Operational and Scaling Wins at Workday
From 50K to 300K Cores

OpenStack Summit
Berlin 2018
Edgar Magana
Moderator

Imtiaz Chowdhury
Architecture Overview and Use Cases

Howard Abrams
Instrumentation
Monitoring, Logging and Metrics

Kyle Jorgensen
Image Challenges
Clearing the Image Distribution Bottleneck

Sergio de Carvalho
API Challenges
Identifying and Fighting Scaling Issues
Workday provides enterprise cloud applications for financial management, human capital management (HCM), payroll, student systems, and analytics.
Our Story

OpenStack @ Workday
Our Journey So Far

- **2013**: Cloud Engineering Team formed
- **2014**: OpenStack Icehouse in Development - Internal workload
- **2015**: Deployment automation tools ready. - 2 Workday services in QA
- **2016**: First production workload
- **2017**: OpenStack Mitaka Development - 14 services - Production workload on Mitaka - 39 services
- **2018**: 50% of production workloads on OpenStack
- **2019**: -
Workday Private Cloud Growth

Revenue

US $273M
Our Private Cloud

- 5 Data Centers
- 45 Clusters
- 4.6k Compute Hosts
- 300k Cores
- 22k Running VMs
- 4k Active VM Images
How Workday Uses the Private Cloud

Immutable Images

Weekly update

Narrow Update Window

https://www.blockchainsemantics.com/blog/immutable-blockchain/
Architecture Evolution
## Key drivers for architectural evolution

<table>
<thead>
<tr>
<th>Driver</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scalability</strong></td>
<td>400% Scale API services horizontally</td>
</tr>
<tr>
<td><strong>High availability</strong></td>
<td>99% Make critical services highly available</td>
</tr>
<tr>
<td><strong>Downtime</strong></td>
<td>0 Provide upgrade path without affecting the workload</td>
</tr>
</tbody>
</table>
Control Plane

Clients

HAPerxy 1

HAPerxy 2

OpenStack Controllers
SDN Controllers

Stateless API services

Stateful services
Instrumentation

Logging and Monitoring and Metrics, Oh My!
Instrumentation Challenges

- No access to production systems: *full automation*
- Dispersed logs among multiple systems
- Sporadic issues with services:
  - “What do you mean RabbitMQ stopped!?”
- Vague or subjective concerns:
  - “Why is the system slow!?”
For each issue, we:

- Fixed the issue/bug
- Wrote tests to address the issue/bug
- Wrote a check to alert if it happened again
Our customers use our project (OpenStack), a particular way...

For each node in each cluster, test by:

- Start a VM with a particular image
- Check DNS resolves host name
- Verify SSH service
- Validate LDAP access
- Stop the VM

Rinse and Repeat
CRITICAL: Health validation suite had failures.

Connection Error - While attempting to get VM details. See logging system with r#3FBM for details.
Troubleshooting with Logs
Troubleshooting with Logs
Troubleshooting with Logs
Troubleshooting with Logs
There’s death, and then there’s *illness*…

What is this guy doing up here?

If all the compute node *load levels* are down here…
Dashboards to Track Changes

- **nbproc=1**
  - mc -set2
  - mc +set2

- **nbproc=2**
  - mc -set2
  - mc +set2

The graphs show the performance metrics over time for different configurations.
Transient Dashboards

What’s up with MySQL?
Instrumentation Takeaways

- Can’t scale if you can’t tweak. Can’t tweak if you can’t monitor.
- Collect and filter all the logs
- Create checks for everything...especially running services
- Invest in a good metric visualization tool:
  - Create focused graphs
  - Dashboards start with key metrics (correlated to your service level agreements)
  - Be able to create one-shots and special-cases
  - Learn how to accurately monitor all the OpenStack services
Image Distribution

