1.11

NETWORKING

MESOSPHERE

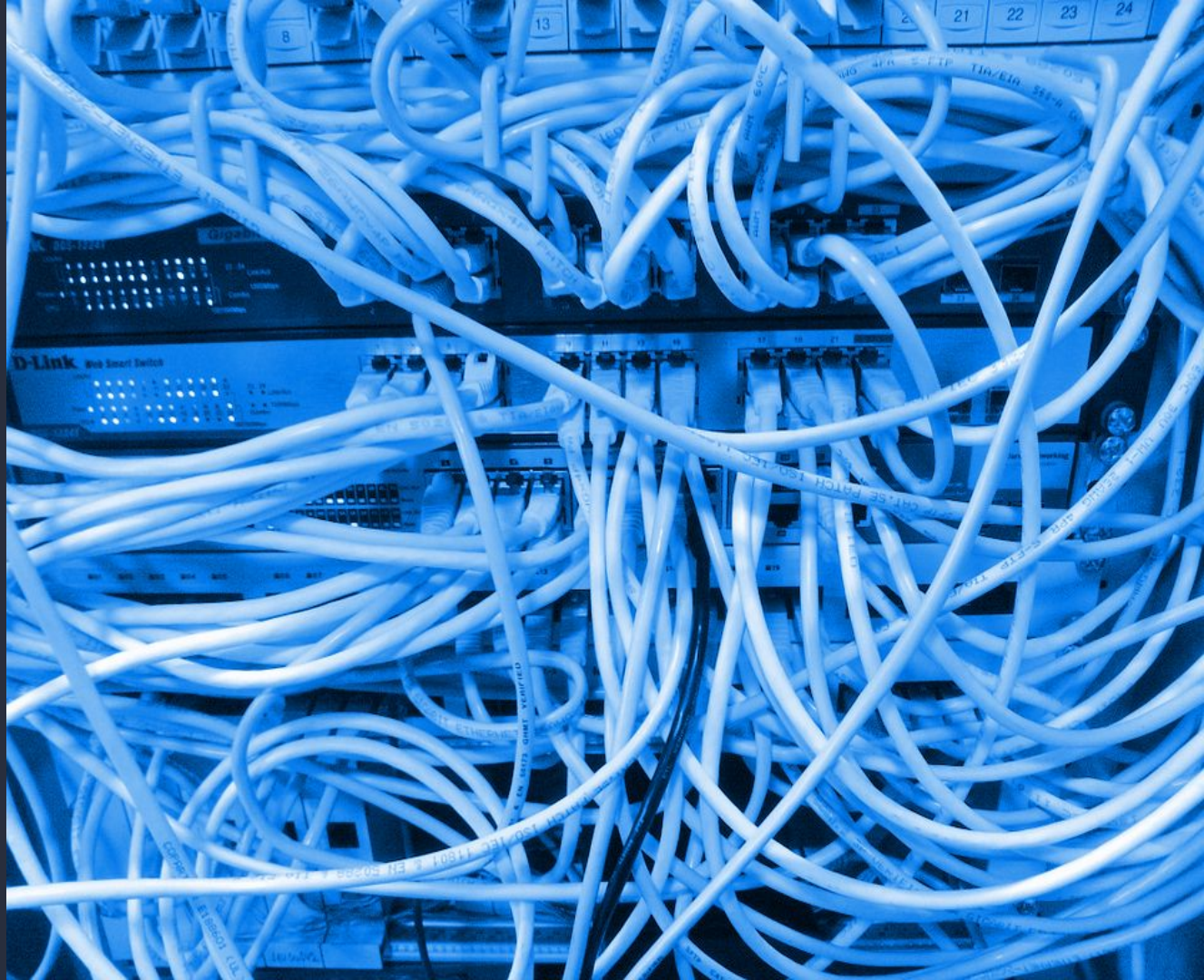
AGENDA

1. Overview
2. Service Discovery
    a. DNS
        i. Mesos-DNS
        ii. Spartan DNS Proxy
    b. Load Balancing / Proxy
        i. Edge Load Balancing
        ii. Distributed Load Balancing
        iii. Admin Router
    c. Application / Client SD
3. Virtual Networking
    a. Ingress

