Application Networks

Xiangfeng Zhu
CSE461
From monolith to microservices

Monolithic Application
From monolith to microservices

Monolithic Application

Microservices
From monolith to microservices
From monolith to microservices

Function calls

Network Calls
Microservices are distributed systems
Microservices are distributed systems

Routing and Discovery

Timeout and Retry

Load Balancing
Microservices are distributed systems
Application networks: A new class of networks

Connect endpoints of an application, not anyone

Need rich message processing, not just IP

Built by application developers, not network engineers
Outline

• Background
• Service Mesh
• Application Defined Networks
Building application networks

Container
Service A
Business Logic
Comm. Config
Security Logic
Retry Logic
Tracing...

Container
JVM
Service B
Business Logic
Comm. Logic
Security Logic
Retry Logic
Tracing

Container
JVM
Service C
Business Logic
Comm. Logic
Security Logic
Retry Logic
Tracing
Building application networks

**Earlier:** Custom code for each microservices

Problems:

- Huge developer burden
- Network policies evolves independently
- Trust Issues
- …
Solution: Service Mesh with Sidecar Pattern

- Sidecar proxy handles all network logics
  - Traffic control / Routing
  - Resilience
  - Observability
  - Security
  - Policy Enforcement
  - …
Service Mesh Architecture

Application

Data Plane

Control Plane

Microservice

Sidecar

Microservice

Sidecar

Control Plane
Traffic Management

A/B testing / Traffic Shifting

Service A

Service B Production 95%
Service B Staging 5%

Traffic Mirroring

Service A

Service B Production
Service B Staging
Resilience

- Timeout
- Retry
- Circuit breaking

Sidecar

Service A

Sidecar

Service B

3s
Chaos Engineering

Fault Injection

Sidecar

Service A

Delay Injection

Sidecar

Service B
Security

mTLS Encryption & Authentication

Sidecar
Service A

Sidecar
Service B
Authorization

Service C

Sidecar

Service A

Sidecar

Sidecar

Service B
Observability
Service Mesh

- 90% organization uses service mesh according to a recent CNCF survey¹
Service Mesh

- Build on general network architecture use by the Internet
  - gRPC/HTTP/TCP/IP
Service Mesh Challenges

● High Overheads
  ○ Throughput / Latency / CPU

Service Mesh Data Path
service mess  /ˈsɜrves mɛs/  
noun

1. the result of spending more compute resources than your actual business logic dynamically generating and distributing Envoy proxy configs and TLS certificates.
End-to-End Performance Overhead

- TCP mode can increase the latency by 0.6X and CPU usage by 0.9X
- gRPC mode can increase the latency by up to 2.7X and CPU usage by 1.6X

Latency and CPU overhead of Envoy
Service Mesh Challenges

- High Overheads
  - Throughput/Latency/CPU
  - Overlapping/Unnecessary functionalities
  - Information hiding
Service Mesh Challenges

● High Overheads
  ○ Throughput/Latency/CPU
  ○ Overlapping/Unnecessary functionalities
  ○ Information hiding

● Non-portability
  ○ Difficult to offload to kernel and hardware
Service Mesh Challenges

● High Overheads
  ○ Throughput/Latency/CPU
  ○ Overlapping/Unnecessary functionalities
  ○ Information hiding

● Non-portability
  ○ Difficult to offload to kernel and hardware

● Difficult to use
  ○ API is complex and evolving
  ○ Poor extensibility

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortage of engineering expertise and experience</td>
<td>47%</td>
</tr>
<tr>
<td>Architectural and technical complexity</td>
<td>41%</td>
</tr>
<tr>
<td>Lack of guidance, blueprints, or best practices</td>
<td>36%</td>
</tr>
<tr>
<td>Choosing between projects and products</td>
<td>22%</td>
</tr>
<tr>
<td>None of the above</td>
<td>15%</td>
</tr>
<tr>
<td>Securing management buy-in</td>
<td>11%</td>
</tr>
</tbody>
</table>

CNCF Survey 2022
Service Mesh Challenges

- **High Overheads**
  - Throughput/Latency/CPU
  - Overlapping/Unnecessary functionalities
  - Information hiding

- **Non-portability**
  - Difficult to offload to kernel and hardware

- **Difficult to use**
  - API is complex and evolving
  - Poor extensibility

---

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: reviews-lua
  namespace: bookinfo
spec:
  workloadSelector:
    labels:
      app: reviews
  configPatches:
  - operation: INSERT_BEFORE
    config:
      context: SIDECAR_INBOUND
      listener:
        portNumber: 8080
      filterChain:
        filter:
          name: "envoy.filters.network.http_connection_manager"
          subFilter:
            name: "envoy.filters.http.router"
        patch:
          operation: INSERT_BEFORE
          value:
            lua_filter_specification:
              name: envoy.filters.http.lua
              typed_config:
                type: envoy.lua.LuaConfig
                inlineCode: |
                  function envoy.on_request(request_handler)
                    -- Make an HTTP call to an upstream host with the following headers, body, and timeout.
                    local headers, body = request_handler.httpCall("lua_cluster",
                      {"method" = "POST",
                       "path" = "/api/"
                      },
                      {"authority" = "internal.org.net",
                       "authorize call",
                       500})
                  end
                
          # The second patch adds the cluster that is referenced by the lua code
          # this patch is omitted as a new cluster is being added
          # applyTo: CLUSTER
          patch:
            context: SIDECAR_OUTBOUND
            patch:
              operation: ADD
              value:
                name: "lua_cluster"
                type: STRICT_DNS
                connect_timeout: 0.5s
                lb_policy: ROUND_ROBIN
                load_assignment:
                  cluster_name: lua_cluster
                  endpoints:
                    - lb_endpoints:
                      - endpoint:
                        address:
                          socket_address:
                            protocol: TCP
                            address: "internal.org.net"
                            port_value: 8888
```

---

Customize Istio Configuration
Application Defined Networks
Idea: Application Defined Networks (ADN)

Developers specify what the network should do at a high level

- Application-relevant abstractions
- Declarative, portable

A controller auto generate an optimized application-specific implementation

- Determine what processing happens and how (incl. hardware offload)
- Determine message headers
Idea: Application Defined Networks (ADN)

Developers specify what the network should do at a high level

- Application-relevant abstractions
- Declarative, portable

A controller auto generate an optimized application-specific implementation

- Determine what processing happens and how (incl. hardware offload)
- Determine message headers

Meets application-specific needs without a burdened implementation that does it all
ADN architecture

Specification → Controller → Compile and Deploy

Controller → Feedback

Controller → Control Signal → Optimizer
Example

S1→S2: LoadBalancing→Logging→Compression→FaultInjection(0.1)

FaultInjection(probability)

SELECT * from input
WHERE input.ver == 1 AND
    random() < probability
Example

S1→S2: LoadBalancing→Logging→Compression→FaultInjection(0.1)
Example

S1→S2: LoadBalancing→Logging→Compression→FaultInjection(0.1)
Example

S1→S2: LoadBalancing→Logging→Compression→FaultInjection(0.1)
Example

S1 → S2: LoadBalancing → Logging → Compression → FaultInjection (0.1)
Example

S1→S2: LoadBalancing→Logging→Compression→FaultInjection(0.1)