# The Evolution of Distributed Systems on Kubernetes

Bilgin Ibryam
Product Manager @RedHat
@bibryam



Red Hat

# Bilgin Ibryam

**@bibryam**

- Product Manager at Red Hat
- Former Architect/Consultant
- Committer at Apache Camel
- Author of “Camel Design Patterns” and “Kubernetes Patterns” books
- Latest interest: cloud native data

# What comes after Microservices?

# Agenda

- Distributed system needs
- Monolithic architectures
- Cloud-native technologies
  - Kubernetes, Istio, Knative, Dapr
- Future architecture trends

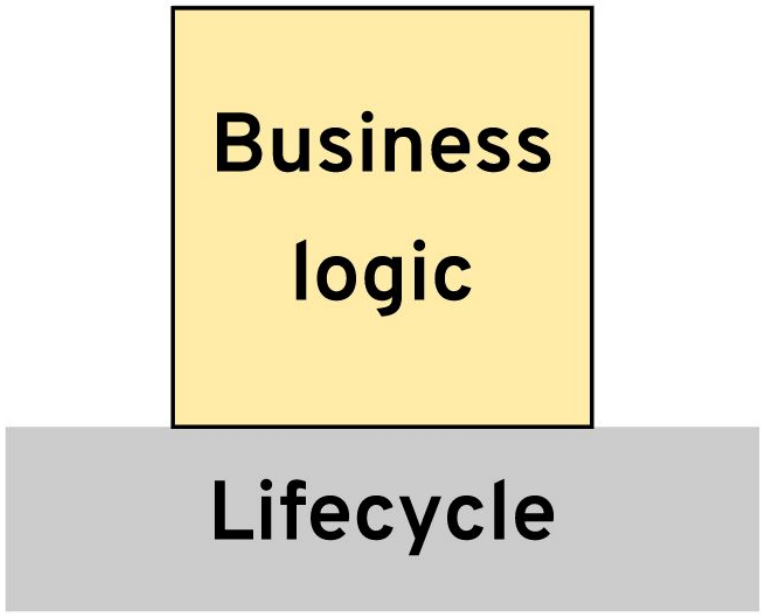
# Modern distributed applications

- 100s of components and 1000s of instances
- Polyglot, independent, and automatable components
- Hybrid workloads on hybrid environments
- Open source, open standards, and interoperable
- Based on Kubernetes ecosystem

# What are the needs of distributed applications?

# Distributed application needs

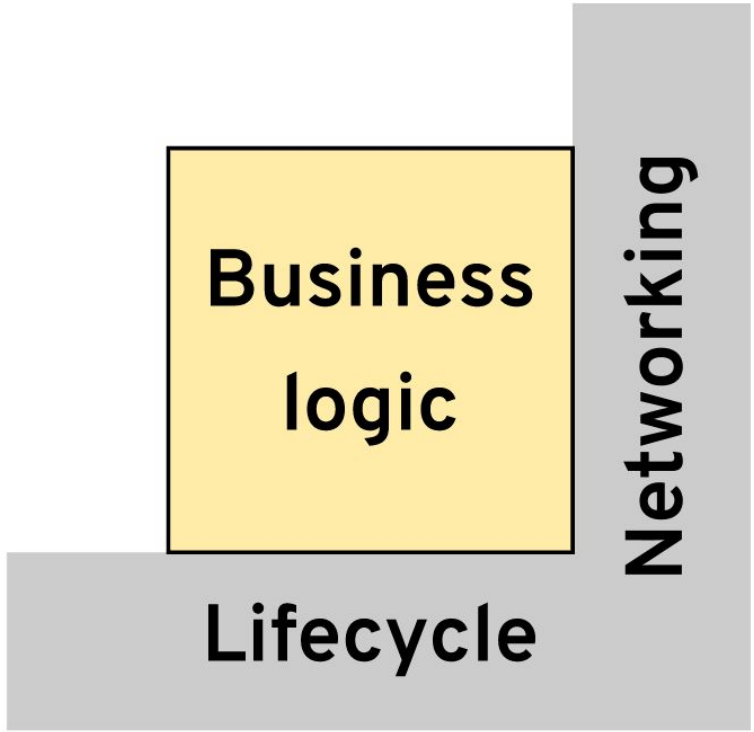
# Distributed application needs




## Lifecycle management

- Deployment/rollback
- Placement/scheduling
- Configuration management
- Resource/failure isolation
- Auto/manual scaling
- Hybrid workloads (stateless, stateful, serverless, etc)

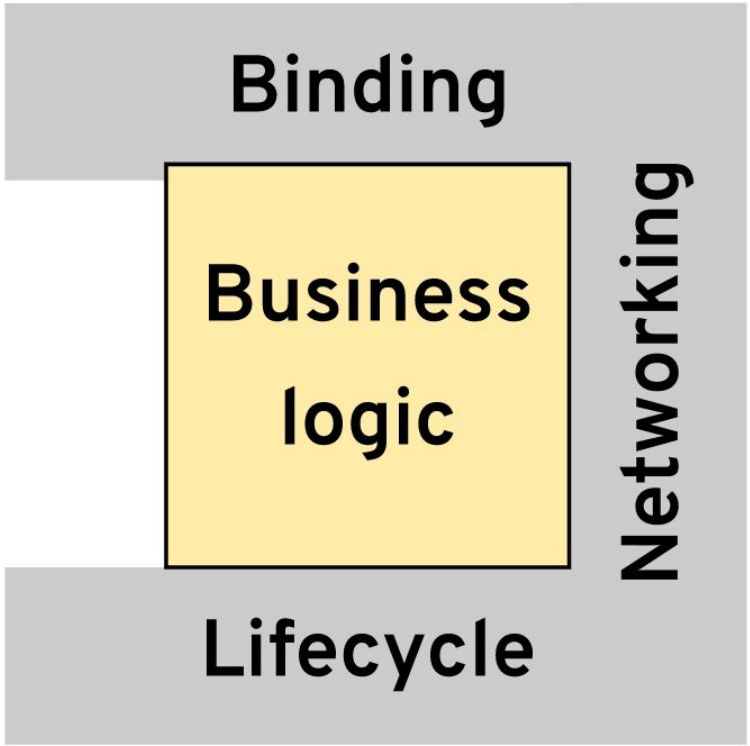
# Distributed application needs




## Advanced networking

- Service discovery and failover
- Dynamic traffic routing
- Retry, timeout, circuit breaking
- Security, rate limiting, encryption
- Observability and tracing

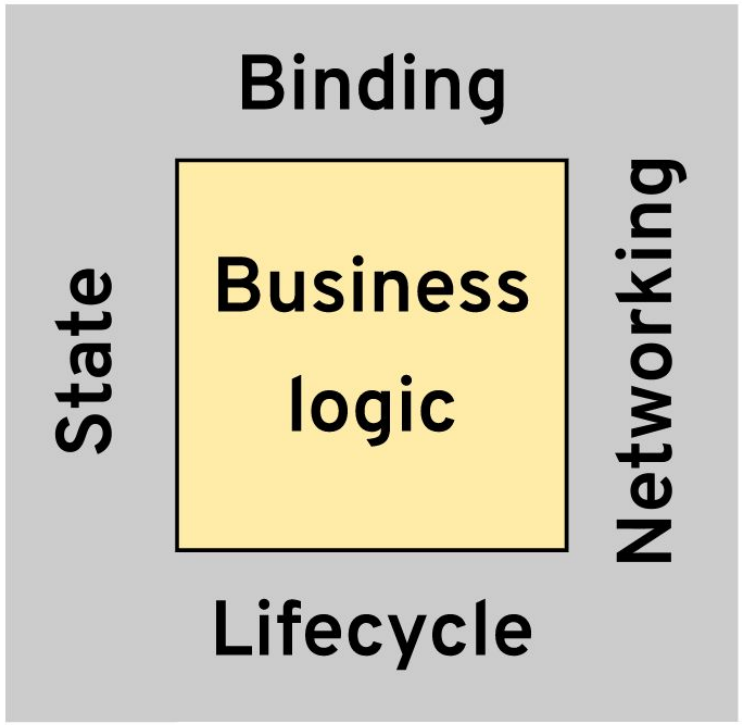
# Distributed application needs




## Resource bindings

- Connectors for APIs
- Protocol conversion
- Message transformation
- Filtering, light message routing
- Point-to-point, pub/sub interactions

# Distributed application needs

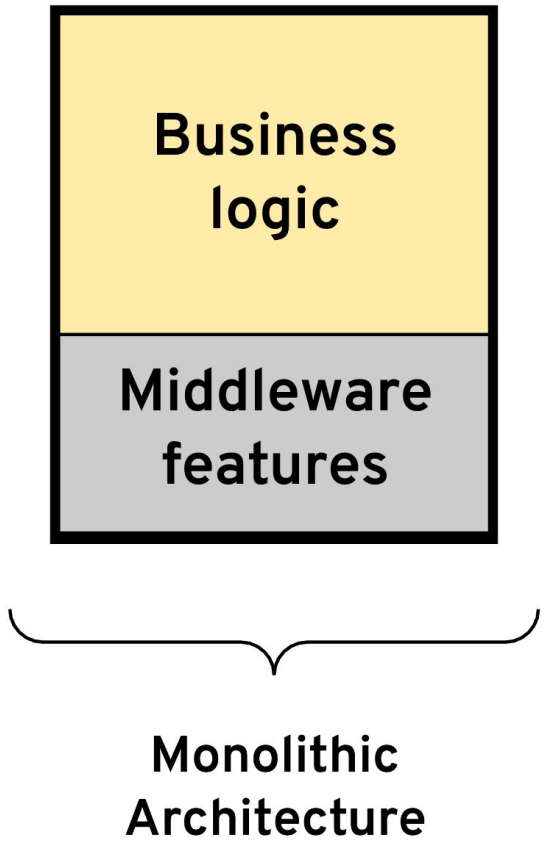



## Stateful abstractions

- Workflow management
- Temporal scheduling
- Distributed caching
- Idempotency
- Transactionality (SAGA)
- Application state

