Making New Friends
Apache Traffic Control + Varnish

Eric Friedrich
Content Distribution Architect
Disney Streaming Services - Who We Are

- Disney Streaming Services brings beloved characters, timeless stories, and epic sporting events to a global audience through world-class direct-to-consumer video services including Disney+ and ESPN+.

- Built Media Caching Solutions around Live Events
  - Wrestlemania 34, Super Bowl LI

- Leveraging Commercial Content Delivery Networks (CDN)

- VOD was a different experience
  - WWE Network VOD and HBO Now
  - Lower peaks but more sustained traffic
  - Diverse content library
  - Variety and library size impacted user experiences
Who I Am

- Previously at Cisco/Synamedia contributing to Traffic Control
  - Focus on turnkey Service Provider CDN product

- Currently with Disney Streaming Services

- Combining existing Varnish caching infrastructure with Apache Traffic Control
Topics

- Content Distribution at Disney Streaming Services
- Varnish Cache Basics
- Integrating Varnish in ATC
- Q&A
Content Distribution at Disney Streaming

- Focused on origin shielding for commercial CDNs and OpenCaching
- Mix of Varnish caches and L7 load balancers
- Custom control plane
Multiple Clusters of caches serve geographic regions

Clients are CDNs so they perform localization on viewers

Loadbalancers do consistent hashing and health checking on caches
Varnish Cache
Varnish Cache

- Varnish is a highly configurable reverse proxy
- Also Open Source! BSD-2 clause
- Configurable through Varnish Config Language (VCL)
  - Handles remapping, rewrites, TTLs, headers, custom logging, etc...
  - Pluggable backend selection
  - Backend health probing
- Separate TLS terminating proxy - Hitch or HAProxy
VCL Subroutines

- `vcl_recv()`
- `vcl_hash()`
- `vcl_hit()`
- `vcl_miss()`
- `vcl_backend_fetch()`
- `vcl_backend_resp()`
- `vcl_synth()`
- `vcl_deliver()`
CDN Request Processing Responsibilities

- Delivery Service Isolation
- Request Scrubbing/Normalization
- Backend Selection
- Cache Key Creation
- Cached Object TTL Management
- Setting Response headers
- Logging
VCL Delivery Service Isolation

- VCL Labels provide isolation between Delivery Services
- Labels are separate bundles of VCL specifying different behaviors
- Label is activated based on incoming request properties (i.e. Host header)

sub vcl_recv() {
    if (req.http.Host ~ "live.cdn.example.com") {
        return (vcl(live_ds));
    } else if (req.http.Host ~ "vod.cdn.example.com") {
        return (vcl(vod_ds));
    } else {
        return (synth(503));
    }
}
Request Normalization

- Stripping Request Headers
- Filter Request Method
- Strip query strings
- URL Rewrite

```vcl
sub vcl_recv() {
    unset req.http.Authorization;
    if (req.method != GET) { 
        return(synth(405));
    }

    set req.url = regsub(req.url, "\?.*$", "");
    set req.url = regsub(req.url, "^", "/req_prefix");
}
```
Backend Selection

import directors;

backend parent1 {
  .host = "192.168.0.1";
}

backend parent2 {
  .host = "192.168.0.10";
}

sub vcl_init() {
  new random_dir = directors.random();
  random_dir.add_backend(parent1, 1.0);
  random_dir.add_backend(parent2, 9.0);
}

sub vcl_recv() {
  set req.backend_hint = random_dir.backend();
}

sub vcl_backend_req() {
  set bereq.http.Host = "origin.example.com";
}
Varnish Consistent Hashing

- Requests are received by any cache in the cluster, i.e. “Edge” host
  - Shield fetches from Origin and caches locally
  - Edge proxies response back to client
- Shield host is determined by consistent hash of path
- Edge host is determined by client connection
Varnish Consistent Hashing

- Requests are received by any cache in the cluster, i.e. “Edge” host
  - Shield fetches from Origin and caches locally
  - Edge proxies response back to client
- Shield host is determined by consistent hash of path
- Edge host is determined by client connection
Varnish Consistent Hashing

- Requests are received by any cache in the cluster, i.e. “Edge” host
  - Shield fetches from Origin and caches locally
  - Edge proxies response back to client
- Shield host is determined by consistent hash of path
- Edge host is determined by client connection
Integrating Varnish in ATC
Integrating Varnish in ATC

