The Evolution of Distributed Systems on Kubernetes

Bilgin Ibryam
Product Manager @RedHat
@bibryam
Bilgin Ibryam

- 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

@bibryam
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

Business logic
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

Business logic

Middleware features

Monolithic Architecture
Traditional middleware limitations

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

Monolithic Architecture
Cloud-native architectures
Microservices and Kubernetes

Monolithic Architecture

- Business logic
- Middleware features

Microservices Architecture

- Business domain
- μS
- μS
- μS
Microservices and Kubernetes

- Business logic
- Middleware features

Monolithic Architecture

μS μS μS

Microservices Architecture

μS μS μS

Cloud native Architecture

kubernetes
Health probes

- Tracing
- Readiness
- Liveness

Container

API

Process health
Process metrics
Process logs
Managed start/stop
Declarative deployment

Rolling deployment

Fixed deployment

Blue-green release

Canary release
Demands & placement

Pod (CPU/MEM)

- CPU
  - Limit: 400m
  - Request: 200m

- Memory
  - Limit: 256Mi
  - Request: 128Mi

Node (CPU/MEM)

- Total capacity
- Limit
- Request
- Used

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

Pod  ✔️  Pod  ✔️  Pod  ✔️  Pod  ✔️  Pod

Stateful Service

Persistent networking
Persistent identity
Persistent storage
Storage provider

Stateless Service

Pod

Global Singleton

Pod

Active component
Container

StatefulSet/ReplicaSet (replicas=1)

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

Pod

Application containers
Sidecar

Init containers
Container 1
Container 2

Main

Execution sequence
Sidecar

Requests → Main container: HTTP

Sidecar container: git

Read → Pod

Write → Disk

Poll →
Controller Pattern

 Desired state

 Control loop

 Actual state

 Default schema
  - ReplicaSet
  - StatefulSet
  - Job, CronJob

 Default controllers
  - replicaset
  - statefulset
  - job, cronjob

 Managed resources state
  - Pod
  - PVC...

 Custom controller -> Custom behaviour
Operator Pattern

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?

App workload

Service A -> Service B

Java, Javascript, Python, .Net
What is Service Mesh?

Pod
Service A
Service proxy
App workload
Data plane
Pod
Service B
Service proxy
Java, Javascript, Python, .Net
Envoy, Linkerd, HAProxy, Traefik
What is Service Mesh?

- App workload
  - Pod
    - Service A
  - Pod
    - Service B
  - Service proxy

- Data plane
  - Configuration
  - Telemetry
  - Security

- Control plane

Java,
Javascript,
Python,
.Net
Envo,
Linkerd,
HAProxy,
Traefik
Istio,
Linkerd,
Consul,
Maesh
What is Service Mesh?

- API gateway
  - Single entrypoint
  - Traefik, Ambassador, Gloo, 3scale

- App workload
  - Pod
    - Service A
    - Service proxy
  - Pod
    - Service B
    - Service proxy
  - Java, Javascript, Python, .Net
  - Envoy, Linkerd, HAPerxy, Traefik

- Data plane

- Control plane
  - Configuration
  - Telemetry
  - Security
  - Istio, Linkerd, Consul, Maesh
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**
Common infrastructure for request-driven interactions that can "scale to zero".

**Eventing**
Common infrastructure for consuming and producing events declaratively.

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

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

```yaml
apiVersion: serving.knative.dev/v1alpha1
class: Service

metadata:
  name: lotto

spec:
  replicas: 1
  selector:
    matchLabels:
      app: lotto
  template:
    metadata:
      labels:
        app: lotto
    spec:
      containers:
      - image: cds19/lotto
```

@bibryam
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

<table>
<thead>
<tr>
<th>Service Invocation</th>
<th>Resource Bindings</th>
<th>State Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Act as a reverse proxy with built-in service discovery, tracing and error handling</td>
<td>Trigger code through events from input and output bindings to external resources.</td>
<td>Provides a key/value-based state API with pluggable state stores for persistence</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Distributed Tracing</th>
<th>Publish &amp; Subscribe</th>
<th>Actors</th>
</tr>
</thead>
<tbody>
<tr>
<td>See and measure the message calls across components and networked services</td>
<td>Secure, scalable messaging between services</td>
<td>Encapsulate code and data in reusable actor objects as a common microservices</td>
</tr>
</tbody>
</table>
Dapr architecture

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

@bibryam
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

Phase I
Basic Install
Automated application provisioning and configuration management

Phase II
Seamless Upgrades
Patch and minor version upgrades supported

Phase III
Full Lifecycle
App lifecycle, storage lifecycle (backup, failure recovery)

Phase IV
Deep Insights
Metrics, alerts, log processing and workload analysis

Phase V
Auto Pilot
Horizontal/vertical scaling, auto config tuning, abnormal detection, scheduling tuning

Welcome to OperatorHub.io
OperatorHub.io is a new home for the Kubernetes community to share Operators. Find an existing Operator or list your own today.

Categories
AI/Machine Learning
Application Runtime
Big Data
Cloud Provider
Database
Developer Tools
Integration & Delivery
Logging & Tracing
Monitoring
Networking

Items
Crunchy PostgreSQL for Kubernetes provided by Crunchy Data
Federator.ai Operator provided by ProphetStor Data Services, Inc.
KubeMQ provided by Kubenq.io
Portworx Enterprise provided by Portworx

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
  - 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**

**Dev Environment**
- **from:** "direct:route"
  - **steps:**
    - **split:**
      - **tokenize:** ","
    - **to:** "mock:split"

**Cloud**
- **Custom Resource**
- **Camel K Operator**
- **Running Pod**

**kamel CLI**

**Fast redeploy!**
**Less than 1 second!**

**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

Live updates!
State trends

Cloudstate Proxy

- Akka Sidecar

GRPC

User Function
(JavaScript, Go, Java,...)

KUBERNETES POD

GRPC

User Function
(JavaScript, Go, Java,...)

KUBERNETES POD

GRPC

User Function
(JavaScript, Go, Java,...)

KUBERNETES POD

Data Store
(Cassandra, Postgres, Spanner,...)
What does all this mean?
Multi-runtime microservices are here

- **Business logic**
  - Scale-to-zero
  - Event binding
  - Orchestration

- **Traffic routing**
  - Traffic routing
  - Network resilience

- **Scheduling**
  - Scheduling
  - Deployment

- **Polyglot**
  - Resource bindings
  - State abstractions
  - Distributed primitives

- **Envoy**
  - Observability
  - Policy enforcement

- **K8s**
  - Configuration
  - Resource mgmt

---

Out-of-process Mecha(nics)

Platform features
Smart sidecars and dumb pipes

Multi-runtime Microservice

Business logic runtime

Binding
State
Networking
Lifecycle

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

Mecha
- Off-the-shelf mechanisms
- Configurable capabilities
- Declarative (YAML, JSON)
- OpenAPI, AsyncAPI, SQL...
What comes after Microservices?

- Monolithic Architecture
- Microservices Architecture
- FaaS/Serverless Architecture
- Multi-runtime/Mecha Architecture
Thank You

@bibryam

https://k8spatterns.io