DC/OS Networking

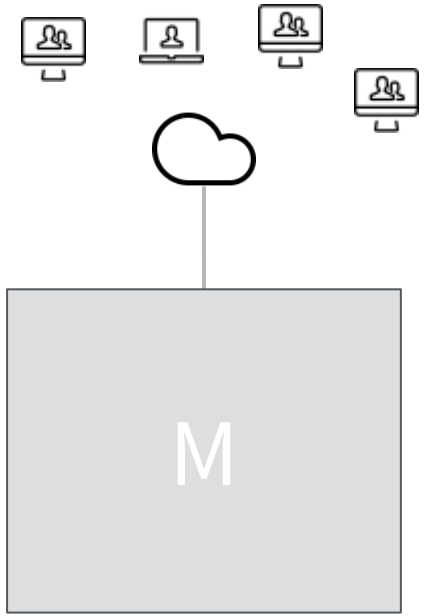
OVERVIEW

LEGACY NETWORKING

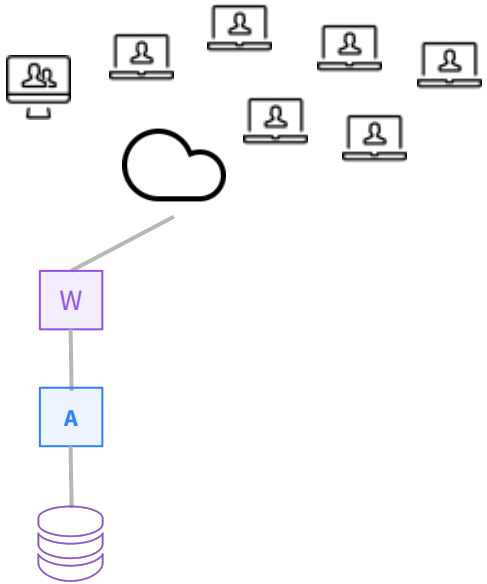


NEW AGE OF NETWORKING

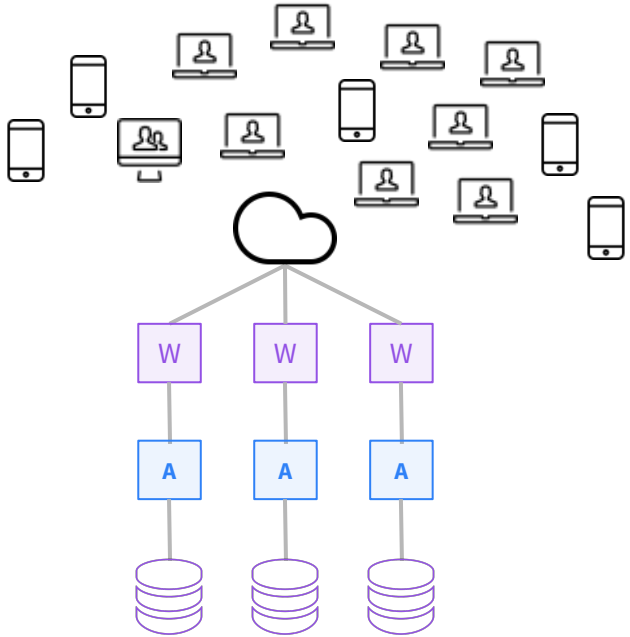
MONOLITH



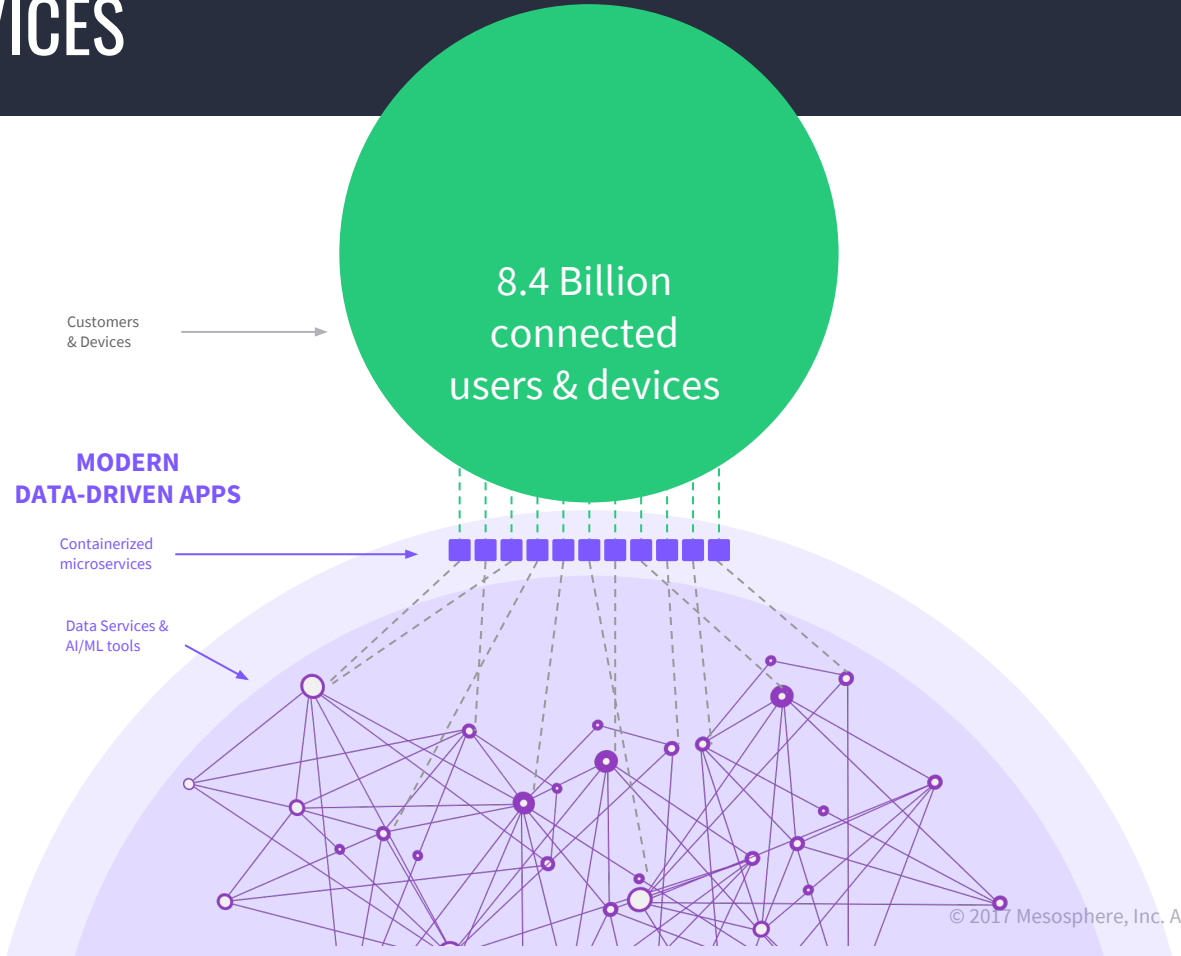
CLIENT SERVER



VIRTUALIZATION (MORE EVERYTHING)



MICROSERVICES



DC/OS Networking

SERVICE DISCOVERY

SERVICE DISCOVERY & LOAD BALANCING

DC/OS and Marathon are not strongly opinionated when it comes to service discovery and load balancing and can support different service discovery solutions

There are three design patterns when it comes to service discovery

1. DNS
2. Proxy / Load Balancing
3. Client Service Discovery

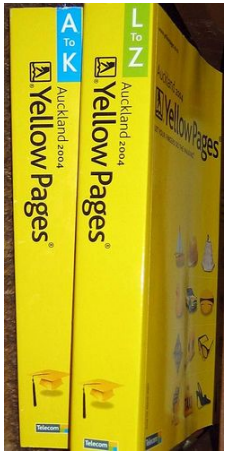
DC/OS Networking

DNS

PATTERN #1 - DNS

Pros:

- Easy to integrate with most existing apps
- Apps do not usually require modification



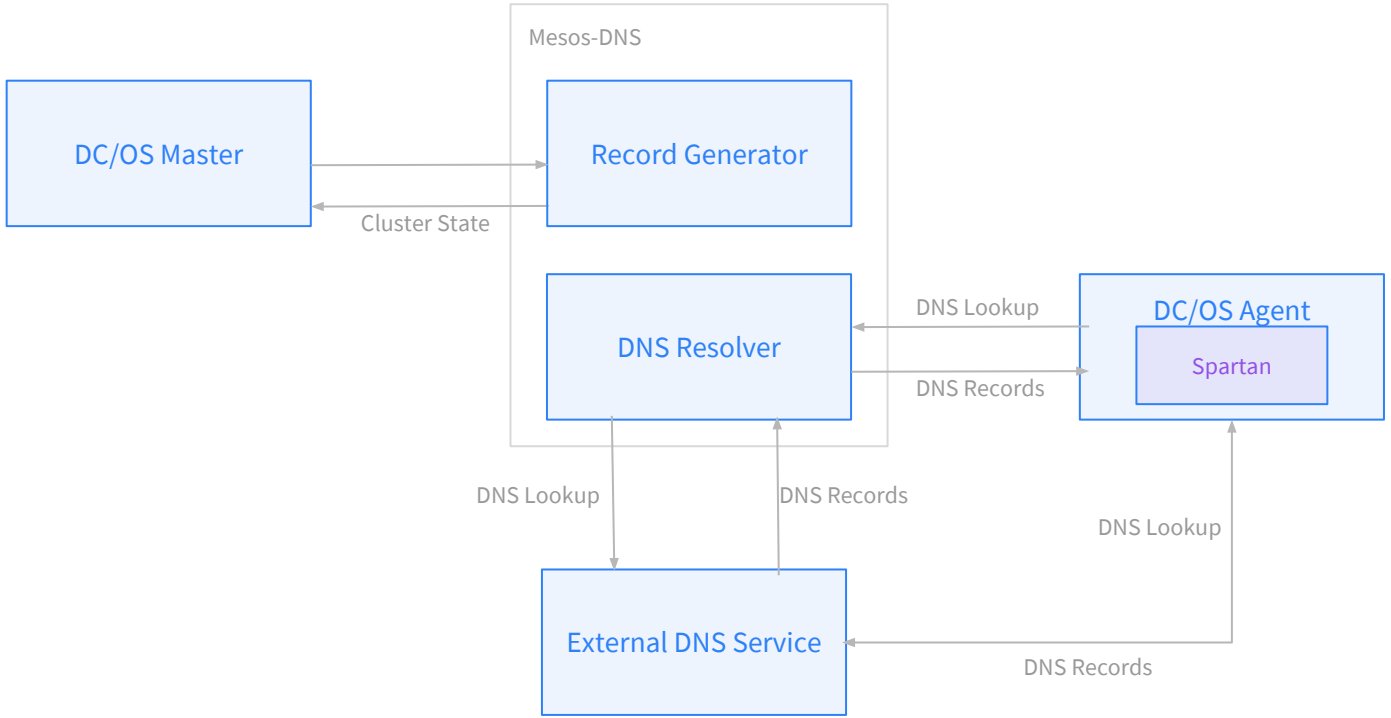
Cons:

