Navigating the Networking Maze of Kubernetes

A Journey of Discovery, Confusion, and (Hopefully) Enlightenment

aojea@google.com

FOSDEM 2024
How it started

Borg (Proprietary)  Ω  Kubernetes (open-source)

How it is going

I need to know why moving our app to the cloud didn't automatically solve all our problems.

You wouldn't let me re-architect the app to be cloud-native. Just put it in containers.

You can't solve a problem just by saying techy things. Kubernetes.
Kubernetes Architecture: extensible and pluggable
Kubernetes Architecture: API + Semantics

Kubernetes API: IPv4

```yaml
apiVersion: v1
kind: Node
metadata:
  name: node1
spec:
podCIDR: 10.0.0.0/24

status:
  addresses:
  - address: 192.168.0.5
type: InternalIP

apiVersion: v1
kind: Pod
metadata:
  name: app-http
namespace: default
spec:
  containers:
    - name: app
      image: myapp:0.1
  status:
    phase: Running
    podIP: 10.0.0.5

apiVersion: v1
kind: Service
metadata:
  name: lb-app
namespace: default
spec:
  clusterIP: 10.96.0.13
  type: LoadBalancer
  loadBalancer:
    ingress:
    - ip: 196.23.45.23
```

End-to-End Testing in Kubernetes

Table of Contents

- End-to-End Testing in Kubernetes
  - Overview
  - Building Kubernetes and Running the Tests
    - Cleaning up
    - Advanced testing
      - Extracting a specific version of Kubernetes
      - Bringing up a cluster for testing
      - Debugging clusters
      - Debugging an E2E test with a debugger (delve)
    - Local clusters
      - Testing against local clusters
      - Version-sliced and upgrade testing
      - Test suite naming convention
    - Kinds of tests
      - Viper configuration and hierarchical test parameters
      - Conformance tests
      - Continuous integration
Kubernetes Implementation: one of multiple
Kubernetes Networking: end to end principle
Kubernetes networking: Nodes

```
apiVersion: v1
type: InternalIP
kind: Node
metadata:
  name: node1
spec:
  podCIDR: 10.0.0.0/24
...
status:
  addresses:
    - address: 192.168.0.5
```
Kubernetes networking: Nodes

1. Kubelet registers the node object

2. Kubelet sets the status and conditions

3. If running with cloud-provider=external, the CCM sets the addresses and some conditions

4. If ipam-controller is running, the PodCIDR fields are populated

```
apiVersion: v1
kind: Node
metadata:
  name: node1
spec:
podCIDR: 10.0.0.0/24
...
status:
  addresses:
  - address: 192.168.0.5
type: InternalIP
  conditions:
  - address: 192.168.0.5
type: InternalIP
```
One of the runtime checks is “NetworkReady”, this is implemented on the container runtimes just as a check that a configuration file for the CNI exists.

Kubelet does not block on startup, it runs the apiserver and runtime checks periodically on independent goroutines and starts to process events. The Node is not declared Ready until all the checks are OK.
Kubernetes networking: Pods

```yaml
apiVersion: v1
class: Pod
metadata:
  name: app-http
  namespace: default
spec:
  containers:
    - name: app
      image: myapp:0.1
  ...
status:
  phase: Running
  podIP: 10.0.0.12
```
Kubernetes networking: Pods

1. Users creates a Pod

2. Kube-scheduler watches Pods and Nodes and schedule the Pod into a specific Node

3. Kubelet receives the Pod and uses CRI-API to communicate with the container runtime to create the Pod

apiVersion: v1
kind: Pod
metadata:
  name: app-http
  namespace: default
spec:
  containers:
  - name: app
    image: myapp:0.1

status:
  phase: Running
  podIP: 10.0.0.12
Kubernetes networking: CRI

REST API: Kubelet - Kube-apiserver

CRI API: Kubelet - Container Runtime

1. Pod
2. RunPodSandbox
3. Network Setup
4. sandboxID, err
5. PodSandboxStatus
6. PodSandboxStatusResponse, err
7. Pod.Status.PodIPs

RunPodSandboxRequest.PodSandboxConfig
- DNSConfig
- PortMapping
- Hostname

PodSandboxStatusResponse.Network
- Ip
- AdditionalIPs
Kubernetes networking: Services

- K8s cluster
- Service Network
- Pod Network
- Node Network
- kubernetes.default
Kubernetes Services: ClusterIP

```
apiVersion: v1
kind: Service
metadata:
  name: service
spec:
  clusterIP: 10.96.0.1
  clusterIPs:
  - 10.96.0.1
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: 6443
  type: ClusterIP

apiVersion: apps/v1
kind: Deployment
metadata:
  name: server-deployment
labels:
  app: MyApp
spec:
  replicas: 2
  selector:
    matchLabels:
      app: MyApp
  template:
    metadata:
      labels:
        app: MyApp
    spec:
      terminationGracePeriodSeconds: 30
      containers:
      - name: agnhost
        image: k8s.gcr.io/e2e-test-images/agnhost:2.39
        args:
        - netexec
        - --http-port=80
```
Kubernetes Services: NodePort

apiVersion: v1
kind: Service
metadata:
  name: service
spec:
  clusterIP: 10.96.0.1
  clusterIPs:
  - 10.96.0.1
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: 6443
  nodePort: 31023
  type: NodePort

apiVersion: apps/v1
kind: Deployment
metadata:
  name: server-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: MyApp
  template:
    metadata:
      labels:
        app: MyApp
    spec:
      terminationGracePeriodSeconds: 30
      containers:
      - name: agnhost
        image: k8s.gcr.io/e2e-test-images/agnhost:2.39
        args:
        - netexec
        - --http-port=80

