cncf.io
This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.


App Definition and Development
Database
Streaming & Messaging
TiKV
CNCF
Vitess
CNCF
CarbonData
Ignite
ArangoDB
BigchainDB
cassandra
Cockroach Labs
Couchbase
CRATE.IO
CRUX
Dgraph
druid
FoundationDB
IBM DB2
iguazio
Infinispan
InterSystems
KubeDB
MariaDB
SQL Server
mongoDB
MySQL
neo4j
noms
NuoDB
ORACLE
entDB
PERCONA
pilosa
PostgreSQL
presto
Qubole
redis
RethinkDB
SCHEMAHERO
SCYLLA
SEATA
ShardingSphere
SingleStore
snowflake
softwareAG
STOLON
TAOS
TiDB
VERTICA
VOLTDB
YugaByte
cloudevents
CNCF
NATS
CNCF
Amazon Kinesis
HERON
nifi
Apache RocketMQ
Spark
STORM
Azure Event Hubs
beam
deepstream
EMQ
Flink
Google Cloud Dataflow
hazelcast
kafka
KubeMQ
Lightbend
OpenMessaging
Pachyderm
Pravega
PULSAR
RabbitMQ
Redpanda
Siddhi
StreamSets
STRIMZI
talend
Tremor

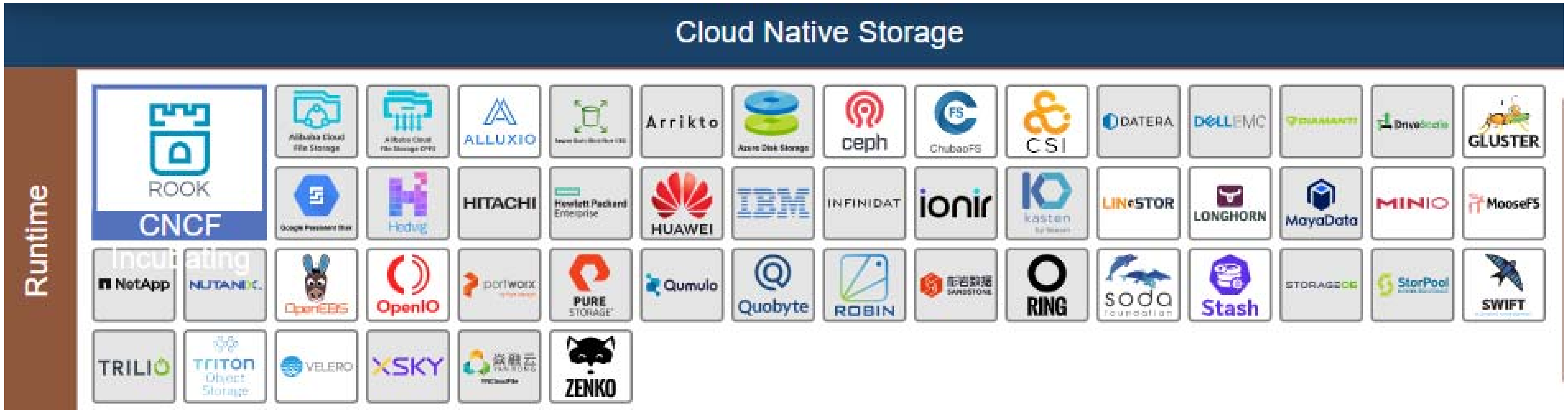
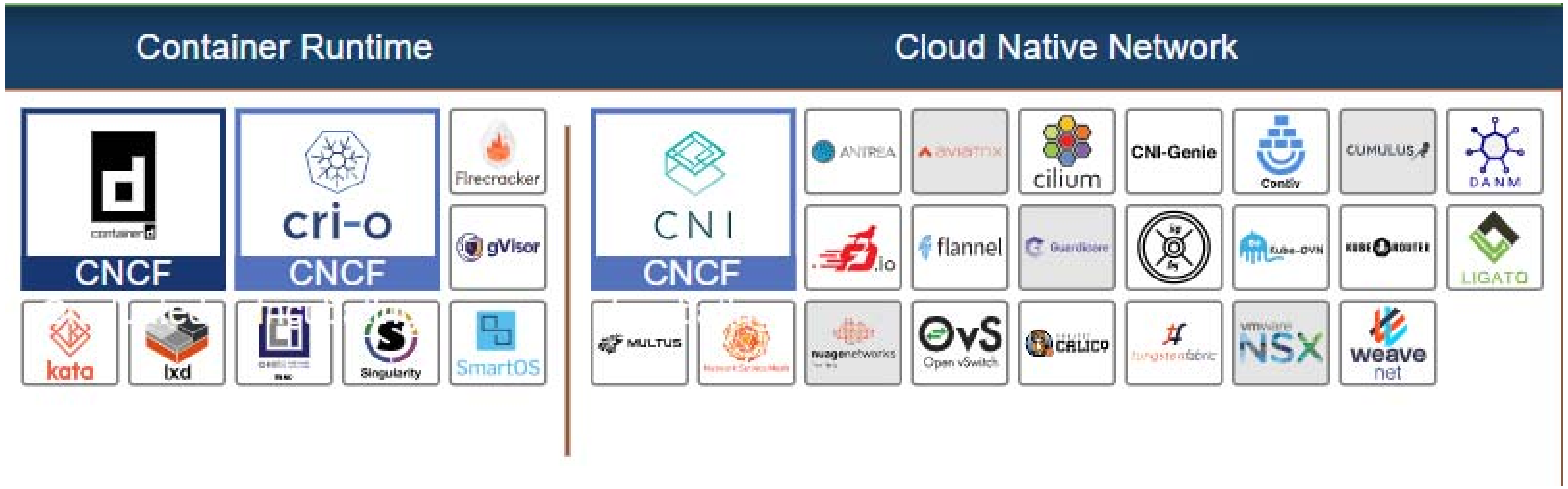

