Service Based Architecture: Kubernetes Implementations

Y. Richard Yang

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

05/03/2021
Admin

- Projects update
  - Week 5 (Apr 30 - May 6): Refinement; iterations
  - Week 6 (May 7 - May 13): Final implementation, final report (6-8 pages)
Recap: Kubernetes Implementation Architecture
Recap: Raft Consensus Protocol

- **An asymmetric, leader-based protocol**
  - At any given time, one server is in charge, others accept its decisions
  - Clients communicate with the leader

- **Reliability based on replicated operations log**
  - Replication => surviving leader crash (a server has state when old leader crashes and this server becomes the new leader)

- **Leader’s log is “the truth”**
  - If a leader has decided that a log entry is committed, that entry will be present in the logs of all future leaders

- **Consistency based on a combination of leader election + commitment rules**
  - Ensures switching leader will not change the ordering of ops applied; partition will not lead to system diversion
Recap: Raft Election
(Server Detects Leader Down)

- Increment current term
- Change to Candidate state
- Vote for self
- Send RequestVote RPCs to all other servers, retry until one of the following:
  1. Receive votes from majority of servers:
     - Become leader
     - Send AppendEntries heartbeats to all other servers
  2. Receive RPC from valid leader:
     - Return to follower state
  3. No-one wins election (election timeout elapses):
     - Increment term, start new election
Recap: Raft Leader Election Rule

- During elections, choose candidate with log from newer terms
  - Candidates include log info in RequestVote RPCs (index & term of last log entry)
  - Voting server V denies vote if its log is “more complete”:
    \[(\text{lastTerm}_V > \text{lastTerm}_C) \lor (\text{lastTerm}_V == \text{lastTerm}_C) \land (\text{lastIndex}_V > \text{lastIndex}_C)\]
  - Leader will likely have “most complete” log among electing majority
Recap: Leader Commitment and Exec Rule

- Client sends commands to leader
- Leader appends command to its log
- Leader **sends** AppendEntries commands to followers and blocks until **commit**:  
  - Must be stored on a **majority** of servers  
  - At least one new entry from leader’s term must also be stored on majority of servers
- Leader **executes** command to its state machine, returns result to client
Exercise (Same Term)

- Which logs can \( s_1 \) commit?
- Who may become next leader if \( s_1 \) crashes or is disconnected?
- Sanity check: If a log is committed by the leader, the next leader should have the entry committed as well.
Exercise (New Term)

- Which logs can $s1$ commit?
- Who may become next leader if $s1$ crashes or is disconnected?
- Sanity check: If a log is committed by the leader, the next leader should have the entry committed as well.
Exercise (New Term)

- Which logs can s1 commit?
- Who may become next leader if s1 crashes or is disconnected?
- Sanity check: If a log is committed by the leader, the next leader should have the entry committed as well.

```

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Leader for term 4
```
Neutralizing Old Leaders

- **Deposed leader may not be dead:**
  - Temporarily disconnected from network
  - Other servers elect a new leader
  - Old leader becomes reconnected, attempts to commit log entries

- **Terms used to detect stale leaders (and candidates):**
  - Every RPC contains term of sender
  - If sender's term is older, RPC is rejected, sender reverts to follower and updates its term
  - If receiver's term is older, it reverts to follower, updates its term, then processes RPC normally

- **Election updates terms of majority of servers**
  - Deposed server cannot commit new log entries
Leader changes can result in log inconsistencies:

Possible followers:

- (a) 1 1 1 4 4 5 5 6 6 6
- (b) 1 1 1 4
- (c) 1 1 1 4 4 5 5 6 6 6 6 6
- (d) 1 1 1 4 4 5 5 6 6 6 7 7
- (e) 1 1 1 4 4 4
- (f) 1 1 1 2 2 2 3 3 3 3 3 3

Implication: New leader must make follower logs consistent with its own
- Delete extraneous entries
- Fill in missing entries
Raft Log Consistency Invariant

High level of coherency between logs:

- If log entries on different servers have same index and term:
  - They store the same command
  - The logs are identical in all preceding entries

- If a given entry is committed, all preceding entries are also committed
AppendEntries Consistency Check

- Each AppendEntries RPC contains index, term of entry preceding new ones
- Follower must contain matching entry; otherwise it rejects request (see later for how this is used)
- Implements an induction step, ensures coherency

