The 12th International Conference on Advances in System Testing and Validation Lifecycle

An Overview of Cloud-Native Networks Design and Testing

Zhaobo Zhang
Futurewei Technologies
zzhang1@futurewei.com
Oct. 2020

Speaker Biography

Zhaobo Zhang
Principal Engineer
Futurewei Technologies, CA, U.S.

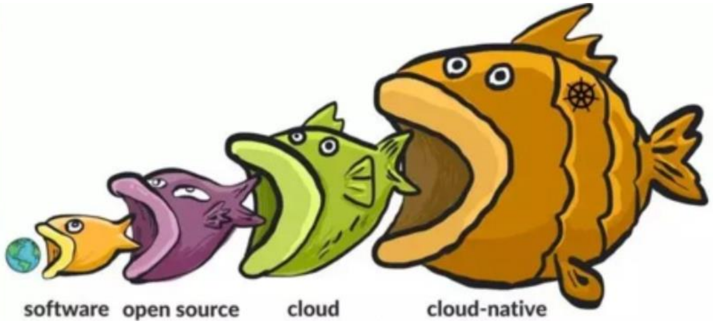
Zhaobo Zhang is a principle engineer in Network Technologies Lab at Futurewei Technologies, Inc. She has been working on machine learning applications for anomaly detection, system testing, fault diagnosis for 10 years. Her recent focus is on cloud-native networking, and machine learning based resource orchestration. She received her B.S. in Electronics Engineering from Tsinghua University, China, and Ph.D. in Electrical and Computer Engineering from Duke University, U.S.

Outline

- Background on Cloud-Native & Cloud-Native Networks
- Open Source Landscape
- Performance Challenges and Acceleration Techniques
- Testing and Observability
- Design Principles
- Takeaways

What’s Cloud Native?

- Combination of Containers, CI/CD, Microservices, Declarative APIs, DevOps
- Key benefits
  - Ship fast, reduce risk
  - Scalability, Agility, Resiliency
- Cloud technologies evolution timeline

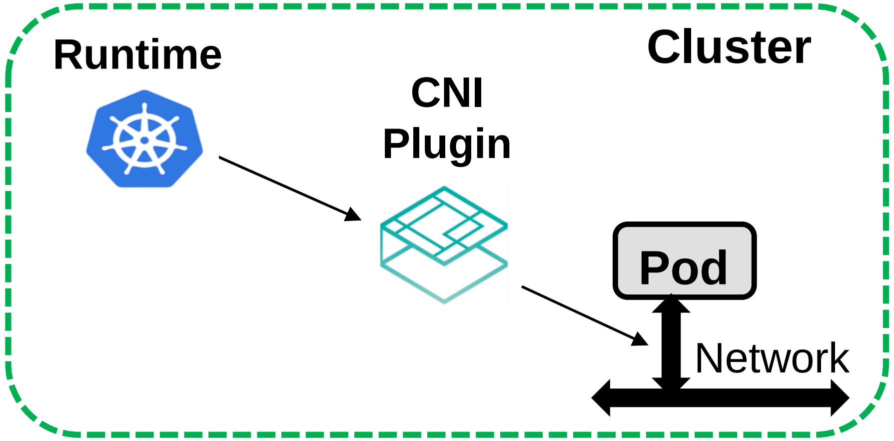


(Internet picture, unknown source)

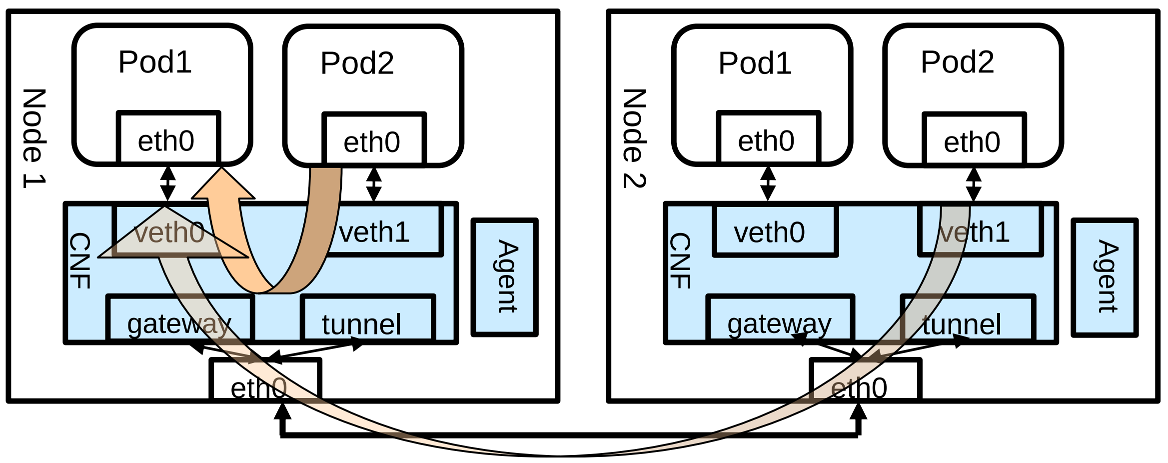
Cloud Computing	Cloud-native Application	Cloud-native Orchestrator, Cloud-native Computing Foundation	Cloud-native Network Function, Cloud-native Telco
2006 Amazon EC2	**2010** Netflix Cloud Migration	**2015** Kubernetes CNCF	**2018** CNFs Definition ONAP containerization

