Federated Varnish & XCache Deployment with SLATE

Throughput Computing 2023
Madison, July 12, 2023

Ilija Vukotic, University of Chicago
Service Deployment Models

There are services that we would like running at most of the ATLAS sites (eg. Perfsonar)

**Standard way** of doing it is to ask sys admins to run it:

Adding a service incurs a significant cost in sys-admin time:

- Learning about service
- Configuring
- Monitoring
- Keeping it up-to-date

People using a service need to communicate with sites, explain changes needed, debug things that break due to skipped updates, etc. sometime take months.

This makes for a unreliable service, security issues, slow rollout of new features, stifles innovation.
Federated way of doing it:

- Have one or (better) two people that know the service inside-out.
- Give them a secure way to deploy, (re)configure, monitor, start/stop/update it, with minimal/no involvement of site’s personal.
- Site personal has only one-off things to do - NoOps.
- SLATE is one way to do Federated Ops.
SLATE: Services Layer At The Edge

- **SLATE** - a value added K8s distribution
  - Support for CVMFS, ingress controller (multi-tenant, scoped privileges), Prometheus monitoring, **curated application catalog w/ Github Actions**

- Site security & policy conscious
  - SLATE works as an unprivileged user
  - Single entrypoint via institutional identity
  - Site owner controls group whitelists & service apps; **retains full control**

- With OSG, WLCG, [trustedci.org](http://trustedci.org) & others worked to establish a "CISO compliant" security posture and **new trust delegation model**
SLATE - Adding a service

Assuming an old-fashioned app...

● Create docker image(s)
  ○ Directly from github using github actions, push them to a registry (DockerHub, OSG Harbor, CERN Harbor,...)
  ○ Check image scan reports.

● Create kubernetes deployments, services, ingress, etc. Test all works correctly (Docker Desktop).

● Create Helm chart
  ○ Basically decide what are the parameters that need to be configurable. Write instructions.
  ○ Add a few SLATE required lines.

● Add the chart to SLATE integration repository. Test.
● Add it to production repository.
SLATE - Managing a service

$ slate instance list
$ slate instance delete <instance name>
$ slate app install --group atlas-xcache --cluster uchicago-prod --conf MWT2.yaml xcache

Web interface

CLI

Kibana monitoring
ATLAS has two situations where data is remotely accessed:

- **Virtual Placement** - jobs scheduled to sites that have no input data
- **ServiceX** - a service that quickly filters, enriches, delivers data in multiple formats for semi-interactive analysis.

Both practically require caching input data for faster subsequent accesses.

- The data is primarily accessed via xroot protocol.
- XCache is a specifically configured XRoot server that caches blocks accessed (also used by OSDF, CMS, etc.).
XCache in SLATE

A rather complex application with multiple containers:

- Server itself
- Proxy renewal
- Rucio heartbeats
- Monitoring stream udp2tcp proxy

Special requirements:

- NodePort service (for performance reasons)
- Dedicated node:
  - Special label `xcache-capable: "true"`
  - Tainted `effect: PreferNoSchedule`
  - A lot of disks as JBODs
  - At least one NVMe for namespace
  - Good NIC (> 25Gbps)
Deploying XCache

Once a site approved application, informed me of disk mounts, IP and labeled node we:

- Prepare configuration. Most of it are defaults.
- Create slate secret (xcache service certificate)
  
  $ slate secret create --group atlas-xcache --cluster esnet-lbl --from-file userkey=xcache.key.pem --from-file usercert=xcache.crt.pem xcache-cert-secret

- Deploy XCache
  
  $ slate app install --group atlas-xcache --cluster esnet-lbl --conf ESnet.yaml xcache

- Check it works
  
  $ xrdcp -f
XCache in SLATE

- We need it at all US Tier2s, Tier1, Analysis Facility, several UK and DE sites.
- Update of all SLATE instances takes <10 min, non-SLATE deployments take days to update.

A data caching network deployed in less than 20 minutes.