![Diagram](image-url)
Repairing Follower Logs

- Leader keeps nextIndex for each follower:
  - Index of next log entry to send to that follower
  - Initialized to (1 + leader’s last index)
- When AppendEntries consistency check fails, decrement nextIndex and try again:
When follower overwrites inconsistent entry, it deletes all subsequent entries:

<table>
<thead>
<tr>
<th>log index</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>leader for term 7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>follower (before)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>follower (after)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- When a follower overwrites an inconsistent entry, it deletes all subsequent entries.
- The leader's log index for term 7 is updated accordingly.
- The follower's log index is also updated to reflect the deletion of entries.
Introducing Clients: Raft as Client/Server Protocol

- **Client sends commands to leader**
  - If leader unknown, contact any server
  - If contacted server not leader, it will redirect to leader
  - If request times out (e.g., leader crash):
    - Client reissues command to some other server
    - Eventually redirected to new leader
    - Retry request with new leader

- **Leader response**
  - Leader does not respond until command has been logged, committed, and executed by leader’s state machine
Introducing Clients: Raft as Client/Server Protocol

- Problem: a leader crashes after executing command, but before responding
  - Must not execute command twice

- Solution: client embeds a unique id in each command
  - Server includes id in log entry
  - Before accepting command, leader checks its log for entry with that id
  - If id found in log, ignore new command, return response from old command

- Result: exactly-once semantics as long as client doesn’t crash
## Raft Protocol Summary

### Followers
- Respond to RPCs from candidates and leaders.
- Convert to candidate if election timeout elapses without either:
  - Receiving valid AppendEntries RPC, or
  - Granting vote to candidate

### Candidates
- Increment currentTerm, vote for self
- Reset election timeout
- Send RequestVote RPCs to all other servers, wait for either:
  - Votes received from majority of servers: become leader
  - AppendEntries RPC received from new leader: step down
- Election timeout elapses without election resolution: increment term, wait random time, start new election
- Discover higher term: step down

### Leaders
- Initialize nextIndex for each to last log index + 1
- Send initial empty AppendEntries RPCs (heartbeat) to each follower; repeat during idle periods to prevent election timeouts
- Accept commands from clients, append new entries to local log
- Whenever last log index ≥ nextIndex for a follower, send AppendEntries RPC with log entries starting at nextIndex, update nextIndex if successful
- If AppendEntries fails because of log inconsistency, decrement nextIndex and retry
- Mark log entries committed if stored on a majority of servers and at least one entry from current term is stored on a majority of servers
- Step down if currentTerm changes

### RequestVote RPC
**Invoked by candidates to gather votes.**

**Arguments:**
- candidateId: candidate requesting vote
- term: candidate's term
- lastLogIndex: index of candidate's last log entry
- lastLogTerm: term of candidate's last log entry

**Results:**
- term: currentTerm, for candidate to update itself
- voteGranted: true means candidate received vote

**Implementation:**
1. If term > currentTerm, currentTerm ← term (step down if leader or candidate)
2. If term = currentTerm, votedFor is null or candidateId, and candidate's log is at least as complete as local log, grant vote and reset election timeout

### AppendEntries RPC
**Invoked by leader to replicate log entries and discover inconsistencies; also used as heartbeat.**

**Arguments:**
- term: leader's term
- leaderId: so follower can redirect clients
- prevLogIndex: index of log entry immediately preceding new ones
- prevLogTerm: term of prevLogIndex entry
- entries: log entries to store (empty for heartbeat)
- commitIndex: last entry known to be committed

**Results:**
- term: currentTerm, for leader to update itself
- success: true if follower contained entry matching prevLogIndex and prevLogTerm

**Implementation:**
1. Return if term < currentTerm
2. If term = currentTerm, currentTerm ← term
3. If candidate or leader, step down
4. Reset election timeout
5. Return failure if log doesn’t contain an entry at prevLogIndex whose term matches prevLogTerm
6. If existing entries conflict with new entries, delete all existing entries starting with first conflicting entry
7. Append any new entries not already in the log
8. Advance state machine with newly committed entries

### Log Entry
- term: term when entry was received by leader
- index: position of entry in the log
- command: command for state machine

---

### Persistent State
Each server persists the following to stable storage synchronously before responding to RPCs:
- currentTerm: latest term server has seen (initialized to 0 on first boot)
- votedFor: candidateId that received vote in current term (or null if none)
- log: log entries
Offline: Configuration Changes

- System configuration:
  - ID, address for each server
  - Determines what constitutes a majority

- Consensus mechanism must support changes in the configuration:
  - Replace failed machine
  - Change degree of replication
Cannot switch directly from one configuration to another: conflicting majorities could arise
Offline: Joint Consensus

- Raft uses a 2-phase approach:
  - Intermediate phase uses **joint consensus** (need majority of both old and new configurations for elections, commitment)
  - Configuration change is just a log entry; applied immediately on receipt (committed or not)
  - Once joint consensus is committed, begin replicating log entry for final configuration
Additional details:
- Any server from either configuration can serve as leader
- If current leader is not in $C_{\text{new}}$, must step down once $C_{\text{new}}$ is committed.