Cloud-Native Networks

- Deliver networking in a cloud env; Network itself is implemented with cloud-native principles
- Kubernetes as container orchestrator; Networking via container networking interface (CNI) plugins; Linux kernel as the building blocks
- Cloud-Native Networks basic functions
  - General Pod connectivity
  - IP address management (IPAM)
  - Service handling and load balancing
  - Network policy enforcement
  - Monitoring and troubleshooting



Cloud-Native Networks in Kubernetes



Bridge Plugin Example

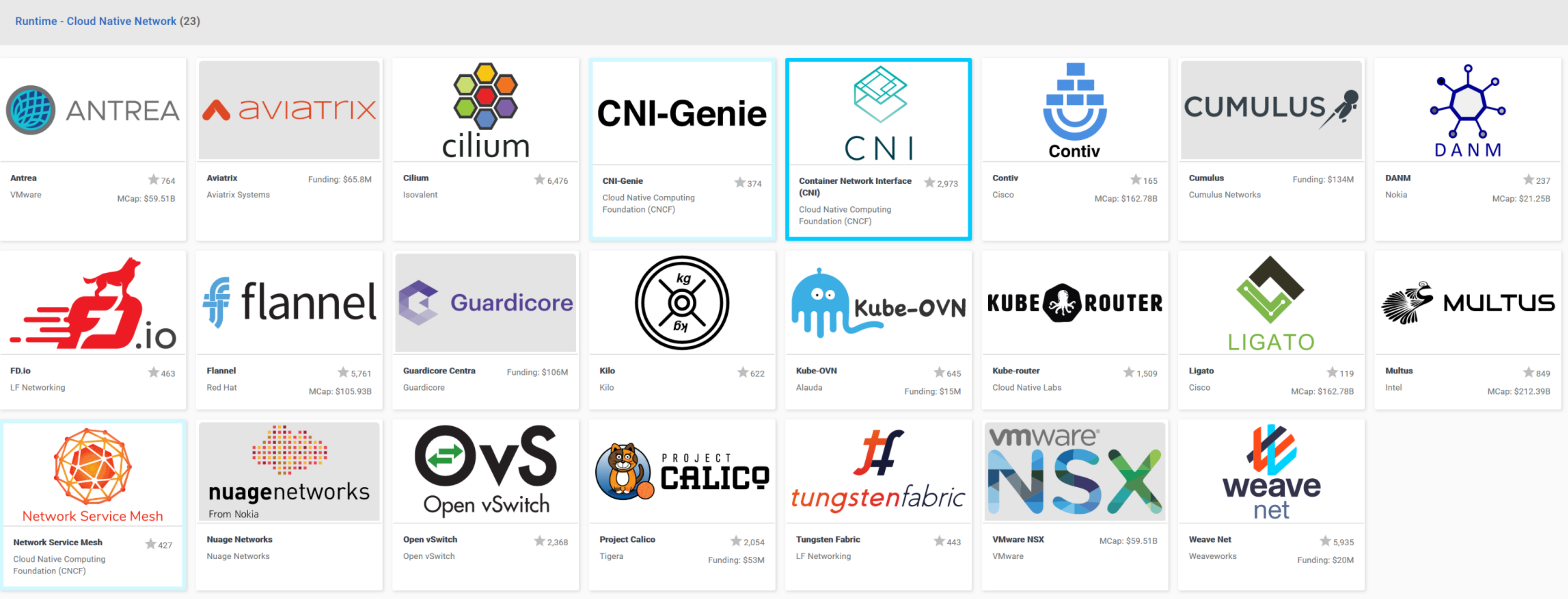
1. Create bridge network
2. Create a veth pair
3. Attach one veth to Pod namespace
4. Attach the other veth to bridge
5. Assign IP address
6. Bring interface up
7. Enable NAT

```
$ bridge add <CID> <Namespace>
```