Upgrades are as simple as re-deploying.
XCache monitoring

Several sources:

- SLATE
- gStream
- Panda
- Pilot
- Functional tests
Squid

- We use Squids for http caching. Squid is a **forward proxy**.
- Single threaded, quite old technology.
- We use them for two different purposes:
  - to cache Frontier requests
  - to cache CVMFS accesses.
- Most sites have the same cache do both.
- Sites are recommended to have two Squids in a round robin configuration.
- Usually configured with 32GB RAM and a persistent disk cache.
Varnish

- “Made for modern hardware. Working with kernel not against it.”
- While it is a reverse proxy, in our case it doesn’t matter as our origins are known.
- Very flexible - Varnish Cache Configuration Language (VCL) allows developers to specify request handling rules and set specific caching policies giving them a lot of control over what and how they cache.
- Nice modern monitoring.
- RAM only version is free, Disk persistence and federated versions are paid for.
Serving Frontier requests

- Varnish can be added to CRIC as a Squid and simply swapped in place.
- A VCL configuration.
  - ACL of WNs
  - List of backends
- Adding support for SNMP monitoring was 20x more effort.
Serving CVMFS requests

- This is configured on WNs. In our case simple Puppet configuration change.
- A VCL configuration.
  - ACL of WNs
  - List of backends
  - Some complications:
    - correctly handling the fact that not all stratum 1 serve all repos.
    - Correct handling of requests for repos that don’t exist anymore.

vcl 4.1;
import dynamic;
import directors;
{{- range $nindex, $be := .Values.backends }}
backend {{ $be.name }} {
  .host = "{{ $be.host }}";
  .port = "{{ $be.port }}";
}
{{- end }}

acl local {
  {{.Values.acl | nindent 4 }}
}

sub vcl_recv {
  if (!(client.ip ~ local)) { return (synth(405));}
  if (req.method != "GET" && req.method != "HEAD") {
    return (pipe);
  }
  {{- range $nindex, $be := .Values.backends }}
  if (req.restarts == {{ $nindex }}) {
    set req.backend_hint = {{ $be.name }};
  }
  {{- end }}
}

sub vcl_backend_fetch { unset bereq.http.host; }
sub vcl_backend_response {
  if (beresp.status == 404 ) {
    {{ $lb := last .Values.backends }}
    if (bereq.backend != {{ get $lb "name" }}) {
      set bereq.uncacheable = true;
      return (deliver);
    } else {
      set bereq.ttl = 180s;
    }
  }
}

sub vcl_deliver {
  if (resp.status == 404) {
    if (obj.uncacheable){ return(restart);}
  }
}
Varnish deployment

- Created two SLATE applications ([https://portal.slateci.io/applications](https://portal.slateci.io/applications)):
  - v4a - Varnish configured to serve Frontier requests
  - v4cvmfs - Varnish configured to serve CVFMS accesses
- Unlike XCache, straightforward Helm charts. Basically only one configmap, one deployment and one ingress.
- Both v4a and v4cvmfs currently in production at two US ATLAS Tier-2 centers: MWT2 (UC, IU, UIUC) and AGLT2
- For more than one year we saw no issues of any kind.
- Added to OSG Topology, configured in CRIC (v4a), configured on worker nodes (v4cvmfs).
Both Varnish and Squid are monitored in both Elasticsearch and ATLAS MRTG monitoring (cern.ch).

Reports request/fetch, I/O data rate, CPU usage, objects & file descriptors.

Response times can’t be compared as Squid rounds them to 0 seconds.

Varnish for Frontier node. File descriptors is 0 since it doesn’t use disk storage.
One of the Squid nodes. Serving both Frontier and Squid.
Performance in Elasticsearch

- Squid
- Varnish

Graphs showing performance metrics for Squid and Varnish.
Testing it - CVMFS

Squid was completely empty.

|x6 faster!|

<table>
<thead>
<tr>
<th>Transactions:</th>
<th>101391 hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability:</td>
<td>100.00 %</td>
</tr>
<tr>
<td>Elapsed time:</td>
<td>235.63 secs</td>
</tr>
<tr>
<td>Data transferred:</td>
<td>7059.98 MB</td>
</tr>
<tr>
<td>Response time:</td>
<td>0.05 secs</td>
</tr>
<tr>
<td>Transaction rate:</td>
<td>430.30 trans/sec</td>
</tr>
<tr>
<td>Throughput:</td>
<td>29.96 MB/sec</td>
</tr>
<tr>
<td>Concurrency:</td>
<td>22.02</td>
</tr>
<tr>
<td>Successful transactions:</td>
<td>93525</td>
</tr>
<tr>
<td>Failed transactions:</td>
<td>0</td>
</tr>
<tr>
<td>Longest transaction:</td>
<td>3.37</td>
</tr>
<tr>
<td>Shortest transaction:</td>
<td>0.03</td>
</tr>
</tbody>
</table>

Varnish was under regular production load.