Diagram:
- $C_{\text{old}}$ can make unilateral decisions
- $C_{\text{old+new}}$ can make unilateral decisions
- $C_{\text{new}}$ can make unilateral decisions
- $C_{\text{old+new}}$ entry committed
- $C_{\text{new}}$ entry committed
- Leader not in $C_{\text{new}}$ steps down here
Raft Summary

1. Basic leader election:
   - Select one of the servers to act as leader
   - Detect crashes, choose new leader
2. Basic log replication (Normal operation)
3. Revised leader election and log replication for safety and consistency
4. Neutralizing old leaders, and repairing follower’s log
5. Client interactions
6. Configuration changes:
Outline

- Admin and recap
- Microservice oriented architecture – Kubernetes
  - Overview
  - Basic model
  - Kubernetes implementations
    - Reliable coordination/configuration data store implementation
    - Controller-manager/scheduler implementation
Kubernetes Implementation Architecture
Controller-Manager+Scheduler+Etcd Bigger Picture

1. Create ReplicaSet
2. Create ReplicaSet
3. Notify create Replica Set event
4. Notify create Replica Set event
5. Create Pod
6. Create Pod
7. Notify create Pod event
8. Notify create Pod event
9. Modify Pod
10. Modify Pod
11. Notify modify Pod event

0. Watch Replica Set
0. Watch Pod(dest Node=""")
0. Watch Pod(dest Node=“Node1”)
0. Watch Pod(dest Node=“Node1”)

kube-apiserver

kubectl
kube-controller-manager
kube-scheduler
kubelet
Kubernetes Implementation Architecture

Discussion: Alternative designs? Benefits of current (data-store+api-server) design?
Controller Managers

Controller Manager

- Replication Controller
- Node Controller
- ResourceQuota Controller
- Namespace Controller
- ServiceAccount Controller
- Token Controller
- Service Controller
- Endpoint Controller
Example: Endpoint Controller
Exercise

- Function of scheduler: match node to pod
- Exercise: How may you design the scheduler?
Scheduler
Scheduler Predicates

- NoDiskConflict
- PodFitsResources
- PodSelectorMatches
- PodFitsHost
- CheckNodeLabelPresence
- CheckServiceAffinity
- PodFitsPorts

- Default loaded
  - PodFitsPorts(PodFitsPorts)
  - PodFitsResources(
    PodFitsResources)
  - NoDiskConflict(
    NoDiskConflict)
  - MatchNodeSelector(
    PodSelectorMatches)
  - HostName(PodFitsHost)
Ranking Algorithms

- **LeastRequestedPriority:**
  
  \[
  \text{score} = \text{int}\left( \frac{(\text{nodeCpuCapacity} - \text{totalMilliCPU})*10}{\text{nodeCpuCapacity}} + \frac{(\text{nodeMemoryCapacity} - \text{totalMemory})*10}{\text{nodeCpuMemory}} \right) / 2
  \]

- **CalculateNodeLabelPriority:**

- **BalancedResourceAllocation:**

  \[
  \text{score} = \text{int}\left( 10 - \text{math.abs}\left( \frac{\text{totalMilliCPU}}{\text{nodeCpuCapacity}} - \frac{\text{totalMemory}}{\text{nodeMemoryCapacity}} \right) \times 10 \right)
  \]
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
  - Overview
  - Basic model
  - Kubernetes implementation
    - Reliable coordination/configuration data store implementation
    - Controller-manager/scheduler implementation
    - Networking/pod implementation
      - Networking background
Overview

- Why:
  - it may look very complex if one does not understand it
  - it puts many early parts of the class (transport, routing systems, request routing) into a bigger context
  - it gives us a chance to cover Linux kernel networking, which is on the original course planning
  - It gives us a chance to look at network system in virtualization, a major trend of computer systems
Background: Namespace