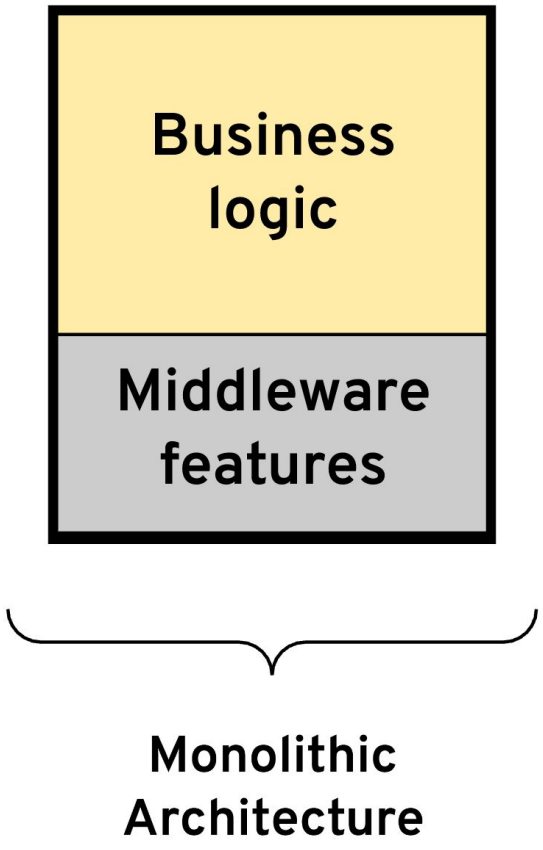
# Monolithic architectures

# Traditional middleware capabilities




- Stateful primitives ✔
- Resource bindings ✔
- Networking ✔

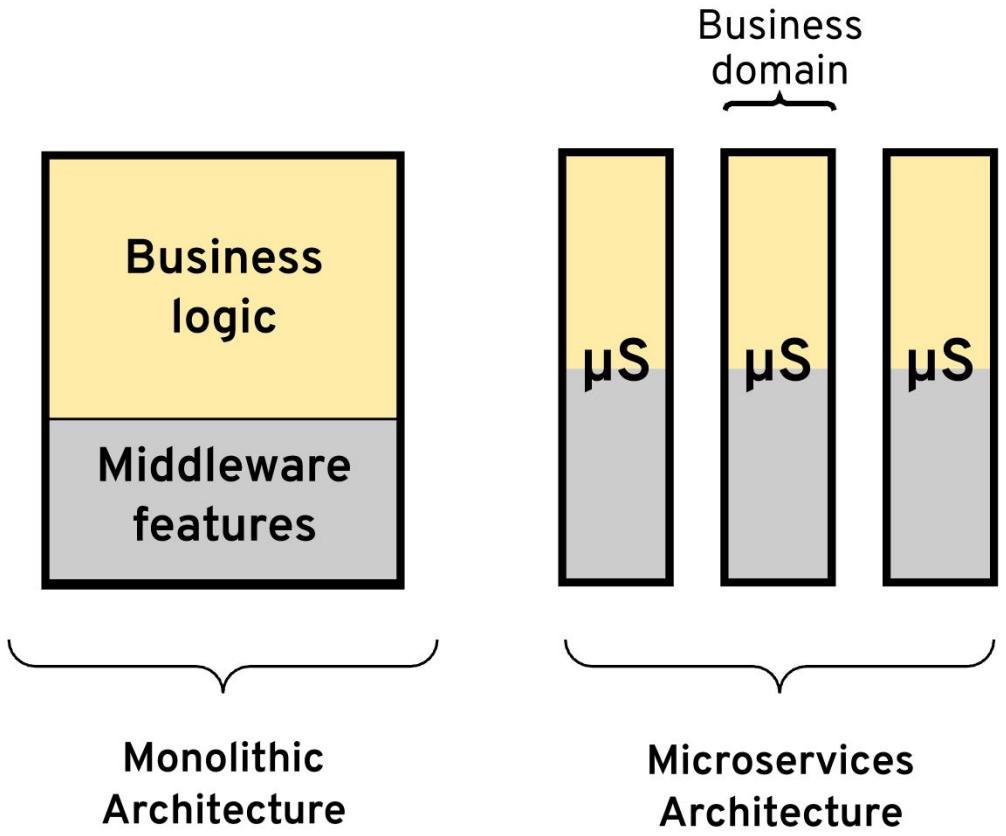
# Traditional middleware limitations



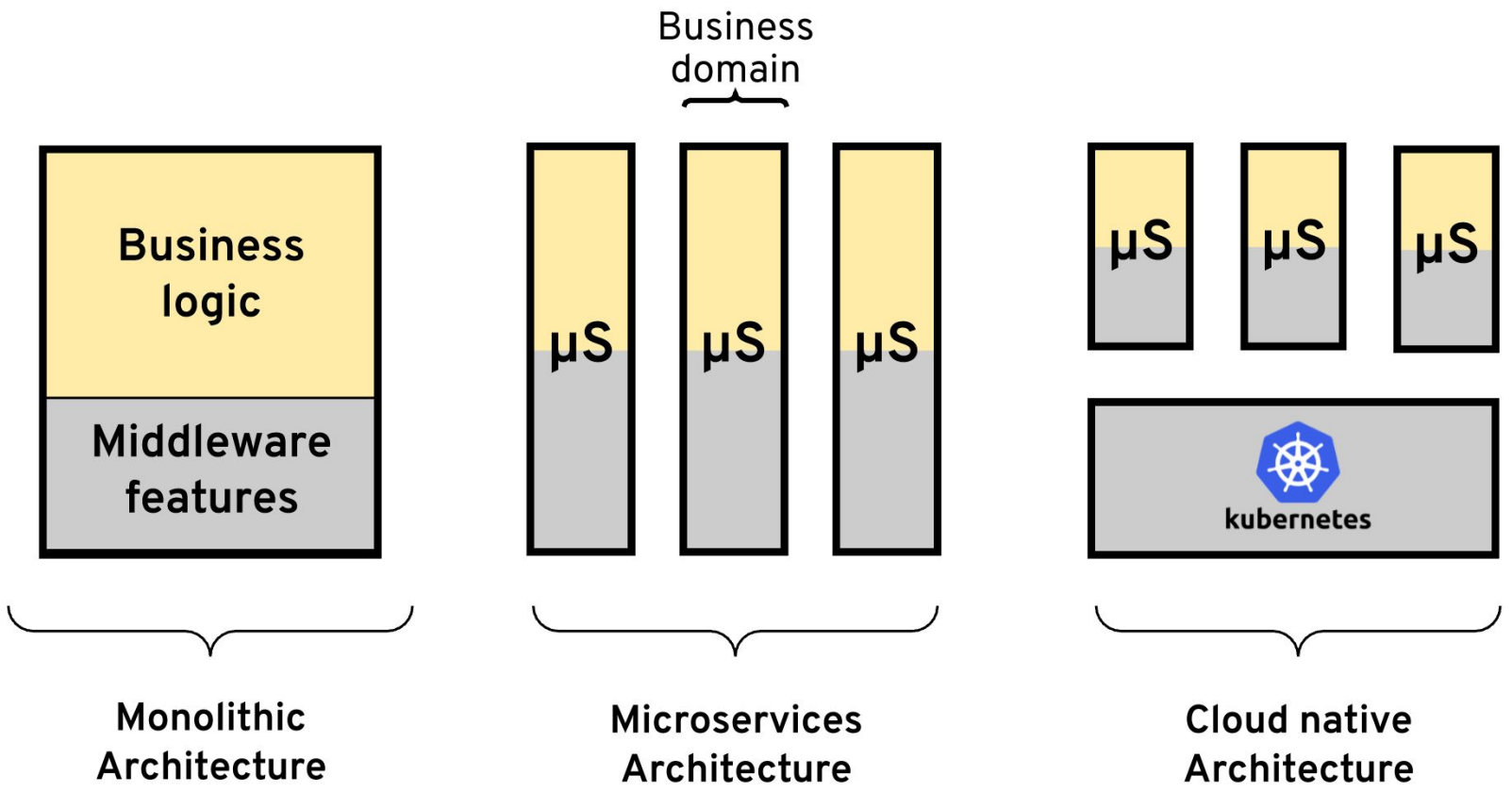

- Lifecycle management
  - Single, shared language runtime
  - Manual deployment/rollback
  - Manual placement
  - Manual scaling
  - No resource/failure isolation