- Basic Kubernetes networking definitions
  - Pod: a group of containers on a same host, IP per Pod, change dynamically
  - Service: a group of endpoints (pods), stable virtual IP
  - Flat network inside cluster, all Pods can communicate without NAT
  - Plugin-based network solution, create networks for pods when Kubernetes initiate Pods
  - Network policy describe the allowed communication among Pods
  - 4 types communication: container-to-container, Pod-to-Pod, Pod-to-Service, External-to-Service

Open Source Landscape – CNCF Cloud-Native Networks

- A view of 23 cards, market cap of $561.87B and funding of $393.8M*



*https://landscape.cncf.io/

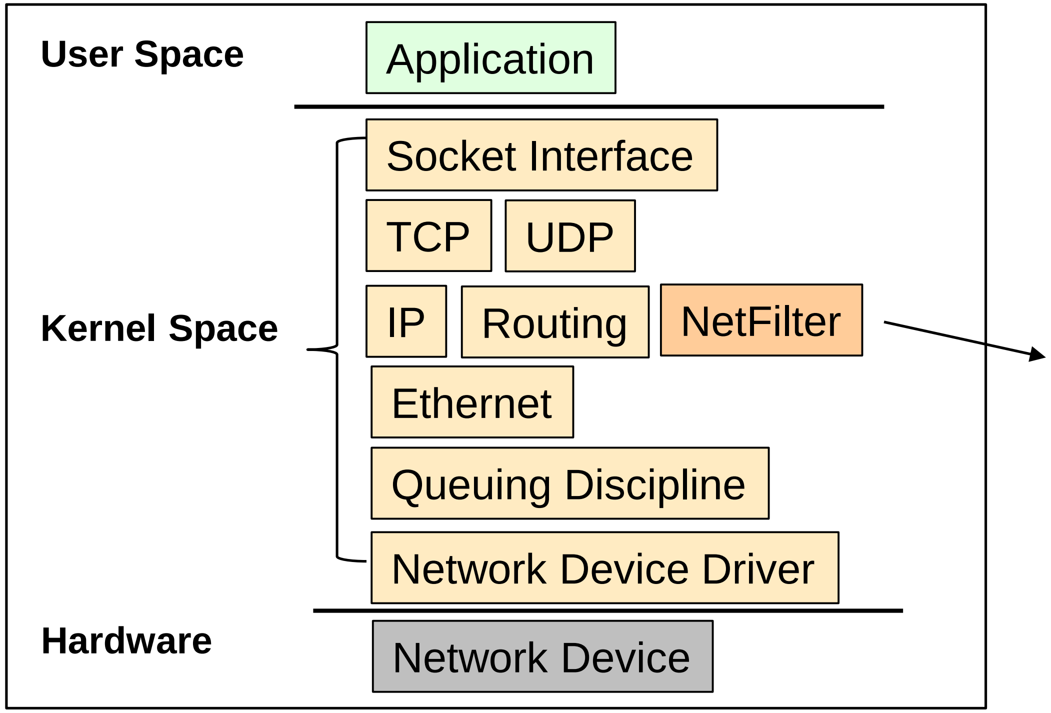
Problems Covered

- Container Network Interface Standards
- Generic Solutions
- Multiple interfaces in a container
- Data plane acceleration
- Hardware acceleration
- Multi-cloud networking

