Delivering Network Services using Cloud Native Methodology
Eddie Arrage
Wenjing Chu
Futurewei Technologies Inc.
Agenda

• **Cloud Native Concepts/Methodology (10 min)**
• **Segmenting & Instrumenting Microservices (15 min)**
  • Instrumentation
  • Example in Clover
• **Managing & Controlling Traffic (20 min)**
  • Service Meshes / Istio
  • Mesh Visibility Tools
  • Mesh Traffic Management
  • Augmenting Meshes
• **Debugging & Monitoring (25 min)**
  • Visibility/Observability Infrastructure
  • Introduction to Clovisor
• **Integrating & Validating (10 min)**
  • L7 Jmeter validation
  • Jenkins integration
• **Deploying & Managing Services, Infrastructure (15 min)**
  • Introduction to Spinnaker – CI/CD
Cloud Native

- Benefits:
  - Portable
  - Scalable
  - Ephemeral
  - Accessible
  - Flexible

- Microservice oriented

- Dynamically managed (Kubernetes)

- Containerized
Operators need to manage network services with cloud-native constructs
- Bridge divide between built-in networking (Kubernetes, service meshes) and apps
- Example: Google Istio as a service cloud offering
- Support traffic management for CI/CD precepts: canary, blue/green, etc.

App evolution to microservices
- Develop, debug and manage individual Lego blocks

App developers want abstracted network
- Usually to support web/REST oriented services
  - Example in LFN: many ONAP services have REST interface
  - Ideal for control-plane services
- Network management model needs to fit paradigm
Cloud Native / Microservice Challenges

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

• Validation difficult as developers need to test system but might only own one service  
  • Integrated testing and ease of system deployment

• Currently CI/CD pipeline in most LFN projects largely stops at CI level  
  • Need to manage deployment pipelines  
  • Support traffic management for cloud native

• Traditional operators need to consider how to offer compelling cloud services  
  • Control traffic in/out of containerized environments  
  • Network components for configurable ingress with security
Cloud Native Methodology

Deploying & Managing Services, Infrastructure

GrPC { REST }
Segmenting & Instrumenting Microservices

Managing & Controlling Traffic

Integrating & Validating

Debugging & Monitoring

Continuously
Segmenting & Instrumenting Microservices
Microservices

- Break down into smaller chunks

- Monolithic App

- 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 Instrumentation

- **ConfigMaps**
  - Manage/inject app configuration
  - Kubernetes resource
  - Keep containers agnostic

- **gRPC**
  - Open-source RPC framework
  - Client/server
  - Bindings for most languages
  - Frequent configuration

- **Shared Data Stores**
  - Exchange network data, state management
Microservice Clover Example

- Clover-System Tools
  - cloverctl - CLI interface
  - clover-controller - in-cluster message routing and UI dashboard
  - clover-jmeter – L4-7 client-emulation for CI/CD, validation

- Sample Network Services
  - Security: IDS, WAF
  - L4-7: proxy, load balancer…
  - Combine in various CNFs
  - Employ Linux services
  - Implement gRPC server for instrumentation
  - Redis data-store to share packet, security event data
gRPC Demo

Install for Python

```bash
$ python -m pip install grpcio protobuf
```

Generate gRPC code

```bash
$ python -m grpc_tools.protoc -I ./ --python_out=. --grpc_python_out=. snort.proto
```
Managing & Controlling Traffic
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/getLatestIstio | sh
$ cd istio-1.0.2
$ export PATH=$PWD/bin:$PATH
$ kubectl label namespace <namespace> istio-injection-enabled
$ kubectl create -n <namespace> -f <your-app-spec>.yaml
$ kubectl apply -f install/kubernetes/istio-demo.yaml
```

Setup

```
$ istioctl kube-inject -f <your-app-spec>.yaml | kubectl apply -f -
```

Install Istio and SDC sample with Clover