# Cloud-native architectures

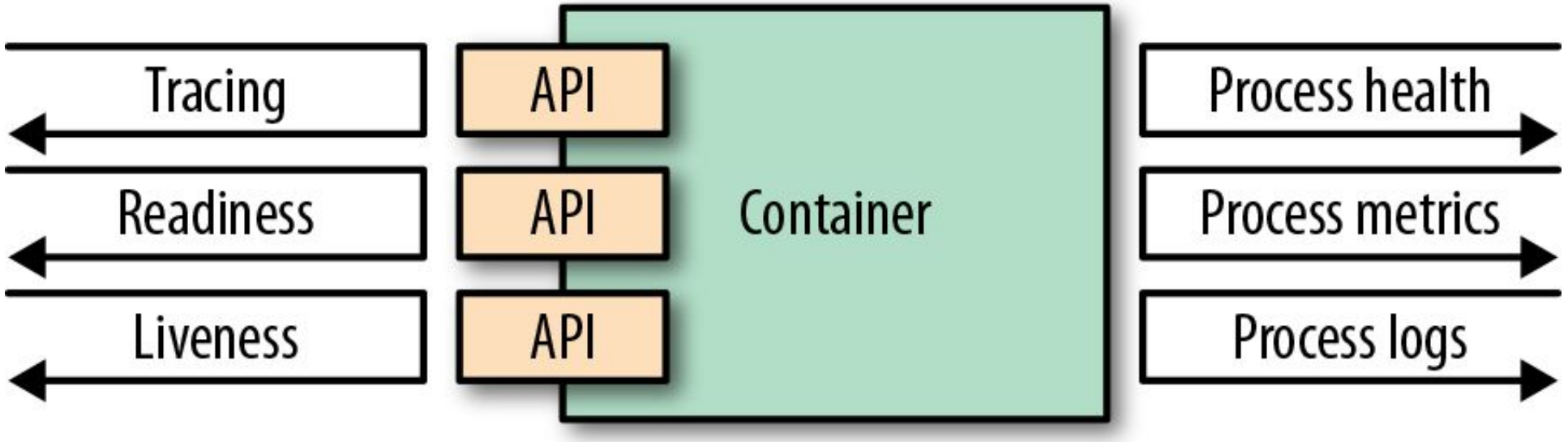
# Microservices and Kubernetes



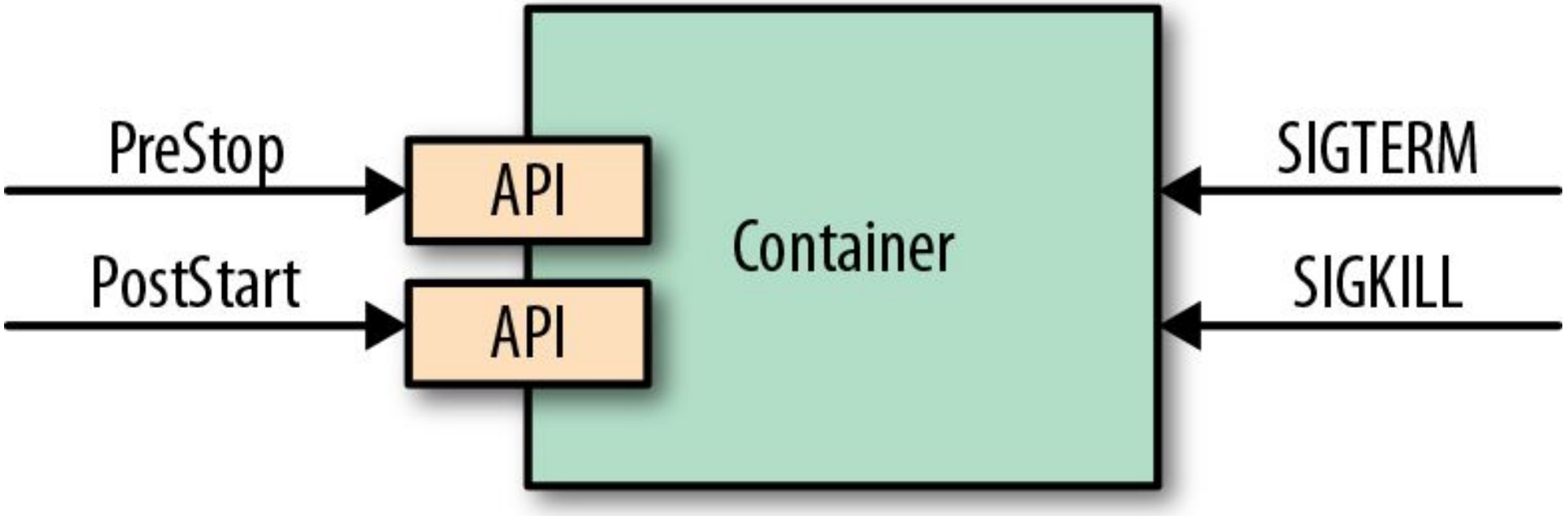
# Microservices and Kubernetes



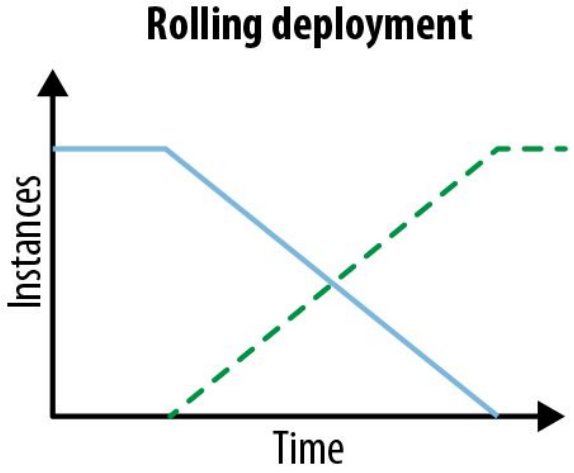
# Health probes

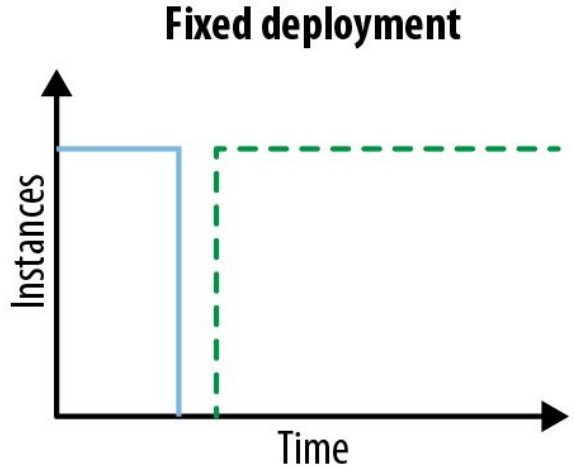


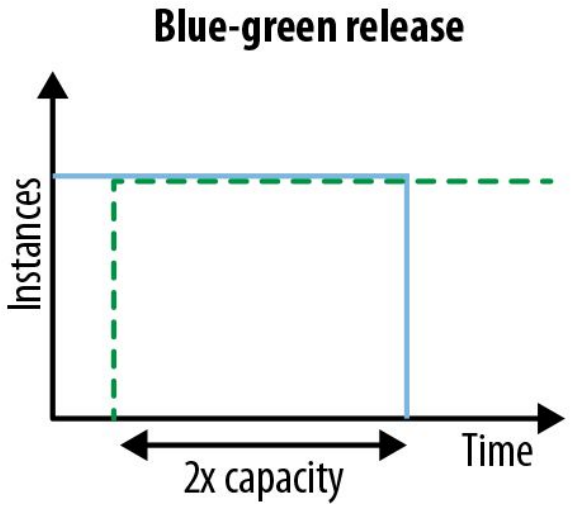
# Managed start/stop

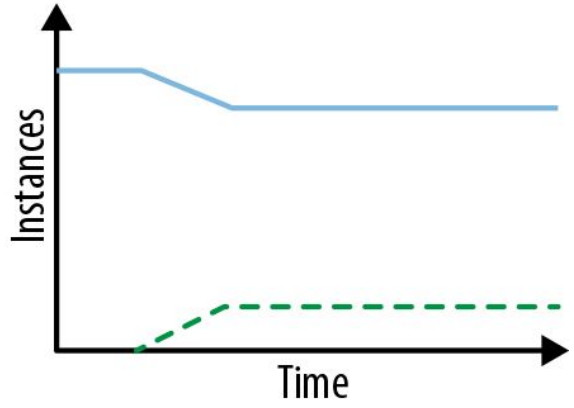


# Declarative deployment

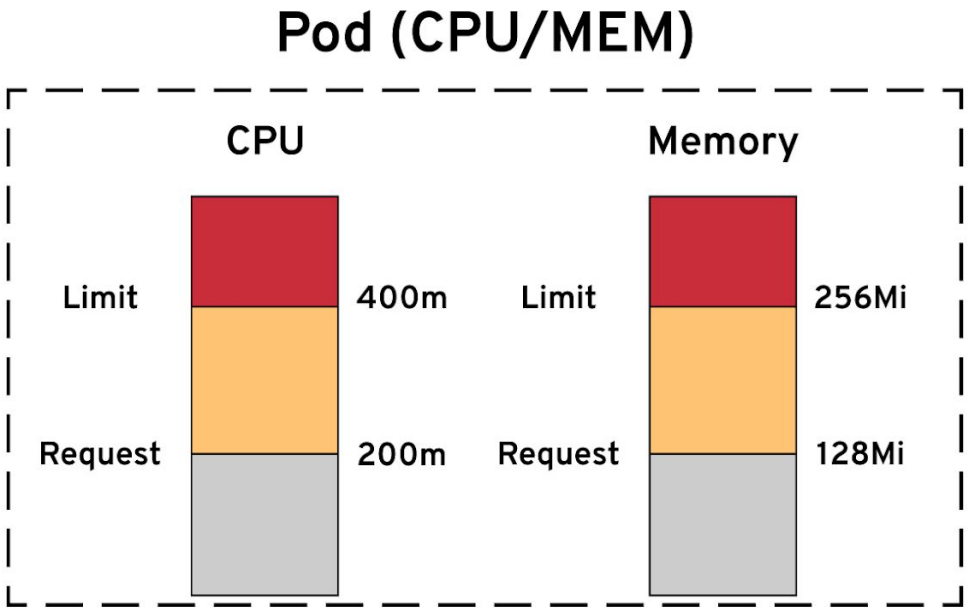


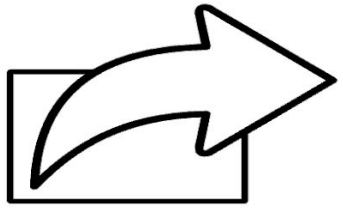



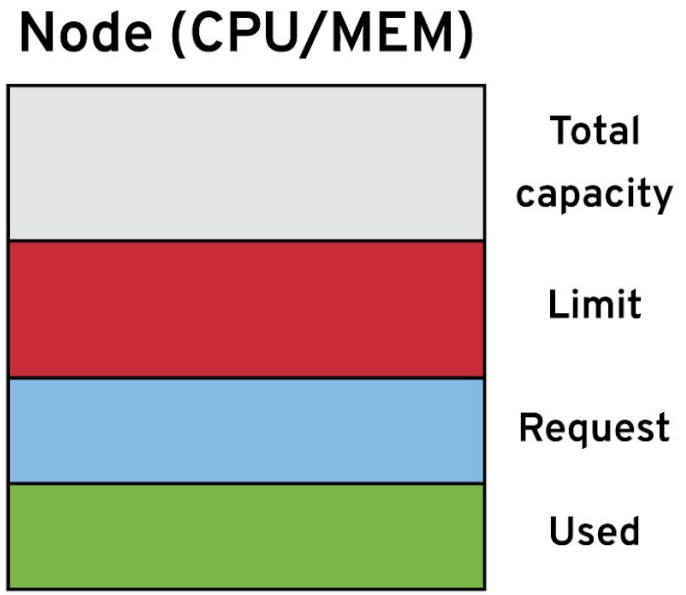



# Demands & placement



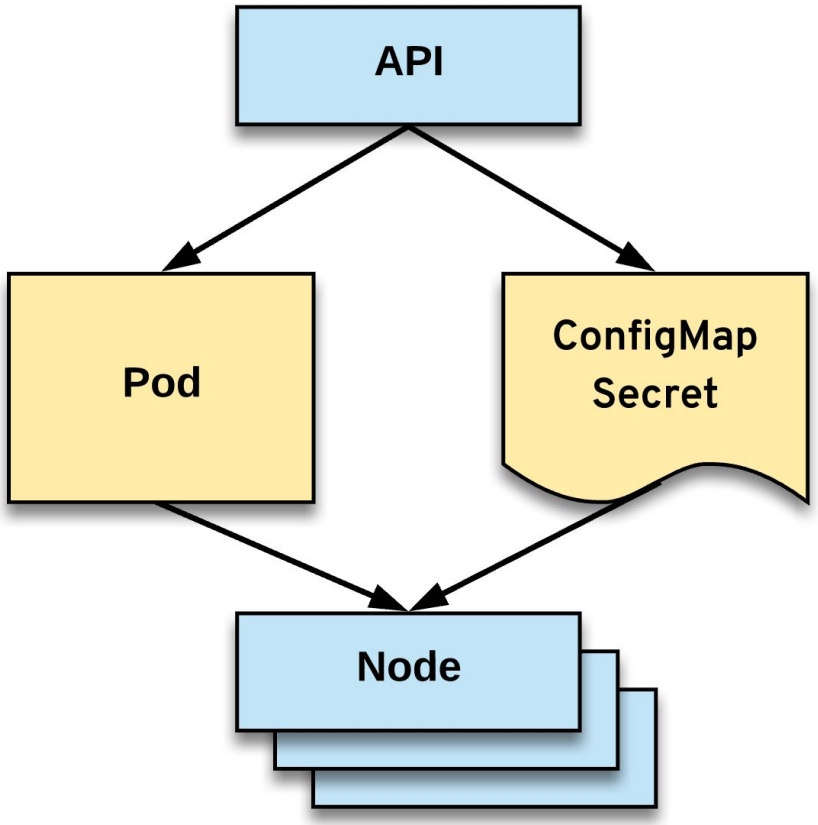






Predictable resource demand

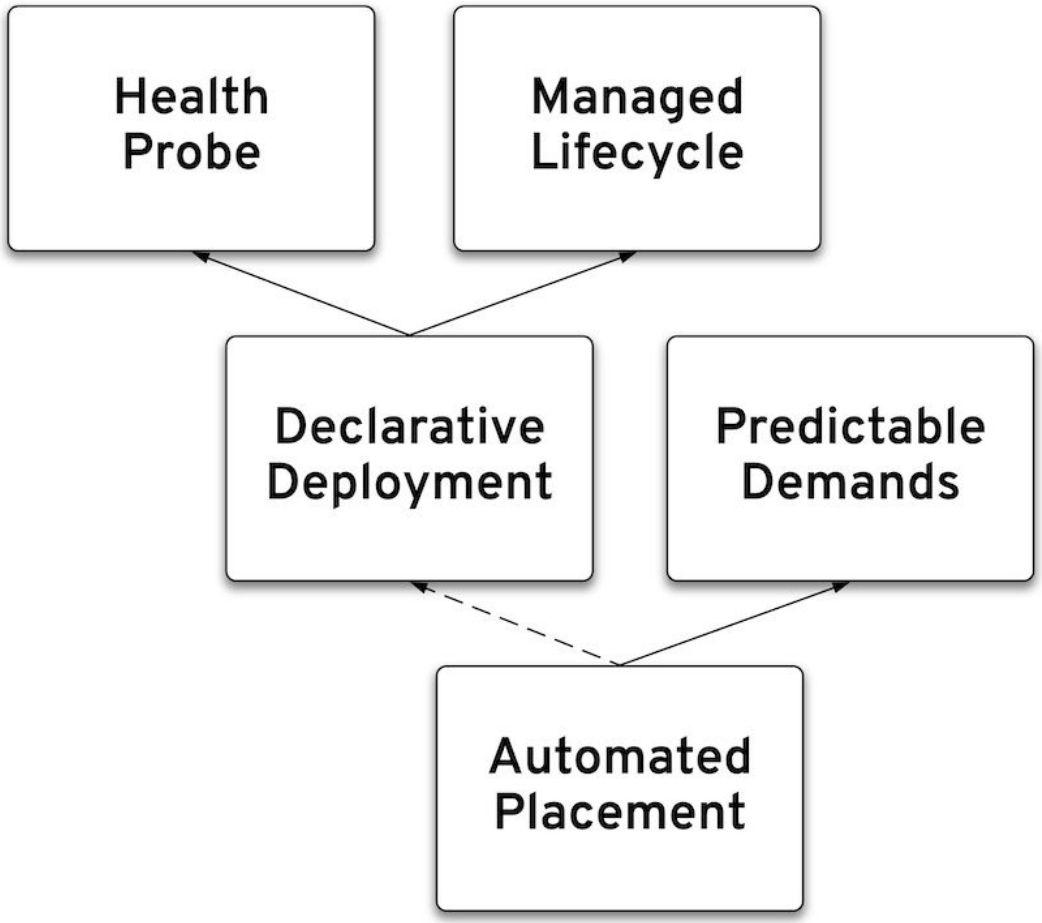
Automated placement

# Configuration management




- ConfigMaps used in Pods as:
  - environment variables
  - volumes
- Secrets:
  - Minimal Node spread
  - Only stored in memory in a tmpfs
  - Encrypted in the backend store (etcd)
  - Access can be restricted with RBAC

# Foundational kubernetes capabilities

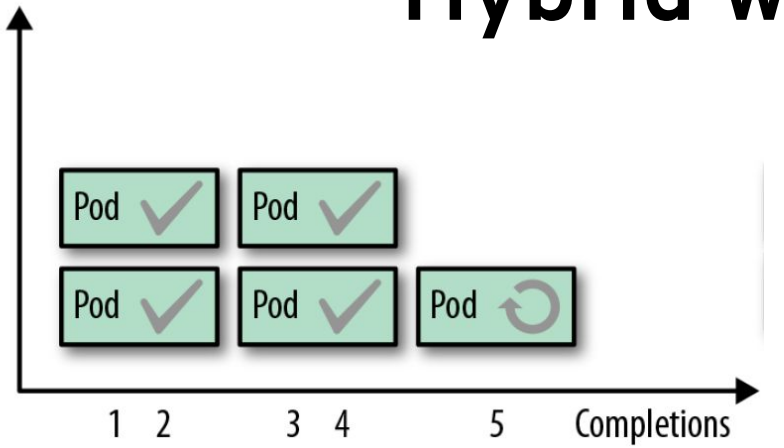



## More Kubernetes Patterns

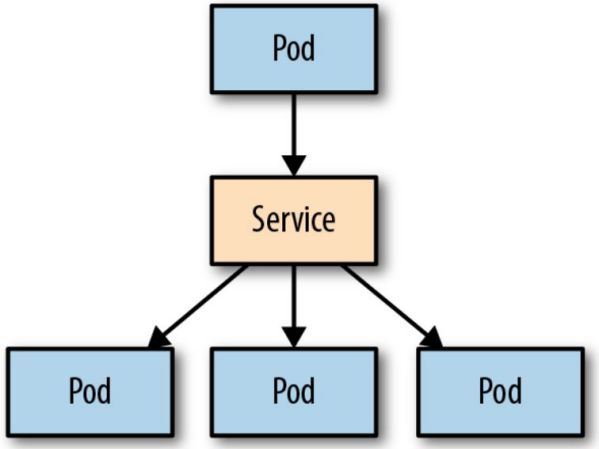
- Foundational patterns
- Structural patterns
- Configuration patterns
- Behavioural patterns