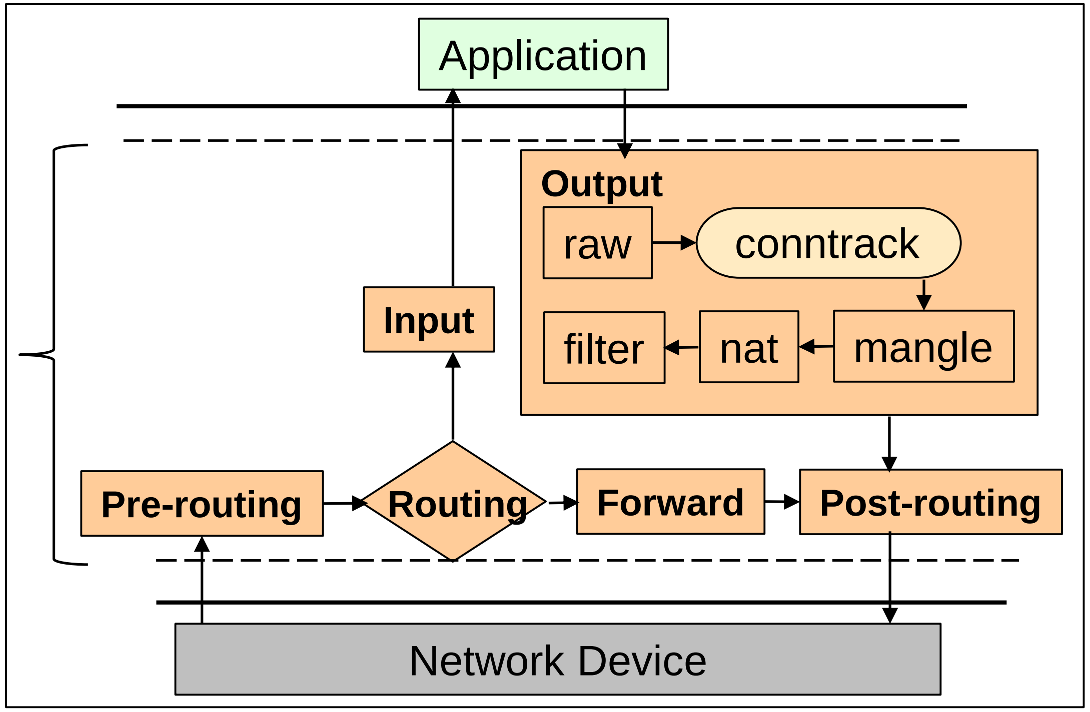
Performance Challenges

- **Linux networking stack issues**
  - Complex, ~12 millions lines of code
  - A copy is needed from user space to kernel space
  - Packet flow is long, especially with NetFilter (port mapping, NAT, etc)

Kernel Networking Stack

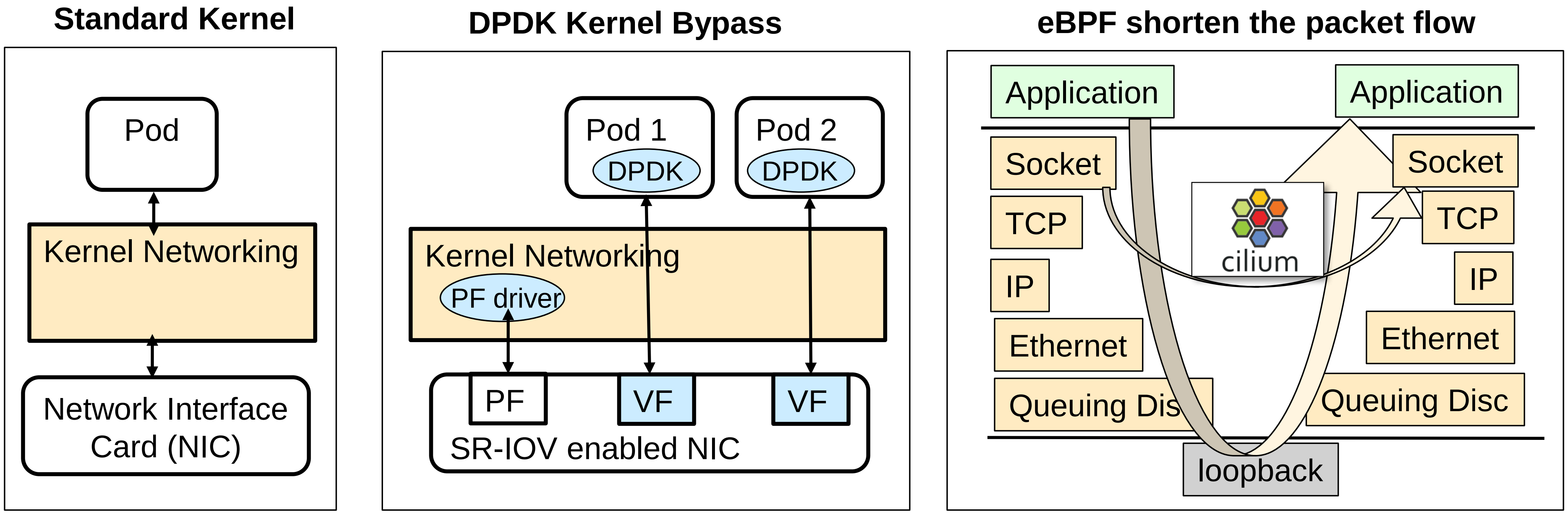


NetFilter packet flow (5 chains, 5 tables)