```
$ docker pull opnfv/clover:latest
$ sudo docker run --rm -v ~/.kube/config:/root/.kube/config:ro
opnfv/clover /bin/bash -c '/home/opnfv/repos/clover/samples/scenarios/deploy.sh'
```

- automatic sidecar (namespace) sidecar injection
- Manual sidecar injection
Network Service Catalog

- 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)

<table>
<thead>
<tr>
<th>Service</th>
<th>Kubernetes Deployment App Name</th>
<th>Docker Image</th>
<th>Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proxy</td>
<td>proxy-access-control</td>
<td>clover-ns-nginx-proxy</td>
<td>HTTP: 9180 GRPC: 50054</td>
</tr>
<tr>
<td>Load Balancers</td>
<td>app: http-lb version; http-lb-v1 version; http-lb-v2</td>
<td>clover-ns-nginx-lb</td>
<td>HTTP: 9180 GRPC: 50054</td>
</tr>
<tr>
<td>Intrusion Detection System (IDS)</td>
<td>snort-ids</td>
<td>clover-ns-snort-ids</td>
<td>HTTP: 80, Redis: 6370   GRPC: 50052 (config) GRPC: 50054 (alerts)</td>
</tr>
<tr>
<td>Servers</td>
<td>clover-server1 clover-server2 clover-server3 clover-server4 clover-server5</td>
<td>clover-ns-nginx-server</td>
<td>HTTP: 9180 GRPC: 50054</td>
</tr>
</tbody>
</table>
Traffic within Mesh

- **Emulated Clients**
  - Inject jmeter into mesh
  - Send traffic within cluster/mesh

- **clover-jmeter-master**

- Service Delivery Controller (SDC) Sample CNF
**External Traffic into Mesh**

- **Istio Ingress**
  - LB at the edge of mesh receiving incoming/outgoing connections
  - Control how traffic is routed within the mesh

---

**Istio Gateway**

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: sdc-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
        name: http
        protocol: HTTP
    hosts: ["*"]
```

**Mesh**

**Services**

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: sdc-sample
spec:
  hosts:
  - ["*"]
  gateways:
  - sdc-gateway
  http:
  - match:
      - uri:
          prefix: /
      - destination:
          port:
            number: 9188
            host: proxy-access-control
```
Content-based steering to determine destination of request
Support CI/CD precepts with canary versions
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

---

**URLs to domain**

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: directserver
spec:
  hosts:
  - "www.sdc.com"
  http:
    match:
    - uri:
        prefix: /test
      route:
      - destination:
          port:
            number: 9180
          host: clover-server2
        match:
          - headers:
              user-agent: chrome
            exact: chrome
      route:
      - destination:
          port:
            number: 9180
          host: clover-server3
    route:
    - destination:
        port:
          number: 9180
          host: clover-server1
```

**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 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 - 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

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: sdc-ext
spec:
  hosts:
  - www.sdc.com
  ports:
    - number: 80
      name: http
      protocol: HTTP
  resolution: DNS
  location: MESH_EXTERNAL
```

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

(assuming this is a valid domain in DNS)
Istio Mesh - Visibility Tools

- Jaeger: Tracing
  - Jaeger UI
    - Find Traces
      - Service (1)
        - proxy-access-control
      - Operation (10)
        - all
      - Tags (1)
        - version=1.0
    - Lookback
      - Last Hour
    - Min Duration
      - exp. 10m, 30m, 1h
    - Max Duration
      - exp. 10m, 30m, 1h
  - 20 Traces
    - Event: Today
      - 4:53:42 PM
      - proxy-access-control
      - http

- Prometheus: Monitoring
  - Good raw data
    - Individual traces in Jaeger
    - Metrics list in Prometheus
  - But difficult to get insight of entire system (aggregate, top-level)
Augmenting Mesh/Kubernetes Ingress

- Bolstering security
- Improve visibility data

Integrate with Istio ingress controller

Traffic

Redirect

Mirror

modsecurity
Open Source Web Application Firewall

clover-gateway services
- New in Clover Gambia release
Debugging & Monitoring
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

$ 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

visibility.yaml

```yaml
sample_interval: "10s"
t_host: tracing.istio-system
t_port: "58612"
m_host: prometheus.istio-system
m_port: "9090"
```

- Configure tracing services that visibility will analyze
- Configure metric prefixes/suffixes to analyze

$ cloverctl set visibility –f metrics.yaml