*(For more Kubernetes Patterns, check out the link at the end of the slides)*

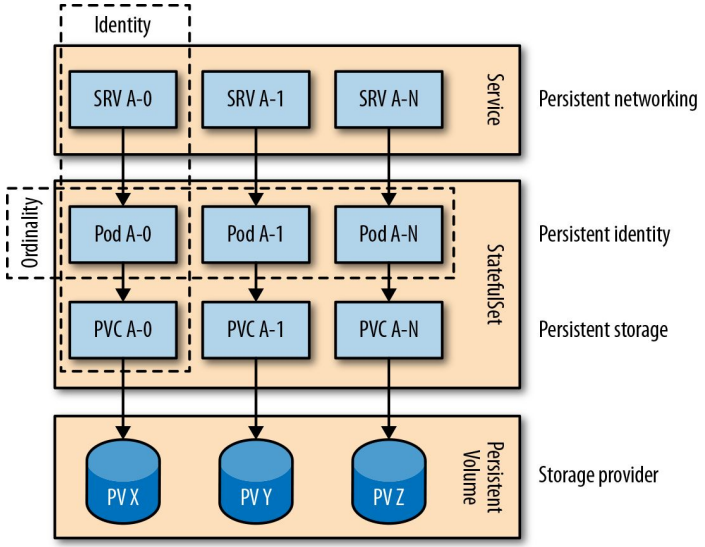
# Hybrid workloads



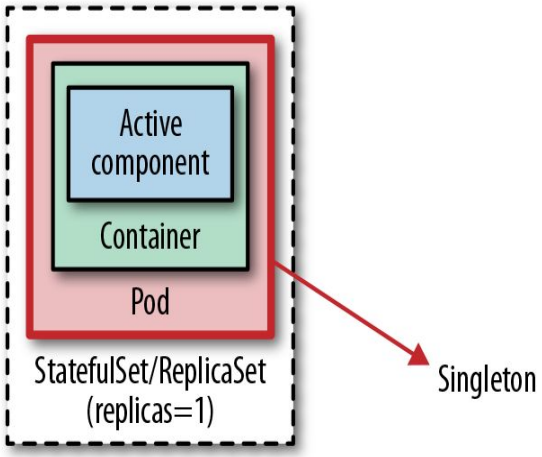

Batch/Periodic Job




Stateless Service

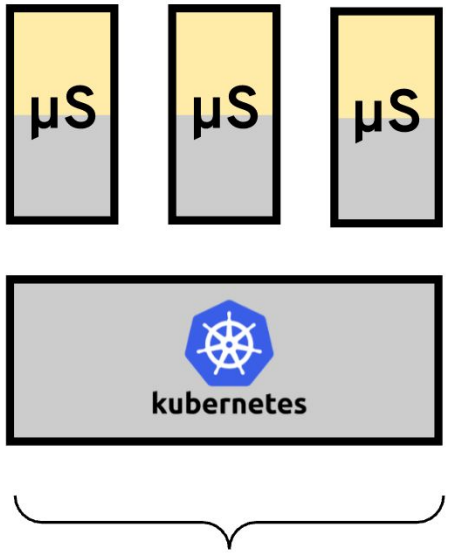



Stateful Service




Global Singleton

# Lifecycle capabilities

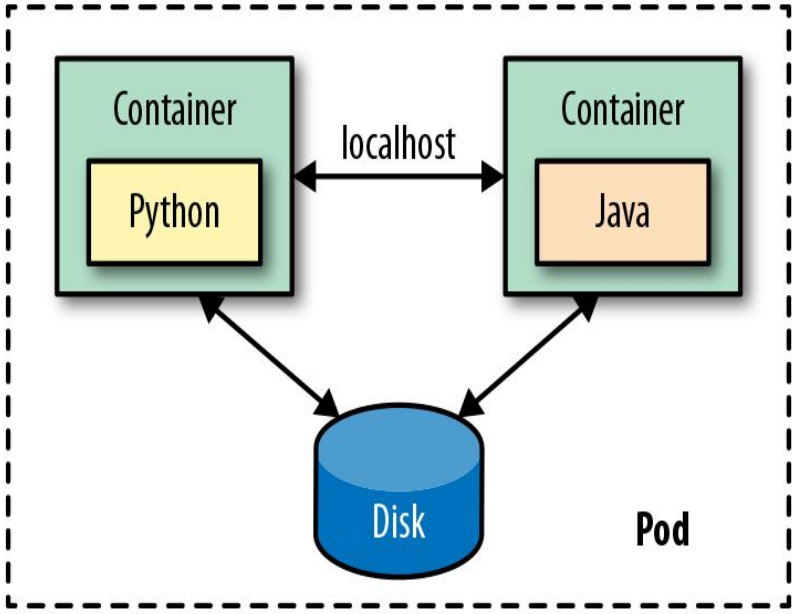



- Deployment/rollback ✔
- Placement/scheduling ✔
- Configuration management ✔
- Resource/failure isolation ✔
- Auto/manual scaling ✔
- Hybrid workloads: stateless, stateful, batch jobs, ~~serverless~~ ✔

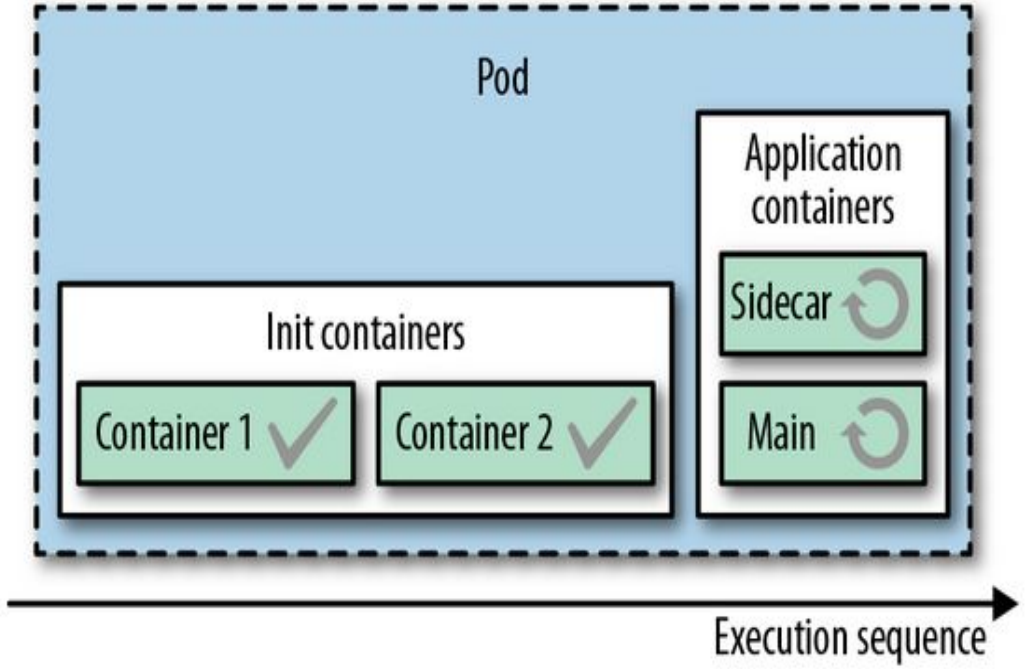
# How do we extend Kubernetes?

# Out-of-process extension mechanism

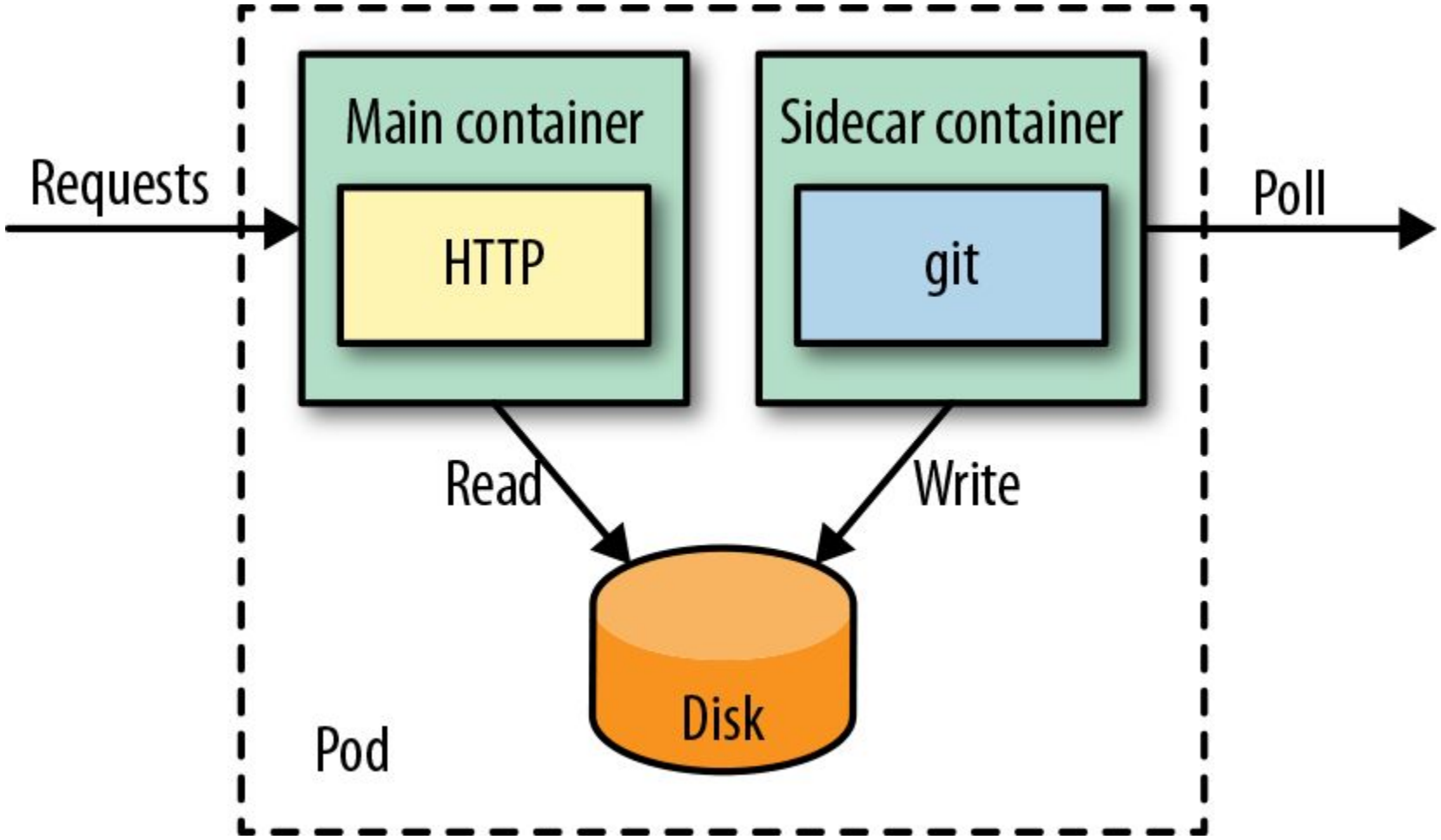
Deployment guarantees



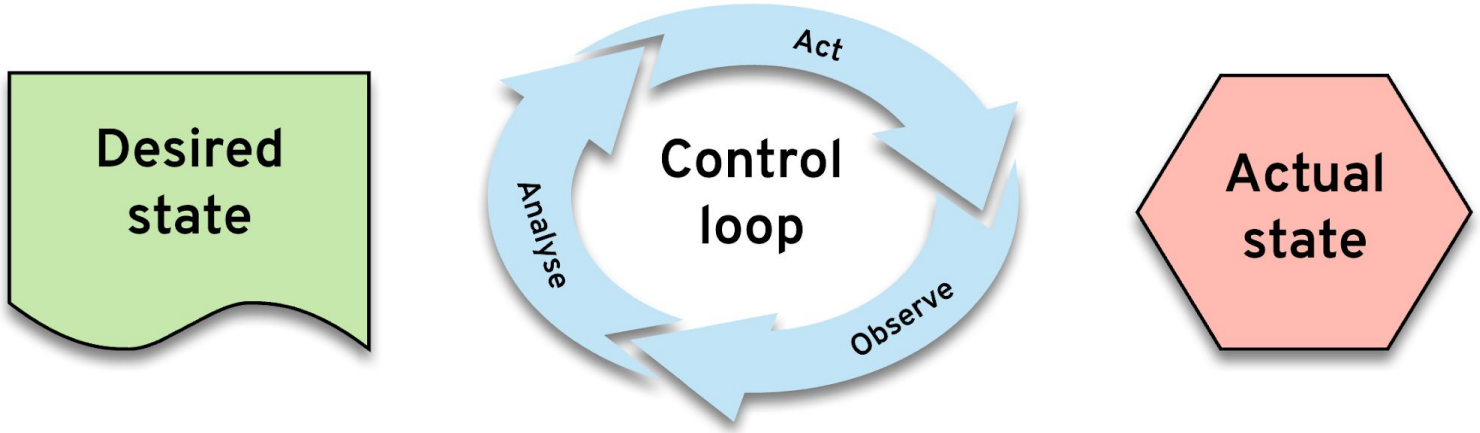

Lifecycle guarantees



# Sidecar



# Controller Pattern




**Default schema**

- ReplicaSet
- StatefulSet
- Job, CronJob

**Default controllers**

- replicaset
- statefulset
- job, cronjob

**Managed resources state**

- Pod
- PVC...