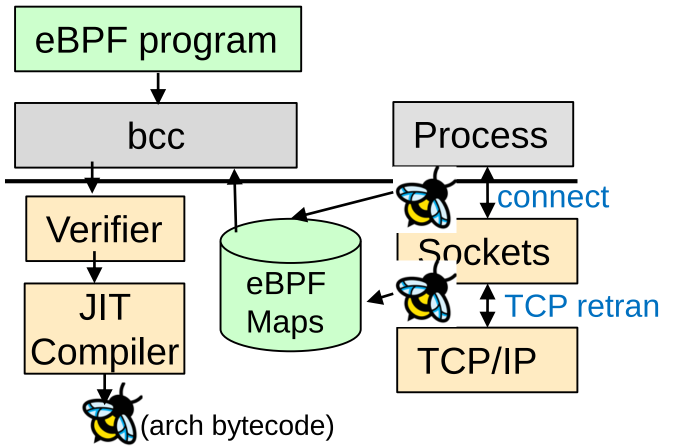
Software Acceleration Technologies

- Software-based, provide fast-path for packets, utilize unique CPU features
  - DPDK/VPP, user space forwarding, bypass kernel
  - eBPF, customize kernel packet processing flow, maximize efficiency



eBPF (extended Berkeley Packet Filter)* eBPF

- "Superpower", reprogram the behavior of Linux kernel without changing source code
- Component: eBPF program and Maps, Hooks, Helper functions
- Toolchains: bcc, bpftrace, go/c/c++ lib
- Applications, run eBPF program on events
  - Networking, Security, Tracing & Profiling, Observability & Monitoring
- Industry adoption
  - Cilium (eBPF-based CNI), Cloudflare (eBPF-based DDos)
  - Facebook, L3-L4 load balancing, network security, profiling, etc.
  - Google, Cilium & eBPF as the new networking data plane for GKE



*https://ebpf.io/what-is-ebpf

Hardware Acceleration Technologies

- Utilize different processing architecture (SmartNIC, FPGA, GPU) to parse and dispatch network packets instead of CPU
  - Network throughput greatly increased, but not the CPU computation power
- Offload network functions to hardware
  - TCP, TLS/IPsec crypto, OVS* ANTREA
- Adoption is driven by hyperscalers
  - Azure, FPGA-based SmartNIC, programmed using generic flow tables
  - GCP, GPU attached VM, throughput is up to 100 Gbps
  - AWS, Nitro card

*https://antrea.io/presentations/

Testing Infrastructure

- CI/CD pipeline, from source to production ASAP

Continuous Integration (Development, Build) → **Continuous Delivery** (Stage, Production)

Development
- Unit Tests
- Static analysis

Build
- Integration Tests
- Regression Tests
- Component Tests
- Vulnerability scan

Stage
- System Tests
- Performance Tests
- Load Tests
- Compliance Tests

Production
- A/B Test
- Canary Tests

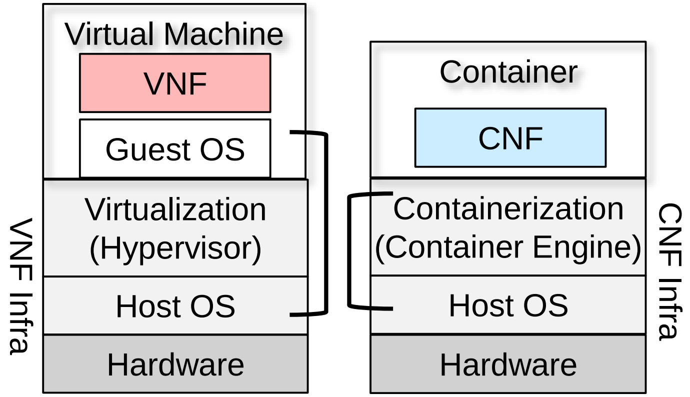
- CI/CD Tools
  - Trigger/schedule tests/tasks; Manage source/artifact/results
  - CNCF has 30+ projects