- Ports not easily managed (in spite of SRV records)
- No fast failover
- DNS records may be stale
- Health checks not observed
- DNS clients may not behave correctly.
- No clients support using SRV records which means you must know the port

PATTERN #1 - MESOS-DNS

- Included by default—no configuration necessary
- Polls the Mesos API at a configurable interval (30s by default)
- Publishes A and SRV records
- Has a RESTful HTTP API
- Stateless by design

MESOS-DNS ARCHITECTURE



TESTING MESOS-DNS

Requirements

- Test host has dig or nslookup installed
- Test host has mesos-dns servers as primary and secondary in `/etc/resolv.conf` depending on how many Mesos Masters

To get an A record for a Mesos task using DNS query:

- `$ dig <app_id>.marathon.mesos A`

To get an SRV record for a Mesos task using DNS query.

- `$ dig _<app_id>._tcp.marathon.mesos SRV`

MESOS-DNS API

Mesos-DNS API uses TCP port 8123 by default

Available through adminrouter

- `http://<mesos-master ip>/mesos_dns/v1/...`

Command API cals

- `GET /v1/version` Get Mesos-DNS version
- `GET /v1/config` Get Mesos-DNS config
- `GET /v1/enumerate` List all Mesos-DNS entries
- `GET /v1/hosts/<app-id>.marathon.mesos` Get DNS A record for task
- `GET /v1/services/_<app-id>._tcp.marathon.mesos` Get DNS SRV record for task

Lab 6a

MESOS DNS

LAB 6A - USING MESOS-DNS

1. List the running Marathon applications

```
$ dcos marathon app list
```

2. SSH into the master node

```
$ dcos node ssh --master-proxy --leader
```

3. Ping the application

```
$ ping -c 3 <app-id>.marathon.mesos
```

4. Do a DNS lookup of the application. Which DNS server does it use?

```
$ dig <app-id>.marathon.mesos
```

5. Do an SRV lookup of the application

```
$ dig _<app-id>._tcp.marathon.mesos SRV
```

LAB 6A - USING MESOS-DNS API

1. SSH into the master node

```
$ dcos node ssh --master-proxy --leader
```

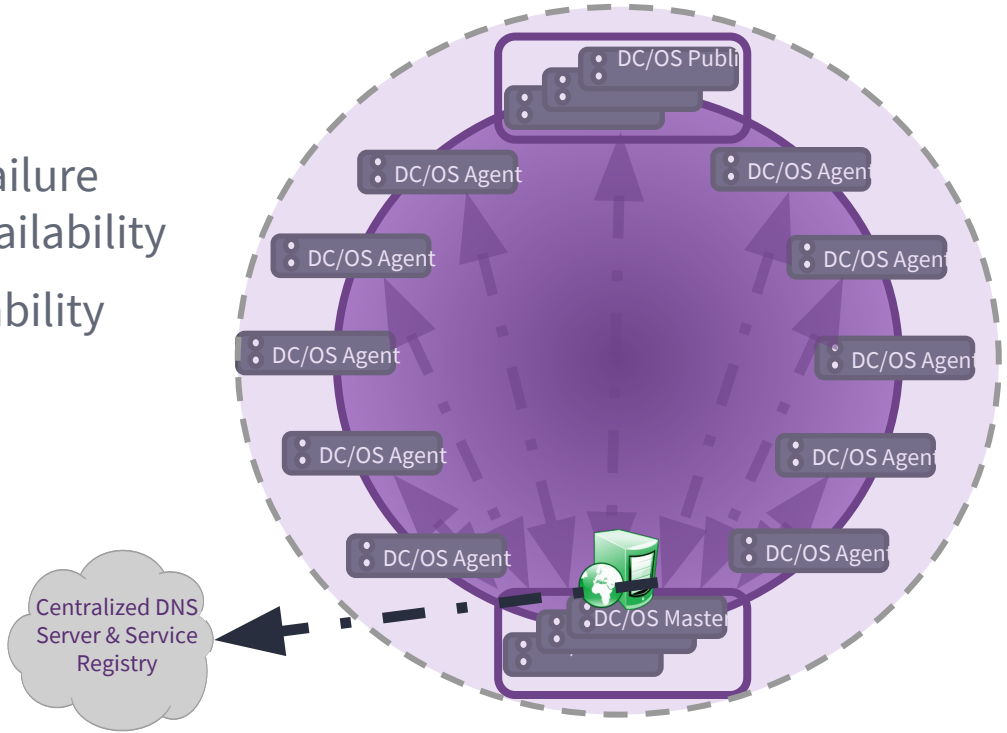
2. Explore the following API calls

```
$ curl http://localhost:8123/v1/version
$ curl http://localhost:8123/v1/config
$ curl http://localhost:8123/v1/enumerate
$ curl http://localhost:8123/v1/hosts/<app id>.marathon.mesos
$ curl http://localhost:8123/v1/services/_<app_id>._tcp.marathon.mesos
```