**Custom controller -> Custom behaviour**

# Operator Pattern

```
kind: ConfigWatcher
apiVersion: k8spatterns.io/v1
metadata:
  name: webapp-config-watcher
spec:
  configMap: webapp-config
  podSelector:
    app: webapp
```

**Custom operator**

- Go
- Helm
- Ansible
- Java
- Python

**Custom application**

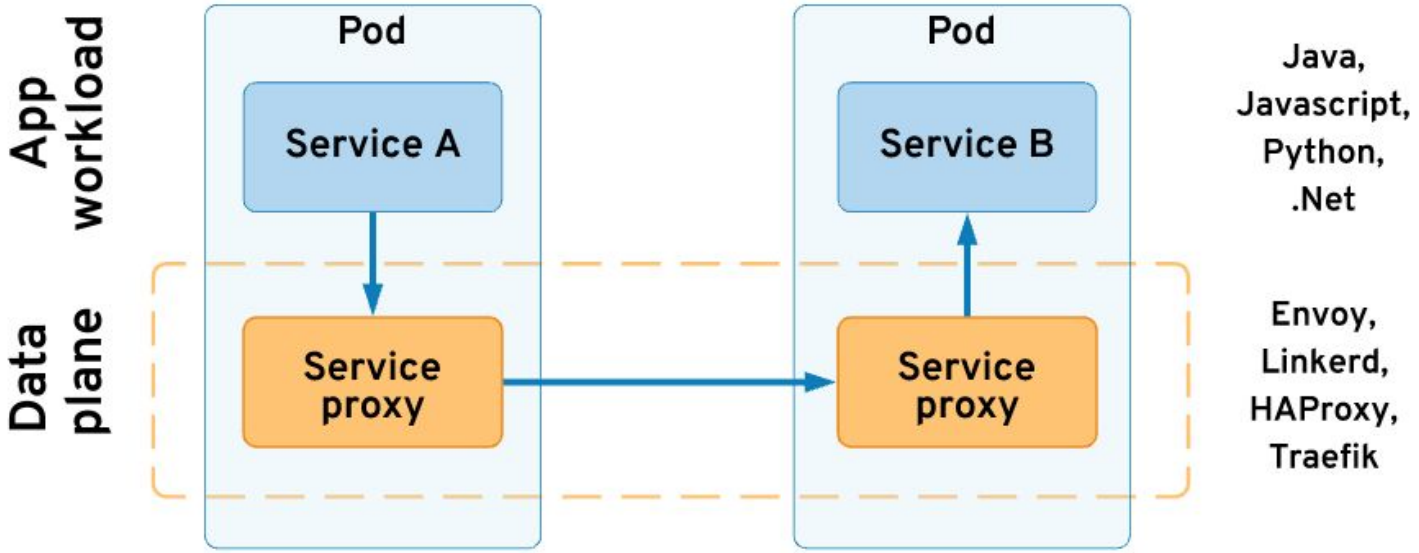
- AI/ML
- Big Data
- Storage
- Streaming
- Monitoring

**CustomResourceDefinition + Controller = Operator**

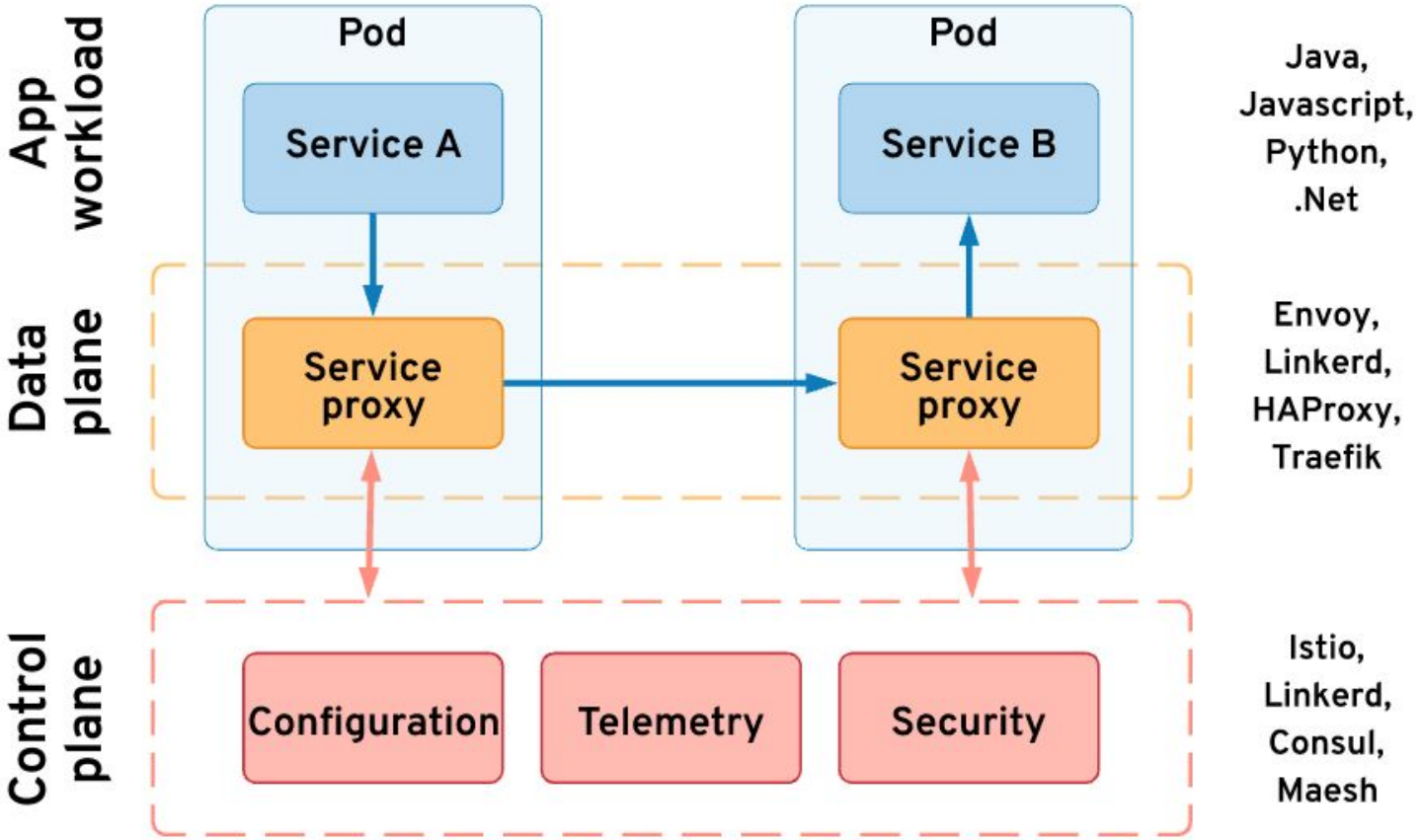
# Kubernetes based platforms

# What is Service Mesh?

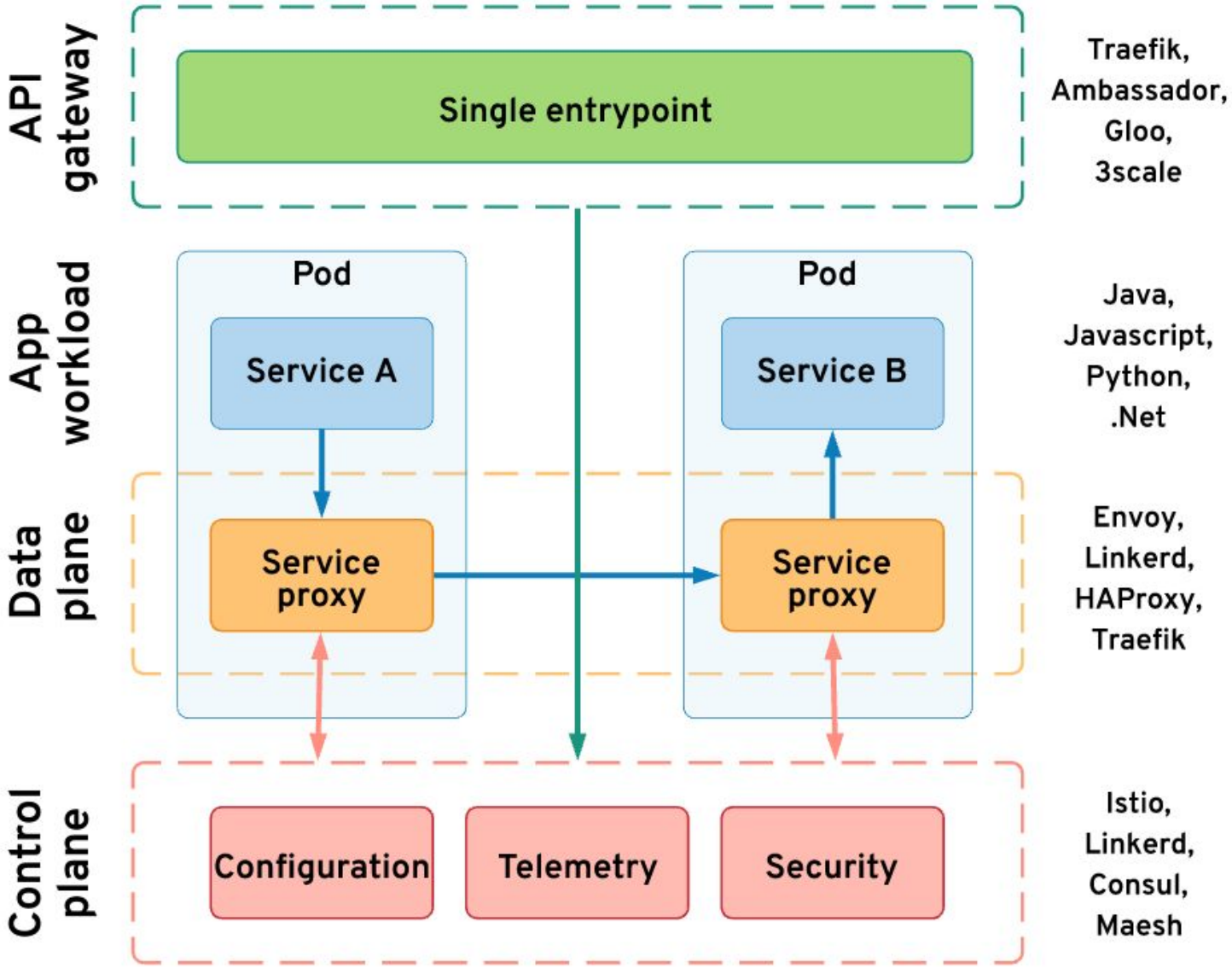
# What is Service Mesh?



