[YARN-4757] Simplified discovery of services via DNS mechanisms

Table of Contents

(1) Introduction
(2) The YARN Service Registry
(3) Key Requirements for a DNS-based Service Registry
(4) Discovery via DNS Records
    Applicable Record Types
    Proposal for the DNS records
        Record name elements
        Per-Application records
        Per-Container records
    Registry Service Record Parsing Guidelines
    Custom Aliases
(5) Architecture
    Overview
    Architecture Details
    High Availability
    On-premise deployment
        Zone Server vs Primary Server
        Supporting reverse-lookups
        Optional Features
    Security/DNSSEC Support
(6) Implementation details
    Implementation Choices
    Configuration
(7) Open Issues
    Application RUNNING state vs DNS record availability
    IPv6 Reverse Lookup Support
    Full DNSSEC support
    HA Support
    Signature configuration
    Zookeeper Dependence
(8) Appendix A: Existing Service Registry Functions
    Registration of information
    Lookup

(1) Introduction

The YARN service registry (YARN-913) today provides facilities for applications to register their endpoints and for clients to discover them.

However, existing lookup mechanisms of the YARN Service Registry are currently limited to a registry specific (java) API and a REST interface. In practice, this makes it very difficult for wiring up *existing* clients and services. For instance, the dynamic configuration of dependent endpoints is not easy to implement using the present registry's read-mechanisms, *without* code changes to existing services. Concretely, as an example, in order to make an existing data-application point to a dynamic HBase instance running on top of YARN, one has no choice but to change the application code to either use REST or Java APIs of YARN's service registry.

A good solution to this problem is to expose the registry information through a more generic and widely used discovery mechanism: ***DNS***. Service Discovery via DNS uses the well known DNS interfaces to browse the network for services.

In this document, we propose a new service that implements this solution: the YARN DNS service (yDNS, or alternatively, DNS for YARN - DARN ;)). Its goal is to provide a standard DNS interface to the information posted into the YARN Registry by deployed applications. The DNS service will serve the following functions:

1. **Exposing existing service-discovery information via DNS -** Information provided in the current YARN service registry's records will be converted into DNS entries, thus allowing users to discover information about YARN applications using standard DNS client mechanisms (for e.g. a DNS SRV Record specifying the hostname and port number for services).

2. **Enabling Container to IP mappings -** Enables discovery of the IPs of containers via standard DNS lookups. Given the availability of the records via DNS, container name-based communication will be facilitated (e.g. 'curl http://myContainer.myDomain.com/endpoint'). (*Note that this is straightforward when the containers are tied to their own network-interfaces like with Docker containers. For containers without their own networks, we can still support the forward container-id -> host-ip mapping, but the reserve lookups obviously are problematic - one IP will map to multiple containers*).

(2) The YARN Service Registry

The YARN Service registry in a Hadoop cluster allows deployed-applications to register themselves and publish the means of communicating with them.

- Applications (specifically ApplicationMasters today) can post information to the registry through a Java Registry library.

- Client applications can then locate services (through Java and REST APIs exposed by the Service Registry) and use the binding information to connect with the services' network-accessible endpoints, e.g. REST, IPC, Web UI, Zookeeper quorum+path or some other protocol.

Please refer to Appendix A for more a slightly more detailed information concerning the YARN Service Registry and the service records associated with application registration. You can also read the original design document for YARN Service Registry attached on YARN-913 JIRA.

(3) Key Requirements for a DNS-based Service Registry

In this section, we lay down a few key requirements that we expect from a DNS based registry for YARN.

The existing Service Registry can be leveraged as the source of information for a DNS Service. The following is a short summary of core-functions desired of a DNS-based Registry:

Functional properties

1. Support creation of DNS records for end-points of the deployed YARN applications
2. Record names shall remain unchanged during restart of containers and/or applications
3. Support reverse lookups (name based on IP).
4. Support security using the standards defined by The Domain Name System Security Extensions (DNSSEC)
5. Highly available
6. Scalable - The service should have the responsiveness (e.g. low-latency) required to respond to DNS queries (timeouts yield attempts to invoke other configured name servers). Thus, the throughput and ability to handle the workloads of a large cluster are of paramount importance.

Deployment properties

1. Support integration with existing DNS assets (e.g. a corporate DNS server) by acting as a DNS server for a Hadoop cluster zone/domain. The server is not intended to act as a primary DNS server or will not be required for forward requests to other servers.

2. The DNS service shall expose a port that can receive both TCP and UDP requests per DNS standards. The default port for DNS protocols is in a restricted, administrative port range (53), so the port must be configurable for deployments in which the service may not be managed via an administrative account.

(4) Discovery via DNS Records

In this section, we lay out our proposal for Discovery via DNS Records. We first describe the various record types generally supported by DNS, and then propose our own conventions for some of those records.

As per the requirements indicated above, service records will be created during application deployments and registered in the YARN ZK-backed registry. These records will be parsed to yield records exposed by yDNS.

Applicable Record Types

The following are some of the key record types that will be leveraged by yDNS to provide application and container information (Note that there are zone level DNS information records that a DNS Service will need to manage but are not relevant to this discussion):

- **A and AAAA Records -** Records that map a FQDN (fully-qualified domain-name) to an IPv4 or IPv6 address and are the most often used record type.
- **CNAME Record** - A Canonical Name record (abbreviated as CNAME record) is a type of resource record used to specify that one name is an alias for another name. This record will allow yDNS to associate multiple names to a specific container or application.
- **SRV Record -** A Service record specifies the hostname and port number for services.
- **TXT Record** - A text record is used to provide the ability to associate some arbitrary and unformatted text with a host or other name, such as human readable information about a server, network, data center, and other accounting information.
- **PTR Record** - Pointer records are used to map a network interface (IP) to a name.

Proposal for the DNS records

In the following section, we explore the conventions used to define the DNS names for both application and container related DNS records.

Record name elements

The names will be composed of the following elements (labels). Note that these elements must be compatible with DNS conventions (see "Preferred Name Syntax" in RFC 1035):

- ***domain*** - the name of the cluster DNS domain. This name is provided as a configuration property. In addition, it is this name that is configured at a parent DNS server as the zone name for the defined yDNS zone (the zone for which the parent DNS server will forward requests to yDNS). E.g. ***yarncluster.com***

- ***user-name*** - the name of the application deployer. This name is simple short-name (for e.g. the primary component of the Kerberos principal) associated with the user launching the application. As the user-name is one of the elements of DNS names, it is expected that this also confirms DNS name conventions (RFC 1035 linked above) - we will need to do special translation for names with special characters like hyphens and spaces.