CENTRALIZED DNS CHALLENGES

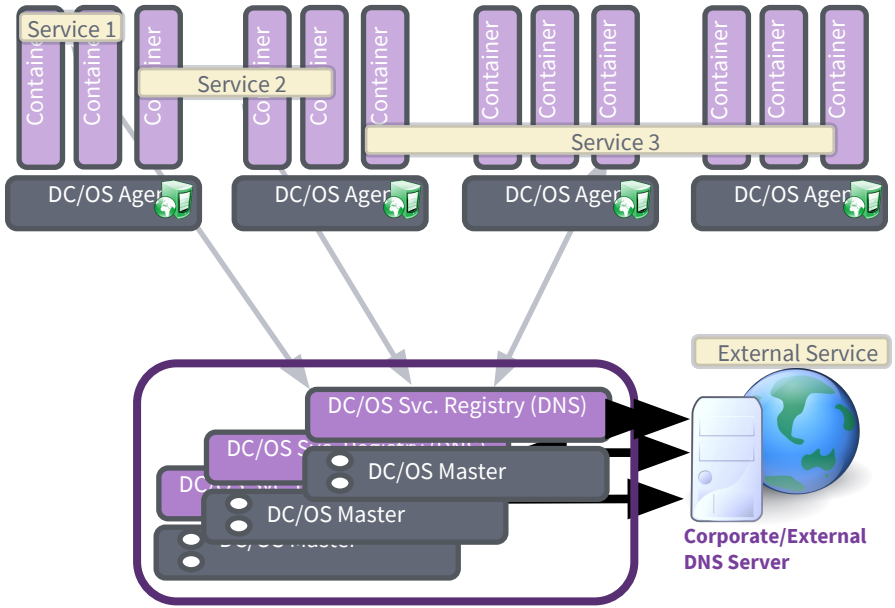
Availability - master/upstream failure results in 1/3 of cluster losing availability

Performance - Time to discoverability



DISTRIBUTED DNS SERVER PROXY (SPARTAN)

- Scale-out DNS Server on DC/OS masters with replication
- DNS Server Proxy with links to all Active/Active DNS server daemons
- DNS Server cache service for local services



Benefits:

- Improved availability of DNS based Service Registry
- Near constant time to discoverability regardless of cluster size
- Application transparent failure tolerance of Service Registry (DNS Server) failure events

DC/OS Networking

EDGE NORTH SOUTH LOAD BALANCER

PATTERN #2 - PROXY / LB

Examples:
  - IP-based: Minuteman
  - HAProxy: Marathon-lb, SmartStack (from Airbnb), nginx
- Pros:
  - No port conflicts
  - Load balancer and service discovery are one in the same
  - Very fast failover
- Cons:
  - No UDP
  - Must maintain list of service ports (or VIPs) to prevent overlap
  - Additional round trip
  - Some services do not work through a LB (even with sticky enabled)

PATTERN #2 - MARATHON-LB

- MLB available as a DC/OS package
- Based on HAProxy
- Only works for apps on Marathon
- Respects health checks
- HAProxy parameters can be configured using app labels
- By default, ports 10000-10100 are allocated for services
- Supports TCP and HTTP so you can do L7 Load Balancing

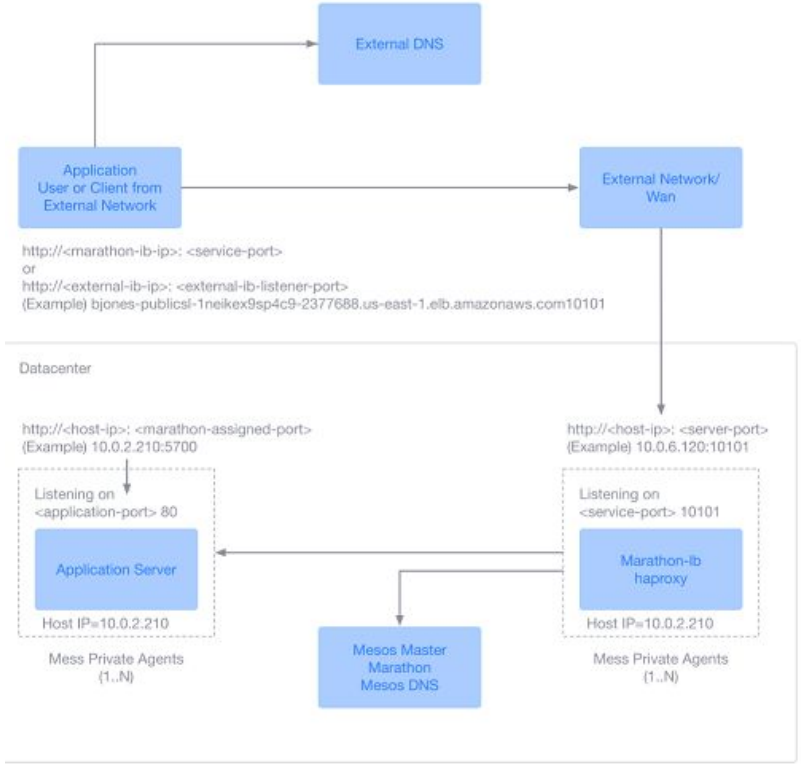
MARATHON-LB

- Based on HAProxy for High Performance and reliability
- Works for HTTP and TCP Applications
- Supports Virtual Hosts, SSL, HTTP Compressions, Health Checks, Lua scripting
- Subscribes to Marathon event bus and updates HAProxy configuration.
- Use Cases
  - Use marathon-lb as your edge load balancer (LB) and service discovery mechanism.
  - Use marathon-lb as an internal LB and service discovery mechanism, with a separate HA load balancer for routing public traffic in.
- Leverages Marathon labels to configure LB

MARATHON-LB CONFIGURATION

- Haproxy “global” configuration parameters modified by loading a new configuration file HAPROXY_HEAD from /templates directory inside MLB container.
  - Can build a new Docker Image with a new Configuration or
  - Download as a Mesos URI and load at Marathon-lb startup.
- For Applications, HAPROXY Frontend and Backend Settings can almost all be done with Marathon App Labels

MARATHON-LB ARCHITECTURE



MARATHON-LB ADMINISTRATION

Component	Host : Port / uri/
Marathon-lb statistics	`<public-node>:9090/haproxy?stats`
Marathon-lb statistics	`<public-node>:9090/haproxy?stats;csv`
Marathon-lb Health Check	`<public-node>:9090/_haproxy_health_check`
Marathon-lb Configuration File VIEW	`<public-node>:9090/_haproxy_getconfig`
Marathon-lb get vHosts to Backend map	`<public-node>:9090/_haproxy_getvhostmap`
Marathon-lb get app ID to backend map	`<public-node>:9090/_haproxy_getappmap`
Marathon-lb Reload the Configuration	`<public-node>:9090/_mlb_signal/hup*`

MARATHON-LB SERVICE PORTS

Service PORT allows multiple Applications to be exposed from the same MLB Instance and distributed to different Marathon Applications.

Clients point to the same hostname and a unique Service Port.
This means you need to know or distribute Service Port

AppA = MLB-01.domain.com:10000 ---> MLB-01 ---> AppA
AppB = MLB-01.domain.com:10001 ---> MLB-01 ---> AppB

MARATHON-LB SERVICE PORT EXAMPLES

```
{
  "id": "/dev/webmenu-lb-host-serviceport-2",
  "cpus": 0.1,
  "mem": 32,
  "cmd": "python -m http.server $PORT0",
  "instances": 1,
  "container":{
    "docker":{
      "image":"python:3",
      "network": "HOST"
    }
  },
 "portDefinitions": [
      {"port": 10003}
    ],
  "labels":{
    "HAPROXY_GROUP":"external"
    }
}
```

```
{
  "id": "/dev/webmenu-lb-host-serviceport-1",
  "cpus": 0.1,
  "mem": 32,
  "cmd": "python -m http.server $PORT0",
  "instances": 1,
  "container":{
    "docker":{
      "image":"python:3",
      "network": "HOST"
    }
  },
  "labels":{
    "HAPROXY_GROUP":"external",
    "HAPROXY_0_PORT": "10002"
    }
}
```

MARATHON-LB VHOSTS

vHOSTs allow you to address all of your applications over the same “common” ports like 80 or 443.

