Footprint Optimization

Eric Debeau (Orange), Adolfo Perez-Duran (Arm), Mike Elliott (Amdocs)
December 11, 2018
ONAP Resource Requirements

Casablanca largest release yet

- 240+ Pods
- 192GB memory for full deployment
- 112 CPUs
- 1 hour+ startup time
- 60GB+ (offline installer - Beijing)
Why do we care?

• Production-grade Deployments
  • Minimize time to instantiate when healing, scaling or migrating components
  • Limit impact of outages and upgrades

• Lower infrastructure costs

• Optimal use of resources in shared environments (i.e. Integration)

• Improved build times

• Fast deploy/update lends itself to automated integration tests

• Shorter develop-test-validate cycles accelerate innovation
What is being done?

• Application Resource Optimizations
• Normative Container Base Images
• Resource Limits
• Deployment Options
• Database Consolidation
• A lot of containers rely on Java code
  - JVM size has a big impact on memory footprint
  - No sharing JVM memory between containers

• Heap tuning for Java-based containers is possible
  - -Xxms: for minimal heap size
  - -Xxmx: for maximal heap size

• Default Heap size defined can be defined in Helm Charts
  - Defined for most of containers
  - Some Heap size still hard-coded in DockerFile scripts and can not be optimized for small deployments
Application Resource Optimizations: JVM configuration in Helm Charts

```yaml
config:
  cassandraUsername: root
  cassandraPassword: Aa123456
  cassandraJvmOpts: -Xmx2536m -Xms2536m
```

Some Examples used for Cassandra

```yaml
config:
  javaOptions: "-Xdebug ... -Xmx1536m -Xms1536m"
  cassandraSslEnabled: "false"
```

```yaml
config:
  heap:
    max: 512M
    min: 100M
  jvmOpts: -Dcassandra.consistent.rangemovement=false
```
Application Resource Optimizations: recos

• Use JVM options in Helm Chart
  - To be used in environment variables when Running the Docker container
  - Can be overridden during deployment (e.g. using ENV variables)
  - Avoid conflict with resource limits
    • Java Memory Size = Heap size + Compiled code + Threads/GC data
    • Resource limit request to be aligned with these constraints

• Use Java 10+
  - Better integration with Docker

• Prefer Java framework springboot to reduce the memory
Application Resource Optimizations: define optional sub-component

• Every project embeds an increased number of containers
  - For some projects, all sub-components are not mandatory to run a dedicated use-case

• Better documentation required to define the role of every container
  - Mandatory to run basic feature
    • Including all sub-components during the installation phase
  - Optional to run optional feature
<table>
<thead>
<tr>
<th>Images</th>
<th>Build time</th>
<th>Deployability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>Long</td>
<td>Low</td>
</tr>
</tbody>
</table>
ONAP Normative Container Base Images

- Reduce Footprint
- Support Multi-CPU Architecture
- Improve Deployability
Alpine Base Images

Reduce Footprint

Support Multi-CPU Architecture

Improve Deployability
Effect of Migrating to ONAP Normative Images

OOF/HAS Base Image Migration

Reduction in image size: 64%
1. Base image

```
FROM python:2.7-alpine
```

2. Package manager

```
RUN apk add --no-cache curl \
  gcc \
  libffi-dev \
  linux-headers \
  musl-dev \
  git \
  libxml2-dev \
  libxslt-dev \
  openssl-dev \
  py-setuptools \
  unzip \
  wget \
  xvfb
```

3. Libraries
What to expect
ONAP Normative Container Base Images
Project Engagement Process/Workflow
Dublin Scope
Resource Limits

- Introduced in Casablanca
  - improves pod scheduling
  - avoids excessive Pod evacuations

- Intention behind flavors:
  - small -> low resources for dev
  - large -> higher resources for production
  - unlimited = take what is needed

- Refinement needed in Dublin

- Ask for project teams to define and manage
Deployment Options

Development
• development and functional testing only
• onap-dev.yaml
  • flavor: small - minimal resource allocation
  • no clustering (default chart configuration)
  • application-specific optimizations for dev and test
  
  ```bash
  helm deploy dev release/onap -f onap-dev.yaml
  ```

Production
• larger resource allocation for deployments under load
• onap-prod.yaml
  • flavor: large - targets production deployment
  • clustering enabled for High-Availability and scaled for load
  • application-specific optimizations for prod and load
  
  ```bash
  helm deploy prod release/onap -f onap-prod.yaml
  ```
Database Consolidation

Components that use the same database technology, can share a single cluster with separate schemas and credentials

Benefits:
- Reduces the ONAP platform footprint
- Common helm charts limit effort required by individual projects
- Project teams share a common redundancy strategy
- Simplifies cluster storage and management across the deployment

Three Steps:
1. Common DB Charts:
   - kubernetes/common/postgres
   - kubernetes/common/mysql
2. Clustered DBs:
   - kubernetes/common/mariadb-galera
3. Shared DBs:
   - Common DB instance, separate tables
How many databases are in ONAP?

27 Databases in Casablanca!

- 3 Mariadb
- 5 Mariadb-Galera
- 1 mySQL
- 5 Cassandra
- 3 postgresSQL
- 1 Mongodb
- 1 JanusGraph
- 1 Titandb
- 5 Elasticsearch
- 2 Redis
Database Consolidation

- Focus will be on migrating to a single shared Mariadb-Galera cluster
- Scalable based on needs
### Call to Action

<table>
<thead>
<tr>
<th>Community and PTLs</th>
<th>TSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embrace recommendations and collaborate with the teams leading the effort.</td>
<td>Make compliance with recommendations mandatory for Dublin M3.</td>
</tr>
</tbody>
</table>

#### Application Resource Optimizations
- Normative Container Base Images
- Resource Limits
- Deployment Options
- Database Consolidation