Traffic Management and Visibility Infrastructure for Rapid Microservice Delivery

Eddie Arrage

Futurewei Technologies Inc
Agenda

- **Cloud Native Concepts (5 min)**
  - Microservices
  - Service Deployment Strategies
  - Challenges

- **Traffic Management and Service Meshes (15 min)**
  - Service Meshes / Istio
  - Mesh Traffic Management
  - Mesh Visibility Tools

- **Visibility/Observability Infrastructure Mesh/Non-Mesh (10 min)**
  - OPNFV Clover (+ Clovisor)
Cloud Native

• Benefits:
  – Portable
  – Scalable
  – Ephemeral
  – Accessible
  – Flexible

  – Microservice oriented

  – Dynamically managed (Kubernetes)

  – Containerized
Microservices

- Monolithic App

- Break down into smaller chunks

- Microservice architecture puts functionality into separate services:
  - Iterative development
  - Division of labor
  - Reduce single point of failure
  - Language/deployment flexibility
  - Build different apps using subsets of services
  - Operations stakeholders are able to manage and upgrade components more easily
Microservice Validation & Deployment Strategies

• **Blue/Green**
  • Two identical environments of all microservices – Example:
    • Green in production
    • Release new version of service(s) in blue and validate
    • Revert to green if issues exist or cut over to blue if not

• **A/B Testing**
  • Support multiple versions of microservice simultaneously to compare variations/versions

• **Canary**
  • Push new code to small group of users to evaluate incremental changes
  • Early warning system for detecting problems

• **Employ ingress network services for traffic management: load balancers, proxies and/or service meshes to support**
Cloud Native / Microservice Challenges

- Microservice sprawl
  - Debug difficult without tools for visibility and traceability of entire system

- 150+ containerized services

- Microservice validation and deployment strategies require integrated traffic management
  - Current CI/CD pipelines in LFN projects have not adopted consist framework/methodology for doing this
Traffic Management and Service Meshes
Service Meshes

- Dedicated layer for managing service communication
  - Intra-service within cluster
  - External traffic entering cluster (ingress)
  - Internal traffic leaving cluster (egress)
  - Fit best for control-plane services

Examples: Istio, Conduit, Apache ServiceComb

- ‘Sidecar’ injected as a service proxy in each pod
- Allows for more advanced routing than native k8s networking
Istio Service Mesh

- Traffic Management
  - Load balancing
  - Request routing
  - Continuous deployment
    - Canary
    - A/B validation
  - Fault injection
  - Mirroring
  - Secure communication

- Visibility Built-in
  - Monitoring, tracing, logging

- Proxy oriented to HTTP/gRPC
  - mTLS (optional)

- Manual or automatic (namespace) sidecar injection
  - Toggle in/out of mesh easily
Istio Install

• Current release at 1.0.2,
• Works best on k8s v1.9+ (with mutating webhook)

Install

$ curl -L https://git.io/getIstioctl | sh
$ istioctl 1.0.2
$ kubectl label namespace <namespace> istio-injection.enabled
$ kubectl create -f <your-app-spec>.yaml
$ istioctl kube-inject -f <your-app-spec>.yaml | kubectl apply -- <<EOT

Setup

- automatic sidecar (namespace) sidecar injection
- Manual sidecar injection

Install Istio and SDC sample with Clover

$ docker pull opnfv/clover:latest
$ sudo docker run --rm \
  -v ~/.kube/config:/root/.kube/config \
  opnfv/clover \
  /bin/bash -c '/home/opnfv/repos/clover/samples/scenarios/deploy.sh'
- Clover developing set of sample L7 network services for use in k8s and meshes
- New in Clover Gambia release: modsecurity
  (Web Application Firewall + Apache web server)
Traffic within Mesh

- **k8s**
- **clover-jmeter-master**
  - Inject jmeter into mesh
  - Send traffic within cluster/mesh
- **Service Delivery Controller (SDC) Sample CNF**
External Traffic into Mesh

- LB at the edge of mesh receiving incoming/outgoing connections
- Control how traffic is routed within the mesh
• Content-based steering to determine destination of request
Istio Request Routing (2-2)

• Flexible request routing with Virtual Service
  • Match traffic and route to back end service
  • Match based on URI, HTTP headers (identity, user-agent)
  • Control with ‘weight’ field

• Ideal to validate REST based APIs and services
  • Support CI/CD deployment workflows
  • Canary validation/deployment