- Container is just a higher abstraction built on top of Linux namespace for isolation
- Each process has its set of namespaces
- Demo: show namespaces of a process
  - ls -l /proc/<pid>/ns

```
$ ls -l /proc/$$/ns
  total 0
  lrwxrwxrwx. 1 dj dj 0 Jan 4 04:12 ipc -> ipc:[4026531839]
lrwxrwxrwx. 1 dj dj 0 Jan 4 04:12 mnt -> mnt:[4026531840]
lrwxrwxrwx. 1 dj dj 0 Jan 4 04:12 net -> net:[4026531956]
lrwxrwxrwx. 1 dj dj 0 Jan 4 04:12 pid -> pid:[4026531836]
lrwxrwxrwx. 1 dj dj 0 Jan 4 04:12 user -> user:[4026531837]
lrwxrwxrwx. 1 dj dj 0 Jan 4 04:12 uts -> uts:[4026531838]
```
Namespace System Calls

- **Three system calls**

  ```c
  int clone(int (*child_func)(void *), void *child_stack, int flags, void *arg);
  ```

  ```c
  int setns(int fd, int nstype);
  ```

  ```c
  int unshare(int flags);
  ```
execv(child_args[0], child_args);
printf("Doops\n");
return 1;
}

int main()
{
    // init sync primitive
    pipe(checkpoint);
    printf(" - [%5d] Hello ?\n", getpid());

    int child_pid = clone(child_main, child_stack+STACK_SIZE,
                          CLONE_NEWUTS | CLONE_NEWIPC | CLONE_NEWPID | CLONE_NEWNS | CLONE_NEWNET | SIGCHLD, NULL);

    // further init: create a veth pair
    char* cmd;
    asprintf(&cmd, "ip link set veth1 netns %d", child_pid);
    system("ip link add veth0 type veth peer name veth1");
    system(cmd);
    system("ip link set veth0 up");
    system("ip addr add 169.254.1.1/30 dev veth0");
    free(cmd);

    // signal "done"
    close(checkpoint[1]);
    waitpid(child_pid, NULL, 0);
    return 0;
}
**Network Namespace**

- **Discussion:** What will you include in a network namespace?

- **Linux network namespace**
  - Devices
  - Routing tables
  - Netfilters
  - Networking stack (sockets, ...)

---

40
Basic Network Namespace: Demo

- Show network namespaces
  - `ip netns list`

- Create a network namespace
  - `ip netns add netns1`

- Show devices in the network namespace
  - `ip netns exec netns1 ip link list`
  - `ip netns exec netns1 ifconfig <dev>`

- Show routing tables/iptables/sockets
  - `ip netns exec netns1 route // or netstat -rn`
  - `ip netns exec netns1 iptables -L // or netstat -rn`
  - `ip netns exec netns1 netstat -p tcp`

- Ping localhost
  - `ip netns exec netns1 ping 127.0.0.1`

- Contrast w/ default name space
Basic Network Namespace: Demo

- Turn up lo
  - `ip netns exec netns1 ip link set dev lo up` // or `ifconfig`  

- Ping
  - `ip netns exec netns1 ping 127.0.0.1`
  - `ip netns exec netns1 ifconfig lo`

- Delete network namespace [if you want, but not now]
  - `ip netns delete netns1`
Out of Network Namespace “Jail”: veth

- A pair of virtual ethernet devices
  - Packet enters one end will appear in the other end
  - A common way to allow a namespace to communicate outside

- Source code: drivers/net/veth.c
Linux veth Pair driver: Demo

- **Create veth pair**
  - `ip link add veth0 type veth peer name veth1`
  - `ip link list // list devices in default namespace`

- **Move one of them to different namespace**
  - `ip link set veth1 netns netns1`
  - `ip link list // devices in default namespace`
  - `ip netns exec netns1 ip link list // devices in netns1`

- **Assign IP addresses and turn on devices**
  - `ip netns exec netns1 netstat -rn // see routing table`
  - `ip netns exec netns1 ifconfig veth1 10.1.1.1/24 up // check routing again`
  - `ip netns exec netns1 ifconfig veth1`
  - **Exercise:** assign veth0 10.1.1.2 and turn on; compare routing tables of netns1 and root

- **Exercise:** in netns1 ping 10.1.1.2, and host 10.1.1.1

- **Note:** physical devices can only be in root namespace
Exercise

- If there are N network namespaces (e.g., those inside a k8s Pod) on a single node, how many veth pairs do we need to set up so that they all can communicate?
Alternative: Linux Bridge: Demo