Application Definition & Image Build
Continuous Integration & Delivery
HELM
CNCF
OPERATOR FRAMEWORK
CNCF
ArtifactHUB
Backstage
bitnami
Buildpacks.io
CAPE
CHEF HABITAT
DeployHub
DevSpace
Eclipse Che
Gitpod
Kaniko
kots
KubeCarrier
KubeVirt
KUDO
Kui
lagoon
Platform
OCTANT
okteto
On-Prem
Open Application Model
OPENAPI INITIATIVE
Packer
podman
Porter
ServiceComb
SKAFFOLD
squash
Tanka
TILT
argo
CNCF
agola
AppVeyor
AWS CodePipeline
Azure Pipelines
Bamboo
BRIGADE
Buildkite
circleci
Skycap
CloudBees
codefresh
Concourse
Dispatch
Drone
flux
GitHub Actions
GitLab
go
Google Cloud Build
harness
hyscale
Jenkins
JENKINS X
keptn
Octopus Deploy
Razee
Screwdriver.cd
semaphore
Spinnaker
TeamCity
TEKTON
Travis CI
weave flagger
werf
XL DEPLOY

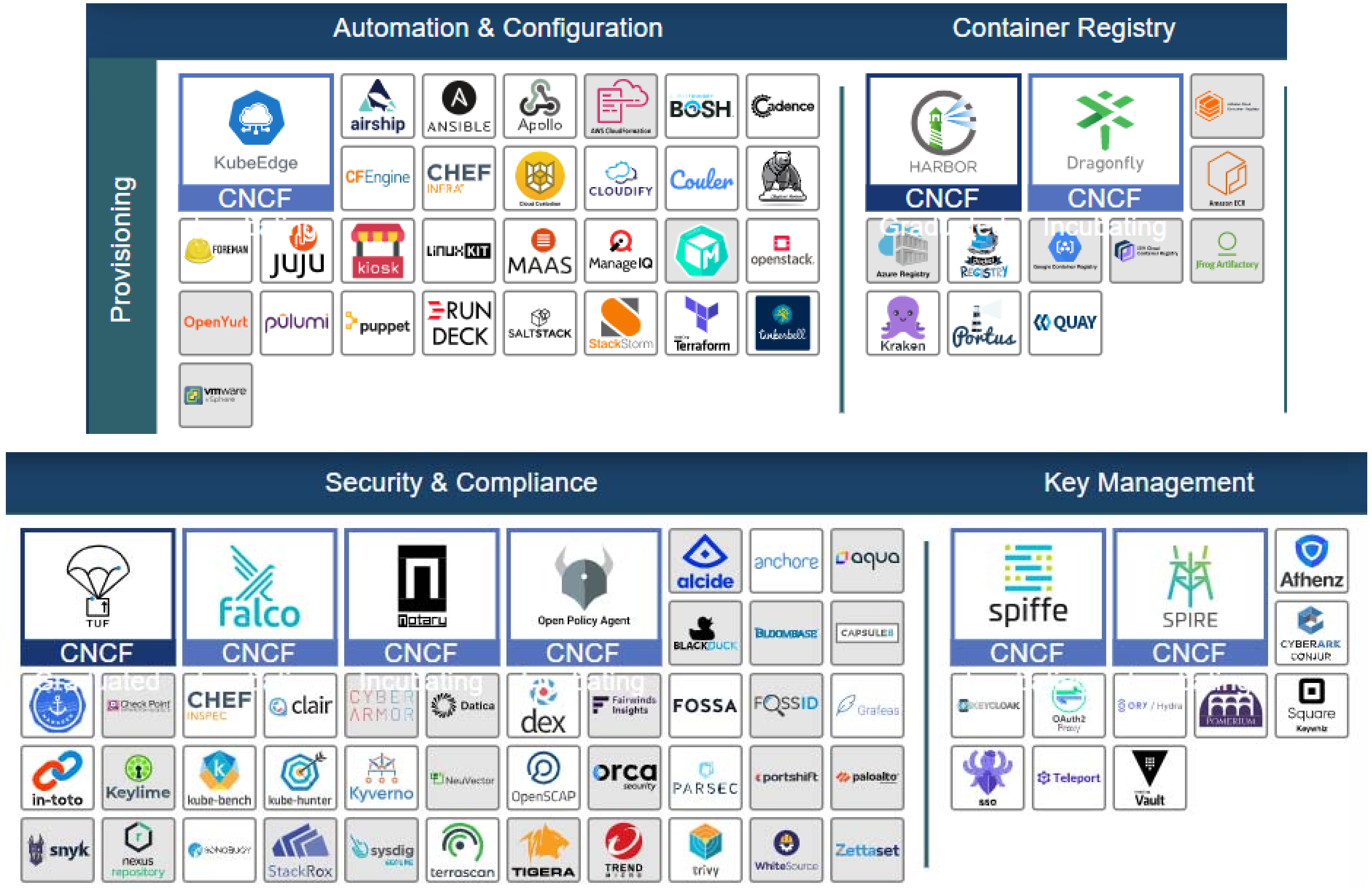


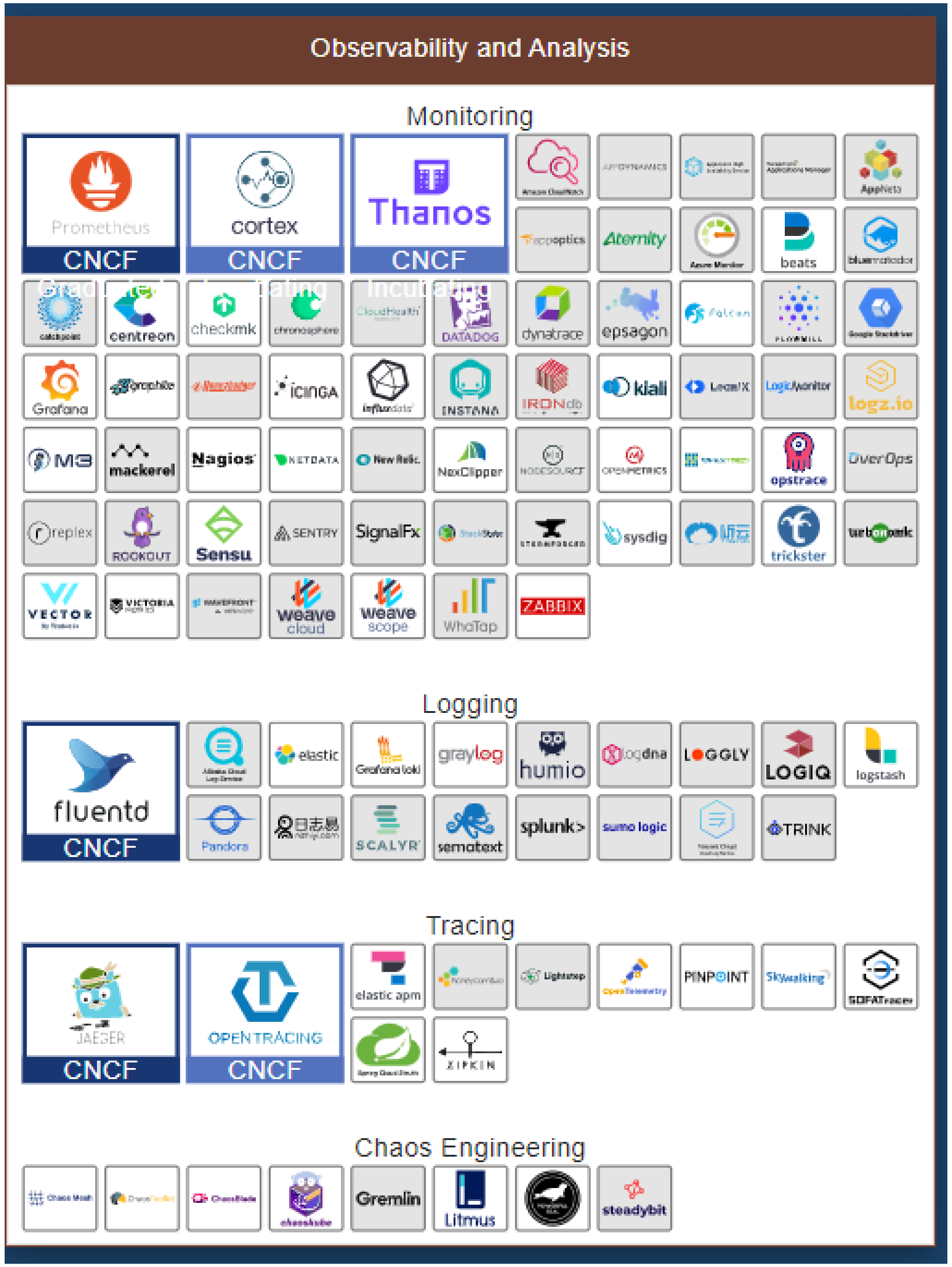
Orchestration & Management
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
kubernetes
CNCF
Amazon ECS
MESOS
Azure Service Fabric
Crossplane
Docker Swarm
Nomad
Open Nebula
VOLCANO
CoreDNS
CNCF
etcd
CNCF
ZooKeeper
NACOS.
NETFLIX OSS Eureka
gRPC
CNCF
Apache Thrift
AVRO
DUBBO
SOFARPC
TARS
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
Reactive Interaction Gateway
Sentinel
Tyk
WSO2 API Microgateway
LINKERD
CNCF
Consul
Grey Matter
Istio
Kuma
MESHERY
NETFLIX OSS Zuul
Open Service Mesh
Service Mesh Interface
SuperGloo
træfik mesh


