MicroService Based Architecture: Kubernetes Basic Service Model

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs434/

04/21/2021
Admin

Projects update

- Week 3 (Apr 16 - Apr 22): Proposal iteration; reading more related work/systems
  - Week 4 (Apr 23 - Apr 29): Prototyping; Mid-point checkpoint; meet w/ the instructor
  - Week 5 (Apr 30 - May 6): Refinement; iterations
  - Week 6 (May 7 - May 13): Final implementation, final report (6-8 pages)
Recap: OLSRv2 Building Block: Local Neighborhood Discovery (HELLO)

- A good methodology of designing a distributed protocol
  - write down the computation using state variables at different nodes
  - protocol is then to share the distributed state variables

- Local state variables at each node n
  - N1_rcv(n): the set of nodes whose direct broadcast n can receive
  - N1(n): n’s 1-hop neighbors using symmetric links
  - N2(n): n’s 2-hop neighbors using symmetric links

- Relationship among (distributed) state variables
  \[
  N1(n) = \{ x \in N1_{rcv}(n) : n \in N1_{rcv}(x) \} \\
  N2(n) = \text{union}\{N1(x) : x \in N1(n)\} \setminus N1(n)
  \]

=> Neighbors share N1_rcv and N1
Recap: Cellular/5G Data Path Handling Roaming

- Using flooding of link state to handle roaming (mobility) is not scalable in large-scale networks

Cellular network key observation: the infrastructure is fixed, only the UEs are mobile

Solution:
- No need to flood the states of the UE devices
- Handle mobility by involving only ingress/egress nodes by creating tunnels (i.e., where the egress is)
- Tunnels set up by control plane
Recap: 5GC Control Plane (CP) Architecture

- Separation of data plane and control plane
- Control plane uses a Service Base Architecture (SBA):
  - each network function (NF) in CP exposes HTTP/2 ReSTful APIs.
  - Separation of Compute and Data to enable stateless Network Functions (NFs) in the Control Plane for scalability and resilience
Networking System Journey

- Routers in a network need to discover attached networks
  - Basic link-state broadcast
- Basic link state is not scalable (multiaccess domain)
  - Introduce networks in graph model, elect designed router, neighboring -> adjacency
- Basic link state is not scalable (large graph)
  - Divide into areas, with a backbone connecting areas; abstracting other areas
- Basic link state transport is not reliable, persistent
  - State synchronization, reliable transport
- Link state is not scalable in large, global setting; does not support decentralization/heterogenous/local decisions
  - Introduce one more abstraction level (autonomous system); link state routing -> policy path vector routing
- Generic decentralization/local decision can be unstable
  - Internet economics provides the “invisible hand” (not a contribution of network system design)
- Internet link state flooding is inefficient in mobile networks
  - Discover and distribute Flooding MPR, routing MPR to reduce overhead
- Update networking system due to mobility is not scalable in large wireless (e.g., cellular) networks
  - Limit mobility-awareness only to ingress and egress nodes, by establishing tunnels
- Provide generic, extensible signaling systems in mobile, cellular networking
  - Extensible, Service oriented architecture in 5G core
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
  - overview
**Motivation**

- The main current implementation of 5gc control plane of 5gc is Kubernetes (k8s), a microservices architecture
  - 5gc NFs run as k8s services
  - Scaling, reliability, management
- Kubernetes can manage other services for other (not-networking) systems as well
- The design and implementation of k8s is a good example of
  - Service based architecture
  - Implementation of itself
K8s Basic Entities
Kubernetes Basic Architecture

- A cluster management system for container based services
5GC Architecture vs Kubernetes Architecture

5gc CP

Kubernetes: A cluster management system
K8s Basic Concepts

- Containers
- Workloads
  - Pods
  - Workload resources: Deployment, ReplicaSet, DaemonSet ...
- Services, load balancing and networking
  - Service, topology-aware traffic routing with topology keys, DNS for services and pods, ...
- Storage
- Scheduling
- Configuration
Outline

- Admin and recap
- Microservice oriented architecture – Kubernetes
  - Overview
  - Basic model
    - Containers
Containers

- Light-weight execution isolation mechanism
  - Realized by Linux namespace and cgroup

- Good references
  - Docker using cgroups, namespace and beyond
    [https://www.youtube.com/watch?v=sK5i-N34im8](https://www.youtube.com/watch?v=sK5i-N34im8)
  - Container networking internal
    - [https://www.youtube.com/watch?v=6v_BDHlgoY8](https://www.youtube.com/watch?v=6v_BDHlgoY8)
Demo: DockerCoin App