NodeIP:NodePort

Node: node

SVC: svc

Pod: pod

Pod: pod

Node: node

Google
Kubernetes Services: Load Balancer

apiVersion: v1
kind: Service
metadata:
  name: service
spec:
  clusterIP: 10.96.0.1
  clusterIPs:
  - 10.96.0.1
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: 6443
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 202.34.23.12

apiVersion: apps/v1
kind: Deployment
metadata:
  name: server-deployment
labels:
  app: MyApp
spec:
  replicas: 2
  selector:
    matchLabels:
      app: MyApp
  template:
    metadata:
      labels:
        app: MyApp
    spec:
      terminationGracePeriodSeconds: 30
      containers:
      - name: agnhost
        image: k8s.gcr.io/e2e-test-images/agnhost:2.39
        args:
        - netexec
        - --http-port=80
Kubernetes Services: Headless

apiVersion: v1
kind: Service
metadata:
  name: headless
spec:
  clusterIP: None
  clusterIPs:
  - None
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: 6443
  type: ClusterIP

apiVersion: apps/v1
kind: Deployment
metadata:
  name: server-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: MyApp
  template:
    metadata:
      labels:
        app: MyApp
    spec:
      terminationGracePeriodSeconds: 30
      containers:
      - name: agnhost
        image: k8s.gcr.io/e2e-test-images/agnhost:2.39
        args:
        - netexec
        --http-port=80
Kubernetes Services

1. Kubelet creates the Pod

2. Kubelet updates the Pod object with the status and IPs


4. If the Service selector matches the Pod an EP/EPS is generated

5. Kube-proxy watches Services and EndpointSlices and installs the corresponding rules on the host

apiVersion: v1
kind: Service
metadata:
  name: lb-app
  namespace: default
spec:
  clusterIP: 10.96.0.13
  selector:
    app: MyApp
  type: ClusterIP
  ports:
  - name: port8080
    port: 8080
    protocol: TCP
    targetPort: 8080
Health Check starts failing to indicate the Load Balancer to not send more traffic to this node.

Pods is Terminating, is is able to handle connections during the grace period.

The Service keeps forwarding traffic since is the only Pod in the node despite is Terminating.
Health Check starts to pass once the new pod is running and ready.

New Pod replaces the previous one with a new version.
Kubernetes Ingress == L7 Load Balancer

```yaml
apiVersion: v1
kind: Ingress
metadata:
  name: minimal-ingress
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /foo
        pathType: Prefix
        backend:
          service:
            name: myapp
            port:
              number: 80
      - path: /bar
        pathType: Prefix
        backend:
          service:
            name: my-second-app
            port:
              number: 80
```

GET /foo

GET /bar
Gateway API aims to standardize the space

- A modern set of APIs for deploying L4 and L7 routing in Kubernetes
- Designed to be generic, expressive, extensible, and role-oriented

Diagram:
- GatewayClass
- Gateway
  - HTTPRoute
  - Service
  - Service
- Infrastructure provider
- Cluster operator
- Application developer
- Application developer
<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>This KEP has not yet been confirmed as something we want to do.</td>
</tr>
<tr>
<td>Alpha</td>
<td>This KEP is designated &quot;alpha&quot; in a specific Kubernetes release.</td>
</tr>
<tr>
<td>Beta</td>
<td>This KEP is designated &quot;beta&quot; in a specific Kubernetes release.</td>
</tr>
<tr>
<td>GA</td>
<td>This KEP is designated &quot;GA&quot; in a specific Kubernetes release.</td>
</tr>
</tbody>
</table>

### Pre-Alpha
- Enhancements #1036: Multi-Network
- Enhancements #1038: Graduated kube-proxy ComponentConfig to v1beta1
- Enhancements #1039: [CN] Kubernetes network reimagined (interface)
- Enhancements #1041: kube-proxy-extended ingress connectivity reliability
- Enhancements #1042: kube-proxy-extended status nodeInfo kubeProxyVersion field
- Enhancements #1043: kube-proxy-extended config
- Enhancements #1044: kube-proxy-extended config

### Alpha
- Enhancements #1050: Make Kubernetes aware of the LoadBalancer behavior
- Enhancements #1051: Multi-Service CIDRs

### Beta
- Enhancements #1051: Field status.should be removed from IP
- Enhancements #1052: Remove transit node predicates from XCM's service controller

### GA
- Enhancements #1053: Support Raft for dynamic and static allocation
- Enhancements #1054: Support Raft for dynamic and static allocation
- Enhancements #1055: Support Raft for dynamic and static allocation
- Enhancements #1056: Support Raft for dynamic and static allocation