- ***application-name*** - the name of the deployed YARN application. This name is inferred from the YARN registry path to the application node. We chose application-name instead of application-id as a way of making it easy for users to refer to human-readable DNS names. This obviously mandates certain uniqueness properties on application-names. For a further discussion about its uniqueness etc, see the notes section immediately below.

- ***container id*** - the YARN assigned ID to a container (e.g. *container_e3741_1454001598828_01_000004*)

- ***component-name*** - the name assigned to the deployed component (for e.g. a *master* component). A component is a distributed element of an application or service that is launched in a YARN container (e.g. *an HBase master*). One can imagine multiple components within an application. A component-name is not yet a first-class concept in YARN, but is a very useful one that we are introducing here for the sake of yDNS entries. Many frameworks like MapReduce, Slider already have component-names though they are not supported in YARN in a first-class fashion yet.

- ***api*** - the api designation for the exposed endpoint (e.g. 'publisher'. See the service record example in Appendix A for more concrete API examples)

Some notes about the record conventions below:

- In most instances, the records below can be easily distinguished by the number of elements/labels that compose the name. The cluster's domain name is always the last

element. After that element is parsed out, reading from right to left, the first element maps to the application-user and so on. Wherever it is not easily distinguishable, we require special conventions to disambiguate the name - using a prefix such as "container-" or suffix such as "api". For example, an endpoint published as a management endpoint will be referenced with the name ***management-api.griduser.yarncluster.com***.
- Unique application-name (per user) is not currently supported/guaranteed by YARN, but it is supported by frameworks such as Apache Slider. The yDNS service currently leverages the last element of the ZK path entry for the application as an application-name. These **application-names will have to be unique for a given user**. Attempts to create an application with the same name, by the same user, should result in DNS record registration failures, and ideally should be flagged or not permitted by the deploying client (e.g. Slider client). For example, during application creation, Slider currently performs an initial check to ascertain whether a cluster directory by the provided application-name already exists, and, if it does, rejects the request. Moving forward, it may be more appropriate for YARN to enforce application-name uniqueness, at least for service-applications.

Per-Application records

There are two types of per-application records that we propose:

<u>Names that maps to an application</u>

- Format: *.***<application-name>.<user-name>.<domain>***
- Example: Application *zkapp1*, user *griduser*, domain *yarncluster.com* would be referenced by ***zkapp1.griduser.yarncluster.com***
- Record types:
  - Address records (A and AAAA) listing the hosts in the provided API URIs
  - CNAME records mapping the API record name (see above) to the generated address records
  - SRV records for the listed APIs that provide a host and port either through the provided URI or the explicit host/port specification (address type "host/port" in the YARN Registry specification)
  - TXT records relating the API types and associated context paths

Application Record Type	Function
A/AAAA	FQDN to an IPv4 or IPv6 address
SRV	Provide host and port of exposed application endpoints
TXT	Provide available application APIs and their associated context paths (URI)

- Application-level API referencing records
    - API records (CNAME and TXT records referencing available service level APIs/URIs primarily hosted by the AM) are referenced by the API referencing name
    - Format: ***<api>-api.<application-name>.<user-name>.<domain>***
    - E.g. ***publisher-exports-api.zkapp1.griduser.yarncluster.com***
    - The “-api” suffix in the name distinguishes it from container record names (see below)

Application Record Type	Function
A/AAAA	FQDN to an IPv4 or IPv6 address
CNAME	Mapping of API name to address
TXT	Provide API type

Per-Container records

There are two types of per-container records that we propose, referenced by two address record (A, AAAA records) with the following formats:

Names that map to a container

- Name Format: ***<container id>.<domain>***
- E.g. ***container-e3741-1454001598828-0131-01-000004.yarncluster.com***
- Underscores in YARN container records are converted to hyphens to align with DNS requirements (see RFC 1123).
- Record types:
    - Address records (A and AAAA) mapping the container ID to the container IP (if available).
    - A reverse lookup record (PTR) mapping the IP address (if available) to the per-container DNS name.

Container Record Type	Function
A/AAAA	FQDN to an IPv4 or IPv6 address
PTR	Reverse lookup from IP

Names that map to a component

- Format: ***<component>.<application-name>.<user-name>.<domain>***
- E.g. ***zk1.zkapp1.griduser.yarncluster.com***
- Periods are NOT allowed in component-names due to the confusion that would yield with period as the general record-separator.
- Record types:
    - Address records (A and AAAA) mapping the component-name to the container IP (if available).
    - A TXT record named using the component ID based record name that lists the YARN container ID

Container Record Type	Function
A/AAAA	FQDN to an IPv4 or IPv6 address
TXT	Provides YARN assigned container ID

- A typical application may have multiple component instances (e.g. multiple HBase region servers). A deployer has the option of:
    - Deploying multiple containers of the same component type/name. In that instance, yDNS queries will yield multiple response DNS records (e.g. multiple address records providing all IPs associated with deployed container instances)
    - Designating all application components with unique names. This would yields single responses for DNS queries.

Registry Service Record Parsing Guidelines

This section provides some additional details concerning the generation of DNS records from YARN registry application registrations. Most of the following relates to how the DNS records are *implemented* in yDNS and so is independent of our API proposals, but we keep this section closer to our record definitions for the sake of readability.

- The last element of the registration node in service-registry is considered as the ***<application-name>*** in yDNS, e.g. the registry path ***/registry/users/aUser/services/org-apache-yarn/myApplicationName*** would yield an application-name of ***"myApplicationName"***.
- In addition, the <user-name> associated with record names will be retrieved from the application node in the YARN Registry. Leveraging the example path above, the user-name would be ***"aUser"***.

Custom Aliases

Rather than make use of the implicit naming mechanism above, application-deployer may have a desire for an alternate mechanism that allows them to select a specific, potentially shorter, alias to leverage for the DNS registry of a given component.