- Motivation
- Configuration Generation
- Health Monitoring of Varnish
Integrating Varnish in ATC

- Prototyping custom solution as component of edge delivery strategy
- Extension of existing origin shield infrastructure
- Uses ATC Traffic Monitor for Health Protocol
- Uses ATC Traffic Router for client localization and redirection
Prototype Edge Delivery

CDN → Varnish → LB → Origin

Control Plane

OpenCaching → Varnish → TR/TM
Configuration Generation

- Implementing ATC TO authentication and APIs

- Hosting static files
  - (Deep)CZF
  - MMDB

- Added generation of
  - crconfig.json
  - monitoring.json
  - sslkeys.json

- Control Plane uses Apache Freemarker templates to generate VCL & ATC files
Config File Templates

- Freemarker combines context variables from DB with templates.
- Separating syntax of file from config variables eases development.
"deliveryServices": {
    <#list deliveryServices as ds>
    "${ds.id}": {
        "anonymousBlockingEnabled": "${ds.atcVpnBlock}",
        <#if ds.atcBypassFqdn?has_content>
            "bypassDestination": {
                "HTTP": {
                    "port": "80",
                    "fqdn": "${ds.atcBypassFqdn}"  
                }
            },
        </#if>
    },
</#list>
"domains": [
    "${{ds.dnsName}}.{{$cdn.domainName}}"
],
},
Health Monitoring of Varnish

- Implemented a Varnish Module (vmod) to replace astats
- Varnish needs a vmod to generate an HTTP response body
- Code references astats implementation when possible
  - Uses same /proc and sysfs interfaces for data gathering
- No DS stats yet, just load avg and NIC bandwidth
  - Need further instrumentation for Delivery Service specific stats
  - Likely based on another vmod for creating custom counters
Calling libvmod_astats

```python
import astats;

acl atc_tm_acl { "192.168.10.0"/24;}

sub vcl_recv() {
    if (req.url ~ "^/_astats") {
        if (client.ip ~ atc_tm_acl) {
            set req.http.astats = true
            return(synth(200, "OK"));
        } else {
            return synth(403);
        }
    }
}

sub vcl_synth() {
    if (req.http.astats) {
        synthetic(astats.info(...));
        return (deliver);
    }
}
```

- **vcl_recv()**
  - Matches URL with regex
  - Apply Traffic Monitor ACL
  - Signals astats mode to synthetic response with internal request header

- **vcl_synth()**
  - Calls astats with interface name
  - Astats internally generates JSON response
  - Body returned to client
Health Monitoring of Varnish

- Proof - Cache polling works
- Many pieces still under construction
  - IPv6
  - Multiple Interfaces sharing single IP - not bonding
  - Reporting as a single interface, with bandwidths summed

<table>
<thead>
<tr>
<th>Server</th>
<th>Type</th>
<th>IPv4</th>
<th>IPv6</th>
<th>Status</th>
<th>Load Average</th>
<th>Bandwidth (mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>varnish04-c01-ewr1</td>
<td>EDGE</td>
<td>true</td>
<td>false</td>
<td>REPORTED - available</td>
<td>0.1</td>
<td>0.11 / 25,000</td>
</tr>
<tr>
<td>varnish03-c01-ewr1</td>
<td>EDGE</td>
<td>true</td>
<td>false</td>
<td>REPORTED - available</td>
<td>1</td>
<td>0.11 / 25,000</td>
</tr>
</tbody>
</table>
TLS Certificate Handling

- TLS keys/certs are stored in Control Plane
- Control Plane publishes Hitch PEM bundle and Traffic Router sslkeys.json
- Found a small bug in Subject Alternate Name wildcard matching
Anycast Traffic Routers

- DNS servers are often run using Anycast
- For Traffic Router, embedded DNS server must bind to port 53 TCP and UDP on the Anycast Virtual IPs
- TR listen IP configuration goes in dns.properties and server.xml
  ```
  dns.udp.host=192.0.2.10
  dns.tcp.host=192.0.2.10
  ```
Conclusion

- Varnish Cache’s flexibility leads to easy integration with ATC
- Will hopefully do an Open Source libvmod-astats contribution
- ORT cache-side config generation could be extended to generate VCLs
Questions?