Video Streaming and Processing in a Service Mesh
The Cloud Native Way

Nikolay Nikolaev
VMware
Open Source Technology Center
Sofia, Bulgaria
January 25, 2019
Agenda

- **Introduction to the problem**
- Initial approach and local host pipeline
- Plain Kubernetes
- Service Mesh
- Future plans
Who am I?

- First computer experience: **PRAVETZ 82**
- Linux user since 1998
- A professional software engineer since 1999
- Open source contributions since 2001
- Mostly networking/telco related jobs since 2003
- Did a fair amount of software development in networking, virtualization, SDN, OpenStack
Use case - edge computing

Video processing on the edge

A popular telecom edge computing use-case is video optimization:

- Stream the high definition video to the edge and cache it
- Provide best possible video quality depending on the subscribers properties
- Optionally embed custom text/watermark
The challenge

A video processing pipeline with containers

- Services
  - Pre-defined video source – loop a video. Size 1920x1080. MP4
  - Apply timestamp/local time overlay
  - Apply logo overlay
  - Resize video
    - 426x240 (240p)
    - 852x480 (480p)
    - 1366x768 (HD)
  - Arbitrary set of services applied to the video source
- Use HTTP transport
- H264 encoder
Agenda

• Introduction to the problem
• **Initial approach and local host pipeline**
• Plain Kubernetes
• Service Mesh
• Future plans
The initial approach

Architecture

- **Source Container**
  - Args
    - `source /media/video.mp4`
    - `port 10100`

- **Resizing Container**
  - Args
    - `source http://video-source:10100`
    - `port 10100`
    - `size 480p`

- **Shell script**
  - Args
    - `source http://video:10100`
    - `port 10100`
    - `size 480p`

DevConf.cz 2019
The initial approach

Running in the cloud

ESXi on 2x6 CPUs + Hyperthreading Xeon E7-2803 @ 1.73GHz (Westmere EX Q2’11)

K8s node: 16 vCPUs 32GB RAM. Ubuntu Bionic + cvlc
The initial approach

Video scaling in Kubernetes - results

- 4 containers:
  - 1 source
  - 3 scalers

- Kubernetes services
  - NodePort

- All traffic is HTTP

- To achieve our final goals we need the apps to process the HTTP requests
The initial approach

VLC and a wrapping shell script

- Abstract away the video encoding engine
  - A shell script wrapping VLC
- VLC is great
  - User-friendly CLI interface
  - Lots of options
- Felt a bit heavier than what we needed
- Did not manage to get the control we wanted
- The shell script was too complex
  - Hard to maintain and extend
  - Lacked advanced functionality
    - HTTP header processing
Redesigned solution

Architecture

Source Container

http://video:10100

Args
- source http://video-source:10100
- port 10100
- size 480p

Streamer

FFMPEG

pipe

spawn

Per Request

Resizing Container

Args
- source /media/video.mp4
- port 10100

Streamer

FFMPEG

pipe

spawn

Per Request

DevConf.cz 2019
Video streaming pipeline on your host

Statically defined processing pipeline

- Resize HD
  1366x768

http://localhost:2002
- Apply timestamp

- Overlay logo

- Video source
  1920x1080
Demo

- Localhost pipeline
Agenda

- Introduction to the problem
- Initial approach and local host pipeline
  - Plain Kubernetes
- Service Mesh
- Future plans
K8s video streaming pipeline

Kubernetes

- Package the Streamer as Docker containers
  - Source – stream the video
  - Process – apply different operations

- Each stage is a deployment and a service
  - Replica 1, but can be more
  - ClusterIP for internal communication
  - NodePort for external facing containers
  - *Static* pipeline definition

```yaml
apiVersion: v1
kind: Service
metadata:
  name: video-transcode-medium
  labels:
    app: video-transcode-medium
spec:
  type: NodePort
  ports:
  - port: 10100
    nodePort: 31382
    targetPort: 10100
    name: http-medium
    selector:
      app: video-transcode-medium

---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: video-transcode-medium
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: video-transcode-medium
    spec:
      containers:
      - name: process-medium
        image: process:latest
        args: ["-source","http://video-timestamp:10100","-preset","480p","-port","10100"]
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 10100
```
Video streaming pipeline in Kubernetes