(NOTE: The following discussion uses Slider application constructs to illustrate the mechanism)

During deployment, the application deployment for a given component can set a property:

```
    "components": {
      "slider-appmaster": {
        "jvm.heapsize": "512M"
      },
      "MY_COMP": {
        "service.record.attribute.name": "mycomponent"
      }
    }
```

This explicitly can be leveraged by slider to augment the Record with a name attribute (see JIRA proposal at https://issues.apache.org/jira/browse/SLIDER-1086):

```
  {
    "type" : "JSONServiceRecord",
    "description" : "MASTER",
    "name" : "mycomponent",
    "external" : [ ],
    "internal" : [ ],
    "yarn:id" : "container_e03_1449766058071_0003_01_000002",
    "yarn:persistence" : "container",
    "yarn:ip" : "172.17.0.20",
    "yarn:hostname" : "myContainer"
  }
```

The yDNS service can pickup the registered record and create a DNS record leveraging the name - ***<alias>.<user-name>.<domain>***, e.g. ***mycomponent.griduser.yarncluster.com.***

Notes:

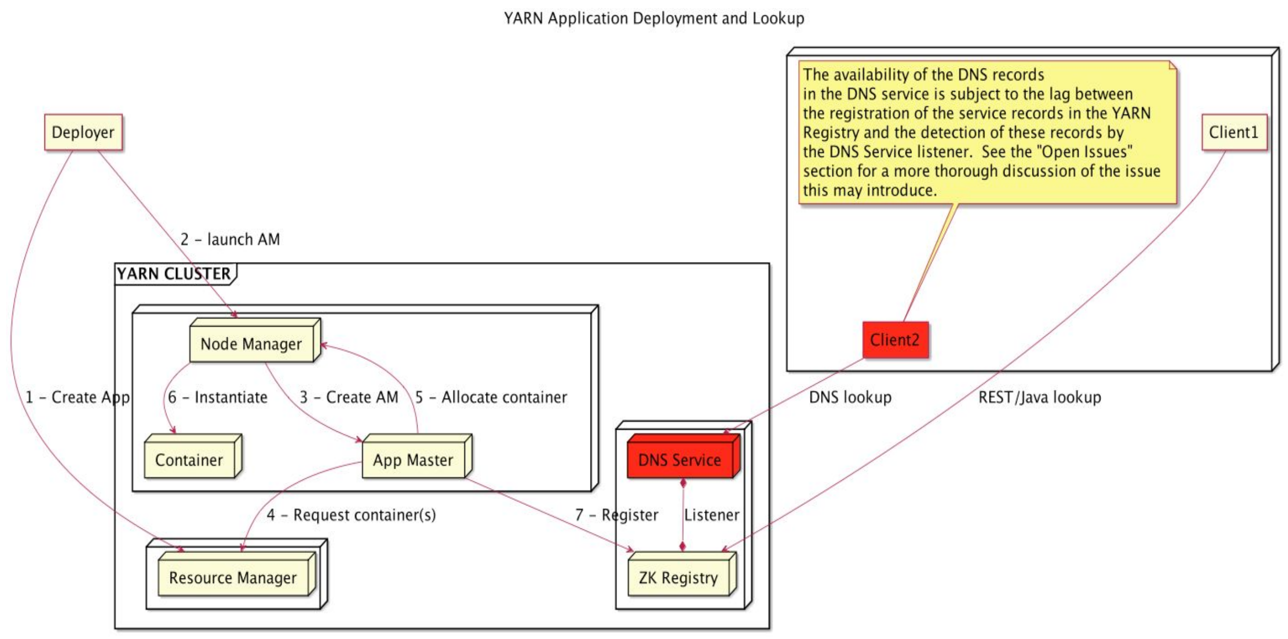
- To ensure there is no ambiguity, the ***"container-"*** prefix is **not** allowed either for aliasing or in an application-name. At attempt to use that prefix will result in a deployment failure.

- Aliased records also still include the user-name to avoid name collisions between apps of different users
- Alias names should perhaps be looked up during application deployment. Name collisions should be treated as a trigger for deployment failure (unless an overwrite flag is specified, likely).
- Given that custom aliases and app-name share the same element position, it is likely that users may not easily disambiguate between these two type of records. However, this is restricted to a user and we leave it up to the users completely.

(5) Architecture

Overview

As depicted in the figure below, a YARN application is deployed and registered as follows:


YARN Application Deployment and Lookup
Deployer
2 – launch AM
1 – Create App
YARN CLUSTER
Node Manager
6 – Instantiate
3 – Create AM
5 – Allocate container
Container
App Master
4 – Request container(s)
7 – Register
Listener
Resource Manager
DNS Service
ZK Registry
The availability of the DNS records in the DNS service is subject to the lag between the registration of the service records in the YARN Registry and the detection of these records by the DNS Service listener. See the "Open Issues" section for a more thorough discussion of the issue this may introduce.
Client1
Client2
DNS lookup
REST/Java lookup

1. A deployer contacts the RM and initiates the application creation. He receives a container allocation for the AM as a response.
2. The deployer interacts with the Node Manager on the designated host to instantiate the AM container.
3. The Node Manager launches the AM container.
4. The App Master initializes and requests the necessary container allocations from the Resource Manager.

5. The App Master interacts with the Node Manager(s) associated with the returned container allocation(s) to instantiate the necessary containers.
6. The Node Manager instantiates the containers leveraging the container launch context provided by the App Master.
7. At this point the App Master interacts with the ZK-backed YARN Registry to register the application-level and container-level service records. Note that there is a need for a management element (e.g. RM) to manage the permissions for the ZK nodes (For example, currently the RM is responsible for creating the root nodes for a given user and setting the write permissions to allow that user to write/update/delete sub-nodes).