- **Create a linux bridge**
  - `ip link add name br0 type bridge` // or `brctl addbr br0`

- **Add veth0 to the bridge**
  - `ip link set dev veth0 master br0` // or `brctl addif br0 veth0`

- **Display devices on br0**
  - `brctl show`
Exercise: Protocol Stack, Bridge, Devices Connection

- Exercise: protocol workflow
  - if we type
    - %ping 10.1.1.1 in root name space

- Verify
  - tcpdump -i veth0
  - Exercise: tcpdump -i veth1
  - tcpdump -i br0
Deeper Dive: Protocol Stack and Bridge Multiplexing

- **Problem**
  - According to routing table, protocol stack uses (sends and receives) veth0
  - veth0 forwards ARP reply to br0, not protocol stack

- **Fixing routing problem**
  - `ip addr del 10.1.1.2/24 dev veth0`
  - `ip addr add 10.1.1.2/24 dev br0`

Exercise: if we add a new network namespace netns3 with IP 10.1.1.3 and want it to communicate with netns1, how do we set up?

Offline exercise: if we add a new network namespace netns4 with IP 10.1.1.4 but want it to communicate only with netns1, not netns3.
Docker: Demo

- Start a container using docker
  - `ip link show` // or `ifconfig` to see current devices
  - `docker run -d nginx` // `ifconfig` to see devices again
  - `docker inspect <container id>` // find PID; see IP address, gateway
  - `ls -l /proc/<pid>/ns` // see the namespaces

- Docker networking
  - `brctl show` // Show all bridges
  - `ifconfig <bridge>`

- See docker container network namespace
  - `ip netns attach <name> <pid>`
  - `ip netns exec <name> ip link show`
Docker: Demo 2

- Start a container using docker with port mapping
  - `docker run -p 6789:80 -d nginx`
  - `docker inspect <container id> // find PID: see IP address, gateway`

- Docker networking addition
  - `iptables -t nat -nL`

- Source rewrite (to visit outside)

- Dest rewrite (port map)

```
# docker inspect 6f5 | grep Pid
"Pid": 15818,  
# 11 /proc/15818/ns/
```
```
total 0
lrwxrwxrwx 1 root root 0 Jan 10 16:16 ipc -> ipc:[4026535715]
lrwxrwxrwx 1 root root 0 Jan 10 16:16 mnt -> mnt:[4026535789]
lrwxrwxrwx 1 root root 0 Jan 10 16:16 net -> net:[4026535718]
lrwxrwxrwx 1 root root 0 Jan 10 16:16 pid -> pid:[4026535791]
lrwxrwxrwx 1 root root 0 Jan 10 16:16 user -> user:[4026531837]
lrwxrwxrwx 1 root root 0 Jan 10 16:16 uts -> uts:[4026535790]
```
iptables: a set of tables (nat, filter, mangle) in the processing flow, with each table consisting of a set chains
**iptables**

- Each table may define a chain for only a subset hook points (INPUT, OUTPUT, FORWARD, PREROUTING, POSTROUTING)
- `iptables -t <table> -L`
  - Default table is filter
  - Try nat
Additional Networking Components

- Tunnel: Allows user space applications to process packets
- Example: Design a VPN server that encapsulates packets to a given destination set (e.g., 192.168.1.0/24) using an external encapsulation
#include <net/if.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <sys/types.h>
#include <linux/if_tun.h>
#include <stdlib.h>
#include <stdio.h>

int tun_alloc(int flags)
{
    struct ifreq ifr;
    int fd, err;
    char *clonedev = "/dev/net/tun";

    if ((fd = open(clonedev, O_RDWR)) < 0) {
        return fd;
    }

    memset(&ifr, 0, sizeof(ifr));
    ifr.ifr_flags = flags;

    if ((err = ioctl(fd, TUNSETIFF, (void *) &ifr)) < 0) {
        close(fd);
        return err;
    }

    printf("Open tun/tap device: %s for reading...\n", ifr.ifr_name);

    return fd;
}

int main()
{
    int tun_fd, nread;
    char buffer[1500];