# What is Service Mesh?



# What is Service Mesh?



# Networking capabilities

## API Gateway

**Abstract away details and decouple consumers from implementations**

- Controls what's allowed in/out
- Bridging security domains
- Request / response transformation
- Protocol, data format transformation
- API composition
- Rate limiting

## Service Mesh

**Enhances the reliability and the visibility of the networking interactions**

- Telemetry, tracing collection
- Service discovery, load balancing
- TLS termination/origination
- Request routing, traffic splitting
- Traffic shadowing
- Rate limiting

# What is Knative?


| Serving | Eventing |
| --- | --- |
| Common infrastructure for request-driven interactions that can "scale to zero". | Common infrastructure for consuming and producing events declaratively. |

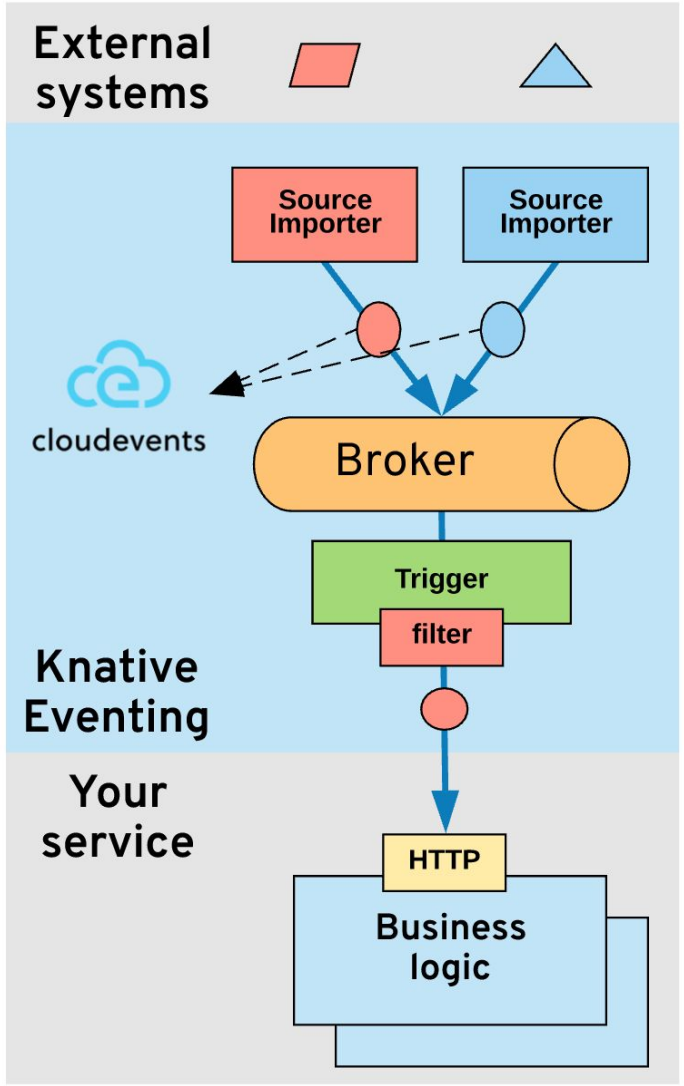
Kubernetes-based platform to deploy, and manage serverless workloads.

# Knative Serving concepts

```
apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
  name: lotto
spec:
  replicas: 1
  selector:
    matchLabels:
       app: lotto
  template:
    metadata:
      labels:
        app: lotto
  spec:
    containers:
    - image: cds19/lotto
```

- Scale-to-zero & activation
- Rapid autoscaling
- Traffic splitting
- Callable by Knative eventing
- Simplified deployment model
  - Single Port
  - No PersistentVolumes
  - Single Container

# Knative Eventing concepts




- Sources (Kafka, CronJob,  Apache Camel 200+, etc)
- Broker implementations (In-memory, Kafka, etc)
- CloudEvents data format
- Trigger with filters
- Sequence: chaining multiple steps composed of containers

# Lifecycle, networking, binding capabilities

- Knative Serving
  - Simplified deployment for stateless workloads
  - Traffic based autoscaling including Scale-to-Zero
  - Traffic splitting for custom rollout / rollback scenarios
- Knative Eventing
  - External triggers for feeding Knative Services
  - Based on CloudEvents
  - Backed by proven messaging systems
  - Declarative messaging infrastructure

# What is Dapr?


**Sidecar architecture**

Developer first, standard APIs used from any programming language or framework.

**Building blocks**

Make it easy for developers to create microservice without being an expert in distributed systems.

A portable runtime for building distributed applications.

# Dapr building blocks

**Service Invocation**

Act as a reverse proxy with built-in service discovery, tracing and error handling

**Resource Bindings**

Trigger code through events from input and output bindings to external resources.

**State Management**

Provides a key/value-based state API with pluggable state stores for persistence

**Distributed Tracing**

See and measure the message calls across components and networked services

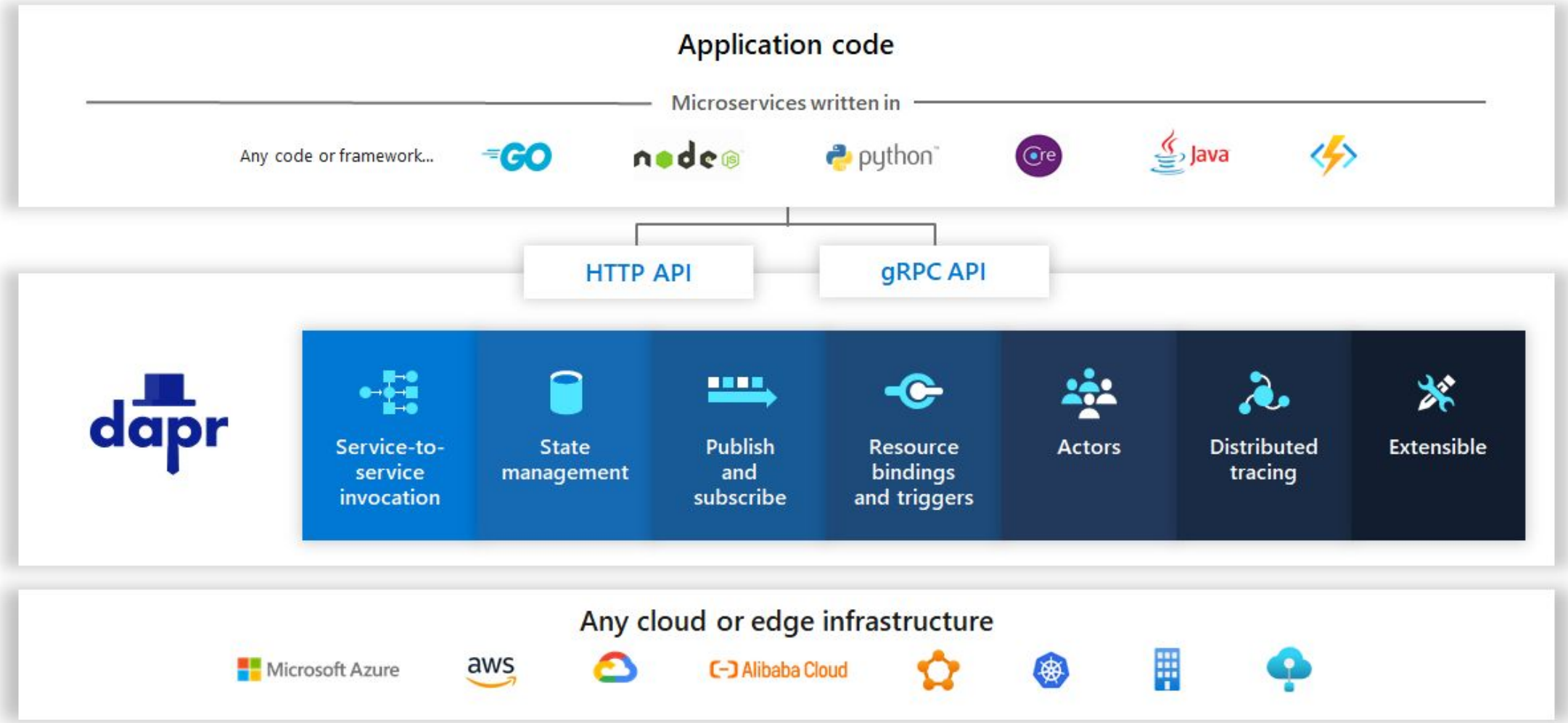
**Publish & Subscribe**

Secure, scalable messaging between services

**Actors**

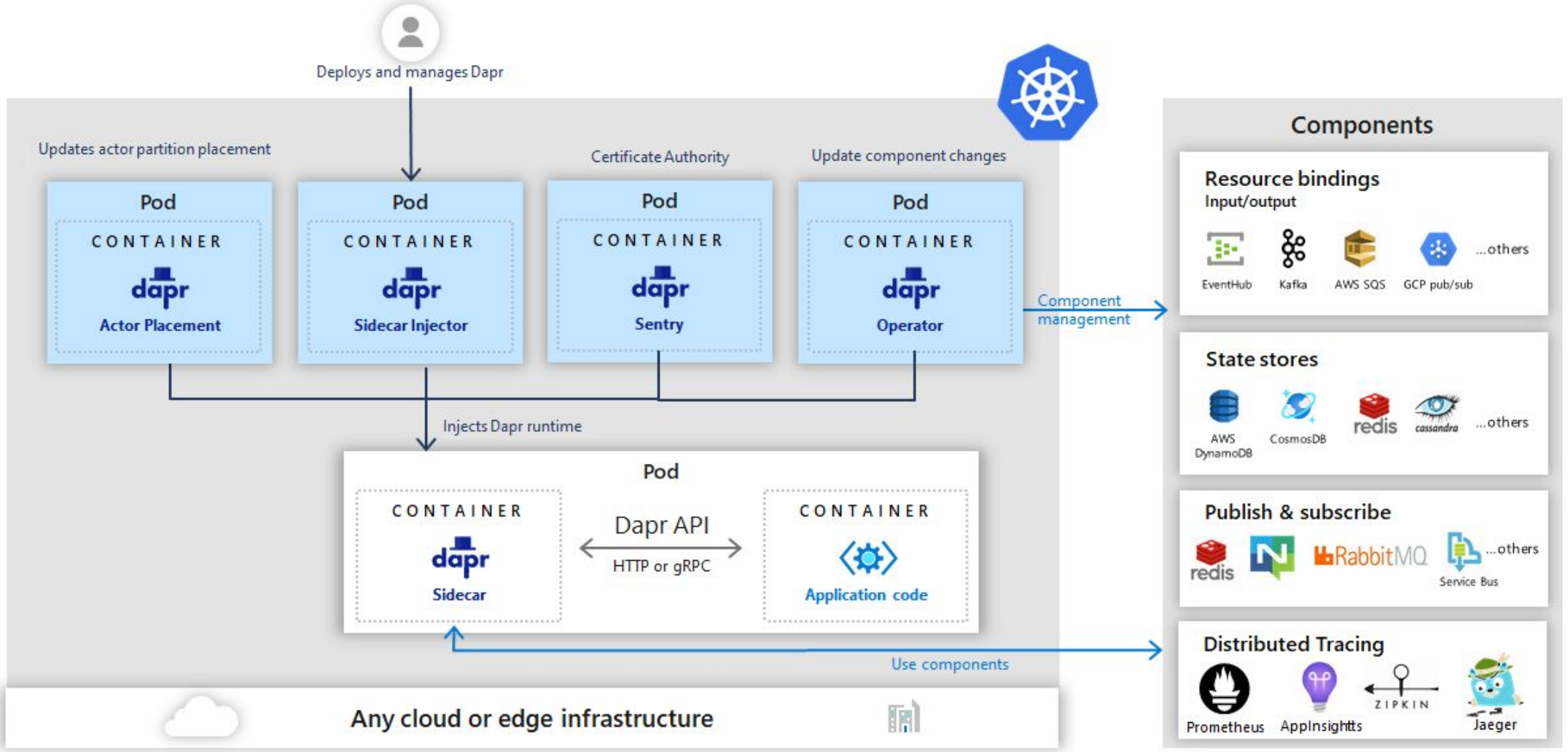
Encapsulate code and data in reusable actor objects as a common microservices