```yaml
services:
  - name: proxy_access_control
  - name: clover_server1
  - name: clover_server2
  - name: clover_server3
prefixes:
  - prefix: envoy_cluster_outbound_9180__
  - prefix: envoy_cluster_inbound_0180__
suffixes:
  - suffix: _default_svc_cluster_local_upstream_rq_2xx
  - suffix: _default_svc_cluster_local_upstream_cx_active
```
• 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

<table>
<thead>
<tr>
<th>User-Agents</th>
<th>Request URLs</th>
<th>Status Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://proxy-access-control.default:9190/">http://proxy-access-control.default:9190/</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**HTTP Details**

**Monitoring Metrics**

<table>
<thead>
<tr>
<th>metric</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>envoy_cluster_inbound_9180__clover_server3_default_svc_cluster_local_upstream_rq_2xx</td>
<td>4558</td>
</tr>
<tr>
<td>envoy_cluster_inbound_9180__proxy_access_control_default_svc_cluster_local_upstream_rq_2xx</td>
<td>12109</td>
</tr>
<tr>
<td>envoy_cluster_outbound_9180__clover_server1_default_svc_cluster_local_upstream_rq_2xx</td>
<td>5967</td>
</tr>
<tr>
<td>envoy_cluster_outbound_9180__clover_server1_default_svc_cluster_local_upstream_rx_active</td>
<td>9</td>
</tr>
<tr>
<td>envoy_cluster_outbound_9180__clover_server2_default_svc_cluster_local_upstream_rq_2xx</td>
<td>5751</td>
</tr>
<tr>
<td>envoy_cluster_outbound_9180__clover_server2_default_svc_cluster_local_upstream_rx_active</td>
<td>6159</td>
</tr>
<tr>
<td>envoy_cluster_outbound_9180__proxy_access_control_default_svc_cluster_local_upstream_rq_2xx</td>
<td>451</td>
</tr>
<tr>
<td>envoy_cluster_outbound_9180__proxy_access_control_default_svc_cluster_local_upstream_rx_active</td>
<td>6124</td>
</tr>
<tr>
<td>envoy_cluster_outbound_9180__proxy_access_control_default_svc_cluster_local_upstream_rx_2xx</td>
<td>4177</td>
</tr>
<tr>
<td>envoy_cluster_inbound_9180__clover_server1_default_svc_cluster_local_upstream_rx_active</td>
<td>0</td>
</tr>
<tr>
<td>envoy_cluster_inbound_9180__clover_server2_default_svc_cluster_local_upstream_rx_active</td>
<td>10</td>
</tr>
<tr>
<td>envoy_cluster_inbound_9180__proxy_access_control_default_svc_cluster_local_upstream_rx_active</td>
<td>0</td>
</tr>
<tr>
<td>envoy_cluster_inbound_9180__proxy_access_control_default_svc_cluster_local_upstream_rx_2xx</td>
<td>7</td>
</tr>
<tr>
<td>envoy_cluster_outbound_9180__clover_server3_default_svc_cluster_local_upstream_rx_active</td>
<td>11</td>
</tr>
<tr>
<td>envoy_cluster_inbound_9180__clover_server3_default_svc_cluster_local_upstream_rx_active</td>
<td>0</td>
</tr>
<tr>
<td>envoy_cluster_inbound_9180__clover_server3_default_svc_cluster_local_upstream_rx_2xx</td>
<td>0</td>
</tr>
</tbody>
</table>
Clover Clovisor

Istio

- Large compute footprint
  - Istio - 13 Containers
  - Sidecar container per service
- 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

- Hooks to OpenTracing, Jaeger
- Leverages eBPF
- Installed on k8s cluster nodes
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 kernel 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
• 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
- 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
Integrating & Validating
**Jmeter Validation (1-2)**

- Jmeter is a mature L4-7 testing open source project
  - HTTP client emulation for functional/performance validation
  - Determine max session/connection rates, connection capacity, etc.
- Clover created a Jmeter service for use within k8s
  - Uses Jmeter master/slave approach
  - Master as a single pod deployment may be used
  - Jmeter slaves can be added for additional scale
    - Master <-> slave communication only works outside of mesh
  - Detailed test plan creation, test control and result collection
  - Integrated into clover-system in CLI, UI and clover-controller

```
$ cloverctl create testplan -f jmeter.yaml
$ cloverctl start testplan
$ cloverctl start testplan -s <slave count>
$ cloverctl get testresult -r log
$ cloverctl get testresult -r results
```
Jmeter Validation (2-2)

- Validate infrastructure including visibility tools, ingress controller, sample app
- Facilitate CI jobs
- Configure clover-server(s) with resources including URL routes and files of varying sizes
Jmeter Test Plan Creation

- Clover Jmeter yaml abstracts test plan XML
- Specify simple parameters:
  - # threads (users)
  - Loops
  - URL List
    - Name (for results)
    - URL (with port and URI)
    - Method (GET, POST, …)
    - User-agent HTTP header

```
load_spec:
  num_threads: 20
  loops: 2
  ramp_time: 60
url_list:
  - name: url1
    url: http://www.sdc.com:80/test/
    method: POST
    user-agent: javascript
  - name: url2
    url: http://www.sdc.com:80/
    method: GET
    user-agent: chrome
  - name: url3
    url: http://www.sdc.com:80/phpmyadmin
    method: GET
    user-agent: robot
```

$ cloverctl create testplan –f jmeter.yaml
$ cloverctl start testplan
$ cloverctl get testresult –r results
Setup Jenkins for CI

- Setup Jenkins within k8s

Deployment yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: jenkins
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: jenkins
    spec:
      containers:
        - name: jenkins
          image: jenkins/jenkins:lts
          ports:
            - containerPort: 8080
```