    /* Flags: IFF_TUN  - TUN device (no Ethernet headers)
     * IFF_TAP  - TAP device
     * IFF_NO_PI - Do not provide packet information
     */
    tun_fd = tun_alloc(IFF_TUN | IFF_NO_PI);

    if (tun_fd < 0) {
        perror("Allocating interface");
        exit(1);
    }

    while (1) {
        nread = read(tun_fd, buffer, sizeof(buffer));
        if (nread < 0) {
            perror("Reading from interface");
            close(tun_fd);
            exit(1);
        }
        printf("Read %d bytes from tun/tap device\n", nread);
    }

    return 0;
}
Tunnel Example: Execution and Configuration

- gcc tun.c -o my_tun
- ./my_tun // assume created tunnel 0 tun0; ip addr
- ip addr add 192.168.1.2/24 dev tun0
- ip link set tun0 up

Exercise: packet flow to ping 192.168.1.3
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
  - Overview
  - Basic model
  - Kubernetes implementation
    - Reliable coordination/configuration data store implementation
    - Controller-manager/scheduler implementation
    - Networking/pod implementation
      - Networking background
      - Basic Pod implementation
Pod Structure

- Demo (MacOS)
  - kubectl get pod
  - docker ps

```c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define STRINGIFY(s) #s
#define VERSION_STRING(a) STRINGIFY(a)

#ifndef VERSION
#define VERSION "" VERSION_STRING()"
#endif

static void sigsegv(int signo) {
    perror(signame, "Shutting down, got signal");
    exit(0);
}

static void sigterm(int signo) {
    while (waitpid(-1, NULL, WNOHANG) > 0)
}

int main(int argc, char **argv) {
    int i;
    for (i = 1; i < argc; ++i) {
        if (strcmp(argv[i], "-v")) {
            printf("\"%s\", VERSION_STRING(VERSION));
            return 0;
        }
    }
    if (getpid() == 1)
        /* Not an error because pause sees use outside of infra containers. */
        fprintf(stderr, "Warning: pause should be the first process's");
    if (sigaction(SIGINT, &struct sigaction{.sa_handler = sigsegv}, NULL) < 0)
        return 1;
    if (sigaction(SIGTERM, &struct sigaction{.sa_handler = sigterm}, NULL) < 0)
        return 3;
    if (sigaction(SIGHLD, &struct sigaction{.sa_handler = sigsegv,
        .sa_flags = SA_NOCLSTP},
        NULL) < 0)
        return 3;
    for (;;)
        pause();
        fprintf(stderr, "Error: infinite loop terminated\n");
        return 42;
}
```
Pod Construction (Structure)

Initialize Pause network namespace

Join Pause network ns

Init Pause cont. net device

Cross-node networking

Pod A

Container A

pause A

eth0

Pod B

Container B

pause B

eth0

Kubernetes

Master

Pod

Kubelet

CRI

dockershim

containerd

kubenet

CNI

network driver

p2p bridge

flannel calico
Simulate Pod using Docker (Structure)

```bash
# docker run -d --name pause gcr.io/google_containers/pause-amd64:3.0

# cat <<EOF >> nginx.conf
> error_log stderr;
> events { worker_connections 1024; }
> http {
>   access_log /dev/stdout combined;
>   server {
>     listen 80 default_server;
>     server_name example.com www.example.com;
>     location / {
>       proxy_pass http://127.0.0.1:2368;
>     }
>   }
> }
> EOF

# docker run -d --name ghost --net = container: pause --ipc = container: pause --pid = container: pause ghost

# docker run -d --name nginx -v `pwd`/nginx.conf:/etc/nginx/nginx.conf -p 8080:80 --net=container:pause --ipc=container:pause --pid=container:pause nginx
```
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
  - Overview
  - Basic model
  - Kubernetes implementation
    - Reliable coordination/configuration data store implementation
    - Controller-manager/scheduler implementation
    - Networking/pod implementation
      - Networking background
      - Basic Pod implementation
      - Kubernetes networking designs
Recap: Node, Container, Pod, K8s Requirements

- Pods are allocated IP addresses from a given range
- IP-Per-Pod: Each Pod has a single IP from the range
- Pods run in a direct-connect, flat address space
  - Containers in the same Pod share IP, same Linux network stack, and can connect using localhost
Settings

Node 1

Cont. 1

eth0

ethXXX

docker0 bridge

Routing

ip1

Cont. 2

eth0

ethXXX

docker0 bridge

Routing

ip2

ip3

Node 1

Same Pod

Cont1

Cont2

Shared Network ns

eth0

docker0 bridge

Routing

ip1

ip2

ip3
Setting

Node1

Pod1
Cont1 Cont2

Shared Network ns

eth0 IP1

ethXXX

docker0 bridge

Pod2
Cont1 Cont2

Shared Network ns

eth0 IP2

ethXXX

Routing

eth0 IP3
Setting

Discussion: Key Issues?
Key Issues (Cross Nodes)