URLs to domain www.sdc.com

Match URI prefix ‘/test’ to clover-server2

Match HTTP header user-agent ‘chrome’ to clover-server3

Everything else to clover-server1
Istio Mirroring

- Mirroring or Shadowing
  - Sends a copy of live traffic to a mirrored service
  - Add an entry to Virtual Service resource under any route rule

Any traffic to `clover-server1` mirrored to `snort-ids`
Istio Destination Weight

- Use *weight* field under destination in Virtual Service to divide ingress traffic specified as percentage

- Two entirely different services
  - clover-server1
  - clover-server2

**URLs to domain www.sdc.com**

Match HTTP header user-agent ‘chrome’ to

- 20% to clover-server1
- 80% to clover-server2
Istio Destination Weight for Service Versions

- Additionally use **subset** field to divide traffic among multiple versions of the same service

- **DestinationRule** resource defines subset labels (original http-lb deployment resource)

- Useful for A/B testing

**Example**

**URLs to domain**

www.sdc.com

**Match HTTP header**

user-agent 'chrome'

to

95% to http-lb (v1)

5% to http-lb (v2)

**DestinationRule**

Defines subset v1/v2 labels

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: serviceversions
spec:
  hosts:
  - "www.sdc.com"
  http:
  - match:
    - headers:
      user-agent: chrome
    - destination:
      port: 9180
      host: http-lb
      subset: v1
      weight: 95
    - destination:
      port: 9180
      host: http-lb
      subset: v2
      weight: 5
```
Istio Fault Injection & Circuit Breaking

• Fault Injection
  • Inject faults to test the resiliency of your application
  • End-to-end failure recovery capability of the application as a whole

  – Delay: timing failures
    • Mimic network latency, or an overloaded upstream service

  – Abort: crash failures
    • mimic failures in upstream services (HTTP error codes)

• Circuit Breaking
  • Ejected from the load balancing pool when thresholds are exceeded
    • number of health check failures or number of conditions such as connection and request limits

• Useful for LFN projects that are planning or using cascading REST services
Istio Mesh - Visibility Tools

• Jaeger: Tracing
  • Good raw data
    • Individual traces in Jaeger
    • Metrics list in Prometheus
    • Dashboards in Istio / Grafana
  • But difficult to get insight of entire system (aggregate, top-level) and use analytics from data-sets

• Prometheus: Monitoring
Visibility/Observability Infrastructure Mesh/Non-Mesh
Clover Visibility

- Analyzes data from CNCF observability tools to provide abstraction
  - Gathers data and analyzes using Spark
- 4 core components (clover-system)
  - clover-collector (within k8s)
  - clover-controller (within k8s)
  - cloverctl (external)
  - clover UI (external)
- User interacts with cloverctl or UI
  - CLI/UI use same REST API from clover-controller service
  - Chooses services to track
  - Outputs analyzed data to Redis
**Clover Visibility Initialization (1-2)**

- Install Istio
- Install clover-system components within k8s
- Expose clover-controller using LB or NodePort k8s service resource
- Gambia release will have CLI / script installation

Use CLI to initialize visibility
- Create traces, spans, metrics Cassandra schemas

Start visibility
- Collector begins gathering data from Jaeger, Prometheus

Clear visibility
- Truncates tables

```bash
$ cloverctl init visibility
$ cloverctl start visibility -f visibility.yaml
$ cloverctl clear visibility
```
Clover Visibility Initialization (2-2)

- Set sampling interval for collector
- Tracing/monitoring k8s DNS names
- Tracing/monitoring listening ports (Jaeger/Prometheus)

$ cloverctl start visibility -f visibility.yaml

metrics.yaml

$ cloverctl set visibility -f metrics.yaml

visibility.yaml