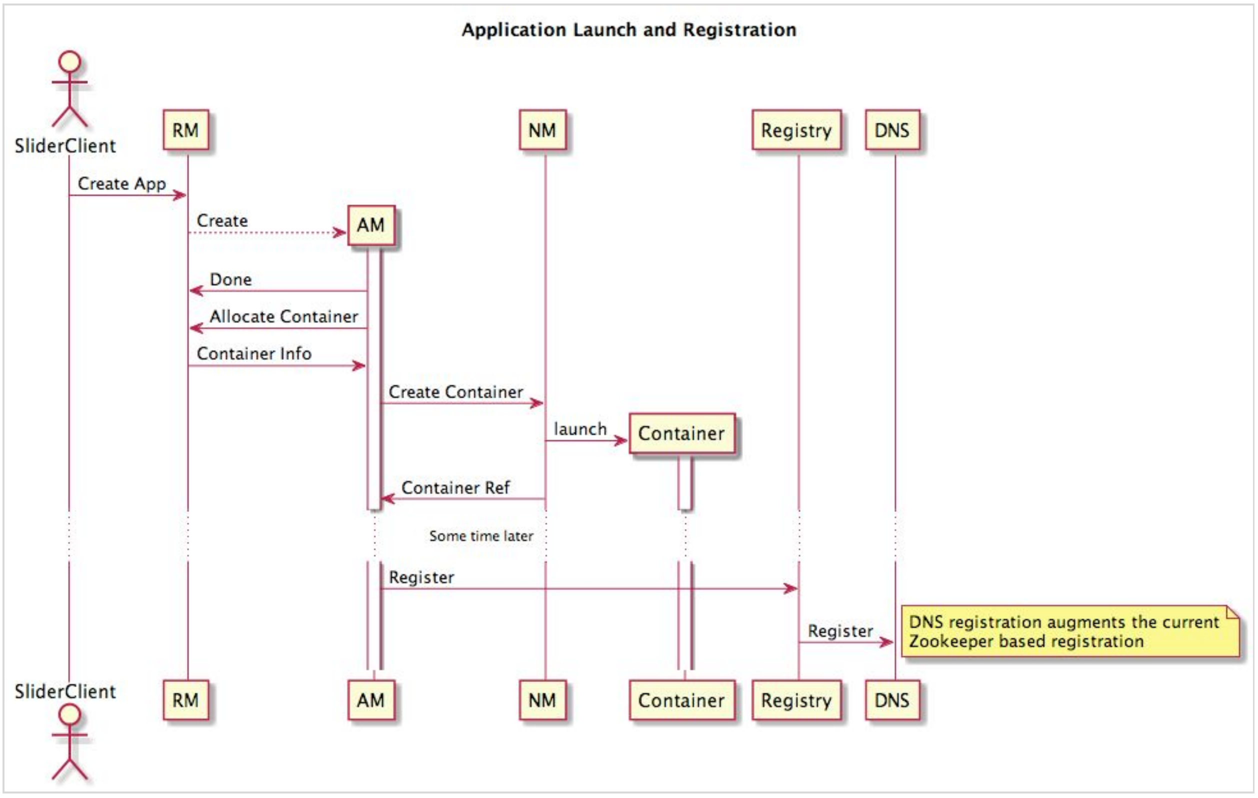
This specification details a DNS-based service that will augment this process by providing a DNS Service that will allow clients to retrieve application and container information via standard DNS mechanisms.

Architecture Details

There are a number of key functions to be performed by the DNS service

Application launch and registration

- Current application launch and registration sequences will be amended with the addition of the registration of DNS information concerning an application and its containers:

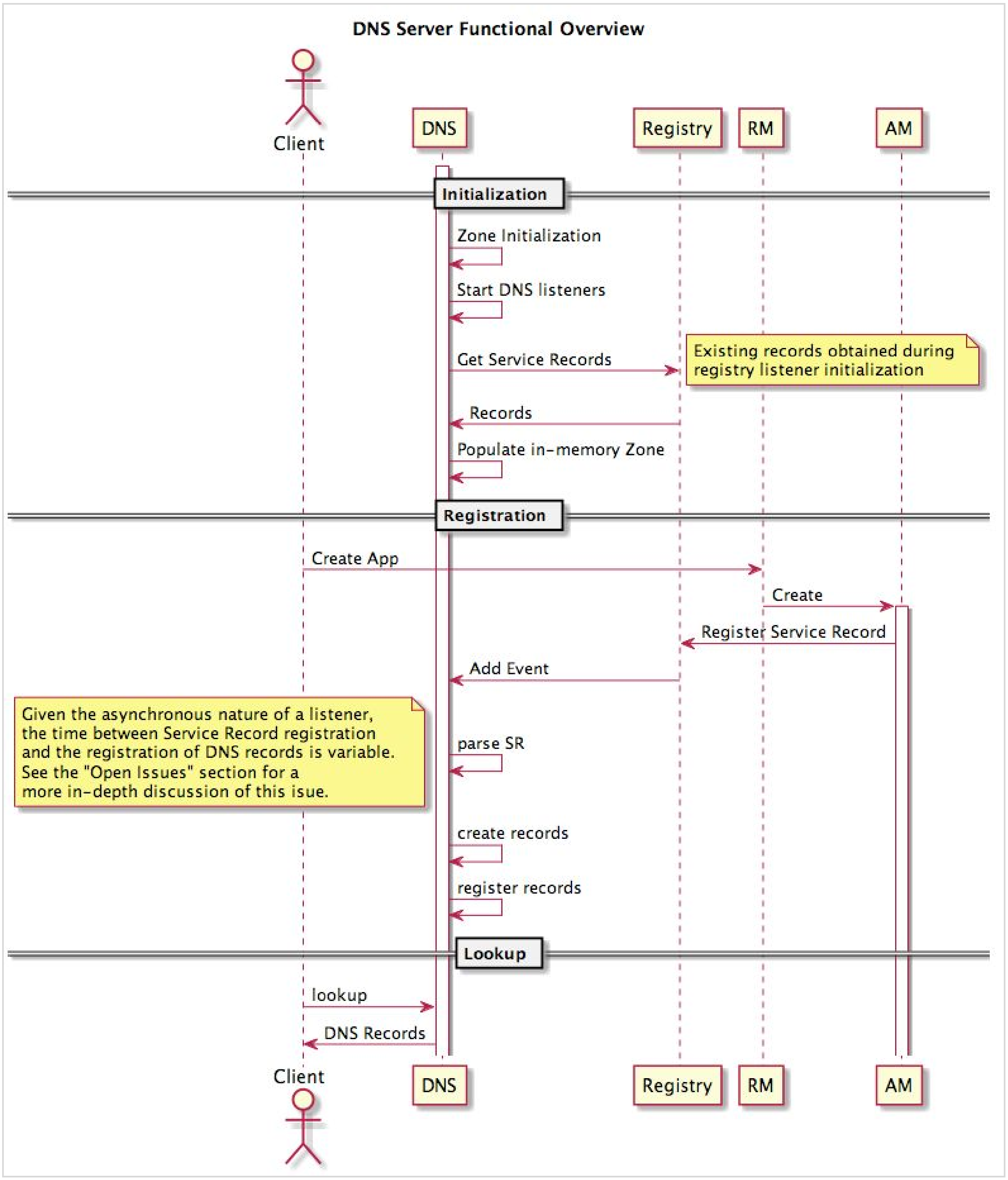


- The DNS service will perform the CRUD operations by monitoring the Zookeeper-based YARN registry. A listener, such as the Curator Tree Cache Listener, will monitor the root node of the YARN registry and will process events that indicate the addition of a node, the removal of a node, or the update of a node. The associated service records will be processed and converted into DNS records.