Jenkins GitLab TEKTON

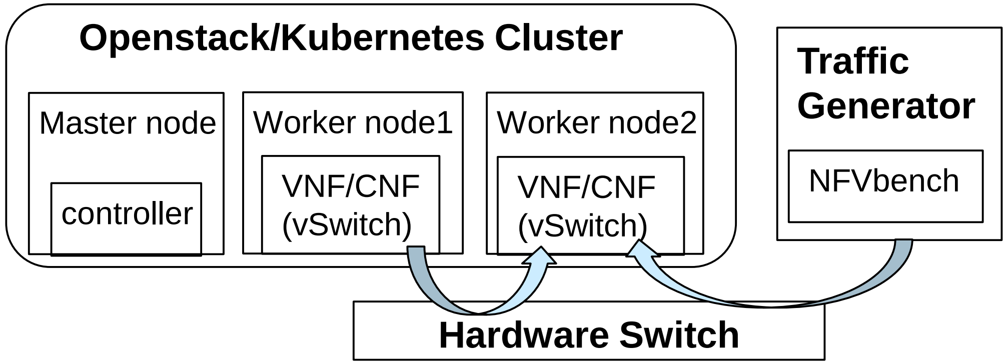
Test Cases

- Functional Tests
  - Connectivity Test (readiness, liveness)
  - Policy Test (firewall rules)
- Performance Tests
  - Function itself (latency/throughput)
  - Function at scale (large no. of requests/nodes, large tables/database)

Performance Comparison VNF vs CNF



- Network Architecture Evolution: PNF -> VNF -> CNF
- CNF Testbed Project*
  - Compare VNFs on OpenStack with CNFs on Kubernetes
  - Workflow: Hardware provision -> Infra provision -> VNF, CNF deploy -> Testing
  - Network functions: Packet Filter, NIC Gateway
  - Use cases: service chaining, SR-IOV device plugin, multiple network paths
  - Preliminary results: CNF leads more metrics*
    - Deploy time, idle state RAM/CPU, throughput
    - Latency, runtime RAM/CPU



*https://github.com/cncf/cnf-testbed

Beyond Testing: Built-in Observability

- DevOps, development and operation together
- Observability: Metrics, Logging, Tracing
  - CNCF standard I/F: OpenMetrics, Fluentd, OpenTelemetry
- CNF design with built-in observability
  - Data source
    - Probes: kprobes, uprobes, dtrace probes
    - Tracepoints: compile tracepoints into CNF/program
  - Data extraction
    - Files(/sys/kernel/debug/tracing), system calls (perf_event_open)
    - eBPF program, attach to probes and tracepoints, send data back by BPF Maps
  - Use cases: Interface changes, table/session updates

Example: user program probe, gobpf/bcc by IOvisor*

```c
int count(struct pt_regs *ctx) {
    if (!PT_REGS_PARM1(ctx))
        return 0;
    strlenkey_t key;
    u64 zero = 0, *val;
    bpf_probe_read(&key, sizeof(key), (void *)PT_REGS_PARM1(ctx));
    val = counts.lookup_or_init(&key, &zero);
    (*val)++;
    return 0;
}
```

```go
func main() {
    pid := flag.Int("pid", -1, "attach to pid, default is all processes")
    m := bpf.NewModule(source, []string{})
    strlenUprobe, err := m.LoadUprobe("count")
    err = m.AttachUprobe("c", "strlen", strlenUprobe, *pid)
    table := bpf.NewTable(m.TableId("counts"), m)
    fmt.Println("Tracing strlen()... hit Ctrl-C to end.")
    fmt.Printf("%10s %s\n", "COUNT", "STRING")
    for it := table.Iter(); it.Next(); {
        fmt.Printf("%10d \"%s\"\n", it.key(), it.Leaf())
    }
}
```

*https://github.com/iovisor/gobpf/blob/master/examples/bcc/strlen_count/strlen_count.go

Cloud-Native Networks Design Principles

- Containerization, network functions packed into containers
- Stateless, states stored separated in a CRD or DB, not local
- Microservices, complex network functions made by CNFs chaining
- Dynamic orchestration via Kubernetes
- Configuration via ConfigMap or other declarative APIs
- Built-in observability, compatible with CNCF standard interfaces
- Software and hardware co-design, hardware support via device plugin

Takeaways

- Extensibility is the foundation of Kubernetes' success; CNI plugin and Device plugin promote various solutions and opportunities
- Cloud-native technologies and tools are fast growing; network domain could leverage their existing success to accelerate its own evolution
- Performance is always a challenge, eBPF brings a new way to improve Linux kernel; hardware acceleration could be more significant; co-design probably yields the best.

THANK YOU