- IP address management (ipam): potentially overlapping addresses if not careful
  - Key solution: Address allocation: in the whole Kubernetes cluster, Pod IP allocation should have no conflict

- Routing: Virtual IP and virtual MAC may not be recognized in the physical network
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
  - Overview
  - Basic model
  - Kubernetes implementation
    - Reliable coordination/configuration data store implementation
    - Controller-manager/scheduler implementation
    - Networking/pod implementation
      - Networking background
      - Basic Pod implementation
      - Kubernetes networking designs
        » flannel
Flannel Overview

- No server
  - There is one agent on each Node

- Coordinating using etcd to solve the ipam and routing issue (how?)
Flannel: Workflow (ipam)

- **Use etcd to coordinate**
  - Share a large network address space and post to etcd
    ```
    # etcdctl set /coreos.com/network/config '{ "Network": "172.17.0.0/16" }'
    ```
  - Partition the space and get a subnet
    ```
    # etcdctl ls /coreos.com/network/subnets
    /coreos.com/network/subnets/172.17.18.0-24
    /coreos.com/network/subnets/172.17.19.0-24
    /coreos.com/network/subnets/172.17.20.0-24
    ```

- **Modify node docker start parameter to allocate pod IP in the range, e.g.,**
  - --bip=172.17.18.0/24
Flannel: Workflow (Routing)

- How does a flannel agent/node know the routing
  - From etcd knows the prefix of each node
    - Build routing using the etcd data

```
# etcdctl ls /coreos.com/network/subnets
/coreos.com/network/subnets/172.17.18.0-24
/coreos.com/network/subnets/172.17.19.0-24
/coreos.com/network/subnets/172.17.20.0-24
```

```
# etcdctl get /coreos.com/network/subnets/172.17.18.0-24
{"PublicIP":"192.168.14.97"}
# etcdctl get /coreos.com/network/subnets/172.17.19.0-24
{"PublicIP":"192.168.14.98"}
# etcdctl get /coreos.com/network/subnets/172.17.20.0-24
{"PublicIP":"192.168.14.100"}
```
Flannel Routing (Backend.Type)

- Overlay (tunnel)
  - UDP
  - VXLAN

- Non overlay
  - Host-Gateway (must be a L2 network)
Host-Gateway

Pod 10.1.1.2

Pod 10.1.1.3

CNI bridge

Route

default via 192.168.1.1
10.1.1.0/24 dev cni0
10.1.1.0/24 via 192.168.1.101

Node1

192.168.1.100/24

Pod 10.1.2.4

Pod 10.1.2.5

CNI bridge

Route

default via 192.168.1.1
10.1.2.0/24 dev cni0
10.1.2.0/24 via 192.168.1.101

Node2

192.168.1.101/24
Flannel Routing: UDP

Overlay network

container A route

host A route

host B route
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
  - Overview
  - Basic model
  - Kubernetes implementation
    - Reliable coordination/configuration data store implementation
    - Controller-manager/scheduler implementation
    - Networking/pod implementation
      - Networking background
      - Basic Pod implementation
      - Kubernetes networking designs
        » Flannel
        » Calico, Weave, Cilium
Calico: BGP vRouter
Weave

- **wRouter**
  - Gossip to synchronize data

- **L2 overlay**
  - **Sleeve (UDP)**
    - `Pcap get packet, UDP encap`
  - **Fastpath (kernel): update vswitch rules (odp)**
    - `VXLAN, odp flow rule`
**Weave Control Plane**

Similar to Calico
Weave Data Plane

Similar to Flannel

Network isolation: 10.0.1/24 use left; otherwise right
Right bridge has no routing and hence cannot connect
Also see DNS
Weave Sleeve Datapath
Cilium

- **Issue: services/security on top of socket ports**
  - HTTP Port 80 can carry many RESTful functions, some safe some not

- **Use BFP**
  - Socket options
    - `SO_ATTACH_FILTER`
      - Classic BPF
    - `SO_ATTACH_BFP`
      - Extended BPF
Backup Slides
Service and Pod
Service and Endpoints

```yaml
kind: Service
apiVersion: v1
metadata:
  name: nginx-service
spec:
  clusterIP: 100.101.28.148
  selector:
    app: nginx
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 8080
```
AddNetwork(net *NetworkConfig, rt* RuntimeConf) (types.Result)

DelNetwork(net *NetworkConfig, rt* RuntimeConf)