Cloud Native Storage
Runtime
ROOK
CNCF
Incubating
Alibaba Cloud File Storage
Alibaba Cloud File Storage CPFS
ALLUXIO
Arrikto
Azure Disk Storage
ceph
ChubaoFS
CSI
DATERA
DELLEMC
DIAMANTI
DriveScale
GLUSTER
Google Persistent Disk
Hedvig
HITACHI
Hewlett Packard Enterprise
HUAWEI
IBM
INFINIDAT
ionir
kasten
LINSTOR
LONGHORN
MayaData
MINIO
MooseFS
NetApp
NUTANIX
OpenEBS
OpenIO
portworx
PURE STORAGE
Qumulo
Quobyte
ROBIN
SANDSTONE
RING
soda foundation
Stash
STORAGEOS
StorPool
SWIFT
TRILIO
Triton Object Storage
VELERO
XSKY
ZENKO

Container Runtime
Cloud Native Network
containerd
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
Network Service Mesh
nuagenetworks
OvS Open vSwitch
CALICO
tungstenfabric
vmware NSX
weave net


Automation & Configuration
Container Registry
Provisioning
KubeEdge
CNCF
Incubating
airship
ANSIBLE
Apollo
AWS CloudFormation
BOSH
Cadence
CFEngine
CHEF INFRA
Cloud Custodian
CLOUDIFY
Couler
FOREMAN
JUJU
kiosk
LinuxKit
MAAS
ManageIQ
openstack
OpenYurt
pulumi
puppet
RUN DECK
SALTSTACK
StackStorm
Terraform
tinkerbell
vmware vSphere
HARBOR
CNCF
Graduated
Dragonfly
CNCF
Incubating
Amazon ECR
Azure Registry
REGISTRY
Google Container Registry
JFrog Artifactory
Kraken
Portus
QUAY
Security & Compliance
Key Management
TUF
CNCF
Graduated
Falco
CNCF
Incubating
Notary
CNCF
Incubating
Open Policy Agent
CNCF
Incubating
alcide
anchore
aqua
BLACKDUCK
BLOOMBASE
CAPSULE8
HARBOR
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
trivy
WhiteSource
Zettaset
spiffe
CNCF
Incubating
SPIRE
CNCF
Incubating
Athenz
CYBERARK CONJUR
KEYCLOAK
OAuth2 Proxy
ORY / Hydra
POMERIUM
Square Keywhiz
SSO
Teleport
Vault


Observability and Analysis
Monitoring
Prometheus
CNCF
Graduated
cortex
CNCF
Incubating
Thanos
CNCF
Incubating
AppNeta
appoptics
Aternity
Azure Monitor
beats
bluemedora
catchpoint
centreon
checkmk
chronosphere
CloudHealth
DATADOG
dynatrace
epsagon
Falcon
FLOWMILL
Google Stackdriver
Grafana
graphite
icinga
influxdata
INSTANA
IRONdb
kiali
LogicMonitor
logz.io
M3
mackerel
Nagios
NETDATA
New Relic
NexClipper
NODESOURCE
OPENMETRICS
opstrace
OverOps
replex
ROOKOUT
Sensu
SENTRY
SignalFx
StackState
STORMFORGE
sysdig
trickster
turbonomic
VECTOR
VICTORIA METRICS
WAVEFRONT
weave cloud
weave scope
WhaTap
ZABBIX
Logging
fluentd
CNCF
elastic
Grafana loki
graylog
humio
logdna
LOGGLY
LOGIQ
logstash
Pandora
日志易
SCALYR
sematext
splunk>
sumo logic
STRINK
Tracing
JAEGER
CNCF
OPENTRACING
CNCF
elastic apm
honeycomb
Lightstep
OpenTelemetry
PINPOINT
Skywalking
SOFATracer
Spring Cloud Sleuth
ZIPKIN
Chaos Engineering
Chaos Mesh
ChaosToolkit
ChaosBlade
chaoskube
Gremlin
Litmus
steadybit

Q & A