To expose different Applications from same Marathon-lb you can either:

1) Use different DNS records pointing to same IP / MLB instance. MLB will use Host Headers to determine target Marathon App
2) Use the same DNS record pointing to same IP/ MLB instance and use /path in App Definition to specify the target Marathon Application

```
{

  "cmd": "python -m http.server $PORT0",
  "instances": 1,
  "container":{
    "docker":{
      "image":"python:3",
      "network": "HOST"
   }
  },
  "labels":{
    "HAPROXY_GROUP":"external",
    "HAPROXY_0_VHOST":
"public.us-east-1.elb.amazonaws.com"
    }
}
```

MARATHON-LB VHOST UNIQUE DNS A RECORDS

Use different DNS records pointing to same IP / MLB instance. MLB will use Host Headers to determine target Marathon App

AppA ---> http://appa.domain.com ---> MLB-01 ---> AppA
AppB ---> http://appb.domain.com ---> MLB-01 ---> AppB
AppC ---> http://appc.subdomain.domain.com ---> MLB-01 ---> AppB

MARATHON-LB VHOST SAME DNS A RECORD

Can use Layer 7 Information in the URI to SEND Traffic to different Marathon Applications.
Leverage HAPROXY_PATH Marathon Label
AppA = http://MLB-01.domain.com/appa ---> MLB-01 ---> AppA
AppB = http://MLB-01.domain.com/appb ---> MLB-01 ---> AppB

```
"labels": {
    "HAPROXY_0_VHOST": "thomaskra-publicsl-mefi1v-.us-east-1.elb.amazonaws.com",
    "HAPROXY_GROUP": "external",
    "HAPROXY_0_PATH": "/corpmenu",
    "HAPROXY_0_HTTP_BACKEND_PROXYPASS_PATH": "/corpmenu"
```

Lab 6b

MARATHON-LB

LAB 6B - EXPOSING WEB SERVICES

1. From the DC/OS CLI (or UI), install marathon-lb

```
$ dcos package install marathon-lb --yes #DONT DO THIS
```

2. Go to the following URL once marathon-lb is installed

```
http://<public-agent>:9090/haproxy?stats
```

3. Edit the previously launched web app and add the following labels

```
“HAPROXY_GROUP”: “external”
```

4. After the app redeploys, refresh the haproxy page to view the new config

```
http://<public-agent>:9090/haproxy?stats
http://<public-agent>:9090/_haproxy_getconfig
```

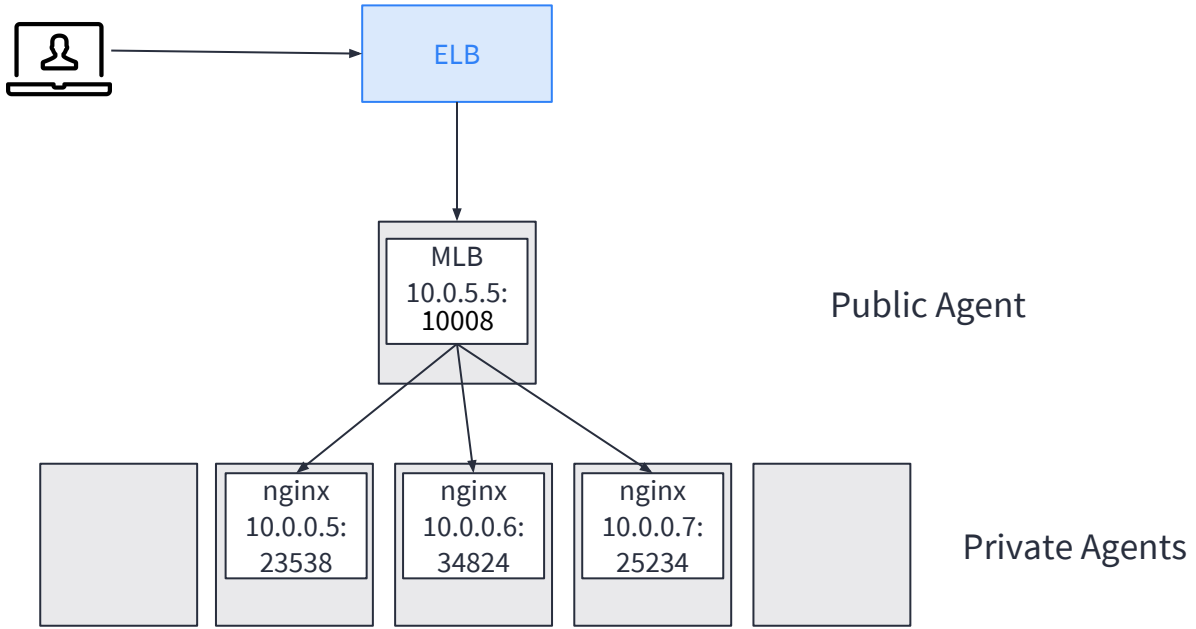
5. Edit the web app and add a label for a virtualhost

```
“HAPROXY_0_VHOST”: “<public-agent DNS>”
```

```
I.e “HAPROXY_0_VHOST”: “benlin-a.southeastasia.cloudapp.azure.com”
```

MARATHON-LB LAB

```
"id": "nginx",
“instances”:3,
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "nginx:1.7.7",
      "network": "BRIDGE",
      "portMappings": [
        { "hostPort": 0,
"containerPort": 80,
"servicePort": 10008 }
      ],
```



Networking

EDGE-LB

EDGE-LB

Overview:

- Edge-LB is an Enterprise only DC/OS package
- Available with a Support contract
- To access the package and for more information contact Mesosphere Support

Features:

- Based on HAProxy for high performance and reliability
- Supports all DC/OS applications and services
- Metrics and health monitoring support
- DC/OS Secrets integration