<table>
<thead>
<tr>
<th>Transactions:</th>
<th>101391 hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability:</td>
<td>100.00 %</td>
</tr>
<tr>
<td>Elapsed time:</td>
<td>42.66 secs</td>
</tr>
<tr>
<td>Data transferred:</td>
<td>6894.09 MB</td>
</tr>
<tr>
<td>Response time:</td>
<td>0.01 secs</td>
</tr>
<tr>
<td>Transaction rate:</td>
<td>2376.72 trans/sec</td>
</tr>
<tr>
<td>Throughput:</td>
<td>161.61 MB/sec</td>
</tr>
<tr>
<td>Concurrency:</td>
<td>16.04</td>
</tr>
<tr>
<td>Successful transactions:</td>
<td>96796</td>
</tr>
<tr>
<td>Failed transactions:</td>
<td>0</td>
</tr>
<tr>
<td>Longest transaction:</td>
<td>4.01</td>
</tr>
<tr>
<td>Shortest transaction:</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Conclusions

● Thanks to SLATE it is easy to test/deploy new caching servers in production and at scale.
  ○ Simple to prepare an application, very simple app deployment/management, monitoring
● XCache is more stable, performant and up-to-date when deployed in Federated way.
● Varnish is definitely faster than Squid, needs less resources, it is easier to monitor. **Now in production at two US ATLAS Tier2s: MWT2 & AGLT2**
● Will be adding more applications to test physics data HTTP proxy caching: Nginx, [Apache Traffic Server (ATS)](http://trafficserver.apache.org), Nuster
Extras
Squid vs Varnish | What are the differences?
(stackshare.io)

**Squid:** A caching proxy for the Web supporting HTTP, HTTPS, FTP, and more. Squid reduces bandwidth and improves response times by caching and reusing frequently-requested web pages. Squid has extensive access controls and makes a great server accelerator. It runs on most available operating systems, including Windows and is licensed under the GNU GPL;

**Varnish:** High-performance HTTP accelerator. Varnish Cache is a web application accelerator also known as a caching HTTP reverse proxy. You install it in front of any server that speaks HTTP and configure it to cache the contents. Varnish Cache is really, really fast. It typically speeds up delivery with a factor of 300 - 1000x, depending on your architecture.
Testing it - Frontier

Harder to test (but will be done).

Squid was completely empty.

Varnish was under regular production load.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Squid</th>
<th>Varnish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions</td>
<td>177595 hits</td>
<td>177602 hits</td>
</tr>
<tr>
<td>Availability</td>
<td>99.96 %</td>
<td>99.96 %</td>
</tr>
<tr>
<td>Elapsed time</td>
<td>1612.89 secs</td>
<td>37.07 secs</td>
</tr>
<tr>
<td>Data transferred</td>
<td>1232.78 MB</td>
<td>1232.79 MB</td>
</tr>
<tr>
<td>Response time</td>
<td>0.26 secs</td>
<td>0.00 secs</td>
</tr>
<tr>
<td>Transaction rate</td>
<td>110.11 trans/sec</td>
<td>4790.99 trans/sec</td>
</tr>
<tr>
<td>Throughput</td>
<td>0.76 MB/sec</td>
<td>33.26 MB/sec</td>
</tr>
<tr>
<td>Concurrency</td>
<td>28.34</td>
<td>16.55</td>
</tr>
<tr>
<td>Successful transactions</td>
<td>177595</td>
<td>177602</td>
</tr>
<tr>
<td>Failed transactions</td>
<td>75</td>
<td>68</td>
</tr>
<tr>
<td>Longest transaction</td>
<td>24.38</td>
<td>4.97</td>
</tr>
<tr>
<td>Shortest transaction</td>
<td>0.21</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Huge difference!