The infamous “Example 4” refactored:

01 name: VitaMinder  
02 components:  
03 -  
04 name: VitaMinder WAR  
05 type: com.example.java:WAR  
06 content: { href: vitaminder.war }  
07 requirements:  
08 -  
09 name: App Server  
10 capabilties:  
11 -  
12 type: com.example:HostedOn  
13 javaVersion: [1.6,)  
14 -  
15 type: com.example.java:JDBC  
16 version: 4.0  
17 injectionMode: CDI  
18 -  
19 name: JDBC Target  
20 fulfillment: id:db  
21 -  
22 name: VitaMinder SQL  
23 type: com.example.sql:SqlScript  
24 content: { href: vitaminder.sql }  
25 requirements:  
26 -  
27 name: SQL Service  
28 fulfillment: id:db  
29 capabilties:  
30 -  
31 type: com.example.db:SQL  
32 version: SQL:2008  
33 globalRequirements:  
34 -  
35 name: VitaMinder DB  
36 id: db  
37 capabilities:  
38 -  
39 type: com.example.db:RDBM  
40 -  
41 type: com.exampl.db:Replication  
42 replicas: 2  
43 strategy: com.example.db:Optimistic

# Changes:

1. Move “distinguishing name” of a requirement from top-level attribute (requirement.type) to second level attribute (requirement.capabilities.type).
2. Requirements are aggregations of desired capabilities.
3. Added “globalRequirements” section for common requirements.

Advantages over previous Oracle proposal (chat room 6/12/2103):

* Requirements are no longer distinguished by a single name (e.g. “com.example.db:RDMB”). This allows the development of requirement specifications (their structure, semantics, registration, etc.) to take place in a more decentralized manner.
* In cases where a requirement is defined using multiple capabilities, the failure paths for non-comprehension of capability types are more graceful. For example, as a platform implementation I may not understand the capability type “com.example.db:SQL” but I may understand “com.example.db:RDBM” so, while I may not be able to auto-wire your components, I can present you with a list of the PCTs that provide the “com.example.db:RDBM” capability. This is much better than ignoring the entire requirement and leaving you to sift through all my PCTs.

Advantages over precious Cloudsoft proposal (<https://www.oasis-open.org/apps/org/workgroup/camp/download.php/49422/camp-spec-v1.1-wd10-issue-4-v4.doc>):

* Makes explicit the notion of a requirement as an aggregation of fine-grained capabilities.
* Doesn’t conflate the specification of requirements with the description of components.

Disadvantages:

* More lengthy/nested.

# Other Examples

## A Component-Less Deployment Plan

00 name: Starter Ruby App  
01 gobalRequirements:  
02 -  
03 name: Ruby Runtime  
04 capabilities:  
05 -  
06 type: com.example.ruby:RubyRuntime  
07 version: 1.9.3  
08 -  
09 name: Rails Framework  
10 capabilities:  
11 -  
12 type: com.exampl.rails:RailsRuntime  
13 version: 3.2.\*  
14 -  
15 name: Database  
16 capabilities:  
17 -  
18 type: com.example.db:RDBM  
19 -  
20 name: Git Repo  
21 capabilities:  
20 -  
21 type: com.example.git:GIT

## Two WAR files that will share a common app server

00 name: Minder  
01 components:  
02 -  
03 name: VitaMinder WAR  
04 type: com.example.java:WAR  
05 content: { href: vitaminder.war }  
06 requirements:  
07 -  
08 name: App Server  
09 fulfillment: id:appServer  
10 name: CalorieMinder WAR  
11 type: com.example.java:WAR  
12 content: { href: calorieminder.war }  
13 requirements:  
14 -  
15 name: App Server  
16 fulfillment: id:appServer  
17 globalRequirements:  
18 -  
19 name: Common App Server  
20 id: appServer  
21 capabilties:  
22 -  
23 type: com.example:HostedOn  
24 javaVersion: [1.6,)

## Two WAR files that use different, but functionally identical app servers

00 name: Minder  
01 components:  
02 -  
03 name: VitaMinder WAR  
04 type: com.example.java:WAR  
05 content: { href: vitaminder.war }  
06 requirements:  
07 -  
08 name: App Server  
09 capabilties:  
10 -  
11 type: com.example:HostedOn  
12 javaVersion: [1.6,)  
13 -  
14 name: CalorieMinder WAR  
15 type: com.example.java:WAR  
16 content: { href: calorieminder.war }  
17 requirements:  
18 -  
19 name: App Server  
20 capabilties:  
21 -  
22 type: com.example:HostedOn  
23 javaVersion: [1.6,)

1. Think of “requirement aggregations” as capabilityProfiles and as a way of reusing and standardizing requirements from the app designer perspective.
2. Did not see value of aggregating capabilities in components since nothing can reference them and the existing semantics already implies all requirements (under capabilities or not) are expected to be satisfied. It’s still possible to specify requirements in-line as before.

01 name: VitaMinder  
02 components:  
03 -  
04 name: VitaMinder WAR  
05 type: com.example.java:WAR  
06 content: { href: vitaminder.war }  
07 requirements:  
08 -  
09 name: Container  
10 fulfillment: id:as  
11 -  
12 name: JDBC Target  
13 fulfillment: id:db

14 # We still can use component-specific in-line requirements

15 -

type: com.example:SomeJEEFeature

mode: session  
21 -  
22 name: VitaMinder SQL  
23 type: com.example.sql:SqlScript  
24 content: { href: vitaminder.sql }  
25 requirements:  
26 -  
27 name: SQL Service  
28 fulfillment: id:db  
33 capabilityProfiles:  
34 -  
35 name: VitaMinder DB  
36 id: db  
37 capabilities:  
38 -  
39 type: com.example.db:RDBM  
40 -  
41 type: com.exampl.db:Replication  
42 replicas: 2  
43 strategy: com.example.db:Optimistic

-

type: com.example.db:SQL  
 version: SQL:2008  
08 -  
09 name: App Server

09 id: as  
10 capabilties:  
11 -  
12 type: com.example:WebContainer  
13 javaVersion: [1.6,)  
14 -   
15 type: com.example.java:JDBC  
16 version: 4.0  
17 injectionMode: CDI