DNS Service internals

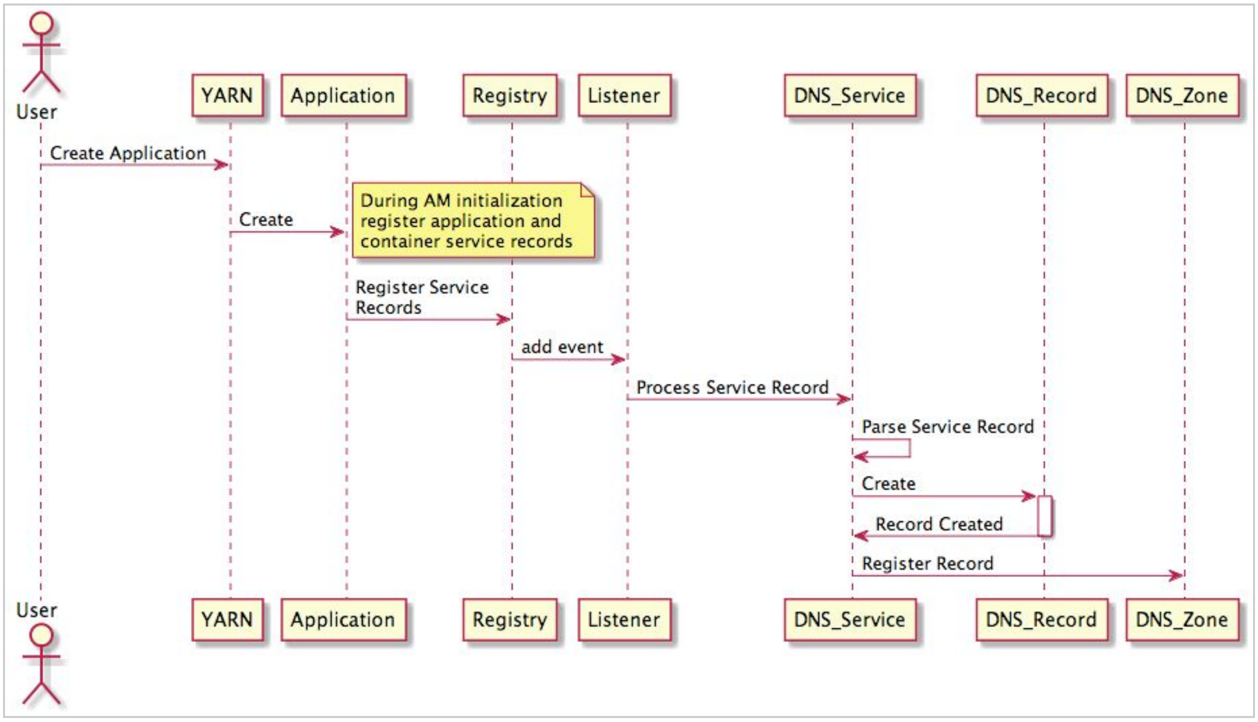
Functional Overview

The primary functions of the DNS service are illustrated in the following diagram


DNS Server Functional Overview
Client
DNS
Registry
RM
AM
Initialization
Zone Initialization
Start DNS listeners
Get Service Records
Existing records obtained during registry listener initialization
Records
Populate in-memory Zone
Registration
Create App
Create
Register Service Record
Add Event
Given the asynchronous nature of a listener, the time between Service Record registration and the registration of DNS records is variable. See the "Open Issues" section for a more in-depth discussion of this isue.
parse SR
create records
register records
Lookup
lookup
DNS Records

