WSRF-based Deployment API

This is an outline of the proposed changes to the deployment API.

Core Changes

1. Split operations into server operations and application operations.
2. Server operations are applied to the server "portal" EPR -core operation will be to create application EPRs.
3. Application EPRs refer to applications. After creating one, it is initialized by deploying something into the application, then it can be started, terminated and ultimately destroyed.
4. Server state will be accessible as WS-Resource Properties: this includes static info, plus dynamic state, including the list of running applications.
5. Application state will be accessible as properties off the application EPRs; state will primarily consist of: the current lifecycle state, the EPR of the top-level deployed component in the component graph, the deployment data.
6. WS-Notification can be used to subscribe to changes in application state.
7. WS-BaseFault extensions will be defined for a deployment fault, with further extensions of that for: a nested SOAP fault; a parser fault (with file/line info), a component fault (with component path). We may define further extensions later.
8. The lifetime of the portal EPR will be determined by any lifetime metadata in the EPR.
9. The lifetime of the application EPR will be defined by that of the application itself, and then some. That is, after the app is terminated, it can be destroyed.
10. If a portal EPR is destroyed, or the portal itself becomes unavailable, applications are expected to continue running, unless the failure of the portal implicitly brings down the application itself. That is, all application state is stored in the application, not the portal, and failure of the portal does not kill the app, unless it is on the same host/process.
11. If a cluster has multiple portals, then it may be that each portal grants access to the applications in the cluster, through EPRs that refer to each portal instance. In this scenario, different EPRs may refer to the same application.
12. Each application therefore also has a GUID, which is required to be strongly unique. If two applications EPRs, resolve to applications with the same GUID, they are determined to be referring to the same application instance.

One aim of this design is to avoid having the failure of a portal server to imply that the application has failed; it can continue running and be reachable through other portal instances. The GUID exists to enable different application EPRs, even those across different portals, to be tested for equality.

Explicitly left out of the spec

These are either WSRF options, or things that WSRF doesn't cover

- There will be no requirement to support WS-MetaData Exchange
- No requirement to implement get/set multiple properties, and if implemented, no requirement for atomicity.
- No requirement to implement QueryResourceProperties
- If a component supports WS-MetaDataExchange, there is no requirement for the list of resources and operations above and beyond the set we define to remain constant for any period of time. That means an application EPR can add and remove attributes, and the metadata could dynamically list those attributes, but they could be removed at any time.

Designing for fault tolerance

We want to avoid the problem of the failure of the portal either terminating apps, or making the applications unreachable. Without relying on fault-tolerant URLs (load balancers, round robin DNS, Rendezvous/ZeroConf hostnames), we have to allow callers to go from the URL of a missing/unreachable host to a valid URL.

The WS-RF docs hint at using WS-Policy to do this, but provide no specifics, and nor does WS-Policy itself.

It is probably out of scope to deal with this, but we do not want to prevent something implementing proper fault-tolerant endpoint renewal. What we can mandate is that failure of the portal should not trigger termination of applications, unless the same underlying system failure also took out the applications.

Another aspect of this is that we cannot use EPR comparison to test for application EPRs being different. Two different portals could refer to the same application in the farm, with correspondingly different EPRs. Application EPRs will have an id property to use in comparison; a case-sensitive comparison of this ID MUST be sufficient for determining if two EPRs point to the same application, or different ones.

Portal EPR

This is the primary deployment front end. With it, callers can create applications, and enumerate those that are deployed.

Properties

Name	*Type*	*Meaning*
`staticInfo`	`xsd:any`	static server info
`dynamicInfo`	`xsd:any`	dynamic server info
`applications`	`xsd:list`	List of wsa:EPRs of applications
WS-Resource Lifetime Properties		

Operations

Name	*In*	*Out*
`create`	`xsd:any`	wsa:EPR of new app
`lookup`	`string`	wsa:EPR
WS-Resource Lifetime Operations		

Application EPR

This represents an application that has been created.

It has an ID property that MUST be strongly unique. That is, it SHOULD be unique for a single deployment of a single application, MUST NOT be re-used, and SHOULD be unique even between different deployment installations. The recommended approach is to use a `guid:` URI with a properly generated GUID.

Properties

Name	*Type*	*Meaning*
`name`	`xsd:string`	user-defined name (optional)
`id`	`xsd:uri`	unique name
`deploymentInfo`	`xsd:any`	deployment data
`state`	`xsd:enum state`	current app state
`stateInfo`	`extra state info`	most recent extra state info
`terminationInfo`	`(message, fault, xsd:any)`	termination info
`started`	`xsd:dateTime`	started time
`terminated`	`xsd:dateTime`	end time
`rootComponent`	`wsa:EPR`	root component of the app
WS-Resource Lifetime Properties		

NB, could have a list of transitions+timestamps, useful for history.

Operations

Name	*In*	*Out*
`init`	`xsd:any`	
`start`		
`terminate`	`xsd:string Message`	
`ping`	`void`	state info
`resolve`	`path`	xsd:any

Name	*In*	*Out*
WS-Resource Lifetime Operations		
WS-Notification Operations		

Notification

WS-N is a lot of extra work, which is why I'm reluctant to make it a MUST, but if we are going to have to do it in the components, we should go ahead and require it in the deployment API.

Fault tolerance is an extra complication; because the origin EPR is included in each notification, the sender must know its EPR. If an app is visible through >1 portal, it must know the EPR used by a subscriber, and return that EPR with the request. If the application/portal EPR is changed when the subscriber switches to a different portal, then the EPRs in the notifications must also be updated.

The easy way to do this is to have the portal manage the notifications, and do not require subscription EPRs to be fault tolerant. If a portal fails, the subscriptions are lost. As they are renewable anyway, this is no real hardship. The biggest risk that a subscriber does not know that a subscription has been lost, and so an event is missed.

Policy

- Implementations SHOULD support a notification mechanism, and that notification mechanism SHOULD be WS-N *-or make this MUST?*
- If WS-N is supported, then the implementation must support the topics defined below, on the identified EPR types.
- Implementations MAY also support Terminate notification events of WS-ResourceLifetime, which are raised after an EPR is destroyed.
- There will be one notification for lifecycle events of applications
- There will be one notification for the portal EPRs, which is raised when an application is created.
- There is no guarantee of fault tolerant subscriptions. Implementations MAY include WS-Policy metadata that informs callers how to renew subscriptions in the event of system failure.

Specifics

1. A WS-TopicSpace that contains one topic: lifecycle events ; applications support this topic.
2. A WS-TopicSpace that contains one topic: application addition events. This is for the portal EPR.