EDGE-LB

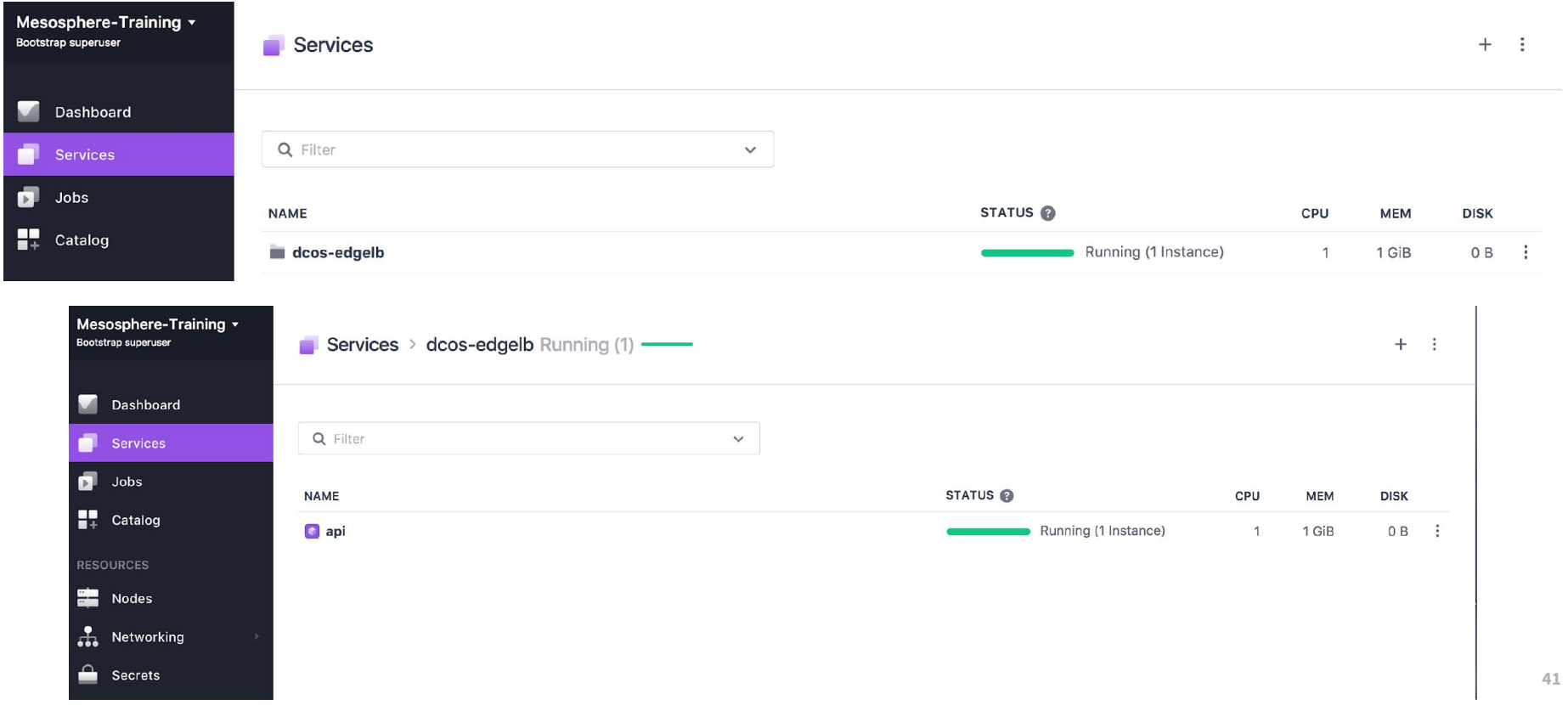
Pre-work

- Get the package from Mesosphere Support
- Add the package: `dcos package repo add edgelb https://<>`
- Ensure it is available: `dcos package repo list`

Installing and Configuring

- Use the DC/OS CLI: `dcos package install edgelb`
- Configure Edge-LB Pools

EDGE-LB


Mesosphere-Training
Bootstrap superuser
Dashboard
Services
Jobs
Catalog
Services
Filter
NAME
STATUS
CPU
MEM
DISK
dcos-edgelb
Running (1 Instance)
1
1 GiB
0 B
Mesosphere-Training
Bootstrap superuser
Dashboard
Services
Jobs
Catalog
RESOURCES
Nodes
Networking
Secrets
Services › dcos-edgelb Running (1)
Filter
NAME
STATUS
CPU
MEM
DISK
api
Running (1 Instance)
1
1 GiB
0 B

EDGE-LB

Uninstalling:

1. Delete each pool: `dcos edgelb pool delete <pool>`
2. Uninstall Edge-LB: `dcos package uninstall edgelb`
3. Remove the repositories:
   - `dcos package repo remove edgelb-aws`
   - `dcos package repo remove edgelb-pool-aws`

DC/OS Networking

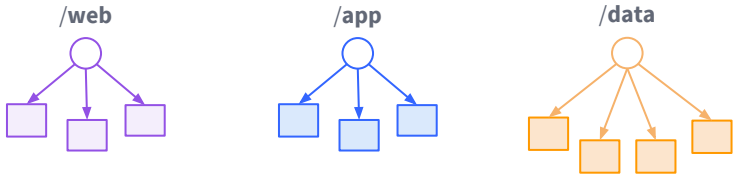
DISTRIBUTED E-W LOAD BALANCER

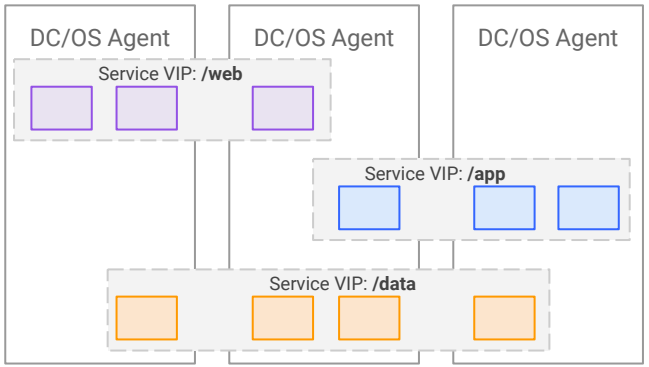
PATTERN #2 - MINUTEMAN

- L4 distributed load balancer (Minuteman) installed by default
- Distributed load balancing of applications
- Respects health checks

DISTRIBUTED LOAD BALANCER

- Fast converging distributed load balancer support for variety of L4 LB algorithms
- Highly available LB with no single choke point
- Highly scalable and tolerant to large # of host failures.





MINUTEMAN REQUIREMENTS

- Do not firewall traffic between the nodes (allow all TCP/UDP)
- Do not change ip_local_port_range
- Have the ipset package installed
- Run a stock kernel from RHEL 7.2+, or Ubuntu 14.04+ LTS

DC/OS NAME VIP

```
{
  "id": "nginx",
  "cpus": 0.1,
  "mem": 65,
  "instances": 3,
  "container": {
    "docker": {
      "image": "nginx",
       "network": "BRIDGE",
       "portMappings": [
         {
           "containerPort": 80,
           "protocol": "tcp",
           "labels": {
             "VIP_0": "/nginxapp:80"
           }
         }
      ]
    },
    "type": "DOCKER"
  }
}
```

```
$ curl app.marathon.l4lb.thisdcos.directory
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial,
sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
```

Lab 6c

MINUTEMAN

LAB 6C - DISTRIBUTED LOAD BALANCING

1. Login to the DC/OS UI and edit the previously created nginx app
2. Go to **Network**, enter a port, and select **Load Balanced**
3. After the app redeploys, SSH into the master node

```
$ ssh -i <pem file> core@<mesos-master>
```