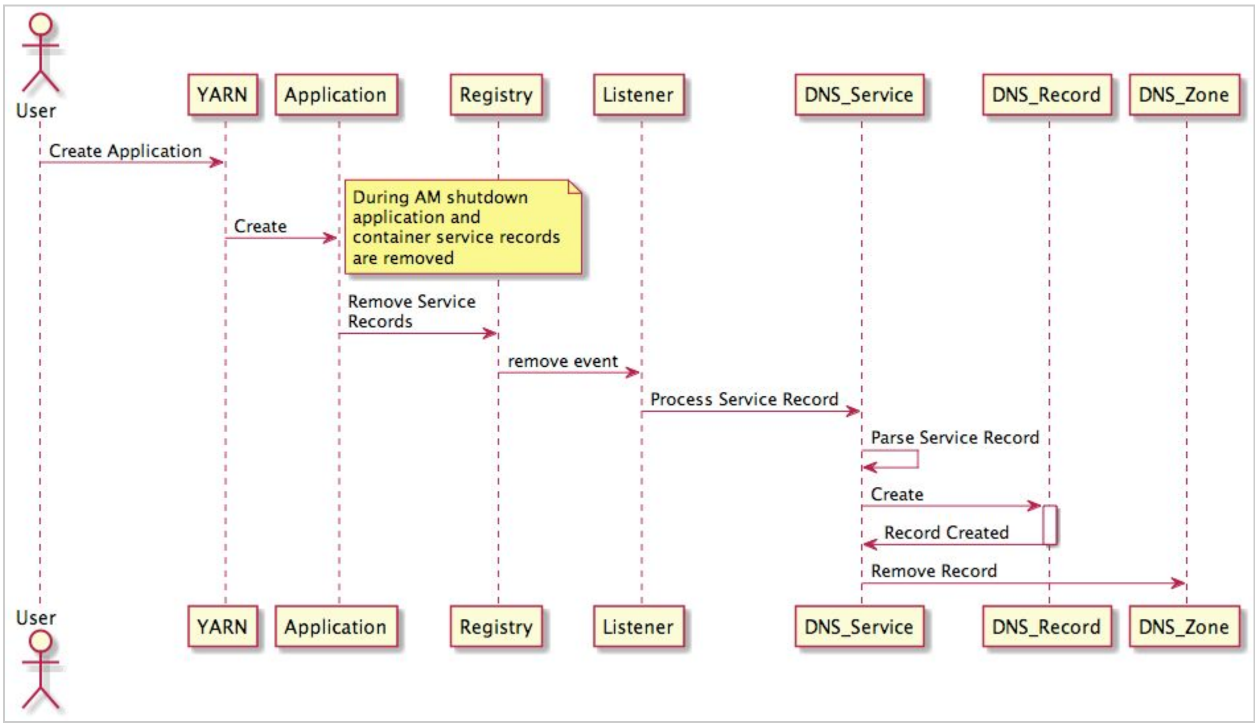
DNS record creation

The following figure illustrates at slightly greater detail the DNS record creation and registration sequence (NOTE: service record updates would follow a similar sequence of steps, distinguished only by the different event type):



DNS record removal

Similarly, record removal follows a similar sequence



(NOTE: The DNS Zone requires a record as an argument for the deletion method, thus requiring similar parsing logic to identify the specific records that should be removed).

DNS Service deployment

- The DNS service will initialize both UDP and TCP listeners on a configured port. As noted above, the default port of 53 is in a restricted range that is only accessible to an account with administrative privileges.

- Subsequently, the DNS service will listen for inbound DNS requests. Those requests will be standard DNS requests from users or other DNS servers (DNS servers that have the YARN DNS service configured as a forwarder).

- There are a number of alternatives for the execution environment of the DNS process: hosted in the RM, hosted in ZK, or operating as a standalone process. It would appear that the latter choice of running as a standalone process is advantageous in the near term since:
    - Advantages
        - It will allow for analysis of the memory, CPU and threading performance of yDNS.
        - A proper design will not preclude the embedding of yDNS in either the RM or ZK should the analysis above reveal that those approaches are viable.
        - The ability to deploy multiple instances aligns well with current DNS HA approaches (see more below).
    - Note that there are some disadvantages as well:
        - A separate process will require management and administrative features
        - As noted above, the asynchronous nature of the interaction between the DNS service and the YARN Registry introduces a lag between the registration of an application and the availability of its associated DNS records.

High Availability

High availability of yDNS can be supported via standard DNS mechanisms. Given that each yDNS instance simply reflects the current state of the ZK-backed YARN Registry (and is in itself fully stateless), there is no need to augment the mechanisms that currently exist for DNS HA.

Specifically, resolution of names is generally supported by iterating through a list of name-servers. In order to perform a lookup, the client iterates through the listed name-servers until it can connect and receive an authoritative response to the name query (note that an NXDOMAIN response - no such name - from a server does not indicate the need to search further, but is rather viewed as an authoritative negative response). Some examples:

- resolv.conf - the resolution logic on a host will iterate through the server listed in the file and attempt to resolve the name by finding a server that accept the connection.

- zone forwarders - a zone defined in a parent server can specify a list of forwarders to which queries for the given domain will be forwarded. Again, the servers listed will be tried one by one until a connection is accepted and an authoritative response is received.

Given the nature of DNS name server configuration, as well as the fact that the actual source of truth for registered applications is the Zookeeper-backed YARN registry, HA support can be achieved by simply starting multiple DNS Service instances and configuring the DNS system accordingly. Each DNS server instance would register as a YARN registry listener, receive the same set of registration events, and create the same set of DNS records. A parent DNS server, configured with multiple instances of the the yDNS Service as forwarders, would invoke the forwarders in sequence. The only reason for a parent server to attempt to query the next forwarder listed is if there is no response from the current forwarder. For example, a forwarded zone in a parent server can have the following configuration:

```
zone "hadoop.site" IN {
        type forward;
        forward first;
        forwarders { host1 port 9900; host2 port 9001; };
};
```

With this configuration, the parent DNS server would attempt to forward “hadoop.site” related requests to the first server listed. If no connection can be established, the second server listed would be queried.

On-premise deployment

Zone Server vs Primary Server

The YARN DNS Service will be designed to serve as a zone that is authoritative solely for records associated with deployed YARN applications. In other words, it will not be appropriate as a primary server that can return a broad set of requested records (e.g. “nslookup google.com”), and thus should not typically be listed in a resolv.conf file. There are a number of reasons for this decision:

1. A fully functional DNS server has to deal with security features such as authentication and denial of service (DOS) attacks, caching facilities to enhance responsiveness etc. Rather than attempt to duplicate those features, it is more appropriate to design a server

that can simply work in concert with existing DNS servers rather than usurp their capabilities.

2. A typical DNS infrastructure is generally comprised of a parent server that can delegate to other servers responsible for specific DNS domains/zones. This configuration is typical of many corporate DNS deployments. It seems sensible to provide a server that fits in with a typical deployment rather that put forward an unproven alternative.

Supporting reverse-lookups

In addition to the ability to act as a zone server for forward lookups, reverse lookups (mapping of IP to name) should be supported to facilitate the multitude of DNS client usage scenarios. To that end, yDNS will:

- Support the creation of a reverse lookup zone
- Application deployments and undeployments will trigger the registration and removal of PTR records (Pointer records are used to map a network interface to a host name).
- DNSSEC support for the reverse zone will mirror the forward zone support (i.e. the same public and private keys will be utilized)

Optional Features

- Primary resolver forwarding - If there is a need to have the yDNS Service serve as an authoritative server in certain deployments, provide a mechanism that allows the forwarding of name lookups to a primary name server for names for which the server has no records (e.g. the ability to lookup names that do not exist in the defined Hadoop cluster zone).
- Static cluster configuration - Support the reading of a zone file, containing static records the server can serve to clients. For example, the zone can be pre-loaded with the address and/or pointer records associated with the cluster hosts.

Security/DNSSEC Support

The Domain Name System Security Extensions (DNSSEC) provide origin authentication and integrity assurance services for DNS data, including mechanisms for authenticated denial of existence of DNS data.