Clearing the Image Distribution Bottleneck
Challenge: Control Plane Usage

Example - Nova Scheduler response time
Challenge: Control Plane Usage

Example - Nova Scheduler response time
Challenge: Control Plane Usage

Example - Count of deployed VMs
Large images: worst offender

~6GB
Image size

~1700
Instance count across DC’s
Problem

Many VM boots in short period of time + large images = bottleneck
Problem

Many VM boots in short period of time + large images = bottleneck
Many VM boots in short period of time + large images = bottleneck
Solution: Extend Nova API

curl https://<host>:8774/v2.1/image_prefetch -X POST \
...
-H "X-Auth-Token: MIIOvwYJKoZIQcCoII0sDCCDasdkoas=" \
-H "Content-Type: application/json" \
-d '{ "image_id": "d5ac4b1a-9abe-4f88-8f5f-7896ece564b9" }'
Solution: Extend Nova API

curl https://<host>:8774/v2.1/image_prefetch -X POST \
...  
-H "X-Auth-Token: MIIOvwYJKoZIQcCoIIOsDCCDasdkoaas=" \
-H "Content-Type: application/json" \
-d '{ "image_id": "d5ac4b1a-9abe-4f88-8f5f-7896ece564b9" }'
Solution: Extend Nova API

HTTP/1.1 202 Accepted
Content-Type: application/json
Content-Length: 50
X-Compute-Request-Id: req-f7a3bd10-ab76-427f-b6ee-79b92fc2a978
Date: Mon, 02 Jul 2018 20:52:37 GMT

{"job_id": "f7a3bd10-ab76-427f-b6ee-79b92fc2a978"}
Solution: Extend Nova API

curl https://<host>:8774/v2.1/image_prefetch/image/<image_ID>
...

OR

curl https://<host>:8774/v2.1/image_prefetch/job/<job_ID>
...
HTTP/1.1 200 OK ...

```json
{
    "overall_status": "5 of 10 hosts done. 0 errors.",
    "image_id": "d5ac4b1a-9abe-4f88-8f5f-7896ece564b9",
    "job_id": "f7a3bd10-ab76-427f-b6ee-79b92fc2a978",
    "total_errors": 0,
    "num_hosts_done": 5,
    "start_time": "2018-07-02T20:52:37.000000",
    "num_hosts_downloading": 2,
    "error_hosts": 0,
    "num_hosts": 10
}
```
Image Prefetch API Result

Before Cache hit

Avg 300 sec of VM boot time reduced
VM creation failure rate decreased by 20 %
HAProxy Bottleneck

Average Concurrent Image Download Time (in seconds)
with and without HAProxy -- image size: 6.5 GB

With HAProxy
Without HAProxy

20: 29.7
40: 43.5
60: 58.4
80: 67.4

595.7
Image Distribution: Key Takeaways

• Under heavy load, downloading images can be a bottleneck
  – Contribute image prefetch back to community

• HA Tradeoffs

• API Specific monitoring allows for unique insights
API Challenges

Identifying and Fighting Fire Scaling Issues
Nova Metadata API

14 seconds!

Average response time (sec)

Each VM makes > 20 API requests
Nova Metadata API & Database Transfer Rate

1 GB/sec

14 seconds!

Bytes sent (MB/sec)

Average response time (sec)

Each VM makes > 20 API requests
SELECT ... 
FROM (SELECT ... 
    FROM instances 
    WHERE instances.deleted = 0 
    AND instances.uuid = ? 
    LIMIT 1) AS instances 
LEFT OUTER JOIN instance_system_metadata 
    ON instances.uuid = instance_system_metadata.instance_uuid 
LEFT OUTER JOIN instance_extra 
    ON instance_extra.instance_uuid = instances.uuid 
LEFT OUTER JOIN instance_metadata 
    ON instance_metadata.instance_uuid = instances.uuid 
    AND instance_metadata.deleted = 0 
...
Instance Object-Relational Mapping

SELECT ...
FROM (SELECT ...
    FROM instances
    WHERE instances.deleted = 0
    AND instances.uuid = ?
    LIMIT 1) AS instances
LEFT OUTER JOIN instance_system_metadata
    ON instances.uuid = instance_system_metadata.instance_uuid
LEFT OUTER JOIN instance_extra
    ON instance_extra.instance_uuid = instances.uuid
LEFT OUTER JOIN instance_metadata
    ON instance_metadata.instance_uuid = instances.uuid
    AND instance_metadata.deleted = 0
    ...

