Testing the Deployment API

The CDDLM deployment API is a SOAP/WS-RF API to deploy systems. This is the recommended test plan for validating an implementation, or a deployment.

These are mostly functional tests; the underlying SOAP stack and language implementation(s) have to be present before these tests can take place,

Portal tests

The following tests do not attempt to deploy a valid system.

1. Get portal state

Examine all the properties of a portal; including languages and notification mechanisms

Dependencies;

Assertions:

- CDL is listed as a supported language
- WS-N is listed as a supported notification mechanism.

2. Create a System

Create a system to deploy to, then destroy it immediately.

Assertions:

- The system EPR returned is a valid URL.
- The system EPR responds to a <Destroy/> operation.

3. List created systems

After creating a system, verify that it is in the list of created systems by looking up its ResourceID in the list of deployed systems.

4. Create a system with invalid hostname hint.

Create a system as above, this time with an invalid hostname hint. The hint should be ignored and a valid EPR returned as before.

5. Create a system with a valid hostname hint.

Create a system as above, this time with an valid hostname hint. The EPR should contain that hostname.

This test is hard to perform, as it needs a set of two or more (potentially virtual) hosts, all part of the deployment infrastructure.

6. Verify uniqueness of ResourceIDs

ResourceIDs are required to be strongly unique. This is hard to test. Simple checks can verify that during a single session, unique identifiers are returned. Long-term tests are more thorough, but harder to perform. Perhaps a unit test verifying that identifiers are always unique would suffice.

Tests should also verify that the same EPR is not reused.

7. Verify destruction

After a <destroy/> operation has destroyed a System endpoint, verify that the endpoint no longer responds to operations.

8. Verify multiple systems can be created.

Verify that a number of systems can be created at the same time.

Paired-portal tests

The deployment API is designed to support multiple portals providing access to the same set of systems. If the implementation enables this, then it must be tested

9. Create a system on one portal, view the listing in another

After creating a system, look in the "deployed systems" property of the peer portal for a system with the same ResourceID.

10. Create a system on one portal, recreate the EPR in another

After creating a system, <lookup />on the peer portal for a system with the same ResourceID, receiving the EPR. Verify the EPR refers to the same item by comparing ResourceID.

11. Destroy one EPR, verify that returned by the other portal now fails with some error.

After destroying one of the two portal EPRs obtained above, verify that a <destroy/> call to one EPR prevents the other EPR from responding to requests such as property read operations.

System Creation Notification tests

The following tests require WS-Notification support in the test harness.

Test systems may wish to provide a private property listing all eprs subscribing to a system or portal EPR. This will enable the outcome of these tests to be easily verified.

12. Subscribe to System creation events

The test system can subscribe via the portal for system creation events.

13. Unsubscribe

After subscribing to events, An unsubscribe should unsubscribe them. After unsubscribing, creating a new system must not result in an event being sent to the (now unsubscribed) endpoint.

14. Create a System while subscribed.

While subscribed to a portal for creation events, create a system.

Assertions:

- A system creation event is received.

15. Create a System while subscribed with an invalid endpoint

Subscribe to a portal for creation events with a notification EPR that is invalid. There are the following types of invalid portal

- Nonexistent URL (e.g. "http://notifications.example.org/epr1").
- endpoint with no open connection (e.g. "http://localhost:8081/epr1"), assuming port 8081 is closed
- 404 URL (e.g. "http://deployment.sourceforge.net/example/epr1").

Assertions:

- In all cases, the notification delivery failure is noted, the endpoint is no longer subscribed

16. Create a System while subscribed to a broken endpoint

Subscribe to a portal for creation events with a notification EPR that is broken. There are the following types of broken endpoint

1. EPR always responds with a SOAPFault of some kind
2. EPR keeps the connection open, and does not terminate the connection.

Test #1 should fail as in the previous test. Test#2 should eventually time out, and be recorded by the portal as a failure.

17. Multiple system event subscribers

Multiple EPRs can be subscribed to system events; events will be delivered to all. When one endpoint unsubscribes, events will still be delivered to the remainder.

18. Invalid XML posts

Each SOAP Operation must be tested with

1. XML documents that do not match the schema. These must be rejected.
2. XML documents that contain extra data where extra data is permitted. These must be accepted.
3. The portal must also reject SOAP actions of the systems themselves, such as an initialize request.

System configuration

Tests that are performed against a created System endpoint. These probe the endpoints' language abilities, and the support for options.

19. Initialize with an invalid language

Try and initialize a system with an invalid language: the request must be rejected

20. Initialize with a URL and valid language

Try and initialize a system with an invalid URL: the request must be rejected. The types of URL are

- nonexistent hostname
- non-responding host
- Blocking endpoint.
- 404 or other HTTP error

All of these should fail.

21. Initialize with descriptor as an attachment

If attachments (DIME, MTOM or SwA) are supported, then deployment using the supported attachment mechanisms must be tested. The tests in such as case are

- Deploy with incorrect URI -> failure
- Deploy request with inline document and attachment ->Unknown action. Ignore the attachment?
- Deploy request with very large (128+MB) attachment (stress test). This may be rejected as too large, or it may be accepted. Most of the request can be generated white space.
- Deploy request that has an include statement whose URI is resolved against another attachment.

22. Initialize with invalid mustUnderstand=true option

The initialization request must include a mustUnderstand option that is not understood. The response must indicate that the option was not understood.

23. Initialize with invalid mustUnderstand=false option

Presentation of this option must not result in a not-understood error. This requires a valid deployment descriptor as a prerequisite.

Deploy Invalid Configurations

The implementation must parse and reject those deployment configurations that can be assessed as invalid during the initialisation phase. That includes

- configurations with looped dependencies
- configurations with unresolved references
- configurations whose values does not match the (enforced) type system.
- Inclusion of a non-resolvable URI

Deploy Valid Configurations

In order to test valid deployments, we will need test components whose behaviour are well defined, and which are implemented on all test systems.

A viable set of components would be

- File component to map to a file; a liveness test can verify its existence if the mustExist flag is set.
- TempFile component extending File; this creates a temporary file on deployment, and optionally delete it when it exits. A text value can be supplied.
- Add component: adds two numbers set as attributes, returning the result as an attribute.
- Concatenate: concatenates two strings
- Delete files
- Fault raising. A component that can raise a fault on initialization, liveness or termination.

with that we can do the following tests

24. deploy a sequence of operations

e.g. Tempfile then deleteFile, or a chain of concatenates, with the result obtainable by asking the underlying components.

25. Deploy a distributed configuration

A configuration in which different components are deployed on different systems, with late binding references between them. This verifies that communication between nodes is functional.

26. Raise faults during the process.

Raise faults during deployment through the fault component. This can verify that <Ping/> works, and can test fault-handling policies of parent containers.

System notifications

27. Subscribe to lifecycle events

A client can subscribe to system lifecycle events.

28. Receive lifecycle events when a system is destroyed

A client that is subscribed to lifecycle events, receives a notification when a system is destroyed.

29. Receive lifecycle events when a system fails

With the Fault component, we can expect to be notified when a system enters the failed state.

30. Support multiple event listeners per system.

More than one endpoint can subscribe to lifecycle events; all will receive them. A more advanced test would verify that the failure/blocking action of one endpoint will not cause the other recipients to be denied their messages.

31. Receive lifecycle events when a system enters the running state.

Subscribers to lifecycle events receive notifications when a system enters the running state.