The YARN DNS Service will not provide support for all DNSSEC related interactions. Rather, it will support the interactions required to support positive query responses (negative responses trigger a series of record requests associated with establishing denial of existence). Specifically, if DNSSEC is enabled via configuration, yDNS will:

- Read a configured DNSSEC public key
- Register a DNSKEY public key record
- Read a standard DNS private key file in order to instantiate a private key used for record signing. This file will have appropriate permissions and reside in a properly secured directory to ensure access is limited to the DNS service. In addition, yDNS will be responsible for ensuring that the above security measures (permission sets) are properly set up.
- Sign the required zone records (SOA, NS)
- Create record signatures for query responses
- Return the appropriate negative responses per section 5 of RFC 2065.

Other security mechanisms leveraged by Hadoop components, such as authentication, are not relevant to the operation of yDNS:

- The service will provide no direct interactions for users. Rather, the state of the service is solely dependent on the state of the YARN Registry.

- The service interacts with the YARN Registry on a read only basis. The information is thus accessible without establishing a client identity.

In addition, wire encryption is not appropriate since there are no accommodations for such security in DNS.

(6) Implementation details

Implementation Choices

There are a number of possible implementation approaches that can potentially be leveraged to address the requirements above.

BIND Server

BIND is open source software that implements the DNS protocols for the Internet. It is a reference implementation of those protocols, but it is also production-grade software, suitable for use in high-volume and high-reliability applications. It is probably the most frequently leveraged DNS server.

The BIND server could potentially be used to implement a YARN associated DNS server by leveraging the dynamic update protocol of DNS. This approach would require code to be written that would act as a YARN Registry listener, parse the service records registered for deployed apps, and send the resulting DNS records via the update protocol messages.

Pros

- DNS compatibility
- Full DNSSEC support
- Relatively low coding effort

Cons

- Complex configuration and maintenance - the installation, configuration, and management of a BIND server is generally an IT Admin task requiring significant effort.
- Dynamic zone management - zones that dynamically changed (records are registered and removed) have more cumbersome security requirements.
- Stateful - the BIND server is responsible for maintaining state by persisting records to the zone file.
- In order to support HA, multiple instances would have to be started, configured, and updated.

Other Service Discovery Solutions

There are various solutions on the market that support the registration of services. Examples include service discovery solutions such as Consul and SkyDNS. Some of these solutions also provide an interface that allows for the lookup of these services via DNS protocols.

Pros

- Existing technology
- Proven performance - some solutions provide HA and other enterprise features
- Simplicity - the API interfaces are generally Java and/or REST based

Cons

- Management complexity - these solutions would require additional management effort to address the deployment and configuration requirements.
- Lack of DNSSEC support - most of the solutions on the market do not appear to support DNSSEC. Therefore, there usability in most corporate deployments would be rather limited.
- Proprietary interfaces - the APIs exposed are tailored to the specific product, limiting the ability to “plug in” new approaches.

Tailored DNS Service

This approach requires coding a server that supports the DNS features required to support the proposed architecture. A DNS Server can be written by leveraging existing Java-based protocol implementations such as dnsjava.

Pros

- Tailored support for necessary DNS features - the server can be coded explicitly to support the necessary DNS protocols as well as work in conjunction with corporate servers.
- Manageability - coded as a first-class YARN component, the server can be managed and configured via existing Hadoop-based mechanisms. In addition, there would be no operational dependencies on elements external to the Hadoop cluster.
- Deployment Flexibility - if coded properly, the decision as to whether the DNS service runs in the same process space as an existing component or as an independent process can be deferred until the load and performance requirements are more clearly understood.
- Existing knowledge - other than gaining an understanding of DNS and its protocols, the coding effort for creating this service will be well understood.
- Record modification - record additions, updates, and deletions will not require an implementation of the relatively complex DNS Dynamic Update protocol.

Cons

- More involved coding effort

Given the pros and cons of each of these approaches, the initial inclination is to proceed with the coding of a tailored DNS Service using the dnsjava framework.

Configuration

The YARN DNS server will read its configuration properties from the yarn-site.xml file using the standard Hadoop Configuration object. Some properties associated with the server:

Property	Description
hadoop.registry.dns.enabled	The DNS functionality is enabled for the cluster. Default is false.
hadoop.registry.dns.domain-name	The domain name for Hadoop cluster associated records.
hadoop.registry.dns.bind-address	Address associated with the network interface to which the DNS listener should bind.

hadoop.registry.dns.bind-port	The port number for the DNS listener. The default port is 53. However, since that port falls in a administrator-only range, typical deployments may need to specify an alternate port.
hadoop.registry.dns.dnssec.enabled	Indicates whether the DNSSEC support is enabled. Default is false.
hadoop.registry.dns.public-key	The base64 representation of the server's public key. Leveraged for creating the DNSKEY Record provided for DNSSEC client requests.
hadoop.registry.dns.private-key-file	The path to the standard DNSSEC private key file. Must only be readable by the DNS launching identity. See dnssec-keygen documentation.
hadoop.registry.dns-ttl	The default TTL value to associate with DNS records. The default value is set to 1 (a value of 0 has undefined behavior). A typical value should be approximate to the time it takes YARN to restart a failed container.
hadoop.registry.dns.zone-subnet	An indicator of the subnet associated with the cluster containers. The setting is utilized for the generation of the reverse zone name. See reverse lookup discussion above.
hadoop.registry.dns.zones-dir	A directory containing zone configuration files to read during zone initialization. See optional features discussion above.

(7) Open Issues

Application RUNNING state vs DNS record availability

While providing a number of advantages as noted above, an implementation based on a ZK listener does introduce a timing issue: The server state will transition to RUNNING prior to the application associated records getting registered in yDNS.

Assume that a network is configured with a primary DNS server, and the YARN DNS server is configured as a zone server - a server to which the primary DNS server forwards requests that are associated with a specific zone (in this case, the zone associated with the YARN cluster). If a client performs a lookup of an application record through the primary DNS server (e.g. an ‘nslookup’), a negative response will be returned from the yDNS Service to the parent server, and the parent server will initiate its negative response procedure. That procedure requires the parent server to cache the negative response for a configured period (negative TTL), which for most servers defaults to 3 hours. In other words, if the record is looked up too quickly the parent server will not return a positive response for 3 hours, even if the zone server has the record available (This policy is more than likely in place to prevent negative lookup DOS attacks). There are indications that the TTL can be modified based on the response from a zone’s authoritative server (minimum field of returned SOA record), but initial tests do not seem to bear this out (see RFC 2308).