...
SELECT ... FROM (SELECT ... FROM instances WHERE instances.deleted = 0 AND instances.uuid = ? LIMIT 1) AS instances LEFT OUTER JOIN instance_system_metadata ON instances.uuid = instance_system_metadata.instance_uuid LEFT OUTER JOIN instance_extra ON instance_extra.instance_uuid = instances.uuid LEFT OUTER JOIN instance_metadata ON instance_metadata.instance_uuid = instances.uuid AND instance_metadata.deleted = 0 ...

Expected result set (metadata union):
50 + 50 = 100 rows

Actual result set (metadata product):
50 x 50 = 2,500 rows!
SELECT ...

Expected result set (metadata union): 
50 + 50 = 100 rows

Actual result set (metadata product): 
50 x 50 = 2,500 rows!

https://bugs.launchpad.net/nova/+bug/1799298

Thanks to Dan Smith & Matt Riedemann!

AND instance_metadata.deleted = 0

...
Commit: Avoid lazy-loads in metadata requests (Feb 5 2016)

The metadata server currently doesn't pre-query for `metadata` and `system_metadata`, which ends up generating *two* lazy-loads on many requests. Since especially user metadata is almost definitely one of the things an instance is going to fetch from the metadata server, this is fairly inefficient.

--- a/nova/api/metadata/base.py
+++ b/nova/api/metadata/base.py

```python
def get_metadata_by_instance_id(instance_id, address, ctxt=None):
    ctxt = ctxt or context.get_admin_context()
    instance = objects.Instance.get_by_uuid(
        ctxt, instance_id, expected_attrs=['ec2_ids', 'flavor', 'info_cache'])
+       ctxt, instance_id, expected_attrs=['ec2_ids', 'flavor', 'info_cache',
+                                           'metadata', 'system_metadata'])
    return InstanceMetadata(instance, address)
```

Nova Pre-loads Metadata Tables (since Mitaka)
Reverting Metadata Pre-load

**Baseline test**
- Average response time (sec): 2.2 sec
- Bytes sent (MB/sec): 700 MB/sec

**No metadata pre-load**
- Average response time (sec): 0.5 sec
- Bytes sent (MB/sec): 345 MB/sec
Can We Do Better?

GET metadata

HAPerxy

VM

Nova Metadata API

Nova Metadata API

Nova Metadata API
Can We Do Better?

GET metadata

VM

HAProxy

Nova Metadata API

Nova Metadata API

Nova Metadata API

Database
Memcached!

GET metadata

HAProxy

VM

Nova Metadata API

Nova Metadata API

Nova Metadata API

Database
Enabling Memcached

Baseline test

- Average response time (sec): 2.2 sec
- Bytes sent (MB/sec): 700 MB/sec

Memcached enabled

- Average response time (sec): 0.2 sec
- Bytes sent (MB/sec): 400 MB/sec
No Metadata pre-load + Memcached

No metadata pre-load

Memcached enabled

Both
Root Causes

- **Heavy SQL query**: Product of metadata tables
- **No Memcached**: Repeated database fetching
- **HA architecture**: Multiple API servers fetching data through load balancers
- **Lots of metadata**: Booting many VMs simultaneously with many metadata
Fixes

- **Reduced SQL load**
  Rolled back pre-load of metadata tables
  (2-line code change)

- **Memcached**
  Enabled Memcached
  (3-line config change)

- **HA architecture**
  SQLProxy?
  Clustered Memcached?

- **Lots of metadata**
  Reduce (ab)use of metadata?
Questions?