NodePort service yaml

```
kind: Service
apiVersion: v1
metadata:
  name: jenkins
spec:
  selector:
    app: jenkins
  ports:
    - name: jenkins
      protocol: TCP
      port: 8080
    - name: http
      protocol: TCP
      port: 8080
  type: NodePort
```

Load Balancer service yaml (expose in GKE)

```
kind: service
apiVersion: v1
metadata:
  name: jenkins
spec:
  selector:
    app: jenkins
  ports:
    - name: jenkins
      protocol: TCP
      port: 8080
    - name: discover
      protocol: TCP
      port: 50000
  type: LoadBalancer
```
Integrate and Validate Demo

- Python script uses clover-controller REST interface:
  - Clear visibility
  - Create jmeter testplan
  - Start jmeter testplan
  - Get visibility stats
- PASS/FAIL from expected requests sent by jmeter checked from visibility
- Set exit status in script for Jenkins job success/failure
Clover Server Instrumentation

- **clover-server**
  - Endpoint to terminate traffic for end-to-end validation through network services
  - Nginx based server

- **gRPC interface to reconfigure:**
  - Setup various paths, listening port, etc.

- Nginx Upload module used for file upload with good performance

**Configure**

- **Listening port**
  - server_port: "9180"
- **Deployment name**
  - server_name: "clover-server1"
- **Site root/index**
  - site_root: "/var/www/html"
  - site_index: index.html
- **Upload path config**
  - upload_path_config: "/upload"
  - upload_path_test: "/upload_test"
  - locations:
    - uri_match: "/test"
      directive: "try_files $uri @default1"
      path: "/test"
    - uri_match: "/new"
      directive: "try_files $uri @default1"
      path: "/new"
    - uri_match: "/clover/clover"
      directive: "try_files $uri @default2"
      path: "/clover/clover"
- **Path URLs**
- **Move uploaded files to paths**
  - files:
    - src_file: "/var/www/html/upload/0000000001"
    - dest_file: "/var/www/html/test/touch.wav"

```bash
$ cloverctl set server -f server.yaml
```
Deploying & Managing Services, Infrastructure
Spinnaker Introduction

• Overview
  – Construct and manage continuous delivery workflows
  – View/manage cloud resources
  – Pipeline-based engine

• Stages
  – A stage in Spinnaker is an atomic building block for a pipeline

• Pipelines
  – Support various deployment strategies: blue/green, canary…
  – Deploy to various clouds
  – Execution manually or based on triggers
  – Stages can be executed sequentially or in parallel
Spinnaker – Stage Types

- **Bake (Manifest)**
  - deploy Helm charts (alpha)

- **Check Preconditions**
  - check environment

- **Delete (Manifest)**
  - delete k8s resources

- **Deploy (Manifest)**
  - deploy k8s based on yaml

- **Pipeline**
  - allow pipeline daisy-chaining

- **Scale (Manifest)**
  - increase k8s replicas

- **Script**
  - run a script (instead of Jenkins option)

- **Undo Rollout (Manifest)**
  - go back to a prior revision

- **Find Artifact From Execution**
  - promote artifacts between executions

- **Find Artifacts From Resource**
  - find image from k8s resource

- **Jenkins**
  - run Jenkins jobs

- **Manual Judgment**
  - prompts user before continuing

- **Wait**
  - introduce delay in pipeline

- **Webhook**
  - execute REST call
Trigger Spinnaker pipelines from many different events including: Jenkins, Git, Docker Registry or other Spinnaker pipelines
Spinnaker – Common Software Deployment Strategies

• Blue/Green
  • Two identical environments – ex. green in production
  • Release new version of services in blue and validate
  • Revert to green if issues exist

• A/B Testing
  • Support multiple versions 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: load balancers, proxies and/or service meshes (ex. Istio) to support
Cloud-Native CI/CD with Spinnaker - Demo

- Spinnaker can deploy to multiple cloud providers
  - Including Kubernetes, GKE
  - Openstack
- Pipelines are used to control flow from commit/build/test to bake and deploy in 'production'
- CI validation scripts are used to determine if individual services and overall use-cases are healthy
Developer