#define CNI

// Plugin is an interface to network plugins for the kubelet
type NetworkPlugin interface {
    // Init initializes the plugin. This will be called exactly once
    // before any other methods are called.
    Init(host Host, hairpinMode componentconfig.HairpinMode, nonMasqueradeCIDR string) error

    // Called on various events like:
    // NET_PLUGIN_EVENT_POD_CIDR_CHANGE
    Event(name string, details map[string]interface{})

    // Name returns the plugin's name. This will be used when searching
    // for a plugin by name, e.g.
    Name() string

    // Returns a set of NET_PLUGIN_CAPABILITY_*
    Capabilities() utilsets.Int

    // SetUpPod is the method called after the infra container of
    // the pod has been created but before the other containers of the
    // pod are launched.
    SetUpPod(namespace string, name string, podInfraContainerID kubecontainer.ContainerID) error

    // TearDownPod is the method called before a pod's infra container will be deleted
    TearDownPod(namespace string, name string, podInfraContainerID kubecontainer.ContainerID) error

    // Status is the method called to obtain the ipv4 or ipv6 addresses of the
    // container
    GetPodNetworkStatus(namespace string, name string, podInfraContainerID kubecontainer.ContainerID) (*PodNetworkStatus, error)

    // NetworkStatus returns error if the network plugin is in error state
    Status() error
}
AddNetwork(net *NetworkConfig, rt* RuntimeConf) (types.Result, error) {
    rt, err := plugin.buildCNIRuntimeConf(ifName, id)
    if err != nil {
        return nil, fmt.Errorf("Error building CNI config: %v", err)
    }

glog.V(3).Infof("Adding %s/%s to %s with CNI %s' plugin and runtime: %s", 
    namespace, name, config.Network.Name, config.Network.Type, rt)
    res, err := plugin.cniConfig.AddNetwork(config, rt)
    if err != nil {
        return nil, fmt.Errorf("Error adding container to network: %v", err)
    }
    return res, nil
}

// Plugin is an interface to network plugins for the kubelet
// type NetworkPlugin interface {
//     // Init initializes the plugin. This will be called exactly once
//     // before any other methods are called.
//     Init(host Host, hairpinMode componentconfig.HairpinMode, nonMasqueradeCIDR string)
//     error
// }
// Called on various events like:
// NET PLUGIN EVENT POD CIDR CHANGE
// Event(name string, details map[string]interface{})
// Name returns the plugin's name. This will be used when searching


## Flannel Routing (Overlay): UDP

<table>
<thead>
<tr>
<th>Cont.</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4.0.100.3</td>
</tr>
<tr>
<td>B</td>
<td>4.0.100.5</td>
</tr>
<tr>
<td>C</td>
<td>4.0.32.3</td>
</tr>
</tbody>
</table>

```bash
# route -n
Kernel IP routing table

<table>
<thead>
<tr>
<th>Destination</th>
<th>Gateway</th>
<th>Genmask</th>
<th>Flags</th>
<th>Metric</th>
<th>Ref</th>
<th>Use</th>
<th>Iface</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0</td>
<td>10.8.65.1</td>
<td>0.0.0.0</td>
<td>UG</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>eth0</td>
</tr>
<tr>
<td>4.0.0.0</td>
<td>0.0.0.0</td>
<td>255.255.0.0</td>
<td>U</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>flannel0</td>
</tr>
<tr>
<td>4.0.100.0</td>
<td>0.0.0.0</td>
<td>255.255.255.0</td>
<td>U</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>docker0</td>
</tr>
<tr>
<td>10.8.64.10</td>
<td>10.8.65.1</td>
<td>255.255.255.255</td>
<td>UGH</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>eth0</td>
</tr>
<tr>
<td>10.8.65.0</td>
<td>0.0.0.0</td>
<td>255.255.255.0</td>
<td>U</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>eth0</td>
</tr>
</tbody>
</table>
```

```bash
# ip addr
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    link/ether 08:00:27:b7:7e:f3 brd ff:ff:ff:ff:ff:ff
    inet 10.8.65.66/24 brd 10.8.65.255 scope global dynamic enp0s3
        valid_lft 67134sec preferred_lft 67134sec
        inet6 fe80::a00:27ff:feb7:7ef3/64 scope link
            valid_lft forever preferred_lft forever
...
5: flannel0: <POINTTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1472 qdisc pfifo_fast state UNKNOWN qlen 500
    link/none
```
Packet Flow