One possible solution is to incorporate a lookup against the yDNS Service that would gate the transition to the RUNNING state (given a check for whether DNS is enabled). The location of the DNS Server is known to YARN components (it’s configured in the yarn-site configuration file), so a direct query could reveal whether the expected records are available without triggering the negative response issue.

IPv6 Reverse Lookup Support

IPv6 reverse lookup support requires a separate, dedicated reverse lookup zone. Is such support as an implementation requirement, or should such support be optional?

Full DNSSEC support

Full DNSSEC support would require support for NSEC/NSEC3 record support. These records can be used by resolvers to verify the non-existence of a record name and type as part of DNSSEC validation. These support is more advanced and would require hashing/signing the current contents of the zone.
NSEC/NSEC3 support may actually not be required. RFC 2065, Section 5 does prescribe a response for non-existent names and types. It may be sufficient to provide the support outlined in this section.

HA Support

As indicated in this document, DNS assumes a set of active DNS servers (fowarders) to which a parent server can forward queries for a specific zone. Therefore, there is an implicit requirement to have at least one backup server, beyond the first listed server, available to handle any failover scenarios.
If the standalone process option specified in the document is selected for the yDNS Service, the HA support would simply require starting up multiple instances of yDNS and specifying the

host and port of each in the parent DNS server's zone configuration. An administrator could, for example, elect to start up a yDNS instance along-side the active and passive RM instances. However, if another deployment option is selected (e.g. embedding yDNS in the RM), there would be a need to think carefully about whether the parent DNS server can query any of the fielded yDNS instances for cluster-associated records (e.g. an active-passive scheme may not align with that requirement).

Signature configuration

Given the nature of DNSSEC signatures and the need to specify an expiration for the provided signature there is probably a need to add configuration properties for defining the period of the signature (with a reasonable default - 1 year?)

Zookeeper Dependence

The current architectural approach has a direct dependency on ZK. Issues with ZK connectivity could impede the availability of cluster associated DNS records as well as the responsiveness of the server to incoming queries.

This issue is somewhat mitigated by DNS server caching. Since yDNS acts as a zone server for the cluster, the parent DNS server will cache query results and respond to client queries (cache hit that are still valid are leveraged for responses rather than querying the zone server). However, given the possibility of connectivity issues and the related problems introduced (no records, negative response caching leading to delays, etc), an alternative approach may be warranted (e.g. a form of direct communication between the AM or RM and the DNS registry).

Other issues that may arise due to this dependency that would require solutions:

- If a network partition isolates the DNS server from part or all of the ZK quorum, it will cause all non-cached discovery lookups to report no mapping in cases where there is in fact a mapping
- Frequent changes in a YARN registry data may lead to load issues given the high volume of rate of change and event generation.

(8) Appendix A: Existing Service Registry Functions

Registration of information

In this section, we describe how application-level and container-level information gets registered with the Service Registry.

Application-level Service Records

During deployment, an application service record is registered that provides information concerning the endpoints and APIs provided by the application (these may be endpoints exposed by the AM). For example, the following is a representation of an application service record registered during the creation of a Slider application:

```
ServiceRecord{description='Application Master'; external endpoints: {{
  "api" : "classpath:org.apache.slider.appmaster.ipc",
  "addressType" : "host/port",
  "protocolType" : "hadoop/IPC",
  "addresses" : [ {
    "host" : "host.example.com",
    "port" : "1024"
  } ]
}; {
  "api" : "http://",
  "addressType" : "uri",
  "protocolType" : "webui",
  "addresses" : [ {
    "uri" : "http://host.example.com:1025"
  } ]
}; {
  "api" : "classpath:org.apache.slider.client.rest",
  "addressType" : "uri",
  "protocolType" : "webui",
  "addresses" : [ {
    "uri" : "http://host.example.com:1025"
  } ]
}; {
  "api" : "classpath:org.apache.slider.management",
  "addressType" : "uri",
  "protocolType" : "REST",
  "addresses" : [ {
    "uri" : "http://host.example.com:1025/ws/v1/slider/mgmt"
  } ]
}; {
```

```
          "api" : "classpath:org.apache.slider.publisher",
          "addressType" : "uri",
          "protocolType" : "REST",
          "addresses" : [ {
            "uri" : "http://host.example.com:1025/ws/v1/slider/publisher"
          } ]
        }; {
          "api" : "classpath:org.apache.slider.registry",
          "addressType" : "uri",
          "protocolType" : "REST",
          "addresses" : [ {
            "uri" : "http://host.example.com:1025/ws/v1/slider/registry"
          } ]
        }; {
          "api" : "classpath:org.apache.slider.publisher.configurations",
          "addressType" : "uri",
          "protocolType" : "REST",
          "addresses" : [ {
            "uri" : "http://host.example.com:1025/ws/v1/slider/publisher/slider"
          } ]
        }; {
          "api" : "classpath:org.apache.slider.publisher.exports",
          "addressType" : "uri",
          "protocolType" : "REST",
          "addresses" : [ {
            "uri" : "http://host.example.com:1025/ws/v1/slider/publisher/exports"
          } ]
        }; }; internal endpoints: {{
          "api" : "classpath:org.apache.slider.agents.secure",
          "addressType" : "uri",
          "protocolType" : "REST",
          "addresses" : [ {
            "uri" : "https://host.example.com:55296/ws/v1/slider/agents"
          } ]
        }; {
          "api" : "classpath:org.apache.slider.agents.oneway",
          "addressType" : "uri",
          "protocolType" : "REST",
          "addresses" : [ {
            "uri" : "https://host.example.com:58047/ws/v1/slider/agents"
          } ]
        }; }, attributes: {"yarn:id"="application_1449759131890_0007"
        "yarn:persistence"="application" }}
```

Container Service Records

The container record is somewhat simpler, relating only some key attributes that are relevant to the given container:

```
{
  "type" : "JSONServiceRecord",
  "description" : "CONTAINER",
  "external" : [ ],
  "internal" : [ ],
  "yarn:id" : "container_e03_1449766058071_0003_01_000002",
  "yarn:persistence" : "container",
  "yarn:ip" : "172.17.0.20",
  "yarn:hostname" : "cfba8909a072"
}
```

Note that a container instance has a one-to-one mapping to a component instance.

Lookup

The service records associated with an application can be retrieved from the YARN registry by leveraging client tooling (that depends on Service Registry’s java APIs) or via the available REST APIs.