4. Curl the VIP

```
curl <app-id>.marathon.l4lb.thisdcos.directory:<port>
```

5. View the network monitoring page on the DC/OS UI

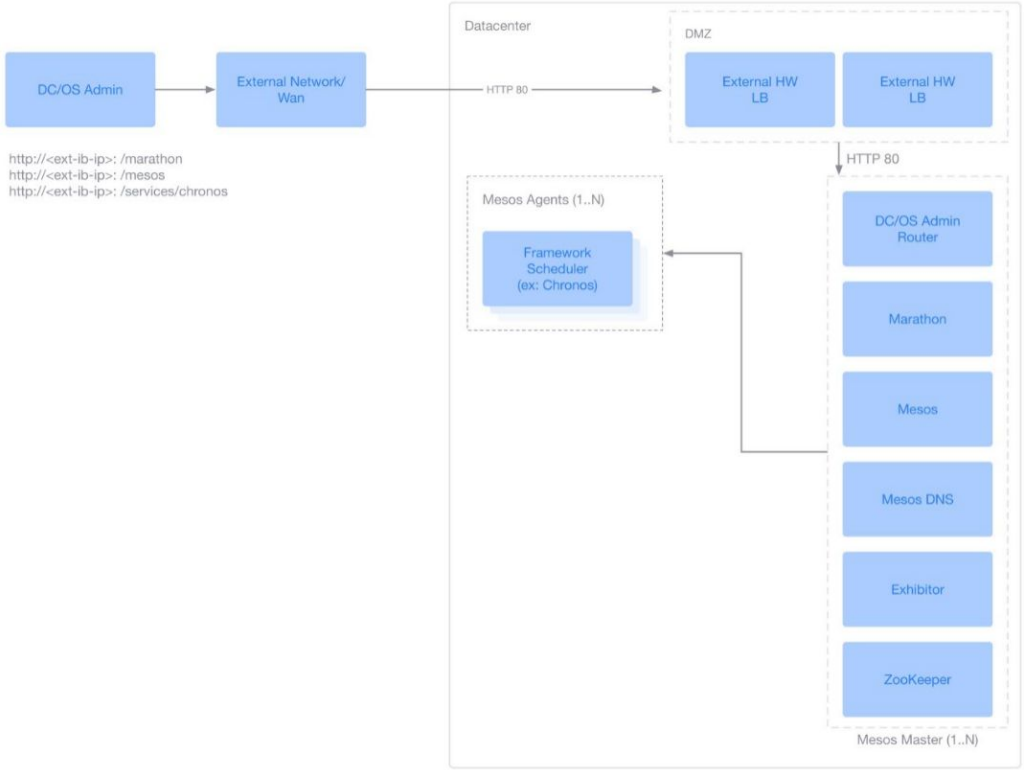
DC/OS Technical

ADMIN ROUTER

DC/OS ADMIN ROUTER

- Entrypoint into the DC/OS cluster for
  - Administrative Traffic over TCP 80 & 443
  - DCOS Services Admin User Interfaces
- Nginx reverse proxy
- Integrates with DCOS ACS (Bouncer) for Authentication & Authorization
- Routes traffic to DC/OS services and frameworks
- Installed on each DC/OS Master

DC/OS ADMIN ROUTER

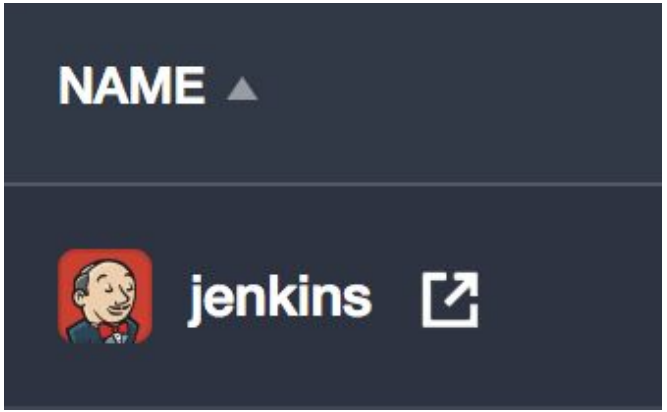


DC/OS ADMIN ROUTER

Component	Open Source Mesos Host : Port	With DC/OS & Admin Router via HTTP
Mesos Master Web UI / API	<mesos-master>:5050	<mesos-master> /mesos/
Mesos Agent UI / API	<mesos-agent>:5051	<mesos-master> /slave/<slave-id>
Exhibitor UI / API	<mesos-master>:8181	<mesos-master> /exhibitor/
Marathon UI / API	<mesos-master>:8080	<mesos-master> /marathon/
Mesos-DNS API	<mesos-master>:8123	<mesos-master> /mesos_dns/v1/...
Frameworks Admin UI	<framework-scheduler> : xyz	<mesos-master> /service/<service-id>
DC/OS History Svc API	N/A	<mesos-master> /DC/OS-history-service/history/...

DC/OS ADMIN ROUTER - SERVICES

- DCOS Services (Frameworks) expose their Schedulers Administrative Interfaces (if they have them) through Admin Router which runs on all of the DCOS Master Nodes. DCOS Service definition for the scheduler defines this
  - DCOS_SERVICE_NAME jenkins
  - DCOS_SERVICE_PORT_INDEX 0
  - http://<master-ip>/service/jenkins



DCOS/COMMONS SERVICES CONNECTION ENDPOINT API

- DCOS Services written against DCOS -Commons (Services SDK) will also expose a Connection endpoint.
- http://<master-ip>/service/kafka/v1/connection


```
{
  - address: [
        "10.0.1.114:9860",
        "10.0.3.172:9152",
        "10.0.2.204:9327"
    ],
    zookeeper: "master.mesos:2181/dcos-service-kafka",
  - dns: [
        "broker-0.kafka.mesos:9860",
        "broker-1.kafka.mesos:9152",
        "broker-2.kafka.mesos:9327"
    ],
    vip: "broker.kafka.l4lb.thisdcos.directory:9092"
}
```

DC/OS Networking

CLIENT SERVICE DISCOVERY

PATTERN #3 - CLIENT SERVICE DISCOVERY

Typically requires a consistency protocol implementation:

- ZK
- etcd
- or some other state store like Consul

Example: Finagle

- Pros:
  - App dev has complete control
  - Full-featured
- Cons:
  - Requires distributed state store
  - Need to write Applications to query SD Service

PATTERN #3 - ZOOKEEPER / ETCD

- Available as DC/OS Catalog service
- Distributed state store
- Can deploy multiple instances
- Apps can discover via mesos-dns or minuteman
- Availability and healthchecks provided by marathon

SUMMARY

In this module we looked at networking capabilities

- Mesos-DNS
- Spartan
- Marathon-LB (ingress)
- Minuteman (distributed load balancing)
- Admin Router
- Virtual networking