# Dapr architecture



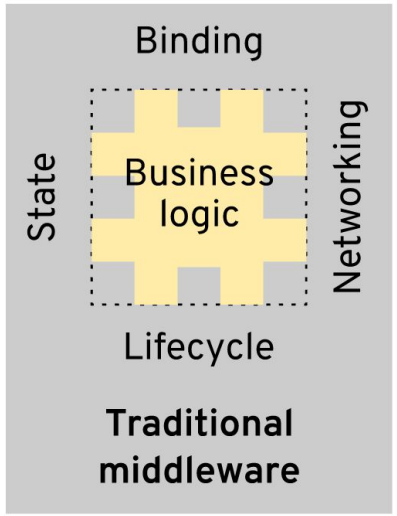

Source: https://github.com/dapr/docs

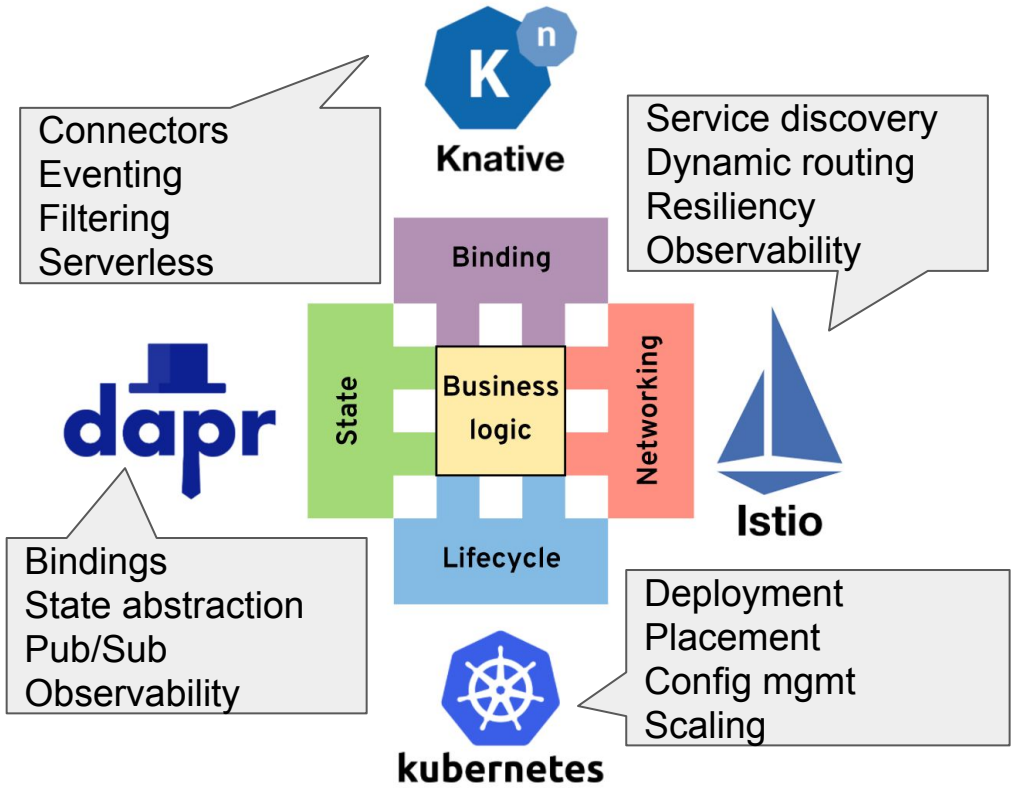
# Dapr on Kubernetes




Source: https://github.com/dapr/docs

# Full circle




- Centralized control plane
- Centralized data plane

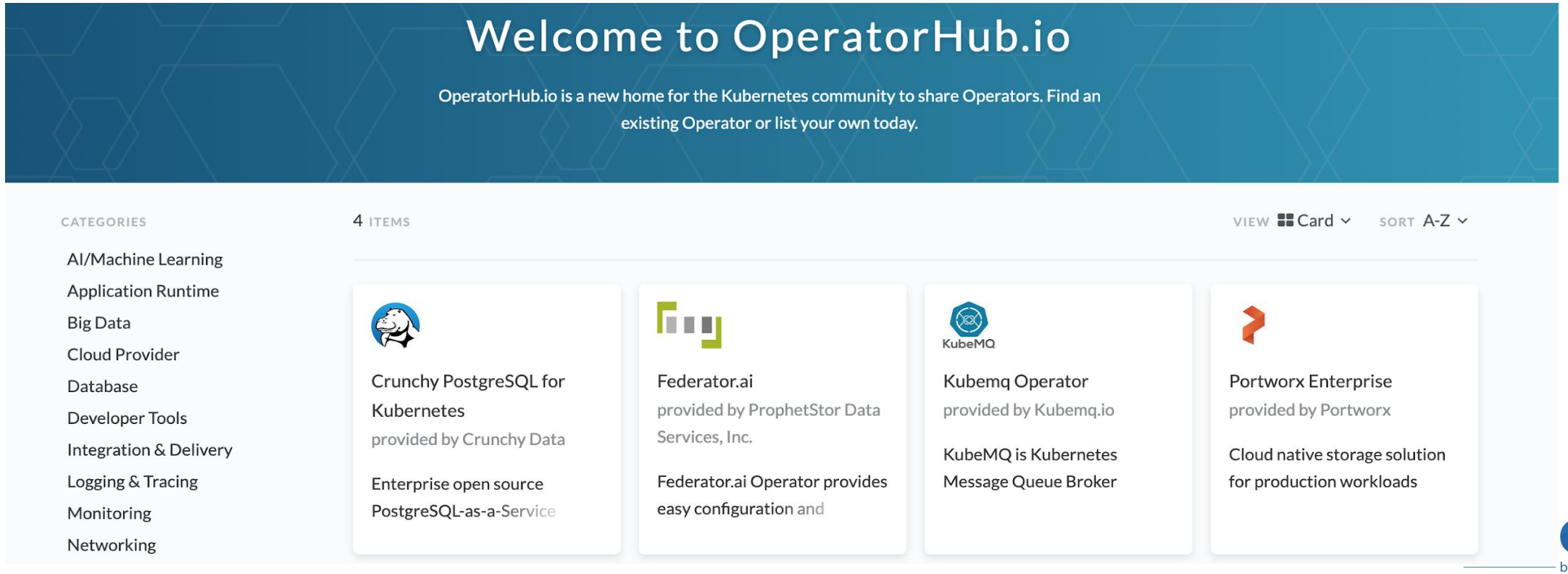
- Centralized control plane
- Decentralized, highly-scalable data plane

# Future cloud native trends

# Lifecycle trends

OPERATOR FRAMEWORK

| Phase I | Phase II | Phase III | Phase IV | Phase V |
|---|---|---|---|---|
| **Basic Install** | **Seamless Upgrades** | **Full Lifecycle** | **Deep Insights** | **Auto Pilot** |
| Automated application provisioning and configuration management | Patch and minor version upgrades supported | App lifecycle, storage lifecycle (backup, failure recovery) | Metrics, alerts, log processing and workload analysis | Horizontal/vertical scaling, auto config tuning, abnormal detection, scheduling tuning |

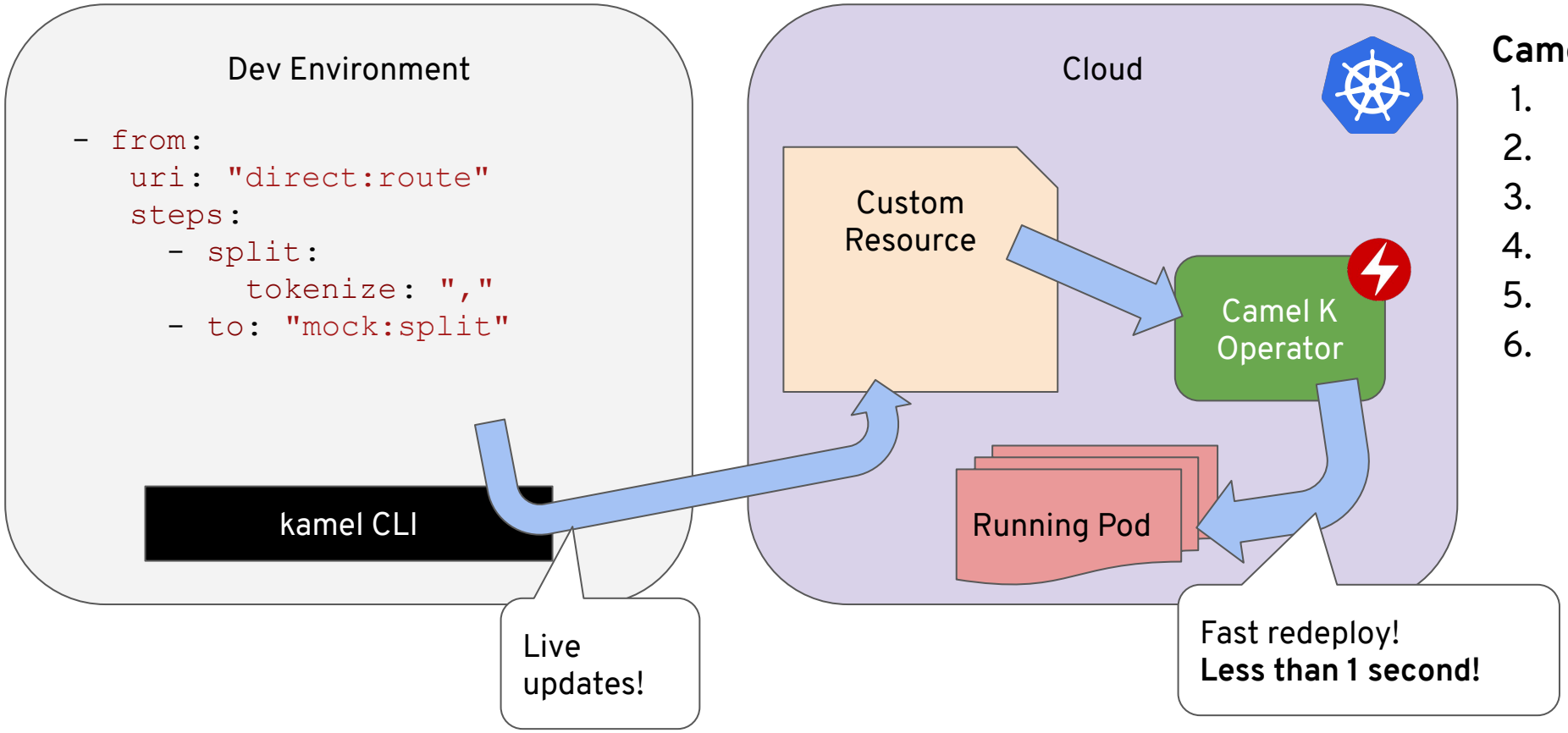



Source: https://operatorhub.io

# Networking trends


- Introduction of Service Mesh Interface specification
- Architecture consolidation of Istio with istiod
- More L7 protocols: MongoDB, DynamoDB, ZooKeeper, MySQL, Redis, Kafka(8188)
  - KIP-559 can enable bridging, validation, encryption, filtering, transformation
- HTTP Cache filter (eCache)
- HTTP tap filter (with matcher)
- WebAssembly (wasm) filters with dynamic loading (C++ -> Rust, Go, etc)