1. Change Code
   Or
   Add Git Tag

2. Use Jenkins git plugin and setup Polling Job

3. Poll for Jenkins Job
   monitor-git

4. Run Jenkins Job
   build-sample-app

5. Push to DockerHub

6. Run rest of pipeline in Google demo

Github Pubic Repo

Job monitor-git

Job build-sample-app

sample-app local docker build

Poll for Jenkins Job
monitor-git

Pipeline Trigger

Run Jenkins Job
build-sample-app

Pipeline
- Deploy to Canary
- Functional Tests of Canary Deployment
- Manual Approval
- Deploy to Production

Clover Spinnaker Sample-App
OPNFV CI/CD with Spinnaker (1-2)

1. Developer
   - Change Code
   - Or
   - Add Git Tag

2. Opnfv Gerrit
   - Github
   - Public Repo
   - Gerrit trigger jenkins

3. Opnfv Jenkins
   - Job
     - build-project-app
   - Run job

4. Push to DockerHub
   - local docker build & push to dockerHub

5. Poll for Jenkins Job
   - build-project-app

6. Spinnaker
   - Pipeline Trigger
   - Deployment
     - Deploy to Canary
     - Functional Tests of Canary Deployment
     - Manual Approval
     - Deploy to Production
OPNFV CI/CD with Spinnaker (2-2)

1. Developer
   - Change Code
   - Or Add Git Tag

2. Opnfv Gerrit
   - Github
   - Public Repo
   - Gerrit trigger
   - Jenkins

3. Opnfv Jenkins
   - Run job
   - Job
   - `build-project-app`
   - local docker
   - build & push
to dockerHub

4. Push to DockerHub

5. Monitor docker hub

6. Spinnaker
   - Pipeline
   - Trigger
   - Deploy to Canary
   - Functional Tests of Canopy Deployment
   - Manual Approval
   - Deploy to Production
Summary
Take-Aways

- LFN projects can be packaged in flexible ways if delivered as microservices in k8s
  - Test projects, ONAP, OS admin services, etc.

- Spinnaker can help manage complex CD pipelines across clouds

- Flexible integrations with Jenkins, DockerHub, Git, etc. allow CI and CD to be combined

- Validation tools required in CI/CD pipeline stage acceptable

- Employ visibility in CI logic

- Meshes allow services to be delivered with cloud-native CD principles

- Ideal for control-plane and REST services

- Cloud-native visibility helps developers pinpoint issues and operators manage infrastructure

Segmenting & Instrumenting Microservices

Deploying & Managing Services, Infrastructure

Managing & Controlling Traffic

Integrating & Validating

Debugging & Monitoring

Continuously
Clover Project Info

- **Project Wiki**

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

- **Github Repo**
  - [https://github.com/opnfv/clover](https://github.com/opnfv/clover)
Appendix
Consider cloud native for OPNFV test projects
- Package as micro-services
- Many are already containerized
  - Functest divided into 8+
- Add gRPC or REST server interfaces
- Make actions more atomic within each
- Orchestrate system level tests using different combinations of services/actions
- Deploy all OPNFV test services in a single manifest potentially
- Use tool-chains such as Spinnaker for CI/CD
- Installer projects are also considering cloud native for some services
Istio Control-Plane Components

Service A

HTTP/1.1, HTTP/2, gRPC or TCP — with or without mTLS

Service B

Policy checks, telemetry

Proxy

Config data to proxies

TLS certs to proxies

Pilot

Mixer

Citadel

Control Plane API
**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
# Integrate and Validate Demo

```python
import requests
import sys
import time

CLOVER_CONTROLLER = 'http://10.145.71.21:39880'
#CLOVER_CONTROLLER = 'http://clover-controller.clover-system'
USER_COUNT = 4
EXPECTED_COUNT = 4 * 4

# Clear visibility
response = requests.get(CLOVER_CONTROLLER + '/visibility/clear')
print(response.text)
if response.status_code != 200:
    print("Failed to clear visibility")
    sys.exit(-1)

# Start JMeter testplan
jmeter_start_url = CLOVER_CONTROLLER + '/jmeter/start'
response = requests.get(jmeter_start_url)
print(response.text)
if response.status_code != 200:
    print("Failed to start jmeter")
    sys.exit(-1)

time.sleep(80)

# Get visibility trace count
get_stats_url = CLOVER_CONTROLLER + '/visibility/stats/toplevel'
data = response.json()  
response = requests.get(get_stats_url)
print("Trace count: {}", format(data['trace_count']))
if response.status_code != 200:
    print("Failed to get visibility stats")
    sys.exit(-1)

# Simple check to determine validation status
if int(data['trace_count']) != EXPECTED_COUNT:
    print("Validation failed ")
    sys.exit(-1)
else:
    print("Validation passed")
```