DC/OS Networking

VIRTUAL NETWORKS

OVERVIEW

IP per container solution for Mesos and Docker containers

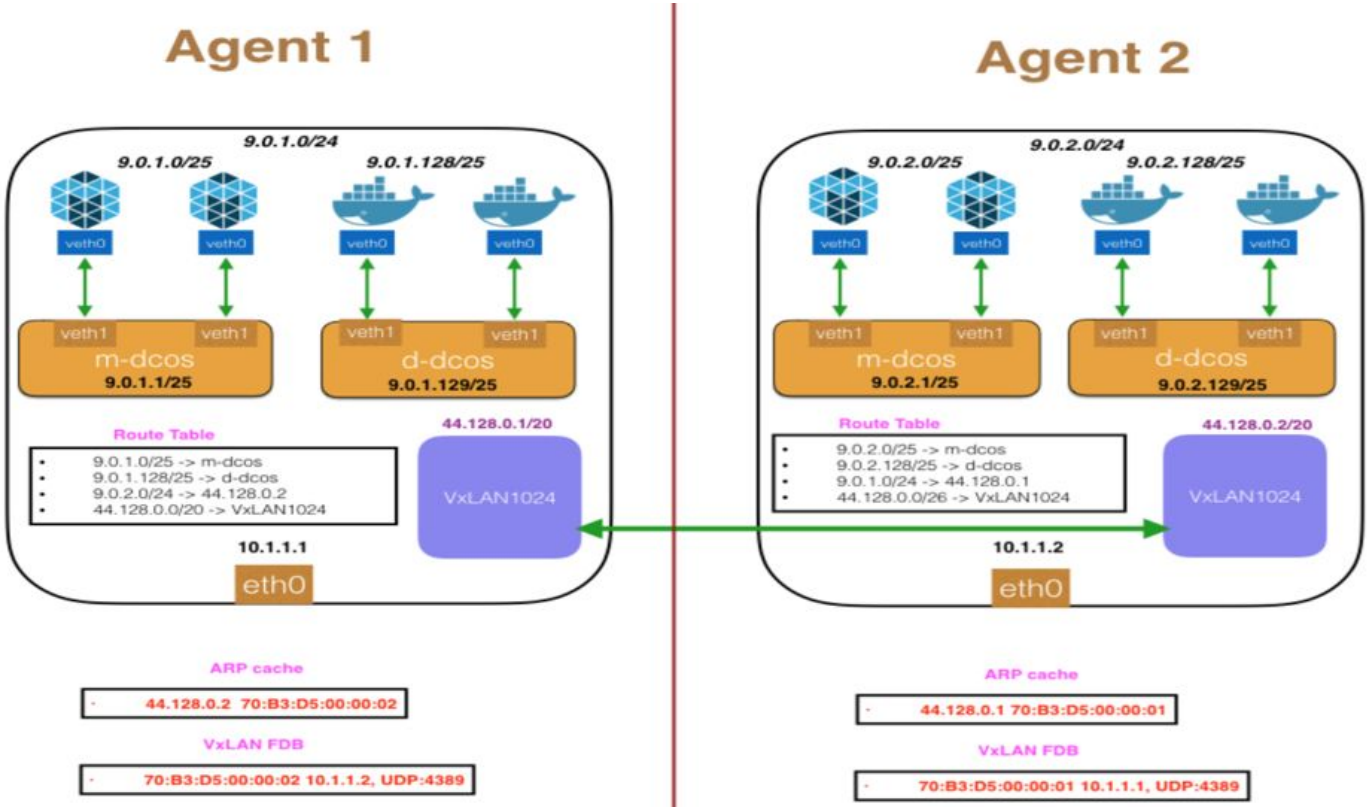
- Full port range for each container
- Connectivity for Mesos and Docker containers
- Intra-node IP discovery via navstar overlay orchestrator
- Overlay backend programmed via gossip protocol
- Highly available and scalable
- Cloud-agnostic virtual networking based on L3 VXLAN

CONFIGURATION

```
dcos_overlay_network:
   vtep_subnet: 44.128.0.0/20
   vtep_mac_oui: 70:B3:D5:00:00:00
   overlays:
     - name: dcos
       subnet: 9.0.0.0/8
       prefix: 26
```

$ curl -H Authorization:token=”<token>” https://leader.mesos/overlay-master/state

DC/OS OVERLAY



Lab 6d

VIRTUAL NETWORKS

LAB 6D - VIRTUAL NETWORKS

1. Login to the DC/OS UI and edit the previously created nginx app
2. Go to **Network** and select **Virtual Network: dcos** from **Network Type**
3. After the app redeploys, SSH into the master node

```
$ ssh -i <pem file> core@<mesos-master>
```

4. Curl the task-specific IP address

```
curl 9.0.4.130
```

SERVICE DISCOVERY & LOAD BALANCING

Tool	Type	When Should I Use It?
VIPs (Minuteman)	Minuteman is a layer 4 load balancer, which can be used for most TCP traffic for any Mesos task within a DC/OS cluster.	When you have internal-only TCP services which do not require layer 7 features.
Marathon-LB	Marathon-LB is an HAProxy based load balancer for Marathon only.	When you require external routing or layer 7 load balancing features. Examples of layer 7 features include: TLS termination, zero-downtime deployments, HTTP sticky sessions, load balancing algorithm customization, network ACLs, HTTP basic auth, gzip compression, and more.
Mesos-DNS	Mesos-DNS is a basic DNS-based service discovery tool that works with any Mesos task.	When neither VIPs or Marathon-LB are adequate, you can use the auto-generated Mesos-DNS names. For example, to discovery Marathon-LB you'd use Mesos-DNS by connecting to Marathon-LB with marathon-lb.marathon.mesos. Or, if you have UDP services such as StatsD.

MESOSPHERE

NEW AGE OF NETWORKING

SDN OpenFlow

NFV OVS **VXLAN**

CNI CNM **Geneve**

MESOS-DNS

Mesos-DNS provides service discovery within DC/OS cluster

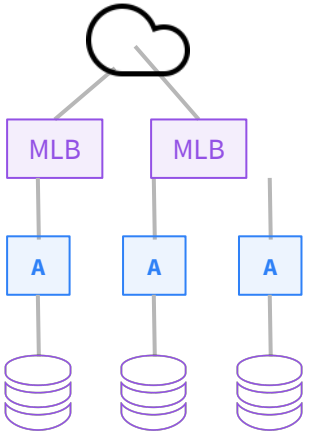
Allows services running on Apache Mesos to find each other using DNS

Mesos-DNS generates DNS records (A & SRV) based on periodic queries to the Mesos Master

Lightweight stateless service

Open-source: https://mesosphere.github.io/mesos-dns/

```
$ dig app.marathon.mesos
```



MESOS-DNS CONFIG

```
$ curl http://<mesos-master>:8123/v1/config | jq .
$ cat /opt/mesosphere/etc/mesos-dns.json

{
  "zk": "zk://zk-1.zk:2181,zk-2.zk:2181,zk-3.zk:2181,zk-4.zk:2181,zk-5.zk:2181/mesos",
  "refreshSeconds": 30,
  "ttl": 60,
  "domain": "mesos",
  "port": 61053,
  "resolvers": ["169.254.169.253"],
  "timeout": 5,
  "listener": "0.0.0.0",
  "email": "root.mesos-dns.mesos",
  "IPSources": ["host", "netinfo"]
}
```