Statically defined processing pipeline

http://video:31383

Resize 240p

http://video:31382

Resize 480p

Apply timestamp

Overlay logo

http://video:31381

Resize HD 1366x768

Video source 1920x1080
Agenda

• Introduction to the problem
• Initial approach and local host pipeline
• Plain Kubernetes
• **Service Mesh**
• Future plans
Networking Evolution

Networks are defined by what they connect

<table>
<thead>
<tr>
<th>Workload host</th>
<th>Physical server</th>
<th>Virtual Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM</td>
<td>VM</td>
<td>VM</td>
</tr>
<tr>
<td>container</td>
<td>container</td>
<td>container</td>
</tr>
</tbody>
</table>

Physically connected wires, switches and servers. L2 and L3 connectivity

Virtual connections, vSwitches and VMs. L2 and L3 connectivity

Connecting applications, CNI L3 connectivity
Networking Evolution

Connecting API endpoints

Managing API endpoints connectivity
HTTP/HTTPS, REST, Application specific protocols
L7 introspection
What is a Service Mesh?

A Service Mesh is ...

- Dedicated infrastructure layer
- Handles service-to-service communication
- Ensures reliable delivery of requests
- Manages complex topology of services
  
  *Implemented as an array of lightweight network proxies*
  
  *Deployed alongside application code*
  
  *Doesn’t need the application to be aware*
Istio - a Service Mesh Implementation

Architecture

Control Plane API

Pilot
Mixer
Citadel

Control Plane

Data Plane

Configuration
Policy Checks, Telemetry
TLS Certs

Traffic Flow

Pod
Envoy
Service A

Pod
Envoy
Service B

DevConf.cz 2019
Dynamic video pipeline composition

Leverage Envoy and Pilot

- Define a dedicated HTTP header “Process-Video” with a 3 field value
  - First field chooses the size - low | medium | high | original
  - Second field enables timestamp application
  - Third field enables the logo insertion
  - Example, 480p video with logo: “Process-Video: medium_disable_enable”

- Parse the header at each Envoy hop and decide the next stage
- Each Streamer has to propagate this specific header down the pipeline to the next virtual service
HTTP header processing

Request flow
Video streaming and processing in Kubernetes

Dynamically defined service chain

- Route matching custom defined HTTP header

```
Gateway
http://video:31380
```

```
Video source
1920x1080
```

```
Source
```

```
Scale
```

```
Timestamp
```

```
Logo
```

```
Process-Video: low/medium/high_*
```

```
Resize 240p
```

```
Resize 480p
```

```
Resize HD 1366x768
```

```
Apply timestamp
```

```
Process-Video: *_enable_*
```

```
Process Video: *_enable_
```

```
Overlay logo
```

```
DevConf.cz 2019
```

```
© 2018 VMware, Inc.
```
Demo

• Istio demo
Processing virtual service

Logo virtual service yaml

- Match for host ‘video-logo’
- If header ‘Process-Video’ ends with ‘enable’, route to video-logo service
- Route all other requests to video-source service
Conclusions

• Istio Service Mesh proved to allow dynamic video pipeline build
• The cost is introducing Envoy proxies at each processing stage
  • Processing overhead
• Application has to propagate the HTTP header through the pipeline
• Bound to HTTP as a transport
• HTTP2 might have benefits
  • VLC 4.0
  • Ffmpeg completely lacks it
• Each stage is unpack-decode-process-encode-pack
  • Ineffective, better to pass raw video stream between the processing stages
  • Saves bandwidth
Agenda

• Introduction to the problem
• Initial approach and local host pipeline
• Plain Kubernetes
• Service Mesh
• Future plans
Cloud Native Video Processing

Available on GitHub

- Released as an open source project
  - The source code of Streamer
  - Docker build files
  - Kubernetes deployment yaml
  - Istio Virtual Service yaml description
  - Helper scripts to run locally and play the video content
  - Some docs
  - `github.com/nickolaev/cloud-native-video-processing`
Cloud Native Video Processing

Future plans

• Implement a solution on top of Network Service Mesh
  • NSM avoids side-car proxies, instead manages connections (non-CNI)
  • Does not need particular transport, can do anything

• Measure the real impact of Envoy

• Deploy at scale
  • Test at scale – multiple clients.
  • Explore the limits
    – how many processing containers per host
    – concurrent connections
Thank You

nnikolay@vmware.com