- Demo: dockercoin application (see schedule page for link to the base tutorial)
  - Function
    - generate a few random bytes
    - hash these bytes and check if a hit
    - increment a counter (to keep track of speed)
    - repeat forever!
  - Realization: divide into containers
DockerCoins Containers Details

- 5 containers
  - redis (keep track of states e.g., counter, found coins)
    - using an official image from the Docker Hub
  - rng, hasher, worker, webui are each built from a Dockerfile

- worker is the main orchestrator
  - invokes web service rng to generate random bytes
  - invokes web service hasher to hash these bytes
  - every second, worker updates redis to indicate how many loops were done

- webui queries redis, and computes and exposes "hashing speed" in the browser
Demo

- See source code of containers
  - Slides: https://qconuk2019.container.training/#35
  - Git: git clone https://github.com/jpetazzo/container.training
  - %cd dockercoins
DockerCoins Application using 5 Containers
Demo

- **Execute dockers**
  - Start all dockers
    - `%docker-compose up`
  - See dockers running (a separate terminal)
    - `%docker-compose ps`
  - See webui results
    - [http://0.0.0.0:8000/index.html](http://0.0.0.0:8000/index.html)
  - Shutdown all containers
    - `%docker-compose down`
Discussion

- Issues one may encounter running the application?
Example Kubernetes Services

- Start 5 containers using image atseashop/api:v1.3
- Place an internal load balancer in front of these containers
- Start 10 containers using image atseashop/webfront:v1.3
- Place a public load balancer in front of these containers
- It's Black Friday (or Christmas), traffic spikes, grow our cluster and add containers
- New release! Replace containers with the new image atseashop/webfront:v1.4
- Keep processing requests during the upgrade; update containers one at a time
Some Other Things Kubernetes Can Do

- Basic autoscaling
- Blue/green deployment, canary deployment
- Long running services, but also batch (one-off) jobs
- Overcommit our cluster and *evict* low-priority jobs
- Run services with *stateful* data (databases etc.)
- Fine-grained access control defining *what* can be done by *whom* on *which* resources
- Integrating third party services (*service catalog*)
- Automating complex tasks (*operators*)
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
  - Overview
  - Basic model
    - Containers and example k8s services
    - Workloads
      - pods
pods

- *A pod* (as in a pod of whales or pea pod) is the smallest deployable units of computing that one can create and manage in Kubernetes.
  - https://kubernetes.io/docs/concepts/workloads/pods/

- A Pod models an application-specific "logical host": it contains one or more application containers which are relatively tightly coupled

- Pod's contents are always co-located and co-scheduled, and run in a shared storage and network context.
Pods (Offline)

- Pods in a Kubernetes cluster are used in two main ways:
  - **Pods that run a single container.** The "one-container-per-Pod" model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container; Kubernetes manages Pods rather than managing the containers directly.
  - **Pods that run multiple containers that need to work together.** A Pod can encapsulate an application composed of multiple co-located containers that are tightly coupled and need to share resources. These co-located containers form a single cohesive unit of service—for example, one container serving data stored in a shared volume to the public, while a separate *sidecar* container refreshes or updates those files. The Pod wraps these containers, storage resources, and an ephemeral network identity together as a single unit.
Demo (Start a pod)

- Start a single pod that consists of a single container using command line
  - // check existing pods
    kubectl get all
  - // watch pods set
    kubectl get pod -w
  - // start a pod using container alpine, ping 1.1.1.1 DNS server
    kubectl run pingpong --image alpine ping 1.1.1.1
  - kubectl logs pod/pingpong --tail 1 -follow
  - kubectl delete pod/pingpong // remove the pod
Background

- `kubectl run pingpong --image alpine` calls `docker run alpine`
- If the Engine needs to pull the alpine image, it expands it into `library/alpine`
- `library/alpine` is expanded into `index.docker.io/library/alpine`
- To use some other registry than `index.docker.io`, specify it in the image name, e.g.,
  - `docker pull gcr.io/google-containers/alpine-with-bash:1.0`
Demo (starting a pod)

- Starting a pod using a .yaml file (using a job)
  - kubectl apply -f pod-echo.yaml
  - Kubectl get all // see if started
  - Kubectl logs pod/... // see outcome
  - Kubectl delete ...
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
  - Overview
  - Basic model
    - Containers and example k8s services
    - Workloads
      - pod
      - deployment
Deployment and ReplicaSet

- A Deployment provides a declarative specification about a desired state, and the Deployment Controller changes the actual state to the desired state at a controlled rate.