# Binding trends

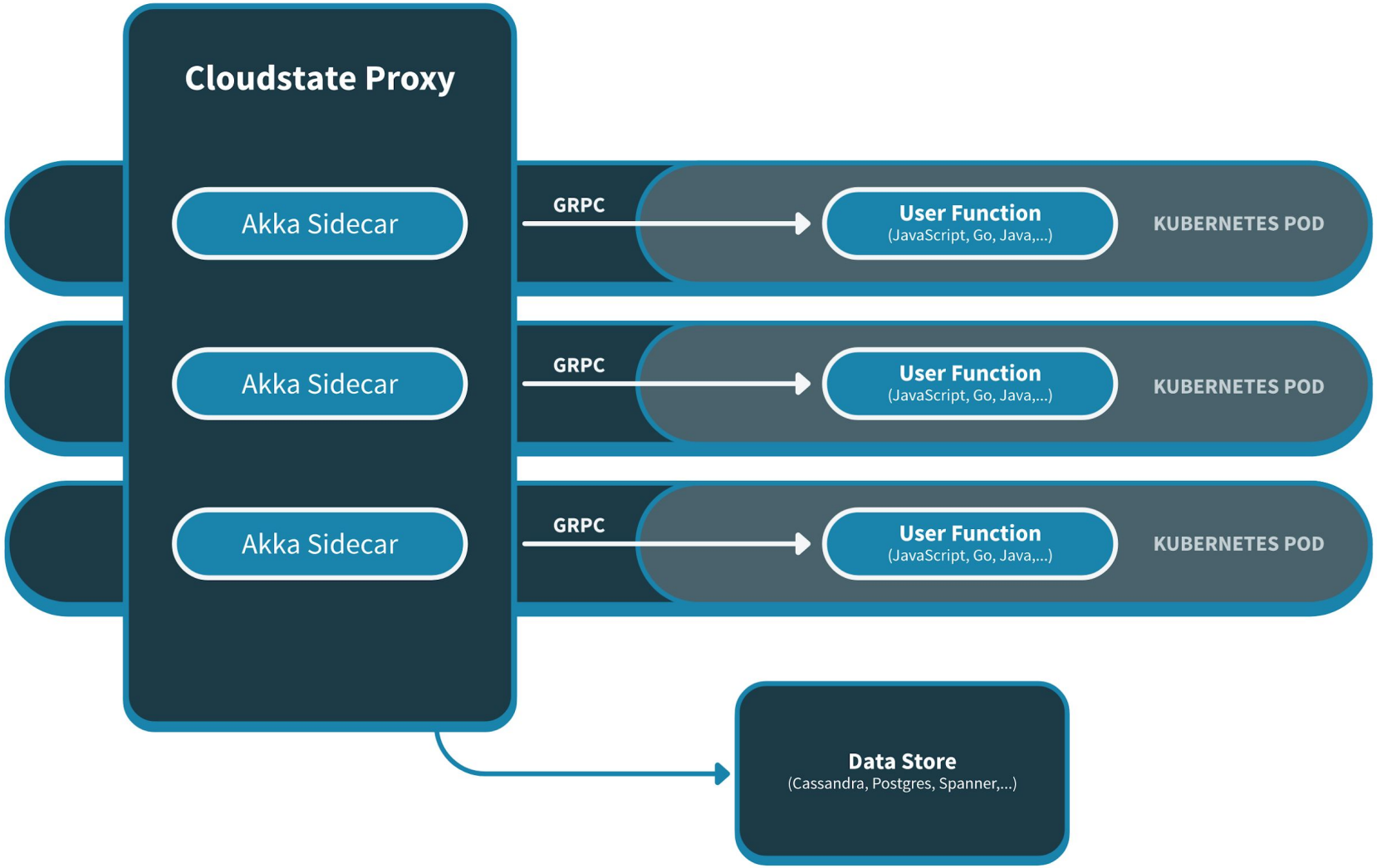



**Camel-K Operator:**

1. Choose a runtime
2. Scaffold a project
3. Add boilerplate
4. Add dependencies
5. Create container image
6. Create Kubernetes resources for deployment

Source: https://github.com/apache/camel-k

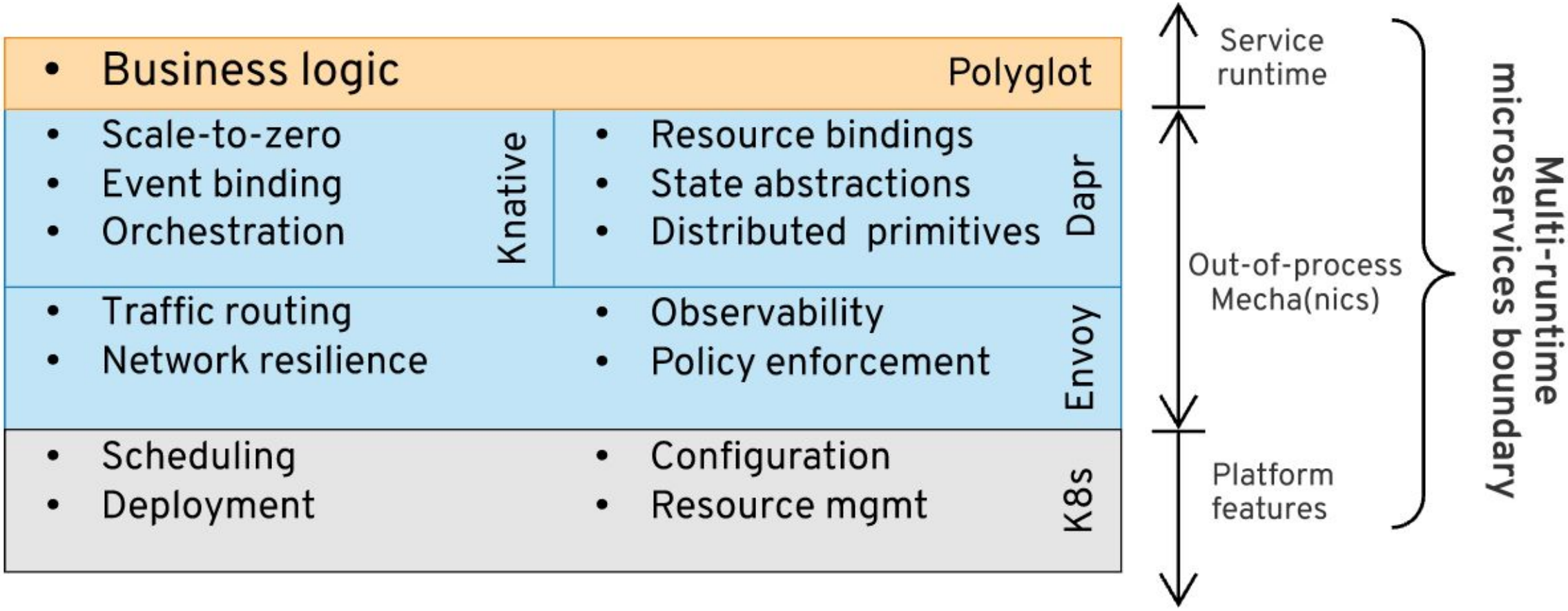
# State trends



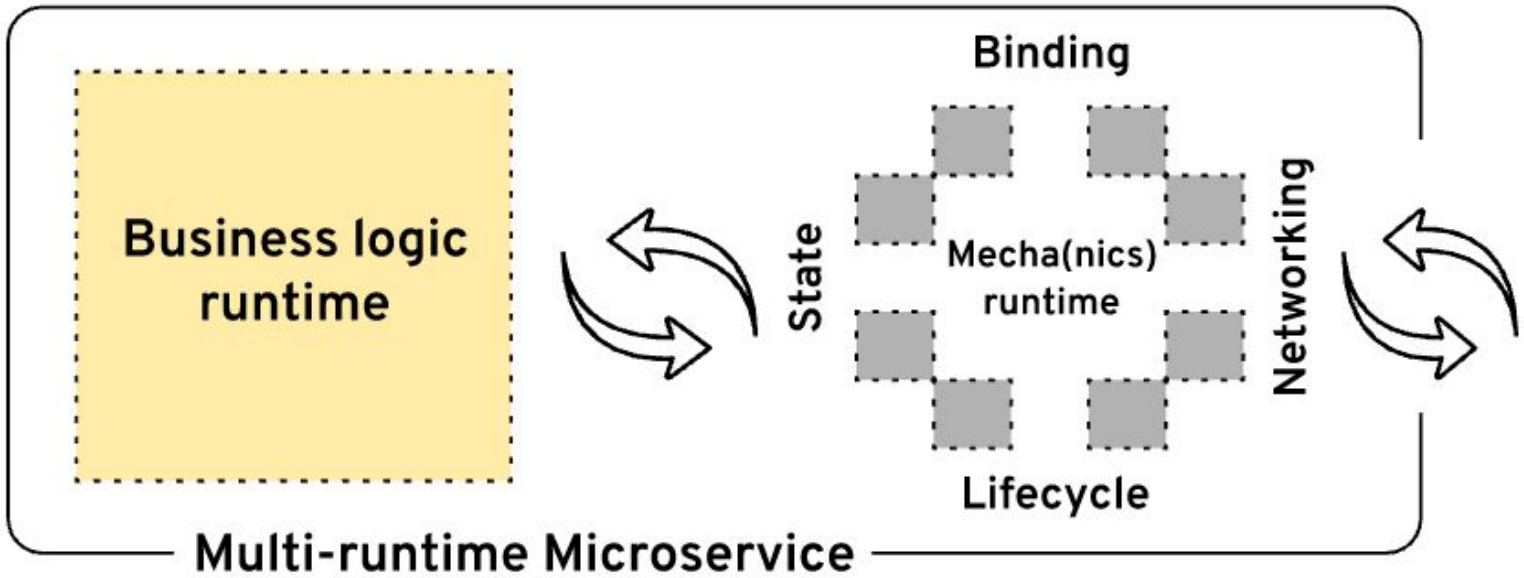

Source: https://github.com/cloudstateio/cloudstate

# What does all this mean?

# Multi-runtime microservices are here



# Smart sidecars and dumb pipes



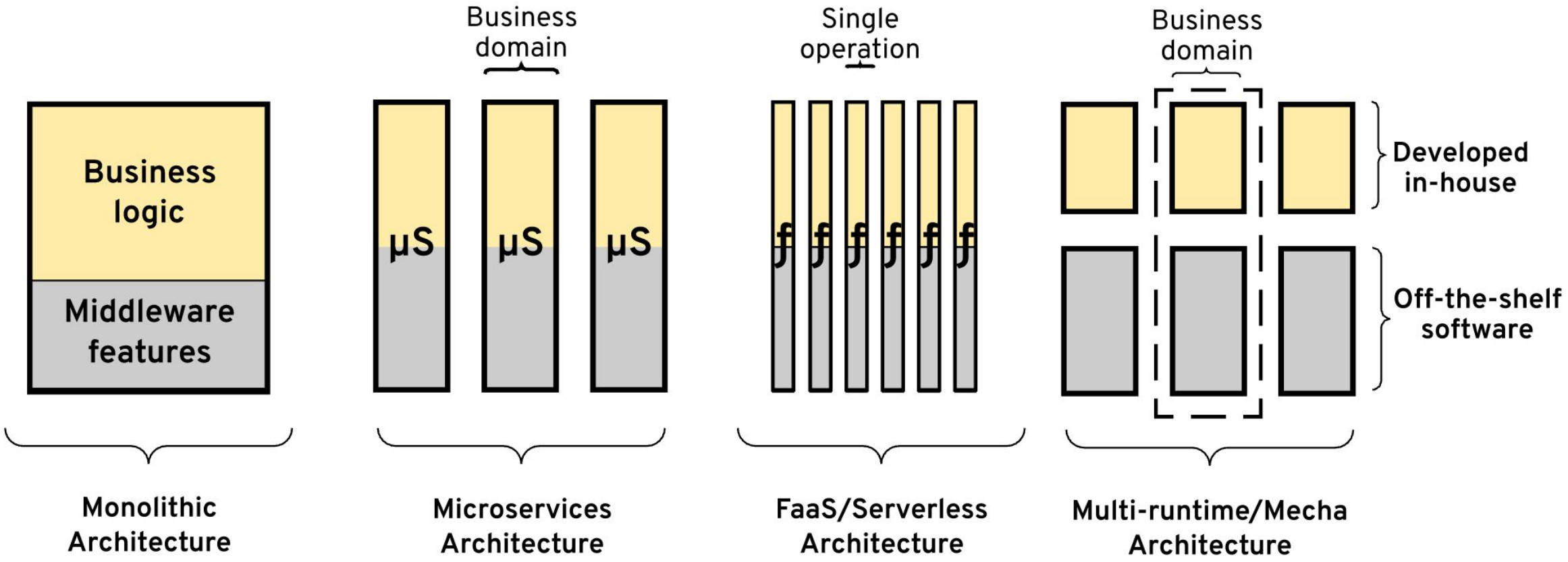

**Micrologic**

- Developed in-house
- Custom business logic
- Higher-level language
- HTTP/gRPC, CloudEvents

**Mecha**

- Off-the-shelf mechanincs
- Configurable capabilities
- Declarative (YAML, JSON)
- OpenAPI, AsyncAPI, SQL...

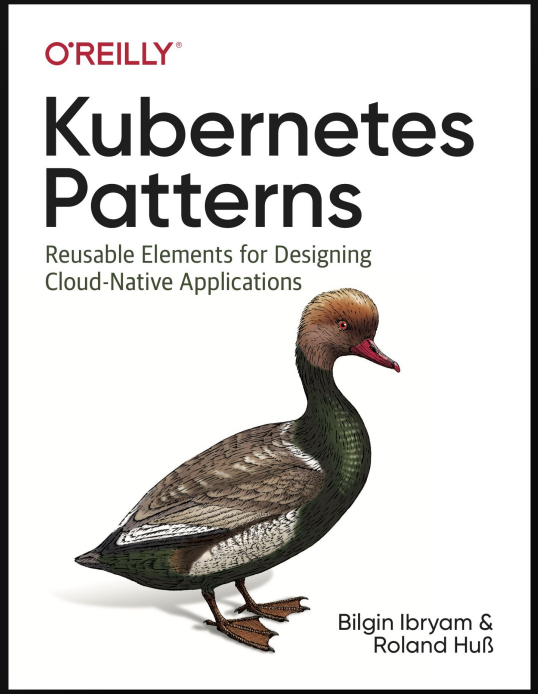
# What comes after Microservices?



# Thank You

@bibryam

https://k8spatterns.io






Red Hat