```yaml
sample_interval: "10s"
t_host: tracing.cloudeye-system
r_port: "80"m_port: "9000"
t_host: prometheus.cloudeye-system
```
• Analyze trace data at aggregate level
  • Calculate average response time for various services
• Break down data in various ways
  • Per URL, Per Service/URL, more TBA in Gambia release
• Find issues with REST services such as service HTTP status codes being returned
• Validate service mesh traffic management policies such as request routing by user-agent (ex. mobile vs desktop)
• Characterize the composition of the traffic

• Output service request/response rates over time, lost requests, etc.
Clover Clovisor

**Istio**
- Large compute footprint
  - Istio - 13 Containers
  - Sidecar container per service
    - Latency overhead with long service chains
- Lacks visibility for:
  - L3 network
  - Other L4-7 content
- Lacks networking breadth for traffic management
  - Doesn’t support wide set of protocols, tunneling, encapsulation

**Clovisor**
- Leverages eBPF
- Installed on k8s cluster nodes
- Hooks to OpenTracing, Jaeger
Clovisor: Network Tracing... the Cloud Native Way

1. Cloud Native:
   a) Cloud Provider Independent
      • Bare-metal servers, GKE, EKS...etc
   b) CNI Plugin Agnostic
      • All CNI plugins should work unless such plugin does ........ bypass
   c) CPU Architecture Independent
      • Any architecture supported by Linux (x86, ARM...etc), code (kernel versions 4.14 and 4.15 currently)

2. Implemented with Cloud Native Design Methodologies:
   a) Config Decoupled from Compute
      • Config store in backing store or through environment variables
   b) Relatively Stateless
      • TCP connection/session tracking only dynamic states
   c) Scale-out Architecture
      • Pod monitoring partitioning via election from datastore
      • DaemonSet —- linearly scale on each node in cluster

3. In-depth Integration with Cloud Native Ecosystem Projects:
   a) Built-in Kubernetes Client
      • Monitoring k8s pod states
   b) Integrate with CNCF Collector Projects
      • OpenTracing to Jaeger, metrics to Prometheus
Clovisor Architecture

• Lightweight, low latency network tracing module
• Utilizes IOVisor (bcc, gobpf) with eBPF to insert bytecode in Linux kernel to examine packets from both ingress / egress direction of a k8s pod

• In cluster client to automate process of monitoring and service port / protocol info
• Stream trace / stats / metrics / logs to respective tracer / collector modules
• Configure monitoring labels (namespace:label-key:label-value)
• In this case: “default” namespace, key: “app”, value: “proxy”
• Start Clovisor (on node, verify if the tc filter is created for device)
• curl www.cnn.com with http-proxy service port (3456)
• curl www.google.com with http-proxy service port (3456)
• Check Jaeger UI to verify traces written/sent
Visibility Use-Cases

- Easily pinpoint issues with individual services
- Integrate into CI to determine success/failure of jobs
  - CI used to determine CD deployment pipeline
- Monitor infrastructure in operations to determine system health
- Characterize the composition of traffic for content delivery or security
- Leverage to automate orchestration or zero-tech provisioning
Summary
Key Take-Aways

• Service meshes allow microservices to be delivered more rapidly with integrated traffic management and visibility hooks
  – Visibility helps developers pinpoint issues and operators manage infrastructure
  – Built-in traffic management allows for microservice CI/validation and deployment strategies
  – Ideal for control-plane and REST services
• Service mesh distributed tracing/monitoring collects data efficiently but lacks an aggregate view of infrastructure/services
  – LFN projects such as Clover can provide high-level analytics for developers and operators
• Service mesh overhead/footprint and lack of networking breadth (both for visibility & routing/security)
  – Clovisor is a promising approach to fill gaps and add additional networking extensions
Clover Project Info

- **Project Wiki**

- **Slack Channel**
  - #clover-project

- **Github Repo**
  - [https://github.com/opnfv/clover](https://github.com/opnfv/clover)
Istio - Control Egress Traffic

- Default Istio-enabled services are unable to access URLs outside of the cluster
  - Pods use iptables to transparently redirect all outbound traffic to the sidecar proxy, which only handles intra-cluster destination

Send traffic outside of mesh to ‘www.sdc.com’

(assuming this is a valid domain in DNS)
Augmenting Mesh/Kubernetes Ingress

- Bolstering security
- Improve visibility data

Integrate with Istio ingress controller

• New in Clover Gambia release
IOVisor & eBPF

- **eBPF**
  - Inject bytecodes to kernel trace points / probes
    - Event driven model
  - Networking: tc
    - Utilizes Linux tc (traffic control) to inject bytecode on ingress and egress direction of a network interface
  - Verifier / JIT (just-in-time compiler)
    - Verifier ensures bytecode does NOT crash kernel

- **IOVisor bcc**:
  - Ease of eBPF Development
    - Helper functions, kernel API wrappers…etc
  - Dynamic Validation and Compilation
    - Userspace eBPF code written in ‘C’ is dynamically verified (static analysis) and compiled
  - gobpf
    - Golang interface for userspace code — more performant than Python