- A deployment is a high-level construct
  - allows scaling, rolling updates, rollbacks
  - multiple deployments can be used together to implement a canary deployment
  - delegates pods management to replica sets

- A replica set is a low-level construct
  - makes sure that a given number of identical pods are running
  - allows scaling
  - rarely used directly

- A replication controller is the (deprecated) predecessor of a replica set
Demo: Deployment

Start a deployment of http server
- // window 1: watch pods being created
  - kubectl get pods -w
- // window 2, // a web server image called httpenv
  - kubectl create deployment httpenv --image=jpetazzo/httpenv
  - kubectl scale deployment httpenv --replicas=3
  - kubectl get all

Deployment management
- // Kill one pod and see the reaction
  - Kubectl delete pod/httpenv-xxxx
  - Kubectl delete deploy/httpenv
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
  - Overview
  - Basic model
    - Containers and example k8s services
    - Workloads
      - pod
      - deployment
      - services
K8s Services

- A service is a stable address for a pod (or a bunch of pods)
  - If you want to connect to the pod(s), you need to create a service
  - Once a service is created, CoreDNS will allow us to resolve it by name (i.e. after creating service hello, the name hello will resolve to something)

- There are different types of services
  - ClusterIP, NodePort, LoadBalancer, ExternalName
**K8s Services**

- **ClusterIP (default type)**
  - a virtual IP address is allocated for the service (in an internal, private range)
  - this IP address is reachable only from within the cluster (nodes and pods)
  - your code can connect to the service using the original port number

- **NodePort**
  - a port is allocated for the service (by default, in the 30000-32768 range)
  - that port is made available *on all the nodes* and anybody can connect to it
  - the code must be changed to connect to that new port number
K8s Services (Offline)

- **LoadBalancer**
  - an external load balancer is allocated for the service
  - the load balancer is configured accordingly (e.g.: a NodePort service is created, and the load balancer sends traffic to that port)
  - available only when the underlying infrastructure provides some "load balancer as a service" (e.g. AWS, Azure, GCE, OpenStack...)

- **ExternalName**
  - the DNS entry managed by CoreDNS will just be a CNAME to a provided record
  - no port, no IP address, no nothing else is allocated
Node, Container, Pod, Service (IP addr)
Demo

- Expose the HTTP port of our server:
  - kubectl expose deployment httpenv --port 8888

- Check service created
  - kubectl get all

- Or look up which IP address was allocated:
  - kubectl get service

- Check the endpoints of the service
  - kubectl describe service httpenv
  - kubectl describe endpoints httpenv
  - kubectl get pods -l app=httpenv -o wide
Try the DockerCoin App Using k8s

- // the dockers are already created
dockercoins/<name>:v0.1
  see tutorial on creating docker images and host at registry

- kubectl create deployment redis --image=redis
- kubectl create deployment worker
  --image=dockercoins/worker:v0.1
- Same for hasher rng webui worker
Try the DockerCoin App Using k8s

- kubectl logs deploy/rng
- kubectl logs deploy/worker

- Error (not expose)
  - kubectl expose deployment redis --port 6379
  - kubectl expose deployment rng --port 80
  - kubectl expose deployment hasher --port 80
Try the DockerCoin App Using k8s

- Create a NodePort service for the Web UI:
  - `kubectl expose deploy/webui --type=NodePort --port=80`

- Check the port that was allocated:
  - `kubectl get svc`

- See webui
  - `http://0.0.0.0:31207/index.html`
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
  - Overview
  - Basic model
    - Containers and example k8s services
    - Workloads
      - pod
      - deployment
      - Services
  - k8s implementation
Kubernetes Cluster

- **Kubectl cluster-info**
  - Kubernetes master is running at [https://kubernetes.docker.internal:6443](https://kubernetes.docker.internal:6443)
    - MacOS
      - `dscacheutil -q host -a name kubernetes.docker.internal`
  - KubeDNS is running at [https://kubernetes.docker.internal:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy](https://kubernetes.docker.internal:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy)

- `kubectl -n kube-system get all`
Kubernetes Architecture

Exercise

- What are the functions of each component in the master?
- What are the main challenge of each function?
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
  - Overview
  - Basic model
  - k8s implementation
    - etcd and RAFT protocol
etcd

- **etcd** is a strongly consistent, distributed key-value store that provides a reliable way to store data that needs to be accessed by a distributed system or cluster
- **etcd** builds on top of the RAFT consensus protocol
- **See**
  - [https://blog.containership.io/etcd/](https://blog.containership.io/etcd/) for etcd and RAFT, in particular the visual guide
  - [https://github.com/etcd-io/etcd](https://github.com/etcd-io/etcd)
  - [https://etcd.io/docs/v3.4.0/learning/](https://etcd.io/docs/v3.4.0/learning/)